Title:

A Test Methodology for Determining Space-Readiness of Xilinx SRAM-based FPGA Devices and Designs

Author(s):

Heather Quinn, Paul Graham, Keith Morgan, Michael Caffrey, and Jim Krone
LANL

Michael Wirthlin and Brian Pratt
Brigham Young University

Intended for:

IEEE Transactions on Instrumentation and Measurement

A Test Methodology for Determining Space-Readiness of Xilinx SRAM-based FPGA Devices and Designs

Heather Quinn, Paul Graham, Michael Wirthlin, Brian Pratt, Keith Morgan, Michael Caffrey, and Jim Krone

Abstract

Using reconfigurable, static random-access memory (SRAM) based field-programmable gate arrays (FPGAs) for space-based computation has been an very active area of research for the past decade. Since these devices are commercially-available, radiation-tolerant devices, the device must be qualified for spacecraft usage. Furthermore, mission requirements often dictate the need to do radiation experiments on the FPGA user circuit. Because both the circuit and the circuit's state are stored in radiation-tolerant memory, both could be altered by the harsh space radiation environment. Both the circuit and the circuit's state can be protected by triple-modular redundancy (TMR), but applying TMR to FPGA user designs is often an error-prone process. Faulty application of TMR could cause the FPGA user circuit to output incorrect data. This paper will describe both device-level static testing and user circuit dynamic testing, including a three-tiered methodology for testing FPGA user designs for space-readiness.

Index Terms

Field programmable gate arrays, Reliability testing, Reliability estimation, Failure analysis, Space technology

I. Introduction

Field-programmable gate array (FPGA) technology, such as the Xilinx Virtex family of devices, has made inroads into space-based platforms over the past decade [1], [2]. These devices have programmable logic and routing that are used to implement user circuits and are well-suited for the digital signal processing algorithms that are often used in space. Unlike radiation-hardened anti-fuse FPGAs that can only be programmed once, radiation-tolerant devices can be programmed an unlimited number of times. The ability to *reconfigure* the device to implement new circuits makes SRAM FPGAs interesting to the space community. Unlike other hardware devices that have the circuit fabricated into the silicon, new circuits can be implemented on an FPGA while on orbit. Therefore, reconfiguration can extend the usable lifetime of the system by changing the FPGA's user circuit to meet changing mission and science goals. We have also found that reconfiguration opens up many avenues for pre-launch testing of the user circuits.

Unfortunately, the SRAM technology used in these FPGAs to implement the user circuit is susceptible to radiation-induced faults, called single-event upsets (SEUs), that can affect the programmable logic and routing or affect the entire device. To adequately qualify an SRAM-based FPGA for space, the device is tested both statically and dynamically in a cyclotron so that radiation-induced faults can be identified, characterized and quantified. Static radiation tests involve testing the entire device to determine the sensitivity of the device to radiation-induced faults, which is called the *cross-section*. As the sensitivity of the device is affected by the source, energy and incident angle of the ionizing particles, static testing tends to be a lengthy process. In many cases, static testing often indicates that the device is not being clocked, but in our case static testing indicates that we are not concerned with either input test vectors or output errors. Dynamic testing measures the sensitivity of a FPGA design to radiation-induced faults and uses both input and output test vectors to test the design's output data for errors. While there have been occasions where dynamic testing has revealed problems that did not present in static tests, most times the dynamic testing of SRAM-based FPGA user circuits indicates that the design is less sensitive than the device's static cross-section.

In this paper we will present methodologies for both static and dynamic testing. In Section II we provide information that affects both static and dynamic testing that needs to be factored into testing. In Section III we will present a methodology for testing SRAM-based FPGA devices statically that will allow the tester to determine the sensitivity of the device to radiation-induced upsets. In Section IV we will present a three-tiered methodology that uses all of these technologies for discovering design flaws in the system before launch. In both Section III and IV we will present results from using these methodologies on Xilinx Virtex family devices. Given the disparate nature of these topics, the related work for these topics will be covered in the individual sections.

Document release number: LA-UR-08-XXXX. This work was funded by the Department of Energy through the Deployable Adaptive Processing Systems and Sensor-Oriented Processing and Networking projects, the Cibola Flight Experiment project at Los Alamos National Laboratory, NASA through the Reconfigurable Hardware in Orbit project under LIST contract #NAG5-13516, and the Air Force Research Laboratory under the FPGA Mission Assurance Center.

H. Quinn, P. Graham, K. Morgan, M. Caffrey, and J. Krone are with ISR-3 Space Data Systems, Los Alamos National Laboratory, Los Alamos, NM, 87545 USA

M. Wirthlin and B. Pratt are with Brigham Young University, Provo, UT, 84602 USA

II. Background on Radiation Testing

All electronic devices that will be used in spacecrafts need to be qualified for the space environment, which includes radiation, thermal, and mechanical testing. There are a variety of references and standards for qualifying devices for space usage. The papers [3], [4] will be useful for a discussion of thermal design and testing. For radiation testing, readers will find the "Handbook of Radiation Effects" [5], EIA/JEDEC standards 57 and 89 helpful. Given the scope of the paper we will focus only on radiation testing. The space environment has a very rich radiation environment of electrons, protons and heavy ions. Each orbit is characterized by an ion spectrum, where each ion has a corresponding energy spectrum. Unfortunately, all commercial SRAM devices are affected by these radiation environments, and SRAM-based FPGAs are no exception. Radiation-induced faults are particularly difficult with FPGAs since both the circuit and the circuit's state are stored in radiation-tolerant SRAM.

Radiation testing covers many different phenomena, including dose rate effects and single-event effects (SEE). To date, no dose-related effects have been shown to affect the Xilinx SRAM-based FPGAs and this topic will not be covered in this paper. SEEs is an umbrella term that covers several different types of radiation effects caused by proton and heavy ion radiation. Most commonly, spacecraft designers are concerned with single-event latchup (SEL), single-event transients (SETs), single-event upsets (SEUs), and single-event functional interrupts (SEFIs). While there are a handful of SEE types that can damage a device, SEL is the predominant concern in this category. The remaining three SEE mechanisms discussed in this paper are not destructive, but can make fault-tolerant computation challenging. These phenomena are discussed below.

Radiation-induced faults from SEUs, which are also known as *bitflips* or *upsets*, cause memory bits to change value from either 0 to 1 or 1 to 0. SEUs are the primary concern for SRAM-based FPGAs in space. SEUs in FPGAs have been shown to cause problems in the programmable logic, the programmable routing, and even device control [6], [7]. SEUs that affect the device control are considered single-event functional interrupts (SEFIs). SEFIs on orbit can have serious consequences, as the device usually needs to be completely reprogrammed and the calculation restarted to recover from many SEFIs. In practice, as discussed in Section III.D, SEFI error rates are very low.

Due to the destructive effects of SEL or *latchup*, spacecraft designers often disqualify devices that latchup from spacecrafts. SEL is a radiation-induced version of latchup that plagues complementary metal-oxide-semiconductor (CMOS) devices and can be destructive to SEL-sensitive devices. Unlike most SRAM-based FPGAs, Xilinx has published several reports verifying latchup-immunity [8], [9], which have made them the preferred choice for space usage.

In comparison, SETs (or *transients*) are less troublesome. Transients are common in many semiconductor circuits. With this phenomena the ionizing particle causes a transient current state. If this transient state can propagate to a register during the setup and hold time (called the *window of vulnerability*), the transient will be latched (called a *latched SET*) and the intermediate data value could be corrupted. For modern CMOS devices with fast clock speeds, latched SETs have become increasingly more common and distinguishing transients from legitimate signals is challenging. Unlike SEUs, latched SETs have a radiation-induced error rate that is dependent on the circuit's operating speed as faster clock speeds are more likely to latch SETs than slower clock speeds. For SRAM-based FPGAs, where the user flip-flops are outnumbered by several orders of magnitude by the configuration memory, the current understanding is that SETs are possible, but observability of SETs is hindered by the sheer number of SEUs in the configuration memory.

There are also a number of compound reliability effects that could affect the sensitivity of the device to radiation, including temperature and voltage. In particular, lowering voltages has been shown to increase the device's dynamic or static cross-section [10], [11]. Temperature-related reliability problems can easily be misinterpreted as radiation-induced faults as well, which can further increase the device's dynamic cross-section. We test all devices at nominal temperature and voltages so that we have a consistent basis of comparison over the different devices.

III. Static Testing

In this section we describe how we have done static testing of the Xilinx Virtex devices. This section covers our methodology toward testing, our analysis methods, and highlights of our results. This section concludes with a short discussion of how these results are used in determining the error rate for these devices in space.

III.A Methodology

To achieve our goals of understanding the frequency and effects of radiation-induced faults, we must first identify the proton and heavy ion static saturation cross-sections. The basic methodology for static radiation experiments of electronic devices involves irradiating the device and counting the number of radiation-induced faults. The amount of ions that irradiate the device and the number of events counted are used to calculate the cross-section using this equation:

$$\sigma_{device} = \frac{events}{fluence \times cos(\theta)} \tag{1}$$

where fluence is the measure of the number of ions that irradiated the device in a set a time, θ is the angle of the test fixture to the beam, and events is the count of the radiation-induced faults.

