2006-07-06

SEU-Induced Persistent Error Propagation in FPGAs

Keith S. Morgan
Brigham Young University - Provo

SEU-INDUCED PERSISTENT ERROR PROPAGATION IN FPGAS

by

Keith Shearl Morgan

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

Brigham Young University

August 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Keith Shearl Morgan

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Michael J. Wirthlin, Chair
Date	Brent E. Nelson
Date	Clark N. Taylor

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Keith Shearl Morgan in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Michael J. Wirthlin
Chair, Graduate Committee

Accepted for the Department

Michael A. Jensen
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering and Technology

ABSTRACT

SEU-INDUCED PERSISTENT ERROR PROPAGATION IN FPGAS

Keith Shearl Morgan

Department of Electrical and Computer Engineering

Master of Science

This thesis introduces a new way to characterize the dynamic SEU cross section of an FPGA design in terms of its persistent and non-persistent components. An SEU in the persistent cross section results in a permanent interruption of service until reset. An SEU in the non-persistent cross section causes a temporary interruption of service, but in some cases this interruption may be tolerated. Techniques for measuring these cross sections are introduced. These cross sections can be measured and characterized for an arbitrary FPGA design. Furthermore, circuit components in the non-persistent and persistent cross section can statically be determined. Functional error mitigation techniques can leverage this identification to improve the reliability of some applications at lower costs by focusing mitigation on just the persistent cross section. The reliability of a practical signal processing application in use at Los Alamos National Laboratory was improved by nearly two orders of magnitude at a theoretical savings of over 53% over traditional comprehensive mitigation techniques such as full TMR.

ACKNOWLEDGMENTS

I would like to first thank my wife Natalie for her many acts of selflessness. She has stuck with me through many late nights in the lab. She has also been my greatest support.

I would also like to thank my family, particularly my parents, for their their encouragement and financial support.

I would also like to thank my advisor, Dr. Michael Wirthlin. He has spent countless hours on my behalf. I particularly want to thank him for challenging me technically and pushing me to become a better writer.

I also want to thank my professors and fellow students at BYU. They have all helped mold me in a positive way.

Finally I want to thank the folks at Los Alamos National Laboratory. They are not only great colleagues but friends as well. They supported this work through the "Improving the Reliability of FPGA Designs through Automated Design Hardening" program under contract #95952-001-04 3C.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Field Programmable Gate Arrays (FPGA) have become an attractive computing solution for space-destined systems. Recent government and military space-based systems have exploited FPGAs. Writer Kevin Morris reported that 76 Actel FPGAs were either on or orbiting mars in 2004. He also reported that Xilinx FPGAs are used in some of the control systems for the high-profile mars rover missions [6].

The increased usage of FPGAs in space can be attributed to the many benefits they provide. FPGAs have high-performance capabilities. The programmable logic and on-chip memories are well-suited to complex high-throughput applications, particularly those used in signal processing. FPGAs' are also very flexible. Their on-demand reconfiguration capability supports the use of multiple applications on the same chip through time multiplexing. In addition, new applications can be 'uploaded' on-mission and existing applications can be 'fixed' if bugs are found or modifications are desired.

Although FPGAs benefit a space system in many ways, the outer space environment poses difficult reliable operation challenges, particularly for FPGAs. The radiation prevalent in space can upset values in the on-chip memories or flip-flops (used by an application to store dynamic data). These incorrect values can manifest as functional errors if they propagate to the chip's output pins. The space radiation can also upset the configuration memory (used to statically store an application's configuration), which can actually modify the configured application. In this case, the corrupted circuit can generate invalid values which will be latched in the on-chip memories or flip-flops. Again, these incorrect values can manifest as functional errors

if they propagate to the chip's output pins. Unfixed, the configuration memory can indefinitely induce functional errors and could permanently interrupt an application's proper operation without external intervention.

As a result of the reliable operation challenges, reliability-constrained FPGA applications must mitigate some measure of functional errors in order to qualify for space. Typical strategies immunize an FPGA application against radiation-induced functional errors by adding redundant copies of all circuit resources and continuously monitoring for and correcting upsets in the configuration memory. The redundancy ensures that no single upset will manifest as a functional error at the chip's output pins. Monitoring and correcting the configuration memory ensures that, for the most part, no more than a single upset exists at any given time.

Unfortunately space-destined systems also often have other design constraints which make complete functional error mitigation infeasible. A system's set of design constraints typically include reliability, availability and cost (area, resources, power etc.). In some cases a system is limited by just one of these three constraints. In other cases a system is limited by a combination of these three constraints or even others. An important combination of constraints for space-destined FPGA applications is reliability and resource count. A completely mitigated implementation of an application may meet reliability constraints, but often exceeds the number of available resources in the FPGA. For these scenarios other solutions are necessary that use fewer resources, but still provide acceptable reliability.

This thesis will introduce a novel mitigation technique that provides acceptable reliability at lower resource costs by giving up availability. To support this technique a system must be able to tolerate temporary interruptions of service. In an FPGA, dynamic functional errors are the interruptions of service. This thesis will show that dynamic functional errors can be divided into two categories based on duration of service interruption. Non-persistent functional errors temporarily interrupt application service. Persistent functional errors permanently interrupt application service until reset. A system which can tolerate non-persistent functional errors only needs to mitigate persistent functional errors. Since persistent functional errors represent only

a subset of all functional errors mitigating them requires fewer redundant resources. Furthermore, mitigating only persistent functional errors reserves valuable redundant resources for mitigating the cause of real application failures. As such, reliability can be improved with far fewer resources.

To quantify the relative reliability improvements presented in this work, on-orbit application failure rates will be presented throughout this thesis for four benchmark applications. Rates will be computed for each of the four applications in three distinct orbits.

1.1 Benchmark Designs

The first benchmark application consists of an array of feed-forward multipliers with no internal state. This circuit represents a typical data-flow application. The second is a large array of 400 8-bit counters (each containing count state). This circuit represents an application with significant amounts of internal state. The third is a synthetic application consisting of LFSRs and a multiplier-adder tree. This circuit represents the typical mixture of data-flow and internal state in an application. The fourth is a Digital Signal Processing (DSP) kernel developed at Los Alamos National Laboratory. This circuit represents a real application. More information about each application, including size and functionality, can be found in Appendix A.

1.2 Orbits

The first orbit is a typical Low-Earth Orbit (LEO) at 560 km altitude and 35.0^o inclination. The second is a Polar orbit at 833 km altitude and 98.7^o inclination. This orbit helps demonstrate the increased failure rates seen at the earth's magnetic poles. The final is the Global Positioning System (GPS) orbit at $22,200$ km and 55.0^o inclination. This orbit is outside the natural shielding of the earth's magnetosphere. Such an orbit also leads to increased failure rates.

1.3 Organization

The organization of this thesis is as follows: Chapter 2 introduces the space radiation environment and FPGA radiation effects. Chapter 3 reviews operational

(dynamic) functional errors in an FPGA. Chapter 4 introduces non-persistent and persistent functional errors. Chapter 5 presents the non-persistent and persistent cross sections, in other words, a mapping from both non-persistent and persistent functional errors to circuit elements. Chapter 6 shows how these mappings can be exploited to mitigate just persistent functional errors. This method can deliver acceptable levels of reliability within other design constraints, often at much lower costs than full functional error mitigation.

As FPGAs are used more and more in space-based systems for their performance and flexibility, FPGA reliability improvement techniques will become more and more essential. The technique presented in this thesis is just one of the many possible methods. It offers one more option or design point in the space of possible reliability enhancement solutions. Systems which can afford to give up some availability can use this technique to improve reliability at lower resource costs.

Chapter 2

Radiation Effects and FPGAs

Developing a method for cost-effective FPGA functional error mitigation requires estimating failure rates for a specific FPGA application. The first step to determine an FPGA application's failure rate is to quantitatively determine the FPGA's sensitivity to radiation. The next step is to determine the radiation flux specific to the FPGA's destined orbit. Based on these two values, static upset rates can be calculated. In later chapters, these rates will be used to predict upset rates for applications before and after functional error mitigation.

This chapter reviews the space radiation environment and generic and FPGA-specific radiation effects. The chapter concludes with a quantitative analysis of orbit-specific upset rates for a specific FPGA in the three orbits described in Section 1.2. Although the results presented are not direct estimates of failure rate for a specific application, they are necessary to compute an application's failure rate. Each of the subsequent chapters (excluding Chapters 4 and 7) will present failure rates based on the values presented in this chapter.

2.1 Space Radiation Environment

Several particle sources contribute to the composition of the space radiation environment. One source, a region known as the Van Allen belts, extends from 800 kilometers above the earth out to 6 earth radii and beyond. This region, depicted in Figure 2.1, mostly consists of trapped electrons and protons. The earth's magnetic field suspends the particles in orbit at a relatively fixed distance. Protons and elec-

trons with energies up to 10 $MeV(millionelectronvolts)$ and several hundred MeV respectively reside in the belts.

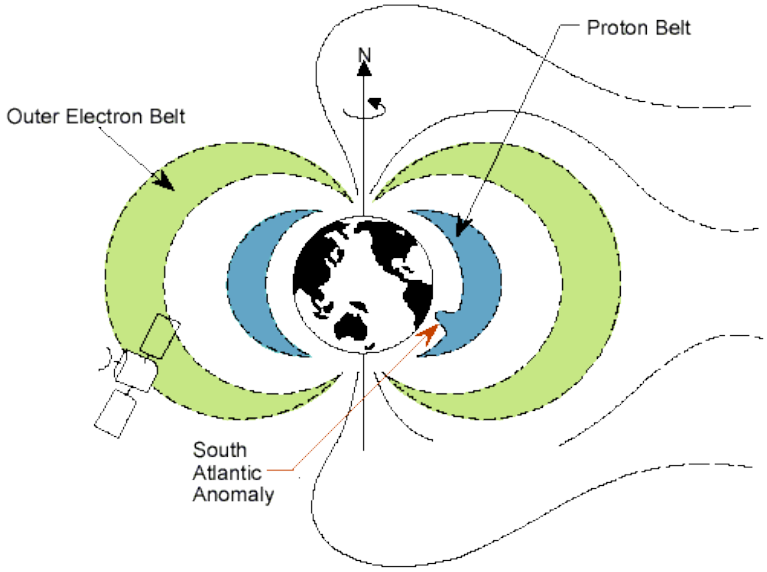


Figure 2.1: Van Allen trapped radiation belts around the earth [2].

The sun also contributes to the radiation environment in space. It periodically ejects intense quantities of energetic particles into space via cosmic rays. Particles in cosmic rays have energies up to several thousand MeV. The earth's magnetic field deflects a majority of these particles. As a result, the particle flux at a given point in space is a function of the earth's magnetic field strength at that location. Peak particle fluxes typically exist at higher altitudes and near the earth's magnetic poles.

In addition to particles from the sun, cosmic rays of unknown origin contribute to particle flux, particularly at high altitudes and near the earth's magnetic poles. Galactic Cosmic Rays (GCR), as they are called, typically consist of about 85 percent protons, about 14 percent alpha particles and about 1 percent heavier nuclei [5]. Particles with energy up to the GeV range exist in GCRs. More information about

the temporal and spatial composition of the space radiation environment can be found in Appendix B.

2.2 Single Event Effects

Energetic particles in space, trapped in the Van Allen belts or transmitted by cosmic rays, can deposit unwanted charge in a microelectronic device. Excess charge can cause transient faults or even permanent damage. Figure 2.2 lists the different types of transient and permanent faults, commonly called soft and hard errors respectively. The set of all soft and hard errors are known as Single Event Effects (SEE). Note that a single particle causes each of the different SEE fault types. Other time-integrated effects also occur as the result of total radiation dose.

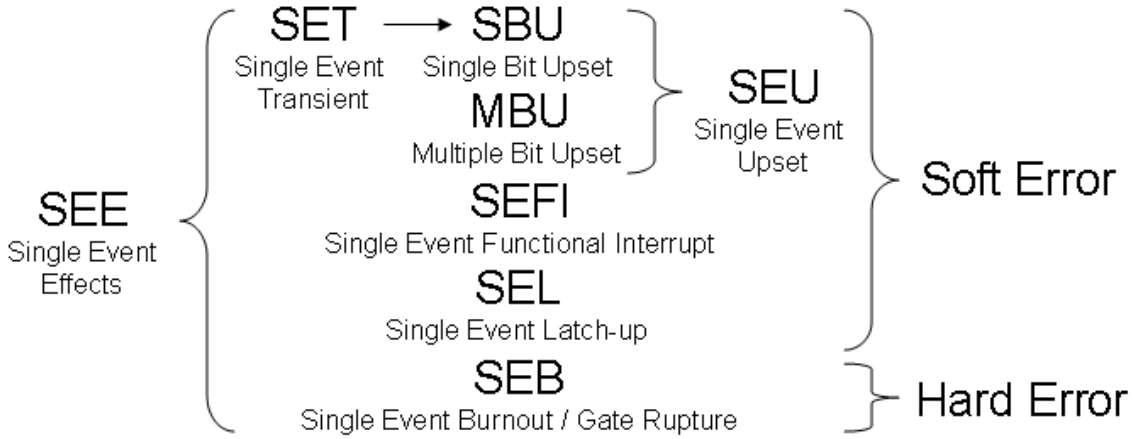


Figure 2.2: The different classes of soft and hard errors collectively known as Single Event Effects [3].

This work focuses on the effects of a subset of soft errors known as Single Event Upsets (SEU). An SEU occurs when deposited charge directly causes a change of state in a dynamic circuit memory element (flip-flop, latch, etc.). The change in state of one node is a single bit upset (SBU). The change in state of more than one node is a multiple bit upset (MBU). Both types of SEUs cause faults which can be repaired dynamically or by a power off/on.

2.3 Static SEU Cross Section

The total fraction of a device sensitive to SEUs is often referred to as its static SEU cross section (measured in units of cm^2) [7]. A device's static SEU cross section is a function of the sensitive volume at each node. A single node's sensitive volume corresponds to the amount of charge that must be collected (in the node) from an external charge deposition to change its state. The static SEU cross section of an entire device is proportional to the sum of the sensitive volumes of *all* nodes.

Since different particle energies deposit different amounts of charge, a node's SEU susceptibility varies with particle energy. E.L. Petersen *et al.* at the Naval Research Laboratory proposed that per-bit static cross section *vs.* energy follows a Weibull distribution [8]. Figures 2.3 and 2.4 depict the Xilinx Virtex XCV1000 FPGA proton and heavy-ion per-bit cross sections respectively, fit to a Weibull distribution. In both figures, the x axis is energy and the y axis is per-bit cross section. The formula for a heavy-ion Weibull distribution is,

$$F(L) = \sigma_{sat}(1 - e^{-[(L-L_o)/W]^s}), \tag{2.1}$$

where,

$F(L)$ = Heavy Ion SEU cross section in μ^2/bit,

σ_{sat} = limiting or plateau cross section in μ^2,

L = effective LET (linear energy transfer) in $MeV - cm^2/mg$,

L_o = upset threshold LET in $MeV - cm^2/mg$,

W = dimensionless width parameter,

and

s = dimensionless exponent parameter.

The formula for a proton Weibull distribution is,

$$F(x) = \sigma_{sat}(1 - e^{-[(x-x_o)/W]^s}), \tag{2.2}$$

where,

$F(x)$ = Proton SEU cross section in cm^2/bit,

$$
\begin{aligned}
\sigma_{sat} &= \text{limiting or plateau cross section in } cm^2, \\
x &= \text{proton energy in } MeV, \\
x_o &= \text{onset energy in } MeV, \\
W &= \text{dimensionless width parameter,} \\
\text{and} & \\
s &= \text{dimensionless exponent parameter.}
\end{aligned}
$$

These formulas and the specific parameter values for the Xilinx XCV100 can be found in [9]. The figures show that at higher energies, the per-bit cross section or node sensitivity goes up. However, above a particular energy the size of the cross section plateaus or saturates. Total device cross section estimates, such as that shown in Table 2.1 for the XCV1000, often assume worst-case, or saturation cross section for each bit.

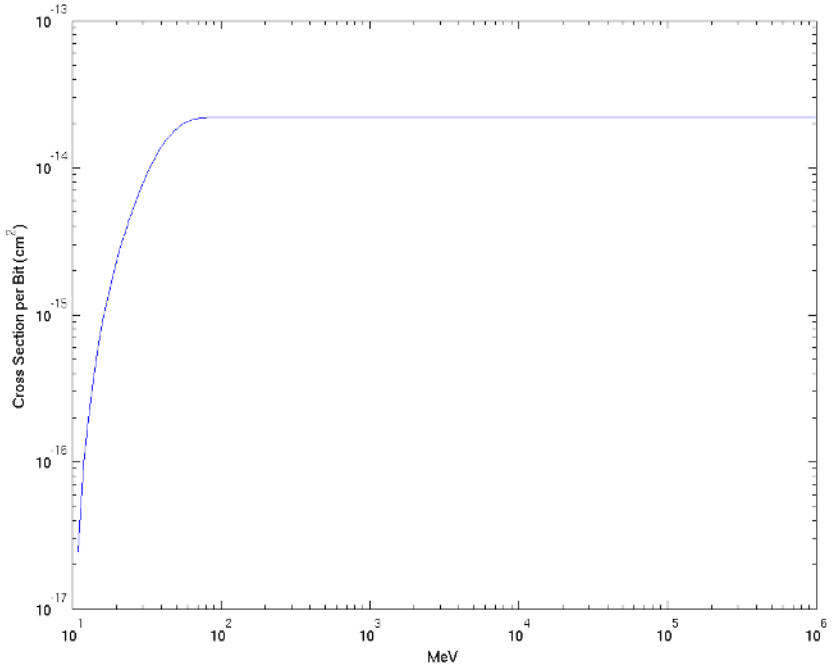


Figure 2.3: Proton SEU configuration latch per-bit cross sections for the XCV1000 FPGA fit to the Weibull distribution.

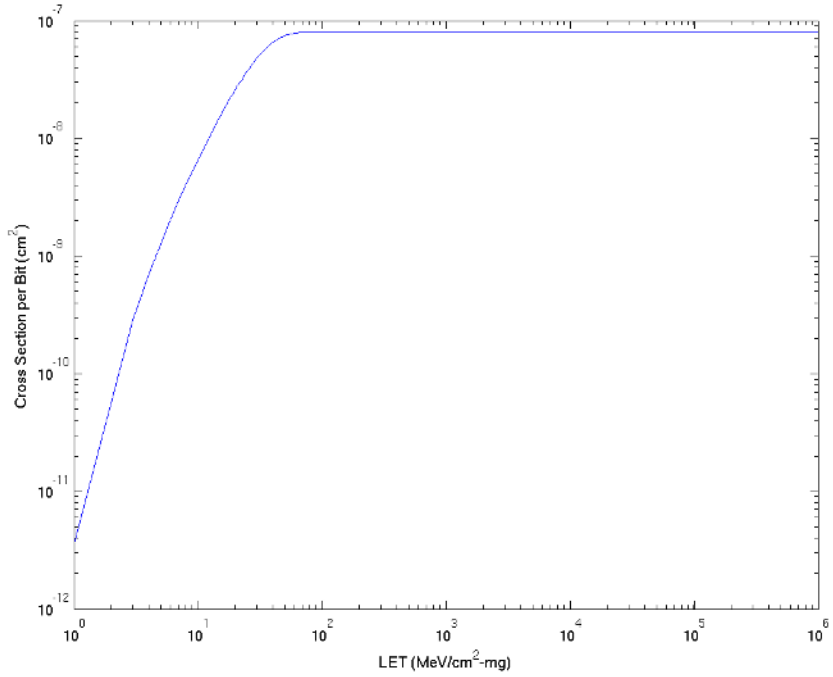


Figure 2.4: Heavy ion SEU configuration latch per-bit cross sections for the XCV1000 FPGA fit to the Weibull distribution.

2.4 FPGA Static SEU Cross Section

The signature of a device's per-bit Weibull cross section varies based on the physical characteristics of its sensitive nodes. In an FPGA, sensitive nodes include configuration memory cells, user flip-flops and user memory cells. Figure 2.5 represents the architecture of a simplified FPGA logic block. Latch cells hold the configuration of the look-up tables (LUT), inter-block routing (not shown in the figure) and other miscellaneous configurable logic. Other memory element architectures are used for the flip-flops and user memories. The sensitive volume characteristics of the configuration cells, flip-flops and user memories dictate the shape of the Weibull per-bit cross section distribution.

Table 2.1 enumerates the total SEU cross section of a Xilinx XCV1000 FPGA as well as the per-bit cross section for each type of sensitive node. The sum of the

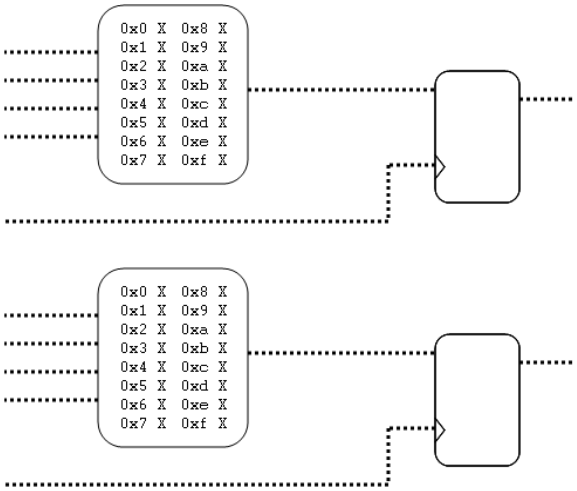


Figure 2.5: A simplified schematic representation of one configurable logic block in an FPGA.

number of each node type multiplied by its respective per-bit cross section equals the total static cross section. This particular device has approximately 5.8×10^6 configuration latches and 24×10^3 user flip-flops. The user flip-flops have a larger per bit cross section, but have a less significant impact on total cross section due to the smaller number of flip-flop nodes. The configuration memory cells comprise more than 95% of the total cross section. As is the case with most RAM-based FPGAs, the configuration memory dominates the static cross section.

Table 2.1: Xilinx Virtex XCV1000 FPGA Static SEU Proton Saturation Cross Section[1]

Circuit Resource	Cross Section (cm^2/bit)	Number on Chip	Cross Section (cm^2)	% of Total Cross Section
Config. Latch	2.2×10^{-14}	5.8×10^6	1.3×10^{-7}	95.5%
Flip-Flop	1.8×10^{-14}	2.4×10^4	4.3×10^{-10}	0.3%
Block-RAM	4.4×10^{-14}	1.3×10^5	5.7×10^{-9}	4.2%
POR	1.8×10^{-14}	43	7.7×10^{-13}	0.0%
Total	-	-	1.36×10^{-7}	100%

Like SRAM memory, an FPGA's total static cross section scales with the size of the device (i.e. number of bits or nodes). Table 2.2 lists the proton static cross section for several FPGAs. Notice that the per-bit cross section is constant for a particular device family, but the total device static cross section scales with the number of bits. Consequently, smaller FPGAs have a smaller static cross section while larger FPGAs have a larger static cross section.

Table 2.2: Proton Static Cross Section Measurements for the Configuration Memory of a Set of Xilinx FPGAs

Device	# Configuration Bits	Proton Static Cross Section (cm^2) per-bit	total
Virtex 300	1.6×10^6	2.2×10^{-14}	3.5×10^{-8}
Virtex 600	3.1×10^6	2.2×10^{-14}	6.9×10^{-8}
Virtex 1000	5.8×10^6	2.2×10^{-14}	1.3×10^{-7}
Virtex II 1000	2.8×10^6	3.8×10^{-14}	1.1×10^{-7}
Virtex II 3000	7.3×10^6	3.8×10^{-14}	2.8×10^{-7}
Virtex II 6000	1.6×10^7	3.8×10^{-14}	6.2×10^{-7}

2.5 Orbit Specific Static SEU Rates

Predicting the frequency of radiation-induced faults in a given device requires knowledge about device's static cross section and destined environment. Methods exist to calculate upset rates based an orbit's estimated energy spectrum and a device's sensitive cross section characteristics described by a Weibull distribution [8, 10, 3, 7]. Readily available computer codes aid in automatic calculation of upset rates.

Static SEU rates for a Xilinx XCV1000 FPGA in several orbits are listed in Table 2.3. Appendix E completely documents the software and parameters used to generate the rates found in this table. The results shows that during Solar Minimum[1] conditions in the example low-earth orbit (LEO), the configuration memory will experience approximately 2.8 configuration upsets every one-hundred hours and roughly

[1]See Appendix B for more information on solar conditions.

1.9 configuration upsets per one-hundred hours during "solar flare" conditions. In a global positioning system (GPS) orbit, approximately 4.5 configuration memory configuration upsets will occur every one-hundred hours during Solar Minimum and approximately 13 configuration upsets will occur per *one hour* during flare conditions. In contrast, Xilinx Corporation estimates that the XCV1000 will statically experience one upset every 2.2×10^6 hours when operating at sea level [11].

The astute reader will notice that in some situations the Solar Maximum SEU rates are actually smaller than the Solar Minimum SEU rates. During Solar Maximum, increased solar activity increases the particle flux outside the magnetosphere. At altitudes below the magnetosphere, however, the magnetic field shields a large portion of the solar particles thus the trapped particle flux dominates even during increased solar activity.

Table 2.3: Static SEU Rate Forecast for a Single Xilinx Virtex XCV1000 FPGA in Several Different Orbits

Orbit	Alt. (km)	Incl. (deg)	Solar Minimum (SEU/hr)	Solar Maximum (SEU/hr)	Worst Week (SEU/hr)	Worst Day (SEU/hr)
LEO	560	35.0^o	2.8×10^{-2}	1.8×10^{-2}	1.9×10^{-2}	1.9×10^{-2}
Polar	833	98.7^o	6.9×10^{-2}	5.5×10^{-2}	1.1	3.8
GPS	$22,200$	55.0^o	4.5×10^{-2}	4.6×10^{-1}	3.7	$1.3 \times 10^{+1}$

2.6 Summary

This chapter introduced the space radiation environment and FPGA radiation effects. The static SEU cross section of an FPGA was also defined. Radiation environment models and static cross section measurements for a Xilinx XCV1000 FPGA were used to calculate static on-orbit upset rates in three benchmark orbits. These upset rates, presented in Table 2.3, do not represent failure rates for any particular FPGA application. The following chapter will show that application failure rate depends on other factors in addition to static SEU rates. All failure rates presented

throughout this thesis are based on the static cross section numbers reported in Table 2.3. Each of the subsequent chapters (excluding Chapters 4 and 7) will present failure rates based on the values presented in this chapter. These failure rates will be used to prove that the mitigation technique introduced in this thesis provides acceptable reliability at lower resource costs by giving up availability.

Chapter 3

Dynamic Cross Section

The previous chapter described the space radiation environment, introduced the concept of single event effects and showed that a device's sensitivity to single event effects can be measured as a static cross section. However, any unique FPGA configuration (user circuit design) does not typically utilize an entire device. Research at Brigham Young University demonstrated that an SEU in an unutilized FPGA resource typically does not affect a circuit's proper operation. Consequently, FPGAs have an operational (dynamic) cross section usually much smaller than their total static cross section [12, 13, 14].

When programmed with a user circuit design, an FPGA's configuration memory has a unique dynamic cross section. FPGA dynamic cross section is the fraction of an FPGA's static cross section sensitive to SEUs. A design's utilization of programmable routing, logic, and I/O resources determines its dynamic cross section. Specifically, nodes which generate functional errors when upset belong to a design's dynamic cross section. Since a design never uses all of an FPGA's resources, the dynamic cross section is generally much smaller than the static cross section.

