SAVE PRINT

LA-UR-08-04992

Approved for public release; distribution is unlimited.

Title:

Xilinx Virtex FPGA Design Guide for Space

Author(s):

Heather Quinn and Paul Graham

Intended for:

FPGA Mission Assurance Center report
Albuquerque, NM

Los Alamos
NATIONAL LABORATORY
EST. 1943

Form 836 (7/06)

SAVE PRINT CLEAR FORM

Xilinx Virtex FPGA Design Guide for Space

July 29, 2008

Contents

List of Tables

List of Figures

[1]Note that any time in the design flow the designer may be required to return to these steps to select a different method of detection, mitigation, or repair due to factors such as time constraints, space requirements, etc.

Acknowledgements

We would like to acknowledge the support of several organizations for this work. First, the majority of the work reported on in this report has been sponsored by Department of Energy's NA-22 Non-proliferation Research and Development Office. This sponsorship has been through the Deployable Adaptive Processing Systems (DAPS) Project, the Cibola Flight Experiment (CFE) Project, the Sensor-Oriented Processing and Networking (SOPraNo) Project, and the MRM-LANL project. We would also like to thank the Air Force Research Laboratory—Kirtland for their valuable support in preparing this document and funding additional research in this area of FPGA mission assurance. Other sponsors of this work has included NASA Goddard Space Flight Center through the Reconfigurable Hardware in Orbit (RHinO) project and other sponsorship.

In addition to our sponsors, we would like to acknowledge our many collaborators both at LANL and external to LANL. At LANL, we would like to thank Michael Caffrey and Keith Morgan for their tremendous and groundbreaking work in this area in addition to their guidance and support. We would also like to acknowledge Xilinx for their strong support of this work and their willingness to collaborate in this research. Mike Wirthlin and his many excellent students from Brigham Young University have contributed and continue to contribute heavily to our research in this area. Other organizations that have contributed and helped include Aerospace Corporation, NASA's Goddard Space Flight Center, NASA's Jet Propulsion Laboratory, SEAKR Engineering, and the Xilinx Radiation Test Consortium.

Chapter 1

Scope

1.1 Identification

The "Xilinx Virtex FPGA Design Guide for Space" was written by Los Alamos National Laboratory's (LANL) International Space and Response Division, Space Data Systems Group (ISR-3) in response to the needs of the aerospace community.

1.2 Document Maintenance

This document is maintained at Los Alamos National Laboratory's (LANL) International Space and Response Division, Space Data Systems Group (ISR-3). All suggestions for updates, items of correction, and document maintenance issues should be issued to the following:

```
Paul Graham
Mail Stop: D440
ISR-3 Space Data Systems
Los Alamos National Laboratory
Los Alamos, 87545
grahamp@lanl.gov

Heather Quinn
Mail Stop: D440
ISR-3 Space Data Systems
Los Alamos National Laboratory
Los Alamos, 87545
hquinn@lanl.gov
```

1.3 Document Overview

This design guide seeks to provide a set of guidelines and techniques the International Space and Response Division (ISR) at Los Alamos National Laboratory has deemed helpful in the task of detecting, mitigating, and resolving single-event effects in SRAM based FPGAs for space-borne missions.

1.4 Terminology

Terminology	
BIST	Built-in self test
BRAM	Block SelectRAM
BRAMi	Block SelectRAM interconnect
CFE	Cibola Flight Experiment
CFESat	Cibola Flight Experiment Satellite
CLB	Configurable logic block
-	Configuration Data Frame
COTS	Commercial off the shelf
CRC	Cyclic Redundancy Check
CREME96	Cosmic Ray Effects on Micro-Electronics (1996 Revision)
DARPA	Defense Advanced Research Projects Agency
DCE	Domain Crossing Event
DOE	Department of Energy
DUT	Device Under Test
ELDRS	Enhanced low dose-rate sensitivity
-	Full (Re)configuration
IOB	Input/Output Logic Block
JPL	Jet Propulsion Laboratory
LANL	Los Alamos National Laboratory
LUT	Lookup table
MBU	Multi-bit upset
MTBF	Mean time between failures
NRE	Non-Recurring Engineering
NRL	Naval Research Laboratory
-	Persistent Cross-Section
PRC	Partial (Re)configuration
SBU	Single-bit upset
SEE	Single-event effect
SEFI	Single-event functional interrupt
SET	Single-event transient
SEU	Single-event upset
STMR	Selective triple-modular redundancy
TID	Total Ionizing Dose
TMR	Triple-modular redundancy
VDSM	Very deep sub-micron
XTMR	Xilinx Triple-Modular Redundancy

Chapter 2

Introduction

Field-programmable gate arrays (FPGAs) offer a significant advantage over microprocessors for space missions. Since they are well suited to digital signal processing (DSP) applications, they can provide the speedup of custom hardware without the cost of development for application-specific integrated circuits (ASICs). For these missions, FPGAs can provide more computing ability per watt than traditional radiation-hardened microprocessors, so more computing could be moved closer to the sensor.

There are currently two types of FPGAs offered to the space community: one-time programmable (OTP) devices and reprogrammable devices. Both types of devices provide a variety of programmable routing and logic resources to maximize the flexibility of the device to implement the user designs. The OTP devices have a configurable fabric controlled by anti-fuse memory technology and the user circuit is permanently "etched" into the device by programming anti-fuses[1] in the fabric. The reprogrammable devices implement user logic through lookup tables (LUTs) stored in SRAM distributed across the device. By their very nature, OTP devices cannot be altered once they are programmed, but reconfigurable devices can be. Therefore, if there are any changes in the algorithm, either due to problems in the user circuit or evolution of the underlying science, OTP devices cannot adapt to these changes like reprogrammable devices. In this manner, reconfigurable devices can adapt to the changing mission needs whether they are driven by new science capabilities or by degrading sensor capabilities. Finally, most of the OTP devices have a much more limited capacity than reprogrammable devices. Therefore, reprogrammable devices can implement much larger algorithms than OTP devices.

Besides these differences, the more defining characteristic for these two types of devices for the space community is their radiation characterization. Many of the OTP devices are radiation hardened or very radiation tolerant, and their inclusion in space-based missions is commonplace. The reprogrammable devices are radiation tolerant and are only just starting to be used in satellites. While reconfigurable FPGAs might provide an order of magnitude more computational ability than OTP FPGAs, using them in the space environment reliably and fault-tolerantly is challenging. Since reconfigurable devices are an out-growth of commercial product lines, they are not designed for the types of radiation found on orbit [42]. To this end, much of the radiation qualification has been taken on by application engineers and customers interested in using the devices in space.

In terms of the Xilinx Virtex line, heavy ion testing has shown that Xilinx Virtex XQVR300 SRAM-based FPGAs are single-event latch-up (SEL) immune to up to a linear energy transfer (LET) of 125 $\frac{MeV-cm^2}{mg}$, but are sensitive to single-event upsets (SEUs) above a threshold LET of 1.2 MeV-cm^2/ mg with an average saturation cross-section of 9.12×10^{-8} cm^2 per bit. The SEL and SEU information for the Virtex-II and Virtex-4 devices is shown in Table 2.1. In a low earth orbit (LEO), the Cibola Flight Experiment (CFE) with nine Virtex-I devices is expected to have radiation-induced upsets 1.2 times/day in low radiation zones and 9.6 times/day when there are solar flares. The newer Virtex devices have nearly two orders of magnitude more bits than the older Virtex devices and are likely to see more upsets on orbit. While it is highly desirable to exploit dense, dynamically reprogrammable commercial SRAM-based FPGAs, clearly a rigorous regime of fault detection and mitigation must be used before these FPGAs can be deployed on satellites or spacecraft

[1]As the name suggests, anti-fuses do not connect their terminals in their unprogrammed state—they must be programmed to make the connection.

for compute-intensive applications [16].

Device	SEL Immunity $(\frac{MeV-cm^2}{mg})$	Configuration Memory On-set Threshold $(\frac{MeV-cm^2}{mg})$	Configuration Memory Saturation Cross-Section $\frac{cm^2}{bit}$
XC300	125	1.0	9.12×10^{-8}
2XCV1000	160	1.0	4.37×10^{-8}
Virtex-4	90.3	0.5	2.5×10^{-7}

Table 2.1: SEL and SEU data for the Virtex-I to Virtex-4 device [41, 3]

Through a great deal of experimentation and development experience with the Cibola Flight Experiment and the MRM payload, the International, Space, and Response Division (ISR) at Los Alamos National Laboratory has assembled a series of techniques, rules of thumb, and guidelines to improve the reliability of SRAM-based FPGA systems in a space environment. This guide is an attempt at distilling this information into a usable medium for the education of engineers and scientists who are developing FPGA-based applications for space-based computation.

Chapter 3

Dose-related Effects

3.1 Total Ionizing Dose

In an orbital environment, protons and heavy ions from galactic cosmic ray (GCR); UV, X-rays, and solar particles from solar flares; and protons and electrons trapped in the Van Allen radiation belts (see Figure 3.1) [38] constantly interact with the spacecraft. After years of exposure to ionizing radiation, some electronics become less reliable or suffer hard failures. For semiconductor devices, the switching and current characteristics of the device could gradually shift until they can no longer be used. Therefore, space-based devices are tested to measure the dose of ionizing radiation the devices can withstand and still operate reliably. This measurement, know as total ionizing dose (TID), has units of Rads (rad) or the the SI unit Grays (Gy).

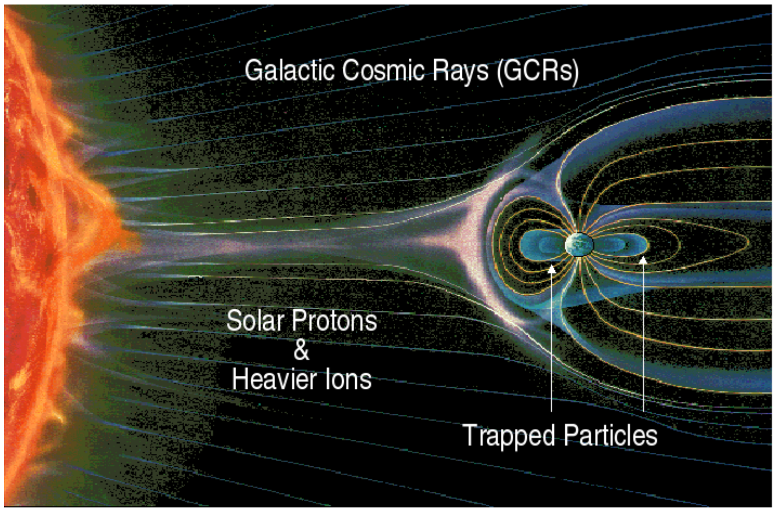


Figure 3.1: Earth's space radiation environment [11][26].

The rate at which a device acquires ionizing radiation (known as the dose rate or gamma-dot phenomena) can have substantial effects on the transistor technology in FPGAs. Specifically, at high dose rates the flux of penetrating radiation introduces a dense plasma of electron-hole pairs in concentrations that are well above the doping densities of most semi-conductor elements. This effectively "swamps" semi-conductor junctions in the FPGA causing photo-currents to be generated that are strong functions of the electric fields present in the device. These photo-generated currents flow in directions that are normally "blocked" and create voltage pulses larger then signals at nodes through out the circuit. Current and voltage spikes of this nature are maintained as long as the FPGA remains exposed to the high-dose-rate radiation environment [10] [21]. Therefore, a device acquiring 100 krads(Si) TID over a long period of time will behave much differently than if it were to acquire 100 krads(Si) in a few seconds.

Low dose rates are also a concern. In devices with a thick oxide enhanced low dose-rate sensitivity (ELDRS)[1] can be problematic. While ELDRS is problematic in bipolar devices, there is some evidence that CMOS devices could also be susceptible [43]. Since the previous investigation into CMOS was conducted on older technology nodes, ELDRS effects needs to be studied in the smaller feature sizes. There is a "suppression of radiation sensitivity" [21] with high dose rates that does not exist at low dose rates. Since the expected on-orbit dose rates are more analogous to ELDER testing than high dose rate testing, testing for ELDRs is essential.

TID testing on the Xilinx XQVR300 device has demonstrated a tolerance in the range of 80 to 100 krads(Si) which meets a variety of orbital applications [14]. Detailed examination of Figure 3.2 shows that rebound and ELDERS effects did not occur in the XQVR300 device despite radiation dose rates varying 4 orders of magnitude at both high and low dose rates [12].

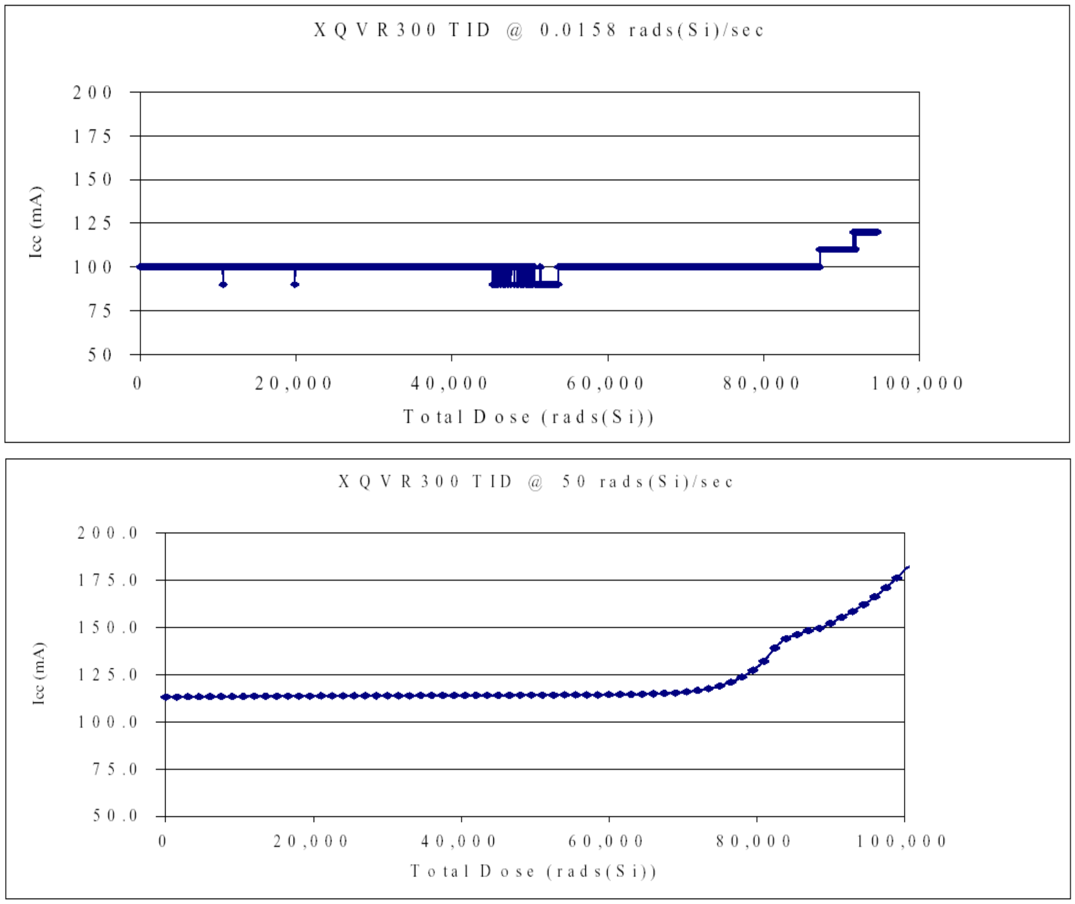


Figure 3.2: XQVR300 TID vs. Leakage Current [12]

.

[1]A discussion of dose-rate dependence and ELDRS effects can be found in the "Handbook of Radiation Effects (second edition)" [21] on page 177.

Chapter 4

Single-Event Phenomena (SEP)

Single-event phenomena (SEP) occur when a single charged particle strikes and deposits energy into space-borne electronics resulting in a fault. There are a number of SEP categories that are relevant to SRAM FPGAs: single-event latchup (SEL), single-event transients (SETs), single-event upsets (SEUs), and single-event functional interrupts (SEFIs). Each phenomenon has different characteristics in terms of observability, consequences, and classification. SEL, or *latchup*, is one of a handful of destructive SEP types and is caused by radiation-induced a positive feedback condition experienced by parasitic transistors in complimentary metal-oxide semiconductor (CMOS) devices. An SET, or *transient*, causes a temporary voltage spike which in turn in CMOS devices can cause temporary changes transistor's output that will dissipate from the system in a short time period, while an SEU, or *upset*, is an event that has inverted the state of a memory cell than cannot be removed without refreshing the original value. Upsets are a particular problem for SRAM-based FPGAs, since SRAM bits are used to control and implement logic functions, routing, and even the voltage rails used in I/O. SEFIs are upsets that occur to part of the FPGA's control circuitry and often cause the device to become unresponsive until reset. SEUs and SEFIs are currently the most commonly observed SEP on SRAM-based FPGAs.

4.1 Single-Event Latch-ups (SELs)

Single-event latchup is a primary concern for many spacecraft designers. SEL is a radiation-induced form of latchup that occurs in CMOS devices due to naturally occurring parasitic transistors that occur between the wells and the substrate. Devices that latchup are generally not used in space because high-current events on devices of any kind can cause latent damage [4]. For this reason, Altera's product lines have been avoided, since they latchup when exposed to heavy ions [7]. Xilinx products do not latchup in either protons or heavy ions. As shown in Table 2.1, the Virtex family devices are latchup immune to at least 90 $\frac{MeV-cm^2}{mg}$.

4.2 Single-Event Transients (SETs)

Single-event transients in CMOS occur when an ionized particle strikes a reversed biased junction in a combinational logic element. For example in invertors, an SET would deposit enough charge in the reverse-biased or "off" transistor to turn it on, which leads to a temporary voltage change on the gate's output that can propagate through the surrounding logic. The most serious consequences will occur if the transient affects critical signals, such as a clock or asynchronous reset, which could cause a number of flip-flops to load incorrect data. On the other hand, if the transient propagates to the data input of a flip-flop during the setup time (the *window of vulnerability*), the transient value may be latched. This phenomenon is called a *latched SET* and is indistinguishable from an SEU in the user flip-flops [8]. SETs have become more common in radiation-hardened application-specific integrated circuit (ASIC) and FPGA technologies with high speed circuitry, where the pulse width of the transient approaches the window of vulnerability width. SETs are difficult to detect and are usually only observable in circuits by monitoring the increase in the number of faults due to increases in clock frequency. Due to the preponderance of SEUs, observing latched SETs has been difficult in SRAM-based FPGAs[9].

It is important to note that the logic design style, storage element behavior, and system timing requirements greatly impact the probability that an SET will cause a fault. Furthermore, SETs have not yet been identified as a significant problem for user logic within FPGAs. This may be, in part, due to the significant amount of capacitance inherent to the programmable routing network of FPGAs. In the future, Xilinx will be providing a more radiation-hardened solution with the SEU-Immune Reconfigurable FPGA (SIRF) device. While these devices will not be completely SEU immune, their level of SEU immunity should be enough to make SETs more observable. Until these devices are released, though, we will not know for certain what their SET response will be.

4.3 Single-Event Upsets and Single-Event Functional Interrupts

The primary single-event phenomenon for SRAM-based devices is the SEU. An SEU, or *upset*, can be viewed as an SET that directly affects the internal circuitry of a memory cell, causing the "instant" latching of an incorrect state. For instance, an SET of sufficient length and strength that affects one of the two cross-coupled gates (e.g., inverters, NANDs, etc.) in an SRAM bit can cause the second gate to change and then hold the incorrect memory state, resulting in an invertion of the stored memory bit. Upsets are often referred to as "soft errors" since it does not result in a permanent, destructive change—the memory bit can be corrected through a subsequent write.

As stated previously, a single bit flip can cause the user circuit to output incorrect data. The ability of an SEU to cause a noticeable output error is dependent on the circuit. If the SEU does not affect a bit relevant to the user design, it will probably not affect the circuit's output. Likewise, logical error masking can prevent an SEU from causing an output error for some input values. For example, if a four-input LUT that implements an AND function has an error in its one high value, then most of the input combinations will still be error-free. Furthermore, if the input combination that causes the error to manifest in the four-input LUT rarely occurs within the input data set, the error will be even less likely to manifest as an output error.

Besides the state that is directly a part of a user's design, some amount of state exists in the FPGA's internal support logic. A bit flip in these resources can affect the functionality of a design, resulting in an event called a single-event functional interrupt (SEFI). Resetting the state of support logic may require a simple write to a global FPGA support register or may require more drastic action—such as a complete reprogramming of the FPGA—for cases where the state is not directly visible or correctable by the user design or system.

For space-based applications, devices are tested for their SEU response to heavy ions and protons. This response is called the *bit cross-section*, which is a measure of the sensitive area per bit that can cause an SEU if struck by a ionizing particle of a particular energy or LET. Bit cross-section has the units of cm^2/bit. SEFIs have a similar cross-section, but are usually reported in terms of $cm^2/device$ since the area contributing to these effects is relatively small. SEU and SEFI cross-sections have 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. The saturation cross-section indicates the maximum areal sensitivity to the radiation source. With modern technology, many devices have an onset threshold for SEUs below 1 MeV for protons[29] and below 1 MeV-cm^2/mg for heavy ions. The SEU saturation cross-section often does not saturate in modern devices due to the presence of multiple-bit upsets (MBUs) [35].

SEU and SEFI cross-sections are found experimentally using radiation accelerators. Table 4.1 has a list of SEU bit cross-sections and SEFI device cross-sections for 63.3 or 65 MeV protons and Figure 4.1 shows the SEU bit cross-sections for heavy ions for Virtex family devices. From this graph, one might note that the newer parts do not necessarily have a lower bit cross-section. While shrinking feature size often causes a decrease in bit cross-section as the "targets" are smaller, the amount of charge necessary to upset a bit also decreases. Clearly, some balance between the size of the transistors and the lower critical charge happens and some times smaller feature sizes do not translate to smaller bit cross-sections. Note that the SEFI device cross-sections from Table 4.1 appear to be on the same scale as the SEU bit cross-sections, but, in reality, when the SEU cross-sections are scaled to device cross-sections (i.e., multiplied by the number of memory bits in the device) the SEU cross-sections are several orders of magnitude larger than the SEFI device cross-sections. These cross-section values are used with orbit prediction tools, such as CREME96 [1], to determine the expected on-orbit error rate for a given device.

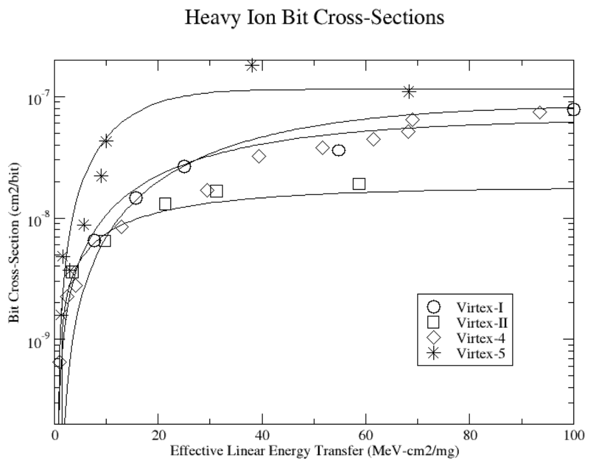


Figure 4.1: Heavy Ion Bit Cross Sections for Virtex Family Devices [37].

Device	Energy (MeV)	σ_{bit} (cm^2/bit)	σ_{SEFI} (cm^2/device)
XCV1000	63.3	$1.32x10^{-14}$	$\approx 7.1x10^{-13}$ (config SEFI)
XC2V1000	63.3	$2.10x10^{-14}$	$9.46x10^{-13}$
XC4VLX25	63.3	$1.08x10^{-14}$	$6.43x10^{-12}$
XC5VLX50	65.0	$7.56x10^{-14}$	Unknown
XC5VLX50	200.0	$1.07x10^{-13}$	Unknown

Table 4.1: Bit Cross-Section for SEUs and Device Saturation Cross-section for SEFIs for Protons for Several Xilinx FPGAs [35, 3]

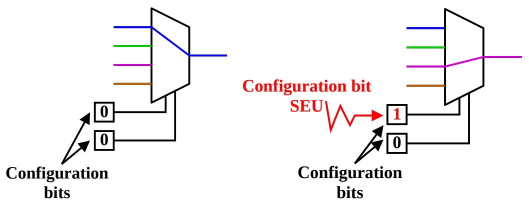
4.3.1 Failure modes from SEUs and SEFIs

FPGAs have many SEU-induced failure modes that conventional ASICs circuits do not have. For example, by changing one configuration bit, a LUT resource may no longer operate as a simple LUT, a wire might not connect the same two endpoints, or an input may suddenly be sourced from elsewhere. There are many possible ways to categorize the failure modes. One could do a broad classification of routing errors, logic errors, user memory errors, and architectural errors. For this paper, eleven specific failure modes are discussed: mux select, PIP short, PIP open, buffer off, buffer on, LUT value change, control bit change, user flip-flop, BlockRAM, half-latches, and logical constant network. Under this classification, SEFIs are classified as architectural errors and a number of SEFI modes are discussed in that section.