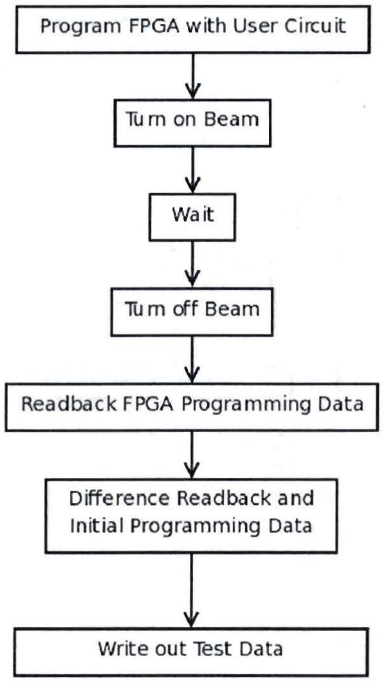

Program FPGA with User Circuit
Turn on Beam
Wait
Turn off Beam
Readback FPGA Programming Data
Difference Readback and Initial Programming Data
Write out Test Data

Fig. 1. Test Methodology for Complete Capture Test System

III.A.1 Test Fixtures

For static testing of SRAM-based FPGA, the basic methodology is often implemented in one of two forms: complete capture systems, and continuous capture systems. In a complete capture system, the test follows the methodology shown in Figure 1. Often times the hardware test fixture uses commercially-available development boards from Xilinx and the software test fixture uses a Xilinx design flow tool, called iMPACT [12]. The advantage of a complete capture system is that each radiation experiment aligns with one run of the accelerator. Therefore, determining the amount of fluence per radiation experiment is simply the entire fluence that the beam control software has measured for that run. The disadvantage is that starting and stopping both the beam and the test fixture can be time-consuming. As iMPACT uses the slowest (JTAG) configuration port for configuring and reading back the programming data, there is a significant overhead to using this software.

The methodology for continuous capture systems is shown in Figure 2. While this methodology removes the overhead of starting and stopping the beam, a custom hardware and software test fixture is often needed. There is a strong advantage in creating a custom test fixture, because the designer can choose to use the faster (SelectMAP) configuration port for configuring and reading back the programming data. While the authors of this paper use a continuous capture test fixture, the Xilinx Radiation Test Consortium (XRTC) use one as well [9].

Our test fixture is a combination of commercial hardware and custom software. One hardware test fixture we use, shown in Fig. 3, is two Xilinx AFX series development boards (one Virtex-II and one Virtex-5) biased nominally. The Virtex-II board communicates to the host computer through a USB card and controls the Virtex-5 board during irradiation. Custom software performs a *readback* of the programming data (or *bitstream*), differences the reference bitstream and the readback data, and immediately reconfigures the device with the reference bitstream. The FPGA is completely reprogrammed and the differential bitstream saved to hard drive every second in this scheme, which allows us to test at high fluences without accumulating too many upsets per readback. We can collect approximately 3,600 differential readbacks per hour. Minimal statistics are taken in real time while irradiating the part as a state of health check.

Th test fixture show in Figure 3 was used to statically test the Virtex-5 device. A similar test fixture was built for the Virtex-4 devices by using the Xilinx AFX series development boards for the Virtex-4. The Virtex-II device can be tested by using only the one Virtex-II AFX development board.

There is a slight disadvantage for this test fixture in determining the overall fluence for a radiation experiment, though, since each loop through the algorithm is an independent radiation experiment. It's possible that during readback that the programming data that has already been read could be upset. Likewise, during configuration it's possible that the part of the programming data that has not been configured could be upset. In both of these cases, the upsets collected in these areas will not be recorded either by not being read in the first case or being overwritten in the second case. These unrecorded upsets means that not all of the ions were used and that the measurement of the fluence for the entire run could be high. We have found, though, that the amount of fluence actually used can be determined using the power logs from the hardware test fixture. As configuring and reading back the device cause spikes in the amount of current the device draws, we can approximately determine the amount

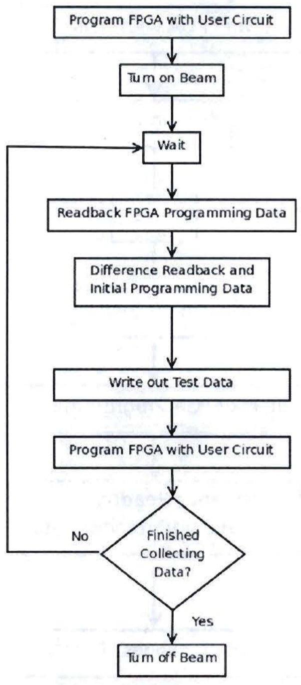

Program FPGA with User Circuit
Turn on Beam
Wait
Readback FPGA Programming Data
Difference Readback and Initial Programming Data
Write out Test Data
Program FPGA with User Circuit
No
Finished Collecting Data?
Yes
Turn off Beam

Fig. 2. Test Methodology for Continuous Capture Test System

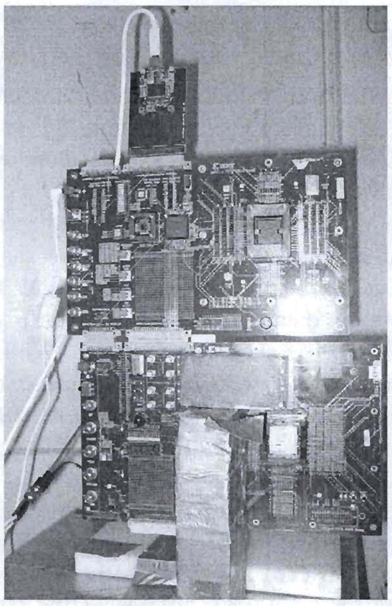


Fig. 3. Hardware Test Fixture for the Xilinx Virtex-5 Device

of used fluence as 1/2 of the configuration time, 1/2 of the readback time, and the wait time. Since we are concerned about keeping the number of events per readback (or *sample*) low, often times the wait time is insignificant. Using this calculation, in the older, smaller devices we used approximately 1/2 of the fluence and in the newer, larger devices we use approximately 1/8 of the fluence. While this might seem wasteful, we are able to collect data much faster with this methodology, and the overall amount of time needed to test is greatly reduced by being able to test at significantly higher fluxes.

III.A.2 Angular Testing

Often times, testers will angle the text fixture to the beam during testing. For older electronic devices, angular testing was done as a way to artificially increase the beam energy. Paradoxically, as ions move through Silicon the energy of the ion will increase until the stopping range is met at the Bragg's peak. The deposited energy beyond the Bragg's peak of the ion is zero. Therefore, one way to increase the ion's energy is to increase the ion's path through the device, which can easily be done by angling the device to the beam. With angular testing, the ions' paths between the surface of the device and the active volume of the semiconductor will be:

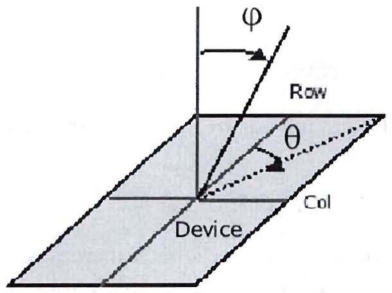


Fig. 4. θ and ϕ Angles Relative to Device

$$distance_{effective} = distance_{normal} \times cos(\theta) \tag{2}$$

where $distance_{normal}$ is the length between the surface of the device and the active volume of the device, and θ is the angle of the test fixture to the beam. This increase in energy is:

$$Energy_{effective} = Energy_{normal} \times cos(\theta) \tag{3}$$

where $Energy_{normal}$ is the energy of the beam at a normal incidence and θ is the angle of the test fixture to the beam. Because the beam's available ions and energies dictate the number of data points that can be taken, angular testing is used as a way to increase the number of data points. Care needs to be taken with angular testing to make certain that ions are not shadowed by the device's socket or that the $distance_{angle}$ does not exceed the ion's range. In both cases, the data will not be consistent with the rest of the data set.

We have found with Xilinx SRAM-based FPGAs that rotating the device in the beam provides an interesting data set. Unlike traditional electronics devices, these devices are laid out heterogeneously with many different types of memory cells. Therefore, rotating the device creates an effect that cannot be explained by only the increase in effective energy. To study these effects, we rotated the device in two different directions, as shown in Fig. 4, to change the beam's angle of incidence. Since the device is columnar in nature, we tested the response when the columns were upright ($\theta = 0$) and when the columns were on their side ($\theta = 90$). Next the beam's angle of incidence (ϕ) was changed by slanting the device relative to the beam. Several different ϕ's were tested for both θ's, to get an idea of how the angular effects changed the radiation characteristics.

III.A.3 Multiple-bit Upset Testing

We are particularly interested in the role of multiple-bit upsets (MBUs) in these devices. MBUs are caused when a single ionizing particle causes multiple bits to change their values. For TMR-protected designs, MBUs often violate the assumption that only one error exists in the system at a time and have been proven to cause TMR defeats in the Virtex-II [13]. Therefore, we often analyze our static data for MBUs.