Since dynamic cross section properly accounts for a circuit's actual resource utilization, it should be used to more accurately estimate an application's failure rate. Such failure rate estimates are calculated by simply scaling static SEU rate by the size of an application's dynamic cross section. Calculating failure rate in this manner leads to more accurate estimates.

This chapter begins with a description of dynamic functional errors and the methodology used to measure dynamic cross section. Next, measurements of dynamic

cross section for a few sample applications are presented. Finally, estimated failure rates for these applications are reported showing how to more accurately calculate an application's failure rate based on its dynamic cross section.

3.1 Dynamic Single Event Upsets in FPGAs

An FPGA resource utilized by a specific design and corrupted by an SEU in the configuration memory may generate functional errors. Figures 3.1 and 3.2 illustrate how an SEU alters an FPGA design and induces functional errors. Figure 3.1 depicts a simplified programmable logic block, the standard programmable unit in an FPGA. The two 4-input function generators (i.e. Look-Up Tables (LUT)) store their configuration in 16x1 memories. In the figure, the top LUT's configuration implements a logical 2-input OR function. Since only 2 inputs are used, 3/4 of the LUT's contents don't matter. The bottom LUT is not configured so none of the contents in this LUT matter. Figure 3.2 depicts the same logic block after a particle strike and subsequent single bit upset (SBU) at address 0x3 in the top LUT. In this case, the upset modified the LUT so that it now performs a 2-input logical XOR function. Until the upset bit is restored to its proper configuration, the LUT will execute the logical XOR function and can produce functional errors. In this example, the affected configuration bit belongs to the dynamic cross section.

In the preceding example a particle strike and subsequent SEU in the bottom LUT would not have adversely altered the programmed application. The unutilized LUT's configuration memory is loaded with "don't care" values. These unprogrammed configuration bits belong to the non-dynamic cross section. An upset in this LUT would change the configured logic function, but would not cause functional errors during operation. Since only utilized resources affect a design's proper operation, only a subset of all configuration bits belong to the dynamic cross section. Therefore, dynamic cross section is smaller than an FPGA's static cross section.

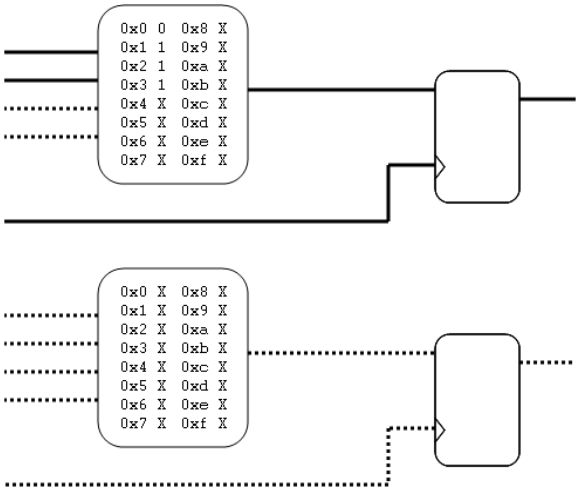


Figure 3.1: Representation of an FPGA configurable logic block. The top half of the block is programmed to perform a 2-input logical OR function. The output of this function is latched in a user flip-flop. The bottom half of the block is not configured.

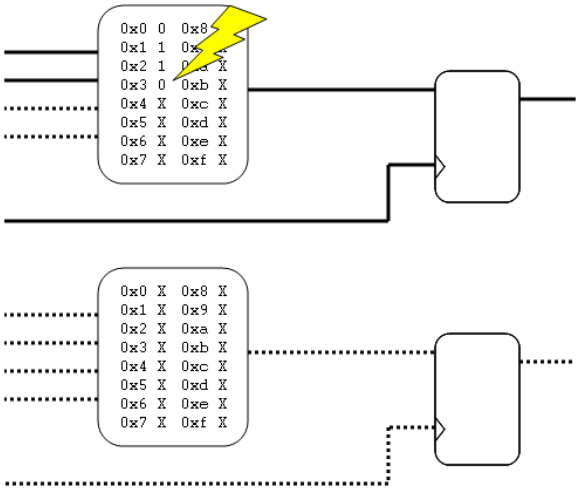


Figure 3.2: Representation of a configured FPGA logic block upset by an SEU. In this case, the particle strike corrupted the top LUT so that its function switched to a 2-input logical XOR.

3.2 Measuring Dynamic Cross Section

Two techniques aid in estimating and measuring dynamic cross section. Fault-injection predicts or estimates a design's dynamic cross section. Particle irradiation

more formally measures dynamic cross section and validates fault-injection predictions.

Analytical techniques such as fault-injection estimates dynamic cross section size by injecting faults into an FPGA's configuration bitstream and simultaneously monitoring the FPGA's outputs for functional errors. Configuration memory locations which induce functional errors when upset belong to the dynamic cross section.

A fault-injection tool developed by Eric Johnson at Brigham Young University accurately estimates dynamic cross section [15]. This tool estimates dynamic cross section for a given FPGA design by artificially upsetting individual bits within the configuration memory of an FPGA device under test (DUT). The tool identifies "sensitive" configuration bits, or bits which belong to the dynamic cross section. Johnson's tool can rapidly test all configuration bits in a bitstream to create an accurate and complete characterization of the dynamic cross section of a given FPGA design. With Johnson's tool, the entire 5.8×10^6 bit configuration memory of a Xilinx Virtex XCV1000 can be tested in approximately 20 minutes [12].

The predicted size of a design's dynamic cross section, x_{d_p}, is equal to the fraction of "sensitive" configuration bits identified in the design, multiplied by the device's measured static cross section

$$x_{d_p} = x_s \times \frac{\#\text{ sensitive bits}}{\#\text{ total bits}}, \tag{3.1}$$

where,

x_{d_p} = predicted dynamic cross section,

and

x_s = measured static device cross section.

Since static cross section has units of cm^2 and the ratio of sensitive to total bits is unit-less, the result has units of cm^2.

To validate fault injection estimates, more formal measurements of dynamic cross section are taken using proton testing. Measured dynamic cross section, x_{d_m}, is

the ratio of functional errors to particle fluence, or mathematically,

$$x_{d_m} = \frac{\#\text{ error events}}{\text{fluence} \times \cos\theta}, \tag{3.2}$$

where,

x_{d_m} = measured dynamic cross section,

and

θ = incident particle angle [7, 16].

The particle fluence is scaled by the cosine of the incident particle angle to account for variance in fluence due to angle of incidence. Since fluence has units of $(\#/cm^2)$ and functional errors has units of $(\#)$, the result has units of cm^2.

3.3 Dynamic Cross Section Measurements

Fault-injection and proton irradiation were used to estimate the dynamic cross section for the four benchmark FPGA designs introduced in Section 1.1. Johnson's fault-injection tool was used for estimates and the 63 MeV proton source at Crocker Nuclear Laboratory in Davis, CA, was used for formal measurements. Table 3.1 reports the predicted and measured size of dynamic cross section for each of the tested applications. The parameters necessary to compute dynamic cross section with Equations 3.1 and 3.2 are also reported. The results show that fault-injection is at least as good as an order-of-magnitude estimate of measured cross section. Previous research by Johnson *et al.* showed their fault-injection tool to be accurate to within 1% [12]. Known differences between the two testing methodologies account for the error. More information about the correlation of estimated and measured dynamic cross section results can be found in Johnson's thesis [4].

The signature of a design's dynamic cross section correlates to the mapping and routing of resources the design utilizes. To show this correlation, graphical representations of resource utilization and dynamic cross section for the Counter, Synthetic and DSP Kernel designs are depicted in Figures 3.3, 3.4 and 3.5 respectively. In each

Table 3.1: Dynamic Cross Section Predictions and Measurements†

Design	Predicted sensitive bits (#)	Predicted size (cm^2)	Measured events (#)	Measured fluence ($\frac{\#particles}{cm^2}$)	Measured θ (deg)	Measured size (cm^2)	% error††
Multiplier	516,494	1.1×10^{-8}	597	7.2×10^{10}	0^o	8.3×10^{-9}	-32.5%
Counter	192,717	4.2×10^{-9}	833	2.4×10^{11}	0^o	3.5×10^{-9}	-20.0%
Synthetic	179,384	4.0×10^{-9}	862	2.7×10^{11}	0^o	3.2^{-9}	-31.3%
DSP Kernel	502,082	1.1×10^{-8}	2,737	3.1×10^{11}	0^o	8.9×10^{-9}	-15.8%

†The device used in testing was a $0.22\mu m$ 5-layer epitaxial process Xilinx Virtex XCV1000 FPGA with a static cross section of 1.28×10^{-7} cm^2 [1].
††$\%error = (measured - predicted)/measured$

figure, the left graphic is a screen capture of the design's utilization from the resource editor tool, `fpga_editor`, provided by the manufacturer. The right graphic is a `Matlab` rendering of the design's dynamic cross section as determined by Johnson's fault-injection tool. The marked points indicate "sensitive" configuration memory bits, according to the physical layout. In other words, marked points indicate a bit which caused functional errors when upset. In both circuits, the location of sensitive bits correlates with device resource utilization.

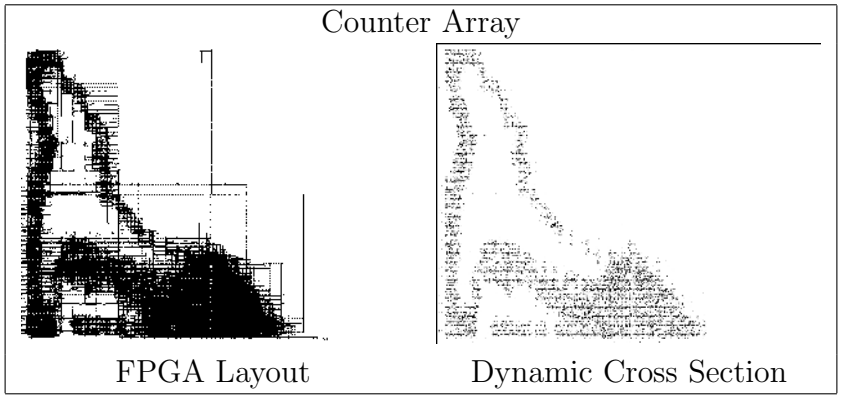


Figure 3.3: Comparison of the resource utilization and dynamic cross section for a Counter Array design. The diagram on the left is a screen capture of the resource layout of the design. The right diagram is a graphical representation of the portion of the Counter Array design layout which constitutes its dynamic cross section.

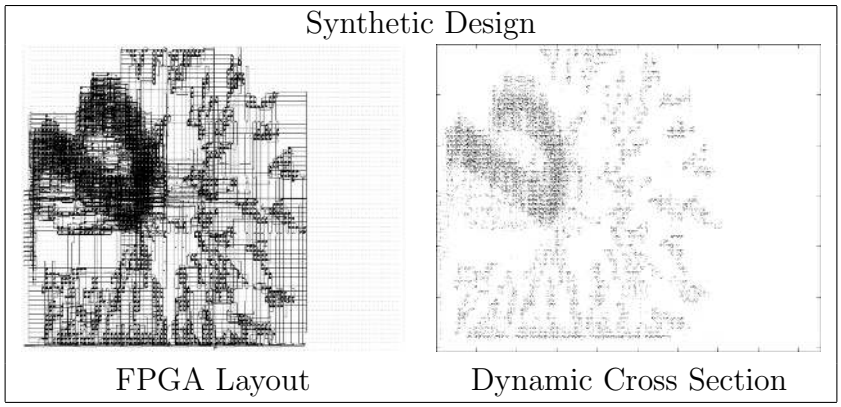


Figure 3.4: Comparison of resource utilization and dynamic cross section for the Synthetic design.

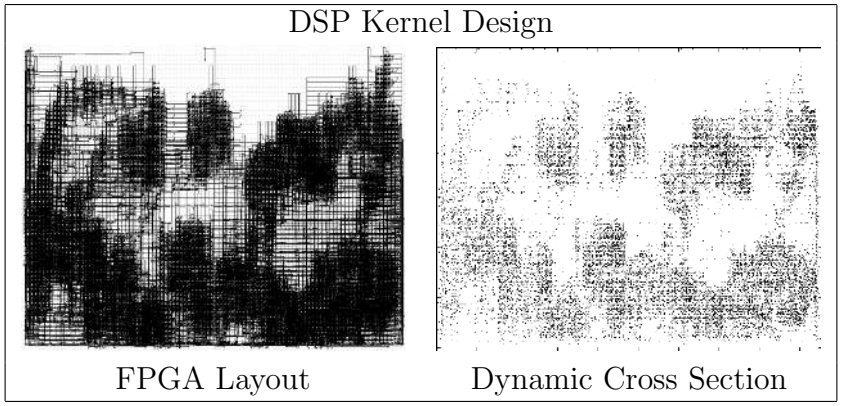


Figure 3.5: Comparison of resource utilization and dynamic cross section for the DSP Kernel design.

Figures 3.3, 3.4 and 3.5 allude to the large disparity between the size of static and dynamic cross section. Figure 3.6 illustrates this disparity. The figure depicts the relative size of the static cross section of a Xilinx Virtex XCV1000 FPGA compared to the dynamic cross section of the four designs listed in Table 3.1. In all cases, the dynamic cross section is at least one order of magnitude smaller than the static cross section.

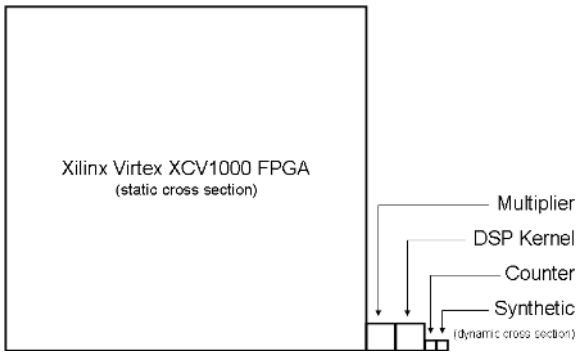


Figure 3.6: Comparison of the relative size of the dynamic to static cross section for several designs.

3.4 MTBF Estimation

Measuring dynamic cross section ultimately leads to more accurate predictions of a design's reliability performance in a specific environment. Failure rate or it's inverse, Mean Time Between Failure (MTBF), are two metrics which measure reliability performance in a real system. Failure rate is the rate per unit time of critical service interruptions in a system. MTBF is the inverse of failure rate or average time between critical interruptions of service. This section explains how to calculate these two quantities and presents MTBF numbers for a few sample applications.

3.4.1 Calculating MTBF

In an FPGA, an application's on-orbit dynamic failure rate λ_d is calculated as the ratio of its dynamic cross section to the FPGA's static cross section, multiplied by the FPGA's static SEU rate in the specified orbit, or mathematically,

$$\lambda_d = \frac{Dynamic\ Cross\ Section}{Static\ Cross\ Section} \times Static\ SEU\ Rate. \tag{3.3}$$

In other words, the fraction of SEUs which cause design failures is directly proportional to the design's particular sensitive subset of the FPGA's static cross section. Since the ratio of cross sections in Equation 3.3 is unit-less, failure rate has the same units as static SEU rate, or events per unit time.

Dynamic mean time between failure, $MTBF_d$, is simply the inverse of dynamic failure rate. Stated mathematically, the dynamic MTBF is

$$MTBF_d = \frac{1}{\lambda_d}. \tag{3.4}$$

As such, $MTBF_d$ has inverse units of λ_d, or time per failure.

3.4.2 Orbit-Specific MTBF Estimates

Table 3.2 reports MTBF estimates for each application listed in Table 3.1 and introduced in Chapter 1. The MTBF values presented here were calculated using Equations 3.3 and 3.4. For each orbit, the static SEU rate parameter in Equation 3.3 comes from Table 2.3. All numbers assume the specified design is configured into a single Xilinx Virtex XCV1000 FPGA.

According to Table 3.2, the Multiplier application will fail once every 410 hours during Solar Minimum[1] conditions in the specified Low-Earth orbit. On the other hand, during the Worst Day[1] solar conditions in the GPS orbit, the Multiplier application will fail once every .85 hours, or approximately once every 51 minutes. This result is consistent with characteristics of the GPS orbit's intense radiation environment. Since a design experiences a higher particle flux in the GPS orbit, it will fail more often.

Table 3.2 also reveals that, in all orbits, the Counter design will fail approximately three times less often than the Multiplier circuit. This results is consistent with the cross section measurements found in Table 3.1. The Multiplier has a dynamic cross section of $1.1 \times 10^{-8}\ cm^2$, while the Counter's dynamic cross section is approximately three times smaller at $4.2 \times 10^{-9}\ cm^2$. As a result, the Counter design will always fail approximately three times less often than the Multiplier design when operating in equivalent radiation conditions.

3.5 Summary

Only utilized resources in an FPGA induce dynamic functional errors when upset. As a result, the fraction of an FPGA sensitive to an SEU is application

[1]See Appendix B for more information on solar conditions.

Table 3.2: On-Orbit Mean Time Between Failure Estimates

Design	Solar Conditions	Mean Time Between Failure (hours)		
		LEO†	Polar††	GPS†††
Multiplier	Solar Min	4.1×10^2	1.6×10^2	2.5×10^2
	Solar Max	6.1×10^2	2.1×10^2	2.4×10^2
	Worst Day	5.6×10^2	3.0	8.5×10^{-1}
Counter	Solar Min	1.1×10^3	4.3×10^2	6.7×10^2
	Solar Max	1.6×10^3	5.5×10^2	6.5×10^2
	Worst Day	1.5×10^3	8.0	2.3
Synthetic	Solar Min	1.2×10^3	4.7×10^2	7.2×10^2
	Solar Max	1.8×10^3	5.9×10^2	7.0×10^2
	Worst Day	1.6×10^3	8.6	2.4
DSP Kernel	Solar Min	4.2×10^2	1.7×10^2	2.6×10^2
	Solar Max	6.3×10^2	2.1×10^2	2.5×10^2
	Worst Day	5.7×10^2	3.1	8.7×10^{-1}

† Low-Earth Orbit at 560 km altitude, 35.0^o inclination
†† Polar Orbit at 833 km altitude, 98.7^o inclination
††† Global Positioning System Orbit at $22,200$ km altitude, 55.0^o inclination

specific. This fraction or dynamic cross section can be predicted with fault-injection or measured with particle irradiation. Using dynamic cross section to estimate FPGA application failure rates yields less pessimistic estimates than using static cross section because dynamic cross section accounts for the unused portions of an FPGA. Dynamic cross section was measured for four benchmark applications. The results were used to estimate failure rates for these applications in three orbits.

Chapter 4

Persistent Functional Errors

As explained in Chapter 3, failure rate predictions should use dynamic cross section since dynamic cross section accounts for the resource utilization of a particular application. However, this prediction methodology ignores the possibility of different functional error modes. It treats all dynamic functional errors equally. This thesis will show that dynamic functional errors can actually be divided into two distinct categories, non-persistent and persistent. Each error mode uniquely affects a system's operation. As a result, failure rate estimates should account for both types. Furthermore, a later chapter will show that sophisticated SEU mitigation strategies can exploit non-persistent functional errors to improve reliability at lower costs.

This chapter begins with a review of related research in error mode classification. Next, the chapter reviews configuration memory scrubbing. Scrubbing makes redundancy-based functional error mitigation schemes possible in an FPGA. Furthermore, scrubbing makes the distinction between non-persistent and persistent error modes possible. Next, non-persistent and persistent functional errors are defined. Examples are presented to demonstrate the typical signature and side-effects of each error type. The distinction between each error type is important for a novel functional error mitigation methodology presented in Chapter 6.

4.1 Related Research

Classification of error modes is not a new idea. For example, researchers in [17] and [18] proposed separating control logic errors from other errors. They termed these control logic errors Single Event Functional Interrupts (SEFI) because a single upset

in the control logic can render a device inoperable. Carmichael *et al.* also proposed dividing FPGA functional errors into three categories: SEFIs, configuration memory errors and user logic functional errors. Configuration memory errors occur as the result of an upset in the user-programmable configuration memory of an FPGA. An upset in the configuration memory can modify the programmed circuit, indirectly causing functional errors to be latched in user memory or flip-flops. In contrast, user logic functional errors occur as the result of an upset directly in user memory or a user flip-flop [19].

Earl Fuller and Michael Caffrey *et al.* were the first to mention the concept of different dynamic functional error modes. They suggested that some functional errors permanently interrupt service while others only temporarily interrupt service. Furthermore, they proposed that systems could more cost effectively improve reliability by simply tolerating temporary blocks of bad data generated by functional errors which temporarily interrupt service [20]. The ideas presented in this thesis are built on these concepts.

4.2 Scrubbing

Functional errors which temporarily interrupt service are only possible in an FPGA if some form of configuration memory scrubbing is employed. In a general sense, scrubbing is the process of repeatedly scanning for and correcting upset bits in the configuration memory. Carl Carmichael from Xilinx Corporation suggested two of the many various scrubbing schemes [21]. In Carmichael's first method, an external circuit reads the configuration memory, compares it against a golden copy, and repairs all upset bits. The process repeats in a round-robin manner, continuously or at pre-defined intervals. Figure 4.1 depicts a single step in one iteration of this method. In the figure, the highlighted box represents a single frame of data read from the configuration memory. If any of the bits in this frame do not match the golden copy, the entire frame is rewritten. In Carmichael's second method the *entire* configuration memory is reloaded at pre-defined intervals regardless of the presence

or absence of errors [21]. Other scrubbing schemes also exist. One popular method speeds up the check-for-upsets step by using a checksum on each frame.

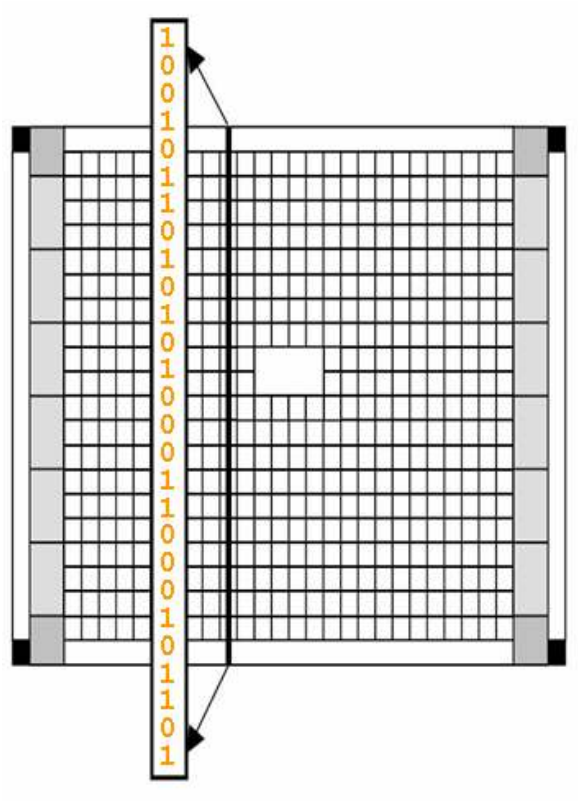


Figure 4.1: A typical scrubbing circuit reads the configuration memory one block at a time, compares each block against a golden copy, and repairs all upset bits.

The frequency at which a system can perform scrubbing is a function of the available hardware, the size of the memory to be scrubbed and the implemented scrubbing method. The total time to perform one scrub t_s is defined by the equation

$$t_s = nt_{rd} + nt_{ck} + kt_{rp}, \tag{4.1}$$

where t_{rd} is the time to read a single block of memory, t_{ck} is the time required to check the contents of one block and t_{rp} is the time to repair one block of memory. The scalar n is the number of blocks in the memory. The scalar k is the number of blocks which require repair.

Many factors dictate how quickly one scrub can be performed. A method which directly checks each value takes longer than a checksum method. The size of the constants t_{rd}, t_{ck} and t_{rp}, which depend on the speed of the scrubbing hardware and memory, also affect the speed of scrubbing. Additionally, the size of the scalars n and k, which depend on memory size and SEU rate, affect the average time to complete one scrub. The scrub time t_s of a real system depends on all of these factors. For example, a checksum version of scrubbing implemented at Los Alamos National Laboratory for a single Xilinx Virtex XCV1000 FPGA averaged 22 milliseconds.

All scrubbing methods, regardless of speed, provide two key benefits essential to reliability enhancement. First, scrubbing prevents the buildup of multiple faults in an FPGA's configuration memory. Without this benefit, combinations of multiple faults, which individually don't cause problems, could collectively generate functional errors. Perhaps more importantly, redundant reliability techniques, such as Triple Modular Redundancy (TMR), which assume just a single fault at any given time, would be ineffective. In TMR, multiple faults could eventually form a majority and break the redundancy.

Scrubbing also provides a second important benefit. Scrubbing limits the time a fault exists in memory. As a result, scrubbing bounds how long a given fault can generate functional errors. An upper bound U on error generation time is given by the equation

$$U = nt_{rd} + nt_{ck} + nt_{rp}, \tag{4.2}$$

where the scalar n on t_{rp} indicates a worst-case scenario in which every block must be repaired. This bound guarantees that a fault does not generate functional errors indefinitely.

4.3 Non-Persistent Errors

Although scrubbing benefits a system's reliability in many ways, it does not actually mitigate functional errors. Furthermore, scrubbing does not prevent functional errors from occurring. Scrubbing only guarantees a corrupted circuit will be repaired within a bounded time. Scrubbing does not and cannot eliminate functional

errors since the dynamic or user logic memory elements' contents cannot be known *a priori*. To force-ably clear dynamic functional errors stored in these memory elements, a system must apply a global reset.

Reseting an entire system to eradicate functional errors may sometimes be overly conservative. In many cases, functional output errors that occur after an SEU exist only temporarily. Shortly after scrubbing repairs an SEU-induced fault, the functional errors go away and all signs of system failure vanish. These functional errors are termed *non-persistent* because they do not persist in a design after configuration scrubbing. Non-persistent functional errors only require time, not reset, to flush from a system. The ability to tolerate temporary functional errors is system dependent. This topic will be discussed in more detail in Section 5.4.1.

An example best demonstrates the concept of non-persistent errors. Two data streams were collected from two identical circuits operating within the fault-injection tool introduced in Section 3.2. The arithmetic difference between these two data streams was computed to identify the impact of configuration upsets. Zero arithmetic difference indicated both circuits operated correctly. A non-zero arithmetic difference signified that one of the circuits had functional errors.

Figure 4.2 illustrates the arithmetic difference between the two circuits' data streams for a specific snapshot in time. The x axis is cycle count and the y axis is the arithmetic difference between the two circuits. For the first 64 time cycles in Figure 4.2, the difference between the two data streams was zero indicating that both circuits operated the same. At time cycle 65 an artificial SEU flipped the value of a configuration bit in one circuit, corrupting its configuration memory. The faulty circuit induced functional errors and immediately the outputs of the two circuits diverged. At time cycle 130, an artificial form of scrubbing fixed the corrupted bitstream. At this point the faulty circuit resumed proper operation and shortly thereafter the difference between the data streams returned to zero. Functional errors only occurred from time cycle 65 to 132.

In this example, the configuration upset caused non-persistent errors, or a temporary interruption of service. A system which can tolerate non-persistent functional

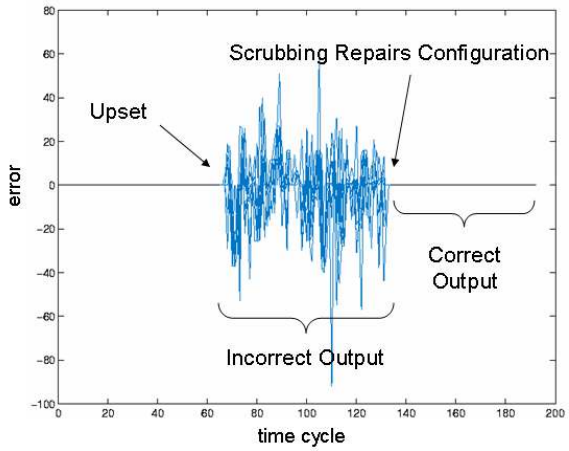


Figure 4.2: Plot of the difference between the outputs of a DUT and golden circuit before, during and after a configuration upset. The upset generated non-persistent functional errors.

errors does not require a global reset to restore service after non-persistent functional errors. These systems can simply throw away the bad data and continue operating.

