SEU-Induced Persistent Error Propagation in FPGAs

Keith Morgan, Michael Caffrey, Paul Graham, Eric Johnson, Brian Pratt, and Michael Wirthlin

***Abstract*—This paper introduces a new way to characterize the dynamic single-event upset (SEU) cross section of an FPGA design in terms of its persistent and nonpersistent components. An SEU in the persistent cross section results in a permanent interruption of service until reset. An SEU in the nonpersistent cross section causes a temporary interruption of service. These cross sections have been measured for several designs using fault-injection and proton testing. Some FPGA applications may realize increased reliability at lower costs by focusing SEU mitigation on just the persistent cross section.**

***Index Terms*—Dynamic testing, error propagation, FPGA, persistence, proton accelerator, radiation, simulator, single-event upset (SEU).**

I. Introduction

FIELD programmable gate arrays (FPGAs) are an attractive solution for space system electronics. The dense array of programmable logic and block memories supports the use of sophisticated, high-throughput signal processing applications in space systems. However, FPGAs are susceptible to radiation-induced single-event upsets (SEUS). SEU mitigation must be provided to insure reliable operation of FPGA devices in a radiation environment [1]–[3].

FPGAs have a static sensitive cross section that has been characterized by device manufacturers [4]. When programmed with a specific circuit design, the device has a dynamic cross section that depends upon the circuit's utilization of logic elements, flip-flops, and other programmable resources. Because a given design does not use all of the FPGA resources, the dynamic cross section is smaller than the static cross section. The dynamic cross section of a specific FPGA design can be accurately characterized by fault-injection tools and radiation testing [5]. Extensive tests have characterized the dynamic cross section for a variety of FPGA applications [6].

Well-defined techniques such as triple-modular redundancy (TMR) exist to mitigate the dynamic cross section of a design [2]. These methods tend to provide increased reliability, but at high resource costs. They increase reliability by applying redundancy to the *entire* dynamic cross section.

Manuscript received July 8, 2005; revised August 26, 2005. This work was supported by the Department of Energy through the Deployable Adaptive Processing Systems and the Cibola Flight Experiment projects at LANL as well as by NASA through the Reconfigurable Hardware in Orbit project under AIST Contract #NAG5-13516. Approved for public release under LA-UR-05-6883; distribution is unlimited.

K. Morgan, B. Pratt, and D. E. Johnson are with Los Alamos National Laboratory, Los Alamos, NM 87545 USA, and also with Brigham Young University, Provo, UT 84602 USA.

P. Graham and M. Caffrey are with Los Alamos National Laboratory, Los Alamos, NM 87545 USA.

M. Wirthlin is with the Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602 USA.

Digital Object Identifier 10.1109/TNS.2005.860674

In this paper we will show that the dynamic cross section of an FPGA design can be separated into two components. Furthermore, we will show that acceptable levels of reliability may be realized at lower costs, by focusing mitigation on just one of these two components.

The first component is the "persistent" dynamic cross section [7]. An SEU in this cross section results in a permanent interruption of service until reset. The second component consists of the remaining "nonpersistent" structures. An SEU in this cross section will cause a temporary interruption of service, before the design returns to proper operation.

This paper will begin by reviewing the dynamic cross section of an FPGA design. Next, the persistent cross section will be introduced and contrasted with the nonpersistent component. We will present examples of circuits with a persistent cross section and demonstrate how the circuit's outputs are affected by an SEU in that cross section. The results of several tests performed using fault-injection and proton irradiation will be presented. We will forecast SEU, data-loss and failure rates for a sample application in several different orbits. Finally we will draw conclusions and propose ideas for future work.

II. Dynamic FPGA Cross Section

A device's static cross section is a function of the surface area and sensitive volume of nodes susceptible to an SEU-induced upset of stored state [8]. Typically manufacturers measure the size of this cross section and report it in a device's data sheets [4]. The static cross section of an FPGA is independent of a user design, meaning that upsets of stored state can occur in all portions of the device regardless of its configuration. Like SRAM memory, an FPGA's static cross section scales with the size of the device.

Physical structures which contribute to an FPGA's SEU static cross section include configuration memory, user flip-flops, BRAMs, half-latches, etc. The configuration memory is the largest fraction of the cross section. For example, in the Xilinx Virtex XCV1000, a 0.22 μm 5-layer epitaxial process FPGA, there are approximately 5.8×10^6 configuration latches compared to 24×10^3 user flip-flops. Consequently, users are typically most concerned with the configuration memory.

When programmed with a specific circuit design, an FPGA's configuration memory has a unique dynamic cross section. A design's utilization of programmable routing, logic, and I/O resources determines its dynamic cross section. Since a design never uses all of an FPGA's resources, the dynamic cross section is generally much smaller than the static cross section.

A. Measuring Dynamic Cross Section

Since the dynamic cross section varies for each design, it is useful to measure cross section of a specific design. A dynamic

0018-9499/$20.00 © 2005 IEEE

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 22, 2009 at 16:56 from IEEE Xplore. Restrictions apply.

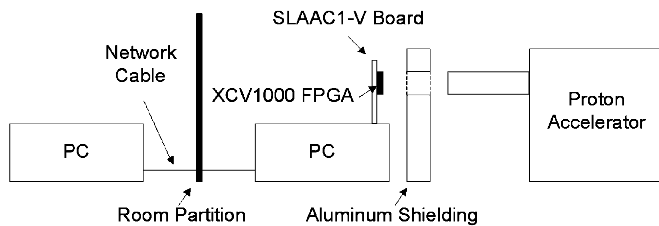

Fig. 1. Test setup at Crocker Nuclear Laboratory, Davis, CA. (63 MeV protons).

cross section estimate allows a user to predict mean time between failures (MTBF) for specific environments and to evaluate the effectiveness of an SEU mitigation technique.