Routing Errors

In Virtex family FPGAs, the routing network largely consists of multiplexers (muxes), programmable interconnect points (PIPs), and buffers. Generally, the input signals to user logic and memory resources are selected using muxes. Likewise, their outputs are selectively connected to the larger routing network through muxes. In the older devices the programmable interconnect relied on PIPs to connect many routing resources, whereas the newer devices only use muxes. Finally, buffers are used throughout these devices in cases where a wire has one or only a couple of drivers. The rest of this section will be devoted to the failure modes of these three types of resources.

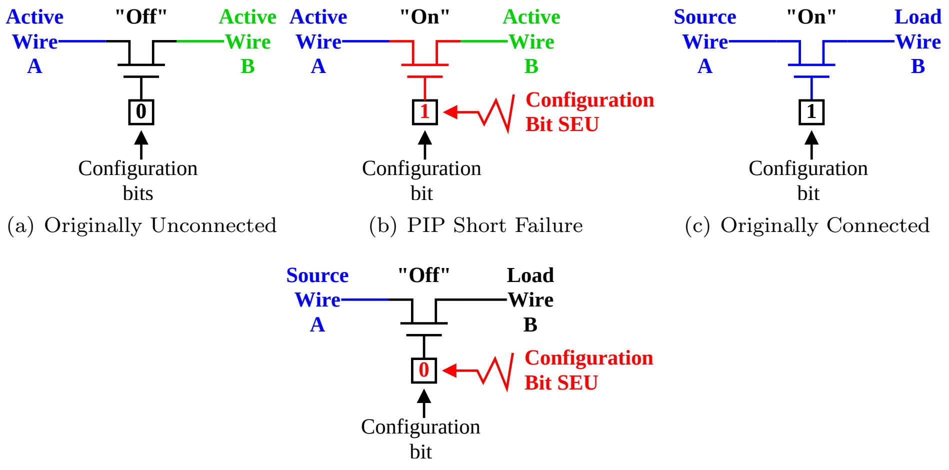
Multiplexers Multiplexers used for routing are very sensitive to SEUs because any change in their select lines will cause a different routing configuration. An example of a mux select failure is shown in Figure 4.2.



(a) Original (b) After Upset

Figure 4.2: Mux Select Failure Example

Programmable Interconnect Points In older Virtex technology the routing network used PIPs. A PIP is a pass transistor between two wires that can either be on or off. Thus, the wires are either connected or not connected. PIPs can cause a few different kinds of SEU-induced failures. The first, shown in Figure 4.3(b), is called a PIP short failure, where two wires with different functions in the design are shorted together. A PIP short can produce contention, causing output errors and increased power consumption. The second type of PIP failure is shown in Figure 4.3(d) and is called a PIP open. This occurs when a PIP, normally turned on in the design, effectively breaks a wire into two pieces. A PIP open causes an interruption in the flow of information from one part of the design to another.

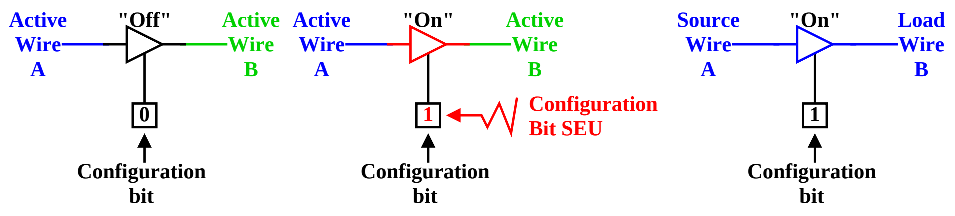


(a) Originally Unconnected (b) PIP Short Failure (c) Originally Connected

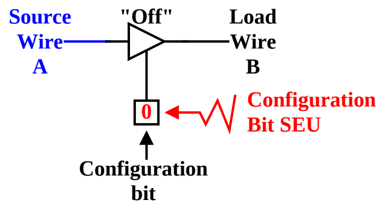
(d) PIP Open Failure

Figure 4.3: PIP Failure Mode Examples

Buffers The final component of the Virtex routing network considered here is the buffer. A buffer in this context is a driver which can either be turned on or off. As shown in Figure 4.4, the buffer has two failure modes associated with it and they are very similar to the PIP failures. The main difference here is that instead of a pass transistor, the failure is being caused by an active driver and, therefore, is unidirectional, in a sense. With a PIP failure, it is quite possible that errors can be caused on both sides of the PIP, but with a buffer failure only the output is affected. Buffers appear on the outputs of some multiplexers, on bi-directional wires (one buffer per direction), or when the number of drivers of a piece of wire is very limited, hence, not requiring a multiplexor.



(a) Originally Unconnected (b) Buffer Short Failure (c) Original Connected

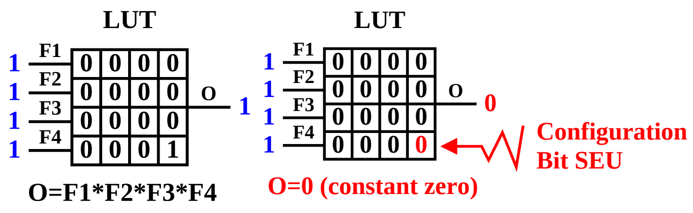


(d) Buffer Open Failure

Figure 4.4: Buffer Failure Mode Examples

Logic Errors

There are two types of logic errors: LUT value changes and control bit changes. The Virtex family of FPGAs use lookup tables to generate most logic functions, so a change in the values stored in a LUT would impact the logic function implemented therein. This failure mode could cause constant or intermittent output errors depending on the inputs to the circuit and which part of the logic function is impacted. An example is shown in Figure 4.5. Here the LUT implements a 4-input AND function. If the one bit that defines the "true" condition is upset, the result is a constant-zero function. For most inputs, the output of the function would still be correct; however, one case would cause problems.



(a) Original 4-input AND Function (b) Upset LUT Function (Constant "0")

Figure 4.5: LUT Upset Example

In contrast to LUT value changes, control bit changes generally cause errors for all, or almost all, possible circuit inputs. The CLBs and IOBs use quite a few control bits to determine miscellaneous functionality. Figure 4.6 shows a partial schematic of a Virtex-I slice. Bits V, E, F, and G are called programmable inversion bits. An upset to one of these will cause the value carried on that particular wire to be inverted, likely resulting in a circuit error when the value is used. The T bits, on the other hand, determine whether the LUT in this slice performs as a LUT, a 16x1 dual-ported RAM, half of a 32x1 RAM, or as a programmable shift register. If a LUT suddenly turns into a shift register, output errors will likely result. Other control bits determine such things as the electrical standard used in off-chip I/O and whether a storage element is a flip-flop or a latch.

User Memory Errors

As with other forms of memory on SRAM FPGAs, user memory is also susceptible to SEUs. These user memory resources include BlockRAM (BRAM), LUT-based RAM, user flip-flops in the CLBs, and I/O block

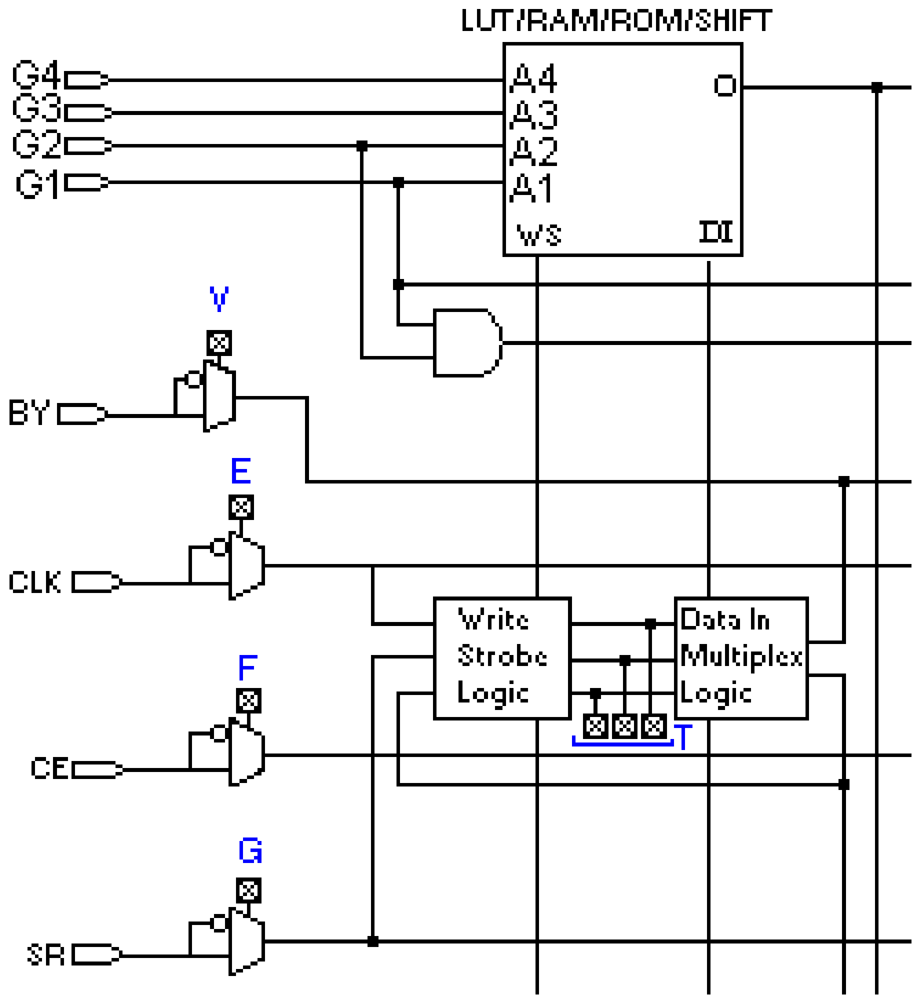


Figure 4.6: Control Bit Examples[2]

flip-flops (IOB-FF). Upsets in these resources can introduce bad data into the operation of a user circuit. In general, upsets in these resources are not easily detectable or correctable through examining an FPGA programming data since it requires some way of predicting what the proper values are. The exception is, of course, when the memory is used effectively as a read-only memory (ROM).

Logical Constants

The generation of logical constants is necessary in Xilinx devices to provide constants for the user designs and tie off unused inputs. Some of these logical constants are an artifact of mapping an HDL circuit description onto the FPGA architecture and compensating for differences in bit widths between the expressed function and the resource used to implement the function. In Xilinx architectures, access to ground and V_{cc} signals have not generally been provided. As a result, logical constants are often generated locally using "soft" user logic. In Xilinx FPGAs, there are two common methods for generating logical constants. Unfortunately both methods are affected by SEUs. Each method is discussed below.

Half-Latch Errors Xilinx Virtex and later FPGAs can use a little known architectural feature called the "half-latch"[18] (see Figure 4.7) to generate constant zero and one logic values used internally by FPGA designs. Half-latches are weak keeper circuits that are widely available in Xilinx FPGA architectures and can be used to drive inputs to I/O, logic, RAM, clocking, and other resources to a constant value when the inputs are not actively driven. By using half-latches to create constant values, more expensive logic resources, such as lookup tables (LUTs), are not needed. Unfortunately, half-latches are susceptible to SEUs (see Figure 4.8). Even worse, half-latches are not directly initialized or controlled with programming data, which makes them hard to observe and modify. Furthermore, in existing architectures, only full reconfiguration of an FPGA re-initializes half-latches while processes that use partial, on-line reconfiguration for repair—such as scrubbing—are unable to reset them.

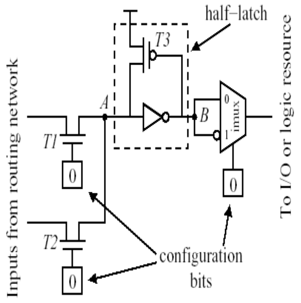
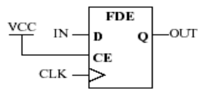
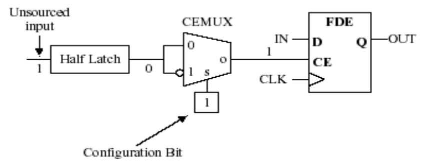


Figure 4.7: Xilinx half-latch [18].

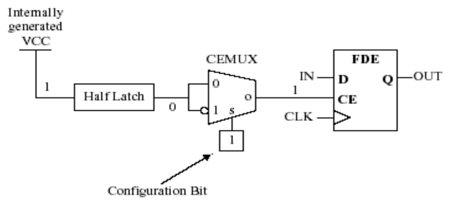
"Global Logic" Network Errors Another commonly used approach to providing logical constants to a user design is generate the constants directly using LUTs and route the constants around the FPGA, as needed. As in the case of half-latches, these methods of producing logical constants are inferred automatically through the FPGA design flow. With the last several generations of FPGA implementation tools (from at least Xilinx's ISE 3.3i on), these automatically inferred logic constant networks that generate logic "0" and logic "1" signals have been called "GLOBAL_LOGIC0" and "GLOBAL_LOGIC1", respectively, in designs that have gone through technology mapping (i.e., "map" in the Xilinx tool suite). As an additional set of optimizations, the place and route tools will generally generate multiple "GLOBAL_LOGIC0" and "GLOBAL_LOGIC1" nets to reduce fanout, load balancing the fanout among the various LUTs providing the constants. As a result of these optimizations and the fact that the conventional FPGA design flow is unaware of fault-tolerant FPGA design techniques, such as TMR, redundant logic modules in a TMR-protected design could share the same logical constant network, introducing potential single points of failure into the design. Since these "global logic" networks are implemented with LUTs and conventional FPGA



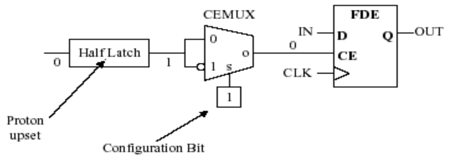
(a) Intended circuit



(b) Usual implementation with half-latch



(c) Initialization during full configuration



(d) Circuit with half-latch SEU

Figure 4.8: Results of a Xilinx half-latch SEU [18].

routing, this approach to generating logical constants is susceptible to all of the failure modes that affect those resources.

Architectural Errors

Architectural errors are those upsets that occur in the control elements of the FPGA, such as the configuration circuit, the JTAG TAP controller, and reset control. Architectural errors are not directly observable and can usually only be measured indirectly through an upset "signature". For example, an SEU in the configuration control circuit may change many of the configuration bits at once, creating a very noticeable effect. For this matter, half latches in the Virtex-I could also be classified as an architectural error. Below is a discussion of SEFIs, which are architectural in nature.

Single-Event Functional Interrupts Internal to the device there are a handful of configuration and control registers that can be affected by SEUs. SEUs in these registers can affect the ability of the device to function properly, and these SEUs are generally classified as SEFIs. Examples of SEFIs that have affected multiple Virtex family devices include JTAG TAP controller upsets, SelectMAP controller upsets, and configuration control logic upsets (Power-On-Reset or POR). The Virtex-4 is also susceptible to Frame Address Register (FAR), Global Signal, Readback and Scrub SEFIs. Below, all of these SEFIs are discussed.

A SEFI in the JTAG controller, whether it is being actively used or not, will move the device into an undesirable state. Compounding the problem, the Virtex device does not make the reset pin (TRST) available to the user, denying designers the ability to hold the JTAG controller in reset while it is deployed.

SEFIs within the programmable SelectMAP interface configuration pins will result in a malfunctioning interface. Since the interface can no longer be accessed reliably, reading and writing to the device should be suspended until fixed. The corrupted interface could return bad values on a read and corrupt the configuration on a write.

Virtex-4 designs that do not use the GLUTMASK configuration mode, which allows the use of SRL16s and other LUT-based memory with on-line reconfiguration, are susceptible to the Readback SEFI. The signature for this SEFI looks like a SelectMAP SEFI, but is really only a false-positive for the SelectMAP SEFI. Since systems using on-line reconfiguration should skip the SRL16s or LUT-based memories to avoid corruption of their values, these parts of the design will not be corrected. Designs that do not use GLUTMASK then return repeated CRC fails based on the SRL16s and the LUT-based memories not being scrubbed. Since a repeated CRC failure is the signature of a SelectMAP SEFI, a SEFI detection circuit would assume the problem is that the device cannot be accessed through the SelectMAP port and not that the on-line reconfiguration circuit is properly skipping areas that should not be reconfigured.

The configuration control state machine of most Virtex family devices is vulnerable to the power-on reset (POR) SEFI. In this case, the device behaves as if $\overline{PROGRAM}$ has been asserted — its configuration is cleared and the $DONE$ pin is driven low.

With the FAR SEFI the FAR will increment "continuously" and "uncontrollably" [3]. While the SelectMAP port will remain accessible during a FAR SEFI, FAR will not be under user control, resulting in unreliable reads and writes of the device's programming data. For many purposes, this SEFI is often included as part of the SelectMAP SEFI.

The Global Signal SEFI is an umbrella SEFI that covers functional interrupts to the the Virtex-4's global signals, such as Global Set/Reset, Global Write Enable, and Global Drive High. These signals can be observed through upsets in device-level control registers.

The final SEFI is the configuration control logic, or "Scrub", SEFI observed in the Virtex-4. In this case, the programming control logic can be upset while scrubbing, causing a corruption of the programming data being written to the FPGA. Not only does this affect the design, but it can cause a high-current draw state in the device due to the bad programming data. While this problem possibly could exist in the Virtex-I and the Virtex-II, the Scrub SEFI never presented during testing.

Chapter 5

SEU and SEFI Characterization

SEU and SEFI characterization testing measures the SEU and SEFI response of a device over a series of radiation sources. This measurement determines the size of the sensitive area, called the *cross-section*, for a given device. While the SEFI cross-section is rarely dependent on how the device is used, the SEU cross-section can be measured both in how the entire device is affected (*static cross-section*) and how a specific design is affected (*dynamic cross-section*).

5.1 Static SEU Cross-Section

The static SEU device cross-section (σ_{device}) for an SRAM FPGA is a measure of the sensitive area/volume of all nodes in a device that can cause a memory state change irrespective of whether faults manifest as observable errors. The static cross section is generally the most pessimistic estimation of a user design's cross section, because not all upsets affect logic utilized by a particular design. The static cross section can be calculated in most cases by using the following equation:

$$\sigma_{device} = \frac{\#ofupsets}{fluence \times cos(\theta)} \tag{5.1}$$

where θ is the angle of incidence of the beam to the part. To provide a method of comparing devices, most people report cross-section values on a per-bit basis using this equation:

$$\sigma_{bit} = \frac{\sigma_{device}}{\#ofbits} \tag{5.2}$$

Comparing devices using the σ_{device} and σ_{bit} is not straight forward, though. All of the different memory cells in a device — look up tables, user flip-flops, BRAM – have different cross-sections. In the newer devices, the percentage of each of these memory types will sway the bit cross-section toward one memory type or another. Figure 5.1 again shows the heavy ion bit cross-sections for the Virtex family devices.

Figure 5.2 displays the various categories of SEUs that contribute to the Virtex-I FPGA's cross-section. SEUs occurring in configuration memory contribute the most to the Virtex-I cross section. Indeed configuration memory is by far one of the largest structures in the Virtex-I of devices. In the Virtex XCV1000 there are 5.8×10^6 configuration latches which dwarfs the 2.4×10^3 user flip-flops present in the device. BRAM has become an increasingly larger percentage of Xilinx FPGAs over the years, which has lead to an accompanying increase of upsets in the BRAM region. For the Virtex-4 and the Virtex-5 FPGA families, devices comes in LX, SX, and FX versions. The SX versions of the Virtex-4 and Virtex-5 devices have a higher BRAM-to-logic ratio than the FX and LX versions of these devices. Since the BRAM is a more traditional memory structure, the cross-section for these devices can be greatly influenced by the BRAM cross-section.

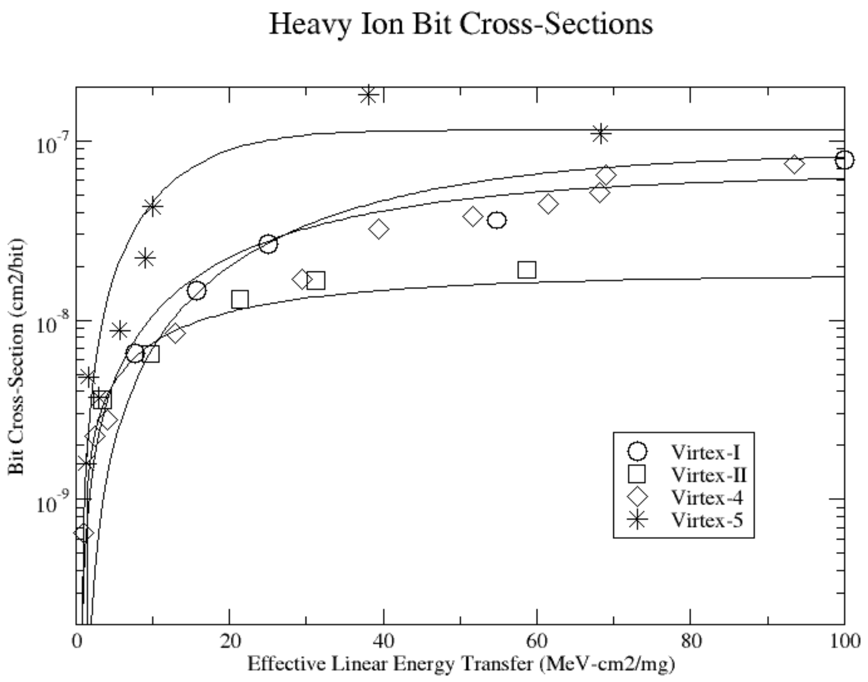


Figure 5.1: Heavy Ion Cross-Section for Virtex Family Devices[5]

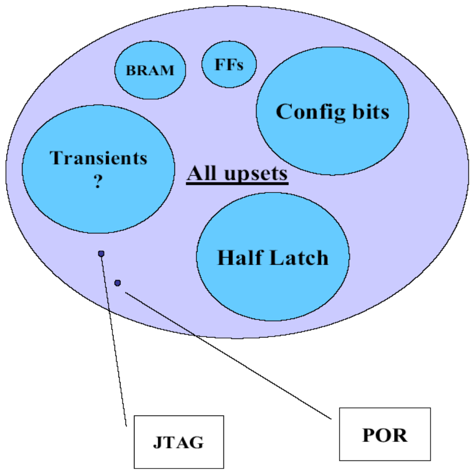


Figure 5.2: Categories of SEUs in the Virtex device cross section [5]

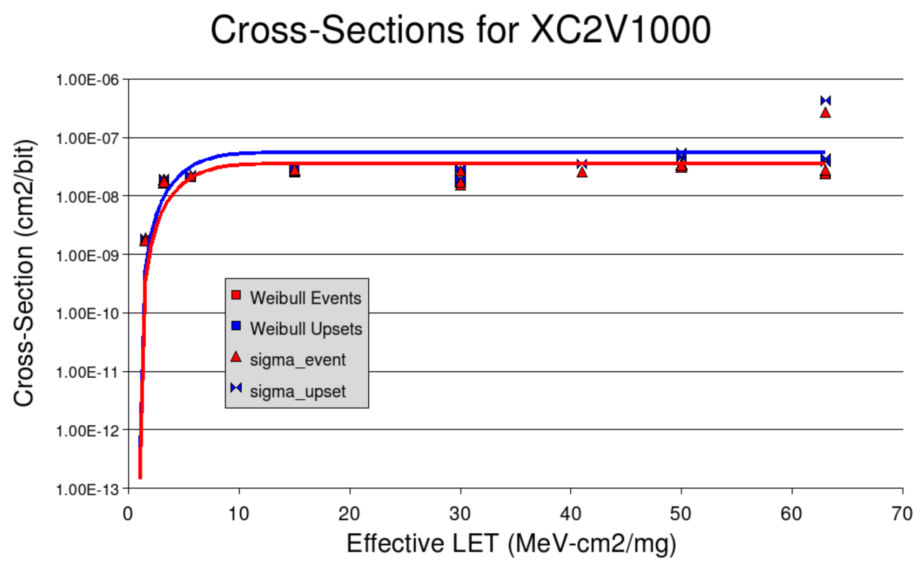


Figure 5.3: Heavy Ion Upset and Event Cross Sections for the XC2V1000 device [34].

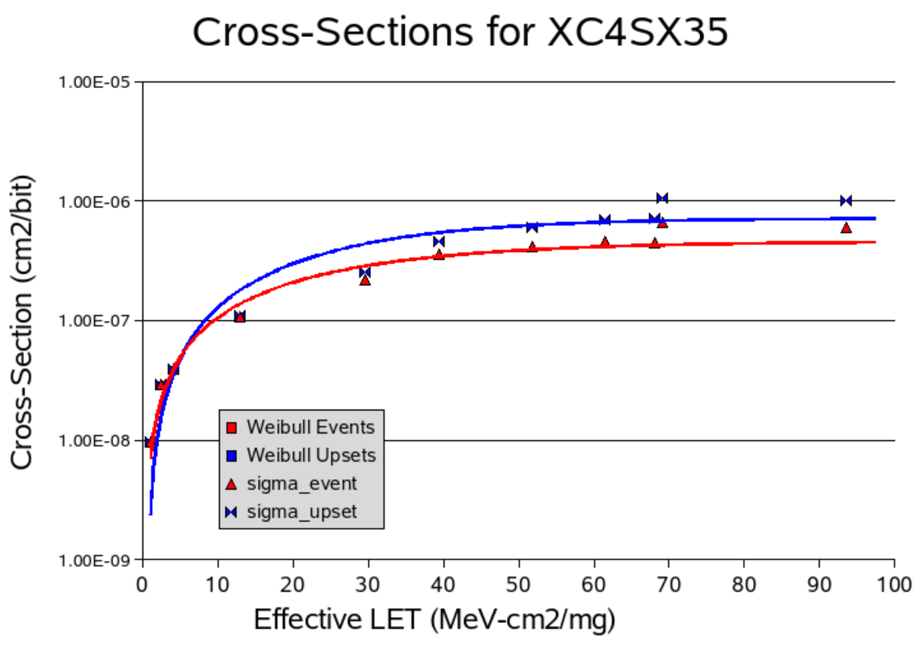


Figure 5.4: Heavy Ion Upset and Event Cross Sections for the XC4XSX35 device.