We have found that it is possible to create MBUs from coincident single-bit upsets (SBUs) if the fluence per sample is too high. As coincident SBUs are indistinguishable from MBUs, limiting their ability to contaminate a data set is necessary for MBU analysis. To determine the quality of our data collection procedures, we have looked at three ways to determine the rate of coincident SBUs in the data set: shape analysis, statistical analysis, and monte carlo analysis. The first attempt at determining and potentially removing coincident SBUs from our data sets focused on analyzing the MBU shapes under the assumption that coincident SBUs were more likely to create "irregularly" shaped MBUs. Analysis of data sets with a low likelihood of having coincident SBUs showed that some of the "irregularly" shaped MBUs were, in fact, common. Therefore, we have not found it possible to remove coincident SBUs from data sets by shape.

To this end, we try to bound the amount of coincident SBUs through statistical and monte carlo analysis. Rigorous statistical analysis of coincident SBUs is difficult given the prevalence of naturally created MBUs and the complexity of the problem. As a worst case analysis, we assume that all of the upsets in a readback are SBUs. While each SBU has an adjacency neighborhood of eight bits, as shown in Figure 9(a), a 2-bit MBU has an adjacency neighborhood of 10-12 bits depending on the shape. Despite the fact that a 2-bit MBU has a larger adjacency neighborhood than a SBU, two SBUs have more adjacent bits total than a 2-bit MBU. Therefore, as a worst case analysis, non-coincident SBUs are used. If the first ion upsets location L_0, the probability that the second ion upsets a bit in the adjacency neighborhood of L_0 is

$$P(CS_1|U_{L_0}) = \frac{8}{N}\frac{1}{N} \tag{4}$$

As there are N combinations of L_0 on an N-bit device, the probability of a coincident SBU on the second upset is:

$$P(CS_1) \quad = N(\tfrac{8}{N}\tfrac{1}{N}) \quad = \frac{8}{N} \tag{5}$$

Similar reasoning is possible until the general equation for m upsets is arrived upon:

$$P(CS_m) \quad = \tfrac{8m}{N} \tag{6}$$

Clearly, from this equation, as the number of upsets m per readback increases, the percentage of coincident SBU increases.

We have tested these equations through the use of monte carlo simulations. For these simulations, we generate m random upsets for the bitstream, then analyze the random upsets using our analysis software described in Section III.B.2 to determine whether co-incident SBUs have created MBUs. We generate and analyze millions of samples in this manner for each device. These simulations have shown that the worst case statistical analysis was approximately a factor of two higher, but still quite reasonable.

III.A.4 Micro-SEFIs

One final methodology concern is a not-completely understood phenomenon that affects the Xilinx Virtex Family parts, which we call the *micro-SEFI* for lack of better terminology. What appears to be happening in the micro-SEFI is that some number of programming bits are locally reconfigured as if the configuration control logic has been upset. The bits that are upset present in the analysis as unusually large MBUs and can wreck havoc on analysis, which will be discussed in greater detail in the next section. Since this phenomena is certainly not an actual MBU and likely not caused by an SEU in the programming data, we eliminate these samples from the data set. To simplify data cleansing, we test either with many more or many less upsets per readback than present in the micro-SEFI so that the micro-SEFI-contaminated data samples can be easily eliminated. For example, if the micro-SEFI presents with approximately 300 upsets per sample, then we test at either greater than 500 upsets per sample or less than 100 upset per readback. To minimize problems with both the micro-SEFIs and coincident SBUs, we recommend testing at less than 100 upsets per readback.

III.B Analysis

The analysis of our data sets takes three distinct phases. First of all, data sets must be cleansed from micro-SEFIs and SEFIs. Second, analysis of the data correlates upsets to physical locations to determine the percentage of MBUs and distribution of upsets by memory cell type. Finally, all of the data sets are combined to create plots of the sensitivity by energy.

III.B.1 Data Cleansing

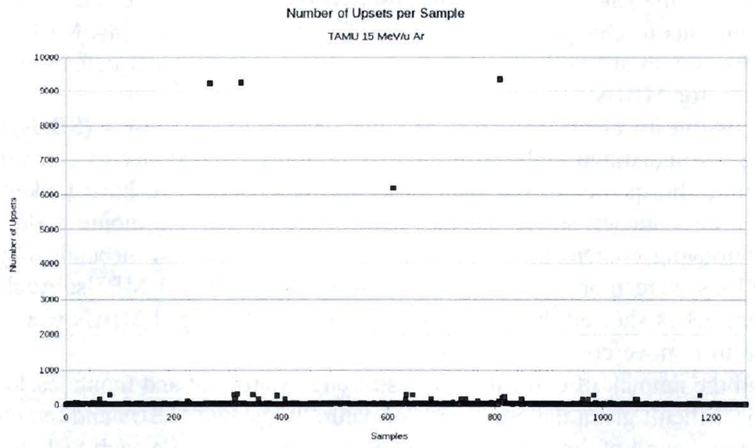


Fig. 5. Entire Set of Samples for One Run on the TAMU 15 MeV/u Argon Beam

Once the data was collected, the data needs to be cleansed to remove both the micro-SEFI-contaminated data samples and the SEFI-contaminated data sets. An entire run of data taken on our continuous capture test system for the Xilinx Virtex-5 device at Texas A&M University's cyclotron for 15 MeV/u Argon is shown in Figure 5. Eliminating SEFI-contaminated data samples is simple, because the number of upsets is greater than 9,000 upsets for a sample. In Figure 6, the SEFIs have been removed from the data set, but the micro-SEFIs remain in the data set. In this figure, it is fairly easy to see the micro-SEFI

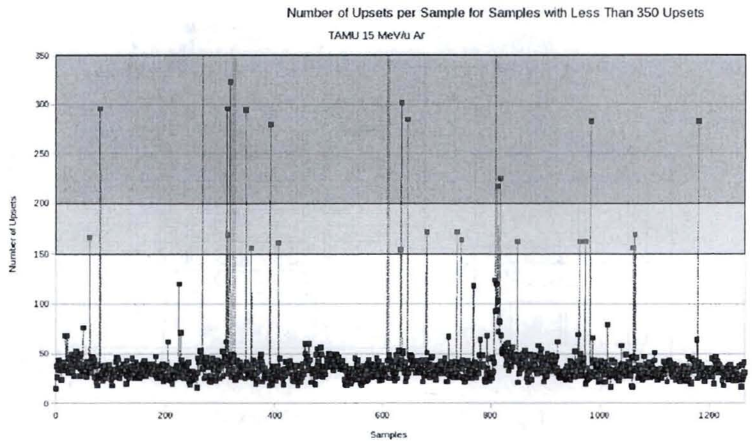


Fig. 6. Highlighting Jackknifing possibilities in the Set of Samples for One Run on the TAMU 15 MeV/u Argon Beam

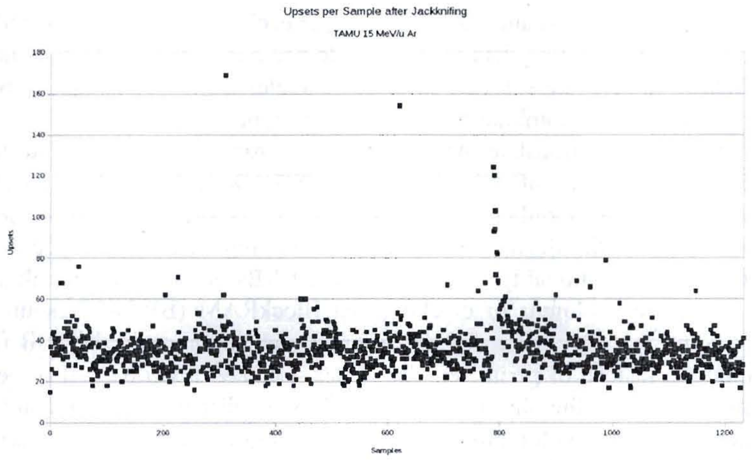


Fig. 7. Set of Samples for One Run on the TAMU 15 MeV/u Argon Beam After Jackknifing

data points as they have far more upsets per sample than the rest of the samples. In Figure 6 these data points are boxed in dark gray. The data in the lighter gray box are suspect data that could either be eliminated or kept based on the testers' desires.

Removing the micro-SEFI-contaminated samples statistically is very difficult, though. On any given run, the micro-SEFIs are either above or below the average number of upsets per sample. If the testers were not careful, though, the micro-SEFIs could be very close statistically to the data that you do not want to eliminate and eliminating too much of the data is possible. Furthermore, as one can see in the data, the average number of upsets per sample fluctuates. These fluctuations are caused by both Poisson statistics that determine the number of ions per sample and the uniformity of the beam. Some times it is possible that beam could be less uniform, causing the beam's flux to drift during the run.

While originally a running average was used to determine whether samples should be eliminated, we found that a custom jackknifing algorithm was more useful. Jackknifing algorithms look at small, contiguous windows of data to determine whether a statistical outlier exists in the window [14]. Since the outliers in our case could either be too high or too low, we designed our jackknifing algorithm to not assume any knowledge about the location of the outlier. We found that that within some windows, the data can be so uniform that the standard deviation for an entire window is less than one upset. We have found that when the standard deviation is very small that much of the data can be eliminated erroneously. Therefore, our algorithm prefers to eliminate data where the standard deviation is larger than 10 upsets. In those cases, any sample that is larger than two sigmas from the jackknifed average is removed from the data set.