We have developed two techniques for measuring dynamic cross section. We use fault-injection to predict, or estimate, a design's cross section. We use proton irradiation to validate our predictions and to also make more formal measurements.

For fault-injection, we use a tool developed at Brigham Young University [9]. This tool estimates the dynamic cross section for a given FPGA design by artificially upsetting individual bits within the configuration memory of a device under test (DUT). The tool identifies which configuration bits, when upset, cause any type of circuit output error. This tool can rapidly test all configuration bits in a bitstream to create an accurate and complete characterization of a given FPGA design. The size of the dynamic cross section is equal to the fraction of "sensitive" configuration bits multiplied by the device static cross section.

We also use proton testing to make more formal measurements of dynamic cross section. At Crocker Nuclear Laboratory in Davis, CA, we use 63 MeV protons to induce SEUs. The physical test setup is illustrated in Fig. 1. Here, our test fixture records data about the time and location of configuration upsets, in addition to the time of output errors. The size of the dynamic cross section is equal to the number of configuration upsets which cause output errors divided by the product of the total fluence and incident particle angle.

B. Dynamic Cross Section Measurements

Fault-injection and proton irradiation were used to estimate the dynamic cross section for many FPGA designs. The results are summarized in columns two and three of Table I. This table reports the dynamic cross section of the design as a *fraction* of the total device static cross section. For example, the dynamic cross section of a DSP Kernel design is approximately one-tenth the size of the 1.28×10^{-7} cm^2 [4] static cross section of a Xilinx Virtex XCV1000 FPGA.

A careful examination of Table I, reveals that our tool is a good estimate of dynamic cross section. In fact, previous work has shown that the error of our tool compared to proton testing is within 1% [5].

A useful by-product of estimating cross section is the ability to evaluate the effectiveness of design mitigation techniques such as TMR. For example, full TMR[1] was applied to a simple counter design. We then used fault-injection and proton irradiation to measure the effective reduction in cross section. Table I shows that the dynamic cross section of the counter design is almost two orders of magnitude smaller after TMR was applied.

[1]It should be noted that here and throughout this paper, the "full TMR" we used does not triplicate the clock, reset and I/O signals.

TABLE I
RATIOS OF THE DYNAMIC AND PERSISTENT CROSS SECTION TO THE STATIC DEVICE CROSS SECTION† FOR SELECTED DESIGNS

Design	Dynamic Fraction of Static Cross Section†		Persistent Fraction of Static Cross Section†	
	Predicted (Injection)	Measured (Accel.)	Predicted (Injection)	Measured (Accel.)
Multiplier	9.5×10^{-2}	6.5×10^{-2}	0‡	0‡
Counter	3.5×10^{-2}	2.7×10^{-2}	1.9×10^{-2}	1.3×10^{-2}
TMR Counter	1.9×10^{-4}	1.2×10^{-4}	1.7×10^{-7}	0‡
DSP Kernel	8.6×10^{-2}	7.0×10^{-2}	1.5×10^{-3}	8.9×10^{-4}
Partial TMR DSP Kernel	8.5×10^{-2}	6.6×10^{-2}	1.9×10^{-5}	2.7×10^{-5}

† The XCV1000 has a static proton cross section of 1.28×10^{-7} cm^2.
‡ No events were observed.

III. PERSISTENT FPGA CROSS SECTION

Typically, users are most interested in measuring the dynamic cross section of an FPGA design because it measures fluence to interruption of service. This section will show that there are actually different modes of disruption. As a result, the dynamic cross section can be subdivided into smaller components.

Separation of cross section is not a new idea. Turflinger *et al.* reported multiple cross sections from different error modes in an ADC [10]. Others have proposed separating the Single Event Functional Interrupts (SEFI) cross section from the dynamic SEU cross section. Our approach simply divides the dynamic cross section into persistent and nonpersistent components, each characterized by the duration of a service interruption induced by an SEU in the respective cross section component.

To the user, a service disruption means lost or corrupted data, or in other words, functional errors. The persistent and nonpersistent dynamic cross sections are characterized by the duration of the functional errors induced by an SEU in the respective cross section. Nodes which, when upset, cause persistent functional errors, or a permanent interruption of service, are part of the persistent dynamic cross section. Upset nodes which induce nonpersistent functional errors, or a temporary disruption of service, belong to the nonpersistent dynamic cross section.

A key concept related to error persistence is configuration scrubbing. This section will begin by describing this mitigation technique. We will then introduce nonpersistent and persistent functional errors.

A. Configuration Scrubbing

One of the most common SEU mitigation techniques for FPGAs is configuration scrubbing [11]. Scrubbing prevents the buildup of configuration faults by repeatedly scanning and cleaning configuration upsets. External scrubbing circuits are added to a system to read the configuration memory, compare it against a golden copy, and repair the configuration when faults are found. The process is repeated throughout the configuration memory in a round-robin manner.

Unrepaired, an upset configuration bit in the dynamic cross section can induce functional errors indefinitely. Configuration scrubbing reduces the duration of dynamic functional errors and eliminates the buildup of multiple faults. Furthermore, systems

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 22, 2009 at 16:56 from IEEE Xplore. Restrictions apply.

must employ configuration scrubbing before any reliability enhancement is observed for circuit mitigation techniques such as TMR. Without scrubbing, these techniques are ineffective. The buildup of multiple faults would eventually break the redundancy.

