



Mitigation Selection and Qualification Recommendations for Xilinx Virtex, Virtex-II, and Virtex-4 Field Programmable Gate Arrays

Gregory Allen
Jet Propulsion Laboratory
Pasadena, CA

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL Publication 09-35 12/09



Mitigation Selection and Qualification Recommendations for Xilinx Virtex, Virtex-II, and Virtex-4 Field Programmable Gate Arrays

NASA Electronic Parts and Packaging (NEPP) Program
Office of Safety and Mission Assurance

Gregory Allen
Jet Propulsion Laboratory
Pasadena, CA

with help from other members of the
Xilinx Radiation Test Consortium

NASA WBS: 724297.40.43
JPL Project Number: 103982
Task Number: 03.04.01

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

<http://nepp.nasa.gov>

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the National Aeronautics and Space Administration Electronic Parts and Packaging (NEPP) Program.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

Copyright 2009. California Institute of Technology. Government sponsorship acknowledged.

Table of Contents

1	<u>INTRODUCTION</u>	1
2	<u>XILINX SRAM-BASED FPGA ARCHITECTURE OVERVIEW</u>	2
3	<u>XILINX SEU SUSCEPTIBILITY</u>	3
3.1	DEVELOPING UPSET RATES	3
3.2	SEFI RATES	4
4	<u>MITIGATION SCHEMES</u>	5
4.1	SEFI MITIGATION	5
4.2	SEU MITIGATION	6
4.2.1	CONFIGURATION ERROR DETECTION	6
4.2.2	CONFIGURATION ERROR CORRECTION	6
4.2.3	FUNCTIONAL ERROR MITIGATION (REDUNDANCY)	7
5	<u>MITIGATION VERIFICATION</u>	9
5.1	FAULT INJECTION	9
5.1.1	FULL RECONFIGURATION FAULT INJECTION	9
5.1.2	PARTIAL RECONFIGURATION FAULT INJECTION	10
5.2	PARTICLE BEAM VERIFICATION	10
5.2.1	SEFI MITIGATION VERIFICATION	10
5.2.2	PARTIAL RECONFIGURATION VERIFICATION	10
5.2.3	FUNCTIONAL MITIGATION VERIFICATION AND SYSTEM ERROR RATE PREDICTION	11
6	<u>MITIGATION AND VERIFICATION SELECTION</u>	15
7	<u>CONCLUSIONS</u>	17
8	<u>REFERENCES</u>	18

1 Introduction

When investigating the effects of space radiation on field programmable gate arrays (FPGA), both total ionizing dose (TID) and single-event effects (SEE) must be considered. In complementary metal oxide semiconductor (CMOS) devices, TID generates electron-hole pairs in the gate insulation layers from the total ionizing energy deposited by photons or particles such as electrons, protons, or heavy ions. This cumulative effect leads to the degradation of electrical parameters at the device, circuit, and system levels. Such degradation can sometimes lead to complete device failure. Conversely, SEEs in digital devices are caused by high-energy particles traveling through a sensitive volume in the semiconductor and leaving an ionized track behind. Such ionization may lead to destructive (e.g., single-event latchup [SEL] or single-event dielectric rupture [SEDR]) or non-destructive events. Non-destructive events may be transient (single-event transients [SET]) or stable events, such as single-event functional interrupt (SEFI), single-event upset (SEU) or multiple-bit upset (MBU). The focus of this document is to provide insight into SEU mitigation selection and verification for space-grade Xilinx static random access memory (SRAM)-based FPGA, as the TID failure levels for such devices are high enough for most missions¹ [1] and destructive events have not yet been observed on Xilinx space-grade parts.

Up front, SRAM-based, reprogrammable FPGA devices provide designers with relatively low-cost and high-speed capability compared to their one-time-programmable (OTP) anti-fuse counterparts. SEU-hardened FPGAs (e.g., OTP anti-fuse devices [2-3]) typically require little to no mitigation efforts by the system designer, as they are mitigated for SEE through process or hardware design. However, the speed and re-configurability of SRAM-based FPGAs come at the cost of the additional need to mitigate against configuration SEU, thereby reducing speed and increasing design complexity. As these devices are used in spacecrafts, system performance now becomes both an engineering verification and radiation reliability problem. Historically, typical FPGA design verification includes simulation, breadboard level test, system tests (engineering models [EM]), and final assembly and in-system test. When designing with SRAM-based FPGAs, not only must the level of functionality be evaluated, but the effectiveness of the mitigation must be quantified as well. While this document does not make in-depth comparisons of OTP and SRAM-based FPGAs (see [4] for an assessment), the intent is that by fully understanding the radiation reliability and engineering tradeoffs of designing for SEU robustness, an educated decision will be made.

¹ While TID failure levels for Xilinx FPGAs are typically high enough for most missions, non-volatile memories that store the FPGA's configuration bit stream need to be selected with care. Many of the current non-volatile memory devices available today have much lower total dose failure levels than Xilinx FPGAs.

2 Xilinx SRAM-based FPGA Architecture Overview

In order to obtain an understanding of SEU mitigation techniques and their corresponding verification methodology, a high-level understanding of the device's architecture is necessary. The majority of a Xilinx FPGA consists of an array of configurable logic blocks (CLBs) surrounded by input/output blocks (IOBs); see Figure 1. Each CLB consists of several slices, and each slice contains look-up tables (LUTs) and flip-flops. The FPGA also contains dedicated memory blocks called block selectRAMs (BRAMs) as well as dedicated functional blocks such as digital clock managers (DCMs), digital signal processing (DSP),² and certain families of devices contain embedded PowerPC processors. A large general routing matrix (GRM) connects the array blocks and dedicated functional blocks. The mapping and connecting of these resources is referred to as the device's configuration. Devices are programmed by loading a configuration bitstream into the device, which is stored in the SRAM. Device re-configurability is implemented by loading a new bitstream into the SRAM at any time through a configuration port. Conceptually, the configuration array in a device can be thought of like a simple memory, and each bit in that memory is accessible through an addressing scheme.