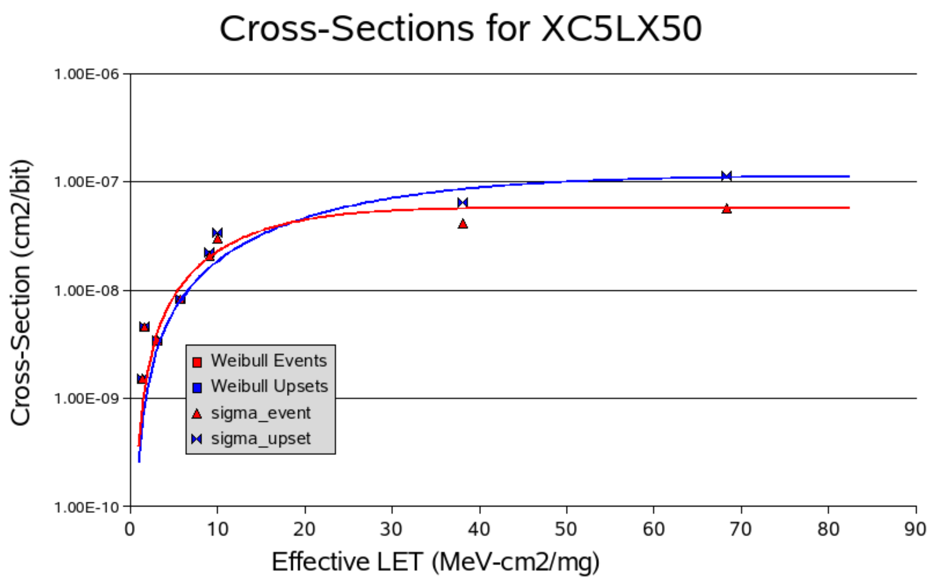


Figure 5.5: Heavy Ion Upset and Event Cross Sections for the XC5VLX50 device.

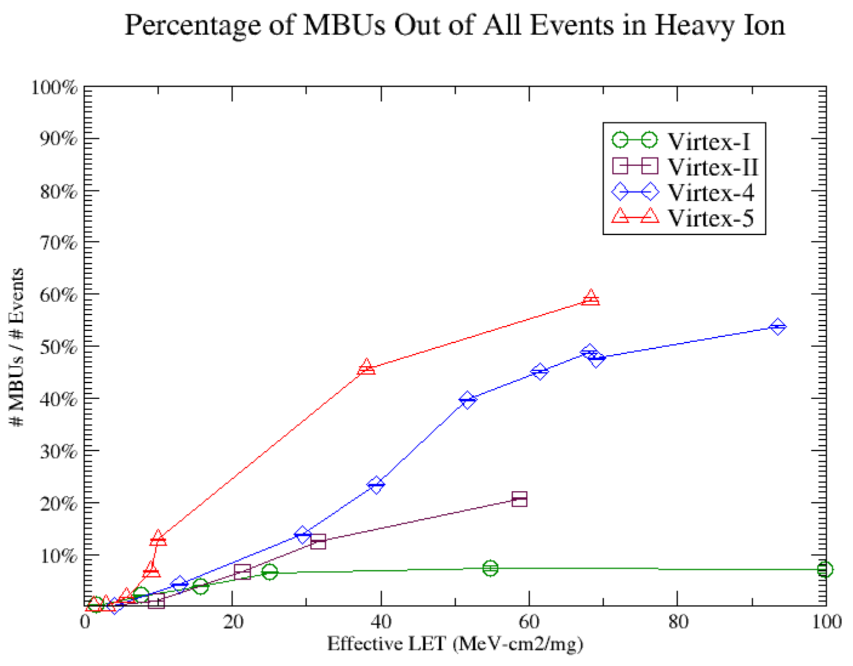


Figure 5.6: Percentage of MBUs in Heavy Ion for Virtex Family Devices [37].

5.1.1 Single-Bit Upset/Multi-Bit Upset Cross Sections

Recent research performed by LANL has shown that the static cross section of a device can be further categorized into single-bit upset (SBU) and multiple-bit upset (MBU) cross sections. Previously, it had been noted that the cross-section was not saturating in what is traditionally called the saturation region, as shown in Figure 5.3. The rationale behind the saturation region is that all of the radiation strikes in the active volumes of the device should cause upsets, so the cross-section should reach an upper limit. The experimental evidence for FPGAs shows that the cross-section of more recent FPGAs do not reach an upper limit. The hypothesis was that MBUs were happening at the higher LETs causing more than one upset to occur with each radiation strike. When upsets were counted as *events*, where each MBU counted as one event instead of multiple upsets, the thought was that the event cross-section would saturate even at high LETs. When the data was reanalyzed MBUs were found at the higher LETs. Figure 5.3 shows a comparison of the event and upset cross-sections for a Virtex-II device. This figure indicates that the event cross-section saturates as expected above an LET of 15 MeV-cm^2/mg. For heavy ions, this trend found in the Virtex-II continued and worsened in the newer devices, as shown in Figures 5.4 and 5.5. In the Virtex-5, MBUs have become the dominant type of SEU at higher LETs and when ions strike the silicon at angles other than normal incidence with the chip's surface[36]. Furthermore, a significant percentage of proton-induced SEUs are now MBUs in the Virtex-5, which could cause problems in LEO.

For the Virtex-I, relatively few MBUs occur, resulting in very little difference between the event and upset cross-sections [34]. For the newer devices, the difference between these two cross-sections widens. As shown in Table 5.1, the newer Virtex device families are increasingly more sensitive to MBUs in proton radiation. Table 5.2 provides a list of measured SBU, MBU, and SEFI cross-sections for various Virtex families. Based on observations of normal incidence data sets, 1–6% of all 65 MeV proton-induced events are MBUs and as much as 59% of all heavy-ion-induced events are MBUs (see Figure 5.6) in newer Virtex technologies [34]. For proton-induced MBUs, more energetic protons are more likely to cause MBUs. For data sets taken at angles other than normal incidence, these percentages can be quite a bit larger. To account for MBUs when calculating the static cross section for Virtex-II and later devices, we suggest using the following equation:

$$\sigma_{device} = \sigma_{1-bit} + \sigma_{2-bit} + \sigma_{3-bit} + ... \tag{5.3}$$

$$= \frac{events_{1-bit}}{fluence \times cos(\theta)} + \frac{events_{2-bit}}{fluence \times cos(\theta)} + ... \tag{5.4}$$

Device	Energy (MeV)	Total Events	1-Bit Events	2-Bit Events	3-Bit Events	4-Bit Events
XCV1000	63.3	241,166	241,070 (99.96%)	96 (0.04%)	0 (0%)	0 (0%)
XC2V250	63.3	337,814	333,639 (98.76%)	4,129 (1.22%)	44 (0.01%)	2 (0.0006%)
XC2V1000	63.3	204,009	199,641 (97.86%)	2,164 (1.06%)	12 (0.006%)	1 (0.0005%)
XC4VLX25	63.3	152,577	147,902 (96.44%)	4,567 (2.99%)	78 (0.051%)	8 (0.0052%)
XC5VLX50	65.0	2,963	2,792 (94.23%)	161 (5.43%)	9 (0.30%)	1 (0.03%)
XC5VLX50	200.0	35,281	31,741 (89.86%)	3,105 (8.79%)	325 (0.92%)	110 (0.43%)

Table 5.1: Frequency of Upset Events and Percent of Total Events Induced by Proton Radiation for Five Xilinx FPGAs [34, 36]

Device	Energy (MeV)	σ_{1-Bit} (cm^2/device)	σ_{MBU} (cm^2/device)	σ_{SEFI} (cm^2/device)
XCV1000	63.3	$1.06x10^{-7}$	$4.67x10^{-11}$	$\approx 7.1x10^{-13}$ (config SEFI)
XC2V250	63.3	$3.52x10^{-8}$	$3.84x10^{-10}$	$9.46x10^{-13}$
XC2V1000	63.3	$7.68x10^{-8}$	$8.37x10^{-10}$	$9.46x10^{-13}$
XC4VLX25	63.3	$7.90x10^{-8}$	$2.50x10^{-9}$	Unknown
XC5VLX50	65.0	$8.20x10^{-7}$	$5.02x10^{-8}$	Unknown
XC5VLX50	200.0	$1.05x10^{-6}$	$1.18x10^{-7}$	Unknown

Table 5.2: Comparison of the SEU, MBU, and SEFI Device Saturation Cross-sections for Protons for Five Xilinx FPGAs [34, 36]

5.2 Dynamic SEU Cross-Section

Often for FPGAs the more important measurement is how a particular user design is affected by radiation-induced upsets. This measurement is called the *dynamic cross-section.* Unlike the static cross-section where all upsets on the device are included in the cross-section, for the dynamic cross-section, only upsets that cause a noticeable output error are included in the cross-section. Therefore, the dynamic cross-section has a design dependency and is not transferable from one design to the next. Because there are a number of ways to cause output errors, the dynamic cross-section is combination of the SEU cross-section, the SET cross-section and the SEFI cross-section. Since there is no noticeable SET cross-section for Xilinx FPGAs [13] and the SEFI cross-section is quite low, the dynamic cross-section is dominated by the SEU cross-section. Therefore, at worst case, the dynamic cross-section could approach the static cross-section. At best case, though, the dynamic cross-section could be as small as the SEFI cross-section.

Based on experience we have found that dynamic cross-section may be only 1–20% of the static cross-section (see Figure 5.7) for designs without fault tolerance. The particular design plays an important role in the dynamic cross-section. First, some designs are more sensitive to errors than other designs. Errors in the LUTs are strongly influenced by input combinations. With only a single bit error in one LUT, the LUT will perform error-free for 15 out of 16 input combinations or nearly 95% of all use cases. If the one input combination that causes the error rarely occurs in the data set, the error state will rarely manifest in the circuit. Furthermore, the LUT's output could be logically masked by the rest of the circuit. A second factor in design dependency is whether the upset occurs in an area of the device that is being used. Given the multitude of mutually exclusive point-to-point routing options, much of the routing remains unused in a design. As routing accounts for approximately 80% of the programming data, much of the programming data is not relevant to a particular design. Therefore, assuming upsets are uniform across the device, most upsets will occur in the unused portion of the device. The final design-dependency is whether mitigation has applied to the design. A fully triplicated design is less likely to manifest output errors than an unmitigated design. Ideally, the dynamic cross-section of a mitigated design should be equal to the SEFI cross-section since there would be no other contributions to the cross-section.

While accelerator testing is the "golden standard" for determining the dynamic cross-section, fault injection techniques based on partial on-line reconfiguration can be used to emulate SEUs before accelerator testing. Since radiation effects are Poisson random processes, accelerators cannot easily or cheaply provide the uniform coverage of fault injection. Furthermore, since many accelerators cannot be tuned to provide only one upset at a time, determining whether faults are caused by a specific upset or multiple independent upsets is very difficult. In contrast, fault injection can be very useful because it allows for precise, targeted, uniform, and inexpensive experiments to characterize the dynamic cross-section. The speed and precision of fault injection have helped uncover effects that were either too hard to distinguish or too small to be found in limited accelerator tests [17]. While fault injection cannot nor should not replace accelerator testing, it can move accelerator testing into more of a verification role than a discovery role.

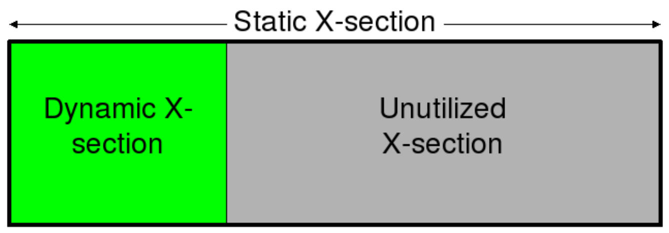


Figure 5.7: The Dynamic Cross Section is roughly 1 – 20% of the static cross section [31].

5.2.1 Persistent/Non-Persistent Cross-Sections

The dynamic cross-section of a device can be further categorized into terms of persistent and non-persistent components [32]. Circuits are often a combination of feed-forward and feedback logic. An SEU in unmitigated feed-forward or feedback logic could cause a stream of bad output data, but there can be a significant difference in how long the error state remains between these two circuit topologies. Figure 5.9 illustrates this difference, plotting the error magnitude for two circuits affected by SEUs versus time. The plot on the left is for feed-forward logic, such as shown in Figure 5.10. Once the SEU is removed the bad data will naturally flow out of the system to be replaced with good data, resulting in a *non-persistent* erro. The plot on the right represents the error signature for a circuit with feedback, such as that in Figure 5.11. Bad data continues to circulate in the system even after the SEU is removed. This case is called a *persistent* error and the bad data may persist until the upset is removed and the device is reset. The root problem with the feedback circuit is that the bad data cannot repair itself from just removing the upset. If the feedback loop is triplicated and the feedback is cut with triplicated voters, not only will the bad data be masked in the system, but the two working modules will be able to repair the bad data in the third module after the upset is removed. Accordingly, designs that are dependent on feedback structures, such as counter arrays, tend to have larger persistent cross sections than designs that do not (see Figure 5.12).

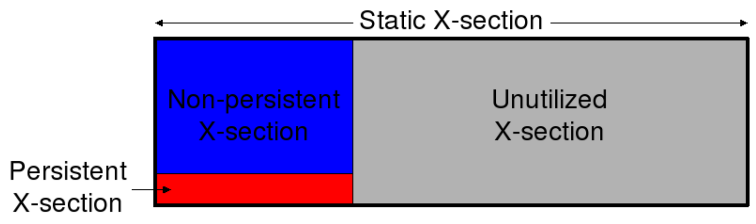


Figure 5.8: The dynamic cross section is broken down into a persistent and non-persistent cross section [31].

5.3 SEFI Cross-Section

The SEFI cross-section is the measure of device-specific sensitive areas that have unusually far reaching consequences when an SEU effects them. As mentioned in 4.3.1, this cross-section includes the JTAG TAP controller SEFI, the power-on reset (POR) SEFI, the SelectMAP configuration pins SEFI, and others. In the XQVR1000 the SEFI cross section in proton measures less then $1 \times 10^{-5} cm^2$ [17].

Measuring the SEFI cross-section is much more time consuming than measuring the SEU cross-section, since it is many orders of magnitude smaller. Unlike SEUs that can be found by reading back the programming data in the device and comparing the values to the uncorrupted bitstream, SEFIs are often observed indirectly. In the case of the POR SEFI, the device becomes completely unprogrammed, so SEFI testing watches for a large number of spontaneous errors. SelectMAP and JTAG SEFIs are usually observed by detecting problems with these interfaces, such as not being able to read or write properly to the device.

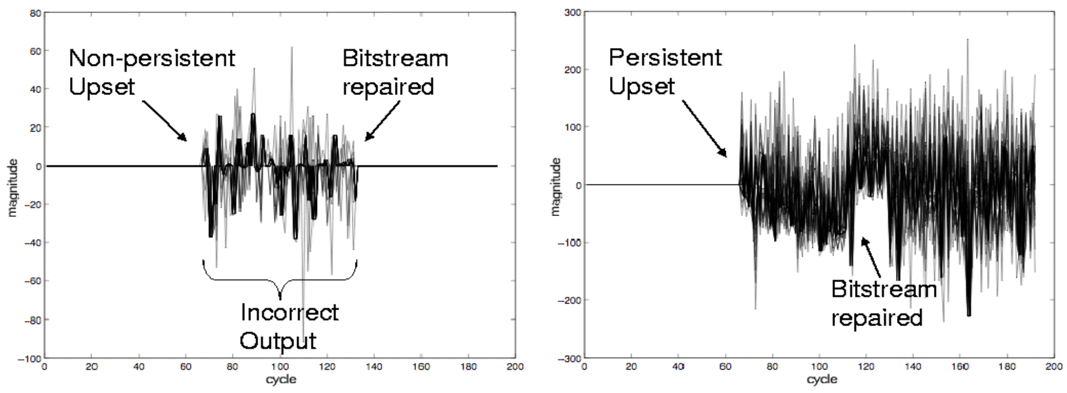


Figure 5.9: The results of a non-persistent vs. persistent upset. Delta is between the outputs of a design under test and a golden control circuit [32].

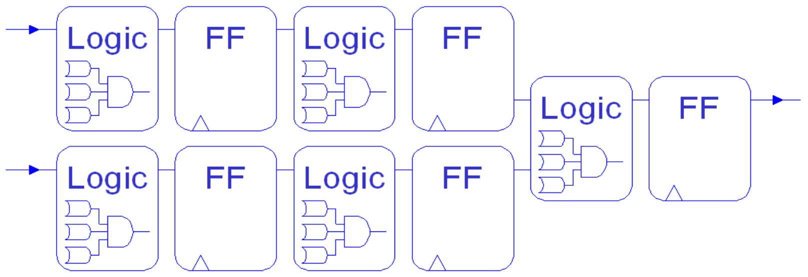


Figure 5.10: The non-persistent cross section consists of the feed-forward structures in a circuit [31].

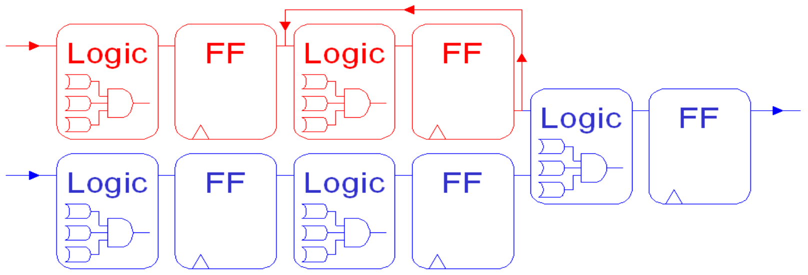


Figure 5.11: The persistent cross section consists of feed-back structures in a circuit [31].

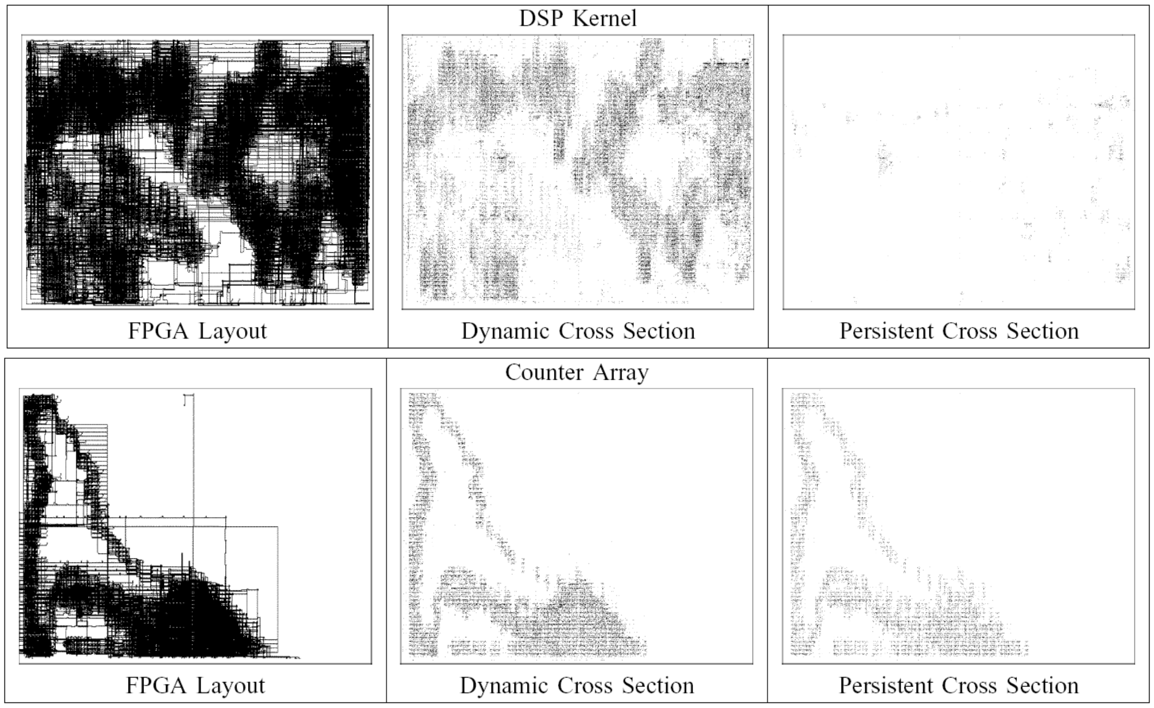


Figure 5.12: Layout of a DSP kernel (low feedback) vs. a counter array (high feedback). Notice that the persistent cross section is larger in the counter array even though it has a smaller dynamic cross section [32].

Chapter 6

On-Orbit SEU Rate Estimation

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 [27] [28] [33] . If a designer needs more accurate results there are a few tools and models that the CREME96 website recommends at https://creme96.nrl.navy.mil/cm/DoseSoftware.htm.

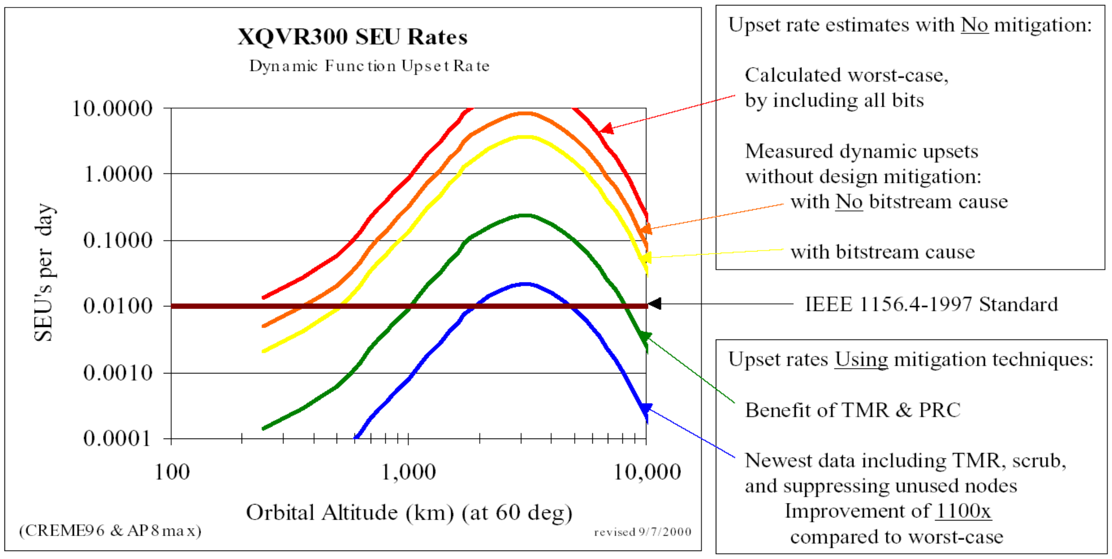


Figure 6.1: CREME96 SEU calculations for a design on the XQVR300 platform using various forms of mitigation [14].

A typical requirement for orbital upset analysis is the heavy ion cross-section vs. LET curve for the device in question (see chapter 5 for more information on device characterization). This curve is usually generated by gathering SEU data over a broad range of LETs (usually 100 to 1000 bit upsets for statistical significance) and fitting them with the Weibull formula [14]:

$$F(L) = \sigma_{sat}(1 - e^{-[(L-L_0)/W]^s})$$

where:

$F(L)$ = SEU cross-section in $\mu m^2/bit$;

σ_{sat} = saturation or sometime called the limiting or plateau cross-section = $8\mu m^2$ for *Virtex*;

L = effective LET in $MeV \cdot cm^2/mg$;

L_0 = upset threshold LET = $1.2\, MeV \cdot cm^2/mg$ for *Virtex*;

W = width parameter = 30 for *Virtex*;

s = a dimensionless exponent = 2 for *Virtex*;

An example of a heavy-ion SEU cross-section for the Virtex XQVR300 using the Weibull formula for fitting is given in Figure 6.2. Typically software used for the calculation of on-orbit SEU rates will provide a way for the device's cross-section curve to be entered using Weibull formula parameters or by inputting a table of cross-section values by LET. An example of the parameters used for the Virtex(-I) FPGA under the CREME96 tool suite are provided in Appendix B.