The exact non-persistent error signature in Figure 4.2, both in magnitude and time, is a function of the design used as well as the upset bit and set of input vectors. In a general sense, however, the error signature represents the typical temporal characteristics of non-persistent functional errors. The arithmetic difference between the outputs of a DUT and golden circuit always return to zero after scrubbing. In other words, a circuit will resume proper operation with proper output data after scrubbing repairs a fault that induced only non-persistent errors.

4.4 Persistent Errors

Not all configuration upsets cause non-persistent errors. Although scrubbing restores a circuit's proper configuration, a fault can still insert incorrect functional state into a circuit. In some cases the circuit will indefinitely propagate the incorrect state. In this error mode the system continues to exhibit signs of system failure, or functional errors, even after scrubbing. Permanent functional errors within a system are termed *persistent* because they persist beyond repair.

Since the duration of persistent errors is indefinite, they represent a permanent interruption of service. Unlike non-persistent errors, persistent errors do not disappear after configuration scrubbing. As a result, persistent errors cannot be corrected, and will not self-correct, without a global system reset.

Figure 4.3 depicts a persistent error, or permanent service interruption. Like the non-persistent example shown in Figure 4.2, the plot represents the arithmetic difference between the outputs of two identical circuits over time. For the specific time period plotted in Figure 4.3, the output stream for the two circuits matched for the first 64 cycles. At time cycle 65 a configuration bit in one circuit was upset. Immediately the arithmetic difference became non-zero, indicating the circuits' outputs diverged. At time cycle 130 configuration scrubbing repaired the bitstream. Unlike the non-persistent example, the output streams in this example did not converge after scrubbing. The internal state trapped the errors and propagated them even after scrubbing fixed the configuration bit. Since the application continued to produce faulty data after repair, it needs a system reset to recover.

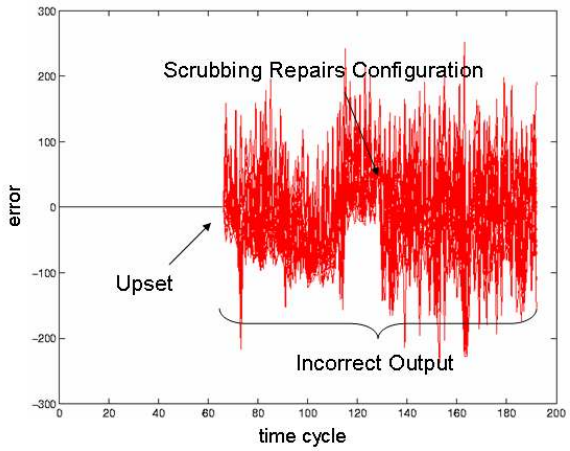


Figure 4.3: Plot of the difference between the outputs of a DUT and golden circuit before, during and after a configuration upset. The upset generated persistent functional errors.

Like the non-persistent error example, the exact persistent error signature in Figure 4.3, both in magnitude and time, is a function of the design used as well as the upset bit and set of input vectors. In a general sense, however, the error signature represents the typical temporal characteristics of persistent functional errors. The arithmetic difference between the outputs of a DUT and golden circuit will likely remain non-zero indefinitely. In other words, even though a circuit will resume proper operation after scrubbing, the circuit will continue to generate bad data as a result of the persistent functional errors still in the system.

From the overall system perspective, persistent functional errors may look like a Single Event Functional Interrupt (SEFI). A SEFI in an FPGA completely eliminates access to its control functions such as configure, read-back, etc. An FPGA often requires a power off/on to recover from a SEFI [3]. Persistent functional errors, however, are specific to the configuration programmed into the FPGA, not the FPGA itself. In addition, service can always be restored after persistent errors with a global reset.

4.5 Summary

An upset in an FPGA application's dynamic cross section will induce dynamic functional errors. In an FPGA system with configuration memory scrubbing, dynamic functional errors can be divided into two categories: non-persistent and persistent. Non-persistent functional errors exist temporarily. After scrubbing, non-persistent functional errors eventually disappear. The side-effect of non-persistent errors is a temporary block of corrupted data. In contrast, persistent functional errors permanently corrupt data. Even after scrubbing, persistent functional errors do not disappear. Most applications cannot tolerate persistent functional errors, but some applications may be able to tolerate non-persistent functional errors. Systems which can tolerate non-persistent errors can efficiently improve reliability by focusing error mitigation on just persistent functional errors. The following chapters will explore these ideas in more detail.

Chapter 5

Persistent Cross Section

As stated in Chapter 4, an upset in the dynamic cross section results in either non-persistent or persistent functional errors. Since the two error modes are mutually exclusive, the dynamic cross section can be divided into two components, non-persistent and persistent. Figure 5.1 depicts the division of the dynamic cross section into non-persistent and persistent components. A configuration upset in the non-persistent component of the dynamic cross section results in non-persistent errors. An upset in the persistent component yields persistent errors. Systems may realize acceptable levels of reliability, at lower costs, by focusing mitigation on just the persistent component of a design.

The structure of this chapter is as follows: the first two sections respectively define the non-persistent and persistent cross sections. Each section also introduces and defines a mapping from circuit elements to the non-persistent and persistent cross sections. The next section shows how to measure each of the cross sections. These measurements and the mappings from circuit components to cross section are particularly important for the development and validation of a mitigation method which focuses on just the persistent cross section. The mapping indicates specifically which circuit elements need reliability enhancement to mitigate persistent functional errors. Measuring the persistent cross section validates that such a mitigation scheme works. Finally, the last section introduces the idea of tolerant and intolerant functional error tolerance levels. In other words, this section defines what type of system can tolerate non-persistent functional errors. Modified estimates of MTBF are reported for each of the four benchmark applications at each tolerance level.

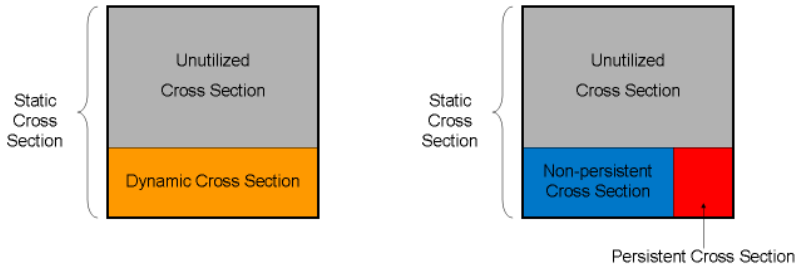


Figure 5.1: The diagram on the left represents the generic relationship between static and dynamic cross section. The diagram on the right shows the non-persistent and persistent components of the dynamic cross section.

5.1 Non-Persistent Cross section

A design's non-persistent cross section corresponds to circuit structures within feed-forward paths. Since feed-forward circuits by definition do not have feed-back, all functional errors within a feed-forward path must exit the path within a bounded time. Furthermore, once the source of the functional errors is repaired and stops generating errors, *all* functional errors in feed-forward paths, generated by that source, will disappear within a finite time.

Like the dynamic cross section, the size and exact footprint of the non-persistent and persistent cross sections are design dependent. The size of a design's non-persistent cross section is proportional to the number of resources and routing connections utilized in all feed-forward sections of the design. The exact footprint of a design's non-persistent cross section correlates to the mapping and routing of the feed-forward resources.

Figure 5.2 depicts a generic circuit with both non-persistent and persistent components. The non-persistent components are highlighted with the dashed line. The first four elements along the bottom path (two logic stages and two flip-flops) in addition to the final set of logic and flip-flop at the output belong to the non-persistent cross section. In this feed-forward path, an SEU-induced fault would generate functional errors and feed them downstream (to the right). As a result, once scrubbing repairs the configuration fault, all previously generated functional errors will eventually flush from the path.

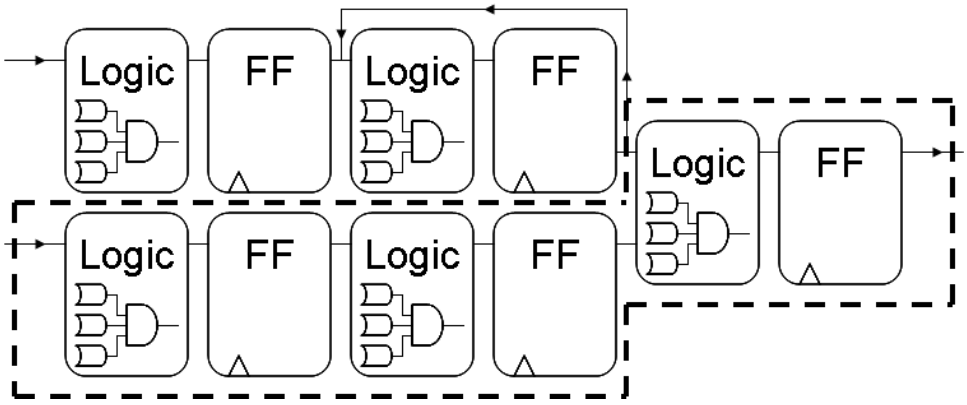


Figure 5.2: The feed-forward sections of a circuit, outlined with the dashed line, belong to the non-persistent cross section.

A specific example best demonstrates which parts of a circuit belong to the non-persistent cross section. A simple 4-bit full-adder has only a non-persistent cross section. Figure 5.3 depicts a simplified schematic of a 4-bit adder implemented in an FPGA. The implementation requires four LUTs, four XOR gates, four muxes and four flip-flops. The circuit computes the 4-bit sum s of two 4-bit operands, a and b. Table 5.1 depicts a sequence of inputs to this circuit. The table also lists the corresponding sequence of actual and desired outputs. For the first two cycles, the actual output and the desired output matched. During cycle three, an SEU flipped the bit stored at address 0x3 in the LUT second from the top (see Figure 5.4). As a result, the output on cycle three did not match the desired output. This faulty data represents a functional error. On the next cycle, bits $a2$ and $b2$ did not exercise the corrupted location in the affected LUT, so the actual and desired outputs again agreed. On cycle five, the operands exercised the corrupted location, resulting in invalid output. Finally, at the beginning of cycle six, scrubbing repaired the corrupted LUT value. The actual and desired outputs for cycles six through eight matched.

In the preceding 4-bit adder example, the circuit is completely feed-forward. As a result, functional errors are stored in the sum flip-flops for one cycle and then exit the circuit completely. In other words, this circuit can only generate non-persistent functional errors. Since an SEU anywhere within this circuit only causes

non-persistent functional errors, the entire circuit belongs to the non-persistent cross section.

Table 5.1: Sequence of Inputs and Outputs for a 4-bit Adder Circuit

Cycle	input a	input b	actual output s	desired output s
1	0b0001 (1)	0b0001 (1)	0b0010 (2)	0b0010 (2)
2	0b0011 (3)	0b1001 (9)	0b1100 (12)	0b1100 (12)
3	0b0110 (6)	0b0100 (4)	**0b0010 (2)**	**0b1010 (10)**
4	0b1001 (9)	0b0010 (2)	**0b1011 (11)**	**0b1011 (11)**
5	0b0111 (7)	0b0110 (6)	**0b1001 (9)**	**0b1101 (13)**
6	0b0101 (5)	0b0010 (2)	0b1110 (7)	0b1110 (7)
7	0b0110 (6)	0b0100 (4)	0b1010 (10)	0b1010 (10)
8	0b1100 (12)	0b0010 (2)	0b1110 (14)	0b1110 (14)

5.2 Persistent Cross section

In contrast to feed-forward circuits, an SEU within a circuit that contains feed-back and stores internal state will cause persistent errors. The feed-back circuit structures "trap" the incorrect state and store it until appropriate reset measures are taken. These feed-back circuit structures belong to a design's persistent cross section.

In addition to strictly feed-back circuits, paths which feed into a feed-back circuit can also generate persistent functional errors. A functional error generated in one of these paths and subsequently deposited into a feed-back circuit will be trapped and propagated by the feed-back. As a result, circuit components in feed-forward paths upstream from a feed-back circuit also belong to a design's persistent cross section.

Figure 5.5 depicts a generic circuit with both non-persistent and persistent components. The persistent components are highlighted with the dashed line. Intuitively, the feedback section in the top path, consisting of logic and a register, belongs to the persistent cross section. In addition, the first two elements in the top path which feed into the feed-back section also belong to the persistent cross section. The

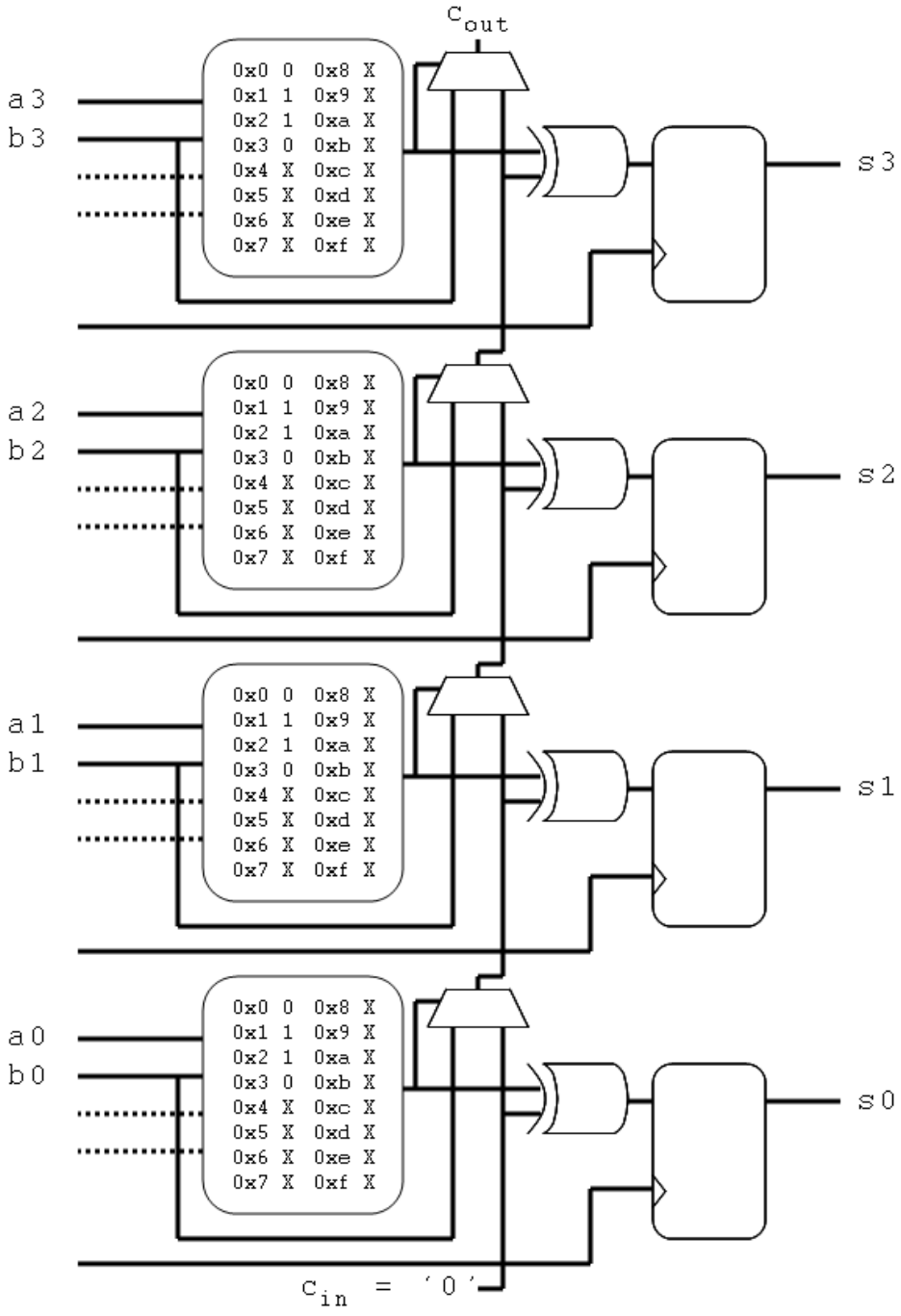


Figure 5.3: Simplified schematic representation of a 4-bit full-adder implemented in an FPGA.

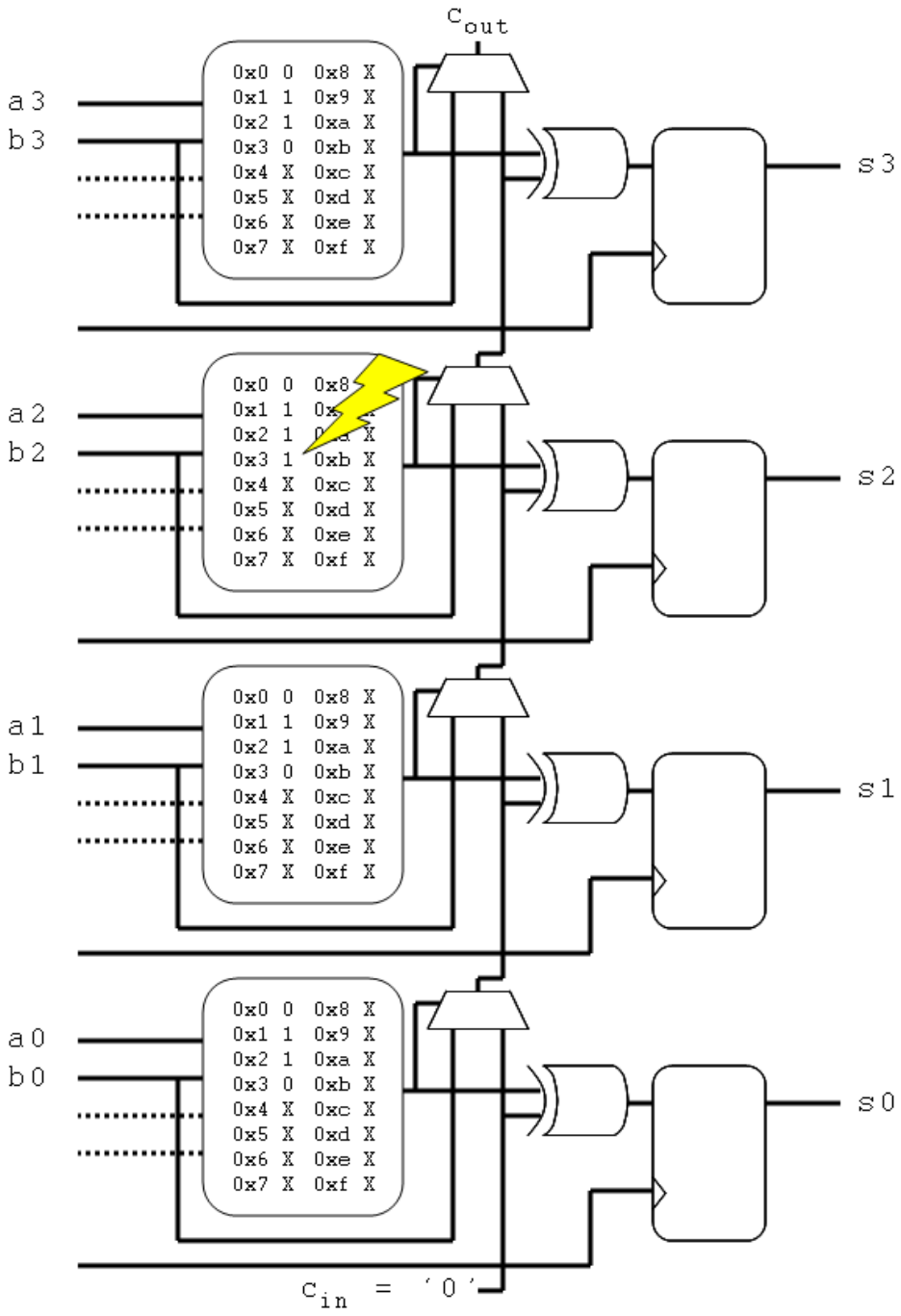


Figure 5.4: Simplified schematic representation of a 4-bit full-adder implemented in an FPGA. The bit stored at address 0x3 in the LUT second from the top was flipped from a 0 to a 1 by an SEU.

feed-back loop would continuously propagate functional errors generated in any stage of the top path of this circuit until reset.

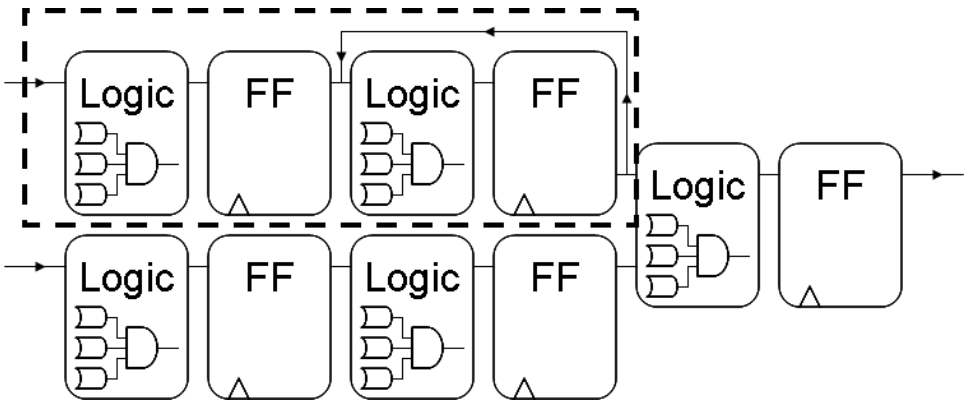


Figure 5.5: The feed-back sections of a circuit belongs to the persistent cross section. Feed-forward sections which feed into a feed-back path also belong to the persistent cross section. Both sections are highlighted with the dashed box.

A specific example best demonstrates what parts of a circuit belong to the persistent cross section. A simple 4-bit accumulate adder primarily has a persistent cross section[1]. Figure 5.6 depicts a simplified schematic of a 4-bit accumulate adder implemented in an FPGA. Note the strong similarity of this circuit to the 4-bit adder circuit depicted in Section 5.1 as Figure 5.3. It is exactly the same circuit with the exception of a bus which feeds the sum s back to the input as one of the operands. This circuit computes the accumulated sum s of a series of operands a. Table 5.2 depicts a sequence of inputs to this circuit. The table also lists the actual and desired accumulated value after each cycle. Operand a is the next value to add to the accumulation. Operand s is the sum from the previous iteration. The output, or accumulated sum, after the current cycle is denoted as s'. For the first two cycles, the actual output and the desired accumulated output matched. During cycle three, however, an SEU flipped the bit stored at address 0x3 in the LUT second from the

[1]The routing on the output of a 4-bit accumulate adder circuit would technically belong to the non-persistent cross section.

top (see Figure 5.7). As a result, the accumulation on cycle three did not match the desired output. This faulty data represents a functional error. On cycle four, the input operands did not exercise the corrupted bit, so they are added together correctly. However, the accumulated value still did not match the desired accumulation since the previous sum was faulty. Even after scrubbing repaired the circuit at the beginning of cycle six the accumulated output continued to not match the desired output. In this case, the invalid data propagated back into the circuit indefinitely. Only a global system reset could ensure the accumulation returned to a known value.

Table 5.2: Sequence of Inputs and Outputs for a 4-bit Adder Circuit

Cycle	input a	input s	actual output s'	desired output s'
1	0b0001 (1)	0b0000 (0)	0b0001 (1)	0b0001 (1)
2	0b0011 (3)	0b0001 (1)	0b0100 (4)	0b0100 (4)
3	0b0110 (6)	0b0100 (4)	**0b0010 (2)**	**0b1010 (10)**
4	0b0001 (1)	0b0010 (2)	**0b0011 (3)**	**0b1011 (11)**
5	0b0010 (2)	0b0011 (3)	**0b0101 (5)**	**0b1101 (13)**
6	0b0000 (0)	0b0101 (5)	**0b0101 (5)**	**0b1101 (13)**
7	0b0001 (1)	0b0101 (5)	**0b0110 (6)**	**0b1110 (14)**
8	0b0001 (1)	0b0110 (6)	**0b0111 (7)**	**0b1111 (15)**

In the preceding 4-bit accumulate adder example, the circuit is primarily a feed-back loop. By definition a feed-back circuit propagates values backward. As a result, functional errors are stored in the sum flip-flops for one cycle and then propagated back as an operand on the next cycle. As such, functional errors never completely exit the circuit. In other words an SEU anywhere within this circuit only causes persistent functional errors. Consequently, almost the entire circuit belongs to the persistent cross section.

5.3 Measuring Persistent Cross Section

Methodologies similar to those used to measure dynamic cross section were used to also measure persistent cross section. Fault-injection was used to estimate

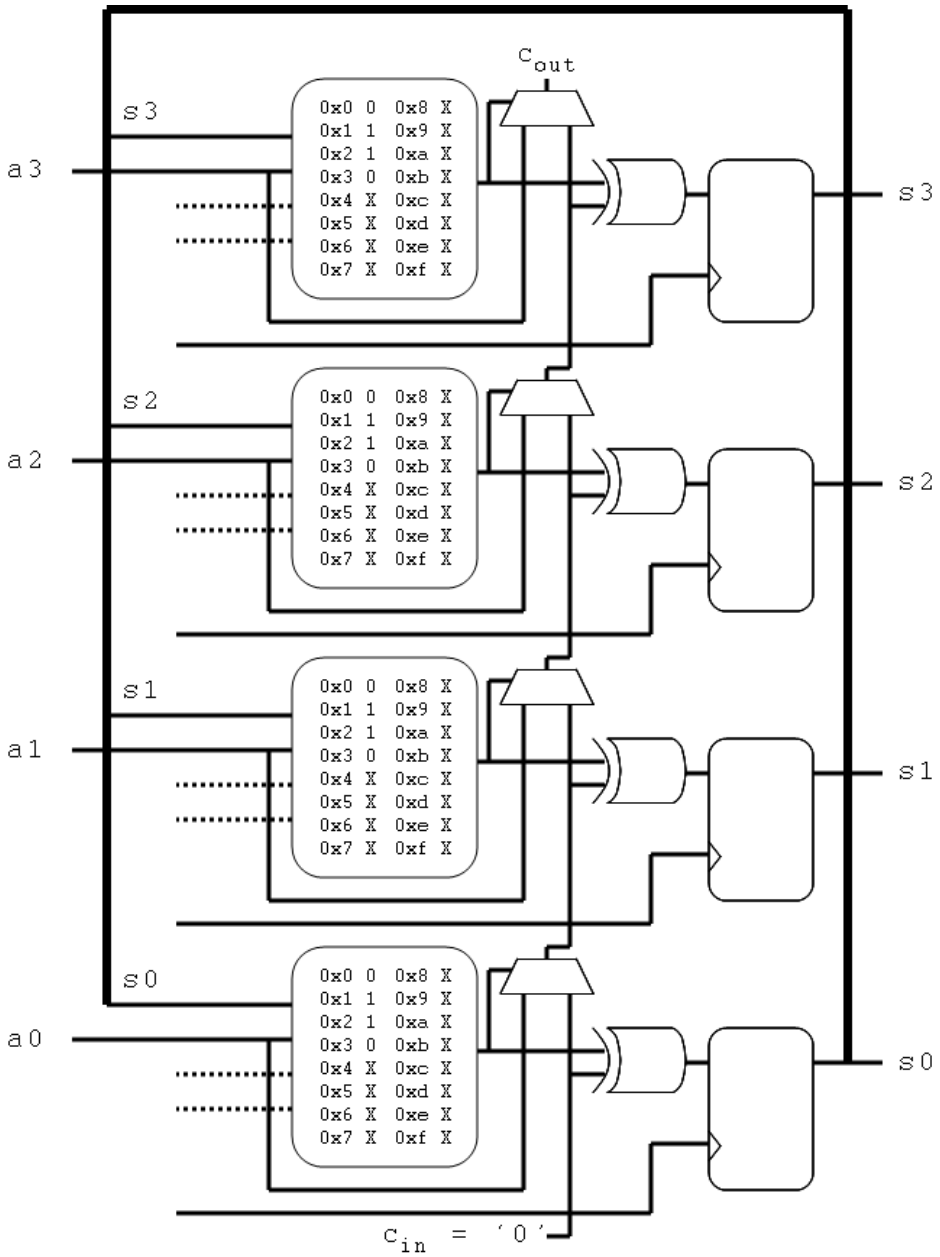


Figure 5.6: Simplified schematic representation of a 4-bit accumulate adder implemented in an FPGA.