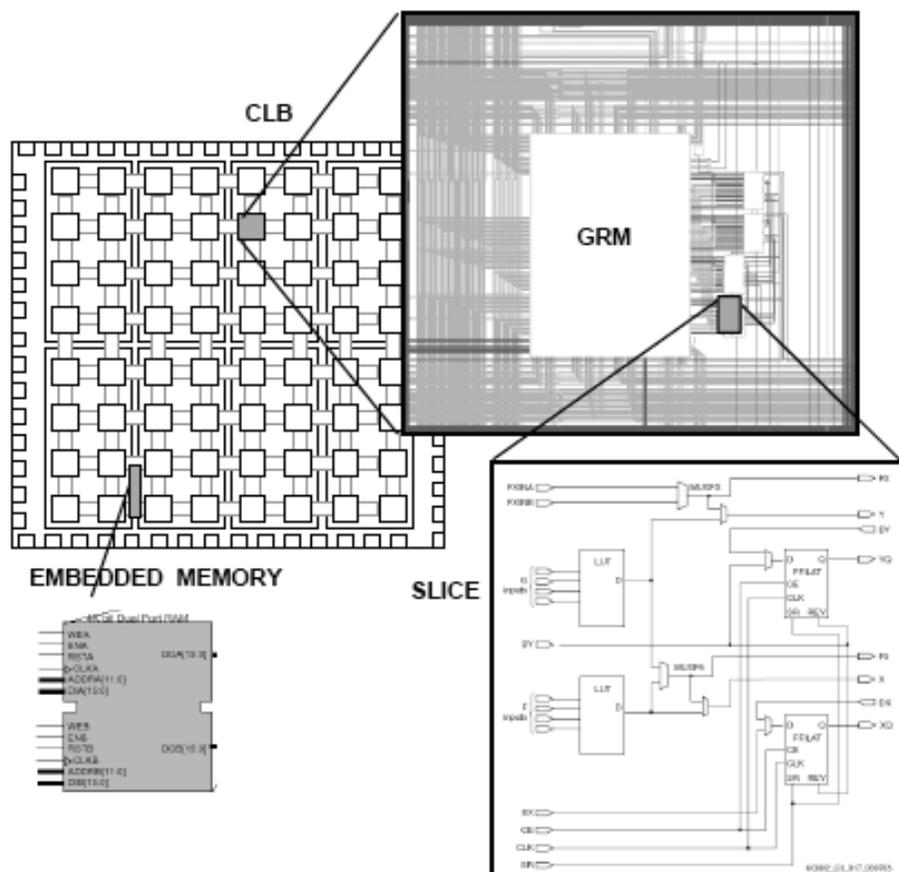


Figure 1. An example of SRAM-based architecture [3].

² DSP are available on Virtex-4 devices only.

3 Xilinx SEU Susceptibility

3.1 Developing Upset Rates

Before appropriate SEU mitigation can be selected, the underlying susceptibility of the device must be understood, and its effect on the design quantified. As SEU of the configuration bits are the dominant upset mode of the device, other effects such as digital SET or SEU of flip-flops are typically ignored. A per-bit cross-section versus effective linear energy transfer (LET) curve is first developed for the configuration cell upset susceptibility from heavy ions, e.g., [6]; see Figure 2. In addition to the heavy ion cross-section plot, proton susceptibility is measured and described with a per-bit cross-section versus proton energy (MeV) plot. From this data, a per-bit SEU rate can be developed for any given space environment.

Once a per-bit cross-section is established, resource utilization must be determined in order to approximate a system level error rate, i.e., not every configuration SEU will cause a given design to fail. To further illustrate the point, 60% to 70% of the configuration bits in a typical device are routing bits. Of those bits, a typical ratio of bits not affecting a design to those affecting a design is between 9:1 and 4:1 [8]. That being said, estimating the number of configuration bit upsets that will affect a design is not as straightforward as using the post place and route “device utilization summary” report from the Xilinx development toolset. While using utilization reports will provide resource use percentages, they don’t include routing bits and will provide an underestimate. However, there are a few other,

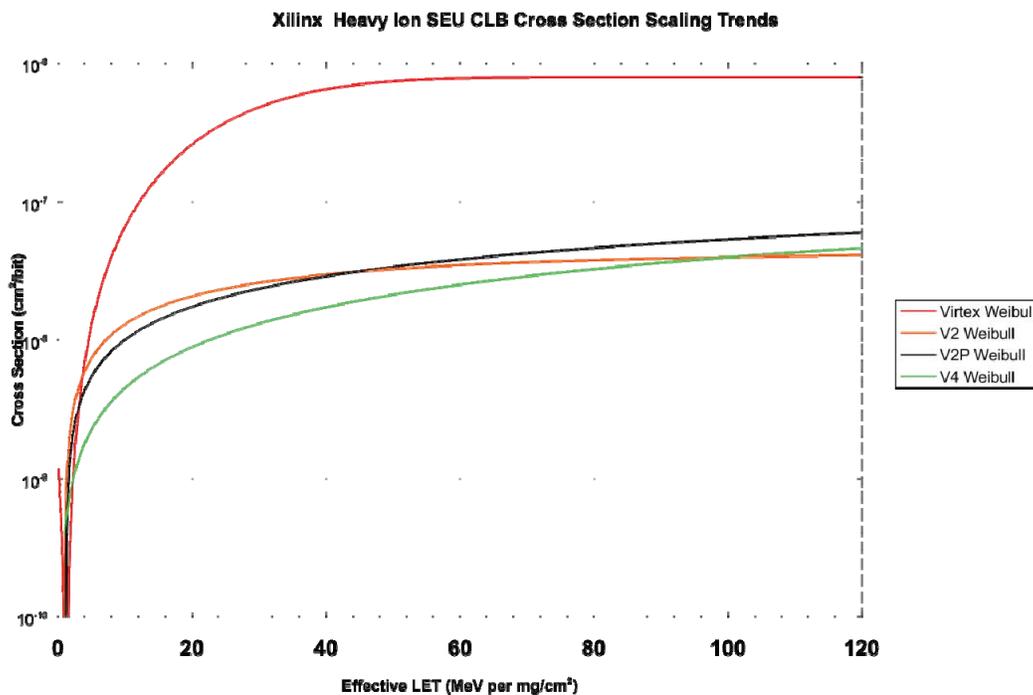


Figure 2. Per-bit cross-sections versus effective LET for four Virtex families of FPGA: Virtex, Virtex-II, Virtex-II Pro, and Virtex-4.