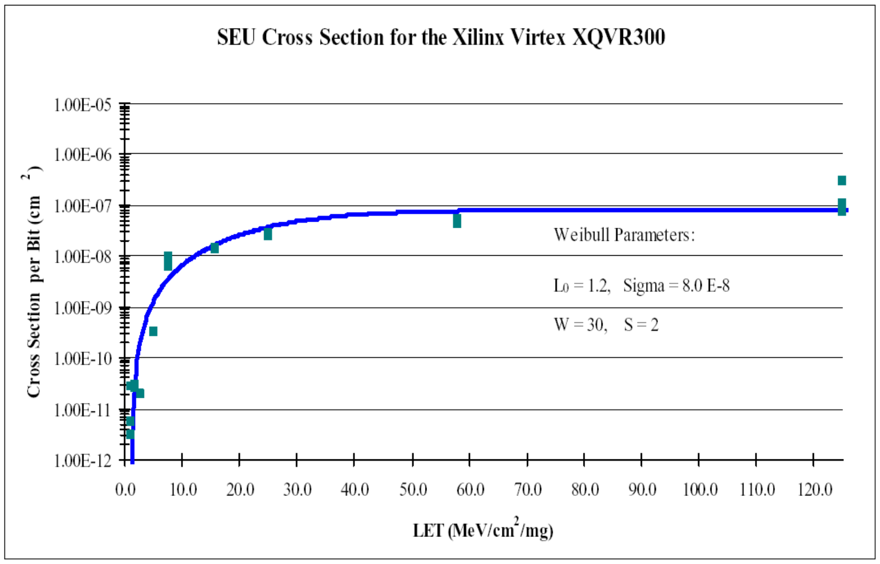


Figure 6.2: Cross Section for the XQVR300 using the Weibull formula for curve fitting [14].

Chapter 7

SEU Detection, Mitigation, and Repair

For most orbits SRAM FPGA devices will experience SEUs, so knowing how to detect, mitigate, and repair SEUs is necessary for identifying faults as well as for achieving fault-tolerant operation. This chapter will provide an overview of several possible approaches for detecting, mitigating, and repairing SEUs for SRAM FPGA circuitry. Due to the differences between SRAM FPGAs and other devices used for space, a number of techniques have been developed to account for the special features and issues of SRAM FPGAs to ensure the integrity of user design operation. Rather than reproducing the vast quantity of information on these topics, we provide overviews of the most common or important approaches in SEU detection, mitigation, and repair and then refer the reader to the key publications on the topics for further study.

7.1 Detection

There are various methods of SEU detection, each suited to different applications and design needs. We will briefly discuss several methods, including readback and comparison, concurrent error detection, the "golden output" method, CRC frame checks, device-supported detection, and SEFI detection and recovery.

7.1.1 Readback and Comparison

The standard method of SEU detection involves verifying programming data by using reading out the FPGAs' programming data (a process referred to as "readback") and performing a bit-by-bit comparison with a *golden* (non-irradiated) version of the programming data (or bitstream). The downside to this method is that it requires three times the amount of memory (the readback stream, the mask stream, and the configuration stream) than normal configuration and readback operations require [44]. The Xilinx application note XAPP138 goes into great detail in describing this process for further reference.

One drawback to this approach is that it can lead to programming interface contention when used incorrectly with distributed (LUT) RAM or Block SelectRAM resources. The device programming interface uses the same logic to read and write these RAMs that user designs use, so reading and writing these memories through a device programming interface such as the JTAG or SelectMAP ports can be problematic when the user design tries to access the memories simultaneously. See subsection C.2.2 for more details on how to deal with this issue with user designs.

GLUTMASK

While the advice of subsection C.2.2 is relevant to all Xilinx Virtex family FPGAs, a new feature has been added as of Virtex-4 that reduces this contention issue. The Virtex-4 has GLUTMASK configuration mode that allows for the use of on-line configuration and readback of programming data without disturbing what is stored in LUT RAMs. This feature does not address programming contention issues for BRAM. By setting the GLUTMASK bit appropriately in the devices configuration control registers (contact your Xilinx field-applications engineer (FAE) for the exact details), writing the device's programming data will actually skip LUT RAM resources so they are not affected when writing the programming data for surrounding resources. Likewise, when GLUTMASK mode is enabled, reading a LUT RAM will return all zeros for the value of the LUT RAM, avoiding an actual read of the RAM.

With this significant improvement in supporting on-line writing and reading of programming data, there are a few additional issues to note and emphasize. First, we strongly suggest contacting your Xilinx FAE for the exact configuration command sequences to operate in GLUTMASK mode. Second, the use of this feature in Virtex-4 results in a few configuration bits that can become difficult to scrub. There are a few work arounds for this issue being tested by Xilinx, LANL, and possibly others. This issue is an additional reason to work closely with your Xilinx FAE to understand how to use this feature. With GLUTMASK off, the advice of subsection C.2.2 is relevant for Virtex-4.

7.1.2 CRC Frame Checks

Another method of programming data verification, developed by LANL and used in CFE and other missions, precomputes and stores a CRC value for each data frame in a design's bitstream. During the readback process a new CRC value is generated and compared to the original CRC value for the data frame being checked. Since the data frame is the smallest form of granularity in any read or write process we only need to know which frame the exception has occurred in for a solution. This method conserves a great deal of memory that would otherwise be occupied with performing standard readback and comparison[44]. More detail on this method is available in Xilinx Application Notes XAPP138 and XAPP216.

7.1.3 Golden Output Method

Th "golden output" method for SEU detection involves taking a predefined set of input test vectors, running it through a design, and comparing the results against a "golden" set of output data (see Figure 7.1) that was generated by the design with no SEUs present. Using this method a system can determine whether the bitstream is still good to some degree. Furthermore, this method will work even if RAM resources are present in the configuration data and the GLUTMASK feature (7.1.1) cannot be used.

There are two down sides to this method of detection. First, testing has to occur at lulls in the system's use so that testing does not interrupt the payload's operational tasks. Systems with high temporal utilization of the hardware may not be able to use this option. Second, the designer has to devise a way to provide enough test coverage using a limited set of test vectors to credibly assure good test coverage. In a design that is extremely large and/or complex good test vector coverage may not be feasible.

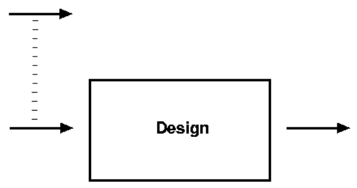


Figure 7.1: In the golden output detection method test vectors are used to test all available paths in a design.

7.1.4 Concurrent Error Detection (CED)

Another method that allows RAM resources to be present in the design while still being able to detect SEUs is the concurrent error detection method [24]. This approach uses two identical designs running in lockstep and later compares their output (see Figure 7.2) to detect a fault condition. For additional protection, a self-checking comparator can be used so that faults in the comparator are not misconstrued as other failures.

With the proper application of this method to a design, a great deal of information can be extracted from the operation of the two designs including whether or not the bitstreams and data are still intact. This information can be leveraged in several ways. One scenario is that the detection of an error can be used to request a scrub of the programming data or a reconfiguration of the FPGA and a reset of the application. Likewise, the detection of an error can be used to tag data as suspect so that potentially bad data is recognized and properly taken care of on the ground.

This ability to detect the effect of an SEU and identify when the circuit is not operating properly with a high degree of confidence comes at a cost. In particular, the the "target" area of the application has effectively doubled, so the design is approximately twice as likely to be affected by an SEU. In some cases, the extra vulnerability may be worth the ability to detect circuit failures.

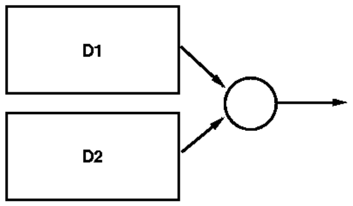


Figure 7.2: Concurrent error detection employs the use of duplicates of a design (D1 & D2) which are run in lock step. A comparator device is used to analyze their output and locate SEUs.

7.1.5 Internal Device Support for SEU Detection

As even terrestrial applications of SRAM FPGAs begin to show some susceptibility to soft errors in the programming data, Xilinx has started to include features in the FPGAs themselves that supports the automated detection of SEUs. We will briefly mention a few of these features.

For Virtex-4, Xilinx has started to provide single-error detect, double error correct Hamming codes (often referred to as ECC) for configuration frames. This is a part of the encoding of each configuration frame by default. A special resource called the FRAME_ECC_VIRTEX4 provides the syndromes for individual frames when performing readback. A designer could use this approach for quick error detection and error correction with few resources when single errors occur since it does not require anything but a single readback configuration frame and the syndrome, both provided by the FPGA. For double-bit errors and larger errors, additional data would be needed for repairs. As downsides, ECC is not perfect for multi-bit error detection above two errors and the FRAME_ECC_VIRTEX4 resource is internal, requiring it to be specifically connected to the FPGA's pins so it can be read by external circuitry, making the reading of the syndromes somewhat prone to SEUs as well—some SEU mitigation, of course, is possible. We should be clear, though. This hardware does not automatically repair or detect errors but provides some coding and decoding aid for SEU detection for the programming bitstream. See Xilinx's UG071 and XAPP714 for more details.

For Virtex-5, Xilinx has gone a step further. Virtex-5 has hardware that automates the detection of programming data SEUs. Basically, it performs continuous readback in the background, checking the integrity of the configuration bitstream. A signal on the FRAME_ECC_VIRTEX5 block is asserted on error. See Xilinx's UG191 for more detailed information. This module does not provide automated correction.

7.1.6 SEFI Detection and Recovery

As discussed in section 5.3, there are three major regions in the Virtex architecture that are particularly vulnerable to SEFIs: the JTAG TAP controller, configuration control state machine power-on reset (POR), and the SelectMAP configuration pins. These SEFIs can be detected in these regions by the unusually large number of upsets they cause during a device readback. Other alternatives will be discussed below.

The JTAG TAP controller:

When the JTAG controller is being actively used as a configuration interface, a SEFI will cause noticeable results as the JTAG protocol is extremely ordered and is very sensitive to upsets. It is possible, however, that a SEFI in the JTAG controller will move the device into an undesirable state even when it is not being actively utilized as an interface. Compounding the problem, the Virtex family of devices do not make the reset pin (TRST) available to the user, denying designers the ability to hold the JTAG controller in reset while it is deployed. Fortunately, this small deficiency can be overcome by applying a pull-up resistor to the test mode select (TMS) pin and a free running clock to the test clock input (TCK) [17][25]. This method clocks in a continuous stream of ones and, in the event of an upset, it will only take 5 clock cycles to return the JTAG tap controller to a reset state (see the JTAG state machine in Figure 7.3).

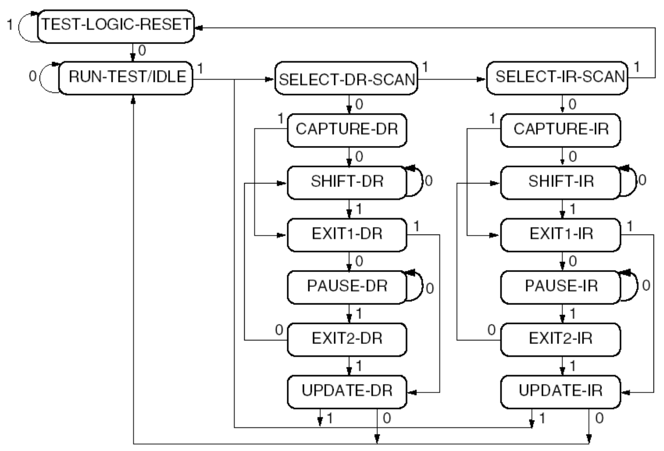


Figure 7.3: The TAP controller is a 16-state finite state machine. The TAP controller can be brought back to the TEST-LOGIC-RESET state in a worst case maximum of 5 clock cycles [48].

Configuration control state machine power-on reset (POR):

One SEFI that can occur within the configuration control state machine of a Virtex device is the power-on reset (POR) SEFI. In this case, the device behaves as if $\overline{PROGRAM}$ has been asserted — its configuration is cleared and the $DONE$ pin is driven low. Currently there is no mitigation for this type of SEFI other than to reconfigure the device and restart processing. Fortunately, the SEFI cross-section for the POR is extremely small and the chance of such a SEFI occurring is unlikely.

SelectMAP configuration pin upsets:

SEFIs within the programmable SelectMAP interface configuration pins ($D0-7$ for 8-bit interfaces or $D0-31$ for 32-bit interfaces, $\overline{WRITE}$, $\overline{CS}$, and $\overline{INIT}$) will result in what appears to be a unusually large number of upsets due to the inability to correctly read and/or write through the corrupted interface. Other possible side effects are the build up of SEUs in a design leading to its eventual failure due to a lack of repair or even corruption caused by scrubbing and/or repair processes acting on the flawed information provided by the bad interface. This is why it is essential to detect and repair a SEFI as quickly as possible when its occurs in the SelectMAP interface. Recovery takes place by asserting $\overline{PROGRAM}$ and reprogramming the device to return the SelectMAP interface to a functional state [17].

One method for detecting this SEFI is to periodically check a SelectMAP configuration register. Once a configuration register becomes corrupted an attempt is made to correct it. If the configuration register for the SelectMAP interface has not been corrected after the attempt to fix it a SEFI has occurred. Potential registers to check include the Control (CTL), the Configuration Option (COR), and the Frame Length (FLR) registers.

Another, perhaps more complete, method of checking for a corrupt SelectMAP interface is given in Figure 7.4. Note that this is a proposed approach that has not been fully tested but might be useful. Two write-read-check sequences utilizing a pre-determined bit sequence are written to a 32-bit scratch register such as the Frame Address Register (FAR)[1]. In the event that $D0-7$ have become corrupted it will be apparent after the first check due to the fact that registers corresponding to each programmable pin (see

[1]This assumes that the FAR is managed correctly by other configuration/readback operations.

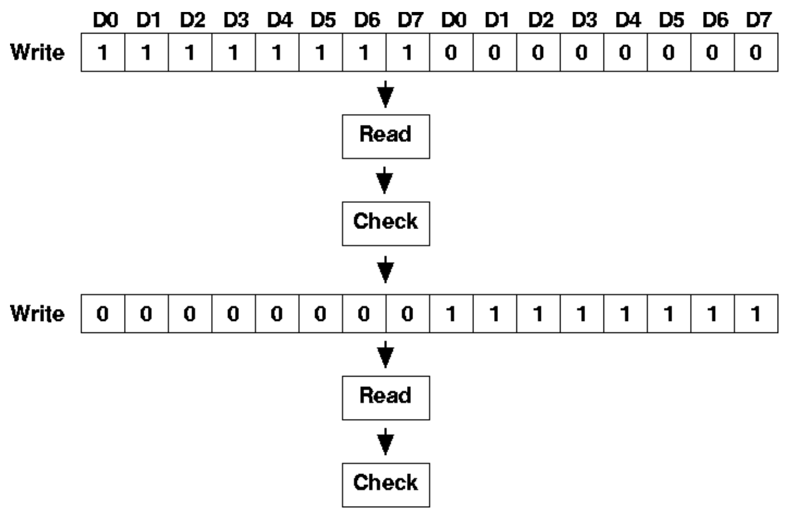


Figure 7.4: A more complete way of checking the SelectMAP interface. Two differing sequences of bit values are written to a scratch register and readback again to ensure the re-programmable pins of the SelectMAP interface are functioning correctly.

D0 - D7 in Figure 7.4) have not changed despite different bit values being written to the register. If the $\overline{WRITE}$ pin has been corrupted it will be apparent during the second read-check cycle when values do not change as they should. Finally, if the values from the register seem to be corrupted during both cycles there is a good chance that the $\overline{CS}$ pin has been corrupted.

7.2 Mitigation

The discussion of mitigation will concentrate on SEUs since problems with TID tolerance and SELs generally require modification of the devices silicon implementation [17], which is well beyond the scope of this document. While SETs could become a problem in the future due to increasingly smaller very deep sub-micron (VDSM) technologies being incorporated into FPGAs, they are currently not a concern for the Virtex family of devices.

7.2.1 Triple Modular Redundancy (TMR)

Conventional TMR in its simplest form involves triplicating logic modules (sometimes referred to as *TMR domains*) and using voters to arbitrate between them. In principle, when any of the design modules fail, the other two modules continue to operate correctly, and the voter outputs the correct value to the outside world. The intended purpose of TMR is to remove all single points of failure in a design, assuming perfect voters. In the actual application of conventional TMR a wide variety options exist (granularity, voting feedback, etc.) with different impacts on performance, area, and reliability. The recommended approach is to do full triplication of all logic (modules and voters) and signals (input, output, clock, and reset).

TMR and its variants can be combined with a partial, on-line reconfiguration repair method such as scrubbing to produce an effective SEU mitigation strategy. An excellent, albeit older, guide explaining the use of TMR in Xilinx Virtex 1000 designs is “Triple Module Redundancy Design Techniques for Virtex FPGAs” [47].

7.2.2 Xilinx Triple Modular Redundancy (XTMR)

There are two primary drawbacks to the conventional TMR approach when implemented in reconfigurable FPGAs. First, conventional TMR leaves designs vulnerable to SEUs and SETs in the voting circuitry. Second, conventional TMR does not provide a way of re-synchronizing state logic after repair processes such

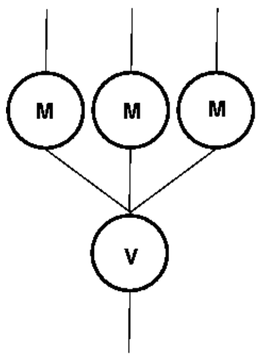


Figure 7.5: TMR in its most simplistic form involves functional modules (M) that are triplicated and run through a single voter (V) to mitigate SEUs.

as configuration scrubbing (see subsection 7.3.1 for more on scrubbing) have occurred. In order for triplicated state machine domains to operate continuously in a TMR-protected design they need to be synchronized with each other. Conventional TMR does not provide a method to do this.

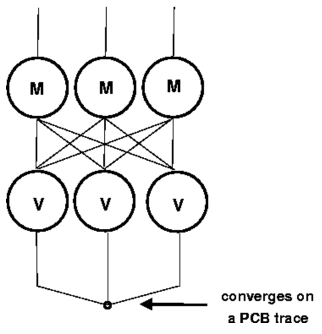


Figure 7.6: In Xilinx TMR functional modules (M) as well as voters (V) are triplicated to mitigate SETs in voting logic in addition to SEUs.

To meet the special needs of a TMR approach for reconfigurable FPGAs, the Xilinx Triple Module Redundancy (XTMR) approach for design triplication was devised by Xilinx in cooperation with Sandia National Labs and others [49]. XTMR solves many of the problems of standard TMR for re-configurable devices by addressing the triplication needs for three fundamental design blocks: inputs and throughput logic, feedback logic, and outputs. Specifically, XTMR eliminates single points of failure that exist in conventional TMR at voters by ensuring that all voters are triplicated and all redundant XTMR design domains begin and end on printed circuit board (PCB) traces, where they are not easily affected by radiation (see Figure 7.6).

XTMR (via the TMRTool) triplicates all inputs (see Figure 7.7), throughput (combinational and sequential) logic, and routing. Voters are not inserted at the inputs and each of the triplicated design domains operate independently of each other. Rather, majority voters are inserted further downstream in registered feedback paths and minority voters are inserted at triplicated output pins when the output pins converge to a single PCB trace.

The circuitry driving the majority voters in registered state logic feedback loops (see Figure 7.8) is designed such that the feedback logic for each state machine domain is a function of the current state of all three state machines. If an SEU upsets a state machine domain, the state machine domain affected is disabled until it is re-synchronize with its redundant partners during a repair process such as scrubbing. This is convenient as state logic can continue to operate uninterrupted in the presence of SEUs and SETs, a major advantage of the XTMR approach.

Finally, if an SEU occurs in the output circuitry of one of the domains, the associated minority voter will drive its output pin into a high impedance state effectively disabling any output (see Figure 7.9). It should be noted however, that even in the worst case scenario, where an SEU in the minority voter circuitry causes

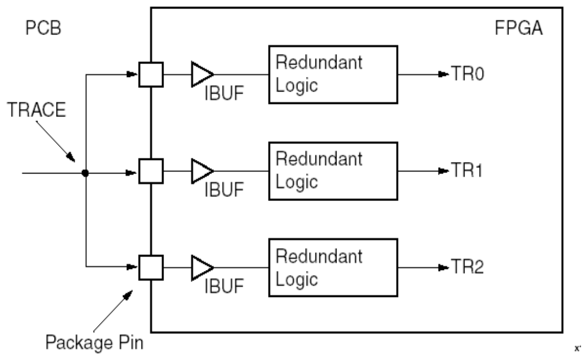


Figure 7.7: XTMR uses triple redundant input pins for every input signal [6].

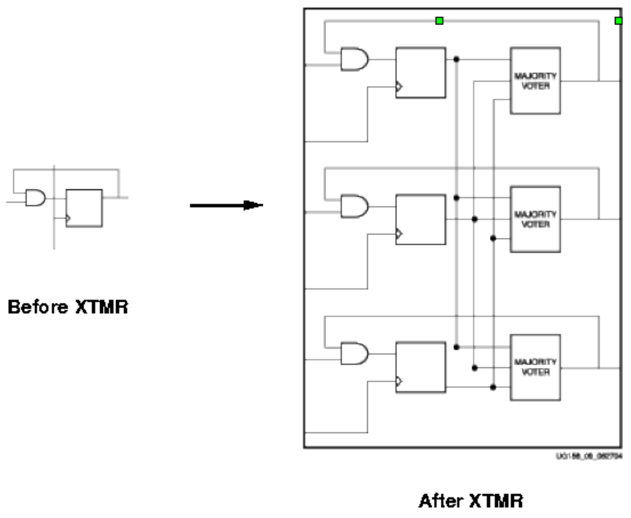


Figure 7.8: State logic before and after the XTMR process [6].

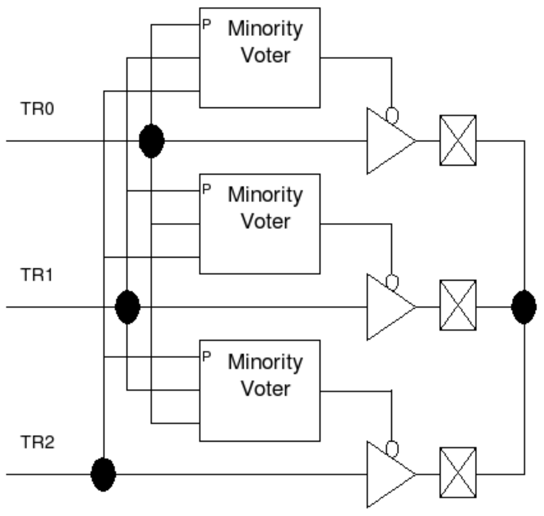


Figure 7.9: XTMR uses triple redundant output pins and minority voters for each output signal. If a one module is different from the others, its output pin is driven to High-Z [6].

it to disable a valid output, there are still two redundant domains providing valid outputs. An excellent guide and reference describing XTMR and its implementation via the Xilinx TMRTool is the "TMRTool User Guide" [49].

7.2.3 Partial Triple Modular Redundancy

Partial triple modular redundancy (also known as selective triple modular redundancy or STMR) is the process of judiciously selecting the most sensitive components in a design for triplication in an effort to trade off some reliability for a savings in resources.

BYU/LANL Partial TMR for Persistent Errors (BL-TMR)

BL-TMR is a variation of partial TMR (which is accomplished through an automated system called the BL-TMR Tool) developed jointly by BYU and LANL [30]. Like many other partial TMR schemes, BL-TMR increases a design's robustness by mitigating its most sensitive design structures. However, BL-TMR differs from other techniques in that it targets the feed-back structures of the design to substantially reduce the persistent cross section of a design (see Figure 7.10).

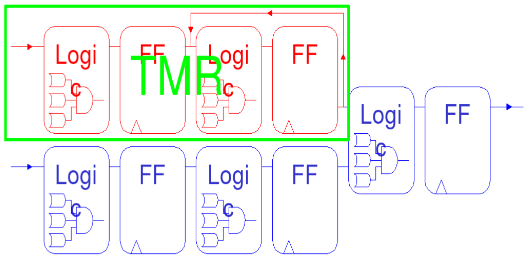


Figure 7.10: BL-TMR specifically targets feed back structures to reduce the persistent cross section of a design [31].

The BL-TMR tool works by breaking a design down into three different components—circuits with feedback, the input circuitry to these circuits (i.e., the input logic cone), and the output circuitry. Depending on what the designer requires, TMR can be applied to the circuitry with feedback only, to the circuitry with feedback plus the input logic cone (as in Figure 7.10) , or the circuitry with feedback plus the input logic cone plus the output logic, which is equivalent to full TMR.

Table 7.1 displays the device utilization of a counter (a feedback-intensive design) and a LANL developed digital signal processing (DSP) design on a XCV1000 platform (12,288 available slices) using both the full TMR and partial TMR abilities of the BL-TMR tool. The counter design—a worst case scenario for persistence—incorporated full TMR for mitigation. In the case of the LANL DSP kernel the BL-TMR tools were used to apply partial TMR only to the persistent cross-section of the design. Unlike the counter design, full TMR was not an option for adding robustness to the DSP design given the its initial size. Further, the DSP design is more representative of what a developer is likely to see when mitigating a typical DSP application.

As the table illustrates, the cost of partial TMR can be considerably smaller than the expected 200% increase in device utilization expected with full TMR. This is due to the fact that the ratio of logic that contributed to the persistent cross-section to the overall cross-section of the circuit is quite low. Hence, fewer circuit modules needed to be triplicated and voted.

Another point to note is that it is not unusual to see the growth of slices in a design with full TMR to exceed the anticipated 200% increase in logic utilization. This is due to how well logic is packed into slices and the contraints regarding how logic can be packed into slices in the triplicated design. For instance, with each TMR domain having its own clock, the flip-flops of two TMR domains cannot co-exist in a single slice since a slice only supports a single clock for its flip-flops. As a result, two slices must be used instead of just one in this case.