Although upsets within the configuration memory will be repaired through scrubbing, there is a finite time between a configuration fault and the corresponding repair. Upsets among the sensitive configuration memory bits will cause functional faults and design failures during this time period. The circuit will operate incorrectly and possibly produce incorrect data for at least as long as it takes to detect and repair the configuration fault through scrubbing.

B. Nonpersistent Errors

The use of configuration scrubbing limits the time in which an SEU-induced fault is present in an FPGA's configuration memory. Once a configuration scrub has completed, all faults in the bitstream have been repaired and the circuit's original configuration has been restored. In many cases, the functional output errors that occur after an SEU are temporary. Once the configuration fault has been repaired, there is no sign of system failure. In other cases, the functional output errors persist indefinitely beyond repair. The concepts of persistence and nonpersistence are based on this idea that, in a system with scrubbing, the duration of some functional errors is finite.

Temporary functional errors are termed *nonpersistent* because they do not persist in a design and are flushed after configuration scrubbing. Once the errors have flushed, the system exhibits no signs of failure. These transient errors represent a temporary interruption of service. The system does not require a reset to recover. We will refer to configuration faults which cause nonpersistent errors as nonpersistent upsets.

The concept of nonpersistent errors can be demonstrated by an example. Two data streams were collected from two identical circuits operating within our fault-injection tool. The arithmetic difference between these two data streams can be computed to identify the impact of configuration upsets. When the two circuits are operating correctly, the arithmetic difference is zero. When one of the circuits has functional errors, the arithmetic difference is nonzero.

Fig. 2 illustrates the arithmetic difference between two data streams generated by identical designs. The figure depicts a nonpersistent error. For the first 64 time steps, the difference between the two data streams is zero indicating that both circuits are operating in sync. At time step 65 a nonpersistent configuration bit is upset in one circuit. This upset causes the corrupted circuit to operate incorrectly. The faulty operation induces functional errors and immediately the outputs diverge. At time step 130 the bitstream is repaired through configuration scrubbing. At this point the corrupted circuit resumes proper operation and shortly thereafter the difference between the data streams returns to zero. There are no longer any signs of functional errors. In this example, the nonpersistent configuration upset caused a temporary interruption of service from time step 65 to 130.

A circuit which exhibits the characteristics of nonpersistence is a feed-forward multiplier. The circuit computes the product between two binary values. The output only depends on the *current* inputs. If an SEU occurs anywhere within the dynamic cross section of the design, the circuit will temporarily compute products incorrectly (temporary service interruption). However, after scrubbing repairs the fault, the multiplier circuit will be repaired and begin to correctly perform multiplication.

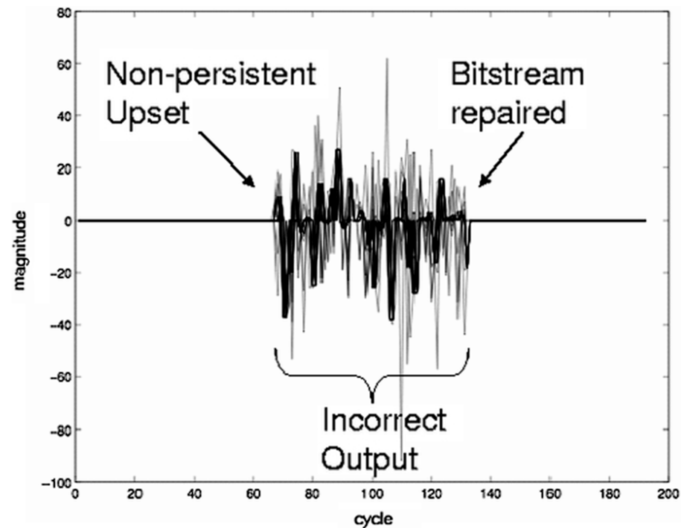

Fig. 2. Plot of the difference between the outputs of a DUT and golden circuit before, during and after an upset within the nonpersistent cross section.

C. Persistent Errors

Sometimes an SEU-induced fault within the configuration memory of an FPGA will introduce functional errors which indefinitely propagate within a circuit, even after configuration scrubbing repairs the fault. In these cases, the fault happened within the persistent cross section, introducing persistent functional errors. Unlike nonpersistent errors, persistent errors do not disappear after configuration scrubbing. Although scrubbing repairs the circuit structure, the temporary circuit failure inserts incorrect state into the system that cannot be corrected, and will not self-correct, without a global system reset. To the user, persistent functional errors look like a SEFI. However, persistent functional errors are specific to the configuration programmed into the FPGA, not to the specific device. In addition, persistent errors can be removed with a global system reset while a SEFI usually requires a system power off/on to recover [12].

Persistent errors are caused by an SEU within the configuration memory corresponding to circuit structures that contain feedback and store internal state. The feedback structures "trap" the incorrect state and store this erroneous state until appropriate reset measures are taken. We will refer to configuration faults which cause persistent errors as persistent upsets.

Fig. 3 illustrates a persistent error, or permanent service interruption. Like the nonpersistent example in Fig. 2, the output stream for the two circuits match for the first 64 time steps. At time step 65 a bit in the persistent cross section of one circuit was upset. Immediately the arithmetic difference becomes nonzero. At time step 130 the bitstream is repaired through configuration scrubbing. Unlike the nonpersistent example, the output streams in this example did not converge. The internal state trapped the

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 22, 2009 at 16:56 from IEEE Xplore. Restrictions apply.