more accurate options. The first of which is an internal Xilinx tool that estimates the number of configuration bits that will affect any given design named COSMIC. Designs can be sent to Xilinx to have the bitstream utilization quantified, or the tool can be requested for use at your institution by contacting the local Xilinx field application engineer. A caveat of this tool is that it can only make estimates for unmitigated applications (designs that have not been triplicated; see Section 4.2.3.1). A secondary and more accurate option is the use of fault injection to estimate the bitstream susceptibility to upset. Fault injection methodology will be discussed later in the document. A third, albeit seldom used, option is two-photon absorption (TPA) laser testing. Finally, the most accurate (and most costly) method by which to quantify FPGA utilization is accelerator testing. By developing a system-error (per device) versus effective LET cross-section, one can accurately quantify the number of configuration bits affecting the design. Yet, because design mitigation is typically an iterative process, already expensive accelerator testing can quickly become cost prohibitive. While there are a handful of ways to estimate the number of bits that will affect a design, the designer and/or mission assurance engineer must understand how that number was quantified and what that means to the overall system rate.

3.2 SEFI Rates

While SEFI detection and mitigation are an important component of configuration management and system mitigation, the SEFI rate cannot be improved through most mitigation techniques. SEFI definitions and rates are found in [6] and [9]. Typically, SEFI rates are relatively low (on the order of once per century per device). Designers and mission assurance engineers need to keep the SEFI rate in mind when designing for an overall system error rate, as that is the lower-bound system error rate for these devices.

4 Mitigation Schemes

The purpose of this section is to provide the reader with high-level methodologies for mitigation schemes, but not design details; see [10-17] for recommended mitigation details. Just as an FPGA can be programmed with an inestimable number of designs, mitigation implementations are nearly as numerous. As previously described, there are two main SEE types to mitigate: SEU and SEFI. Mitigation for each effect should be selected such that they complement each other. Yet, while mitigation schemes for SEFI and SEU overlap, they should be approached separately. SEFI is the simpler of the two and will be discussed first.

4.1 SEFI Mitigation

One cannot prevent the occurrence of a SEFI through device-level design, but can only detect and correct the SEFI. The simplest scheme is to employ watchdog-timer-level detection. Such a system detects a system-level failure, and would either force a power cycle or reconfigure the FPGA to recover. It should be noted that while power cycling the device will recover the FPGA from SEFI modes, it is not required. Pulsing the program status (PROG) pin will always recover the FPGA in Virtex-4, Virtex-II, and Virtex devices. The implementation of this recovery mode is simple in terms of hardware, and therefore typically requires minimal power, real estate, and design complexity. However, it provides little feedback to the overall system concerning the specific SEFI mode. A more detailed level of system feedback described in the following paragraph can be useful when trying to debug on-orbit anomalies. Furthermore, a watchdog-timer-level implementation can take significantly longer to detect a SEFI, or may incorrectly detect a SEFI.

A second SEFI mitigation option is to develop an external configuration management controller. This may be done in another FPGA, ASIC, processor, or other custom hardware. The design will typically include a form of watchdog timer in addition to specific hardware monitoring. For example, a power-on-reset (POR) SEFI, as defined in [6], will cause the 'DONE' pin of the FPGA to de-assert and result in a sudden decrease in core voltage current consumption, high readback error counts, and loss in functionality. A configuration management controller may utilize one or all of the POR signatures to detect and mitigate (where mitigation again includes either power cycle or a forced reconfiguration). Other SEFIs may be detected via the monitoring of internal configuration status and/or control registers. Refer to [6], [9], and [10] for SEFI error modes and signatures. Such a mitigation scheme will scale with the complexity of the mitigation design, but will typically require a second programmable device that is radiation-hardened or otherwise mitigated against radiation-induced faults. While the added visibility into SEFI-induced failures can be helpful when attempting to understand and quickly detect and mitigate on-orbit anomalies, it does come at a penalty of increased power, area, and hardware cost.

Currently, the only known methodology for providing uninterrupted system performance during a SEFI is system-level redundancy; i.e., multiple FPGAs working in parallel with a radiation-hardened voter. Such an implementation comes at a very high cost, and should only be implemented for missions with a very high level of data criticality that also require reconfigurability.

4.2 SEU Mitigation

SEU mitigation is twofold, consisting of static configuration upset mitigation as well as functional mitigation. For missions with significantly long device-operating periods, configuration mitigation, e.g., scrubbing, is necessary to prevent the accumulation of configuration upsets. Without configuration mitigation, the FPGA develops an increased probability of functional error over time. Functional mitigation, i.e., redundancy, is a complicated design-reliability tradeoff that mitigates the FPGA as a system from propagating system errors due to configuration, distributed RAM (LUTRAM), or flip-flop SEUs. It should be noted that LUTRAMs, shift registers (SRL16s), and BRAMs cannot be scrubbed. In Virtex-4 devices, the GLUTMASK_B bit must be set to mask LUTRAMs and SRL16s from being corrupted by readback or partial reconfiguration. In Virtex-II or earlier devices, these primitives need to be removed from the design prior to readback or partial reconfiguration.

4.2.1 Configuration Error Detection

There are several methods by which to detect a configuration upset. In some cases, designers may wish to implement the system in such a way that the configuration is restored without knowing if an SEU occurred. This is often referred to as constant, or blind scrubbing and will be discussed further in Section 4.2.2.2. If error detection is desired, a configuration controller must be implemented that generates a cyclic redundancy check (CRC) calculated from the configuration bitstream. Typically a 16- or 32-bit CRC is calculated at the frame or device level, but can be implemented in other ways. Data is read from the FPGA through either the serial JTAG port or parallel SelectMAP (SMAP) port. The most significant design implications come from deciding on implementing the configuration controller as an internal (inside the FPGA being mitigated) or external (in a second FPGA or ASIC) implementation.