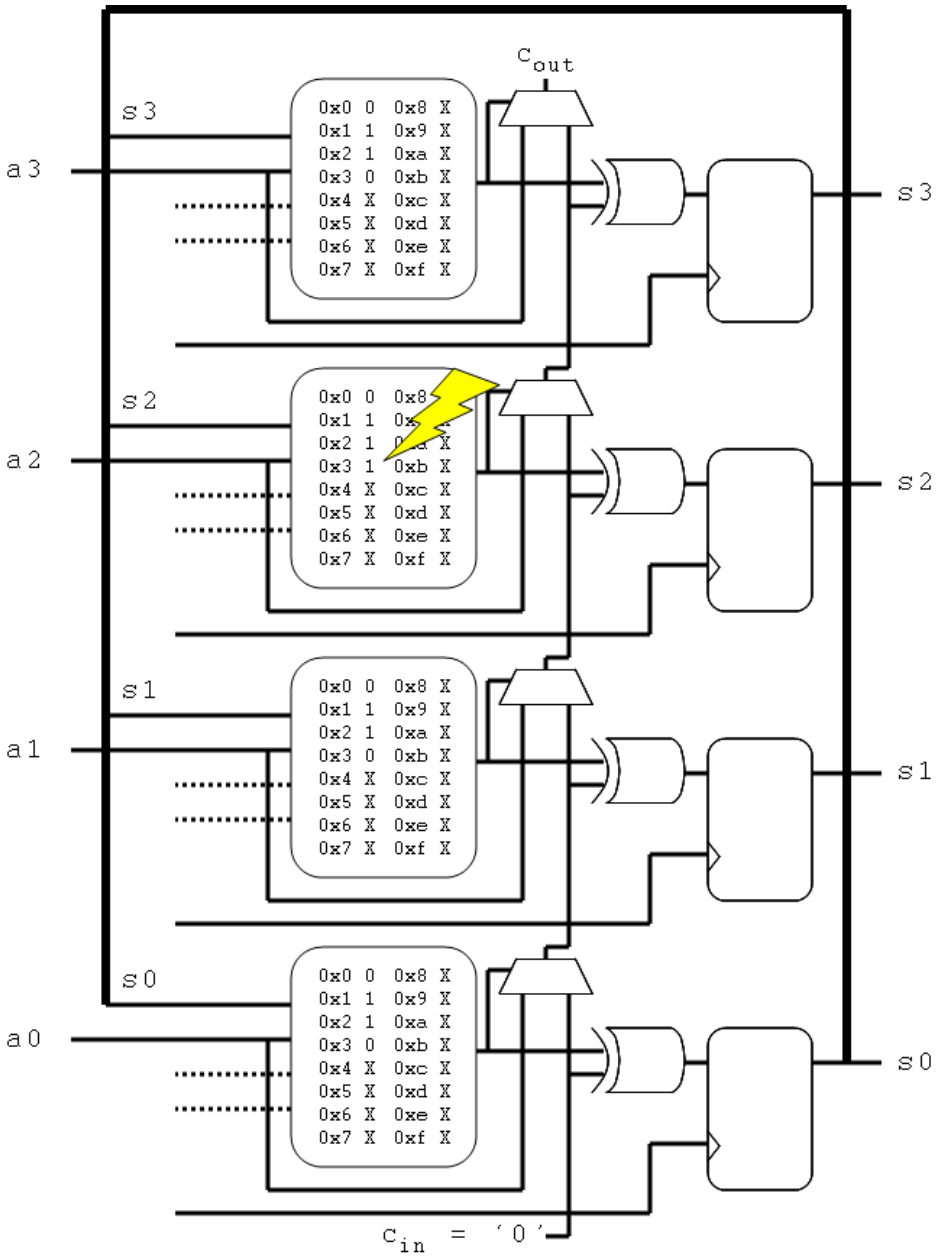


Figure 5.7: Simplified schematic representation of a 4-bit accumulate adder implemented in an FPGA. The bit stored at address 0x3 in the LUT second from the top was flipped from a 0 to a 1 by an SEU.

persistent cross section. To confirm these estimates, persistent cross section was also measured at Crocker Nuclear Laboratory at the University of California, Davis. This section will detail the two testing methodologies and report results from both types of testing. Appendix C also describes these methodologies in more detail.

5.3.1 Persistent Testing Methodologies

To predict the size of a design's persistent cross section, I modified and augmented Johnson's fault injection tool to identify bits which cause persistent functional errors when upset. As such, the tool can now also rapidly create an accurate and complete characterization of the persistent cross section of a given FPGA design.

In general, the fault-injection algorithm is as follows: A bit within the configuration bitstream is toggled from its correct state. The bit is left in this corrupted state for a finite duration (corresponding to the expected scrubbing time t_s). The corrupted bit is then restored to its original state. If errors occurred during this time, the design is allowed to operate for an additional delta time t_f to let errors flush. If at the end of t_f errors still exist, then the originally corrupted bit is classified as contributing to persistent errors (also called a "persistent bit"). Every bit within the FPGA's configuration memory is tested in this manner and the results are recorded. The size of a particular design's predicted persistent cross section, x_{p_p}, is equal to the fraction of "persistent bits" bits in the design multiplied by the device static cross section. Stated mathematically, the predicted persistent cross section is

$$x_{p_p} = x_s \times \frac{\#\text{ persistent bits}}{\#\text{ total bits}}, \tag{5.1}$$

where,

$$x_{p_p} = \text{predicted persistent cross section,}$$

and

$$x_s = \text{static device cross section.}$$

Persistent cross section predictions were verified at Crocker Nuclear Laboratory in Davis, California using proton irradiation and a method very similar to measuring dynamic cross section. A 63 MeV proton source caused the SEUs. Since

the inter-arrival time of the protons cannot be controlled, a different algorithm is used to identify bits which cause persistent functional errors. Two processes operate simultaneously on a host PC. The first process records the time and location of configuration memory upsets as reported by on-board scrubbing circuitry. The second process records the time of all functional errors as reported by auxiliary on-board circuitry. When a functional error occurs, the second process logs the timestamp and then sleeps for a delta time t_f to let functional errors flush. If at the end of this time functional errors still exist, the *error* is marked as persistent. Post processing of data from the two processes matches persistent errors to configuration upsets[2]. The size of the measured persistent cross section equals the number of configuration upsets which cause persistent output errors divided by the product of the total fluence and incident particle angle. Stated mathematically, the measured persistent cross section is

$$x_{p_m} = \frac{\#\text{ persistent error events}}{\text{fluence} \times \cos\theta} \tag{5.2}$$

where,

x_{p_m} = measured persistent cross section,

and

θ = incident particle angle.

5.3.2 Persistent Cross Section Measurements

Both fault-injection and proton irradiation confirm that, as expected, the persistent cross section does exist. Table 5.3 reports the predicted and measured size of the persistent cross section for each of the tested applications. The parameters necessary to compute persistent cross section with Equations 5.1 and 5.2 are also reported. The results show that fault-injection estimates are within a factor of two of actual measured cross section. Table 5.4 repeats the predicted and measured cross section sizes for each design along with the predicted and measured values for the design's corresponding dynamic cross section. The size of each design's non-persistent cross

[2]See Appendix D for more information about post processing of data.

section is simply the difference of the persistent from the dynamic cross section. This table shows that in all cases the persistent cross section is smaller than the dynamic cross section.

Table 5.3: Persistent Cross Section Predictions and Measurements

Design	Predicted		Measured				%
	persistent bits (#)	size (cm^2)	events (#)	fluence ($\frac{\#particles}{cm^2}$)	θ (deg)	size (cm^2)	error††
Multiplier	0	‡	0	7.2×10^{10}	0^o	‡	n/a
Counter	108,750	2.5×10^{-9}	392	2.4×10^{11}	0^o	1.6×10^{-9}	−56.3%
Synthetic	72,991	1.7×10^{-9}	324	2.7×10^{11}	0^o	1.2×10^{-9}	−41.7%
DSP Kernel	8,930	2.0×10^{-10}	35	3.1×10^{11}	0^o	1.1×10^{-10}	−81.8%

The device used in testing was a $0.22\mu m$ 5-layer epitaxial process Xilinx Virtex XCV1000 FPGA with a static cross section of 1.28×10^{-7} cm^2 [1].
‡No events were observed.
††$\%error = (measured - predicted)/measured$

The signature of a design's persistent cross section correlates to the mapping and routing of resources in feed-back loops. This feed-back can be spread throughout a design. To illustrate this result, graphical representations of resource utilization, dynamic cross section and persistent cross section for the Counter, Synthetic and DSP Kernel designs are depicted in Figures 5.8, 5.9 and 5.10 respectively. In each figure, the left and middle graphics are repeated from Chapter 3. The left graphic is a screen capture of the design's utilization from the resource editor tool, `fpga_editor`, provided by the manufacturer and the middle graphic is a `Matlab` rendering of the "sensitive" configuration memory bits or dynamic cross section. The right graphic is a a `Matlab` rendering of the "persistent" configuration memory bits or persistent cross section. Both Matlab plots are based on data generated by the fault injection tool.

Table 5.4: Cross Section Predictions and Measurements

Design	Resource Utilization (slices)	(%)	Dynamic Cross Section (cm^2) Predicted	Measured	Persistent Cross Section (cm^2) Predicted	Measured
Multiplier	10,305	83.9	1.1×10^{-8}	8.3×10^{-9}	‡	‡
Counter	2,151	17.5	4.2×10^{-9}	3.5×10^{-9}	2.5×10^{-9}	1.6×10^{-9}
Synthetic	2,538	20.7	4.2×10^{-9}	3.2^{-9}	1.7×10^{-9}	1.2×10^{-9}
DSP Kernel	5,746	46.8	1.1×10^{-8}	9.5×10^{-9}	2.0×10^{-10}	1.1×10^{-10}

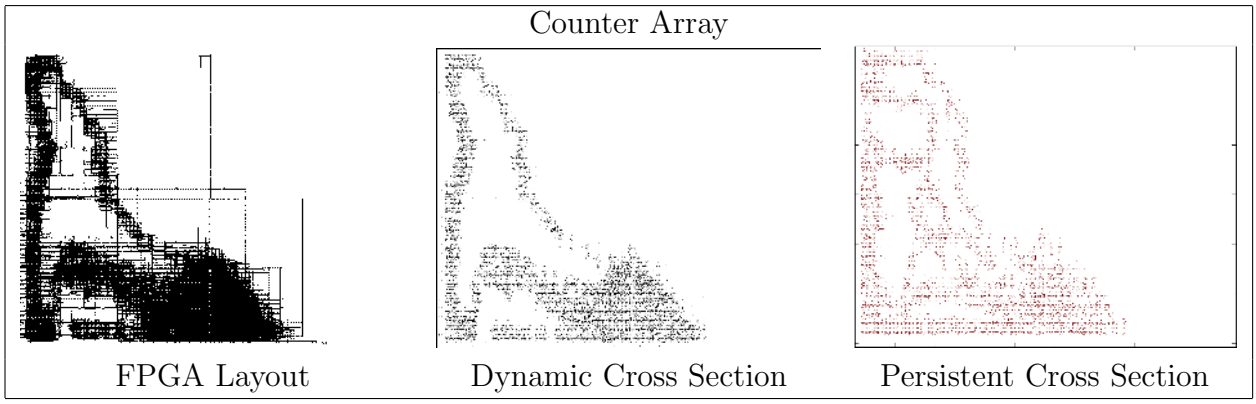


Figure 5.8: Comparison of resource utilization, dynamic cross section and persistent cross section for a Counter Array design. The diagram on the left is a screen capture of the resource layout of the design. The right and left diagrams are respectively the dynamic and persistent cross section of the Counter Array design.

For each of the designs of Table 5.3 estimates of persistence for fault injection exceeded the measured values of persistence by up to a factor two. These results suggest that there is a large disparity between the estimated persistence and the actual measured persistence. While this is true, the purpose of these tests was not to validate the accuracy of the fault injection tool but to demonstrate the existence of the persistence cross section and show that this component of the sensitive cross section is very small. Both fault injection and measured results from the radiation source prove this point. Persistence was seen in both cases and in both cases the percentage of sensitive configuration bits that are persistent is extremely small.

The disparity in results between the fault injection tool and the measured results from the radiation source do suggest that the methodology used to predict

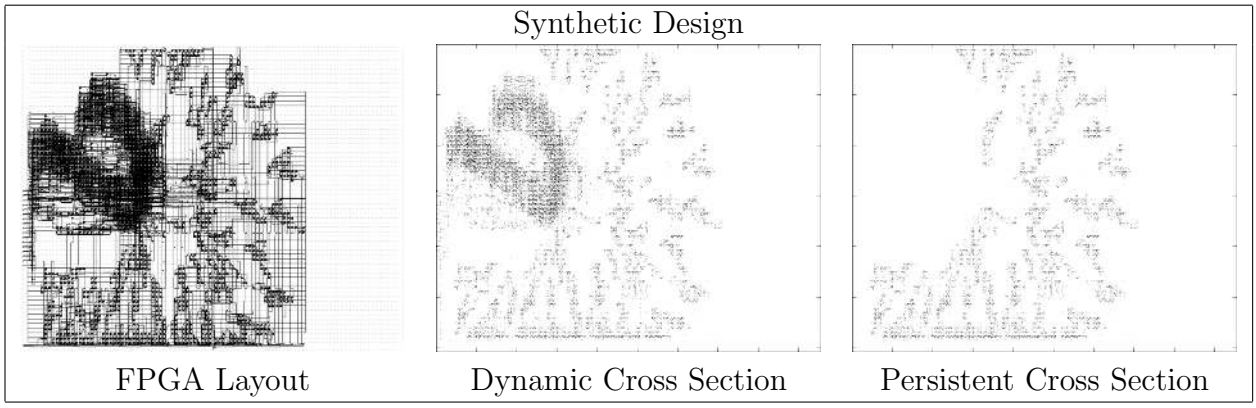


Figure 5.9: Comparison of resource utilization, dynamic cross section and persistent cross section for a Synthetic design.

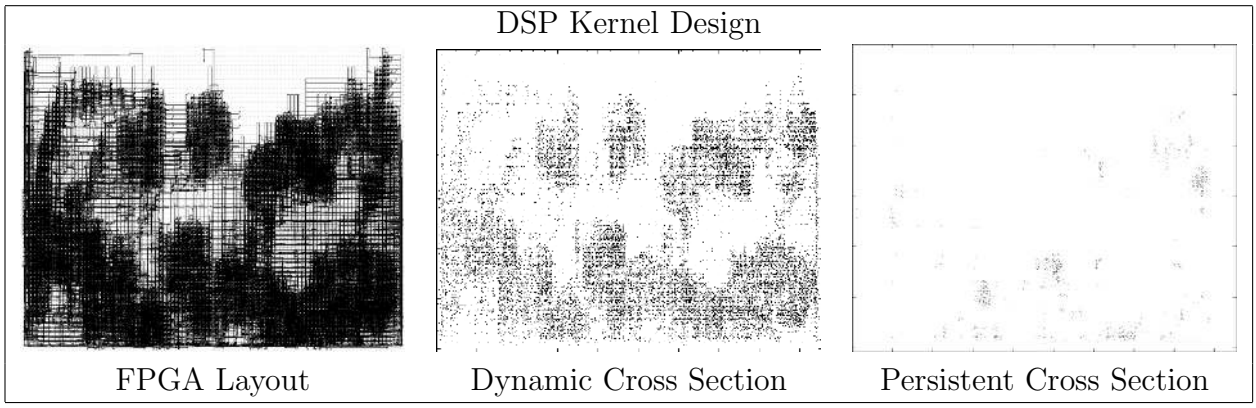


Figure 5.10: Comparison of resource utilization, dynamic cross section and persistent cross section for the DSP Kernel design.

persistence and measure persistence needs to be improved. Several factors make this very difficult to do. First, because of the small persistence cross section, it is very difficult and expensive to obtain sufficient data from radiation measurements. Second, it is very difficult to identify independent persistent upsets from a radiation source because of the random nature of arrival time of high energy particles. In order to limit the effect of independent particles causing a persistent failure, the particle flux must be significantly reduced. Reducing the particle flux will require far more time and expense to perform the test. A more detailed discussion of the disparity in

results between the fault injection tool and radiation measurements can be found in Appendix D.

5.4 MTBF Estimation

Systems can have varying degrees of tolerance with respect to non-persistent and persistent functional errors. This section will define the different tolerance levels and discuss the implications of each when calculating mean time between failure (MTBF). The section will conclude with modified estimates of the MTBF predictions first reported in Section 3.4.2.

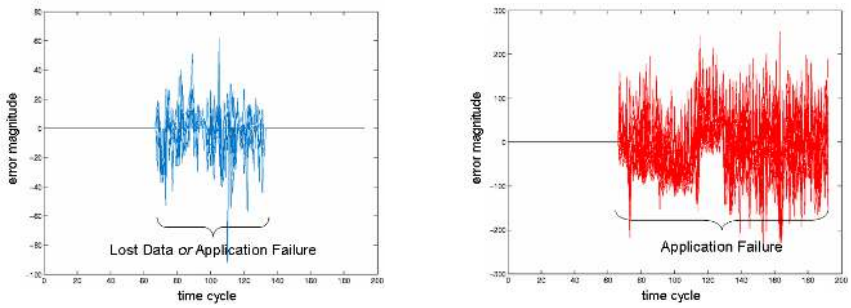
5.4.1 Application Service Interruption Tolerance

As stated in Chapter 4, non-persistent and persistent errors represent temporary and permanent service interruptions respectively. Figure 5.11 depicts the typical error modes of a temporary and permanent service interruption. Most applications cannot tolerate permanent service interruptions, but some applications can tolerate temporary service interruptions. By tolerating temporary interruptions of service, an application automatically fails less often. Furthermore, focusing mitigation on circuit elements susceptible to the more critical, permanent service interruption can significantly increase MTBF. This ability to tolerate non-persistent errors justifies analyzing a design's dynamic cross section for non-persistent and persistent components.

Some applications cannot tolerate any type of service interruption. These "intolerant" applications *cannot* tolerate temporary *nor* permanent service interruptions. Either type of service interruption depicted in Figure 5.11 would be considered an application failure. Because non-persistent and persistent functional errors represent a form of service interruption, it follows that *any* dynamic upset will cause an intolerant system to fail.

Some applications, on the other hand, can tolerate temporary service interruptions. In a "tolerant" application, the left plot in Figure 5.11 simply represents a short loss of data. Only a permanent service interruptions results in application failure. Since a "tolerant" application loses data during a temporary service interruption

and only fails following a permanent service interruption, it follows that only upsets in the persistent cross section cause a tolerant application to fail.



Temporary Service Interruption Permanent Service Interruption

Figure 5.11: The diagram on the left represents a temporary service interruption, or non-persistent errors. The diagram on the right depicts a permanent service interruption, or persistent errors.

The ability to tolerate temporary service interruptions is application specific. Ultimately an application's tolerance level depends on the criticality of a continuous stream of uncorrupted output data. The user must decide an application's level of tolerance because it directly affects the amount and type of functional error mitigation needed.

5.4.2 Calculating MTBF

As mentioned in Chapter 3, MTBF estimates the average time between critical interruptions of service. Since the definition of "critical" changes with an application's tolerance level, calculating MTBF changes with an application's tolerance level as well.

Failure in an intolerant application is defined as any type of service interruption, so it follows that both non-persistent and persistent errors result in "critical" interruptions of service. As a result, Equations 3.3 and 3.4 presented in Chapter 3 can be used to calculate MTBF in intolerant systems. The equations are repeated

here as

$$\lambda_i \;=\; \frac{Dynamic\ Cross\ Section}{Static\ Cross\ Section} \;\times\; Static\ SEU\ Rate, \tag{5.3}$$

and

$$MTBF_i \;=\; \frac{1}{\lambda_i}, \tag{5.4}$$

for convenience. For the same reasons mentioned in Chapter 3, intolerant failure rate, λ_i, has units of event per unit time and intolerant mean time between failure, $MTBF_i$, has units of time per failure.

In contrast to intolerant applications, tolerant applications only fail after a permanent interruption of service. As a result, only persistent errors cause "critical" interruptions of service in a tolerant system. Consequently, tolerant failure rate, λ_t and tolerant mean time between failure, $MTBF_t$, are calculated as

$$\lambda_t \;=\; \frac{Persistent\ Cross\ Section}{Static\ Cross\ Section} \;\times\; Static\ SEU\ Rate, \tag{5.5}$$

and

$$MTBF_t \;=\; \frac{1}{\lambda_t}, \tag{5.6}$$

respectively. In Equation 5.5, the cross-section ratio reflects the persistent portion of the static cross section. Since the persistent component of the dynamic cross section is always smaller than or equal to the size of the entire dynamic cross section, λ_t will always be smaller than λ_i. As a result, the denominator in Equation 5.6 will always be smaller than the denominator in Equation 5.4. Thus, MTBF will always be larger for tolerant applications.

5.4.3 Orbit-Specific MTBF Estimates

Table 5.5 is a modified version of Table 3.2, but with two MTBF values for each combination of design, orbit and solar conditions. The two values represent MTBF for the particular design if it cannot tolerate or can tolerate persistent functional errors.

Table 5.5: Modified On-Orbit Mean Time Between Failure Estimates

Design	Tolerance Level	Solar Conditions	Mean Time Between Failure (hr/failure)		
			LEO †	Polar ††	GPS †††
Multiplier	Intolerant	Solar Min	4.1×10^2	1.6×10^2	2.5×10^2
		Solar Max	6.1×10^2	2.1×10^2	2.4×10^2
		Worst Day	5.6×10^2	3.0	8.5×10^{-1}
	Tolerant	Solar Min	*	*	*
		Solar Max	*	*	*
		Worst Day	*	*	*
Counter	Intolerant	Solar Min	1.1×10^3	4.3×10^2	6.7×10^2
		Solar Max	1.6×10^3	5.5×10^2	6.5×10^2
		Worst Day	1.5×10^3	8.0	2.3
	Tolerant	Solar Min	2.2×10^3	8.9×10^2	1.4×10^3
		Solar Max	3.4×10^3	1.1×10^3	1.3×10^3
		Worst Day	3.1×10^3	1.6×10^1	4.7×10^0
Synthetic	Intolerant	Solar Min	1.2×10^3	4.7×10^2	7.2×10^2
		Solar Max	1.8×10^3	5.9×10^2	7.0×10^2
		Worst Day	1.6×10^3	8.6	2.4
	Tolerant	Solar Min	2.9×10^3	1.1×10^3	1.8×10^3
		Solar Max	4.3×10^3	1.5×10^3	1.7×10^3
		Worst Day	3.9×10^3	2.1×10^1	6.0
DSP Kernel	Intolerant	Solar Min	4.2×10^2	1.7×10^2	2.6×10^2
		Solar Max	6.3×10^2	2.1×10^2	2.5×10^2
		Worst Day	5.7×10^2	3.1	8.7×10^{-1}
	Tolerant	Solar Min	2.4×10^4	9.4×10^3	1.5×10^4
		Solar Max	3.5×10^4	1.2×10^4	1.4×10^4
		Worst Day	3.2×10^4	1.7×10^2	4.9×10^1

† Low-Earth Orbit at 560 *km altitude*, 35.0^o *inclination*
†† Polar Orbit at 833 *km altitude*, 98.7^o *inclination*
††† Global Positioning System Orbit at $22,200$ *km altitude*, 55.0^o *inclination*
* The design has no observable persistent cross section, therefore making it impossible to calculate tolerant MTBF for this design.

The results from Table 5.5 indicate that some applications' failure rate may substantially improve when they can tolerate non-persistent functional errors. For example, when treated as a tolerant application, the DSP Kernel fails $\approx 55\times$ less often[3].

[3]Note that the ratio between tolerant and intolerant MTBF for a particular design is constant across all orbits and all solar conditions. This result follows from taking the ratio of tolerant to intolerant MTBF, or Equation 5.6 to Equation 5.4 with Equations 5.5 and 5.3 substituted in respectively. The resulting ratio is $\frac{Persistent\ Cross\ Section}{Dynamic\ Cross\ Section}$. Thus, the ratio of tolerant to intolerant

The Multiplier application does not have a measurable persistent cross section (see Table 5.3). As such, the Multiplier should theoretically never experience permanent service interruptions, or in other words, never fail as a "tolerant" application. Applications like the Multiplier with a persistent cross section substantially smaller than their dynamic cross section substantially improve their failure rate by tolerating non-persistent functional errors. For other applications, the tolerance distinction makes less difference. For example, the Counter application only fails $\approx 2\times$ less often when treated as "tolerant". Since the persistent cross section of the Counter design roughly equals the size of it's dynamic cross section, the distinction between "tolerant" and "intolerant" is less important.

5.5 Summary

In this chapter, a qualitative method for determining circuit elements int the non-persistent and the persistent cross sections was defined. Measurements of the non-persistent and persistent cross sections for several designs were also presented. The data indicates that the non-persistent and persistent cross sections exist and can be measured for any arbitrary circuit. Designs that can tolerate non-persistent functional errors can substantially improve their MTBF without any mitigation effort. The next chapter will show that by leveraging the qualitative mapping from circuit elements to cross section, MTBF can further be improved at lower resource costs by focusing mitigation on just a design's persistent cross section.

MTBF only depends on the size of a design's persistent and dynamic cross sections, not the destined orbit or solar conditions.

Chapter 6

Functional Error Mitigation

The previous chapter showed that an application's failure rate improves substantially when it can tolerate non-persistent functional errors. Even so, tolerant applications still fail after persistent functional errors. To further improve a tolerant application's failure rate, persistent functional errors must be mitigated. Many approaches exist to mitigate functional errors, but typically an application's complete set of design constraints dictates the approach.

The most common method of FPGA reliability enhancement, Triple Modular Redundancy (TMR), is most effective in a reliability-limited only system. In a resource-constrained system, TMR may not be feasible. The system may simply not have enough available resources for full triple redundancy. Furthermore, in some cases the system cannot afford the power consumed by using 3× redundant resources in the FPGA.

The existence of non-persistent and persistent cross sections adds a new dimension to the space of possible approaches for systems constrained by more than just reliability. More specifically, a resource-limited FPGA application with relaxed availability requirements can focus mitigation on just the persistent cross section to efficiently improve reliability at minimum resource costs.