	Size (slices)		
Design	Unmitigated	Mitigated	% Increase
Counter	2,151	11,251 (Full TMR)	423%
DSP Kernel	5,775	7,936 (Partial TMR)	37%

Table 7.1: Device utilization comparing full TMR to partial TMR provided by the BL-TMR tool [31].

University of South Florida/Honeywell Space Systems method of STMR

Researchers at the University of Florida (USF) and Honeywell Space Systems have devised a variation of STMR that with a small loss of SEU immunity can reduce the area overhead of a hardened circuit to 2/3 of the area required by standard TMR. This method along with the readback and reconfiguration features of the Virtex device can be used to obtain a very high immunity against SEUs [40]. The approach is based on the ability to identify and triplicate the logic most likely to propagate errors rather than to triplicate logic that is likely to mask errors.

7.2.4 Half-Latch Replacement

As discussed in Section 4.3.1 Xilinx half-latches are prone to SEUs and are not directly observable or controllable through the configuration bitstream making it difficult to observe or correct SEUs. One solution is to replace half-latches with explicit constant sources that can be configured (and fixed) directly through FPGA programming data [18].

With this said, half-latch replacement is not without its trade offs. By replacing half-latches a designer is effectively increasing the amount of logic being used in a design and may introduce single points of failure (and, hence, additional SEU cross-section), if the explicit constant sources are not appropriately triplicated.

Half-Latch Alternatives

Figure 7.11 provides three half-latch alternatives that are both observable and configurable directly through Xilinx device programming data. The IOB source for constants (see Figure 7.11 a.) is basically an FPGA input pin driven externally by a logic “0” or “1”, the LUT ROM source (see Figure 7.11 b.) is a a LUT ROM filled with “0” or“1” values, and the flip-flop source for constants (see Figure 7.11 c.) is a self-correcting flip-flop source.

Half-Latch Replacement Approaches

Half-latches can be replaced at various stages in the FPGA design cycle using different types of tools (see Figure 7.12). At the source code level (.vhd, .sch), careful coding can be used to make sure that half-latches are never introduced in the first place. However, preventing the usage of half-latches at the source code level is extremely tedious and difficult to automate.

On the other end of the spectrum half-latches could, in theory, be replaced at the bitstream level (.bit) using tools such as Xilinx’s JBits. Unfortunately, there are some resources in the Virtex-I bitstream which JBits is unable to edit and the effects of half-latch replacement on timing is not directly observable. Furthermore, JBits only supports the Virtex-I and the Virtex-II devices. Additionally, modifications at this late stage in the design process may result in less optimal solutions since placed and routed designs represented as bitstreams are not easy to modify and effectively require special knowledge that Xilinx alone has.

A better solution is to replace half-latches at either the netlist (.edf,.ngd) or physical database (.ncd,.xdl) levels. At the netlist level half-latches can be removed using TMRTool provided by Xilinx or using the BL-TMR tool. One of the advantages to this approach is that TMR domains are correctly preserved. The disadvantage is that not all half-latches may be replaced in the design when using netlist-level approaches due to design optimizations during technology mapping or later operations or due to other limitations in what can be expressed or observed at the netlist level. For instance, the BL-TMR tool cannot perform half-latch replacement for design modules expressed in Xilinx’s NGC format—a common format for Xilinx CoreGen cores. This latter issue should not be a limitation of Xilinx’s TMRTool.

An example of half-latch replacement at the physical database level are two tools by LANL called RadDRC (for Virtex-I) and RadDRC-II (for Virtex-II). RadDRC parses the XDL physical database representation of a design to locate half-latch issues and then generates a new XDL design, automatically replacing half-latch constants with observable, repairable constants sources. The advantage to this approach are that all of the half-latches in a design are replaced since it occurs in the design cycle after the placement and routing phase of the design flow. A TMR-aware form of half-latch replacement can be performed with RadDRC if TMR was inserted by BL-TMR. For other methods of adding TMR, RadDRC might introduce some additional cross-section due to being unaware of which logic belongs to which TMR domain.

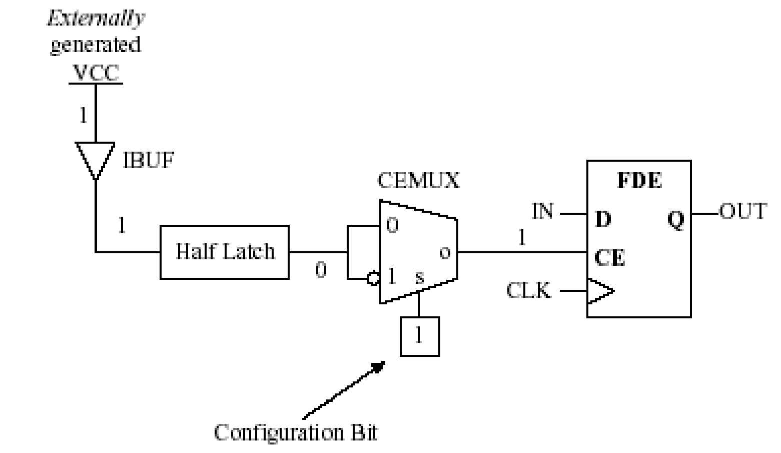
It is suggested that either the Xilinx TMRTool or the BL-TMR tool along with LANL’s RadDRC tool during Virtex-I development at the netlist and physical database levels to ensure that TMR domains are appropriately preserved (with the XTMR or BL-TMR tool) and that all half-latches have been removed (with the RadDRC tool). The RadDRC tool, in this case, can be used as a method of assuring that XTMR and BL-TMR (or other TMR-aware methods for half-latch replacement) have removed all half-latches from the design.

Finally, it should noted that there are some special half latches that cannot be controlled via the EDIF netlist level. One such half-latch is the global clock buffer enable present in the Virtex-I device, since a global clock buffer primitive with an enable cannot be directly instanced. This is a case that can be mitigated at the physical database level by RadDRC.

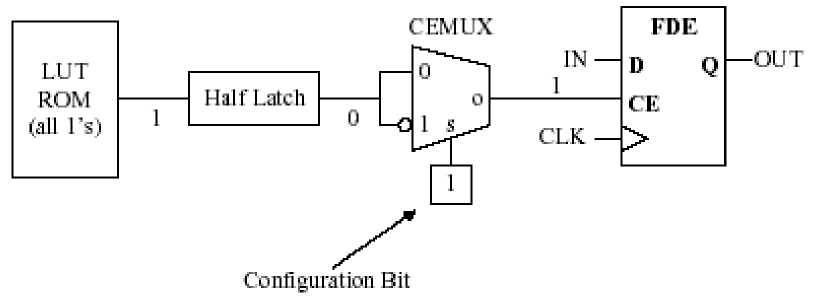
7.2.5 Domain-Crossing Events

A domain-crossing event (DCE) occurs when two or more TMR domains in a circuit are affected by one radiation event and enter into the same incorrect operational state. A DCE could be caused by an MBU upsetting the logic of two separate TMR domains, effectively defeating any mitigation provided.

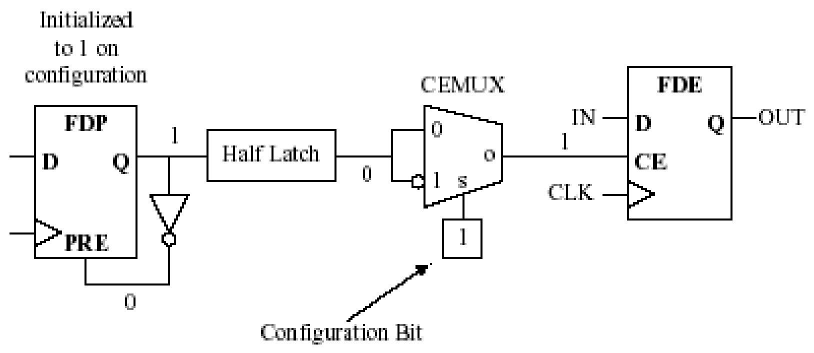
One such vulnerability is possible if the designer is not careful and a pair of flip-flops from multiple TMR domains ends up in the same slice during optimization. In such a case single points of failure are introduced where the clock, clock enable, and set/reset signals branch out to the separate TMR domains occupying the same slice making them vulnerable to SBUs (see Figure 7.13). Single point failures of this nature can be resolved by triplicating clock, clock enable, and/or set/reset signals. When these signals are triplicated, components of each TMR domain are mapped into separate slices, eliminating the source of the single point



(a) IOB source for constants



(b) LUT ROM source for constants



(c) Flip-flop source for constants

Figure 7.11: Alternatives to the Xilinx half-latch [18].

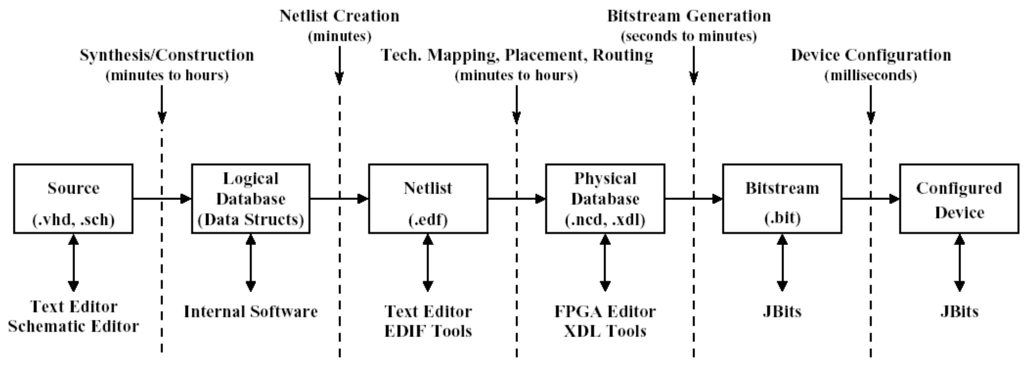


Figure 7.12: FPGA design flow and half-latch replacement [5].

failure. While this placement fix will provide an immediate fix for single points of failure, in the Virtex-II it is still possible to have MBU-induced DCEs when no single points of failure exist.

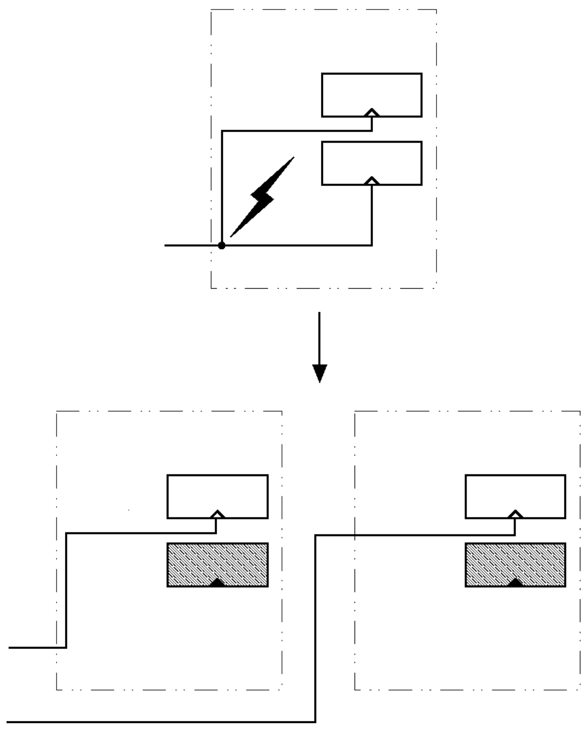


Figure 7.13: A single point of failure in a slice can be removed by triplicating CLK, CE, and S/R signals at the expense of space.

Separating logic domains is not without its trade offs, though. Once the logic domains have been separated it is possible that one logic cell may remain unused in each of the slices due to limitations imposed by timing and placement constraints. The shaded logic cells in Figure 7.13 illustrate one such possible under-utilization of resources. The exact amount of resources underutilized in the triplication process depends on the design and the constraints placed on technology mapping. However, experiments at BYU, LANL, and Sandia National Laboratories have shown that a 5x size utilization ratio of untriplicated logic to triplicated logic in terms of slices is not uncommon in designs. That is, a design that occupies 20% of a device will expand to "fill" the entire device once full TMR is applied such that little if any additional logic can be added to the design. In reality the number of flip-flops and LUTs only increase by a factor of three, but the inability to pack the logic tightly results in more (underutilized) slices being used.

In cases where triplicated clocks or I/O signals are not desirable, other methods besides triplicating clock and reset logic do exist to separate TMR domains among slices in a design. These include time-consuming manual floorplanning and/or technology mapping in which each TMR domain is physically separated to prevent SBUs and MBUs from causing DCEs. The exact amount of separation depends on how much protection against SBUs and MBUs the designer requires. Research at LANL shows that MBUs rarely cross CLB columns [34] but it is felt even this much separation is over constraining. A designer can be confident with the isolation provided by placing TMR domains separated by a CLB. Another method that BYU and LANL have experimented with is to temporarily triplicate clock signal logic to separate the domains among slices and then remove the triplicated logic later on at the XDL/FPGA editor level.

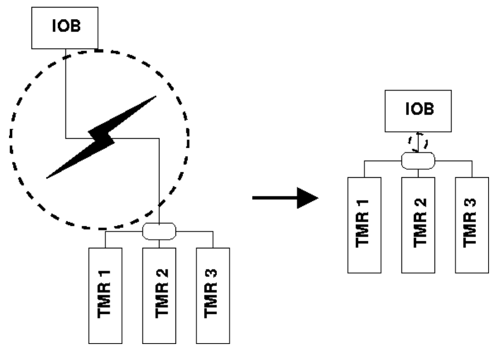


Figure 7.14: By reducing the cross sectional areas of single points of failure (the circled region) DCEs can be avoided.

Scarce resources that cannot afford to be triplicated also introduce potential single points of failure. For example, IOBs are a scarce resource and often it does not make sense due to resource constraints to triplicate them, producing a single point of failure. The best policy in this case is to reduce the single point of failure cross-section as much as possible. In Figure 7.14 reducing the physical distance between the IOB and the TMR-protected logic reduces the amount of routing resources used and thus the single point of failure cross-section (the circled section in Figure 7.14).

Routing networks are another portion of the FPGA design. Design that are vulnerable to DCEs are often defeated when the routing network between two TMR modules is corrupted by an SBU or MBU DCE. SBUs and MBUs can effect the routing networks switch boxes by upsetting the memory controlling the transistors and muxs. For example, in Figure 7.15 two signals (S1 and S2) propagate along different paths through the same switch box. A SEU occurs within a memory element and the two signals are shorted.

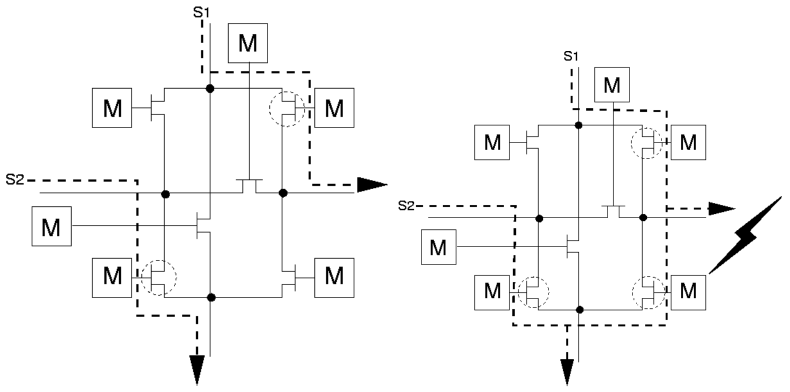


Figure 7.15: Two signals (S1 and S2) propagate along different paths through the same switch box (left). After an SEU occurs in a memory element the two paths are shorted (right).

We believe that the routing in the Virtex-I line of devices is more prone to SBU DCE upsets then later architectures due to its routing architecture. Specifically, single length wires in Virtex-I devices often use pass transistors to connect to other wires instead of using only muxes and buffers to connect slices to CLB routing resources as is the case with the Virtex-II, Virtex-II Pro, and Virtex 4 devices.

Finally, when an SBU occurs within LUT or BRAM it does not result in a DCE, as long as the resource is not shared between domains. The upset by definition is within the same TMR domain and the fault should be isolated. As for inter-LUT MBU DCEs, they are not yet well understood and this is an area of on-going research. Initial research indicates that, for the Virtex-II, these MBUs are far less problematic than MBUs in the routing network. While inter-BRAM MBU DCEs might not be uncommon in Virtex-4 and Virtex-5 devices, we have yet to study the affect of MBUs in the BRAM on these devices.

7.3 Repair

Repair is a critical part of any SEU mitigation strategy. Without a repair process such as scrubbing, SEU induced faults will build up in a device and eventually break any redundancy provided by mitigation strategies such as TMR [32].

The exact repair method used depends on what the needs of the mission are. If a designer only needs to ensure that the bitstream is maintained and no corruptions are allowed to build up, a method such as scrubbing can be used. If the mission additionally requires that data is gathered on the repair and mitigation process such as the number of SEUs that have taken place and where they have occurred, a read and repair method can be used instead. In any case, we strongly recommend consulting Xilinx before implementing your own scrubbing algorithms since their implementation can have a number of subtleties.

7.3.1 Scrubbing

Scrubbing is typically the process of using partial reconfiguration to reload the used portion of the FPGAs configuration bitstream at periodic intervals, removing any SEUs that may have accumulated in the bitstream between scrubs. An occasional readback is used to ensure the scrubbing process is actually occurring. This method requires substantially less overhead then other repair methods since it requires minimal readback and data verification operations and does not require the generation of repair-specific partial configuration bitstreams when reloading the data frames. The trade off is that SEU data cannot be gathered from this process as readback is only used to ensure that the interface is active, and, thus, scrubbing is occurring. Further, it does increase the risk that bad programming data may be written to the FPGA since it is writing the entire bitstream regularly and the configuration logic itself is susceptible to SEUs.

For the Virtex family FPGA devices, the portion of the FPGA bitstream that is generally scrubbed includes all of the configuration data for the global clock (GCLK), BRAM interconnect, IOB, and CLB configuration data. The rest of the bitstream contains the BRAM data which is generally not touched in the scrubbing process to avoid corruption (see Sections 4.3.1 and 7.1.1 as well as the next subsection for some additional information).

There are, however, limitations one must be aware of when using a scrubbing process. For instance, an error-free readback of the configuration bitstream does not guarantee that an SEU did not occur. The FPGA contains hidden state that cannot be read back and upsets to hidden state can conceivably cause errors in the design without any bitstream errors being detected. Furthermore, SEUs in flip-flop and half-latch states can occur without disturbing the bitstream. Additionally, though no programming data readback is being performed, designers must still be careful if the FPGA user designs contain LUT RAM or Block RAMs when scrubbing. If these memories are scrubbed unintentionally, the user design's state can be corrupted.

Though not ideal for most Virtex FPGAs and most mission scenarios from LANL's point of view, "blind" scrubbing is especially not recommended for newer devices. Since many of the control registers used in scrubbing are not SEU-immune, they can be affected by SEUs while scrubbing. In the Virtex-4, this situation could lead to a high current state, called the Scrub SEFI, that can dramatically increase current draw, power dissipation and, potentially, damage the device. For this reason, this method of scrubbing is strongly discouraged for newer devices.

The ABORT Sequence

When using the SelectMAP interface (see subsection C.4.1) as the primary configuration interface for a partial on-line reconfiguration process such as scrubbing, it is good practice to use the SelectMAP ABORT sequence to clear any errors that may have accumulated in the configuration logic itself due to SEUs. A conservative approach would be to use an ABORT sequence and resynchronize the SelectMAP interface before each scrubbing or readback process takes place. Another approach (and the one that is used in the CFE) is to use an ABORT sequence and resynchronize after a problem has been detected during a periodic readback check. The ABORT sequence is covered in detail in "Correcting Single-Event Upsets Through Virtex Partial Configuration" [44].

Scrub Rate

In general the rule of thumb for a device's scrub rate is to scrub ten times faster than the upset rate. For example, the scrub rate for a XV1000 device (5,810,048 configuration latches not including BRAM) using the SelectMAP interface (50 MB/s) for an environment that experiences one SEU per hour is:

$$\begin{aligned}
\textit{scrub cycle time} &= \frac{5,810,048 \text{ configuration latches}}{50\frac{MB}{s}} && (7.1)\\
&= 11.6 \text{ ms} && (7.2)\\
& && (7.3)\\
\textit{scrub rate} &= \frac{1 \text{ hr}}{10} && (7.4)\\
&= 6 \text{ min} && (7.5)\\
\textit{scrub rate percentage} &= \frac{11.6 \text{ ms}}{6 \text{ min}} && (7.6)\\
&= .19\% && (7.7)
\end{aligned}$$

This is the maximum scrub cycle time required to ensure proper scrubbing occurs. Often it is more convenient and reduces the complexity of scrubbing design to use a smaller time interval between scrub cycles (such as scrubbing whenever possible). As for power considerations, the CFE scrubber consumes roughly 50 to 100 mW of power per FPGA scrubbed.

7.3.2 Read and Repair

The alternate to simple scrubbing, Read and Repair, involves using partial, on-line re-configuration to correct the upset once a detection method (see section 7.1) has determined an SEU has occurred. This allows much more data can be gathered on upsets that occur within the system and reduces the risk of writing bad configuration data due to an SEU affecting configuration logic since only the configuration data frames that have errors are written. The downside to this method is the overhead requirements for implementation.

Like the simple scrubbing approach there is a possibility for user memory (LUT RAM, BRAM, etc.) corruption through configuration interface contention while writing programming data. Unfortunately, reading programming data, an essential part of this approach, also has the potential of corrupting programming data for most Virtex family devices. As mentioned in 7.1.1, interface contention occurs in Virtex family devices when trying to either write or read the configuration data of Distributed RAM within Configurable Logic Blocks (CLBs) while the user design is also trying to access the same RAMs. Examples of Distributed Select RAM include LUTs being used as memory elements (i.e., RAM16X1S) and Shift Registers (i.e., SRL16s).

As a result of this issue, we must either disable the configuration data fault manager while running a design that uses distributed memory elements, stop the clock to do readback, or temporarily remove the sections of the bitstream that contains these primitives from SEU management. In the Virtex-I family of devices this poses a particular problem, as the configuration data for a slices's LUTs are spread out across 32 frames of configuration data. In the Virtex-II and -II Pro device families, the LUT values are contained within 2 frames of configuration data for each CLB column, making it easier to "skip" the data during a repair process such as scrubbing. For Virtex-4, as discussed in 7.1.1, enabling GLUTMASK provides a nice solution to this issue for both writing and reading CLB configuration data in addition to avoiding the Readback SEFI issue (see 4.3.1). Without GLUTMASK enabled, though, a designer must use the same caution as with other Xilinx FPGAs when performing writes or reads of configuration data in Virtex-4.

To help avoid this problem, SLR16s can fortunately be replaced in existing designs through the use of tools such as the Xilinx TMRTool and BL-TMR, which replace each SRL16 with 16 CLB slice registers and one LUT. This, of course, requires a significant number of resources. TMRTool, however, cannot automatically extract other Distributed RAM elements. With the exception of Virtex-4's GLUTMASK mode, in general, you should avoid using Distributed RAM whenever possible, using Block RAM instead when RAM is needed.

For similar reasons as those described above, the on-chip Block SelectRAM (or BRAM) memory cannot be reliably read back without stopping the design's clock. Further, the output registers of the BRAMs suffer what is known as "Destructive State Sampling" during readback and become corrupted [19]. Thus, for BRAM arrays, error detection and correction must be handled via ECC or checksums since readback of BRAM cannot be reliably performed while the design is running. However, it should be noted that if BRAM is used strictly as a ROM and readback is disabled it can (theoretically) be scrubbed without risk of corruption. Also note that in any scrubbing process involving BRAM, while it is not usually feasible to scrub the BRAM data itself, it is absolutely essential to scrub the BRAM interconnect (BRAMi) portion of the bitstream.

As a related issue and a good example of why to consult with Xilinx when creating a scrubber, LANL noticed (and Xilinx has confirmed) that reading the last two "pad" frames at the end of the BRAM interconnect portion of the Virtex-4's bitstream can cause the data corruption mentioned above. In affect, reading either of these two frames effectively causes the configuration logic to prefetch the first frame of the BRAM data portion of the bitstream, leading to the contention issues described eariler if that portion of BRAM is employed in the user design.

Chapter 8

Testing and Validation

This chapter provides insight into how testing methodologies and tools are used to gather and verify data on dose-related effects (see chapter 3), single-event effects (see chapter 4), and device characterization (see chapter 5). Much of this text is based on the testing methodology that LANL has used in the past to obtain radiation characterization data when evaluating the Virtex family of devices. A significant emphasis is placed on predicting the SEU performance of user FPGA designs.

8.1 Testing for Survivability

The first consideration for use of any FPGA technology in space is a survivability demonstration. This generally involves measuring the devices tolerance to dose-related effects, especially TID, and the LET at which SEL begins to occur. The dose-related effects test involves high and low dose rate exposures typically from a ^{60}Co source. These tests are usually performed statically since the particular FPGA user design has little effect on either SELs or dose-related effects.

8.2 SEU Testing/Validation Methodology

SEU testing is usually done in two steps: static testing to determine the baseline sensitivity of the memory cells and dynamic testing to determine how the resources and the designs manifest errors. Static testing is discussed in detail in section 5.1. What follows is a discussion of dynamic testing.