An external, radiation-hardened, configuration controller will typically be the most reliable implementation, albeit it comes at the cost of extra hardware and engineering costs. Conversely, an internal configuration controller will reduce the complexities associated with external hardware, but the controller itself would need to be mitigated against SEU. An internal scrubber can be implemented via routing directly to a configuration interface (JTAG or SMAP), or through the use of the internal configuration access port (ICAP). Note that the ICAP is only available in Virtex-II or later devices. If implementing a completely custom internal configuration controller, it is recommended that an external watchdog timer be used. Some designs may include radiation-hardened tristate buffers on the data IO. Using the ICAP configuration manager reduces routing and external hardware (as the connections are, in this case, made internally), but it cannot be implemented redundantly within a single device. This approach leaves the controller a single point of failure, therefore increasing the system failure rate significantly.

4.2.2 Configuration Error Correction

There are two main methodologies by which to refresh device configuration: full and partial reconfiguration. As discussed in Section 4.2.1, this can be performed at a constant interval, blindly, or upon detection of a fault.

4.2.2.1 Full Reconfiguration

Full reconfiguration is achieved through either power cycling the device or by asserting the PROG pin of the FPGA. While this is probably the simplest implementation of configuration error correction, it is also the most intrusive to the system. During the device configuration, all IOs are tristated until the process is complete. Careful analysis of downstream devices must be performed to ensure they can tolerate such an event.

4.2.2.2 Scrubbing/Partial Reconfiguration

Partial reconfiguration allows the configuration memory to be refreshed without interrupting the functionality of the device. Partial reconfiguration is performed in a similar manner to configuration readback, requiring the same interfaces and, when implemented, external hardware. That is to say, if a designer implements a configuration readback scheme, partial reconfiguration typically is implemented as well. Therefore, the same engineering-reliability tradeoffs must be made in deciding amongst partial reconfiguration schemes.

4.2.3 Functional Error Mitigation (Redundancy)

As previously discussed, static configuration error correction alone will not prevent the possibility of functional or system errors. In order to prevent a functional error, design redundancy must be employed. Redundancy can come in many forms—multiple designs within an FPGA voted upon in that FPGA, single designs in multiple FPGAs voted by a radiation-hardened device, or multiple designs in multiple FPGAs voted by a radiation-hardened device. The tradeoffs between reliability and power, speed, and design time quickly become very complicated. One of the most common single device redundancy schemes is triple modular redundancy (TMR).

4.2.3.1 Triple Modular Redundancy

There are many ways to implement a TMR scheme, but they are all hinged upon the same basic concept: combinatorial and sequential logic is triplicated and voted. The TMR scheme exemplified in this document is Xilinx TMR (XTMR) [11]; however, there are several other design redundancy philosophies available [20-21]. In the XTMR scheme, all combinatorial logic and IO are triplicated, and triplicated majority voters are inserted into every registered loop or feed-back path. If this is executed correctly, there will be no single points of failure in the design, excluding SEFI-inducing circuitry, which cannot be mitigated.

While automated tools exist to generate TMR'ed designs, e.g., XTMR Tool, they certainly are not fool proof. TMR typically increases a design size by approximately 3.5 to 5 times the single string design. This leads to increased power consumption (typically, a 3× increase), reduced ability to meet timing constraints, and more complex board layout if the IO is fully triplicated. Furthermore, single points of failure are not guaranteed to be removed from a design run through a redundancy tool. For example, imagine a case where two separate TMR domains are mapped into two LUTs within the same slice. This is a common situation as the place and route tools are going to put domains as close as possible to each other in order to meet timing constraints. If a configuration upset occurs that changes the function of that slice, both domains will be affected and the TMR mitigation may fail. The point of this example is to illustrate that automated tools rarely will produce a fully mitigated

design without design iteration and careful constraint implementation. This mitigation verification and design verification is an important and necessary step.

4.2.3.2 Redundant Devices

A second option for design redundancy is to use multiple devices. If a TMR'ed design will not fit in a single device, resources are limited (e.g., I/O resources), or the design's data criticality is significantly high (three or more devices can prevent data interruption due to SEFI), redundant devices are a practical option. Such a system increases data reliability at the cost of increased layout complexity, power consumption, and cost. Furthermore, device re-synchronization is fairly difficult to achieve, and may reduce the overall timing capabilities of the application.

5 Mitigation Verification

Mitigation verification can be as convoluted as the mitigation design process. Typically, when a mission assurance radiation effects engineer wants to characterize a device for SEE, he or she will perform a series of cross-section versus LET measurements, as illustrated in Figure 2. From this, the engineer can estimate an error rate in a given space environment of interest. When error detection and correction (EDAC) is introduced into a digital device, such as a memory or FPGA, system error event probability becomes a function of cross-section versus LET, SEU correction rate, and ion flux (number of ions subjected to the device per second per cm²) versus LET [17-19]. Details regarding accelerator testing of EDAC designs is outlined below. While accelerator testing will provide accurate error rates, the testing is often costly to projects. This is especially true if several design iterations are required to acquire the desired system failure rate. An alternative to ion beam testing, or at least a first pass screening methodology, is fault injection. Fault injection allows the designer to corrupt one or many configuration bits in the device under test (DUT) with the same methodology as the configuration controller. Only in this case, instead of repairing upset configuration cells, the object is to flip the cells and check for errors.

5.1 Fault Injection

It should first be noted that fault injection cannot provide exhaustive coverage of error modes in any given design, as it will only simulate upsets to the configuration logic. Complete coverage is only feasible through particle accelerator testing. A second note concerns fault-injection-induced SEFI. Most SEFIs cannot be induced via fault injection, specifically POR and frame address register (FAR) SEFI. However, in Virtex-4, the SMAP port can be corrupted through fault injection, and the fault injection engine should be designed to deal with such an effect, or avoid corruption of the culpable bits.