The jackknifed data set is shown in Figure 7. One can see that all of the micro-SEFIs have been removed. Some of the data in the middle group was accepted. In particular, the samples from the beam variation around the 800th sample were kept. Since that cluster of potentially bad data would be processed in the same window, removing the potentially bad data through jackknifing is impossible. Since there is no way to prove from just the number of upsets that the data is bad, the data is accepted. If the data turns out to have other characteristics indicating data corruption, the samples will be removed by the second phase of analysis. As it stands, the initial run had 1264 samples, 32 samples were removed through jackknifing, and 1232 samples remained. Therefore, jackknifing removed only 2.5% of the total data set and removed 100% of all of the micro-SEFIs.

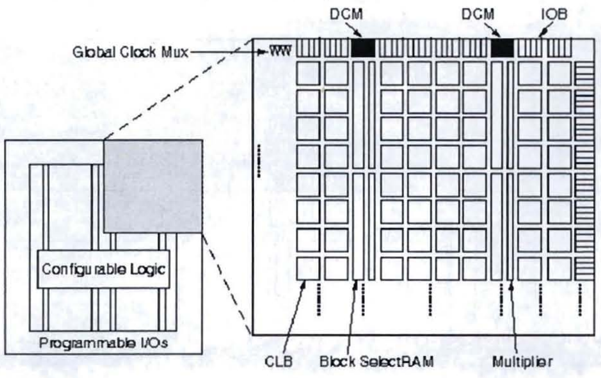
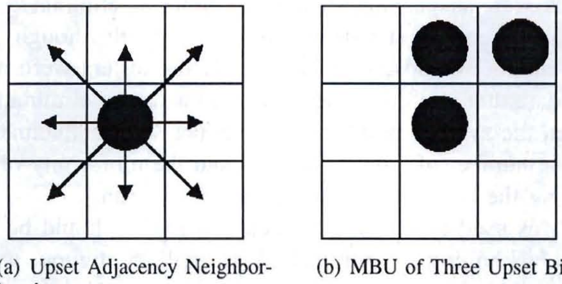


Fig. 8. Physical Layout for Virtex-II

III.B.2 Physical Correlation and Analysis

After jackknifing the data, the data set can be analyzed. We analyze each sample separately and then aggregate the samples for one beam run into one data point. Combining data from separate beam runs is not recommended, unless the same source, energy, θ, and ϕ were used. Data from separate sources, energies or angles should be analyzed as separate data points. In the next section, there is a discussion of how to combine data points into graphs.

The first step of analyzing the data is to translate the upset patterns from the readback data to the physical layout of the programming data to determine the adjacency of upset bits. Figure 8 shows a block diagram of the Virtex-II Family. This layout is similar to the layout for the other families tested. The exterior layout is devoted to the input/output blocks (IOBs) that connect the reconfigurable fabric to the device's pads. The interior region of the device is predominantly configurable logic blocks (CLBs) where the circuit functionality is deployed. The CLBs are laid out in columns with several columns of CLBs laid out contiguously. Occasional columns of clocking and BlockRAM (BRAM) resources are interspersed between the CLB columns. The Virtex-4 and Virtex-5 are a slightly different architecture, and the IOB resources are now located in columns in the reconfigurable fabric, instead of perimeter. The Virtex-4 and Virtex-5 parts that we tested also included Digital Signal Processing (DSP) resources. We use the physical layout to classify adjacent upset bits and their affected resources. We classify a bit as adjacent to another if it lies within one of the eight neighboring memory cells surrounding that bit. Figure 9(a) illustrates the adjacency neighborhood used. Adjacent upsets are classified as MBUs. In Figure 9(b) three upset bits are grouped together in a single MBU. In this way, maximally sized MBUs are found to give an understanding of the size of MBU events. The location of each SBU and MBU is recorded to determine what FPGA resources are affected and the frequency of SBUs and MBUs by resource type.



(a) Upset Adjacency Neighborhood (b) MBU of Three Upset Bits

Fig. 9. Upset Adjacencies and MBUs

For each data point, it is also necessary to determine the amount of error in the data point. For cross-section data the error is often expressed as:

$$\sigma_{error} = \pm \frac{(events)^{1/2}}{fluence \times cos(\theta)} \tag{7}$$

where fluence is the measure of the number of ions that irradiated the device in a set a time, θ is the angle of the test fixture to the beam, and events is the count of the radiation-induced faults. There can be significant error introduced by a number of sources — bad beam dosimetry, contaminants in the beam line and systematic test fixture error. Therefore, within the radiation effects field, even meticulously collected data can be off by a factor of two.

III.B.3 Combining Test Results

After each data set is analyzed, the data is combined to give a view of how energy effects the sensitivity of the device to SEUs. Often times, proton-based results and heavy-ion-based results are separated into different graphs. We have also found that separating angular data from normal-incidence data is often helpful, as angular data can often be confusing when only represented as an increase in effective energy. By standard, data sets are plotted in log-normal graphs.

When plotted together, cross-sections have two interesting characteristics: an *onset threshold* and a *saturation cross-section*. The onset threshold indicates the lowest energy or energy equivalent needed to cause an SEU or a SEFI, which can be less than 1 MeV-cm^2/mg for heavy ions. The saturation cross-section indicates the maximum sensitivity to the radiation source and often does not saturate in modern devices due to the presence of multiple-bit upsets [15]. The data is traditionally fitted to a Weibull curve. This fitting can be done either through Matlab or by hand tuning using a least-squared fit.

III.C Static Test Results

Table 1. Xilinx Parts Tested

Family	Part	Config. Bits	CLBs	Block RAM (Kb)	IOB Pads
Virtex	XCV300	1,751,808	1,536	64	316
	XCV1000	6,127,744	6,144	128	512
Virtex-II	XC2V250	1,593,632	384	432	200
	XC2V1000	4,082,592	1,280	720	432
Virtex-II Pro	XC2VP40	15,868,192	19,392	3,456	804
Virtex-4	XC4VLX25	7,819,520	24,192	1,296	448
Virtex-5	XC5VLX50	13,579,200	7,200	480	560

In this section, we will present data from a number of tests that we have performed using this methodology. The parts we have tested are listed in Table 1. Los Alamos National Laboratory and the Xilinx Radiation Testing Consortium (XRTC) have tested these parts extensively in both proton and heavy ion radiation. Information regarding energies, angles and fluence can be found in [15], [16]. To determine the quality of our data collection procedures, we estimated the probability of a coincident two-bit upsets in our data sets using the monte carlo technique described in Section III.A.3. Table 2 shows the worst case probability that the readback data will have coincident SBUs based on the worst case number of upset bits per device in each device's data set. The proton data sets indicate that coincident SBUs in the data are very unlikely and the probability of real MBU events are 40–80 times larger than the coincident SBUs rate. In the case of the heavy ion data sets much more of the configuration bitstream is upset, which leads to a higher probability of coincident SBUs in the data. Despite this, the probability of a true heavy ion radiation-induced MBU is seven to 105 times larger than the probability of a coincident SBUs. We have used this analysis in the past to determine whether to retest parts, which lead to the re-qualification of the Virtex-II part.

Table 2. Worst Case Percentage of Coincident SBUs in the Data

Family	Worst Case Coincident SBUs	Worst Case upsets/device
	Proton Test Data	
Virtex	0.0006%	0.0002%
Virtex-II	0.0298%	0.0075%
Virtex-II Pro	0.0277%	0.0069%
Virtex-4	0.0379%	0.0095%
Virtex-5	0.0005%	0.0001%
	Heavy Ion Test Data	
Virtex	5.4077%	1.283%
Virtex-II	5.8943%	1.577%
Virtex-II Pro	1.1697%	0.289%
Virtex-4	0.0472%	0.0098%
Virtex-5	0.0110%	0.0022%

Table 3 has a list of SEU bit cross-sections and SEFI device cross-sections for 63.3 or 65 MeV protons and Figure 10 shows the SEU bit cross-sections for heavy ions for Virtex family devices. Note that in proton the SEFI device cross-sections from Table 3 appear to be on the same scale as the SEU bit cross-sections, which is consistent with our understanding that the control logic is controlled by tens to hundreds of configuration bits. It should also be noted that the sensitivity to heavy ions is five to seven orders of magnitude larger than protons.

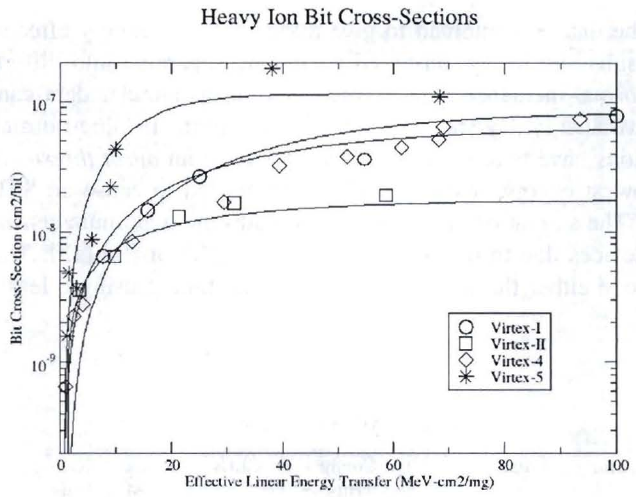


Fig. 10. Heavy Ion Bit Cross Sections for Virtex Family Devices [13].