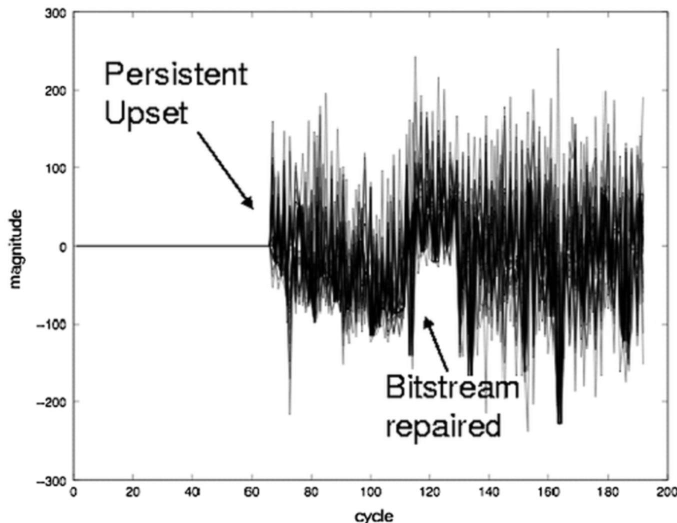

Fig. 3. Plot of the difference between the outputs of a DUT and golden circuit before, during and after an upset within the persistent cross section.

errors and continued to propagate them even after the configuration bit was repaired. The application continued to produce faulty data after repair and will therefore need a system reset to recover.

The exact error signatures in Figs. 2 and 3, both in magnitude and time, are a function of the design used as well as the upset bit and set of input vectors. In a general sense, however, the error signatures represent the typical effect of a nonpersistent versus persistent upset. The difference between the outputs of the DUT and golden circuit will always return to zero after a nonpersistent upset, whereas after a persistent upset, the difference will indefinitely remain nonzero.

A circuit which exhibits the characteristics of persistence is a binary counter. A counter depends on the count of the *previous* cycle to compute the next value in the counting sequence. If an SEU occurs within the persistent cross section of the design, the counter will likely start counting out of sequence. Even after scrubbing repairs the circuit fault, the counter will likely not return to the correct counting sequence (permanent service interruption). The internal state will propagate the error indefinitely without the intervention of an external reset.

IV. Measuring Persistent Cross Section

Two of our main goals in testing were to validate the existence of a persistent cross section and estimate its size for several designs. In this section we will explain how we measured the persistent cross section using our two testing methodologies, fault-injection and proton irradiation. We will also report the results of our tests.

In this and the following sections we will refer to several designs we used in our testing. The first is an array of feed-forward multipliers with no internal state. The second is a large array of 8-bit counters (each containing count state). The third is a mitigated implementation of the 8-bit counter array using full TMR. The fourth design is a digital signal processing (DSP) kernel developed at Los Alamos National Laboratory. The final design is a mitigated implementation of the DSP Kernel, with TMR

TABLE II
Device Utilization for Selected Designs†

Design	Device Utilization	
	Slices	%
Multiplier	10, 305	83%
Counter	2, 151	17%
TMR Counter	11, 251	91%
DSP Kernel	5, 775	46%
Partial TMR DSP Kernel	7, 936	64%

† The device used was a Xilinx Virtex XCV1000 which has 12,288 available slices.

applied to just circuit elements in the persistent cross section. Table II lists the various designs and their utilization of logic in the Xilinx Virtex XCV1000 FPGA.

A. Testing Methodologies

We extended the fault-injection tool mentioned in Section II-A to predict the size of a design's persistent cross section. In addition to sensitive bits, the tool now also identifies configuration bits which, when upset, induce a persistent error. These bits constitute the persistent cross section.

The time-line in Fig. 4 shows the sequence of events used to determine if an individual configuration memory bit contributes to persistent errors. At the beginning of the test, marked by the diamond, the tool emulates an SEU by corrupting a bit in the configuration memory. Some of those result in a dynamic error. The delay t_s is introduced to emulate the average time required for scrubbing to repair the bitstream, or mean time to repair (MTTR). Next, the bit is corrected. Finally, if functional errors occurred, the design is allowed to operate for an additional finite amount of time t_f to see if the errors flush. If any errors persist then the originally corrupted bit is classified as contributing to persistent errors. The size of the persistent cross section is equal to the fraction of persistent configuration bits multiplied by the device static cross section.

We also measured the persistent cross section at Crocker Nuclear Laboratory in Davis, CA, using proton irradiation and a method very similar to measuring the dynamic cross section. A 63 MeV proton source caused the SEUs. The time of all upsets and output errors was recorded. In addition to recording the time of all output errors, the record of each output error was appended with a flag indicating if the error persisted or not.

Fig. 5 illustrates the sequence of events used to determine if a single configuration bit contributes to persistent errors. To compensate for the dynamic length of the scrub time t_s, we used an extended flush time t_f to cover both the worst-case (longest) scrub time and the minimum alloted time to allow functional errors to flush. Errors which persisted beyond the flush time t_f were classified as persistent. Post processing of data matched persistent errors to configuration upsets [13]. The size of the persistent cross section is equal to the number of configuration upsets which cause persistent output errors divided by the product of the total fluence and incident particle angle.