Dynamic testing is used to measure the dynamic response of particular resources, such as IOBs, or a particular design. In this manner, dynamic testing should provide insight into how faults manifest (stuck-at faults with IOBs or bad data in designs) in addition to the single points of failure in mitigated designs. This is useful because static testing does not detect the contribution to sensitivity from combinational circuitry operating at system speeds nor does it determine the consequence of a specific upset in a specific FPGA design, which is necessary to validate mitigation schemes.

The problem with dynamic testing is that it is very difficult to perform in an accelerator environment due to poor fault attribution and the extreme costs. A trade-off exists when collecting accelerator data between SEU rates and fault attribution – either you have good fault attribution and low error rates or high error rates and poor fault attribution (i.e., ambiguity regarding which particular upset caused the problem). The more ambiguous the fault attribution, the more difficult the analysis of the data gathered from the accelerator tests will prove to be. For example imagine several SEUs occur for different bits in a short time interval and then an output error occurs. The cause of that output error could be any of those SEUs depending on the error's propagation time or a combination of the SEUs.

The level of this difficulty depends somewhat on the output error sample rate and scrubbing rate of the test fixture being used in relation to the SEU rate. The higher these sample rates are for a given SEU rate, the easier it is isolate the effects of a single SEU while collecting SEU data at a high rate. High SEU rates are desirable so that signficant SEU and cross-section statistics are gathered.

Then again, merely increasing a testing fixture's sampling rate is not a perfect solution either. Depending on the time it takes for an error to propagate to the outputs, it may still be difficult to say with certainty which SEU caused a particular fault. For example, assume that during an experiment only one SEU occurs during an error detection and SEU scrubbing cycle (a single sample time). During each sample period, the following happens: an SEU occurs; the resulting error (if one results) propagates through the circuit; the test fixture (hardware and software) determines the error state; the fixture removes the SEU that causes the error (assuming it occurred in the FPGA's bitstream); and the test fixture resets the design to a known state. No matter how high the sample rates are, there is still a possibility of multiple SEUs occur during this process, complicating the attribution of the fault to a specific SEU.

Also, since the likelihood of error propagation is dependent on the input data combinations and the state of the circuit at the time of the SEU, allowing time for the error to propagate to an output is necessary for good fault detection. Of course, not all error states propagate immediately to the outputs of the circuit, so waiting a single sample period for discovering the effects of latent bad state may not be sufficient. Additionally, the SEU might occur late in the sample period, reducing the time for error propagation.

As an additional subtlety, several options exist for how to treat resets during an experiment. One option is to reset the state of the design after each sample period. This reduces the chance that the effects of the current SEU will be attributed to an SEU in the following sample period. Clearing out latent bad state also has a downside: an SEU that causes latent bad state may not be recognized because the bad state is not allowed to remain until it causes an output error. As a result, the sensitive cross-section might be underestimated.

Another approach is to reset the design to a known good state after an output error occurs. This approach is a little more realistic in operation in that it allows more of the bad latent state to propagate to the circuit's outputs because resets only occur when an output error occurs. As a result, the experimenter will have a better estimation of the design's SEU cross-section, but there is additional ambiguity regarding the source of the error in non-trivial designs since it can be difficult to predict or model which particular SEU caused a particular error. Again this is a function of the state of the design at the occurrence of the SEU, the input data to the circuit, and the existence of latent bad state—all of which may be hard to know considering the real-time nature of this testing approach.

Accelerator testing of candidate space designs can be very expensive. To illustrate this point, assume you have a design you would like to dynamically test in an accelerator environment. You would like to observe 1,000,000 SEUs within your design for thorough testing of the design. Your test fixture takes one sample per second and the beam power provides 1 SEU per sample. Taking into account that accelerator time runs about \$500 an hour minimum, back of the envelope calculations show:

$$\begin{aligned}
total\ test\ time &= \frac{1,000,000 SEUs}{1\frac{SEU}{s}} \times 60\frac{s}{min} && (8.1)\\
&= 16666.67\,\text{min} && (8.2)\\
&= 277.78\,\text{hours} && (8.3)\\
& && (8.4)\\
total\ cost &= 277.78\,\text{hours} \times \$500\,\text{per accelerator hour} && (8.5)\\
&= \$138,888.88 && (8.6)
\end{aligned}$$

So, at this rate, it will cost you \$138,888.88 and 277.78 hours of beam time plus additional costs to get to the accelerator facilities, setup your testing apparatus to adequately test your design, and the labor to monitor the operation. Of course, with only 1 million upsets, the test coverage of a design it not very complete considering that a Virtex-4 LX200 FPGA can have about 50 million programming bits in their bitstreams and the limited number of input test vectors used during the experiments. A more thorough examination of a design's SEU performance could easily cost an order of magnitude more money.

Compare this to a design dynamically tested on an SEU emulator (see section 8.4) or through fault injection into the actual platform (see section 8.3). Using the LANL SEU V1 emulator 5,800,000 SEUs can be emulated in a design through fault injection in about 30 minutes. Since the emulated SEUs occur in an ordered, methodical manner and time is provided between each SEU for error propagation, the problems with poor fault isolation disappear. Cost-wise in both money and time the SEU emulator and fault injection systems are much cheaper to implement than continuously performing accelerator tests. It is for these reasons that we recommend dynamic testing be performed with the SEU emulator during a design's development phase and then finally it be verified by dynamic accelerator testing once it is deemed mature.

8.3 Fault Injection within the Actual Platform

Based on LANL's experience with CFE, performing fault injection within a design while it is incorporated into its host platform is extremely critical. Not every scenario possible can be emulated by stand-alone static and dynamic SEU testing. By inducing faults into an FPGA sub-system while it is integrated into other satellite sub-systems a designer can greatly reduce the chance of faults creating unexpected results.

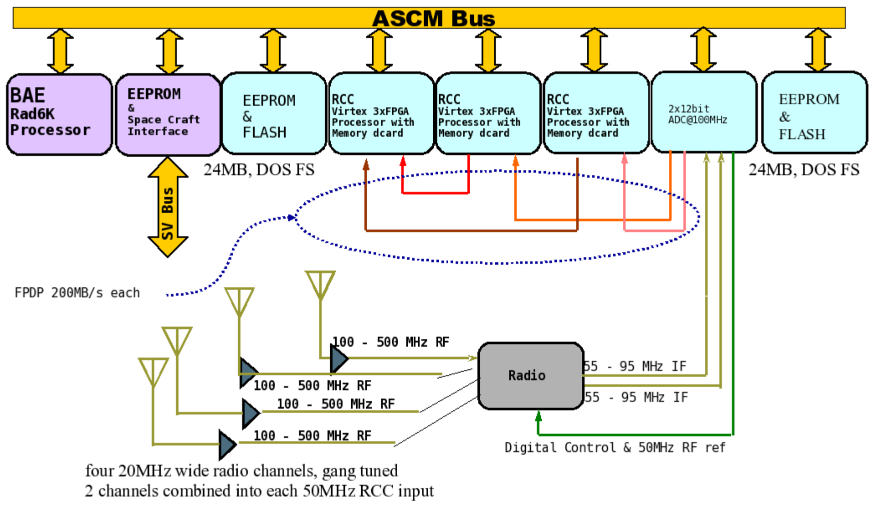

ASCM Bus
BAE Rad6K Processor
EEPROM & Space Craft Interface
EEPROM & FLASH
RCC Virtex 3xFPGA Processor with Memory dcard
RCC Virtex 3xFPGA Processor with Memory dcard
RCC Virtex 3xFPGA Processor with Memory dcard
2x12bit ADC@100MHz
EEPROM & FLASH
24MB, DOS FS
24MB, DOS FS
SV Bus
FPDP 200MB/s each
100 - 500 MHz RF
100 - 500 MHz RF
100 - 500 MHz RF
100 - 500 MHz RF
Radio
55 - 95 MHz IF
55 - 95 MHz IF
Digital Control & 50MHz RF ref
four 20MHz wide radio channels, gang tuned
2 channels combined into each 50MHz RCC input

Figure 8.1: The Cibola Flight Experiments reconfigurable radio payload [17]

The CFE payload sub-system (see Figure 8.1) uses a software/hardware-based fault injection system to emulate upsets within the actual hardware and test the CFE on-orbit fault detection and correction system (see Figure 8.2). The CFE payload is a reconfigurable radio based on multiple Virtex FPGA blocks (labeled RCC). These blocks are managed by a BAE RAD6000 processor and a rad-hard Actel FPGA. The Actel serves both as an interface between the RAD6000 device and the Virtex RCC modules as well as a fault manager for the Virtex FPGAs. It scans each Virtex FPGA device for upsets by performing a readback on the configuration stream of each device and creating a CRC for it. This CRC is then compared to golden CRCs which have been stored in hardened memory. The readback process itself takes roughly 180 ms to readback three devices and occurs continuously. If there are any discrepancies the frames that contain them are reloaded. More information on the CFE payload and mission can be found in Appendix A.

The fault-injection process itself begins when a list of desired SEU hit locations are up-loaded into the RAD6000 processor. From there the RAD6000 disables the Actel FPGA fault management system, takes a frame from the desired configuration memory location to corrupt (located in one of the RCC modules), corrupts a bit within the frame, and writes the corrupted frame to the FPGA. Readback and the Actel FPGA fault management system are then re-activated. Observation has shown that it takes roughly tens of milliseconds for the the system to fully recover.

We would like to stress that implementing some method of fault injection into a flight payload is essential to being able to test the system in a more realistic, flight-like manner. Again, it can enable the discovery of unexpected SEU consequences at the system level and it can be extremely useful for directly validating SEU detection and correction software and hardware.

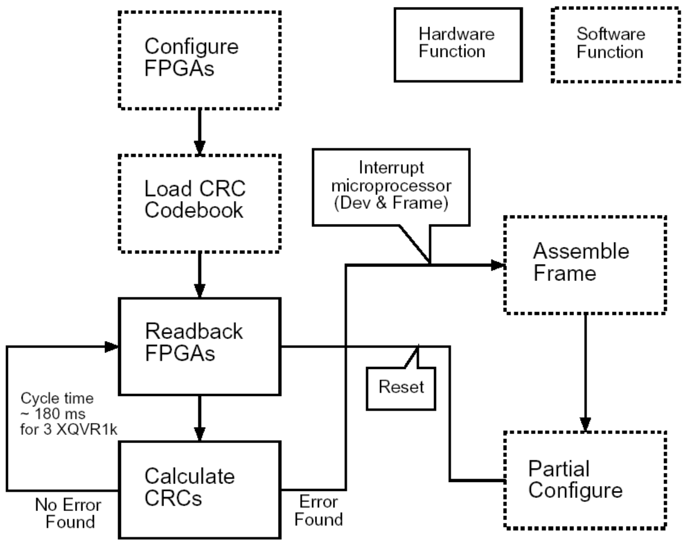


Figure 8.2: On-Orbit SEU-Induced Fault Detection and Correction [16]

8.4 SEU Emulators

LANL has created several SEU emulators to test the behavior of FPGA designs in the presence of SEUs within the configuration memory. Because all information about an FPGA design is stored in the configuration bitstream, whenever the state of a bit within the configuration memory is upset the function of the FPGA design may change. Signals may be rerouted, logic functions changed, or even the clock disconnected. The emulator monitors a design to detect if a configuration upset causes an output error.

The original LANL SEU emulator is based on the SLAAC1-V board, a high-speed FPGA board containing three Xilinx Virtex 1000 FPGAs based on an open design that was developed as part of the DARPA ACS program. The architecture of the SEU emulator mapped to the SLAAC1-V board is shown in Figure 8.3. PE0 is used to provide stimulus to designs in PE1 and PE2 which operate synchronously and, under normal circumstances, behave identically. During SEU emulation, the configuration memory of PE1, the design under test (DUT), is artificially upset through partial reconfiguration, which changes the contents of the configuration bitstream stored on the FPGA. The design is then executed to determine its true behavior in the presence of a configuration bitstream SEU. PE0 monitors the circuit outputs of PE1 and PE2 to determine if the introduction of an artificial configuration upset into the bitstream of PE1 has caused an output error. If so, the configuration bit which was upset is marked as a "sensitive" location.

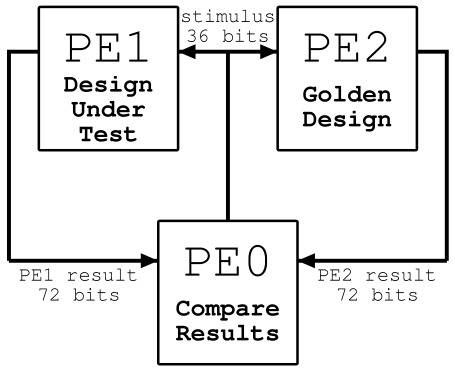


Figure 8.3: SLAAC-1v SEU emulator implementation [22].

Because there is such a large number of configuration bits within the bitstream, it is essential that this test be performed as quickly as possible. The SLAAC-1V Board supports high-speed partial reconfiguration to accomplish this. The usual test procedure with the SLAAC1- V board tests all individual configuration bit, meaning that the entire configuration bitstream can be tested in approximately 30 minutes. Since the tests are sensitive to input combinations, often tests are performed multiple times to achieve better test coverage through using additional input data vectors.

Test capability wise, the SEU emulation process has the ability to emulate 99.58% (not including half-latches) of a XCV1000's static-cross section via configuration bits (see Figure 8.4). The 0.42% (not including some hidden architectural features) that remains untested is the devices flip-flop state.

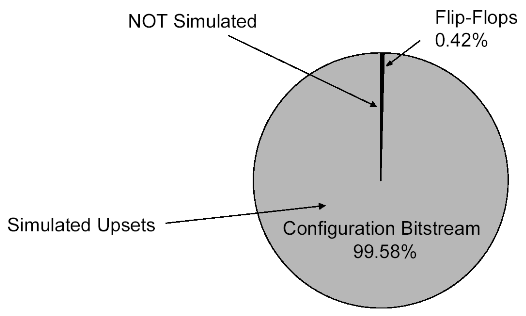


Figure 8.4: The SEU Emulation process has the ability to simulate exactly 99.58% of a Virtex 1000's static-cross section [22].

Since this original SEU emulator, LANL has moved toward a more general and flexible emulator framework. By abstracting the interfaces for reading and writing configuration data from FPGA hardware as well as the interfaces for manipulating configuration bitstreams, this framework has been extended to support several generations of Xilinx FPGAs, including Virtex-II, Virtex-II Pro, Virtex-4, and Virtex-5. The first example of this SEU emulation framework came with the creation of LANL's Virtex-II SEU Emulator. A brief but detailed guide to the LANL SEU emulator can be found in "A Brief Description of the Virtex-II SEU Emulator Hardware and Its Construction"[20]. This SEU emulator is based on Xilinx AFX Virtex-II FG456 boards (HW-AFX-FG456-200) and uses COTS hardware with the addition of some custom software, custom firmware and a small, inexpensive custom PCB. Recently, this setup has been translated to a Virtex-4 SEU Emulator using the XRTC SEAKR test board and daughter cards.

8.5 Accelerator Tests

The purpose of radiation testing within an accelerator facility is to perform dose-related testing, determine the static cross section of a device, and validate the results of SEU emulation (i.e. dynamic testing). In general, accelerator testing is fairly time intensive and costly so it is often used only when absolutely necessary.

The proton radiation test fixture that LANL uses to test the XCV1000 line of devices is a modification of the SLAAC1-V board outfitted with a socket for the convenient replacement of the DUT FPGA (PE1/X1). The idea is to irradiate the DUT while operating synchronously with a shielded golden design. See Figure 8.5

Figure 8.6 gives the flow chart of a typical dynamic accelerator test. SEUs are slowly introduced into the DUT by exposure to a proton or heavy ion beam. During this exposure the real time comparator monitors the output for errors (i.e., when the output of the DUT does not equal the output of the golden design) and notes the time of such events. This is followed by a readback of the DUT device configuration bitstream, recording the time and location of upsets. Bitstream upsets are later repaired by a partial reconfiguration process. If any output errors have been encountered the DUT and golden device are reset after they have occurred to resynchronize the designs.

The actual SLAAC1-V test setup used at the proton accelerator is shown in Figure 8.7 and Figure 8.8. A PCI extender card was used to lift the SLAAC1-V board away from main body of the host Linux PC. This prevents the sensitive PC components from being exposed to the proton beam while still allowing PE1, the socketed DUT, to be irradiated with protons. A socket was used for the DUT so that the Virtex part could

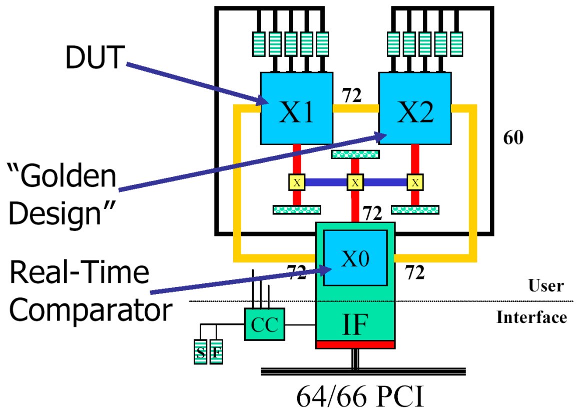


Figure 8.5: The SLAAC-1v Proton Radiation Test Fixture [23].

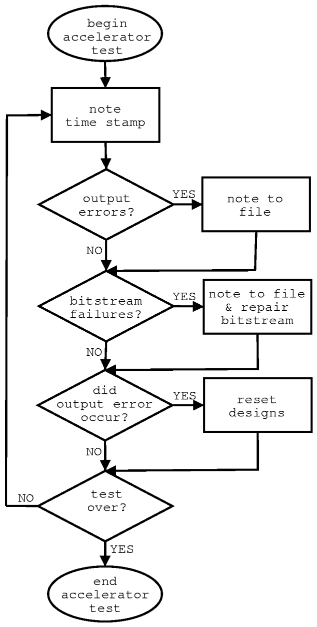


Figure 8.6: Accelerator Testing Flow Diagram [22].

be exchanged if necessary, rather than trading out an entire board. The remainder of the test equipment, the PC and other portions of the SLAAC1-V, were protected from the beam using .75" aluminum shielding.

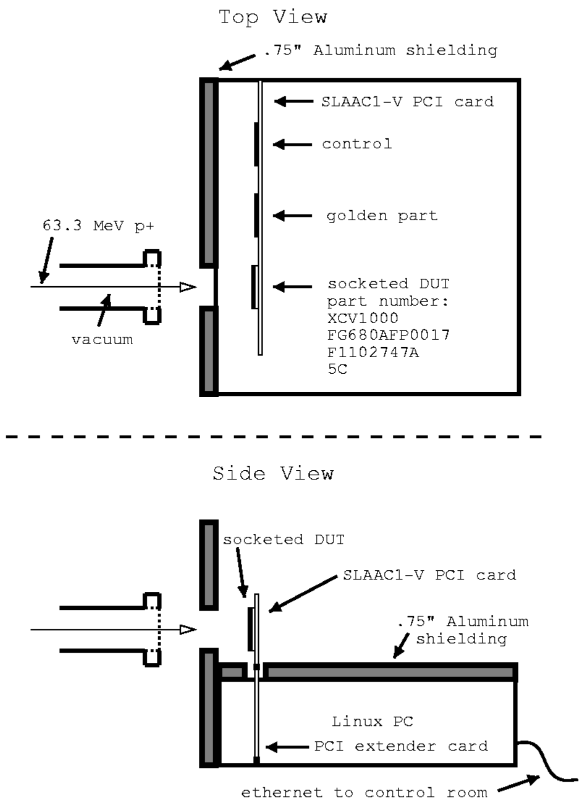


Figure 8.7: Accelerator test setup using the SLAAC1-V Proton Radiation Test Fixture [22].

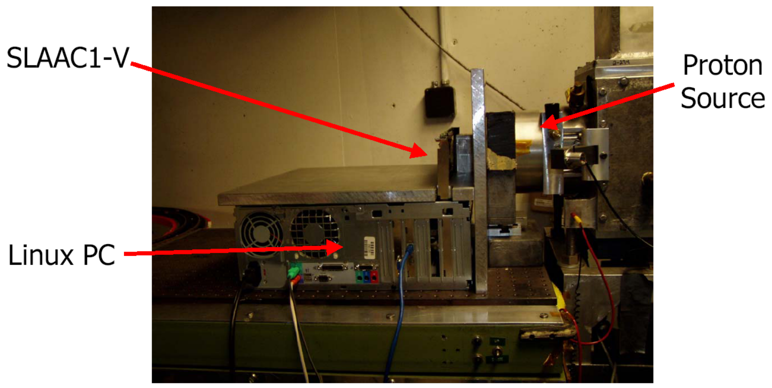


Figure 8.8: The SLAAC1-V Proton Radiation Test Fixture in action.

Chapter 9

Summary of the Design Flow

9.1 Introduction

The intention of this chapter is to provide the designer with a clear understanding of the design flow process used when developing space-based SRAM FPGA systems, how the various sections of this guide tie into that design flow, and what trade-offs are likely to be encountered during system implementation. The most important of these trade offs is the amount of reliability provided by a system versus the amount of resources utilized to achieve those requirements. This central theme is summed up in Figure 9.1. As the reliability of the system increases the amount of resources consumed to provide that reliability does as well. These resources can be the area used on the device, the time required for implementation, or even the speed at which the system functions. At some point in the design trade space, the amount of reliability achieved no longer justifies the amount of resources used to achieve it. It is important that the designer keeps this in mind when going through the design flow.

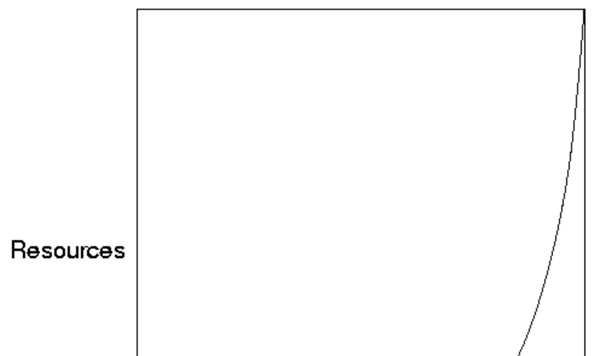


Figure 9.1: The amount of resources required to implement a certain level of reliability.

The design flow is broken up into three sections: environment and device characterization in which the radiation characteristics of the device to be used and the environment in which the device and its platform will be immersed are characterized; detection/mitigation/repair trade off analysis in which methods of detection, mitigation, and repair are compared and evaluated; and the actual implementation of the system, which employs the aforementioned mitigation tools and methodologies.

9.2 Steps in the Design Flow

9.2.1 Environment and Device Characterization

Does the device meet Total Ionizing Dose (TID), Dose Rate, and Single-Event Latch-up requirements?

The first tests that should be performed on any device are basic survivability tests. The device should be evaluated against the TID (see section 3.1) and the maximum dose rate (see chapter 3) that is likely to be encountered in a worse case scenario. Further, it should be tested for its immunity to single-event latch-up. If the device does not meet any of these requirements, it probably should not be used. There are ways of dealing with not meeting these requirements inherently, but they often require system-level mitigation. Many system-level designers prefer not to have to deal with these issues.

Do you have accurate static SEU and SEFI cross sections for the device?

Often manufacturers of FPGAs do not provide static SEU and SEFI cross sections for their device (see chapter 5). In such cases accelerator testing (see section 8.5) may be necessary to gather data on these cross sections if they cannot be found in relevant literature. Another consideration that needs to be taken into account is the "MBU skew" which occurs in new, denser FPGA devices and may effect the static cross section measurements obtained if not accounted for properly (see subsection 5.1.1). Fortunately, this is not a concern for the Virtex-I line of devices as their MBU cross section and, thus, skew is too small to be noticeable.

The static SEU cross section curve is then fitted using the Weibull formula or an equivalent cross-section model. The parameters gathered from this fitting are used in programs like SpaceRad (CREME) and CREME 96 to calculate on-orbit SEU rate estimations. As an example, parameters for a Weibull fit curve of the static SEU cross section per bit for the XQVR300 are provided in chapter 6. This provides a rough estimate of the static SEU cross section per bit in the Virtex-I line of devices and can be scaled to different devices from that same generation of FPGAs.

Perform On-Orbit SEU Rate Estimations using SpaceRad (CREME) or CREME 96.

Use the older CREME model or the more up-to-date CREME 96 model to estimate the SEU rates that are expected to be encountered in the orbital environment on which the device/platform will be deployed. While the CREME 96 model is more accurate, the web interface it uses might be a bit difficult to learn (see Appendix B). Using a program suite such as SpaceRad, which is based on the older CREME model and yet has a much easier user interface, can help reduce the learning curve associated with these tools. It also should be noted that some contracts require that the older CREME model be used for on-orbit SEU estimation. In these cases, a tool such as SpaceRad might prove to be convenient. As an example, Virtex 1000 HUP device parameters have been provided in Table B.1 for use with the Direct Ionization-Induced SEE Rate Calculation (HUP) module in the CREME96 suite of tools. Combined with the Weibull parameters provided, this information should allow a designer to make a complete set of on-orbit SEU rate estimations for a Virtex-I device. For Virtex-II and Virtex-4 parameters, we suggest using the XRTC static radiation test reports and/or the Xilinx data sheets for their radiation tolerant parts.

Does the device's SEFI cross section meet on-orbit environment requirements?