5.1.1 Full Reconfiguration Fault Injection

There are several methods by which fault injection can be accomplished, each method paralleling the configuration controller approach. The first method is full reconfiguration fault injection. This is a very simple, but manual process whereby the designer hand codes, or rewrites the bitstream manually through a text or graphical user interface (GUI) editor such as FPGA Editor. After the bitstream has been corrupted, the device is reconfigured and checked for functionality. While this tactic is simple to implement, it is not feasible to perform exhaustive verification in a realistic timeframe. However, an effective first go/no-go approach is to break each of the individual clock domains of the triplicated system. If the system is fully triplicated, the design should be completely functional with any one of the three clock domains disabled. If the design is partially triplicated, it should remain functional with either of the two clock domains that are not routed to the single-string portion of the design disabled. Each of the three domains should be disabled, checked for functionality, and re-enabled without reconfiguration to ensure functionality. Furthermore, if the design has a global reset implemented, the same approach should be taken for the three reset domains.

5.1.2 Partial Reconfiguration Fault Injection

Partial reconfiguration fault injection is performed in a similar manner to a configuration readback-scrubbing routine. When this approach is applied, a configuration controller dynamically and partially reconfigures the configuration bitstream through the SMAP or JTAG port. While it is possible to perform dynamic partial reconfiguration through programming tools such as iMPACT, similarly to full reconfiguration, it is a very manual practice, and again, full coverage is impractical to accomplish.

One method to attain full coverage is accomplished through iterative single-bit, partial reconfiguration fault injection. Such a system requires an automated configuration controller and functional monitor that handshake with each other. A state flow for verifying a design is as follows: The configuration controller corrupts the first valid bit of the first valid frame (ignoring header frames) and alerts the functional controller of the completion of the bit flip. At this point, the functional monitor exercises the DUT by executing an exhaustive set of test vectors. If no functional error is found, the functional monitor asserts completion and the configuration controller corrects the first bit while corrupting the second. If a functional error is detected, recovery methods such as system reset, configuration scrub, or full reconfiguration are attempted sequentially and recorded until full functionality is regained. This process is repeated for the entire bitstream.

Depending upon the size of the device and speed of the fault injector, this process may take hours to weeks to complete. However, the resultant number of single points of failure is used to achieve an effective approximation to the system error rate of the device.

5.2 Particle Beam Verification

While fault injection is an excellent approximation of how well a design is mitigated, a particle accelerator test will always provide the most accurate error rate.

5.2.1 SEFI Mitigation Verification

The SEFI rate of a given FPGA is static, and therefore does not need to be fully characterized. However, at least one data point should be taken in the saturation region of the SEFI cross-section versus effective LET or energy curves to show that the designed SEFI detection detects and corrects all SEFI modes, and that the cross-section shows parity with the known SEFI cross-sections. During testing, design functionality must be maintained to ensure all SEFI modes are detected and mitigated. That being said, static configuration correction must be maintained. Because SEFI modes are, to date, design independent, a simple yet robust functional design may be used if the intended mission design cannot withstand the flux required to reach a statistically significant amount of fluence (on the order of 10^7 ions/cm²) typically used to characterize SEFI modes.

5.2.2 Partial Reconfiguration Verification

Verification of the partial reconfiguration controller in the beam is straightforward in nature. If SRL16s, LUTRAMs, or BRAMs are to be used in the flight design, be sure to instantiate them in the DUT design to ensure the scrubber is not corrupting them [13]. The partial configuration controller is shown to be effective by irradiating the device,

scrubbing, and performing a bitstream readback through iMPACT and comparing to a golden (not irradiated) bitstream readback file.

5.2.3 Functional Mitigation Verification and System Error Rate Prediction

5.2.3.1 Single-String Design Error Rate Prediction

Designs without any functional n-modular redundant error mitigation are tested in the usual manner at a particle accelerator. If testing is being performed at a heavy ion facility, the effective LET is varied by changing ion species. For each effective LET, the number of system errors (i.e., per device as opposed to per-bit configuration errors) is counted for a given amount of fluence. The system error cross-section as a function of effective LET is simply the number of errors divided by the total fluence (ions/cm²) subjected to the device. Once this cross section is measured, system error rates are estimated based on environmental models and system operational parameters. Dividing the per-bit upset rate into the system error rate should produce a fairly accurate approximation of the number of configuration bits affecting the design. This number should always be an upper bound to the fault injection number discussed in Section 5.1.

5.2.3.2 Partial and Fully Triplicated Error Rate Prediction

Before discussing how to predict and measure error rates on TMR'ed systems, a set of terminology must be established. As discussed in Section 4.2.3.1, for every feedback path or register, a triplicated set of minority voters is inserted. This grouping of triplicated logic, registers, and voters is referred to as Groups, and the number of them is denoted in the following equations as 'M.' The scrub time, 'TC', is the period by which the device is monitored and scrubbed for SEU. The underlying per-bit upset rate (bit-errors per bit-second), 'r', is a product of two measurement parameters: the per-bit configuration SEU cross-section and the ion flux. A per-bit cross section is calculated based on the effective LET selected by the test engineer and the given static cross-section. See Figure 2. Typically, once a convenient effective LET is selected, only the flux (ions/cm²/sec) is varied to achieve data that is plotted against the theoretical prediction. While either parameter can be varied to achieve various values of r, the flux is generally varied to save time at the accelerator. For each run, the system-error cross-section (per device) is calculated in the usual manner (the number of system errors divided by the total fluence). The system-error cross-section is multiplied by the recorded flux and the system-error rate (system errors/sec), 'R', is the product. The system-error rate is then plotted against the raw bit-flip rate showing an experimental determination of R as a function of r.

The theoretical determination of R as defined in [18] is given by

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

where N_i represents the number of configuration bits in a single triplicated domain of the i^{th} group. The exact equation (1) can be approximated [18] as

$$R \approx \frac{1}{T_C} \left\{ 1 - \exp \left[-3M(\mathcal{M}_2 r T_C)^2 + 5M(\mathcal{M}_3 r T_C)^3 - \frac{37}{4}M(\mathcal{M}_4 r T_C)^4 \right] \right\}. \quad (2)$$

where M_2 , M_3 , and M_4 are three moments instead of the complete set of M moments. These three are defined by

$$\mathcal{M}_2 \equiv \left[\frac{1}{M} \sum_{i=1}^M N_i^2 \right]^{1/2}, \quad \mathcal{M}_3 \equiv \left[\frac{1}{M} \sum_{i=1}^M N_i^3 \right]^{1/3}, \quad \mathcal{M}_4 \equiv \left[\frac{1}{M} \sum_{i=1}^M N_i^4 \right]^{1/4}. \quad (3)$$

In practice, M_2 , M_3 , and M_4 typically have the same values. Approximating the number of bits (N_i) in a triplicated domain is very impractical. The number of groups, ‘ M ,’ is fairly easy to estimate using a tool such as FPGA Editor that can count the number of instantiations given by a name, and T_C is typically well defined by the designer.