Although the persistence fault-injection tool and radiation testing environment operate almost identically, there are some known differences that will affect the testing results. The first difference between the two testing methodologies is the number of the configuration bits tested. The fault-injection tool will test

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 22, 2009 at 16:56 from IEEE Xplore. Restrictions apply.

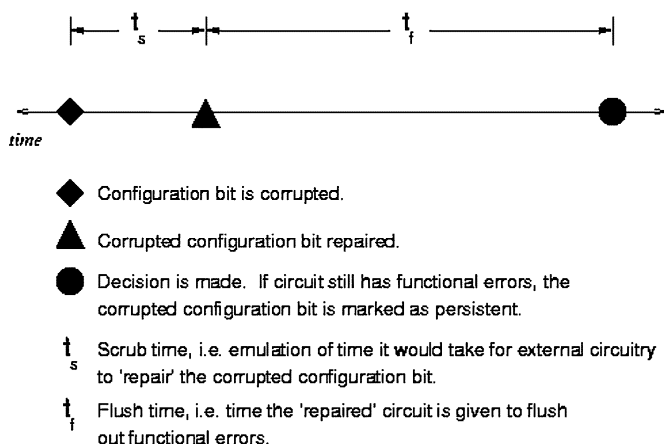

Fig. 4. Fault-injection test time-line: Sequence of events in a single trial to test a configuration bit for persistence.

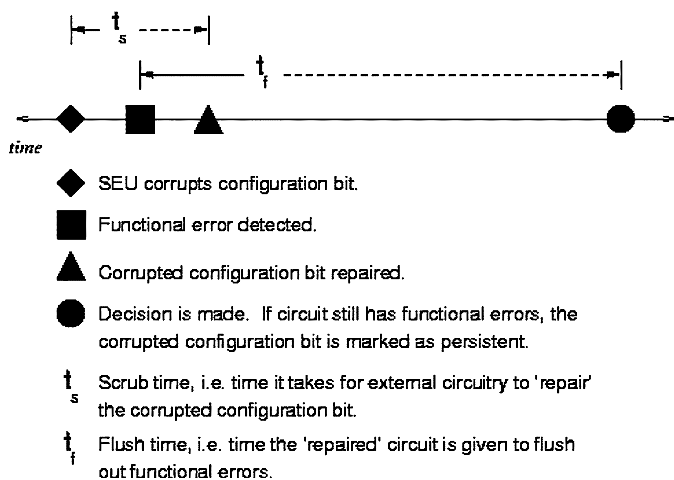

Fig. 5. Proton-irradiation test time-line: Sequence of events in a single trial to test a single configuration bit for persistence.

every configuration bit while the radiation test will only test a small subsection of the configuration memory. Sampling limitations of the radiation test may produce slightly different results from the exhaustive fault-injection tool.

The second difference between the two testing methodologies is the timing of the configuration upsets. In the fault-injection tool, the configuration upsets are carefully controlled and synchronized with the run time and flush time of the persistence test. The arrival of proton upsets in the radiation test, however, is random and cannot be correlated with run time and flush time. Although the average arrival time of protons can be controlled by the proton flux, the inter arrival of protons will follow a Poisson distribution. It is expected that secondary configuration upsets will occur during the flush time t_f of a trial. If this does occur, the first upset configuration bit will falsely be tagged as persistent. The number of such false persistent events can be estimated through statistical analysis.

After accounting for the known error in radiation testing, the data we collected indicates that fault-injection is a reliable method of measuring the persistent cross section. Columns 4 and 5 in Table I list the ratio of the persistent cross section to the device static cross section for several designs. Predicted and measured values are reported. In all cases, fault-injection is at least as good as a first-order estimate of the persistent cross section.

TABLE III
RAW DATA COLLECTED AT CROCKER NUCLEAR LABORATORY, DAVIS, CA

Design	SEUs	Dynamic Functional Errors	Persistent Functional Errors
Multiplier	5,927	597	0
Counter	21,068	833	347
TMR Counter	28,656	5	0
DSP Kernel	30,623	2,737	35
Partial TMR DSP Kernel	28,074	2,501	1

B. Persistent Cross Section Measurements

Using both fault-injection and proton irradiation we found that, as expected, the persistent cross section does exist. We also found that, for our applications, the persistent cross section is significantly smaller than the overall dynamic cross section. Table III lists the "raw" data values collected at Crocker laboratory. Trials which had multiple upsets (SEUs) during the test flush time were removed [13].

In Table I we list both the dynamic and persistent cross section as a ratio of the overall static cross section. Both the predicted and measured values are given. For all the designs, the dynamic cross section is at least one order of magnitude smaller than the static cross section. Of course, the relative size of the dynamic cross section depends on the application's resource utilization. Our measurements also indicate that the persistent cross section is significantly smaller than the dynamic cross section. Again, the absolute size is application dependent.

To emphasize the relative magnitude of the persistent to dynamic cross section and its application dependence, we plotted the resource layout for the DSP Kernel and non-TMR counter array designs along with a graphical representation of the dynamic and persistent cross sections of those resources in Figs. 6 and 7.

In Figs. 6 and 7, the left graphic represents the design layout and resource utilization as rendered by the Xilinx FPGA Editor tool. The center and right graphics represent the fraction of configuration bits which constitute the dynamic and persistent cross sections respectively. These diagrams were created using Matlab in conjunction with the fault-injection tool mentioned in Section II.

The Counter Array design largely consists of feedback structures so it is not surprising that its persistent cross section nearly mirrors its dynamic cross section. The DSP Kernel design, on the other hand, has only a few scattered feed-back structures for control. Its persistent cross section is over an order of magnitude smaller than its dynamic cross section. It is not surprising then that the graphical representation of the DSP Kernel persistent cross section is nearly blank.