The SEFI cross-section of is the smallest possible cross-section that a device can provide in the best case scenario. If the SEFI cross section does not meet the requirements for orbital environment to be used then the device will probably not be adequate for use and a new device probably needs to be considered.

9.2.2 Detection/Mitigation/Repair Trade Off Analysis

It is important to remember while reading this section that designs often use a hybrid of detection, mitigation, and repair schemes depending on the needs of particular parts of designs and, often, they are heavily intertwined and their operation overlaps in various areas.

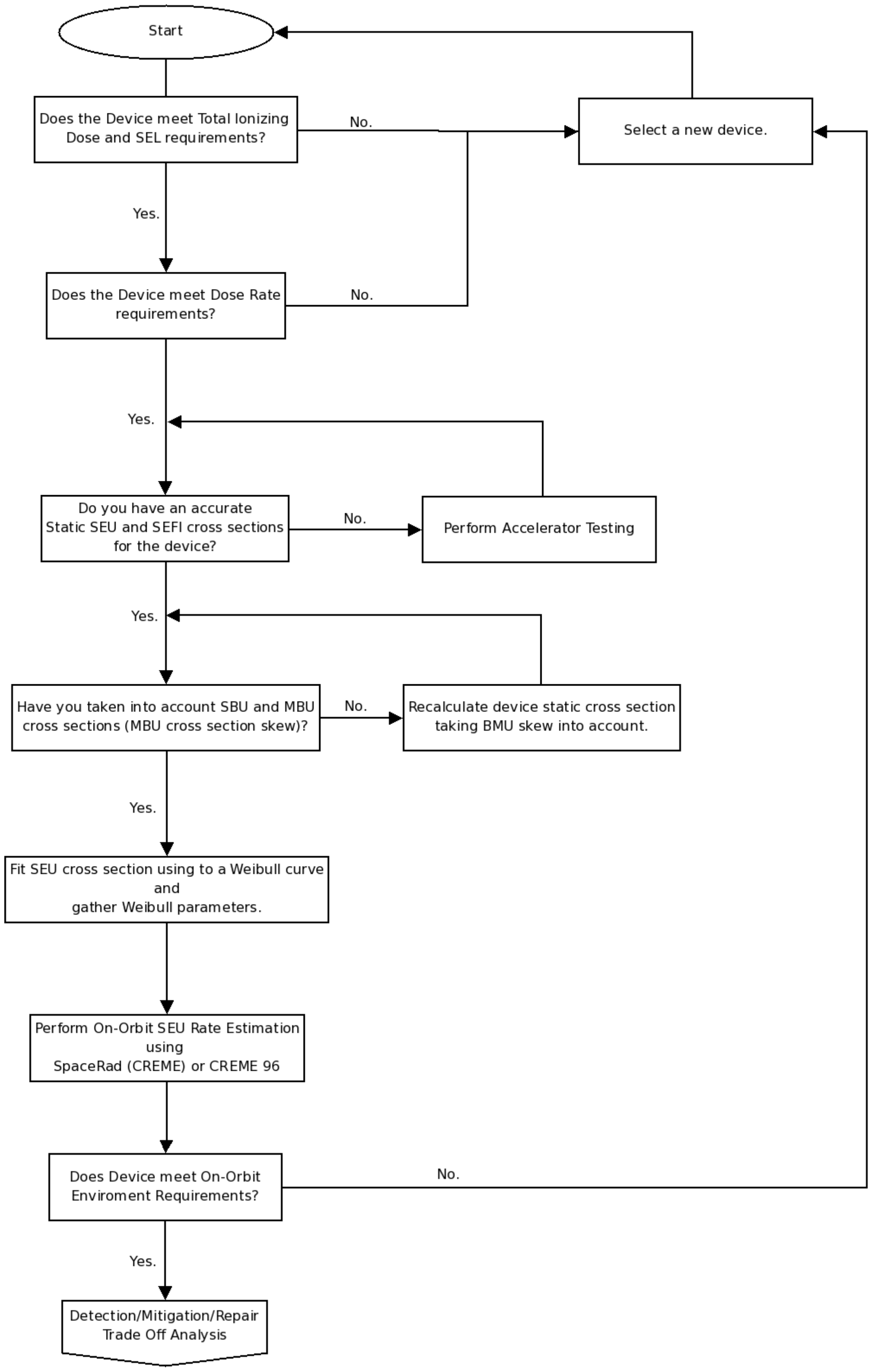

Start
Does the Device meet Total Ionizing Dose and SEL requirements?
No.
Select a new device.
Yes.
Does the Device meet Dose Rate requirements?
No.
Yes.
Do you have an accurate Static SEU and SEFI cross sections for the device?
No.
Perform Accelerator Testing
Yes.
Have you taken into account SBU and MBU cross sections (MBU cross section skew)?
No.
Recalculate device static cross section taking BMU skew into account.
Yes.
Fit SEU cross section using to a Weibull curve and gather Weibull parameters.
Perform On-Orbit SEU Rate Estimation using SpaceRad (CREME) or CREME 96
Does Device meet On-Orbit Enviroment Requirements?
No.
Yes.
Detection/Mitigation/Repair Trade Off Analysis

Figure 9.2: Environment and Device Characterization flow chart.

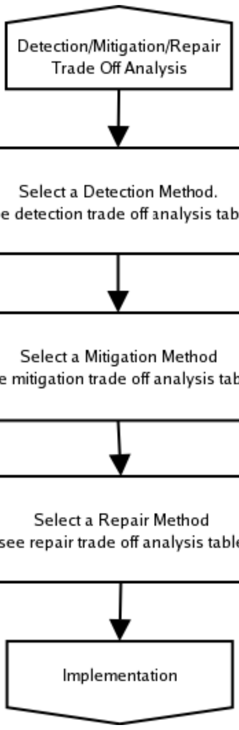

Detection/Mitigation/Repair Trade Off Analysis
Select a Detection Method. (see detection trade off analysis table)
Select a Mitigation Method (see mitigation trade off analysis table)
Select a Repair Method (see repair trade off analysis table)
Implementation

Figure 9.3: Detection/Mitigation/Repair flow chart.*

*Note that any time in the design flow the designer may be required to return to these steps to select a different method of detection, mitigation, or repair due to factors such as time constraints, space requirements, etc.

Select a Detection Method.

The trade-offs of possible detection methods are compared in Table 9.1. Some critical points to consider when deciding on a detection method is how much data needs to gathered on SEUs that occur in the system, whether or not RAM resources will be present in the system's configuration data, and what amount of overhead the detection method will require. Furthermore, some detection methods need lull points in a system's operation and require that test coverage be provided by test vectors in a timely manner. Often a system employs a hybrid of detection methods and no one detection method maybe the best solution. A more detailed overview of various detection methods can be found in section 7.1.

Select a Mitigation Method.

When determining which mitigation method(s) to use it is suggested that a detailed risk analysis be performed taking into account the consequences of errors in different parts of the design. This will help the designer apply appropriate amounts of mitigation to the various areas of a design in a meaningful way. Such a risk assessment will ensure that critical parts of the design have the proper mitigation in place while conserving resources by not wasting mitigation on non-critical areas.

Various methods of mitigation and their trade-offs are evaluated in Table 9.2. Nowhere else in a design is the trade off between resource utilization and reliability more apparent then the selection of a mitigation system. Some considerations when choosing a mitigation method include whether scrubbing is to be used, how much space is available for the mitigation method, and how much time is required to implement the mitigation method. If space is at a premium it might be wise to consider a mitigation method such as PTMR/STMR or BL-TMR, which apply TMR to only the most important components in a design. If implementation time is more important than space considerations, then using a mitigation method that employs an automated tool might be more important. Finally, continuous operation even during a procedure such as scrubbing would require a mitigation method such as XTMR or BL-TMR which re-synchronizes

Detection Method	Can handle RAM resources in configuration data? (no GLUTMASK)	Needs lull points in system operation?	Requires adequate test coverage through test vectors?	Can provide detailed data on upsets?	Relative resource usage?	Level of Comp.
Readback and Comparison	-	-	-	✓	Medium	Bitlevel Bitstream
Golden Output Method	✓	✓	✓	possible	Low	Bitlevel Outputs
Concurrent Error Detection	✓	-	-	possible	High	Bitlevel Outputs
CRC Frame Checks	-	-	-	-	Low	Checksum Compares
Internal ECC/Checksum (Virtex 4/5)	✓	-	-	-	Low	Checksum Error Codes

Table 9.1: Detection Trade-Off Analysis.

state logic after the scrubbing process. It should be noted that regardless of what method is employed not triplicating input global signals such as clocks and resets can introduce single points of failure seriously compromising the SEU robustness of a design.

Select a Repair Method.

Finally, a repair method must be chosen. Table 9.3 provides the trade-offs encountered between two commonly used repair methods. Some considerations that are important when choosing a repair method include whether or not detailed data needs to be gathered on the upsets that are occurring in the system, how much overhead is required for repair operation, and whether or not the repair method can handle RAM resources in the configuration data of the device without causing interface contention. It is important to note that, without a continuous repair process to clean up errors caused by upsets, mitigation will eventually fail. When considering scrubbing and the scrub rate it will employ, it is critical that the scrub rate be greater then the upset rate so that no more then one redundant domain is compromised between scrubs.

9.2.3 Implementation

It should be noted that this flow does not show all possible paths. For instance, iteration maybe necessary at several stages to meet design specifications. The dashed lines show a few possible SEU mitigation tools that can be added to the flow.

VHDL/Verilog/Simulation/Synthesis Cycle

Here the system is developed and refined via a standard development cycle of coding, simulation, and synthesis.

BL-TMR Applied

Once the EDIF for a design has been created after synthesis, the BL-TMR Tool can be used to apply BL-TMR if this was one of the SEU mitigation methods selected. One thing not shown in the Figure 9.4 is the fact that the BL-TMR Tool can produce reports for later use by the RadDRC tool for TMR-aware half-latch removal.

XTMR Applied

The TMRTool is used to apply XTMR after "ngdbuild" has translated the design to the Xilinx internal ".ngd" format. XTMR can provide half-latch mitigation in addition to applying Xilinx's style of TMR to an FPGA design.

RadDRC Half-Latch Removal

Once a placed and routed NCD has been created, the RadDRC tool can be used in conjunction with the Xilinx "xdl" and "par" tools to remove half-latches from Xilinx Virtex-I/-II designs. TMR-aware half latch removal is possible after the technology mapping phase due to the domain reports generated by the BL-TMR tool. Without TMR mapping information, RadDRC can still remove half-latches, but may introduce single points of failure since it will not know how to allocate constant sources among TMR'ed modules. At a minimum, running RadDRC is a good check to ensure that what ever method was used for half-latch removal earlier in the design flow was successful.

Perform SEU emulator testing to determine the dynamic cross section.

At this stage, an SEU emulator is used to determine the dynamic cross section of the design that has been implemented. If the the dynamic cross section does not meet on-orbit SEU requirements by a small amount, the design is re-coded, simulated, and synthesized to reduce the dynamic cross section. If the dynamic cross

Mitigation Method	Triplicates modules?	Triplicates voters?	Re-synchronizes state logic feedback after SEU scrubbing?	Conserves resources by mitigating only the most sensitive structures?	Substantially reduces a designs persistent cross section?	Automated?
Simple TMR	✓	-	-	-	-	-
XTMR	✓	✓	✓	Manual [*] (not automatic)	-	✓[†]
PTMR/STMR	✓	?	?	✓	Needs analysis.	?
BL-TMR	✓	✓	✓	✓	✓	✓[‡]

Table 9.2: Mitigation Trade-Off Analysis.

[*]Must be performed by hand.
[†]XTMR is automated through the TMRTool.
[‡]BL-TMR is automated through the BL-TMR Tool.

Detection Method	Can handle RAM resources in configuration data?	Can provide detailed data on upsets?	Relative resource usage?	Can detect configuration interface SEFIs?
Read and Repair	-	✓	Medium	✓
Scrubbing	✓[*]	-	Low	✓[†]

Table 9.3: Repair Trade-Off Analysis.

[*]Must skip RAM resources during scrubbing process. Scrubbing can break RAM by writing in old values, destroying the correct current values.

[†]Perform periodic readbacks to ensure the configuration interface is still functional.

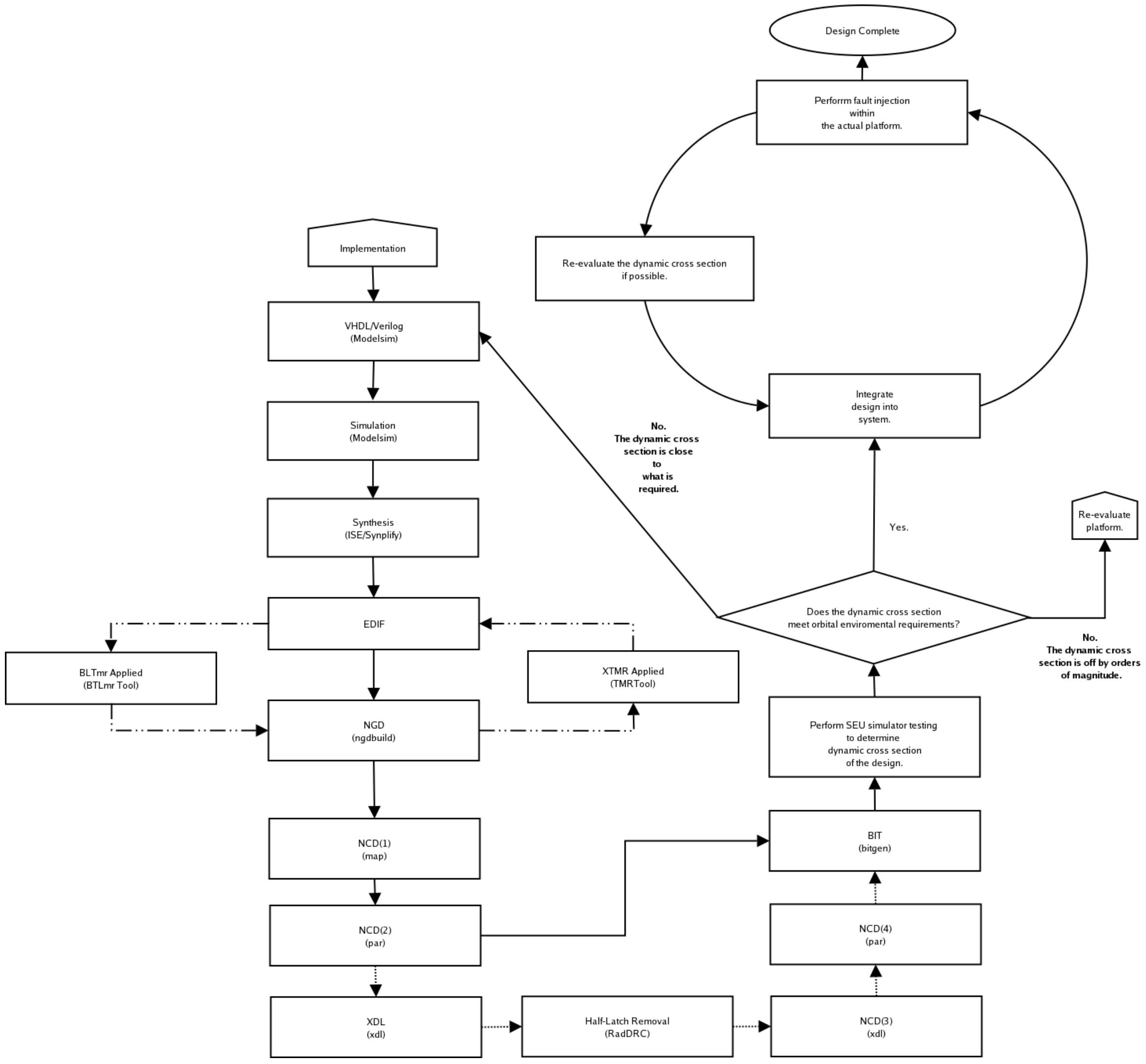

Design Complete
Perform fault injection within the actual platform.
Re-evaluate the dynamic cross section if possible.
Implementation
VHDL/Verilog (Modelsim)
Integrate design into system.
No. The dynamic cross section is close to what is required.
Simulation (Modelsim)
Synthesis (ISE/Synplify)
Yes.
Re-evaluate platform.
EDIF
Does the dynamic cross section meet orbital enviromental requirements?
No. The dynamic cross section is off by orders of magnitude.
BLTmr Applied (BTLmr Tool)
XTMR Applied (TMRTool)
NGD (ngdbuild)
Perform SEU simulator testing to determine dynamic cross section of the design.
NCD(1) (map)
BIT (bitgen)
NCD(2) (par)
NCD(4) (par)
XDL (xdl)
Half-Latch Removal (RadDRC)
NCD(3) (xdl)

Figure 9.4: Implementation flow chart.

section does not meet on-orbit SEU requirements by orders of magnitude then a new device may need to be selected or a different mitigation or design architecture may need to be used.

Implementation/Fault Injection Cycle.

At this point, the FPGA user design is integrated into the physical platform. We recommend that the integrated design undergo fault injection within the actual platform to validate the dynamic cross-section. This phase of development is extremely critical as it effectively gives the developers a chance to test their design in the actual platform and verify its operation, including testing for unanticipated errors through fault injection.

Bibliography

[1] Cosmic ray effects on micro-electronics (1996 revision). on web https://creme96.nrl.navy.mil/.

[2] http://www.xilinx.com/labs/projects/jbits/.

[3] Greg Allen, Gary Swift, and Carl Carmichael. Virtex-4VQ static SEU characterization summary. Technical Report 1, Xilinx Radiation Test Consortium, 2008.

[4] H. N. Becker, T. F. Miyahira, and A. H. Johnston. Latent damage in CMOS devices from single-event latchup. *IEEE Transactions on Nuclear Science*, 49(6):3009 – 3015, 2002.

[5] Michael Caffrey, Paul Graham, Michael Wirthlin, Eric Johnson, and Nathan Rollins. Single-event upsets in SRAM FPGAs. In *Proceedings of the 5th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2002.

[6] Carl Carmichael, Brendan Bridgford, Gary Swift, and Matt Napier. A triple module redundancy scheme for SEU mitigation of static latch-based FPGAs ("Birds-of-a-Feather"). In *Proceedings of the 7th Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2004. Presentation.

[7] Steven L Clark, Keith Avery, and Robert Parker. TID and SEE testing results of the Altera Cyclone Field Programmable Gate Array. *IEEE Radiation Effects Data Workshop*, pages 88 – 90, 2004.

[8] Fernanda Gusmo de Lima Kastensmidt, Gustavo Neuberger, Renato Fernandes Hentschke, Luigi Carro, and Ricardo Reis. Designing fault-tolerant techniques for SRAM-based FPGAs. In *1st ACM Conference on Computing Frontiers*, pages 419–432, November-December 2004.

[9] FG F. de Lima Kastensmidt. Designing fault-tolerant techniques for SRAM-based FPGAs. *IEEE design and test of computers*, 21(6):552 – 562, 2004.

[10] Y. Deval, H. Lapuyade, P. Fouillat, H. Barnaby, F. Darracq, R. Briand, D. Lewis, and R. D. Schrimpf. Evaluation of a design methodology dedicated to dose-rate-hardened linear integrated circuits. *IEEE Transactions on Nuclear Science*, 49(3):1468 – 1473, June 2002.

[11] K. Endo. The radiation environment. Image. Nikkei Science, Inc. of Japan.

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

[13] 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 *Proceedings of the 2nd Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 1999.

[14] 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 *Proceedings of the 3rd Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, page P30, 2000.

[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 *Proceedings of the 3th Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, 2000. Presentation.

[16] Maya Gokhale, Paul Graham, Michael Wirthlin, D. Eric Johnson, and Nathaniel Rollins. Dynamic reconfiguration for management of radiation-induced faults in FPGAs. *International Journal Of Embedded Systems*, 2005. To appear in 2005.

[17] Paul Graham, Michael Caffrey, Michael Wirthlin, D. Eric Johnson, and Nathan Rollins. Reconfigurable computing in space: From current technology to reconfigurable systems-on-a-chip. In *Proceedings of the 2003 IEEE Aerospace Conference*, pages T07_0603.1–12, Big Sky, MT, March 2003. IEEE.

[18] Paul Graham, Michael Caffrey, Michael Wirthlin, D. Eric Johnson, and Nathaniel Rollins. SEU mitigation for half-latches in Xilinx Virtex FPGAs. *IEEE Transactions on Nuclear Science*, 50(6):2139–2146, December 2003.

[19] Paul S. Graham. *Logical Hardware Debuggers for FPGA-Based Systems*. PhD thesis, Brigham Young University, Provo, UT, December 2001.

[20] Paul S. Graham. A brief description of the Virtex-II SEU Emulator hardware and its construction. Technical Report LA-UR-05-1343, Los Alamos National Laboratory, 2005.

[21] Andrew Holmes-Siedle and Len Adams. *Handbook of Radiation Effects Second Edition*. Oxford University Press, 2002.

[22] Eric Johnson, Michael Caffrey, Paul Graham, Nathan Rollins, and Michael Wirthlin. Accelerator validation of an FPGA SEU simulator. *IEEE Transactions on Nuclear Science*, 50(6):2147–2157, December 2003.

[23] Eric Johnson, Michael J. Wirthlin, and Michael Caffrey. Single-event upset simulation on an FPGA. In Toomas P. Plaks and Peter M. Athanas, editors, *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pages 68–73. CSREA Press, June 2002.

[24] J. Johnson, W. Howes, M. Wirthlin, D. L. McMurtrey, M. Caffrey, P. Graham, and K. Morgan. Using duplication with compare for on-line error detection in fpga-based designs. In *2008 IEEE Aerospace Conference, 1-8 March 2008, Big Sky, MT, USA*, volume 2008. Piscataway, NJ, USA : IEEE, 2008, 2008.

[25] R. Katz, J. J. Wang, R. Koga, K. A. LaBel, J. McCollum, R. Brown, R. A. Reed, B. Cronquist, S. Crain, T. Scott, W. Paolini, and B. Sin. Current radiation issues for programmable elements and devices. *IEEE Transactions on Nuclear Science*, 45(6):2600–2610, December 1998.

[26] Kenneth LaBel, Rich Katz, and Igor Kleyner. An insider's guide to designing spacecraft systems and instruments for operation in the natural space radiation environment. In *Proceedings of the 5th Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, Sep. 9 2002.

[27] Naval Research Laboratory. Cosmic ray effects on micro-electronics rev. 96, December 2003. https://creme96.nrl.navy.mil/cm/AP8Accuracy.htm.

[28] Naval Research Laboratory. Cosmic ray effects on micro-electronics rev. 96, December 2003. https://creme96.nrl.navy.mil/cm/trplimits.htm.

[29] P. Marshall, K. Label, M. Friendlich, C. Seidleck, M. Berg, A. Phan, J. Schwank, M. Shaneyfelt, P. Dodd, K. Rodbell, D. Heidel, M. Xapsos, A. KleinOsowski, and M. Hakey. Proton, alpha particle and heavy ion SEU test results in 65 nm SRAMs fabricated in IBM's SOI process. In *Single-Event Effects Symposium*, 2008.

[30] 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*, 52(6):2438 – 45, 2005.

[31] Keith Morgan, Michael Caffrey, Paul Graham, Eric Johnson, Brian Pratt, and Michael Wirthlin. SEU-induced persistent error propagation in FPGAS, 2005. Presentation given at the Nuclear and Space Radiation Effects Conference (NSREC) 2005.

[32] Keith Morgan, Michael Caffrey, Paul Graham, Eric Johnson, Brian Pratt, and Michael Wirthlin. Seu-induced persistent error propagation in FPGAS. *IEEE Transactions on Nuclear Science*, 52(6), December 2005.

[33] E. L. Petersen. Predictions and observations of SEU rates in space. *IEEE Transactions on Nuclear Science*, 44(6), December 1997.

[34] Heather Quinn, Paul Graham, James Krone, and Micheal Caffery. Radiation-induced multi-bit upsets in SRAM-based FPGAs. In *IEEE Transactions on Nuclear Science*, volume 52(6), December 2005.

[35] Heather Quinn, Paul Graham, Jim Krone, Michael Caffrey, and Sana Rezgui. Radiation-induced multi-bit upsets in SRAM-based FPGAs. *IEEE Transactions on Nuclear Science*, 52(6):2455 – 2461, December 2005.

[36] Heather Quinn, Keith Morgan, Paul Graham, Jim Krone, and Michael 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*, pages 177 – 184, July 2007.

[37] Heather Quinn, Keith Morgan, Paul Graham, Jim Krone, Michael Caffrey, and Kevin Lundgreen. Domain crossing errors: Limitations on single device triple-modular redundancy circuits in Xilinx FPGAs. *IEEE Transactions on Nuclear Science*, 54(6):2037 – 43, 2007.

[38] NASA/GSFC Radiation Physics Office. Space radiation environment, December 1998. http://radhome.gsfc.nasa.gov/radhome/environ.htm.

[39] Nathan Rollins, Michael Wirthlin, Michael Caffrey, and Paul Graham. Reliability of programmable input/output pins in the presence of configuration upsets. In *Proceedings of the 5th Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2002.

[40] Praveen Kumar Samudrala, Jeremy Ramos, and Srinivas Katkoori. Selective triple modular redundancy for SEU mitigation in FPGAs. In *Proceedings of the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, page C1, Washington, D.C., 2003. NASA Office of Logic Design, AIAA.