This chapter begins with a discussion of reliability, availability and cost constraints. Next, the benefits and drawbacks associated with full mitigation are discussed in terms of reliability, availability and cost. Partial mitigation, or elimination of just a subset of an application's functional errors, is then introduced. Next, a novel partial mitigation strategy is introduced which focuses on just the persistent

functional errors. The chapter concludes with measurements of cross section and modified mean time between failure (MTBF) estimates for two applications with just the persistent cross section mitigated.

6.1 Design Constraints

Designing a reliable system often involves many constraints beyond reliability including availability and cost (area, resources, power etc.). Reliability is a measure of an application's ability to withstand failure. Availability refers to the percentage of time an application is available and/or performing its intended function. Cost can have many facets. In the context of this work, resource usage and/or power are the limiting costs. Other constraints are also sometimes important, but will not be discussed here.

In some instances a system is limited by only one design constraint. A reliability-limited only system requires a specific level of reliability no matter the cost. An availability-limited only system requires maximum operation time no matter the cost. A resource-limited only system has a hard number of available resources. In many instances a system is limited by a combination of constraints. This work will focus on systems which are reliability and availability limited, but have a relaxed availability constraint.

6.2 Full Mitigation

Full mitigation can typically only satisfy the constraints of a reliability-limited only system. If the system has a resource constraint, full mitigation may not be possible. In other words, sometimes not enough extra resources are available for the required redundancy. Also, if the system has a power constraint, full mitigation may not be possible.

In FPGAs, the most popular method of full functional error mitigation is triple-modular redundancy (TMR). TMR is a simple form of single error detection and correction (SEDAC). All circuit resources are triplicated and a majority vote determines the circuit's outputs.

Lima showed that TMR can achieve 100% reliability in the face of single bit upsets (SBU) at a certain flux [22]. However research by Rollins *et al.* showed that the cost in terms of resources and power is at least 3× and can be as high as 5× [23, 24].

6.3 Partial Mitigation

Some systems do not require maximum reliability and availability. Brendon Bridgford *et al.* proposed that not all applications require full TMR nor scrubbing. They hypothesized that a system can achieve acceptable reliability with any number of combinations of TMR and scrubbing [25]. Gary Swift *et al.* proposed partial redundancy of the input/output blocks on an FPGA [26]. Vikram Chanrasekhar *et al.* introduced a method of partial redundancy based on probability analysis [27]. This work will introduce a novel partial redundancy mitigation method based on a static analysis of a circuit's critical components.

In some systems a certain level of reliability is required, but the system has a tight cost constraint. If the cost of full mitigation is above this constraint then full mitigation is not feasible. In these cases partial functional error mitigation can sometimes meet both the reliability and cost constraints. Partial mitigation schemes incrementally add mitigation resources until a certain level of reliability is achieved. These methods can achieve a minimum reliability level and hopefully stay within a specified maximum cost. All functional errors are not mitigated, but the mitigation cost is still lower than full mitigation.

Even though partial mitigation saves cost, the evidence of the persistent and non-persistent cross sections reveals a more efficient partial approach. Traditionally partial mitigation strategies arbitrarily mitigate both persistent and non-persistent functional errors, but the previous chapter showed that some applications can tolerate non-persistent errors. For these tolerant applications, arbitrary partial mitigation wastes valuable resources mitigating non-persistent functional errors which do not cause failure. For this reason, a better partial mitigation strategy for tolerant applications focuses mitigation on just the persistent cross section. Such an approach allocates 100% of mitigation resources to eliminating failure. As a result, this method

realizes desired levels of reliability at a much lower cost than either arbitrary partial mitigation or full mitigation.

Persistent-functional-error-focused partial mitigation is only possible if components can be identified, at the circuit level, which contribute to an application's persistence. Section 5.2 showed that a correlation exists from persistent cross section to its constituent circuit elements. This mapping allows circuits to be statically analyzed to identify components in the persistent cross section. These identified components can then be targeted for focused partial mitigation.

6.4 Modified Persistent Cross Section Measurements

Algorithms which can identify structures in a circuit's persistent cross section have been developed at Brigham Young University [28]. I assisted in the development of an edif-based TMR tool that included these algorithms which apply TMR to just circuit elements that contribute to an application's persistent cross section. This tool was used to mitigate the persistent cross section in a pair of the applications introduced in Chapter 1. Fault-injection and particle irradiation were used to respectively measure each application's persistent cross section.

Table 6.1: Modified Persistent Cross Section Predictions and Measurements

Design	Mitigation Level	Resource Utilization (slices)	(%)	Persistent Cross Section (cm^2) Predicted	Measured
Synthetic	None	2,538	20.7	1.7×10^{-9}	1.2×10^{-9}
	Partial	9,867	80.3	1.8×10^{-11}	†
	Max/Full	11,961	97.3	1.5×10^{-10}	†
DSP Kernel	None	5,746	46.8	2.0×10^{-10}	2.8×10^{-10}
	Partial	8,036	65.4	2.4×10^{-12}	3.4×10^{-12}
	Max	11,114	90.4	3.5×10^{-12}	
	Full‡	17,238	140.4	‡	‡

† Design not tested at accelerator.
‡ Theoretical design which would require more resources than available on the XCV1000.

Table 6.1 lists the unmitigated and partially mitigated persistent cross section sizes. As expected, partial mitigation drastically reduced the size of the persistent cross section in both designs. In the partially mitigated DSP kernel, its persistent cross section sized dropped by nearly two orders of magnitude, but only increased resource utilization by 1.4×. (Theoretically the full TMR version would increase utilization by 3×, but as the next design shows, a practical implementation is much larger.) Figure 6.1 graphically illustrates this reduction in cross section. Persistence in the partially mitigated synthetic application also dropped nearly two orders of magnitude. However in this case, resource utilization increased by 3.8×. Note that the resource utilization increase for a full TMR version of this design was 4.7× Also note that the persistent cross section was not completely eliminated in either design. The remaining persistent cross section is likely from the input and output ports, and reset and clock distribution trees, which all belong to the persistent cross section, but could not be triplicated due to resource constraints on the FPGA.

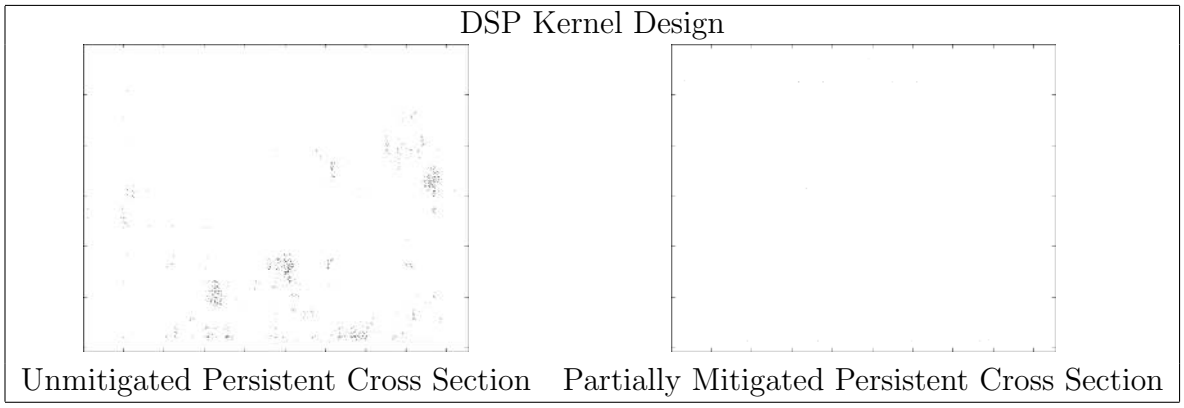


Figure 6.1: Comparison of the DSP Kernel design's persistent cross section before and after partial mitigation with TMR.

6.5 Modified MTBF Estimates

Reducing persistent cross section reduces failure rate and MTBF for tolerant applications. The cross section estimates reported in the previous section were used

to update the MTBF estimates in Table 5.5. The results are reported in Table 6.2. Again, MTBF values are reported in each of three distinct orbits during three different levels of solar activity. The orbits are the same as those described in Chapter 1. The values shown only represent MTBF for the specified application in a "tolerant" scenario. Values are reported for each design with two different mitigation levels, none and partial. The results corresponding to no mitigation were transferred directly from Table 5.5. The results corresponding to partial mitigation were calculated using Equations 5.5 and 5.6 and the cross section values in Table 6.1.

Table 6.2: Modified On-Orbit Mean Time Between Failure Estimates for Tolerant Applications

Design	Solar Conditions	Mitigation Level	Mean Time Between Failure (hr/failure)		
			LEO †	Polar ††	GPS †††
Synthetic	Solar Min	None	2.9×10^3	1.1×10^3	1.8×10^3
		Partial	2.7×10^5	1.1×10^5	1.7×10^5
	Solar Max	None	4.3×10^3	1.5×10^3	1.7×10^3
		Partial	4.1×10^5	1.4×10^5	1.6×10^5
	Worst Day	None	3.9×10^3	2.1×10^1	6.0
		Partial	3.7×10^5	2.0×10^3	5.7×10^2
DSP Kernel	Solar Min	None	2.4×10^4	9.4×10^3	1.5×10^4
		Partial	1.9×10^6	7.5×10^5	1.2×10^6
	Solar Max	None	3.5×10^4	1.2×10^4	1.4×10^4
		Partial	2.8×10^6	9.6×10^5	1.1×10^6
	Worst Day	Worst Day	3.2×10^4	1.7×10^2	4.9×10^1
		Partial	2.6×10^6	1.4×10^4	3.9×10^3

† Low-Earth Orbit at 560 *km altitude*, 35.0^o *inclination*
†† Polar Orbit at 833 *km altitude*, 98.7^o *inclination*
††† Global Positioning System Orbit at $22,200$ *km altitude*, 55.0^o *inclination*

The results in Table 6.2 indicate that, for tolerant applications, MTBF increases in direct proportion to a decrease in persistent cross section. For example, with partial mitigation focused on the persistent cross section, the Synthetic appli-

cation's MTBF increased nearly two orders of magnitude in all orbits. This increase corresponds directly to the nearly two orders of magnitude reduction in the Synthetic design's persistent cross section, after partial mitigation, from $1.7 \times 10^{-9}\ cm^2$ to $1.8 \times 10^{-11}\ cm^2$, reported in Table 6.1.

Figure 6.2 illustrates the incremental improvements in MTBF for the DSP Kernel and Synthetic applications in a low-earth orbit during solar max. The applications are assumed to be non-persistent-error-tolerant. For both applications the far left point represents the "static MTBF" if all SEUs caused failure. The next point to the right indicates the unmitigated MTBF. The next point indicates MTBF after persistent functional error mitigation. The final point indicates MTBF after maximum functional error mitigation (in both cases full mitigation was not feasible due to the size of the FPGA). The slope of each line segment indicates the incremental reliability gained from one additional resource. For both designs the slope of the line between the unmitigated and partially mitigated data points is far steeper than the slope of the last line segment to maximum mitigation. As expected, mitigating persistent functional errors in a non-persistent-error-tolerant application is significantly more cost-effective than mitigating non-persistent functional errors.

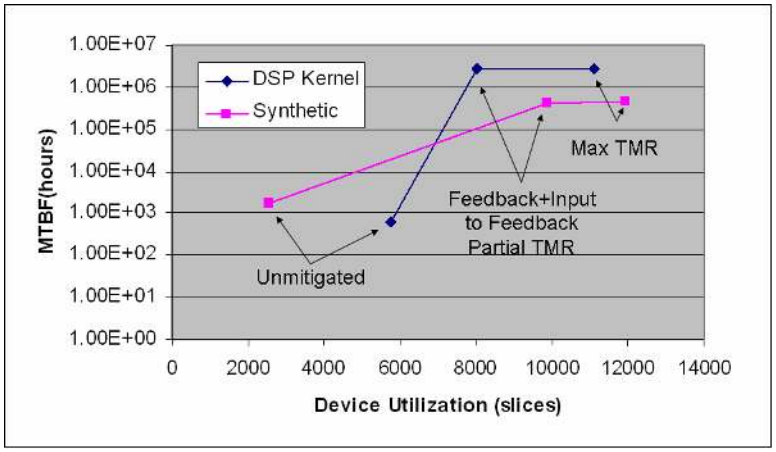


Figure 6.2: Plot of device utilization vs. MTBF for the DSP Kernel and Synthetic designs in a Low-Earth Orbit at Solar Max. Both applications are assumed to be non-persistent-error-tolerant.

6.6 Summary

Often a system has a set of design constraints which must be considered when designing a reliable system. Three important constraints are reliability, availability and cost. Some systems may have a reliability constraint which does not require full mitigation. Others may have a cost constraint that does not allow full mitigation. Yet other systems may have both a reliability and cost constraint which must be balanced. This chapter introduced a new partial mitigation method to meet these hybrid constraints at lower costs. The proposed strategy focuses on persistent functional errors. The results indicate that persistent-error-focused partial mitigation can improve MTBF to acceptable levels for non-persistent-error-tolerant applications at significantly lower costs.

Chapter 7

Summary and Conclusion

As the previous chapter showed, an FPGA application which can give up availability can improve reliability to acceptable levels with far fewer resource costs. Specifically, Figure 6.2 shows that mitigation of just the persistent functional errors in two benchmark applications improved reliability to levels nearly equivalent to the reliability levels attained with maximum mitigation. In one design 21% fewer resources were used than full mitigation (see Table 6.1. Most of this design consists of feedback components and thus persistent cross section. In this case, partial TMR was less beneficial. In a second design 38% fewer resources were used than maximum mitigation. Far less of this design consists of feedback components. Furthermore, maximum mitigation was not full TMR mitigation. If it could have been fully triplicated, partial mitigation would theoretically use 115% fewer resources (the potential savings would likely be larger since full TMR rarely costs just 3×). Thus in some cases a mitigation scheme that focuses on just persistent functional errors can indeed give up availability for a reduction in mitigation cost with nearly no loss in reliability.

The findings of this work will be of immediate benefit to any space-based FPGA projects that have insufficient resources available for full TMR. This is immediately useful at Los Alamos National Laboratory (LANL). The mitigation method introduced in this thesis was implemented on several FPGA applications to be used in a LANL satellite payload scheduled for launch in late 2006. One of the applications, the DSP kernel, was presented and discussed throughout this thesis. This application gained nearly two orders of magnitude reliability at a theoretical savings of 53% over

full mitigation. Other space-destined platforms with FPGAs could similarly benefit from this work.

Other benefits of this work include finer resolution of mitigation. Designers can select the appropriate amount of mitigation so as to not over-engineer their specific problem. Less mitigation also provides less power consumption. This benefit is particularly attractive to space-based applications which are traditionally power-limited. Also, less mitigation means that smaller FPGAs can be used or more functionality can be packed into the same FPGA.

Future work is still needed to provide more options for functional error mitigation in FPGAs. Even in the context of this work more can be done to optimize mitigation. The persistent cross section could be more finely characterized. Probability analysis could be used to statically assign mitigation priority levels in both the persistent and non-persistent cross sections. Work has already been done by Brian Pratt *et al.* to develop algorithms that incrementally mitigate the persistent cross section and then the non-persistent cross section[28]. More work could be done to optimize these algorithms. Other non-TMR approaches could also be developed that are optimized specifically to mitigate either the non-persistent or persistent cross section.

Future research should identify more cost-effective methods for FPGA reliability enhancement. Specifically, risk analysis should be incorporated into these reliability improvement techniques. Application and domain-aware methods are the key to future improvements in the realm of FPGA application reliability.

Bibliography

[1] "Qpro Virtex 2.5v radiation hardened FPGAs," Xilinx Corporation, Tech. Rep., November 5, 2001, dS028 (v1.2).

[2] [Online]. Available: http://www.eas.asu.edu/~holbert/eee460/spacerad.html

[3] R. Baumann, "Single-Event Effects in Advanced CMOS Technology," in *2005 IEEE NSREC Short Course*, Seattle, WA, July 2005, pp. II–1 – II–59.

[4] D. E. Johnson, "Estimating the dynamic sensitive cross section of an FPGA design through fault injection," Master's thesis, Brigham Young University, August 2005.

[5] E. G. Stassinopoulos, "Microelectronics for the natural radiation environments of space, chapter I: Radiation environments of space." in *1990 IEEE NSREC Short Course*, Reno, NV, July 1990.

[6] "FPGAs in space: Programmable logic in orbit," August 2004. [Online]. Available: http://www.fpgajournal.com/articles/20040803_space.htm

[7] L. Massengill, "SEU modeling and prediction techniques," in *1993 IEEE NSREC Short Course*, Snowbird, UT, July 1993, pp. III–1 – III–93.

[8] E. L. Petersen, J. C. Pickel, J. H. Adams, and E. C. Smith, "Rate prediction for single event effects-a critique." *IEEE Transactions on Nuclear Science*, vol. 39, no. 6, pp. 1577 – 1599, DEC 1992.

[9] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, and J. Fabula, "Radiation testing update, SEU mitigation, and availability analysis of the Virtex FPGA

for space reconfigurable computing," in *3rd Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, 2000, p. P30.

[10] E. Petersen, "Cross section measurements and upset rate calculations." *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2805 – 2813, DEC 1996.

[11] X. Corporation, "Radiation effects and mitigation overview." [Online]. Available: http://www.xilinx.com/esp/mil_aero/collateral/presentations/radiation_effects.pdf

[12] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2147–2157, December 2003.

[13] M. Wirthlin, E. Johnson, P. Graham, and M. Caffrey, "Validation of a fault simulator for field programmable gate arrays," in *Proceedings of the IEEE 2003 Nuclear and Space Radiation Effects Conference*, IEEE. Monterey, CA: IEEE, July 2003, p. TBA, accepted.

[14] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham, "The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets," in *Proceedings of the 2003 IEEE Symposium on Field-Programmable Custom Computing Machines*, K. Pocek and J. Arnold, Eds., IEEE Computer Society. Napa, CA: IEEE Computer Society Press, April 2003, p. TBA.

[15] E. Johnson, M. J. Wirthlin, and M. Caffrey, "Single-event upset simulation on an FPGA," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, T. P. Plaks and P. M. Athanas, Eds. CSREA Press, June 2002, pp. 68–73.

[16] L. Connell, P. McDaniel, A. Prinja, and F. Sexton, "Modeling the heavy ion upset cross section," *IEEE Transactions on Nuclear Science*, vol. 42, no. 2, pp. 73–82, April 1995.

[17] "Test procedures for the measurement of single-event effects in semiconductor devices from heavy ion irradiation," Electronic Industries Association, Arlington, VA, Tech. Rep., 1996.

[18] R. Koga, W. A. Kolasinksi, M. Marra, and W. Hanna, "Techniques of microprocessor testing and seu-rate prediction," *IEEE Transactions on Nuclear Science*, vol. NS-32, no. 6, p. 1985p, DEC 1985.

[19] C. Carmichael, E. Fuller, J. Fabula, and F. D. Lima, "Proton testing of SEU mitigation methods for the Virtex FPGA," in *Proceedings of the IEEE Microelectronics Reliability and Qualification Workshop*, Pasadena, CA, December 2001.

[20] E. Fuller, M. Caffrey, P. Blain, C. Carmichael, N. Khalsa, and A. Salazar, "Radiation test results of the Virtex FPGA and ZBT SRAM for space based reconfigurable computing," in *MAPLD Proceedings*, September 1999.

[21] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," Xilinx Corporation, Tech. Rep., June 1, 2000, xAPP216 (v1.0).

[22] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of Virtex FPGA TMR design methodology," in *Proceedings of the 6th European Conference on Radiation and its Effects on Components and Systems (RADECS 2001)*, 2001.

[23] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," in *Proceedings fo the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD).* Washington, D.C.: NASA Office of Logic Design, AIAA, September 2003, p. P63.

[24] N. Rollins, M. Wirthlin, and P. Graham, "Evaluation of power costs in triplicated FPGA designs," in *Proceedings of the MAPLD Conference*, September 2004.

[25] B. Bridgford and C. Carmichael, "SEU mitigation in re-configurable FPGAs: Picking the right tool for the job," in *Proceedings of the MAPLD Conference*, September 2005.

[26] G. M. Swift, S. Rezgui, J. George, C. Carmichael, M. Napier, J. Maksymowicz, J. Moore, A. Lesea, R. Koga, and T. F. Wrobel, "Dynamic testing of xilinx Virtex-II field programmable gate array (FPGA) input/output blocks (IOBs)," *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3469–3474, December 2004.

[27] V. Chandrasekhar, S. N. Mahammad, V. Muralidaran, and V. Kamakoti, "Reduced triple modular redundancy for tolerating SEUs in SRAM-based FPGAs," in *Proceedings of the MAPLD Conference*, September 2005.

[28] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," in *Proceedings of the MAPLD Conference*, September 2005.

[29] R. Katz, K. LaBel, J. Wang, B. Cronquist, R. Koga, S. Penzin, and G. Swift, "Radiation effects on current field programmable technologies," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 1945–1956, December 1997.

[30] R. Katz, J. J. Wang, R. Koga, K. A. LaBel, J. McCollum, R. Brown, R. A. Reed, B. Cronquist, S. Crain, T. Scott, W. Paolini, and B. Sin, "Current radiation issues for programmable elements and devices," *IEEE Transactions on Nuclear Science*, vol. 45, no. 6, pp. 2600–2610, December 1998.

[31] R. B. Katz and J. J. Wang, "Using IEEE 1149.1 JTAG circuitry in Actel SX devices," NASA/Actel, Tech. Rep., August 1998, available at www.actel.com.

[32] S. M. Guertin, G. M. Swift, and D. Nguyen, "Single-event upset test results for the Xilinx XQ1701L PROM," in *IEEE Radiation Effects Data Workshop*, 1999, pp. 35–40.

[33] J. Wang, R. Katz, J. Sun, B. Cronquist, J. McCollum, T. Speers, and W. Plants, "SRAM based re-programmable FPGA for space applications," *IEEE Transactions on Nuclear Science*, vol. 46, no. 6, pp. 1728–1735, December 1999.

[34] J. Fabula and H. Bogrow, "Total ionizing dose performance of SRAM-based FPGAs and supporting PROMs," in *3rd Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, 2000, p. C2.

[35] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," Xilinx Corporation, Tech. Rep., November 1, 2001, xAPP197 (v1.0).

[36] C. Carmichael, E. Fuller, J. Fabula, and F. D. Lima, "Proton testing of SEU mitigation methods for the Virtex FPGA," in *4th Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, 2001, p. P6.

[37] P. Brinkley and C. Carmichael, "SEU mitigation design techniques for the XQR4000XL," Xilinx Corporation, Tech. Rep., March 15, 2000, xAPP181 (v1.0).

[38] *SLAAC-1V User VHDL Guide*, USC-ISI East, October 1, 2000, release 0.3.1.

[39] *Virtex Series Configuration Architecture User Guide, Xilinx Application Notes 151, v1.5*, Xilinx, Inc., September 2000.

[40] E. A. Bezerra, F. Vargas, and M. P. Gough, "Improving reconfigurable systems reliability by combining periodical test and redundancy techniques: A case study," *Journal of Electronic Testing: Theory And Applications - JETTA*, vol. 17, no. 3, pp. 701–711, 2001.

[41] M. Caffrey, "A space-based reconfigurable radio," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, T. P. Plaks and P. M. Athanas, Eds. CSREA Press, June 2002, pp. 49–53.

[42] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Reliability of programmable Input/Output pins in the presence of configuration upsets," in *Pro-*

ceedings of the 5th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD), September 2002.

[43] M. Caffrey, P. Graham, M. Wirthlin, E. Johnson, and N. Rollins, "Single-event upsets in SRAM FPGA," in *Proceedings of the 5th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2002.

[44] P. Sundararajan, S. McMillan, and S. Guccione, "Testing FPGA devices using JBits," in *Proceedings of the 4th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2001, p. E5.

[45] P. Graham, M. Caffrey, M. Wirthlin, D. E. Johnson, and N. Rollins, "Reconfigurable computing in space: From current technology to reconfigurable systems-on-a-chip," in *Proceedings of the 2003 IEEE Aerospace Conference*. Big Sky, MT: IEEE, March 2003, pp. T07_0603.1–12.

[46] ——, "SEU mitigation for half-latches in Xilinx Virtex FPGAs," in *Proceedings of the IEEE 2003 Nuclear and Space Radiation Effects Conference*, IEEE. Monterey, CA: IEEE, July 2003, p. TBA, accepted.

[47] ——, "SEU mitigation for half-latches in Xilinx Virtex FPGAs," vol. 50, no. 6, pp. 2139–2146, December 2003.

[48] N. Cohen, T. Sriram, N. Leland, D. Moyer, S. Butler, and R. Flatley, "Soft error considerations for deep-submicron CMOS circuit applications," in *Technical Digest - International Electron Devices Meeting*, IEEE Electron Devices Society. Washington, D.C.: IEEE Electron Devices Society Publisher, December 1999, pp. 315–318.

[49] M. Caffrey, M. Echave, C. Fite, T. Nelson, A. Salazar, and S. Storms, "A space-based reconfigurable radio," in *Proceedings of the 5th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2002, p. A2.

[50] M. Ceshia, M. Bellato, A. Paccagnella, S. C. Lee, C. Wan, A. Kaminski, M. Menichelli, A. Papi, and J. Wyss, "Ion beam testing of Altera Apex FPGAs," in *Proceedings of the 2002 IEEE Radiation Effects Data Workshop*, S. Crain and T. Turflinger, Eds., IEEE Nuclear and Plasma Sciences Society. Phoenix, AZ: IEEE, July 2002, pp. 45–50.

[51] C. Yui, G. Swift, and C. Carmichael, "Single-event upset susceptibility testing of the Xilinx Virtex II FPGA," in *Proceedings of the 5th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*. Laurel, MD: NASA Office of Logic Design, September 2002.

[52] P. K. Samudrala, J. Ramos, , and S. Katkoori, "Selective triple modular redundancy for SEU mitigation in FPGAs," in *Proceedings fo the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*. Washington, D.C.: NASA Office of Logic Design, AIAA, 2003, p. C1.

[53] W.-J. R. Huang, "Dependable computing techinques for reconfigurable hardware," Ph.D. dissertation, Stanford University, June 2001.

[54] Xilinx, "Radiation hardened Virtex-II QPRO 1.5V platform FPGAs: Introduction and overview," Xilinx, Inc., San Jose, CA, Datasheet DS124-1, July 2003.

[55] J. Moore, "Military & aerospace," Xilinx, Inc., Scottsdale, AZ, Presentation, November 2003. [Online]. Available: http://www.xilinx.com/products/milaero/MilAero.pdf

[56] P. Sundararajan, C. Patterson, C. Carmichael, S. McMillan, and B. Blodget, "Estimation of single event upset probability impact of FPGA designs," in *Proceedings of the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*. Washington, D.C.: NASA Office of Logic Design, September 2003, p. P67, http://klabs.org/richcontent/MAPLDCon03/papers/p/p67_sundararajan_p.doc.

[57] C. Yui, G. Swift, and C. Carmichael, "Single event upset susceptibility testing of the Xilinx Virtex II FPGA," in *Proceedings of the 5th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD).* Laurel, MD: NASA Office of Logic Design, September 2002.

[58] G. M. Swift and S. M. Guertin, "In-flight observations of multiple-bit upset in DRAMs," *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2386–2391, December 2000.

[59] R. Koga, J. George, G. Swift, C. Yui, L. Edmonds, C. Carmichael, T. Langley, P. Murray, K. Lanes, and M. Napier, "Comparison of Xilinx Virtex-II FPGA SEE sensitivities to protons and heavy ions," *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2825–2833, October 2004.