V. MEAN TIME BETWEEN FAILURES

An important motivation for measuring cross section is to determine how often a given system will fail. Like dynamic and

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 22, 2009 at 16:56 from IEEE Xplore. Restrictions apply.

TABLE IV
MEAN TIME BETWEEN FAILURE FOR A DSP KERNEL DESIGN

Orbit	Alt. (km)	Inc. (deg)	Intolerant DSP Kernel MTBF (days)			Tolerant DSP Kernel MTBF (days)			Tolerant, Partially Mitigated DSP Kernel MTBF (days)		
			Typical Solar Min	Stormy Solar Max	Worst Day Solar Max	Typical Solar Min	Stormy Solar Max	Worst Day Solar Max	Typical Solar Min	Stormy Solar Max	Worst Day Solar Max
LEO	560	35.0^o	17.5	26.3	25.1	980.3	1.5×10^3	1.4×10^3	7.9×10^4	1.2×10^5	1.1×10^5
Polar	833	98.7^o	7.0	8.8	0.1	390.0	494.5	7.2	3.1×10^4	4.0×10^4	580.8
GPS	22,200	55.0^o	10.8	10.4	3.6×10^{-2}	604.2	584.3	2.0	4.9×10^4	4.7×10^4	164.2
GEO	36,000	0.0^o	10.7	8.7	3.6×10^{-2}	597.8	488.1	2.0	4.8×10^4	3.9×10^4	162.3

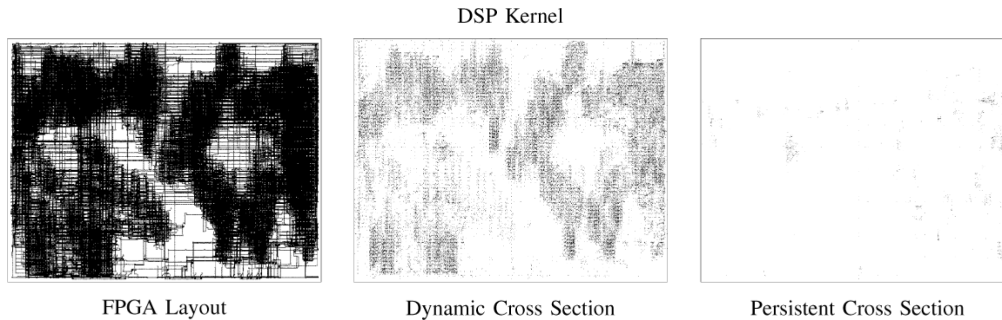

Fig. 6. The diagram on the left is a screen capture of the layout of the DSP Kernel design. The center and right diagrams are graphical representations of the portion of the DSP Kernel design layout which constitute the dynamic and persistent cross sections respectively.

persistent cross section, MTBF is application dependent. However, MTBF also depends on the destined system environment, or more specifically the environment's SEU rate. Additionally, MTBF depends on the application's definition of failure. Some applications may have a higher tolerance of service interruptions. A user can leverage a design's persistent cross section and its tolerance level to improve MTBF at a lower cost.

In this section we will first introduce the different tolerance levels of application service interruption. Next, we will estimate the static SEU rate for several orbits. We will then forecast MTBF for the DSP Kernel application. Finally, we will show how we made low-cost improvements of MTBF for the DSP Kernel by mitigating just its persistent cross section.

A. Application Service Interruption Tolerance

Most applications cannot tolerate permanent service interruptions (persistent errors), but some applications can tolerate temporary service interruptions (nonpersistent errors). To aid our discussion we will classify applications as either tolerant or intolerant, based on their tolerance of service interruptions.

"Intolerant" applications *cannot* tolerate temporary *nor* permanent service interruptions. Either type of service interruption would be considered an application failure. Because nonpersistent upsets cause temporary service interruptions and persistent upsets cause permanent service interruptions, it follows that any dynamic upset will cause an intolerant system to fail. In addition to scrubbing, a mitigation technique such as TMR would need to be applied to the entire circuit to insure uninterrupted service [2].

"Tolerant" applications *can* tolerate temporary service interruptions. As Fig. 2 illustrates, these service interruptions simply cause a temporary loss of data. Put another way, an application is tolerant if it can withstand temporary data loss. Only permanent service interruptions cause a tolerant application to fail. It then follows that only persistent upsets cause a tolerant application to fail. We intend to show that *only* circuit components in the persistent cross section need mitigation like TMR to eliminate these failures.

It is important to point out the trade-offs made by treating an application as tolerant. The benefit of treating an application as tolerant is that failure only occurs after persistent upsets. In many designs the persistent cross section is orders of magnitude smaller than the dynamic cross section. Consequently, a tolerant application, even without any mitigation, will fail less often. On the other hand, even though nonpersistent upsets will not cause "failure" in a tolerant application, data will be lost after all nonpersistent upsets (see Fig. 2).

The ability to tolerate temporary service interruptions, i.e., data-loss, is application specific. Ultimately an application's tolerance level depends on the criticality of a continuous stream of uncorrupted output data. The user must decide an application's level of tolerance because it directly affects MTBF and the amount of SEU mitigation required.