The above approximation (Equation 2) was recently augmented [22] to fit data that contained single points of failure, i.e., unmitigated circuitry. The augmented approximation is given as,

$$R' = \frac{1 - e^{-M_U r_U T_C}}{T_C} + e^{-M_U r_U T_C} R \quad (4)$$

where M_U is the total number of unmitigated configuration bits, and R is calculated from (2) for the mitigated bits.

The question may arise, ‘Given the theoretical model, why do I need to accelerator testing?’ For one, without a tool such as fault injection, there is no way to know if the design was properly mitigated, i.e., if there are any single points of failure in the design. As previously stated, it is also highly difficult to quantify N_i in a design. Once accelerator data are acquired, the theoretical model complements and provides more insight into the level of mitigation subjected to the system. The following example is presented to show how the theory can be used to supplement test data.

5.2.3.3 Functional Mitigation Error Rate Example

In the following examples, a Virtex-4QV LX200 was configured with a simple counter design, triplicated using Xilinx’s XTMR tool. One of the designs was fully triplicated, and one of the designs contained unmitigated circuitry (mainly a data sampling circuit). It required a few months of fault injection and design iteration to achieve full mitigation. SEAKR Engineering, Inc. performed most of the design development, with assistance from the Xilinx Radiation Test Consortium. The fully mitigated design utilized 37% of the device’s slices, 29% of its LUTs, and 10% of the flip-flops. The partially mitigated design utilized 41% of the slices, 29% of the LUTs, and 15% of the flip-flops. Fault injection was performed on each design, and after several iterations, it was found that there were no single points of failure in the fully mitigated design. The partially mitigated design exhibited 12 fault injection points that recovered via reset, and 4,016 that recovered via a scrub. For each design, T_C was engineered to be 0.669 seconds. M was estimated to be 11,440 for the fully

mitigated design, and 8,650 for the partially mitigated design through the use of the Xilinx PlanAhead Tool. M_U was assumed to be 4,016 for the partially mitigated design from the fault injection results. Figures 3 and 4 show accelerator data and the theoretical fit for the fully and partially mitigated designs, respectively.

The remaining variables in the theoretical fit, \mathcal{M}_2 , \mathcal{M}_3 , and \mathcal{M}_4 (the three moments of the N_i quantities), are defined by fitting the model to the experimental data. Once an appropriate fit is defined, rates in a given space environment should be estimated by first estimating the expected raw bit-flip rate for that environment. Then, the appropriate fit (e.g., the fit used to produce the curve in Figure 3 for the fully mitigated design) is used to estimate the system error rate. For example, the expected raw bit flip rate for a Virtex-4 device due to galactic cosmic ray (GCR) in solar minimum conditions is 3.2×10^{-12} bit-errors/bit-second. When that is applied to the fully mitigated design, we can expect an error rate of 3.98×10^{-15} system errors/second, or 1.26×10^{-4} system errors/millennium. When applied to the partially mitigated design, we can expect an error rate of 1.29×10^{-8} system-errors/second, or approximately one event every 2.5 years. Furthermore, this failure rate for the partially mitigated design is the same as the expected upset rate for 4,016 configuration bits, the number of single points of failure in the design (i.e., 3.2×10^{-12} bit-errors/bit-second times 4,016 bits equates to 1.29×10^{-8} bit-errors/second). What this shows us is that if properly implemented, fault injection can be a very accurate tool to predict system error rates. However, the accelerator will always produce the most exhaustive verification. Furthermore, the model can also be used to predict system error rates from high-flux events such as solar flares.

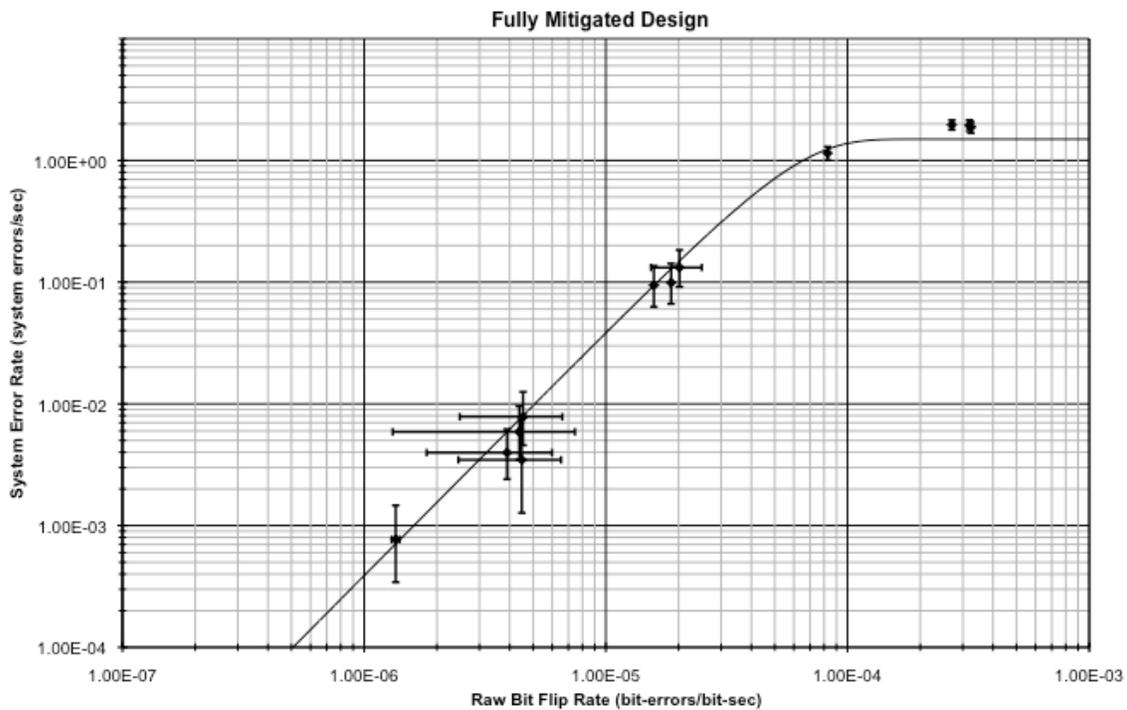


Figure 3. Experimental data shown with the theoretical model (2) for R versus r of the fully mitigated design. Vertical error bars represent the Poisson-distributed statistical error in counting system error events. The horizontal error bars represent recorded fluctuations in the ion beam flux when recorded (instantaneous flux was not recorded for every run).