[60] R. Koga, K. B. Crawford, P. B. Grant, W. A. Kolasinski, D. L. Leung, T. J. Lie, D. C. Mayer, S. D. Pinkerton, and T. K. Tsubota, "Single ion induced multiple-bit upset in IDT 256K SRAMs," in *Proceedings of the Second European Conference on Radiation and its Effects on Components and Systems (RADECS)*, St. Malo, France, September 1993, pp. 485–489.

[61] G. M. Swift, "Virtex-II static SEU characterization," Xilinx Radiation Test Consortium, Tech. Rep. 1, 2004.

[62] E. Johnson, K. Morgan, N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Detection of configuration memory upsets causing persistent errors in SRAM-based FPGAs," in *Proceedings of the MAPLD Conference*, September 2004.

[63] G. Asadi and M. B. Tahoori, "Soft error rate estimation and mitigation for SRAM-based FPGAs," in *Proceedings of the FPGA Conference*, Monterey, CA, February 2005, pp. 149–159.

[64] A. Holmes-Siedle and L. Adams, *Handbook of Radiation Effects*, 2nd ed. Oxford University Press, 2002.

[65] K. S. Morgan and M. J. Wirthlin, "Predicting on-orbit SEU rates," July 2005. [Online]. Available: https://dspace.byu.edu/handle/1877/64

[66] ——, "Correlation of fault-injection to proton accelerator persistent cross section measurements," July 2005. [Online]. Available: https://dspace.byu.edu/handle/1877/63

[67] T. L. Turflinger and M. V. Davey, "Understanding single event phenomena in complex analog and digital integrated circuits," *IEEE Transactions on Nuclear Science*, vol. 37, no. 6, pp. 1832–1838, December 1990.

[68] "CREME96 Homepage." [Online]. Available: https://creme96.nrl.navy.mil/

[69] "Space Radiation Associates Homepage." [Online]. Available: http://www.spacerad.com/

[70] Wikipedia, "Magnetosphere." [Online]. Available: http://en.wikipedia.org/wiki/Magnetosphere

[71] M. G. Kivelson and C. T. Russell, *Introduction to Space Physics*. Cambridge University Press, 1995.

Appendix

Appendix A

Benchmark Designs

This chapter describes the four benchmark applications referred to throughout this thesis. The purpose of the suite of benchmarks designs is to demonstrate the effects of the concepts presented in this thesis for a variety of typical designs. The first is an array of feed-forward multipliers with no internal state. This circuit represents a typical data-flow application. The second is a large array of 400 8-bit counters (each containing count state). This circuit represents an application with significant amounts of internal state. The third is a synthetic application consisting of LFSRs and a multiplier-adder tree. This circuit represents the typical mixture of data-flow and internal state in an application. The fourth is a Digital Signal Processing (DSP) kernel developed at Los Alamos National Laboratory. This circuit is represents a real application. The information included in this chapter includes design functionality and size.

Table A.1: Resource Utilization

Design	Resource Utilization	
	slices (#)	(%)
Multiplier	10,305	83.9
Counter	2,151	17.5
Synthetic	2,538	20.7
DSP Kernel	5,746	46.8

A.1 Multiplier

The first application is a feed-forward multiplier depicted as a block level diagram in Figure A.1. The design takes as inputs two 36-bit operands a and b and computes a 72-bit output $o = 8 * a * b$. In stage one, each operand enters eight independent parallel feed-forward multiply circuits. The output of each multiply circuit feeds the input of one of four full adders in stage two. In stage three, the outputs of the full-adders in stage two feeds another stage with two more full-adders. Finally, in stage four, the outputs from stage three enters a final single full-adder. In a Xilinx FPGA this application requires $10,305$ slices (see Table A.1). This many slices occupies 83.9% of a Xilinx Virtex XCV1000 FPGA (see Table A.1).

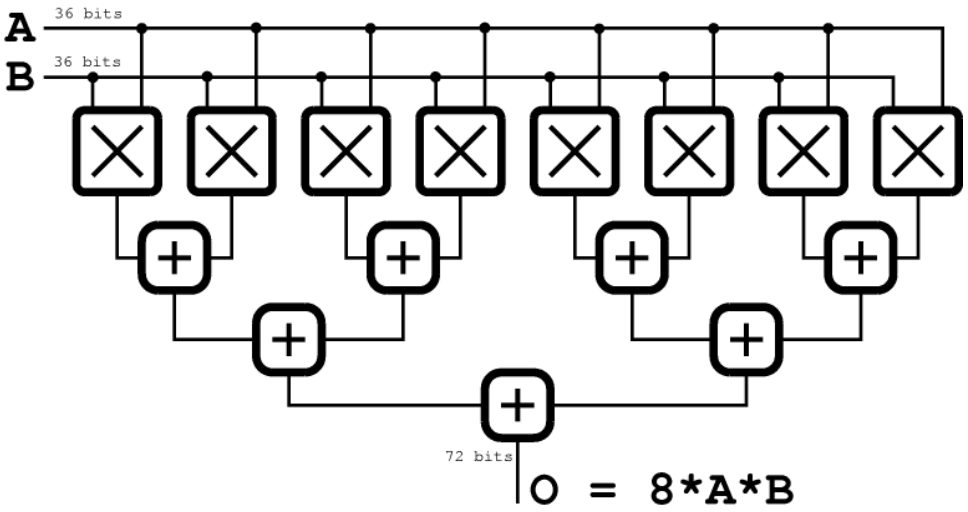


Figure A.1: A feed-forward multiplier circuit [4].

A.2 Counter Array

The second application is an array of 400 8-bit counters depicted as a block level diagram in Figure A.2. The design has no inputs and generates a 50-bit output o. Stage one has 400 8-bit counter modules. The eight outputs of each module are paired down to a single parity-check bit through a logical XOR operation. These

400 bits are further reduced to 50 bits through a 3-level XOR operation. The XOR operation allows detection of all single bit upsets in the configuration memory of this design. In a Xilinx FPGA this application requires 2,151 slices (see Table A.1). This many slices occupies 17.5% of a Xilinx Virtex XCV1000 FPGA (see Table A.1).

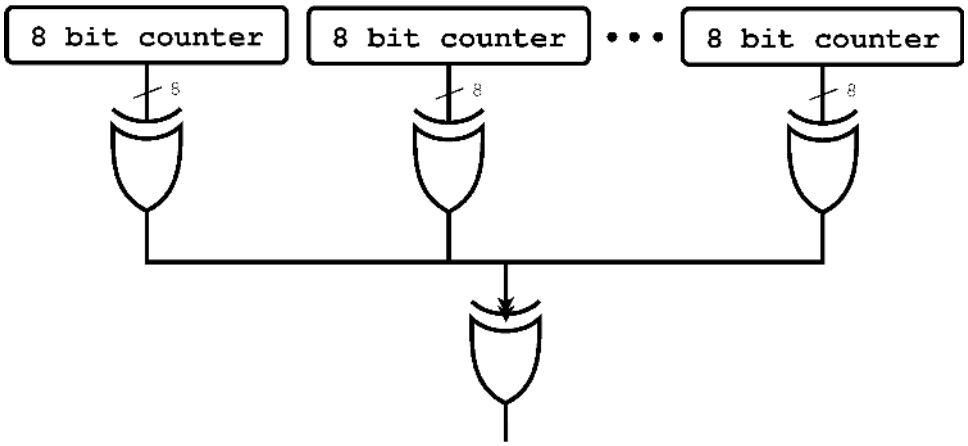


Figure A.2: A feed-forward multiplier circuit [4].

A.3 Synthetic

The third application is a synthetic application with an array of n LFSR modules followed by a multiplier-adder tree (see Figure A.3). Each LFSR module contains six 20-bit LFSR circuits. Each LFSR circuit consists of a 20 bit shift-register with an XOR'd version of a subset of its own bits as its input. In each LFSR module, the outputs of the six LFSR circuits are XOR'd together to form a single bit output for the module. The single bit output of the n modules feed into a multiply-add tree. In a Xilinx FPGA this application requires 2,538 slices (see Table A.1). This many slices occupies 20.7% of a Xilinx Virtex XCV1000 FPGA (see Table A.1).

A.4 DSP Kernel

The fourth and final application is a DSP kernel design, depicted in Figure A.4 as a block diagram. This application was developed at Los Alamos National Labora-

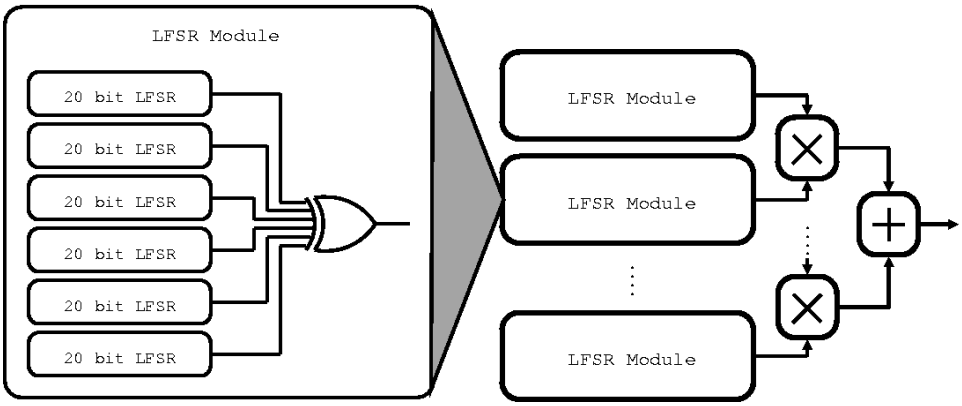


Figure A.3: A synthetic FPGA design with LFSRs and a multiplier-adder tree [4].

tory. The application takes as input a stream of data. The data is filtered through a polyphase filter and separated into 32 separate channels. The filter is followed by an FFT and a magnitude operation. The output is also a stream of data. In a Xilinx FPGA this application requires 5.746 slices (see Table A.1). This many slices occupies 46.8% of a Xilinx Virtex XCV1000 FPGA (see Table A.1).

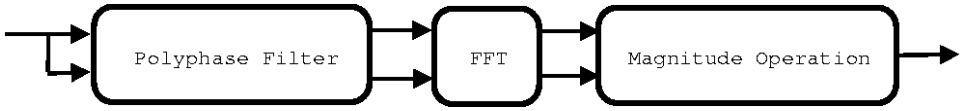


Figure A.4: A digital signal processing kernel design developed at Los Alamos National Laboratory [4].

Appendix B

Space Radiation Environment

This chapter describes both terrestrial and cosmic radiation sources, including trapped particles and galactic and solar cosmic rays.

B.1 Trapped Radiation

Moving from the earth outward, the first major source of radiation is a region of "trapped" particles. American physicist James Van Allen is credited with discovering this region of protons and electrons. He found that this range of particles extends from 800 kilometers to 6 earth radii and beyond. Figure B.1 depicts these trapped particle regions, now known as the Van Allen belts. The earth's magnetic field causes the particles to constantly move in a complicated pattern, shown in Figure B.2. They gyrate around and bounce along magnetic field lines. The field also reflects the particles back and forth between regions of maximum field strength in opposite hemispheres. In addition, electrons drift eastward while protons drift westward around the earth [5]. The Van Allen belts have electrons with energies up to 10 MeV and protons with energies up to several hundred MeV; energies high enough to easily cause SEEs in micro-electronics.

Figure B.3 is the integral Linear Energy Transfer (LET) spectrum for trapped protons along a hypothetical low-earth orbit at 560 km altitude and 35.0^o degrees inclination. The y-axis is the integral proton flux for one revolution along this orbit at each energy along the x-axis. Note that particles with energy above 10^3 do not exist for this orbit.

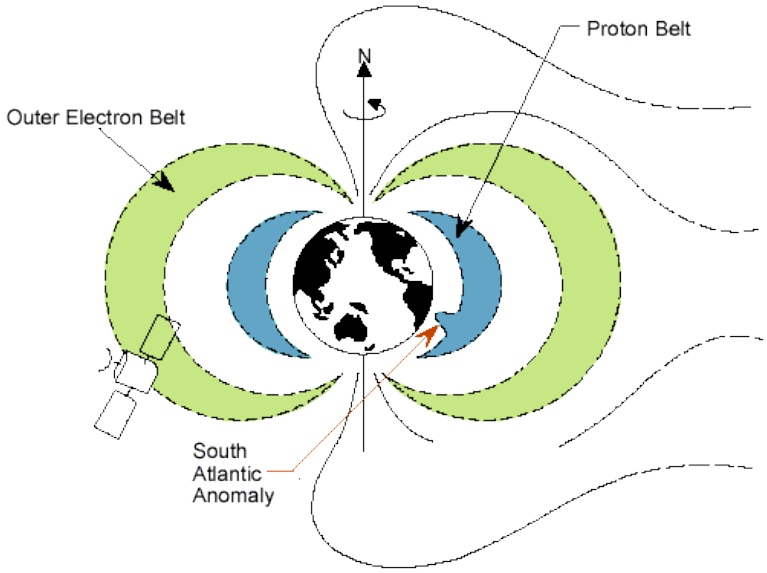


Figure B.1: Van Allen trapped radiation belts around the earth [2].

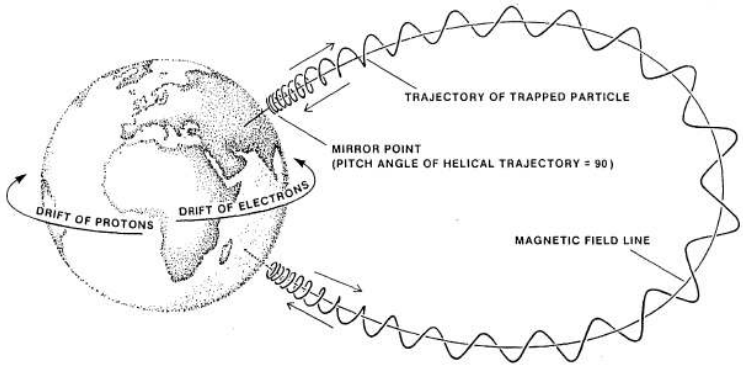


Figure B.2: Motion of trapped particles in the Van Allen radiation belts [5].

B.2 Cosmic Radiation

Beyond the trapped radiation belts exist more energetic particles of galactic and solar origination. These celestial particles include electrons, with energies in the eV to GeV range. The flux of the lower-energy particles can be significant.

Fortunately the earth's magnetic field, or magnetosphere, protects us on the ground from the most damaging galactic and solar particles. Figure B.4 is a graph-

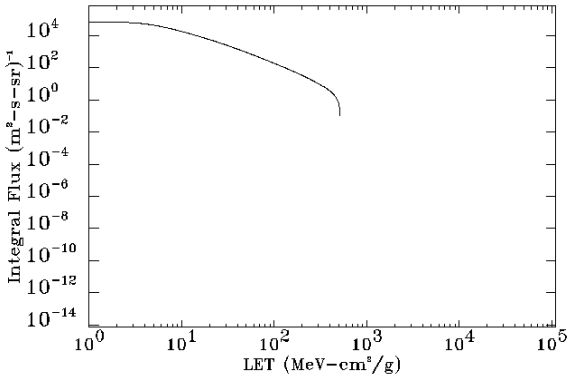


Figure B.3: Integral trapped proton LET spectrum for a low-earth orbit at 560 km altitude and 35.0^o degrees inclination.

ical representation of the magnetosphere. Near the earth, a magnetic dipole model (slightly tilted from the rotation axis and offset from the center of the earth) approximates the magnetic field [70, 71]. For this reason, the magnetosphere less effectively deflects cosmic rays at higher altitudes, particularly near the earth's poles. Further away from the earth's surface, the dipole model of the magnetic field breaks down. Due to effects from the solar wind, the anti-solar side of the magnetic field extends beyond 200 R_e ($1R_e = 1\ Earth\ Radius \approx 6370km$). For the same reasons, the sunward facing side of the field is compressed so that it only extends to approximately $10R_e$. As such, the ability of the magnetic field to deflect particles also decreases with increasing altitude.

B.2.1 Galactic Cosmic Radiation

Cosmic rays of unknown origin significantly contribute to particle flux at high altitudes above the earth and near the earth's magnetic poles. These rays, known as Galactic Cosmic Rays (GCR), typically consist of about 85 percent protons, 14 percent alpha particles and 1 percent heavier nuclei [5]. Particles with energy in the GeV range also sometimes exist in GCRs. Again, these energies easily induce faults in micro-electronics.

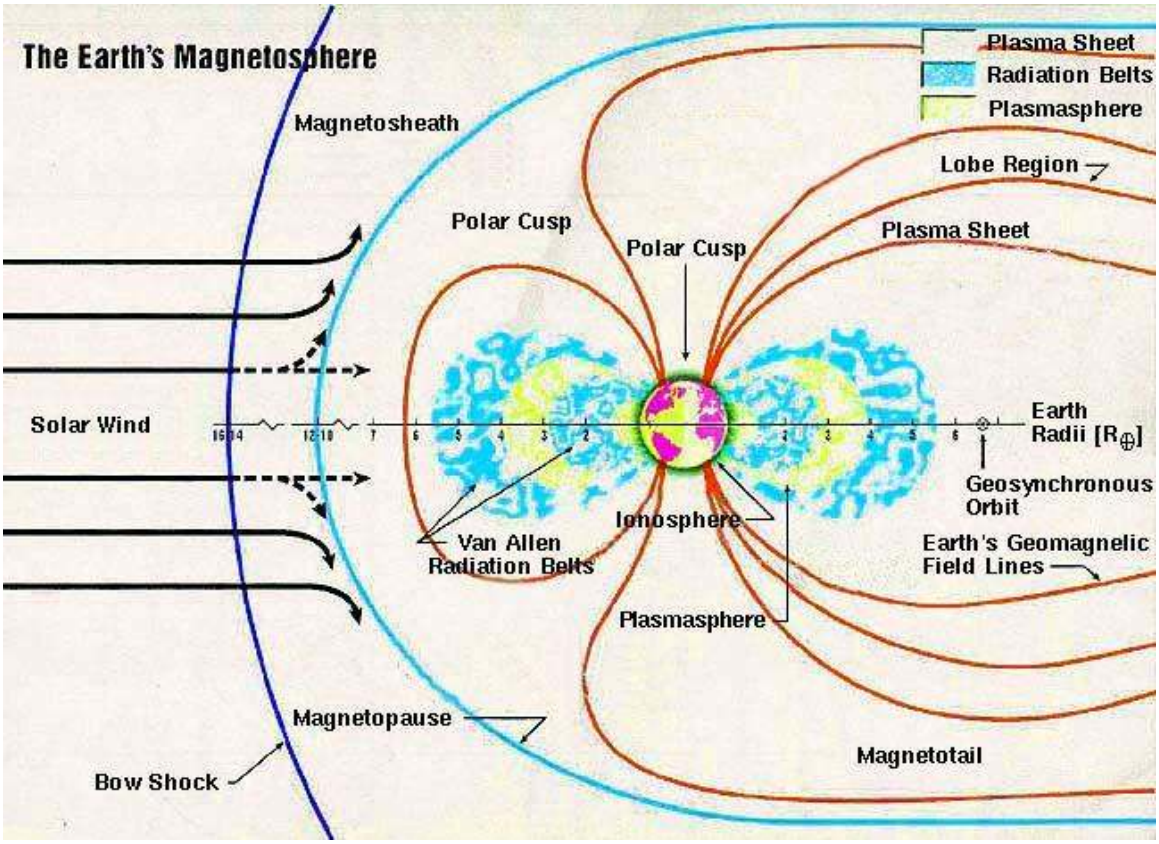


Figure B.4: The earth's magnetosphere [2].

B.2.2 Solar Cosmic Radiation

The sun also periodically ejects intense quantities of cosmic rays. The particles in these rays can also penetrate the earth's magnetosphere, particularly at high altitudes and near the poles, and significantly increase the particle flux at those locations [1]. The frequency and intensity of these solar ejections follows an approximate 22 year cycle. For the first 11 years, known as Solar Minimum, the sun is relatively quiet. For the next 11 years, known as Solar Maximum, the sun emits extreme amounts of protons and heavy ions from time to time. These events, colloquially known as flares, can last from a few hours to several days.

[1] In general, increased particle flux will lead to more SEEs. However, the increased particle flux from solar ejections does not disseminate down to lower altitudes. In fact, at solar maximum, increased solar rays in the atmosphere cause the atmosphere to expand. This allows more trapped protons to be removed from the trapped radiation belts [68]. Consequently, low-altitude particle flux actually decreases during Solar Maximum.

Figure B.5 is the combined integral LET spectrum for galactic and solar cosmic rays along a hypothetical low-earth orbit at 560 km altitude and 35.0^o inclination. The y-axis is the integral flux for one revolution along this orbit at each energy along the x-axis. Particles exist with significantly higher energies than trapped protons, beyond 10^4 LET, but the flux significantly drops after about 10^3 LET. In addition, the flux of even the lower-energy particles is orders of magnitude less than trapped proton flux at the same energy. As such, trapped particles play the dominant role in this hypothetical orbit. The opposite would be true for orbits at altitudes beyond the radiation belts or near the earth's poles.

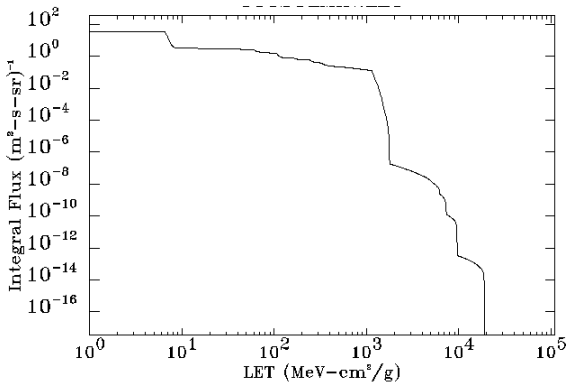


Figure B.5: Integral heavy-ion LET spectrum for a low-earth orbit at 560 km altitude and 35.0^o degrees inclination.

Appendix C

Testing Methodologies

To validate the existence the of non-persistent and persistent cross sections it was necessary to measure their respective sizes through fault-injection and proton irradiation experiments. The size of the cross sections was estimated using fault-injection. The estimates were validated by actually measuring the cross sections with proton irradiation. This appendix will describe the test-fixture and methodology for both of these tests. The limitations of each method will also be discussed and it will be shown that fault-injection is a reliable alternative.

C.1 Test-Fixture

Both the fault-injection and proton irradiation tests used the SLAAC1-V configurable computing board as a test-fixture. Figure D.1 is a schematic representation of the SLAAC1-V. It has three Xilinx Virtex XCV1000 FPGAs and one smaller Xilinx FPGA. FPGA X1 houses the circuit Design Under Test (DUT). FPGA X2 holds a second, "golden", or identical copy of the circuit programmed into X1. FPGA X0 contains difference circuitry. When the outputs of X1 and X2 do not match, X0 detects the difference and notifies the host PC. The fourth, smaller FPGA labeled XVPI has the configuration circuitry to program X0, X1 and X2. During proton radiation it also houses the scrubbing circuitry.

C.2 Fault-Injection

Fault-injection with respect to an FPGA is the process of emulating an SEU, followed by scrubbing. In general, the emulation algorithm is as follows. A bit within

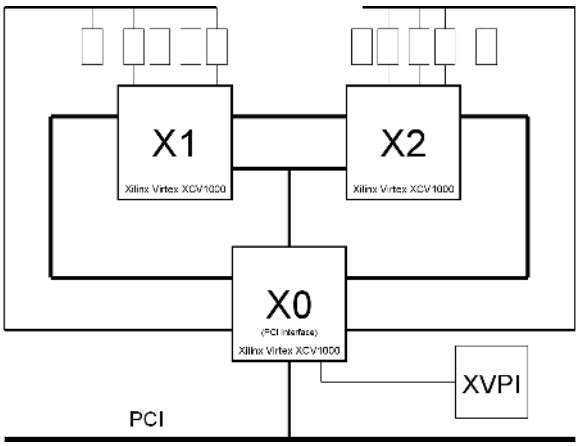


Figure C.1: The SLAAC1-V configurable computing platform.

the configuration bitstream is toggled from its correct state. The bit is left in this corrupted state for a finite duration. The length of time generally corresponds to Mean Time To Repair (MTTR) defined in Section 4.2. An upper bound on scrubbing time to repair, MAX(TTR) can also be used. Upon expiration of this time, the corrupted bit is toggled again, restoring its original state.

To perform these experiments the capability of an existing fault-injection tool developed by Eric Johnson at Brigham Young University was modified and extended. His tool estimates the size of a design's dynamic cross section by finding the sensitive bits. In other words, the tool identifies which bits, when upset, cause any type of output error. A more detailed description of this tool can be found in [4].

The primary experiment tried to identify which bits, when upset, caused persistent errors. Figure C.2 shows the sequence of events used to determine if a given configuration bit contributes to persistent errors. At the beginning of the test, marked by the diamond, the tool emulates an SEU by corrupting a bit in the configuration memory of FPGA X1. Some of those result in a dynamic error, meaning the outputs of X1 and X2 no longer match. Next, the delay t_s is introduced to emulate the time that would be required for scrubbing circuitry to find and repair the corrupted bitstream location (MTTR). At the next event, marked by the triangle, the bit is corrected. If functional errors occurred during t_s, the design is allowed to operate

for an additional time t_f to see if the errors flush. At the final event, marked by the circle, the bit is classified. If at this point errors still exist, then the originally corrupted bit is classified as contributing to persistent errors. The tool tests every bit within the configuration memory of the entire FPGA and records the result.

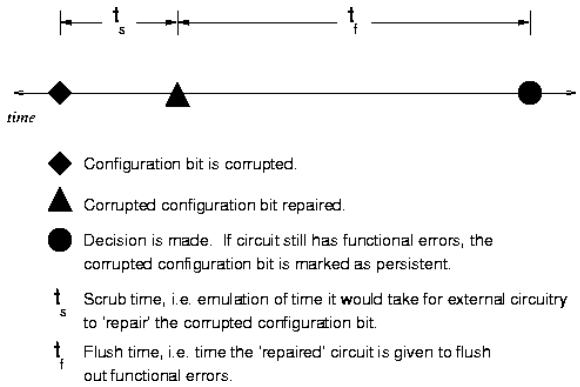


Figure C.2: Fault-injection test time-line: Sequence of events in a single trial to test a configuration bit for persistence.

The estimated size of a design's persistent cross section x_p is proportional to the sum of persistent configuration bits identified in the experiment described above. More formally, the persistent cross section is,

$$x_p = x_s \times \frac{\text{persistent bits}}{\text{total bits}}, \tag{C.1}$$

where,

$$x_s = \text{device static cross section.}$$

In other words, the persistent cross section x_p equals the ratio of persistent bits to total bits multiplied by the device's total static cross section x_s. In other words, the fraction of the original static cross section that corresponds to persistent bits.

C.3 Radiation Testing

To validate fault-injection persistent cross section estimates, proton irradiation experiments were performed at Crocker Nuclear Laboratory at the University

of California, Davis, to actually measure the persistent cross section. Here, rather than emulate SEUs and scrubbing, energetic particles and a scrubbing circuit were used. At Crocker the particles are 63 MeV protons. The on-board configuration controller chip (XVPI) was modified to perform scrubbing. The scrubbing circuit did a round-robin check of the entire bitstream; reconfiguring each corrupted frame[1] in the part.