B. On-Orbit SEU Rate

It is useful to estimate and contrast MTBF for both tolerant and intolerant applications. In order to predict MTBF it is necessary to first know the static SEU rate for the destined environment. To estimate static SEU rates (SEU/hour) we modeled the environment energy spectra for a few sample orbits. For the trapped proton and solar proton environments we used the AP-8 and JPL models respectively. For the heavy ion environment we used the CREME96 model.[2] We then used the Xilinx static proton and heavy-ion cross section data for a single Xiliinx Virtex XCV1000 reported in [4], combined with the forecast energy spectra, to predict static SEU rates for our sample orbits [15].

[2]A detailed explanation of the process and software we used to predict static SEU rates can be found in [14].

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 22, 2009 at 16:56 from IEEE Xplore. Restrictions apply.

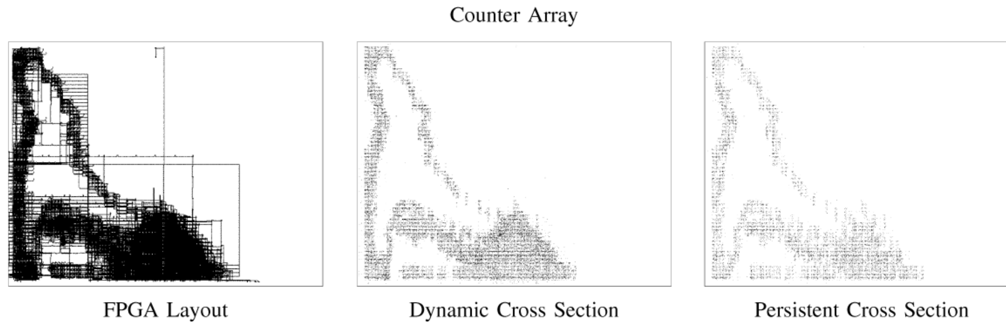

Fig. 7. The diagram on the left is a screen capture of the layout of the Counter Array design. The center and right diagrams are graphical representations of the portion of the Counter Array design layout which constitute the dynamic and persistent cross sections respectively.

TABLE V
STATIC SEU RATE FORECAST FOR A SINGLE XILINX VIRTEX XCV1000

Orbit	Alt. (km)	Incl. (deg)	Typical Solar Minimum (SEU/hr)	Stormy Solar Maximum (SEU/hr)	Worst Day Sol. Max. (SEU/hr)
LEO	560	35.0^o	2.8×10^{-2}	1.8×10^{-2}	1.9×10^{-2}
Polar	833	98.7^o	6.9×10^{-2}	5.5×10^{-2}	3.8
GPS	22,200	55.0^o	4.5×10^{-2}	4.6×10^{-2}	$1.3 \times 10^{+1}$
GEO	36,000	0.0^o	4.5×10^{-2}	5.5×10^{-2}	$1.3 \times 10^{+1}$

TABLE VI
MEAN TIME BETWEEN DATA-LOSS FOR A DSP KERNEL DESIGN

Orbit	Alt. (km)	Inc. (deg)	Tolerant DSP Kernel MTBL (days)		
			Typical Solar Min	Stormy Solar Max	Worst Day Solar Max
LEO	560	35.0^o	17.8	26.7	25.6
Polar	833	98.7^o	7.1	9.0	0.1
GPS	22,200	55.0^o	11.0	10.6	3.7×10^{-2}
GEO	36,000	0.0^o	10.8	8.9	3.7×10^{-2}

Table V lists our predicted static SEU rates in solar minimum, stormy (average flare flux) solar maximum and worst day (peak flare flux) solar maximum conditions. From the static SEU rate, one can derive the dynamic and persistent SEU rates, and subsequently MTBF, for a specific application.

C. On-Orbit MTBF

Using the static SEU rates from Table V and our estimated cross section measurements from Table I, we predicted MTBF for the DSP Kernel application. Our predictions are summarized in Table IV. The first set of values (columns 4–6) in Table IV corresponds to an intolerant scenario. The second set of values (columns 7–9) in Table IV corresponds to a tolerant scenario. In both cases, the system *only implements configuration scrubbing*.

Since intolerant applications are service interruption averse, they will "fail" after all upsets. A tolerant application, on the other hand, will only "fail" after persistent upsets, or permanent service interruptions. As an intolerant application, the DSP Kernel on a system in a low-earth orbit (LEO) at 560 km altitude and 35.0° inclination will fail once every 17 days during Solar Minimum. As a tolerant application in the same orbit, it will only fail once every 980 days.

D. Increasing MTBF Through Partial Mitigation

Increasing an application's MTBF requires some form of mitigation. However, increasing a tolerant application's MTBF does not require a comprehensive mitigation strategy such as full TMR. By applying TMR to just circuit components within the persistent cross section of a design that is tolerant of temporary service interruptions, a significant increase in MTBF can be realized at a much lower cost than full TMR.

The final set of values (columns 10–12) in Table IV correspond to a tolerant scenario for the DSP Kernel, but with TMR applied to its persistent components. As a result, nearly all configuration upsets will only cause temporary service interruptions. In the same 560 km orbit, we predict that the DSP Kernel, as a tolerant partially mitigated application, will only fail once every 216 *years*, a factor of approximately 80 less often.

It is important to analyze the trade-offs made by applying TMR to just the persistent cross section. Since full TMR and other comprehensive mitigation techniques are costly in terms of area and power [1], [16], the positive benefit of partial TMR is a reduction in required mitigation circuitry. For example, Table II shows that a completely unmitigated implementation of the DSP Kernel design utilized 5775 slices. Full TMR would require at least a 200% increase. A partial TMR implementation, on the other hand, needed only 7936 slices, or a 37% increase.