Table 3. Bit Cross-Section for SEUs and Device Saturation Cross-section for SEFIs for Protons for Several Xilinx FPGAs [15]

Device	Energy (MeV)	σ_{bit} (cm^2/bit)	σ_{SEFI} (cm^2/device)
XCV1000	63.3	1.32×10^{-14}	$\approx 7.1 \times 10^{-13}$ (config SEFI)
XC2V1000	63.3	2.10×10^{-14}	9.46×10^{-13}
XC4VLX25	63.3	1.08×10^{-14}	6.43×10^{-12}
XC5VLX50	65.0	7.56×10^{-14}	Unknown

Table 4. Frequency of Upset Events and Percent of Total Events Induced by Proton Radiation (63.3 and 65 MeV) for Five Xilinx FPGAs

Family	Total Events	1-Bit Events	2-Bit Events	3-Bit Events	4-Bit Events
Virtex	241,166	241,070 (99.96%)	96 (0.04%)	0 (0%)	0 (0%)
Virtex-II	541,823	523,280 (98.42%)	6,293 (1.16%)	56 (0.01%)	3 (0.001%)
Virtex-II Pro	10,430	10,292 (98.68%)	136 (1.30%)	2 (0.02%)	0 (0%)
Virtex-4	152,577	147,902 (96.44%)	4,567 (2.99%)	78 (0.05%)	8 (0.005%)
Virtex-5	2,963	2,792 (94.23%)	161 (5.43%)	9 (0.30%)	1 (0.03%)

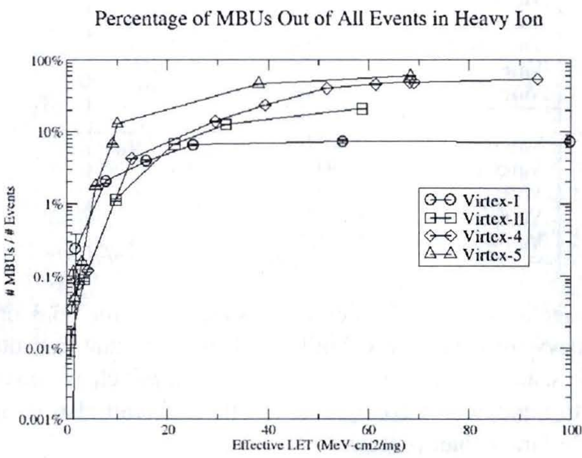


Fig. 11. Percent of MBU Events Out of All Events Induced by Heavy Ion Radiation for Four Xilinx FPGAs

III.D Error Rates in Space

Essentially every orbital scenario is different with regards to radiation effects. Therefore, it is generally helpful to perform mission analysis and planning to make orbital SEU rate estimations. A complete discussion of on-orbit SEU rate estimation is beyond the scope of this design guide, but we will provide a few references for further information and study in the following paragraphs.

Two of the more accessible tools that are commonly used for on-orbit SEU rate estimation are the Cosmic Ray Effects on Micro-Electronics Rev. 96 (CREME96) tool suite developed by the Cosmic Ray Physics Section at the Naval Research Laboratory (NRL) and the commercial tool suite SpaceRad developed by Space Radiation Associates. SpaceRad provides a nice GUI around the older NRL CREME and other models. Both tools use the AP8 trapped proton model which was developed by Drs. J. I. Vette and D.M. Sawyer, first at the Aerospace Corporation and later at NASA's National Space Science Data Center (NSSDC) under the Space Environment and Effects program. Like most models, the AP8 models have some limitations which preclude them from being particularly accurate [17] [18] [19] . If a designer needs more accurate results there are a few tools and models that the CREME96 website.

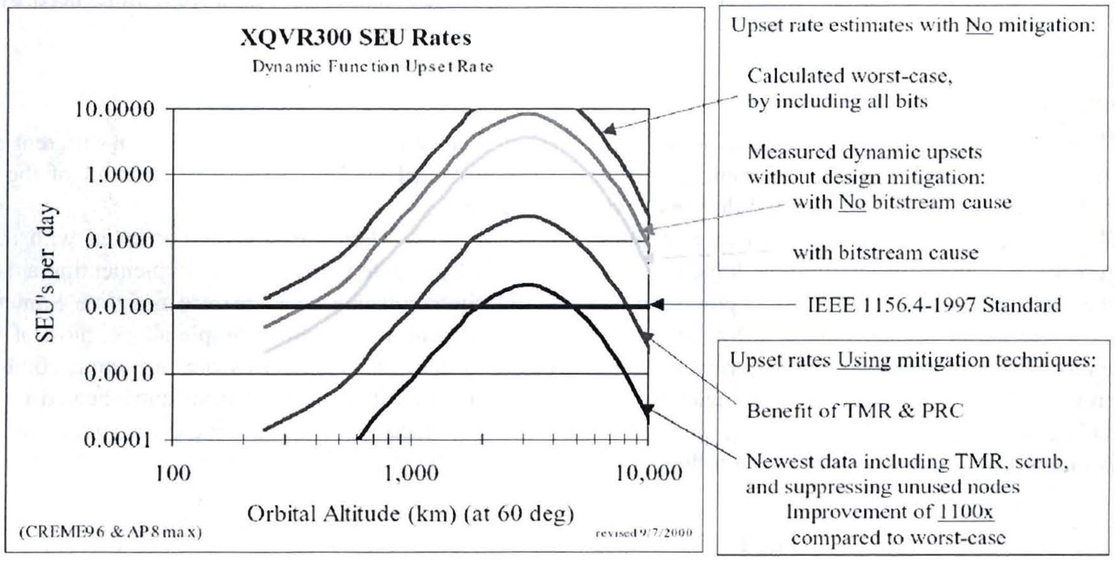


Fig. 12. CREME96 SEU calculations for a design on the XQVR300 platform using various forms of mitigation [20].

IV. Dynamic Testing

The best practices for FPGA-based spacecraft design encourages the use of triple-modular redundancy (TMR) in the user circuit to mask SEU-induced errors on the FPGA, in addition to error detection and correction of data stored in the device's programming memory. Not only is applying TMR an error-prone process, but sometimes the designers are unable to apply "full" TMR to the user circuit due to device size constraints. The user circuit needs to be tested pre-launch to determine whether it is working as expected, including whether TMR has been applied effectively, and whether the availability requirements are met.

The current "gold standard" for pre-launch testing of user circuits is validation through radiation experiments at a particle accelerator. For these tests the designer has the choice of using either a proton or a heavy ion accelerator, the most likely ionized particles to cause SEUs while on orbit. Fully space-qualifying a design could take days worth of time and tens of thousands of dollars at an accelerator to exercise all of the possible radiation-induced failure modes and find all of the problems with a user design. Since radiation-induced faults are statistical in nature, it may be too expensive to get good test coverage and difficult to understand how the output errors correlate to design flaws in the user design. We have found that fault injection and modeling tools are much better at providing feedback about specific design flaws to the designer. We have both a fault injection tool — the SEU Emulator — and a modeling tool — the Scalable Tool for the Analysis of Reliable Circuits (STARC) — that can be used by FPGA designers to augment radiation experiments. By using these tools, the accelerator testing is only needed for final, pre-launch validation of the user design.

IV.A Dynamic Testing Background

SEUs affect the user circuit usually in one of two ways: by changing circuit functionality or by changing the flow of data through the circuit. SRAM bits that cause output errors when affected by an SEU are called *sensitive programming data* *bits* or more simply *sensitive bits*. Good testing should help designers quantify and locate sensitive bits that are the result of untriplicated logic, placement-related issues, and the application of TMR. In a fully TMR-protected circuit SEUs in the user logic should not affect the output data in the system, with the exception of some occasional placement-related issues that depend on how the circuit is implemented on the device [13]. In a design that has only had TMR partially applied to the design (*Partially TMR-Protected*), there will also be untriplicated logic and routing that will have some sensitive bits. In the remaining portion of this section will discuss how SEUs affect partially and fully TMR-protected user circuits.

IV.A.1 Fully TMR-Protected User Circuits

In fully TMR-protected user circuits, no single-bit SEUs should cause output errors unless TMR was not applied properly or problems with logical constants exist. We have found that TMR-protected systems can be vulnerable to SEUs if the implementation of the logical constants is not carefully controlled. These logical constants are frequently used to tie off unused resource inputs, such as the "carry in" to ripple carry logic or unused address lines to a memory. With newer devices multiple bit upsets (MBUs), where a single SEU causes multiple bits to fail, have become more common [15], especially with heavy ions. We have observed MBU-induced TMR defeats [13]. These TMR defeats appear to be strongly influenced by placement issues.

IV.A.2 Partially TMR-Protected User Circuits

When TMR is only applied to a portion of a circuit due to resource constraints, SEUs can affect two different areas of the circuit: untriplicated logic and untriplicated routing. Partially TMR-protected designs could also have all of the placement-related issues that affect fully TMR-protected designs as described above.