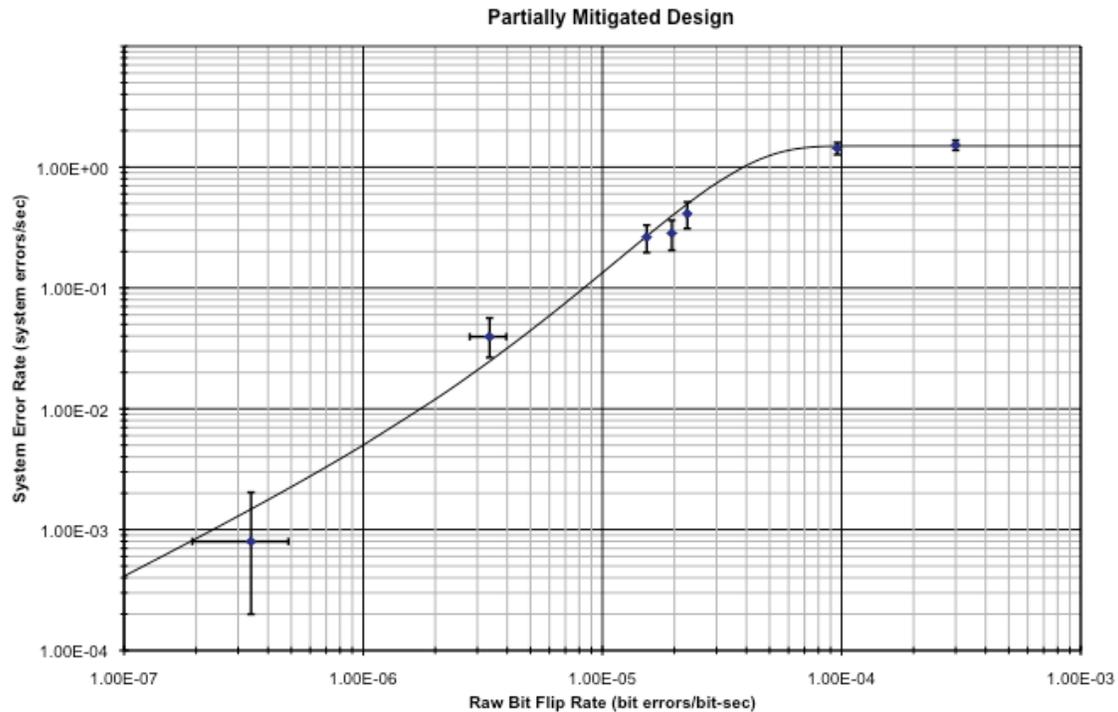


Figure 4. Experimental data shown with the theoretical model (3) for R' versus r of the partially mitigated design. Vertical error bars represent the Poisson-distributed statistical error in counting system error events. The horizontal error bars represent recorded fluctuations in the ion beam flux when recorded (instantaneous flux was not recorded for every run).

Other predictions can also be made. Once accelerator data is acquired to verify accuracy of the R-model, the design engineers can then alter scrub times to see how that will affect the failure rate. For example, instead of scrubbing the fully mitigated design at $TC=0.669$ seconds, we implement a scheme where the configuration manager blindly reconfigures the part once per day. The predicted system error rate then becomes 5.13×10^{-10} , or a little more than 1.6 system errors per century (on the order of the device SEFI rate). It should be noted that if the same reconfiguration methodology is applied to the partially mitigated design, it does not affect the failure rate, as the single points of failure dominate the system error rate. However, the scrub time will affect the recovery rate, i.e., unless the configuration manager is alerted to the functional failure, the system will not recover until the blind reconfiguration is executed.

6 Mitigation and Verification Selection

This section provides a series of questions in order for the reader to gain a sense of the mitigation and verification selection process.

- What is the underlying, unmitigated, system error rate?
 - The question behind this question is: How many configuration bits affect the design? As discussed throughout the paper, this can be determined several ways: through fault injection, accelerator testing, or software estimation. Fault injection and accelerator testing are the most accurate methods by which to acquire the error rate, but can be costly in terms of development time and beam charges, respectively. If software estimates are implemented, care must be taken when interpreting the results. Once the number of bits that will cause a system error is determined and space environment is selected, the system error rate is calculated. Then the question becomes:
- What is the probability of observing a system error?
 - The probability of observing a system error is a Poisson-distributed process of error rate and operating period. For example, if our design contains 10 million configuration bits that cause system errors, but the application only operates for 5 minutes a day in deep space (GCR environment, in which the raw bit flip rate was estimated in the previous section to be 3.2×10^{-12} bit-errors/bit-second), there is a 0.96% probability of a system error occurring during each operational period (assuming the device is not accumulating upsets between operating periods). However, if the same design operates for 6 hours a day, there is a 50% probability of a system error. Each of the applications mitigation will be approached very differently.
- What level of mitigation is going to be required?
 - Before this question is answered, several quantities need to be defined:
 - What are the performance requirements of the application?
 - Introducing redundancy to systems will reduce timing performance.
 - What are the power and area constraints of the system?
 - Increasing redundancy, whether through multiple devices or design triplication increases power consumption and real-estate demands.
 - How cost limited is the application?
 - The more mitigation required, the more engineering time required.
 - What is the system error persistence tolerance?
 - How fast does the system need to recover?
 - Can it stand the SEFI rate? If not, three redundant devices and a radiation-hardened voter may be required.
 - What level of tolerance does the system application have to system-errors?
- What level of configuration correction is needed, if any, and how often does it need to occur? What level of redundancy needs to be implemented, if any?
 - The answer to these questions is function of the probability of a system-error and the system's tolerance for error persistence. The higher the level of the

application's data criticality, the less tolerance there is for error persistence in the mitigated design. This can be solved in two ways: either the probability of a system error needs to be reduced (redundancy), or the rate of repair needs to be increased (or potentially using error detection and correction as opposed to periodic correction). However, increasing one or the other of these mitigation techniques will only improve the robustness of the design so much. Eventually, the two mitigation techniques will need to be combined to achieve a higher level of system-error robustness if needed.