The negative trade-off for only applying partial TMR, is that the nonpersistent cross section is still vulnerable to SEUs. However, in a tolerant application scenario, the system will only temporarily lose data after nonpersistent upsets.

Since a tolerant application will lose data after nonpersistent upsets, it is useful to also forecast the average number of days between data-loss. Table VI reports the mean time between data-loss (MTBL) for the DSP Kernel in a tolerant scenario. In the 560 km orbit example, we predict that the DSP Kernel, as a tolerant application, will lose data once every 17.8 days. Note that it does not make sense to predict data-loss for an intolerant application because, by definition, data-loss means failure.

Clearly an application's design, tolerance level and destined orbit strongly affect its failure and data-loss rates; and subsequently its SEU mitigation requirements. Some applications will require more extensive mitigation strategies while others stand to realize tremendous savings in area and power by limiting mitigation.

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 22, 2009 at 16:56 from IEEE Xplore. Restrictions apply.

VI. Conclusion

In this paper, we have reviewed the dynamic SEU cross section of an FPGA design. For our applications, we have confirmed that the dynamic cross section is much smaller than the device static cross section. Further, the dynamic cross section is different for each FPGA design and can be characterized using our fault-injection tool or performing a proton radiation test. We have performed these tests for a variety of designs and have shown that our fault-injection provides a suitable alternative to proton radiation tests.

We have also confirmed that the dynamic cross section can be divided into two different categories. The persistent component corresponds to those upsets that cause trapped errors, or in other words a permanent service interruption. Only a global reset or global reconfiguration, in addition to configuration scrubbing, can flush such errors. Nonpersistent errors or temporary service interruptions, however, are flushed from a design through traditional configuration scrubbing and may not pose a problem for certain FPGA applications.

The presence of a persistent and consequently a nonpersistent dynamic cross section has a significant impact on the cost of mitigation. If an FPGA design can tolerate temporary nonpersistent errors, then comprehensive mitigation such as TMR may not be needed. Instead, mitigation can be limited to the persistent cross section that introduces trapped errors into the system. Tolerant FPGA designs may even forgo comprehensive mitigation if forecast dynamic SEU rates are low enough.

Since the persistent component of the dynamic cross section is substantially smaller than the overall dynamic cross section, limited TMR can significantly reduce the cost of mitigation. In the future, a tool that can identify and mitigate just circuit elements in the persistent cross section may save resources substantially over comprehensive mitigation.

Acknowledgment

The authors would like to acknowledge the support of J. Fabula, H. Bogrow, and others at Xilinx.

References

[1] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," in *Proc. 6th Annu. Int. Conf. Military and Aerospace Programmable Logic Devices (MAPLD)*, NASA Office of Logic Design, AIAA, Washington, D.C., Sep. 2003, p. P63.

[2] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," Xilinx Corp., Tech. Rep. xAPP197 (vl.O), Nov. 1, 2001.

[3] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy for SEU mitigation in FPGAs," in *Proc. 6th Annu. Int. Conf. Military and Aerospace Programmable Logic Devices (MAPLD)*, NASA Office of Logic Design, AIAA, Washington, D.C., 2003, p. Cl.

[4] "Qpro Virtex 2.5v radiation hardened FPGAs," Xilinx Corp., Tech. Rep., dS028 (vl.2), Nov. 5, 2001.

[5] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 6, pp. 2147–2157, Dec. 2003.

[6] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham, "The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets," in *IEEE Symp. Field-Programmable Custom Computing Machines*, K. Pocek and J. Arnold, Eds. Napa, CA, Apr. 2003, pp. 133–142.

[7] E. Johnson, K. Morgan, N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Detection of configuration memory upsets causing persistent errors in SRAM-based FPGAs," in *Proc. MAPLD Conf.*, Sep. 2004. Paper P135.

[8] A. Holmes-Siedle and L. Adams, *Handbook of Radiation Effects*, 2nd ed. Oxford, U.K.: Oxford Univ. Press, 2002.

[9] E. Johnson, M. J. Wirthlin, and M. Caffrey, "Single-event upset simulation on an FPGA," in *Proc. Int. Conf. Engineering of Reconfigurable Systems and Algorithms (ERSA)*, T. P. Plaks and P. M. Athanas, Eds, Jun. 2002, pp. 68–73.

[10] T. L. Turflinger and M. V Davey, "Understanding single event phenomena in complex analog and digital integrated circuits," *IEEE Trans. Nucl. Sci.*, vol. 37, no. 6, pp. 1832–1838, Dec. 1990.

[11] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," Xilinx Corp., Tech. Rep. xAPP216 (vl.O), Jun. 1, 2000.

[12] R. Baumann, "Single-event effects in advanced cmos technology," presented at the IEEE NSREC Short Course, Seattle, WA, Jul. 2005.

[13] Correlation of fault-injection to proton accelerator persistent cross section measurements. (2005, Jul.). [Online]. Available: http://dspace.byu.edu

[14] Predicting on-orbit SEU rates. (2005, Jul.). [Online]. Available: http://dspace.byu.edu

[15] 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 *Proc. 3rd Annu. Conf. Military and Aerospace Programmable Logic Devices (MAPLD)*, 2000, p. P30.

[16] N. Rollins, M. Wirthlin, and P. Graham, "Evaluation of power costs in triplicated FPGA designs," in *Proc. MAPLD Conf.*, Sep. 2004. Paper P136.

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 22, 2009 at 16:56 from IEEE Xplore. Restrictions apply.