First of all, any untriplicated logic could cause output errors to manifest when the logic is corrupted with an SEU. For example, Figure 13(a) shows a programmable logic element, called a *lookup table* (LUT), that is implementing a 4-input AND function. If the one bit that defines the "true" condition has an SEU, the result is a constant-zero function. Sometimes SEUs in untriplicated logic can be logically masked by the data the circuit is executing. For our example above, most of the possible input combinations will return the correct output. If the data in the system never exercises the one input combination that causes the error to manifest, the error will be logically masked. Output errors that manifest from untriplicated logic can only be fixed by changing the design. Therefore, the number of sensitive bits due to unprotected logic are immutable to how the user circuit is placed on the device by the design flow tools, although the location of these bits might change by rerunning the tools.

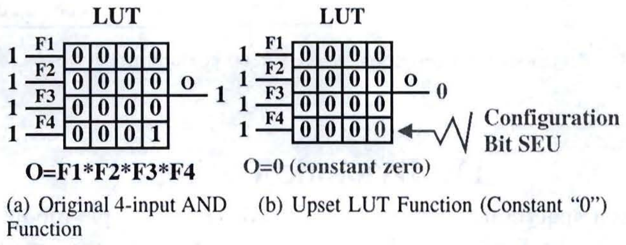


Fig. 13. LUT Upset Example

A second set of output errors in partially TMR-protected designs stem from the programmable routing network in the untriplicated parts of the design. SEUs in the routing network changes the flow of data through the circuit. For example, an SEU in the routing network could cause an input to a LUT to float. Unlike untriplicated logic, some of the SEUs in the routing network can be influenced by the design flow tools that determine how the user circuit is placed and routed on the device. Within blocks of untriplicated logic, the design flow tools could affect the the number and location of sensitive bits in the routing network by lengthening/shortening routes or changing their location on the device.

Sometimes the logic in a design might be completely triplicated, but some or all of the input signals might not be due to the lack of I/O resources. While the input data signals will be triplicated once the data is registered the first time, the clock and reset trees will remain untriplicated. SEUs in the programmable routing along the main trunks of the clock and reset trees are very likely to affect the entire circuit, but SEUs in the leaves of these trees will often be masked if only one module in the TMR-protected design is affected. The number and location of sensitive bits can be affected by placement, but cannot be eliminated through rerunning the placement tools.

IV.B Modeling Tools

Reliability analysis is traditionally done with modeling tools. For designers of many types of systems, these tools allow the designers to focus on creating accurate models of their systems, instead of focusing on how to calculate the reliability. Since FPGA user circuits already have accurate functional models in terms of hardware descriptions and netlists, the right modeling

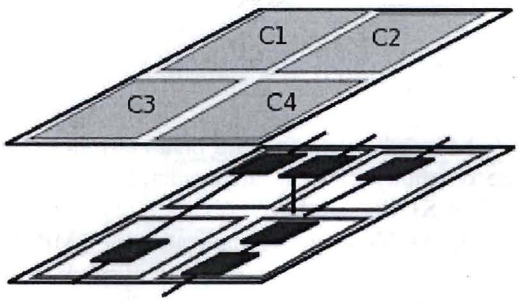


Fig. 14. Hierarchical Exploration of Circuit Design

tool can leverage these descriptions directly for reliability analysis, enabling the detection and correction of design flaws in the design phase where fixing flaws is significantly cheaper.

Traditionally, circuit reliability has been determined using purely analytical approaches [21] or techniques that model Boolean networks as probabilistic systems [22]–[25]. These modeling techniques represent circuits as probabilistic transfer matrices, stochastic Petri-nets, Markov chains or Bayesian networks. These analytical approaches have been found to be error-prone and computationally complex for the analysis of large designs. Similarly, a number of limitations have been identified for many modeling-based approaches. First of all, model and input data set creation greatly increase the time commitment of using these tools. Transforming circuits into intermediate probabilistic system models is an additional, computationally complex task. The complexity of calculating the circuit reliability also grows exponentially with circuit size and the number of input vector sets and the computation can take a prohibitively long time to finish. The exception to these problems is the SETRA tool [26] that directly addresses the state space issues as well as automated model generation.

For these reasons, traditional tools are not well-suited for the size of designs used in most FPGA systems. All of these limitations have led to the development of the Scalable Tool for the Analysis of Reliable Circuits (STARC) tool, which specifically addresses the limitations of model creation, input data sets and computational complexity with these solutions:

- industry-standard Electronic Design Interchange Format (EDIF) representation of a circuit as the input model,
- no input vector sets,
- memoization to reduce the computational complexity, and
- combinatorial reliability calculations.

By using the EDIF circuit representation, the designer can assess the reliability of a circuit during the design process, even if the design is not complete, the design does not work, or the hardware is not available. Without the use of input vector sets reliability is determined through the probability of device or input failure and is not dependent on specific input data sets. Without input data sets, the reliability of sub-circuits are determined by type, such as a two-bit adder, and memoized for reuse. In this manner, large-scale circuits are analyzed in a fraction of the time required by traditional approaches, making design exploration more worthwhile.

There are a few disadvantages to this approach. First, since EDIF does not contain information about the routing and the placement on the device, routing reliability is currently statistically estimated from case studies of routing placement. Furthermore, currently there is no way to assess placement-related issues, such as MBU-induced TMR defeats. We are currently working on a solution for this limitation for designs that have gone all the way through the design flow. Second, without input vector sets logic masking cannot be taken into account, and STARC estimates the worst case failure rate. While this value may be lower than the value determined by other tools [27], STARC provides a useful lower bound on the circuit's reliability.

By using the EDIF circuit representation the hierarchy in the circuit should be preserved. Since designers tend to create complex circuits by creating less complex sub-circuits, maintaining this structure can be very useful in calculating the reliability. In particular, STARC can readily exploit memoization by memoizing the reliability of sub-circuits and reusing the reliability values for sub-circuits of the same type. This reuse allows the computation to grow polynomially instead of exponentially. This hierarchical nature allows circuits to be examined at the highest level of abstraction or the most minute level of detail. STARC automatically determines the appropriate level of the hierarchy that needs to be explored. An example of this hierarchy is shown in figure 14. In this example, the reliability of components 1-4 are determine first, memoized, and then used to determine the reliability of the entire circuit.

During hierarchical exploration, dependency graphs for each primary output at each level of the hierarchy are determined. The dependency graph has all of the sub-circuits between the output and the reachable inputs. Since not all logic or inputs are reachable from every output, this technique removes unrelated logic from the reliability calculation. Once the dependency graph for an output is determined, the reliability can be calculated. In unmitigated designs, the quantity of sensitive bits is the total area of the dependency graph:

$$A(O) = \sum_{i=0}^{m} A(C_i), \tag{8}$$

where $A(X)$ is the sensitive area of X (where X is either a wire or a cell) and $C = \{C_0, ..., C_m\}$ is the set of cells that can be reached from output wire O. The reliability of basic architectural elements, such as LUTs and user flip-flops, are pre-determined and are statically loaded when STARC starts.

STARC was designed to help designers find problems in the application of TMR. For mitigated circuits, the sensitive area is confined to the part of the design that is not triplicated, as triplication will mask errors as long as there is one voter for each redundant module. There are cases where the design flow tools, in particular synthesis tools, will alter the circuit so that the TMR modules are no longer functionally equivalent or independent. In these cases two modules will share a partial calculation with the third module and the shared partial calculation becomes a single point of failure. Feedback loops in TMR-protected systems are also sensitive to *persistent errors* [28], and need to use triplication and voters to break the feedback loops. If the feedback loops are not handled in this manner, the feedback loop's state will not be able to autonomously resynchronize after the SEU is removed. In this scenario, while the first SEU in the feedback loop will be masked, another SEU in the feedback loop is not guaranteed to be masked.

In all of these cases, STARC provides warnings and information about the design to the designer. The output of the tool provides the designer a list of sub-circuits that are untriplicated, and warnings about potential single points of failures from functionally nonequivalent modules and logical constants. Since EDIF is tightly coupled to the circuit design, the designer should be able to directly use STARC's output to find and fix the design flaws in the user circuit.

IV.C Fault Injection Testing

Once a design is completed and hardware is available, it is possible to move on to fault injection. Unlike modeling tools, fault injection works with the actual hardware implementation of the user circuit, allowing placement-related issues to be assessed. If designed well, a fault injection tool should have good fidelity to accelerator testing and on-orbit behavior, since the hardware and the operational behavior mimic actual usage. Finally, we would like to note that fault injection on the actual "flight" hardware is highly desirable since it is more likely to mimic or illustrate the consequences of individual upsets.

Fault injection is possible, because the interfaces that control device programming (or *configuration*) are accessible to the designer. These interfaces can be used by the designer to purposefully corrupt the programming data to mimic SEUs in programming data. While LANL designed one of the first fault injection testbeds for FPGAs with the SLAAC1-V SEU Emulator [29], since then many other organizations have created them [30]–[32]. We have also gone on to make other versions of our fault injection tool to support newer hardware devices and support MBU testing.