To measure persistence at an accelerator I wrote software to control the SLAAC1-V while it was subjected to irradiation. The software ran on a host PC to which the SLAAC1-V board was attached. Figure C.3 details the sequence of events used to determine if a particular bit is within the persistent cross section of a particular design. A proton randomly induces an SEU; i.e. a configuration bit is corrupted. Within $MAX(\text{TTR})$, denoted as t_s, the scrubbing circuit will find and reconfigure the corrupted bit. At some point, marked by the square, the corrupted bit will potentially cause output errors. Here the experiment begins. The DUT, configured in FPGA X1, is allowed to operate for time t_f to see if the functional errors will flush from the design. Note that t_f is set to cover both $MAX(\text{TTR})$ and the desired experimental flush time. If at the end of t_f functional errors still exist, then the originally corrupted bit is classified as persistent.

The measured size of the persistent cross section, x_p of a design can be calculated as,

$$x_p = \frac{\#\text{ persistent events}}{\text{fluence} \times \cos\theta}, \tag{C.2}$$

where,

$$\theta = \text{incident particle angle.}$$

Here the persistent cross section is equal to the number of persistent events divided by the product of the fluence and cosine of the incident particle angle. Where the number of persistent events is defined to be the number of trials which caused persistent errors.

[1] A frame is the smallest divisible unit of the configuration memory which can be reconfigured in a Xilinx FPGA.

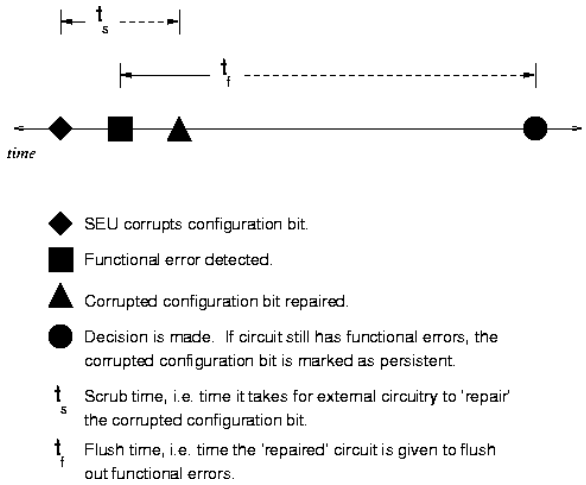


Figure C.3: Proton irradiation test time-line: Sequence of events in a single trial to test a configuration bit for persistence.

C.4 Testing Conclusions

Even though proton irradiation is expensive and time-consuming, it is a more accurate representation of the radiation environment a system will face. The primary purpose then in testing at a proton accelerator was to validate the accuracy of fault-injection predictions against proton irradiation. If it can be shown that there is a high degree of correlation between fault-injection predictions and radiation measurements, then the fault-injection tool can be used to make future "measurements" of cross section with a high level of confidence.

Unfortunately testing for persistence is an intractable problem which can only be estimated. For instance, at an accelerator the corruption of bits cannot be precisely controlled by the user due to the random flux of particles in both space and time. Therefore, reconciliation of upset bits to persistent errors is done by post-processing of collected data. A detailed explanation of the post-processing methodology, including a more detailed description of the data collection method and an extended discussion on the limitations of testing for persistence, can be found in Appendix D.

Despite the limitations in the two testing methodologies, the data collected indicates that fault-injection is a reliable method of measuring cross section. In all test cases, fault-injection is at least is accurate to within 1% for measuring dynamic cross

section and is at least as good as an order-of-magnitude estimate of the persistent cross section [12].

Appendix D

Correlation of Accelerator Data to Simulation Results

In Chapter 5 a fault injection tool was introduced that can predict the size of an FPGA design's persistent cross section. This tool was validated using proton irradiation testing at Crocker Nuclear Laboratory in Davis, CA to show that it can correctly predict the cross section measured at an accelerator.

This document outlines the data collection process used at the accelerator and the process used to correlate the results of fault-injection and proton irradiation. Section D.1 describes the accelerator hardware and Section D.2 describes the accelerator software. Section D.3 discusses the limitations of both fault injection and accelerator testing. Section D.4 introduces the software developed to correlate the results of fault injection and proton irradiation. Sections D.5 and D.6 discuss correlation for sensitivity and persistence respectively. Finally, Section D.7 accounts for the differences in persistent cross section results between fault injection and proton irradiation.

D.1 Data Collection Hardware

A SLAAC1-V configurable computing board was used as the hardware test-fixture for radiation testing. Figure D.1 is a block diagram of the SLAAC1-V board. It has four FPGAs on a PCI card. There are three Xilinx XCV1000s available for user configuration, labeled X0 through X2. The fourth, smaller FPGA labeled XVPI, acts as a configuration controller. For the persistence tests, X1 was used for the circuit design under test (DUT) and X2 for an identical copy of X1 to be used as a "golden" or control circuit. X0 housed the logic to compare outputs from X1 and X2. XVPI

contained the circuitry for configuration programming and for SEU monitoring of the X1 configuration bitstream.

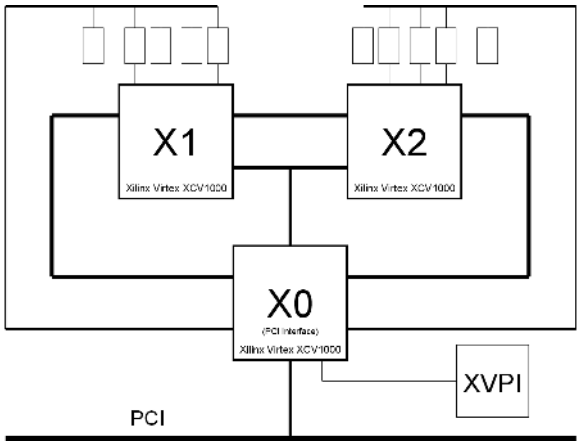


Figure D.1: Block diagram of the SLAAC1v configurable computing PCI board. The SLAAC1V has three Xilinx Virtex XCV1000 FPGAs.

D.1.1 Output Error Detection

FPGA X0 continuously monitored the stream of outputs from the X1 DUT and X2 golden circuits for discrepancies. On the first cycle on which a mismatch occurred, X0 issued an interrupt via the PCI interface to indicate an output error (OE) occurred. The interrupt remained high until it was reset externally from software. X0 also stored several user registers. One register, R0, held the number of cycles on which an OE occurred since the last register reset.

D.1.2 SEU Detection

FPGA XVPI continuously monitored the configuration bitstream for faults. It used an external memory to house a "golden" copy of the bitstream for comparison. Mismatches, i.e. faults, were logged in a FIFO by configuration bitstream offset. If after a read of the entire bitstream it had logged one or more faults, an interrupt was issued via the PCI interface. The interrupt remained high until it was reset

externally from software. Note that XVPI did not correct faults. The bitstream remained in a corrupted state until software issued data and commands to perform partial reconfiguration.

It is important to note that, on average, a read of the entire bitstream took 22 milliseconds. As a result, most SEUs which caused functional errors were were not reported until some time after the functional errors were reported. In other words, OEs induced by an SEU were actually logged and timestamped before the SEU which could have caused it.

D.2 Data Collection Software

Software running on the host PC which housed the SLAAC1-V PCI card continuously waited for and logged interrupts issued by the test-fixture hardware. The software also monitored the X0 user registers and sent commands for partial reconfiguration of the bitstream. The software contained two independent threads to perform these functions. One thread reacted to the OE interrupt and one responded to the bitstream fault interrupt.

D.2.1 Output Error Thread

The OE monitoring thread continuously pended on the OE interrupt. Upon an OE interrupt, the thread gained context. It then followed the sequence of events on the time-line in Figure D.2. Figure D.3 is a more detailed flow diagram of the operations performed by the thread. First, the thread disabled the OE interrupt, logged a timestamp from a generator with millisecond accuracy, and immediately went back to sleep for t_f milliseconds. After t_f elapsed, context returned to the thread. At this point the thread reset the register R0 which held the number of cycles on which an error occurred since the last register reset. It then went back to sleep for t_m milliseconds. After t_m elapsed, context returned to the thread again. At this point the thread read the contents of R0. If R0 was non-zero, or in other words, if OEs occurred during t_m the log was annotated with a flag to indicate that the original OE (represented by the timestamp) was persistent. Otherwise a flag was

added to the log to indicate a non-persistent OE occurred. After a persistent output error (POE), the thread issued a global reset of X1 and X2. Finally it re-enabled the OE interrupt and went back to sleep to wait for the next interrupt.

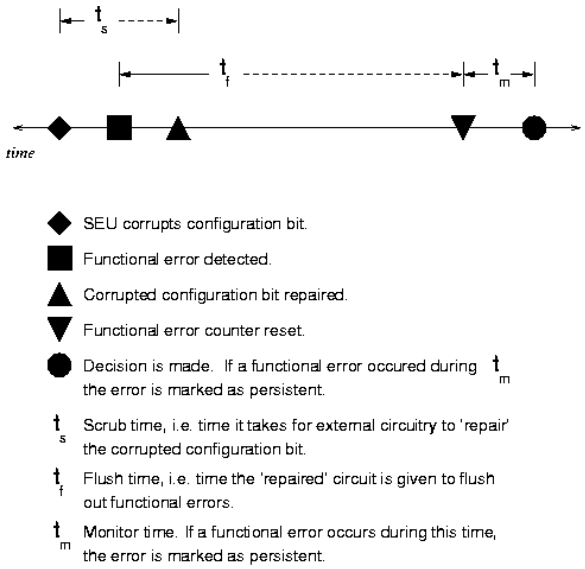


Figure D.2: Timeline of events in the fault-injection tool to test a configuration bit for persistence.

D.2.2 Bitstream Fault Thread

The software also included a second thread which continuously pended on the bitstream fault interrupt. This thread remained inactive until a bitstream fault interrupt was issued at which time it gained context. The thread then disabled the bitstream fault interrupt and logged a timestamp from the same generator used by the OE thread. Next, it read each entry from the FIFO in XVPI and repaired the bitstream at the reported location. Next to the timestamp in the log, each bitstream offset was also added. Finally the bitstream fault interrupt was re-enabled and the thread went back to sleep to wait for the next interrupt.

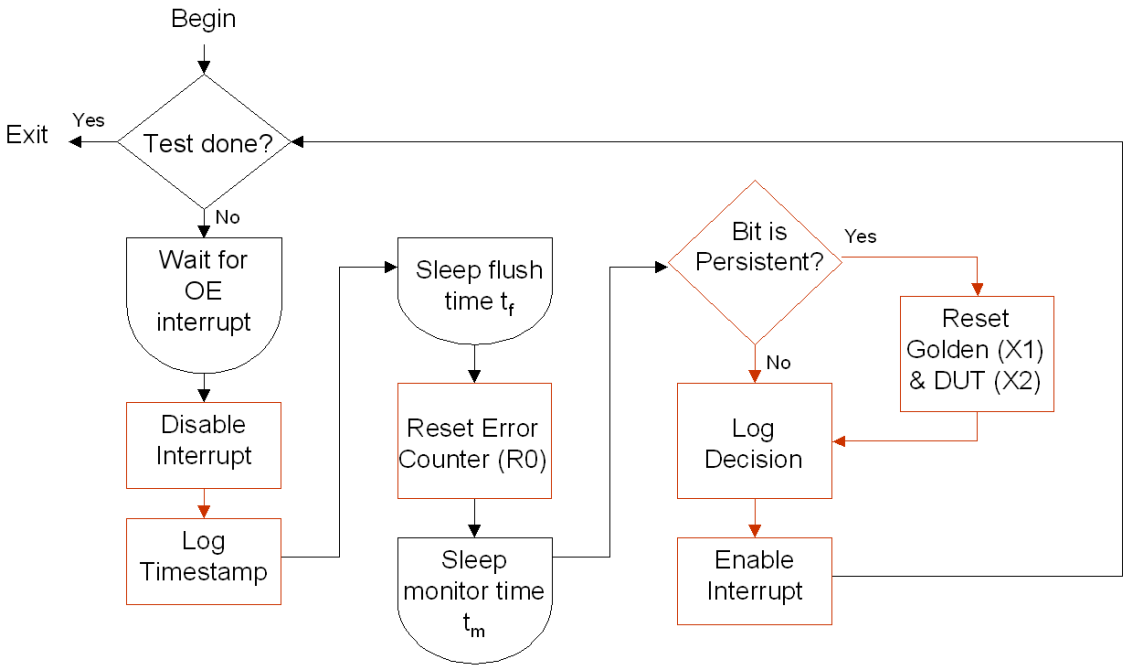


Figure D.3: Flow diagram of the Output Error thread in the data collection software for accelerator persistence tests.

D.3 Testing Limitations

Several factors affect the ability to measure persistent cross section. Consequently there are distinct differences between the fault injection and proton irradiation methodologies. The first difference between the two testing methodologies is the number of configuration bits tested. The fault-injection tool tests *every* configuration bit while the radiation test only tests a small random subsection of the configuration memory. Sampling limitations of the radiation test may produce slightly different results than the exhaustive fault-injection tool.

The second difference between the two testing methodologies is the timing of the configuration upsets. In the fault-injection tool, the configuration upsets are carefully controlled and synchronized with the run time and flush time of the persistence test. In the radiation test, however, protons randomly arrive and cannot easily be correlated with run time and flush time. Although the proton flux controls the average arrival time, the inter arrival of protons follows a Poisson distribution.

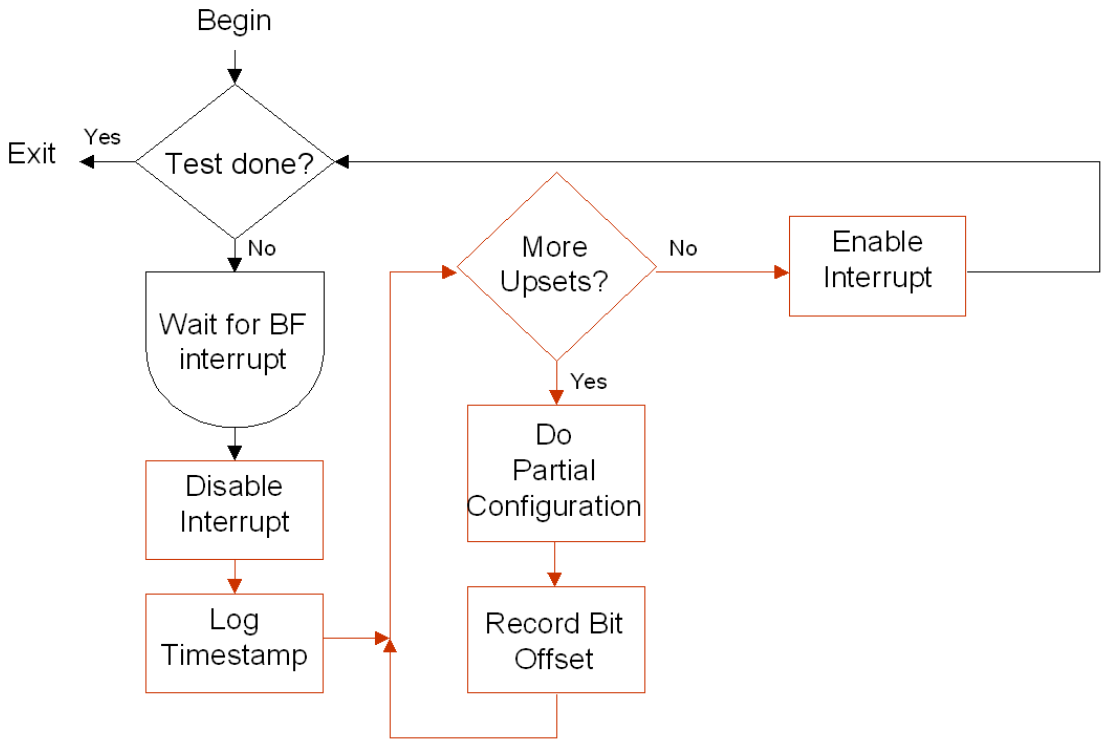


Figure D.4: Flow diagram of the Bitstream Fault thread in the data collection software for accelerator persistence tests.

Expectedly, secondary configuration upsets occur during the flush time t_f of a trial. If this happens, the test software falsely tags an upset configuration bit as persistent. The number of such false persistent events can be estimated through statistical analysis.

Upsets of user flip-flops is the other factor which affected the accuracy of our testing. Injection of faults into user flip-flops during dynamic operation is difficult within the Virtex architecture. Fortunately, the ratio of flip-flops to configuration memory latches is small, therefore it is a generally accepted practice to ignore flip-flops during fault-injection tests. However, protons do cause SEUs within flip-flops at an accelerator. This leads to seemingly unexplainable POEs not predicted by the fault-injection tool. The number of such unexplainable events can also be estimated through statistical analysis.

D.4 Data Correlation Software

Since the ultimate goal in performing radiation experiments was to validate the use of fault-injection as an accurate method to predict cross section, the data collected using proton irradiation had to be correlated with the data collected using fault-injection. The first step in correlation was to parse the data collected during proton irradiation and fault-injection and store it in a suitable data structure. Figures D.6 and D.7 are sample data files collected by the OE and RB threads respectively. Figure D.5 is a UML diagram of the classes used to store data parsed from these files. All events are a derivative of the Entry class which simply stores a timestamp. Each event is either a record of an OE or a bitstream fault. The only additional information an OE Entry stores is the real-time decision made about the persistence of that particular error. Each bitstream fault Entry holds an array of one or more configuration upset events. Each configuration upset event in turn holds an address or offset to the bit which was corrupted at the accelerator. In addition, each upset event record stores a sensitive and persistent percent probability predicted by fault-injection for that bit.

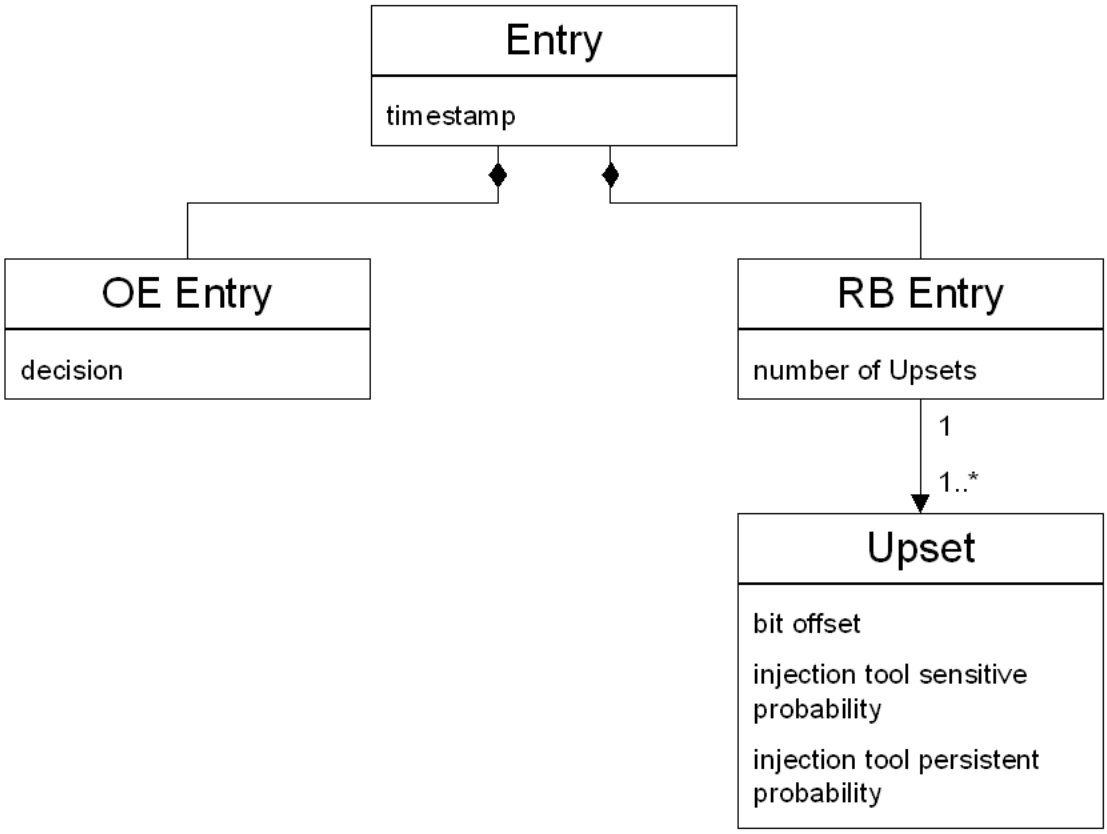


Figure D.5: UML diagram of the software classes used to represent the data structure of events parsed from the data recorded at an accelerator using the data collection software.

```
\% Radiation Effects Tests performed at UC Davis
\% by LANL in conjunction with BYU
\% Tue Jun 29 2004
\% 12:16:40
\% X1 Design: /test_designs/SSR2/ssra-new-fixed.bit
\% X2 Design: /test_designs/SSR2/ssra-new.bit
\% Frequency: 10.000000
\% Flush Time: 45 (ms)
\% Observation Time: 10 (us)
\% Beam Run Number: 10
\% OE Format:
\%       Timestamp Decision
\%       (note: Decision: 0=Non-Persistent, 1=Persistent)

13256 0
15391 0
15689 0
15963 0
17824 0
18881 0
18968 1
20408 0
20856 0
```

Figure D.6: Sample output error (OE) data log.

```
\% Radiation Effects Tests performed at UC Davis
\% by LANL in conjunction with BYU
\% Tue Jun 29 2004
\% 12:16:40
\% X1 Design: /test_designs/SSR2/ssra-new-fixed.bit
\% X2 Design: /test_designs/SSR2/ssra-new.bit
\% Frequency: 10.000000
\% Flush Time: 45 (ms)
\% Observation Time: 10 (us)
\% Beam Run Number: 10
\% RB Format:
\%       Timestamp SMAPcnt FSMcnt SMAP_STAT_REG
\%       [Byte-Offset Expected@Actual]. (one or more)
\%       (note: Each entry is a pair of two lines)

13042 1 483 3d
537604 30b
13291 1 12 3d
162487 20a
13312 1 1 3d
633763 302
13374 1 3 3d
31034 c0c1
13499 1 6 3d
75518 2
```

Figure D.7: Sample read-back error (RB) data log.

After the parser creates the data structure for each event, all Entries are placed in an array in chronological order by timestamp. Figures D.9, D.12 and D.14 are typical timelines corresponding to the events stored in an Entry array. The reader will recall from Section D.1.2 that due to the timing constraints on the SEU detection hardware, OE events come before the SEU which would have caused it.

D.5 Sensitivity Correlation

The next step in the correlation process was to re-validate the ability of the fault injection tool to predict the size of the dynamic cross section. Correlation of OEs to the sensitive upsets which caused them turns out to be a rather trivial process. For each OE reported at the accelerator it is enough to find out if it was predicted by the fault-injection tool. Figure D.8 is a flow diagram of the logic used to evaluate each OE. If after an OE event a configuration upset was reported with a non-zero sensitive probability within a window w, then the fault-injection tool correctly predicted that OE. OEs *not* near a configuration upset with a non-zero sensitive probability were termed unpredicted. Some of these unpredicted OEs can be attributed to upsets of configuration bits which the fault-injection tool incorrectly identified as non-sensitive. The remaining unpredicted OEs likely were caused by SEUs within user flip-flops.

Figure D.9 is a hypothetical snapshot in time of events recorded at the accelerator. The OE depicted in this graphic would have been classified as predicted if the upset which occurred during the time w had a non-zero sensitive probability.

It is important to note how the size of the window w was selected. To make this selection, the delta time between an OE and the first upset reported after it chronologically was evaluated. Figure D.10 is a histogram of the deltas for a particular design. The resulting distribution is normal. As expected, the mean is approximately 22 milliseconds which corresponds to the average bitstream read time mentioned in Section D.1.2. Excluding the outliers (which correspond to OEs caused by an upset of a user flip-flop), the edge of the distribution is approximately 45 milliseconds. This worst-case fault reporting time was used as the size of the window w so as to insure that all possible upsets which could have caused an OE were included.

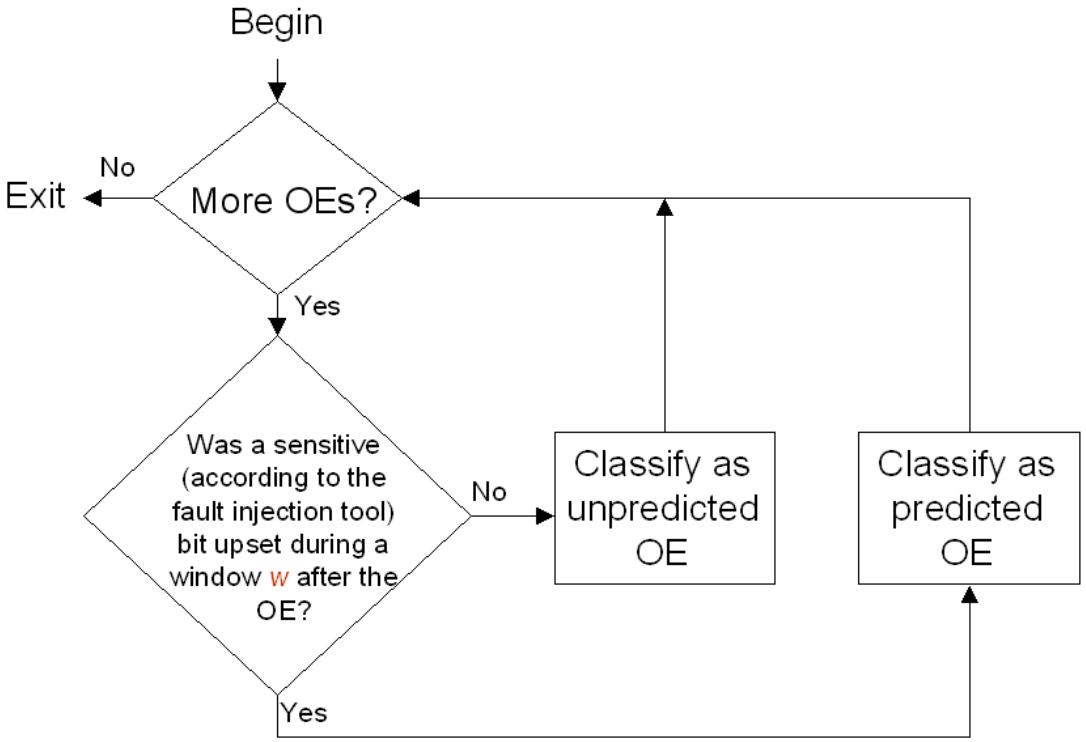


Figure D.8: Flow diagram of the algorithm to classify sensitive configuration bit data collected at an accelerator.

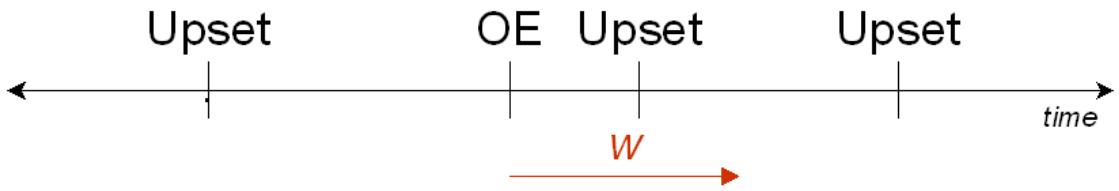


Figure D.9: Example timeline of events recorded at an accelerator.