- The mitigation scheme has been selected, how should it be verified?
 - Ideally, this is an iterative process that consists of as much bench top testing as possible (fault injection and software simulation), followed by an accelerator test. Only the level of data or application criticality can determine the extent of the testing. But at a minimum for most missions, fault injection should be performed to accurately estimate the error rate.

7 Conclusions

The use of OTP antifuse FPGAs, particularly radiation-hardened Actel FPGAs, is fairly well understood from both a cost and mission assurance perspective, and as such was not explicitly discussed in this document. The design process is well known, and the system-error rate calculation for OTPs is much simpler as compared to SRAM-based FPGAs as the mitigation is built-in to the devices. However, SRAM-based FPGAs appeal to designers mainly due to their reconfigurability and higher speed potential. Yet, this higher processing power comes at the cost of custom mitigation schemes, potentially reducing speed, and increasing project cost through the need to verify the mitigation process.

The overall cost of SRAM-based FPGA implementation is highly application and program dependent. While through various levels of mitigation, most missions can lower system-error rates to acceptable levels, there will always be a radiation-reliability to performance and cost tradeoff. Such a tradeoff can only be quantified through implementation, on an application-by-application basis. Each type of FPGA has advantages and disadvantages, and it is important to understand the full design cycle, levels of verification, and corresponding costs for each FPGA technology before deciding upon its use.

8 References

- [1] "Radiation Tolerant Virtex-4 QPro Family Overview," December 16, 2008.
http://www.xilinx.com/support/documentation/data_sheets/ds653.pdf
- [2] Actel Datasheet: "RTAX-S/SL RadTolerant FPGAs."
http://www.actel.com/documents/RTAXS_DS.pdf
- [3] Actel Datasheet: "RTSX-SU RadTolerant FPGAs (UMC)."
http://www.actel.com/documents/RTSXSU_DS.pdf
- [4] M. Berg and K. LaBel, "Determining the Best-Fit FPGA for a Space Mission: An Analysis of Cost, SEU Sensitivity, and Reliability." MRQW, 2007.
<http://www.aero.org/conferences/mrqw/2007-papers/VI-2.pdf>
- [5] P. Adell, and G. Allen, "Assessing and Mitigating Radiation Effects in Xilinx FPGAs,"
<http://parts/docs/NEPP07/NEPP07FPGARadiationEffectsGuideline.pdf>
- [6] G. R. Allen, G. Swift, and C. Carmichael, "Virtex-4QV Static SEU Characterization Summary," <http://parts.jpl.nasa.gov/docs/NEPP07/NEPP07FPGA4Static.pdf>
- [7] G. R. Allen, "Virtex-4QV Dynamic and Mitigated Single Event Upset Characterization Summary," <http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/41104/1/09-04.pdf>
- [8] Private communications with Chen Wei Tseng of Xilinx. September 3, 2009.
- [9] G. Swift, "Xilinx Single Event Effects 1st Consortium Report: Virtex-II Static SEU Characterization," January 2004. http://parts/docs/swift/virtex2_0104.pdf.
- [10] G. Miller, C. Carmichael, and G. Swift, "Single-Event Upset Mitigation for Xilinx FPGA Block Memories,"
http://www.xilinx.com/support/documentation/application_notes/xapp962.pdf
- [11] C. Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs,"
http://www.xilinx.com/support/documentation/application_notes/xapp197.pdf
- [12] B. Bridgford, C. Carmichael, C. Tseng, "Single-Event Upset Mitigation Selection Guide,"
http://www.xilinx.com/support/documentation/application_notes/xapp987.pdf
- [13] C. Carmichael, and C. Tseng, "Correcting Single-Event Upsets in Virtex-4 Platform FPGA Configuration Memory,"
http://www.xilinx.com/support/documentation/application_notes/xapp988.pdf
- [14] C. Carmichael, and C. Tseng, "Correcting Single-Event Upsets with a Self-Hosting Configuration Management Core,"
http://www.xilinx.com/support/documentation/application_notes/xapp989.pdf
- [15] G. Miller, C. Carmichael, and G. Swift, "Single-Event Upset Mitigation Design Flow for Xilinx FPGA PowerPC Systems,"
http://www.xilinx.com/support/documentation/application_notes/xapp1004.pdf
- [16] M. Berg et al. "Effectiveness of Internal vs. External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis." IEEE Transactions on Nuclear Science, Vol. 55, Issue 4, Part 1, December 2007.
- [17] J. A. Zoutendyk et al., "Single-Event Upset (SEU) in a DRAM with On-Chip Error Correction," IEEE Transactions on Nuclear Science, Vol. NS-34, No. 6, December 1987.

- [18] L. Edmonds "Analysis of Single-Event Upset Rates in Triple Modular Redundancy Devices," Internal JPL Memo, October 2005.
- [19] D. G. Mavis et al., "Multiple Bit Upsets and Error Mitigation in Ultra-Deep Submicron SRAMS," IEEE Transactions on Nuclear Science, Vol. 55, No. 6, December 2008.
- [20] K. Morgan et al., "Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs." IEEE Transactions on Nuclear Science, Vol. 54, Issue 6, Part 1, December 2007.
- [21] M. Berg, "Design for Radiation Effects", MAPLD 2008.
http://nepp.nasa.gov/MAPLD_2008/presentations/i/01%20-%20Berg_Melanie_mapld08_pres_1.pdf
- [22] L. Edmonds, "System Error Rates for Devices Having Partial TMR Protection," August 2009, Internal JPL Memo.