Fault injection tools for FPGAs all have the same basic algorithm, as shown in Figure 15. With this algorithm, faults can be injected throughout the entire programming data. It is important to run a large number of input vectors through the system after the fault is injected to avoid logical error masking. Since running a complete set of test vectors is often infeasible, our SEU emulator generates input vectors randomly so that better coverage is possible by running multiple tests on the same design. Each test provides coverage for 250,000-500,000 test vectors. It is also feasible to run a complete set of test vectors for limited portions of the circuit. Resetting and resynchronizing the user circuit after the SEU is removed is also important so that the effects of each emulated SEU is kept independent from others. Independent trials ensures that errors are properly attributed to the right programming data bit and that latent bad state from one fault injection iteration does not affect the next one.

There are usually two types of fault injection systems based on whether one or two FPGAs are used. In our SEU Emulator tool two FPGAs are used, each one hosting the same user design. Faults are injected into the *design under test* (DUT) FPGA and then run in lockstep with the same input vectors with the *golden* FPGA, which receives the same input vectors. The advantage of this system is that sharing input vectors, detecting output errors, and testing the system for resynchronization is very easy. The disadvantage is the complexity of designing the lockstep system.

In the single FPGA fault injection systems, the input vectors are run through the system twice: once without fault injection and once with fault injection. The advantage of this system is that it takes less hardware and is easier to design than a lockstep system, but the disadvantage is that the input vectors and correct output vectors need to be saved in the system. Furthermore, determining miscompares in the output data is not simple.

In general, a good fault injection system should be able to handle different types of user circuits. With many fault injection systems, the number of clock and reset pins, the width of input and output buses, and the pin locations are often set. Due to these restrictions, sometimes the user design has to be changed to fit the fault injection system, which can reduce the usefulness of fault injection. On occasion, we have found some cases that do not lend themselves to fault injection. In these cases, the use of modeling tools is even more important.

Once fault injection is completed, the SEU locations need to be tied back to the design. If fault injection only reports a handful of errors, the designers can decide that the user circuit meets the availability requirements for the system and that further design exploration to fix design flaws is unnecessary. Unlike when using STARC, tying design problems found through

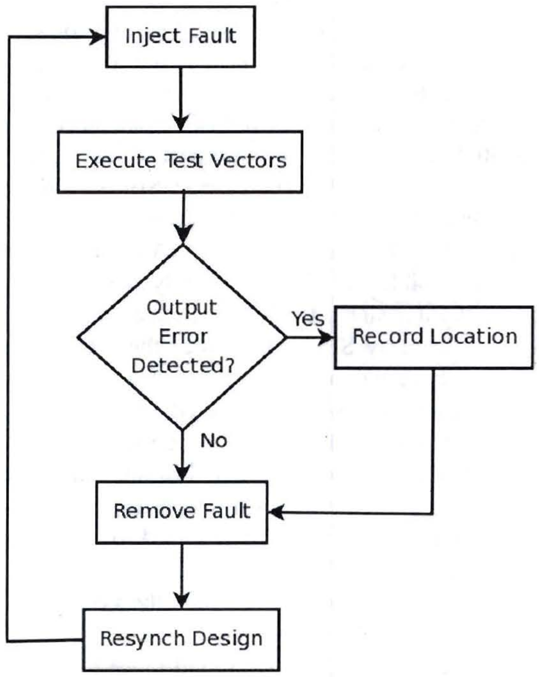


Fig. 15. Basic Fault Injection Algorithm [29]

fault injection to the design can be quite difficult and time consuming. While the fault injection tool returns the locations of sensitive bits, most designers do not know how to translate that location into a physical location on the device. If the physical location is determined, it is possible to use a Xilinx design flow tool, called FPGA Editor, to determine what part of the user circuit is in that location. There are times, though, that even knowing the part of the design that is causing the problem does not help. Since many errors manifest in the routing network of TMR-protected designs, it is possible the fault is caused by a signal that is passing through a routing switch, rather than an error in the user logic.

IV.D Accelerator Testing

One of the advantages of doing accelerator testing after the use of fault injection and modeling tools is that the designers should be better prepared for accelerator testing. To this end, the designers should know the areas of the circuit design that should cause output errors from the modeling tools and know the locations of these faulty areas through fault injection. Furthermore, if the designer has been using a lockstep fault injection tool, the fault injection hardware can be used as the test fixture for the accelerator tests. Even the lockstep fault injection software can be used as part of the accelerator test fixture with minor modifications. If a lockstep system was not used for fault injection, a test fixture that can easily detect output errors is highly desirable so that real-time feedback is available during the test to ensure that the test is functioning properly. A number of FPGAs will be needed for the test, since the parts are only guaranteed to operate properly up to 100 KRads of ionizing dose.

The algorithm for the software aspect of the test fixture is very similar to the fault injection tool's algorithm. Instead of injecting faults artificially, though, the particle accelerator will be injecting radiation-induced SEUs. Unlike fault injection, controlling the number of upsets that occur during one loop of the algorithm is more difficult. SEU removal and SEU-induced single-event functional interrupts (SEFIs) that affect the functionality of the entire device complicate testing. These three problems will be discussed below.

The arrival time of radiation-induced faults are a Poisson random processes. As the designer will want to reduce the probability of multiple independent upsets (MIUs) causing an output error, the beam's flux is tuned so that on average only one SEU occurs per algorithm loop. Even still, Poisson statistics tell us that, even if the beam's flux is tuned to one SEU per algorithm loop, there is a 37% chance that no SEUs occur, a 37% chance that one SEU occurs, and a 26% chance that two or more SEUs occur during the given time period. Since not all SEUs cause output errors and only a few sensitive bits might exist, it can take some time for errors to manifest. The formula for determining the time interval for one output error to manifest is:

$$N_{OE} = \left(\frac{N_{bits}}{N_{device}}\right)^{-1} \quad (9)$$

$$T_{OE} = N_{OE} \times T_s, \quad (10)$$

where N_{OE} is the number of samples until an output error, N_{bits} is the number of sensitive bits, N_{device} is the total number of bits in the device, T_s is the time length for each sample, and T_{OE} is the average time span until an output error.

Removing SEUs quickly is important so that the user circuit can recover before the next SEU occurs. There are two ways to remove an SEU during an accelerator test. First, the SRAM FPGA can be completely reprogrammed through *off-line* programming, where the FPGA is taken off line for the express purpose of reconfiguring the device. Taking the device off line tends to be costly in terms of time, but often needed in the case of SEFIs, where a full reprogramming of the device is the only reliable method for restoring the FPGA to a known, functional state. A second approach available to Xilinx SRAM FPGAs is to use *on-line* programming capabilities. In this case, the FPGA remains operational while its programming data is repaired. Using on-line reprogramming, it is possible to either completely rewrite all of the programming data or to only fix the portions that have SEUs. This later case is safer since it affects the least amount of programming data at a time and since the FPGA's programming circuitry can be affected by SEUs. To reduce the time required to identify an SEU and fix it, we recommend the use of external SEU detection hardware as opposed to software.

After the accelerator test is completed, the results need to be examined so that correlations between output errors and known sensitive bits can be determined. Since the SEUs in accelerator testing do not present themselves in the system uniformly or at specified time intervals, correlating output errors to specific SEU locations can be a challenge. In some cases, the output error follows the SEU by several algorithm loops and other times the reporting software will output the existence of the output error before the SEU location. On other occasions, some output errors need to be dropped from the data set, such as when the incidence of multiple independent upsets are the cause of the output error. Often times all of the results around a SEFI event will need to be ignored, since removing the SEFI is time consuming and the system will likely report output errors for several iterations until the circuit state resynchronizes.

As long as the user circuit that is being tested is the same one tested in fault injection, the results from fault injection can be used to disambiguate the accelerator test results. Due to the problems described with attributing SEUs to output errors, the most effective approach for analyzing accelerator results is to look at several SEU locations before and after the output error in the log. This "window" of SEU locations can then be compared to fault injection results to determine if any of these SEU locations caused an output error in fault injection. While this method can usually help a designer correlate output errors with fault injection results, some output errors cannot be completely correlated. In some cases, the errors are due to SEUs in user memory, such as flip-flops, and not due to programming data upsets. The number of these types of errors can be predicted based on the amount of user memory used in the design and the susceptibility of these memory elements to SEUs. In other cases, sometimes the accumulation of errors in the circuit state caused the output error. For these cases, sometimes part of an accelerator test can be "played back" using the fault injection tool, where the tool uses the accelerator log to inject faults in specific locations in a particular order. Through this attribution process, the designer can determine whether the output error can be explained and whether further design exploration is needed to address potential design flaws.

For fully or mostly mitigated designs, accelerator testing should be uneventful and the user circuit should be able to operate for minutes or longer without any output errors. For example, using Equation 10, if fault injection only found 100 sensitive bits in a device with 75 million bits, at one algorithm loop per second the first output error is only guaranteed to occur randomly within a 208 hour time span. Some designers will do multiple rounds of tests with different flux levels and different durations. In particular, one test might be very low flux over several hours, mimicking average operation on orbit, and another test might have a very high flux over a couple of minutes, mimicking solar flare conditions or to otherwise reduce test time. If, at the end of these tests, the design is able to operate either error-free or within the availability requirements, the design is considered space ready.