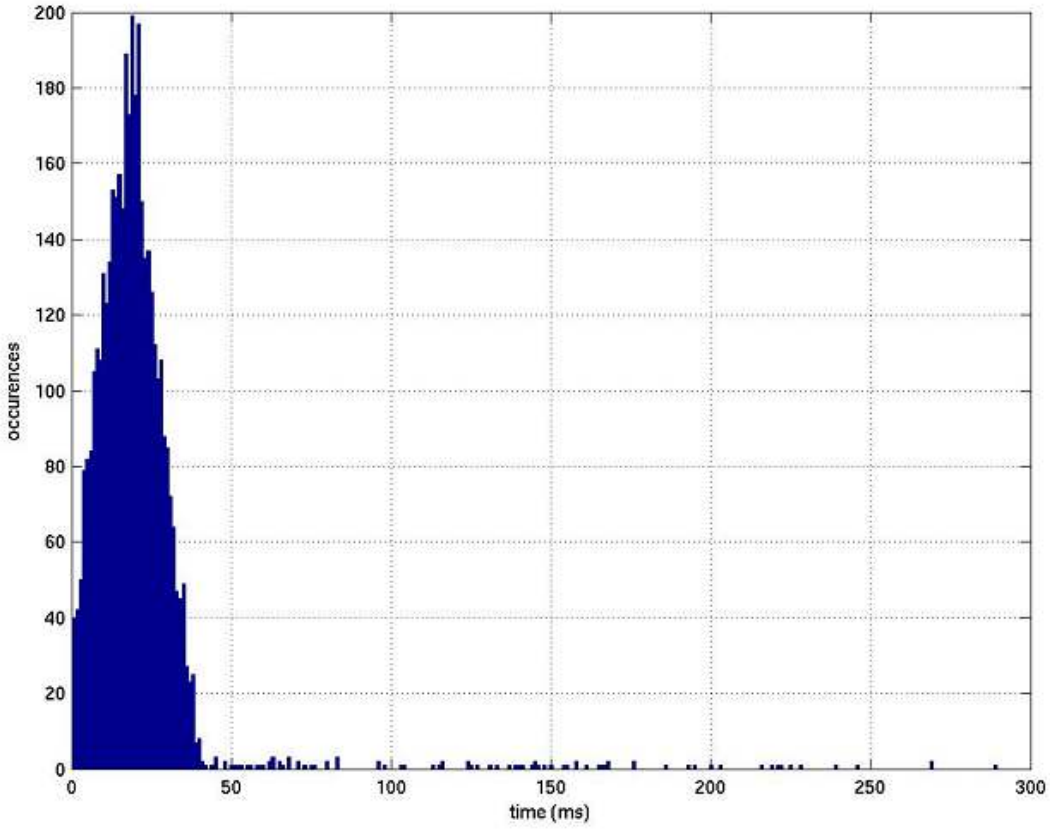


Figure D.10: Histogram of the delta time in ms between an output error event and an configuration upset event.

D.6 Persistence Correlation

After confirming the fault injection tool's ability to accurately predict dynamic cross section, several different approaches to validate its ability to predict persistent cross section were used. Correlation of POEs to the persistent upsets which caused them is a much more difficult process than sensitivity correlation. This section describes the inherent limitations in persistence correlation and the different algorithms used.

D.6.1 Detection Algorithm 1

The easiest and most intuitive approach to persistence correlation is to mimic the methodology for correlating sensitivity. For each POE reported at the accelerator, it was determined if it was predicted by the fault-injection tool. Figure D.11 is a flow diagram of the logic used to evaluate each POE. If after a POE event a configuration upset was reported with a non-zero persistent probability within a window w, then the fault-injection tool correctly predicted that POE. POEs *not* near a configuration upset with a non-zero persistent probability were termed unpredicted. Some of these POEs can be attributed to upsets of configuration bits which the fault-injection tool incorrectly identified as non-persistent. The remaining unpredicted POEs likely were caused by SEUs within user flip-flops.

Figure D.12 is a hypothetical snapshot in time of events recorded at the accelerator. The POE depicted in this graphic would have been classified as predicted if the upset which occurred during the time w had a non-zero persistent probability.

Again it is important to note how the size of the window w was selected. For this and the following persistence correlation algorithms the same approach was used. Associated with each POE in the log files was the actual flush time t_f for that particular trial. This time varied due to operating system overhead for context switching. Due to the dynamic length of the time t_f, a different w for each POE was used. So as to insure that all upsets which could have possibly induced the POE in question, the worst-case bitstream fault reporting time described in Section D.5 was added to the real flush time t_f for the window w.

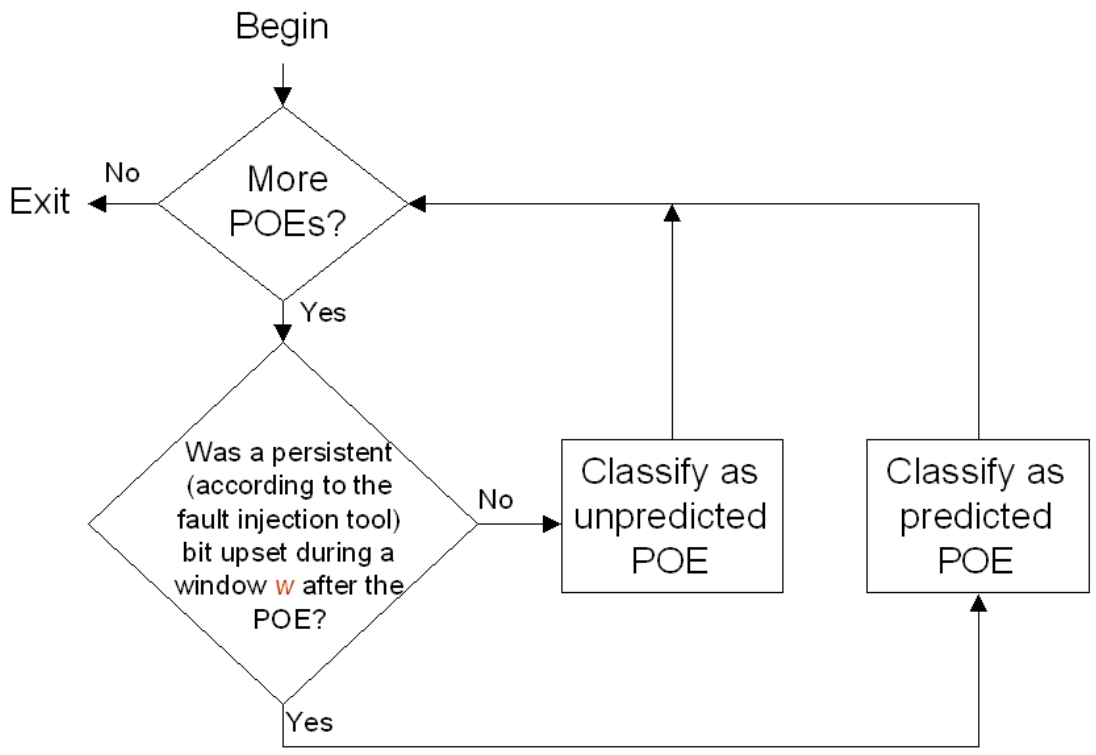


Figure D.11: Flow diagram of an algorithm to classify persistent configuration bit data collected at an accelerator.

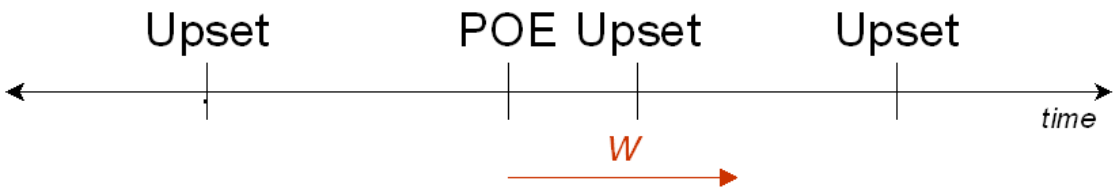


Figure D.12: Example timeline of events recorded at an accelerator.

D.6.2 Prediction Algorithm

The second approach looked at configuration upsets rather than POEs. This algorithm determined if each configuration upset with a non-zero persistent probability, as predicted by fault-injection, actually caused a POE. Due to the non-binary probability distribution of persistence an approach which weighted the occurrence of a configuration upset by its persistent probability was used. Figure D.13 is a flow diagram of the logic used to evaluate each persistent upset. The algorithm keeps a running sum of the persistent probability of each persistent upset. The sum equals a weighted prediction of the number of POE events the fault-injection tool predicted based on which configuration bits were upset. Next, the number of upsets that had a POE within a window w before the upset was counted. The count equals the actual number of POE events seen. A percent error can be calculated from the equation $(predicted - actual)/predicted$.

Figure D.14 is a hypothetical snapshot in time of events recorded at the accelerator. Each upset has been labeled according to its fault-injection persistent probability. The persistent upset depicted in this graphic had a POE within a window w chronologically before it and therefore would have contributed to the count of actual POEs. The upset's persistent probability would have also contributed to the sum defining the weighted prediction of POEs.

D.6.3 Detection Algorithm 2

The final approach used looked at every event within a window w after a POE. This more detailed analysis leads to a better explanation of why each POE event occurred. In this algorithm, each POE event was placed in one of five categories. The different categories were 1) matched, 2) anomalous, 3) one or more sensitive upsets, 4) one or more non-sensitive upsets, and 5) nothing in window. POEs with one or more persistent upset within the window w were placed in the *matched* category. Errors with more than three upsets within its window were called *anomalous*. From the remaining POEs, those with at least one sensitive upset in the window w were placed in the *one or more sensitive* category. Those with a least one upset (which by process

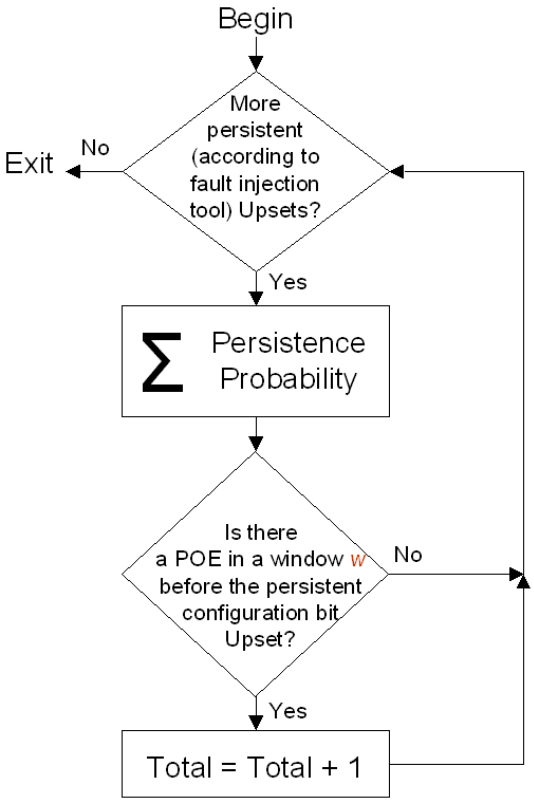


Figure D.13: Flow diagram of an algorithm to classify persistent configuration bit data collected at an accelerator.

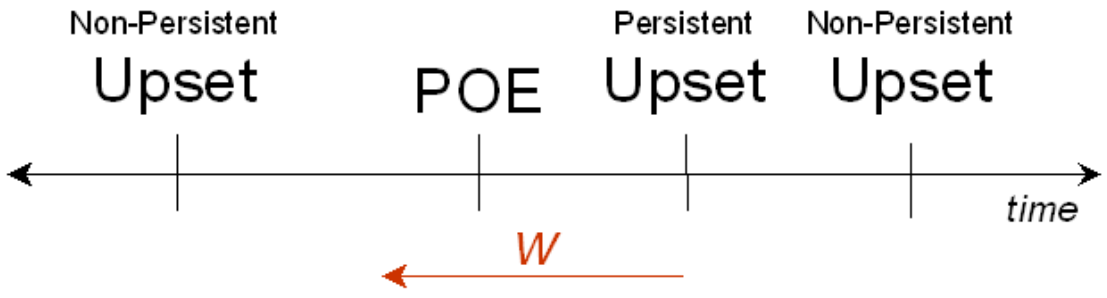


Figure D.14: Example timeline of events recorded at an accelerator.

of elimination could only be non-sensitive) were put in the *one or more non-sensitive* category. And finally, the remaining errors (which by process of elimination could have no upsets in their window) were placed in the *nothing in window* category.

D.7 Accounting for Testing Error

In his Master's thesis, Eric Johnson explained how to determine confidence intervals for dynamic radiation testing of cross section for FPGA applications with respect to fault injection [4]. In this work Johnson showed that the bounds a and b on a 95% confidence interval can be computed as

$$a = -2.81\sigma + \mu \tag{D.1}$$

and

$$b = 2.81\sigma + \mu, \tag{D.2}$$

where σ is the square root of the variance and μ is the mean. The magic numbers -2.81 and 2.81 come from the 95% confidence interval for a unit normal distribution $\Phi(x)$. For a unit normal distribution, $\Phi(-2.81) = 0.025$ and $\Phi(2.81) = 0.975$.

Johnson further showed that the variance σ^2 and mean μ can be computed as

$$\sigma^2 = \frac{k+1}{n+2}\left(\frac{k+2}{n+3} - \frac{k+1}{n+2}\right) \tag{D.3}$$

and

$$\mu = \frac{k+1}{n+2}, \tag{D.4}$$

where n is the number of SEU events and k is the expected number of persistent (or sensitive etc.) events.

Based on Johnson's derivations, 95% confidence intervals were computed for the four designs introduced in Appendix A and referred to throughout this thesis. The parameters, lower bound a, and upper bound b are shown in Table D.1. In addition, the ratio of the raw persistent cross section to static cross section is also listed for each design in column seven. This ratio is computed as

$$x_{p_m:s} = \frac{1}{\text{static cross section}} \times \frac{\#\text{ POE events}}{\text{fluence}}. \tag{D.5}$$

Only the raw measured ratio for the Synthetic design falls within the expected confidence interval. The discrepancies for the remaining designs may be explained due to the effects of flux variations and user flip-flop upsets (see Section D.3). As a result, the collected data was post-processed in various ways in an effort to show that the data could fit within the expected confidence interval. The first method discarded trials which had one or more additional SEUs within the flush time t_f. The second method only counted trials which had one or more upsets which were known to cause persistent errors that occurred within the flush time t_f. The ratio of the persistent cross section to static cross section using these methods is computed as

$$x_{p_m:s} = \frac{1}{\text{static cross section}} \times \frac{\text{adjusted \# POE events}}{\text{fluence}}. \tag{D.6}$$

As might be expected, the second method had better agreement with fault injection. The ratios using this method are listed in column eight of Table D.1.

In some cases adjusting the data by discarding trials moves the data to within the 95% confidence interval. However, this adjustment also sometimes moves data out of the confidence interval. Sometimes the raw data collected already fell within the 95% confidence interval. Ultimately all of the data collected are close to their expected 95% confidence interval. As Section D.3 indicated, measuring persistence is a difficult problem. The flux, flush time and sample rate parameters can make a significant difference in the accuracy of the collected data. In the future, more testing could be done at much lower sampling rates with much longer flush times to collect more accurate data.

Table D.1: Confidence Intervals for Tested Designs

Design	Predicted Persistence	n	expected k	a	b	Measured Persistence	
						Raw	Adjusted
Multiplier	0.0000	5,927	0.0010	-0.0003	0.0006	0.0007	0.0001
Counter	0.0187	21,068	394.0	0.0161	0.0214	0.0132	0.0128
Synthetic	0.0126	136,727	1717.1	0.0117	0.0134	0.0122	0.0094
DSP Kernel	0.0014	30,697	41.7	0.0008	0.0020	0.0026	0.0009

Appendix E

Predicting On-Orbit SEU Rates

In order to calculate Mean Time Between Failure (MTBF) it is necessary to first predict static Single-Event Upset (SEU) rates for device-orbit pairs. Many methods and associated computer programs exist to forecast on-orbit SEU rates. For this work, the Space Radiation 5.0 (SPACERAD) [69] and CREME96 [68] software packages were used for prediction. This appendix documents the parameters necessary to recreate the MTBF values, found in Tables 3.2, 5.5 and 6.2, based on the static SEU rates found in Table 2.3.

E.1 Software Packages

As mentioned in Chapter 2, the space radiation environment contains electrons, protons and heavy-ions of various originations. Of importance to SEU calculations are typically protons, both trapped and solar, and heavy-ions. The computer program SPACERAD was used to predict static SEU rates from trapped protons and solar protons. According to its website [68], SPACERAD models the ionizing space and atmosphere environments. It includes models for trapped protons and electrons, solar protons, galactic cosmic radiation and neutrons. In addition to SEU rates, SPACERAD can also predict total ionizing dose, solar cell damage and Single-Event Latchup (SEL) for any orbit or trajectory. More information can be found at the SPACERAD website [68].

The Cosmic Ray Effects on Micro-Electronics 1996 (CREME96) suite of software programs was used to predict static SEU rates from cosmic rays, both solar and galactic. A public web-based interface exists to access this software package [68].

Using CREME96, heavy ion SEU rates and total ionizing dose can be predicted for a particular orbit. More information can be found at the CREME96 website [68].

E.2 Integral Rectangular Parallelepiped Method

The underlying method in both SPACERAD and CREME96 used for static SEU rate prediction is the well-established Integral Rectangular Parallelepiped (IRPP) technique. More details on the procedure and mathematics of this technique can be found in [7, 8].

E.3 Rate Categories

The energy spectrum used for IRPP SEU rate forecasting is based on particle flux. Trapped protons, solar protons, heavy-ions and even trapped electrons all have different models to describe their respective flux at varying locations in space. As such, the component of a device-orbit SEU rate due to each particle type must be computed separately.

Each particle model has different categories to describe the variations in flux due to changing space conditions. For example, trapped proton flux has two categories, solar min and solar max, corresponding to the peak and minimum of the approximately 22 year solar activity cycle. Heavy ion rates can also be divided into solar min and solar max, but additional categories are also available to describe conditions during the worst week and worst day of solar activity. Consequently, it is important to combine the separate particle SEU calculations in a meaningful and valid manner. Table E.1 indicates which components were included for the different categories used to report SEU and MTBF rates found elsewhere in this work.

E.4 AP-8 Trapped Proton Model

The parameters to do an IRPP prediction of trapped proton SEU rates from within SPACERAD are found in Table E.2. The Weibull characteristics are entered directly, but the energy spectrum is based on a independently generated file. The file it depends on is an energy transport file. An energy transport file represents the energy

Table E.1: Solar Condition Categories and the Set of Values Necessary to Calculate a Total Rate

Category	Trapped Proton		Solar Proton		Heavy Ion			
	Solar Min	Solar Max	Solar Quiet	Solar Stormy	Solar Min	Solar Max	Worst Week	Worst Day
Solar Min	√		√		√			
Solar Max		√		√		√		
Worst Week		√		√		√	√	
Worst Day		√		√		√		√

outside the device after it has gone through spacecraft shielding. The parameters to create an energy transport file can be found in Table E.3. Here, no values are entered directly, but the energy transport file does depend on both a spacecraft shielding file and a trapped proton energy environment file, which are independently created. A trapped proton energy environment file represents the energy spectrum outside the spacecraft. The parameters for spacecraft shielding and trapped proton environment files are listed in Tables E.4 and E.5 respectively. Spacecraft shielding files are stand-alone, or in other words, all values are entered directly, but a trapped proton energy environment file depends on an orbit file. Orbit files are also stand-alone. The parameters for the orbits represented in this work are listed in Table E.6.

The categories for trapped proton SEU rates are solar min and solar max. The category is selected by the model picked for the energy environment file. (See column column three of Table E.5.)

Table E.2: Input Parameters to Predict SEU Rates in SPACERAD due to Trapped Protons

Energy Transport Spectrum	Cross Section (cm^2)	Energy Threshold (MeV)	Weibull Shape	Weibull Width
∗(see Table E.3)	$1.276x10^{-7}$	10	2	30

Table E.3: Input Parameters to Create a Trapped Proton Energy Transport File in SPACERAD

Trapped Proton Environment	Spacecraft Shielding
∗(see Table E.5)	∗(see Table E.4)

Table E.4: Input Parameters to Create a Spacecraft Shielding File in the SPACERAD Package

Material	Density	Shell Thickness
Aluminum	2.698 g/cm^3	100 $mils$

Table E.5: Input Parameters to Create Trapped Proton Environment Files in SPACERAD

	Orbit Orbit	Model Model	Year Year	Geomagnetic Field Model	Peak Flux
Solar Max	∗(see Table E.6)	AP8MAX [Epoch=1970]	0	IGRF/DGRF Internat. Geo. Ref. Field[10]	No
Solar Min	∗(see Table E.6)	AP8MIN [Epoch=1964]	0	IGRF/DGRF Internat. Geo. Ref. Field[10]	No

Table E.6: Input Parameters to Create Orbit Files in SPACERAD

Apogee (km)	Perigee (km)	Incl. (deg)	Duration (days)	Long. Asc. Node (deg)	Asc. Node to Perigee (deg)	Perigee to Spacecraft	Precession
450	450	51.6^o	365	0^o	0^o	0^o	Yes
560	560	35.0^o	365	0^o	0^o	0^o	Yes
800	800	22.0^o	365	0^o	0^o	0^o	Yes
833	833	98.7^o	365	0^o	0^o	0^o	Yes
1,200	1,200	65.0^o	365	0^o	0^o	0^o	Yes
22,200	22,200	55.0^o	365	0^o	0^o	0^o	Yes
36,000	36,000	0.0^o	365	0^o	0^o	0^o	Yes

E.5 JPL 1991 Solar Proton Model

In a similar fashion to trapped protons, the component of SEU rates due to solar protons can be determined using SPACERAD and the JPL 1991 model. The parameters to do a Weibull prediction of solar proton SEU rates are found in Table E.7. Again, the Weibull values can be entered directly, but the calculation also depends on an energy transport file, representing the energy at the device after it has gone through spacecraft shielding. The parameters to create a solar proton energy transport file are listed in Table E.8. The energy transport spectrum for solar protons depends on the solar proton energy environment outside the spacecraft, in addition to the spacecraft and geomagnetic shielding. The parameters to create solar proton environment, spacecraft shielding and geomagnetic shielding files are found in Tables E.9, E.4 and E.10 respectively. All three file types are stand-alone.

The categories for solar proton SEU rates are Quiet magnetosphere and Stormy magnetosphere, corresponding to the level of disturbance within the earth's magnetic field, typically caused by the sun. The category is determined by the field condition selected when creating the geomagnetic shielding file.

Table E.7: Input Parameters to Predict Solar Proton SEU Rates in SPACERAD

Energy Transport Spectrum	Cross Section (cm^2)	Energy Threshold (MeV)	Weibull Shape	Weibull Width
$*$(see Table E.8)	$1.276x10^{-7}$	10	2	30

Table E.8: Input Parameters to Create a Solar Proton Energy Transport File in SPACERAD

Trapped Proton Environment	Spacecraft Shielding	Geomagnetic Shielding
$*$(see Table E.9)	$*$(see Table E.4)	$*$(see Table E.10)

Table E.9: Input Parameters to Create a Solar Proton Environment File in SPACERAD

Model	Mission Duration (years)	Confidence Level (%)
JPL 1991	1	95

Table E.10: Input Parameters to Create Geomagnetic Shielding Files for the SPACERAD Software Package

Orbit Integration	Earth's Shadow	Particle Arrival	Field Condition
Orbit-Average	Include	Omni directional	Quiet
Orbit-Average	Include	Omni directional	Stormy

E.6 CREME96 Cosmic Radiation Model

The methodology for calculating SEU rates using CREME96 is very similar to that with SPACERAD, just with a different user-interface. Just as with solar protons and trapped protons, an IRPP prediction of SEU rates can be made. The appropriate Weibull values for heavy ions are listed in Table E.11. The heavy ion IRPP calculation also requires an energy transport spectrum, but CREME96 requires it to be first converted to Linear Energy Transfer (LET). The parameters used to create an LET file are listed in Table E.12. Again, the energy transport spectrum being converted represents the energy at the device after it has been transported through spacecraft shielding. Table E.13 enumerates the parameters to create an energy transport file. The energy transport files depend on the heavy ion environment. An heavy ion environment file can be created using the parameters in Table E.14. In turn, the environment depends on the orbit and consequent geomagnetic shielding in that orbit. CREME96 uses one file to represent an orbit and its geomagnetic shielding. The parameters to create an orbit file can be found in Table E.15.

The categories for heavy ion SEU rates are solar min, solar max, worst week and worst day. Solar min and solar max only account for the non-solar anomalous cosmic rays. Worst week and worst day account for "solar flare enhanced" flux averaged over the respective time period.

Table E.11: Input Parameters to Predict Heavy Ion SEU Rates in CREME96

Energy Spectrum	X (μ)	Y (μ)	Z (μ)	Funnel (μ)	Weibull Onset (MeV)	Weibull Width	Weibull Exponent	Weibull X-sctn (μ^2)
*(see Table E.12)	0	0	0	0	1.2	30	2	8

Table E.12: Input Parameters to Convert an Energy Transport File to an LET Spectrum in CREME96

Energy Transport Spectrum	Lightest z#	Heaviest z#	Min. Energy (MeV/nuc)	Device Material
*(see Table E.13)	2	28	0.1	Silicon

E.7 Static SEU Rates

The combined SEU rates for trapped protons, solar protons and heavy ions can be found in Table E.16. The rate for each orbit and each category is listed.

Table E.13: Input Parameters to Create an Energy Transport File in CREME96

Cosmic Radiation Environment	Shielding Material	Shielding Thickness (mils)
*(see Table E.14)	Aluminum	100

Table E.14: Input Parameters to Create an Energy Transport File in CREME96

Environment Model	Lightest z#	Heaviest z#	Geomagnetic Shielding
Solar Min	1	28	∗(Table E.15)
Solar Max	1	28	∗(Table E.15)
Worst Week	1	28	∗(Table E.15)
Worst Day	1	28	∗(Table E.15)

Table E.15: Input Parameters to Create Geomagnetic Shielding Files in the CREME96 Package

Apogee (km)	Perigee (km)	Incl. (deg)	Duration (days)	Environment Model	Magnetic Weather	Orbit Section
450	450	51.6^o	365	†	†	Whole
560	560	35.0^o	365	†	†	Whole
800	800	22.0^o	365	†	†	Whole [1]
833	833	98.7^o	365	†	†	Whole
1,200	1,200	65.0^o	365	†	†	Whole
22,200	22,200	55.0^o	365	†	†	Whole
36,000	36,000	0.0^o	365	†	†	Whole

Table E.16: Static SEU Rate Forecast for a Single Xilinx Virtex XCV1000 FPGA in Several Different Orbits

Orbit	Alt. (km)	Incl. (deg)	Solar Minimum (SEU/hr)	Solar Maximum (SEU/hr)	Worst Week (SEU/hr)	Worst Day (SEU/hr)
LEO	450	51.6^o	$2.0x10^{-2}$	$3.5x10^{-3}$	$4.0x10^{-1}$	1.4
LEO	560	35.0^o	$2.8x10^{-2}$	$1.8x10^{-2}$	$1.9x10^{-2}$	$1.9x10^{-2}$
LEO	800	22.0^o	$9.6x10^{-2}$	$6.6x10^{-2}$	$6.7x10^{-2}$	$6.7x10^{-2}$
Polar	833	98.7^o	$6.9x10^{-2}$	$5.5x10^{-2}$	1.1	3.8
Const.	1,200	65.0^o	$2.5x10^{-1}$	$2.0x10^{-1}$	1.0	3.1
GPS	22,200	55.0^o	$4.5x10^{-2}$	$4.6x10^{-1}$	3.7	$1.3x10^{+1}$
GEO	36,000	0.0^o	$4.5x10^{-2}$	$5.5x10^{-1}$	3.7	$1.3x10^{+1}$