If the error rate is much higher than indicated by the fault injection tool, either the flux could be too high or there might be problems with either the fault injection or accelerator test fixture. When designing new fault injection and accelerator test fixtures it is important to test the setup by correlating output errors to the source of the errors to ensure fault injection works, as well as correlating the results of fault injection and the accelerator tests. If the results cannot be correlated, then the methodologies for both systems need to be examined.

IV.E Dynamic Testing Results

In this section, we will compare the use of these three methodologies on a circuit. The circuit, an adder tree, is fully triplicated and was designed originally to test for placement-related issues due to both MBUs and logical constants. This design was implemented for a Xilinx Virtex-II FPGA (XC2V1000). All three methodologies were used on this design. In the following paragraphs, we will describe the amount of time, the quality of the results, and the cost of using these methodologies.

In terms of time, STARC is comparatively much faster than the other two methods. Within a minute, the tool returned the result that the design was triplicated properly and with warnings that placement-related issues could exist from logical constants. As STARC cannot currently estimate the placement-related issues, it is unable to estimate how many bits in the design could cause output errors. In terms of cost, STARC is free to government users.

In terms of test coverage, the SEU Emulator was much more complete than the other two methods. With fault injection, we were able to find 285 single-bit SEU locations, 18,733 2-bit SEU locations, 11,264 3-bit SEU locations, and 19,464 4-bit SEU locations that cause the design to output bad data. Each pass through the fault injection test takes two hours per run and each MBU test is a separate test. As the MBU tests are run with specific MBU shapes based on our analysis of how MBUs affect the Virtex-II, we were able to constrain the MBU tests to the six most common shapes. In all, fault injection tests took 14 hours for the seven tests. In terms of cost, the fault injection hardware is about $6,000 and the software is free to government users.

As validation for both of these tests, we did a two hour long test at at the University of Indiana Cyclotron Facility's proton accelerator. During this test we were able to observe 50 output errors, of which 21 were attributed to SEFIs, 13 were attributed to MIUs, and 16 were attributed to the two phenomenas that we were looking for. Of the 16 output errors, 88% we were able to later correlate to known fault injection error locations. At three algorithm iterations a second, we would have been able to test all of the single bit errors in no less than 16 days, assuming that no single-bit fault location was exercised multiple times. As is, we were able to test at a higher flux to be able to shorten the time frame of the test considerably. Since the MBU-related issues have only a 2% chance of occuring due to proton radiation, completing the 2-bit test would have taken over two years. In terms of cost, we were able to use the hardware and software from fault injection and only had to pay the accelerator fees of $1,200 for two hours of test time and $500 for the FPGA. Had we completed the single-bit test, we would have to pay for at least 385 hours of testing and 192 FPGAs for a total cost of $288,000.

Since we shortened our accelerator test considerably, the initial cost of the hardware for the fault injection tool is the highest of the three test methodologies. Had we done a complete validation of the user circuit, though, the accelerator test would have been the most expensive. Also, it should be noted that the cost of the fault injection tool is amortized across all of the fault injection tests and the accelerator testing. Since the hardware infrastructure can be reused an unlimited number of times, if the FPGA is not irradiated, the cost is reasonable. When the test coverage is factored in, the amount of time and cost invested in the fault injection tool is the best option. While fault injection should never replace accelerator testing, the accelerator test was shortened when we were able to confirm our fault injection results. We also believe that using STARC decreased the overall time commitment and iteration that takes place in fault injection and accelerator testing. Finally, since the design had TMR properly applied to the circuit from the beginning, which was confirmed throughout the testing, there was no need to determine what was wrong with our design.

V. Conclusions

In this paper we presented a three-tiered methodology that finds design flaws in FPGA user circuits and locates the source of the faults on the FPGA. One methodology used the circuit representation to find design flaws through modeling. The second methodology used fault injection to locate how the design flaws translated to physical locations on the FPGA. The final method was an accelerator test to validate the previous results. We also showed how these three methodologies compared in terms of test coverage, time, and cost. While the modeling tool was the fastest, fault injection was the best methodology in terms of cost and test coverage.

References

[1] 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 *Proceeding of the Military and Aerospace Programmable Logic Devices International Conference(MAPLD)*, Laurel, MD, September 1999.

[2] 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.

[3] T. D. Swanson and G. C. Birur, "NASA thermal control technologies for robotic spacecraft," *Applied Thermal Engineering*, vol. 23, no. 9, pp. 1055–1065, 2003.

[4] A. Minichiello and C. Belady, "Thermal design methodology for electronic systems," in *Thermal and Thermomechanical Phenomena in Electronic Systems*, 2002, pp. 696–704.

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

[6] P. Graham, M. Caffrey, M. Wirthlin, E. Johnson, and N. Rollins, "Consequences and categories of SRAM FPGA configuration SEUs," in *Proceeding of the Military and Aerospace Programmable Logic Devices International Conference(MAPLD)*, Washington, DC, September 2003.

[7] H. Quinn, P. Graham, K. Morgan, J. Krone, M. Caffrey, and M. Wirthlin, "An introduction to radiation-induced failure modes and related mitigation methods for xilinx sram fpgas," in *in the proceedings of The Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2008.

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

[9] G. Allen, G. Swift, and C. Carmichael, "Virtex-4VQ static SEU characterization summary," Xilinx Radiation Test Consortium, Tech. Rep. 1, 2008.

[10] O. Flament, J. Baggio, C. D'hose, G. Gasiot, and J. Leray, "14 mev neutron-induced seu in sram devices," *IEEE transactions on nuclear science*, vol. 51, no. 5, pp. 2908 – 2911, 2004.

[11] J. Ziegler and H. Puchner, *SER – History, Trends and Challenges: A guide for designing with Memory ICs*. Cypress, 2004.

[12] J. George, R. Koga, G. Swift, G. Allen, C. Carmichael, and W. Tseng, "Single event upsets in xilinx virtex-4 FPGA devices," in *Radiation Data Workshop of the Nuclear and Space Radiation Effects Conference*, 2006, pp. 109–113.

[13] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain Crossing Errors: Limitations on Single Device Triple-Modular Redundancy Circuits in Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2037 – 43, 2007.

[14] B. Efron and R. Tibshirirani, *An Introduction to the Bootstrap*. Chapman and Hall/CRC, 1993.

[15] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2455 – 2461, December 2005.

[16] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "Static proton and heavy ion testing of the Xilinx Virtex-5 device," in *The Proceedings of Data Workshop for Nuclear and Space Radiation Effects Conference*, July 2007, pp. 177 – 184.

[17] N. R. Laboratory, "Cosmic ray effects on micro-electronics rev. 96," December 2003, https://creme96.nrl.navy.mil/cm/AP8Accuracy.htm.
[18] ——, "Cosmic ray effects on micro-electronics rev. 96," December 2003, https://creme96.nrl.navy.mil/cm/trplimits.htm.
[19] E. L. Petersen, "Predictions and observations of SEU rates in space," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, December 1997.
[20] 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.
[21] J. A. Abraham and D. P. Siewiorek, "An algorithm for the accurate reliability evaluation of triple modular redundancy networks," *IEEE Transactions on Computers*, vol. 23, no. 7, pp. 682–692, July 1974.
[22] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Accurate reliability evaluation and enhancement via probabilistic transfer matrices," in *Design, Automation and Test in Europe (DATE'05)*, vol. 1. New York, NY, USA: ACM Press, 2005, pp. 282–287.
[23] C. Hirel, R. Sahner, X. Zang, and K. Trivedi, "Reliability and performability using SHARPE 2000," in 11^{th} *Int'l Conf. on Computer Performance Evaluation: Modeling Techniques and Tools*, vol. 1786, 2000, pp. 345–349.
[24] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla, "Evaluating the reliability of NAND multiplexing with PRISM," *IEEE Transactions on CAD*, vol. 24, no. 10, pp. 1629–1637, 2005.
[25] F. V. Jensen, *Bayesian Networks and Decision Graphs*. New York: Springer-Verlag, 2001.
[26] D. Bhaduri, S. K. Shukla, P. S. Graham, and M. B. Gokhale, "Reliability Analysis of Large Circuits Using Scalable Techniques and Tools," *IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications*, vol. 54, no. 11, pp. 2447 – 60, November 2007.
[27] D. Bhaduri and S. Shukla, "NANOLAB—a tool for evaluating reliability of defect-tolerant nanoarchitectures," *IEEE Transactions on Nanotechnology*, vol. 4, no. 4, pp. 381–394, 2005.
[28] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-Induced Persistent Error Propagation in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2438 – 45, 2005.
[29] 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.
[30] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, G. Sechi, and R. Weigand, "Evaluation of single event upset mitigation schemes for sram based fpgas using the flipper fault injection platform," in *Proc. of the 22th IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT07)*, September 2007, pp. 105–113.
[31] M. Berg, C. Perez, and H. Kim, "Investigating mitigated and non-mitigated multiple clock domain circuitry in a Xilinx Virtex-4 field programmable gate arrays," in *Single-event effects symposium*, 2008.
[32] G. Swift, C. W. Tseng, G. Miller, G. Allen, and H. Quinn, "The use of fault injection to simulate upsets in reconfigurable FPGAs," 2008, submitted to the military and aerespace programmable logic devices conference (MAPLD08).