[41] Gary M. Swift. Virtex-II static SEU characterization. Technical Report 1, Xilinx Radiation Test Consortium, 2004.

[42] J.J. Wang, R. Chan, G. Kuganesan, B. Cronquist, and J. McCollum. Total ionizing dose effect on programmable input configurations. In *Proceedings of the 8th Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2005. Presentation.

[43] S.C. S. Witczak. Dose-rate sensitivity of modern nmosfets. *IEEE transactions on nuclear science*, 52(6):2602 – 2608, 2005.

[44] Xilinx, Inc. *Correcting Single-Event Upsets through Virtex Partial Configuration, Xilinx Application Notes 216, v1.0*, June 2000.

[45] Xilinx, Inc. *Virtex Architecture Guide*, September 2000. Provided as part of the JBits 2.8 SDK.

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

[47] Xilinx, Inc. *Triple Module Redundancy Design Techniques for Virtex FPGAs, Xilinx Application Notes 197, v1.0*, November 1, 2001.

[48] Xilinx, Inc. *Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan, Xilinx Application Notes 139, v1.6*, September 2003.

[49] Xilinx, Inc. *TMRTool User Guide (TMRTool Software V6.2.03i), Xilinx User's Guide 156, v1.0*, September 2004.

[50] Xilinx, Inc. *Virtex FPGA Series Configuration and Readback, Xilinx Application Notes 138, v2.8*, March 2005.

Appendix A

Case Study: The Cibola Flight Experiment

A.1 Introduction

The Cibola Flight Experiment (CFE) is a DOE experiment developed by LANL to validate the use of commercial, reconfigurable FPGA technology in low earth orbit (LEO) platforms, to develop on-board data analysis/reduction capabilities for large amounts of sensor data, to test mitigation and repair techniques, and to demonstrate the ability to adapt to new missions using RCC capabilities. The CFE will survey portions of the VHF and UHF radio spectra (20 - 500 MHz) with the objective of detecting and measuring impulse events that occur in a complex spectrum background. The CFE Satellite (CFESat) was launched in March 2007.

A.2 The CFE Payload

The design of CFE was approached with the intention to develop relatively small and inexpensive collection units that could be used as payloads in microsat platforms. The current payload design (see Figure 8.1) consists of three reconfigurable computing (RCC) modules composed of 3 Virtex XQVR1000 each. These modules are networked together to provide processing for data collected from the payload radios via two high-speed analog-to-digital converters (ADC). A BAE Rad6000 processor provides tasking for the sub-system and a radiation-hardened Actel FPGA acts as a configuration manager/interface for the RCC modules. Finally, the radio-antenna chains consist of four antennas connected to two 50-MHz RF channels gang-tuned to an intermediate frequency (IF) on the range of 55-95 MHz. It should be noted that the two analog-to-digital converters which sample the channels operate at 100 MHz and have 12-bit sampling resolution.

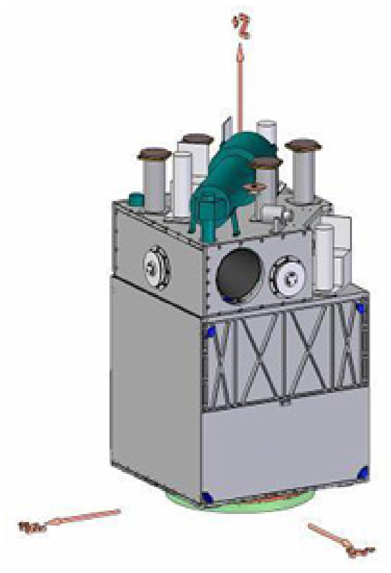


Figure A.1: Drawing of CFESat bus developed by Surrey Satellite Technology Ltd. (SSTL).

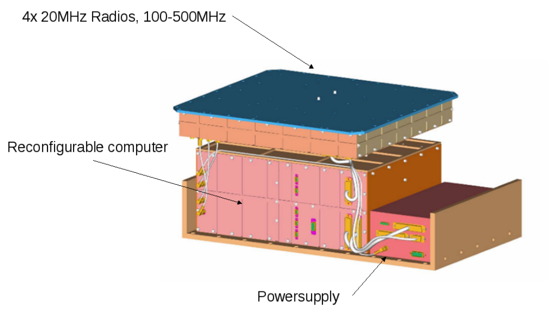


Figure A.2: Drawing of the CFE payload developed by Los Alamos National Laboratory (LANL).

A.3 Mitigation Techniques Employed by CFE

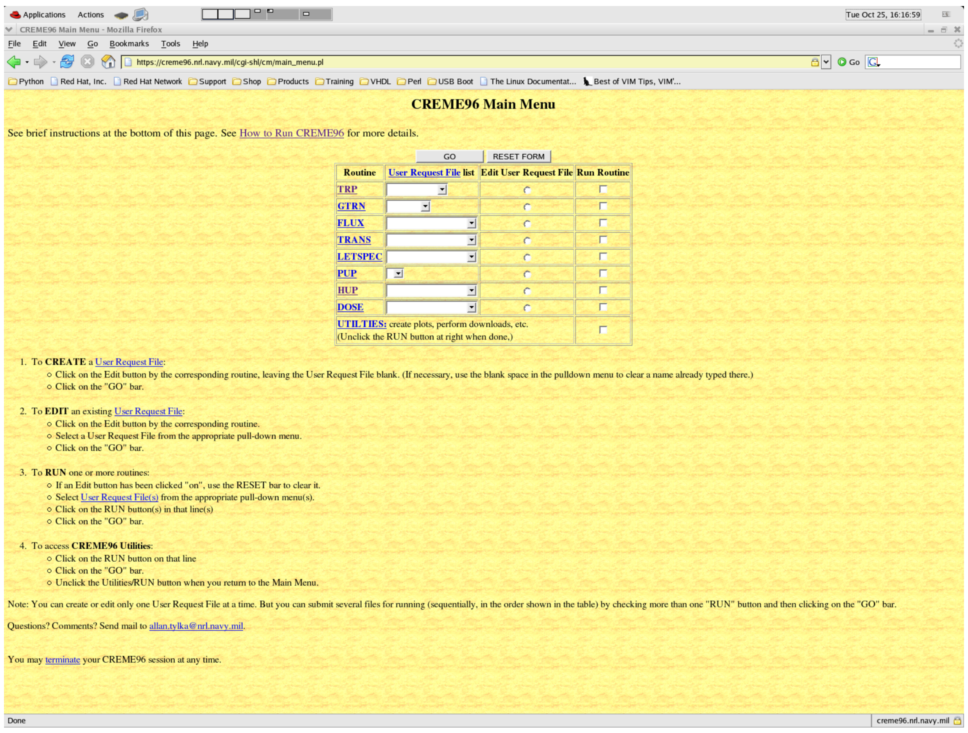
By default all of the RCC modules are checked via a CRC readback process (see subsection 7.1.2) to identify corrupted frames within the configuration data of the 9 XQVR1000 devices present in the CFE payload. This has the dual purpose of collecting information on SEUs as they occur in the system as well as triggering a targeted on-demand scrubbing process (see subsection 7.3.1) to repair the corruption.

In addition to this, CFE has been specifically designed to test and validate various TMR mitigation schemes (see section 7.2) within the RCC modules through reconfiguration. Currently, with no TMR mitigation in the XQVR100 FPGAs, fault-injection (see section 8.3) has revealed that about $\frac{1}{2000}$ SEUs cause a bus error on accessing a software task, leading to a payload reset due to the way the CFE spacecraft bus has been designed. It is believed the number of resets due to SEUs can be substantially improved by the use of a design incorporating partial TMR. This hypothesis is carefully being tested.

Appendix B

CREME96: Example and Parameters for the XQVR1000

The CREME96 suite of tools can be accessed at https://creme96.nrl.navy.mil/. The interface to CREME96 is web based and a fairly straight forward registration process is required for access. The CREME96 software package consists of 8 modules plus miscellaneous utilities to aid in plots, downloads, etc.



CREME96 Main Menu

See brief instructions at the bottom of this page. See How to Run CREME96 for more details.

Routine	User Request File list	Edit User Request File	Run Routine
TRP			
GTRN			
FLUX			
TRANS			
LETSPEC			
PUP			
HUP			
DOSE			
UTILTIES: create plots, perform downloads, etc. (Unclick the RUN button at right when done,)			

1. To **CREATE** a User Request File:
   - Click on the Edit button by the corresponding routine, leaving the User Request File blank. (If necessary, use the blank space in the pulldown menu to clear a name already typed there.)
   - Click on the "GO" bar.
2. To **EDIT** an existing User Request File:
   - Click on the Edit button by the corresponding routine.
   - Select a User Request File from the appropriate pull-down menu.
   - Click on the "GO" bar.
3. To **RUN** one or more routines:
   - If an Edit button has been clicked "on", use the RESET bar to clear it.
   - Select User Request File(s) from the appropriate pull-down menu(s).
   - Click on the RUN button(s) in that line(s)
   - Click on the "GO" bar.
4. To access **CREME96 Utilities**:
   - Click on the RUN button on that line
   - Click on the "GO" bar.
   - Unclick the Utilities/RUN button when you return to the Main Menu.

Note: You can create or edit only one User Request File at a time. But you can submit several files for running (sequentially, in the order shown in the table) by checking more than one "RUN" button and then clicking on the "GO" bar.

Questions? Comments? Send mail to allan.tylka@nrl.navy.mil.

You may terminate your CREME96 session at any time.

Figure B.1: The CREME96 web interface.

General information needed to use the CREME96 package includes 6 orbital elements: apogee, perigee, orbital inclination, initial longitude of the ascending node, and the displacement of perigee from the ascending node. The designer will also need to specify the number of orbits to be simulated. Additionally, the range of elements (by atomic number) to be included in ion and spectrum calculations needs to be known along with the minimum energy of included nuclei to be considered in LET spectra calculations. For shielding simulations parameters including thickness and the distribution of the shielding will be needed (note: CREME96 requires that shielding thickness be specified in mils, cm, or g/cm^2 of aluminum).

For the Direct Ionization-Induced SEE Rate Calculation (HUP) module some device parameters will be needed such as the RPP Dimensions (https://creme96.nrl.navy.mil/cm/RPP.htm), Bits/Device, funnel length (https://creme96.nrl.navy.mil/cm/Funnel.htm), and SEE cross section parameters which require the Weibull curve parameters for onset, width, Weibull exponent value, and limiting cross section (please refer to chapter 6 for information on Weibull curve parameters).

Device	X	Y	Z	Funnel	Bits/Device	Weibull Parameters			
						L_0	W	s	σ_{sat}
XVQR1000 (no funnel)	0	0	1	0	5810024	1.2	30	2	8
XVQR1000 (2 um funnel)	0	0	1	2	5810024	1.2	30	2	8

Table B.1: CREME96 HUP Device Parameters for the XVQR 1000 device (with and without funneling).

A manual explaining the use of each of the modules and the parameters required can be found at https://creme96.nrl.navy.mil/cm/howtorun.htm. It has been noted that often it is easier to learn the general CREME package through software such as the SpaceRad Suite and then move to CREME96 simulations for more accurate results.

Appendix C

Xilinx Virtex-I Device Overview

The following section provides a brief overview of the highly versatile Xilinx Virtex-I family of devices. Those already familiar with the Xilinx Virtex line of devices or working on another platform may wish to skip over this section. This is not meant to be all encompassing but rather focused on the scope of this guide. More detailed Xilinx device information can be found in the Xilinx application notes “Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan” [48], “Virtex Series Configuration Architecture User Guide“ [46], and the “Virtex FPGA Series Configuration and Readback” [50].

C.1 Introduction

The Xilinx Virtex line of devices, introduced beginning in 1998, are SRAM-based, reconfigurable FPGAs containing 1,728 to 200,448 logic cells and using 220-*nm* (Virtex-I), 150-*nm* (Virtex-II), and 90-*nm* (Virtex-4) CMOS processes.

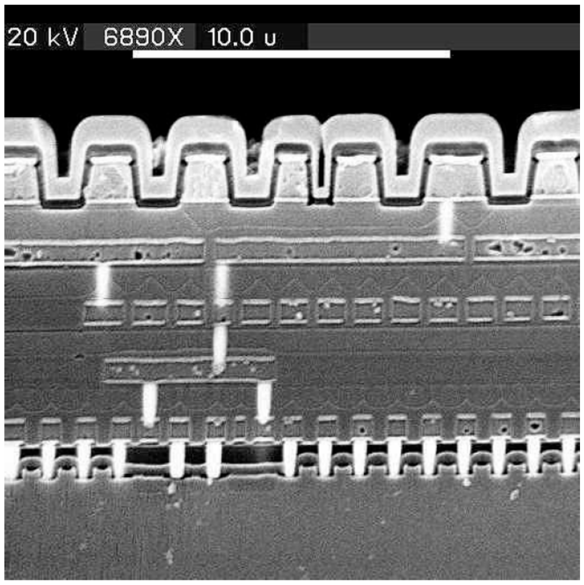


Figure C.1: Image of the Virtex-I 5-layer-metal 220-*nm* process [15].

While all of the Virtex families have much in common they also differ in various, important ways. The rest of this Appendix will concentrate specifically on the Virtex-I line of devices for brevity.

C.2 Basic Logic Structure

Each Virtex device contains configurable logic blocks (CLBs), input-output blocks (IOBs), block SelectRAMs, clock resources, programmable routing, and configuration circuitry. Configuration memory by far makes up the largest substantial portion of the logic within a Virtex FPGA device. As an example, the internal configuration memory of a Virtex V1000 device requires almost 6 million bits to describe the interconnect, logic, I/O pins, and operating modes of a user design [46].

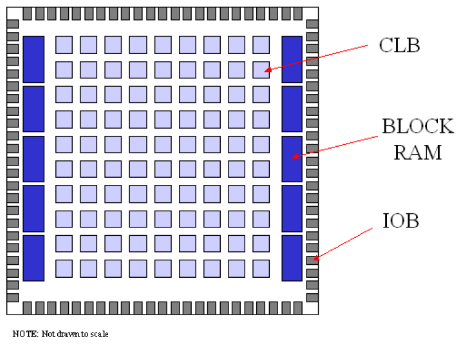


Figure C.2: Simplified structure of the Xilinx line of devices (not to scale) [45].

C.2.1 Configurable Logic Blocks (CLBs), Slices, and Logic Cells

Configuration Logic Blocks (CLBs) form the fundamental functional logic elements with in the Virtex line of devices. Each Virtex CLB can be further divided into Slices (see Figure C.3), which in turn contain Logic Cells (LC). There are two, vertical slices in each Virtex-I CLB and two Logic Cells per slice. Each LC includes a 4-input function generator, carry logic, and a storage element. In addition to slices and logic cells, each Virtex CLB contains a certain amount of distributed RAM, depending on the device.

C.2.2 Distributed RAM and Block Select RAM (BRAM)

The Xilinx Virtex line of devices each contain distributed RAM which is intended to provide fast, localized, small data buffers, FIFOs, or register files. In addition to distributed RAM the Xilinx Virtex family also provides fast, larger, discrete blocks of RAM known as Block Select RAM (BRAM). While distributed RAM allows a designer to construct shallow RAM constructs associated with individual CLBs, block SelectRAM allows for the creation of wider/deeper RAM structures. The Xilinx CoreGen program can be used to generate the VHDL/Verilog code for memory structures using the BRAM features or, more generally, they can be inferred through appropriate HDL descriptions. Please refer to "Using the Virtex Block SelectRAM+ Features" for more information on the various types of RAM and the tools used with them.

Unfortunately, if readback is performed at the same time as a write occurs to Virtex RAM components there is a possibility of write lockout occurring due to interface contention in the case of BRAM or even corruption in the case of SelectRAM. It should also be noted that output register corruption occurs when performing readback on BRAM components of the Virtex-I line of devices. Corruption occurs because the readback mechanism uses the same interface for retrieving state data from the BlockRAMs that FPGA designs use for normal operation. The readback process effectively overwrites the output registers of the Block SelectRAM leading to corruption. This condition is known as "Destructive State Sampling" and is a serious and undocumented problem. Please refer to "Logical Hardware Debuggers for FPGA-Based Systems" [19] for more information.

C.2.3 Input-Output Blocks (IOBs)

Input-Output Blocks (IOBs) are the standard Xilinx I/O cell. The IOBs are how the Virtex FPGA interfaces to its external pins and pads. Figure C.4 shows a simplified model of the IOB architecture [39]. Each IOB

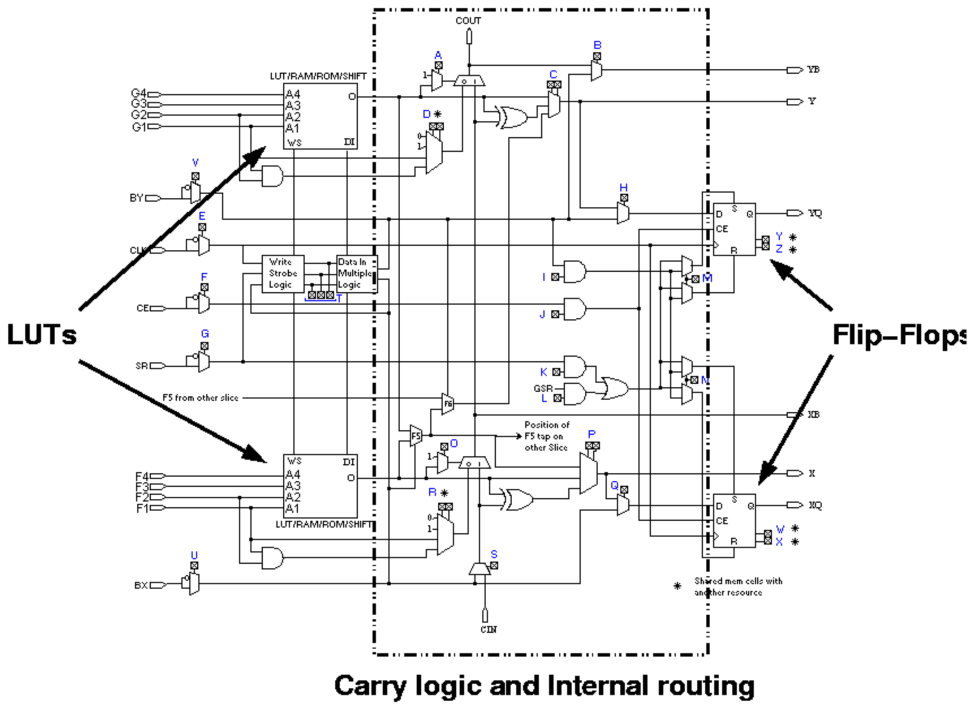


Figure C.3: Structure of a Xilinx Slice ($\frac{1}{2}$ a CLB) [45].

can be configured as an output, input, or a bi-directional I/O pin. In addition to this, each IOB flip-flop includes three flip-flops to provide registering capability at the input, output, and tri-state enable.

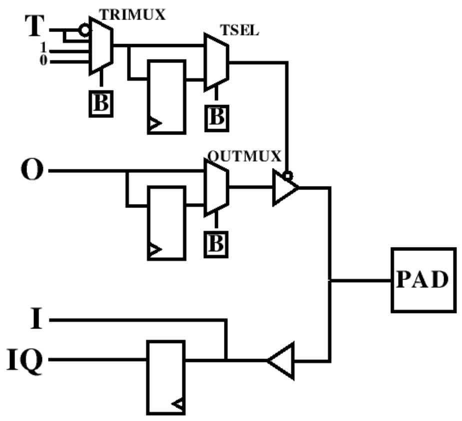


Figure C.4: Simplified structure of the Xilinx Programmable I/O Block [39].

It should be noted that Figure C.4 only provides a simplified schematic of the standard IOB. Much of the functionality such as the ability to change the I/O standard and output signal drive strength is left out for simplicity.

C.2.4 CLB Routing

Figure C.5 displays a simplified diagram of a typical CLB routing resources present in the Virtex line of devices. Each CLB possesses two lengths of wires for routing known as singles (in red) and hexes (in blue) respectively. Singles are intended for local networking and can connect to any directly adjacent CLB to the north, west, east, and south. Hexes are intended for non-local routing and use a hex routing switch to connect to another hex wire 6 CLBs away.

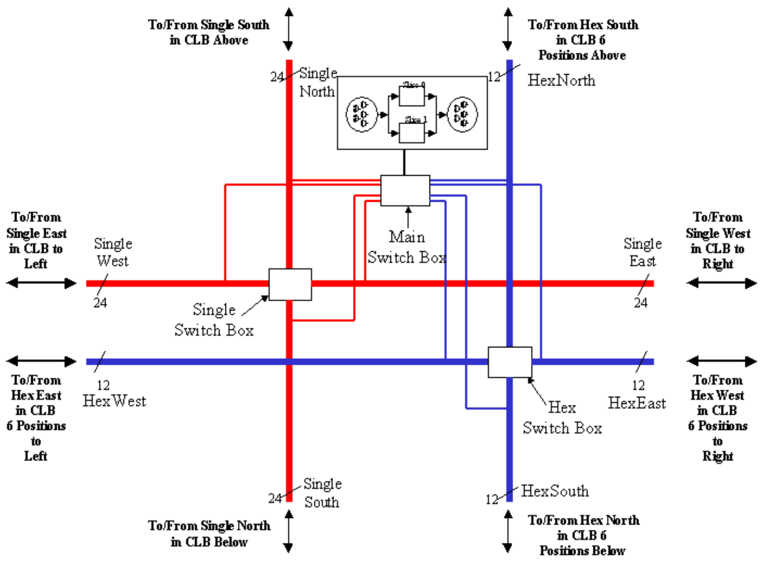

To/From Single South in CLB Above
To/From Hex South in CLB 6 Positions Above
24 Single North
12 HexNorth
To/From Single East in CLB to Left
Single West
24
Main Switch Box
Single East
24
To/From Single West in CLB to Right
Single Switch Box
To/From Hex East in CLB 6 Positions to Left
12 HexWest
Hex Switch Box
12 HexEast
To/From Hex West in CLB 6 Positions to Right
24 Single South
12 HexSouth
To/From Single North in CLB Below
To/From Hex North in CLB 6 Positions Below

Figure C.5: A simplified view of a CLB routing block [45].

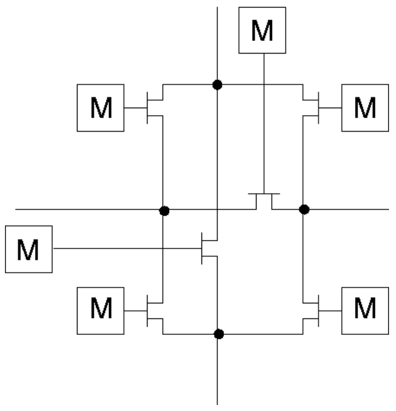


Figure C.6: A simplified view of the internal structure a routing switch possesses.

C.3 The Bitstream

The logic functions of the Virtex devices are configurable through the programming, or *configuration*, bitstream. The bitstream is a binary stream very similar to the machine language used by conventional computers containing a series of configuration commands and configuration data. The bitstream itself is covered in detail in the “Virtex Series Configuration Architecture User Guide”. [46]

C.4 Configuration/Readback

Configuration is the process of loading a design bitstream into the FPGA’s internal programming data memory. Readback is the process of reading that data from the device. Virtex devices can be configured through the SelectMAP” interface, master/slave serial interfaces, or the Boundary-Scan (JTAG) interface.[46]

C.4.1 SelectMAP Interface

The SelectMAP interface is the fastest method by which to program the Xilinx Virtex family of products. The SelectMAP interface operates at 8-bits per clock cycle and has a maximum effective clock rate of 50 MHz producing a data rate of 50 MB/s. This makes the SelectMAP interface the preferred method of (re)configuration. The SelectMAP interface's I/O pins are programmable and can be affected by SEUs and fault injection.

C.4.2 Joint Test Action Group (JTAG)/Boundary-Scan Interface

The Joint Test Action Group (JTAG)/Boundary-Scan Interface is also known as the industry wide IEEE 1149.1 boundary-scan standard. The JTAG interface functions at 1/8th the bitwidth of the SelectMAP interface (one bit per clock cycle) and functions at a maximum speed of 33.3 MHz, producing a maximum data rate of 4.16 MB/sec. This makes the JTAG interface much less desirable in terms of speed. However, in addition to being an industry standard, there are some applications where using the JTAG interface is necessary. For example, some of Xilinx software (such as ChipScope) requires the JTAG interface to access on chip structures that are not available to the SelectMAP interface. To its benefit the JTAG interface on the Xilinx devices has dedicated pins and has a smaller SEU cross-section than the SelectMAP interface.

C.4.3 Master/Serial Slave Modes

Master mode allows the Virtex device to be configured from a serial PROM whereas Slave Mode allows it to be programmed from other logic devices, such as microprocessors, or in a daisy-chain fashion with another Virtex device [50]. These modes are not commonly used for space systems mostly due to the fact that programming data readback is not supported.

C.5 Methods of Device Reconfiguration

C.5.1 The Frame/Smallest Form of Granularity

The configuration data frame (also known as the *configuration frame*) is the smallest amount of data that can be read or written in one operation during partial reconfiguration. There are 48 frames of configuration data to each column of Configuration Logic Blocks (CLBs) with the associated I/O blocks above and below for Virtex-I.

C.5.2 Full, Off-line Reconfiguration

Full, off-line reconfiguration involves reconfiguring all the device's resources and usually involves a complete reset of the device. Total reconfiguration is not the most efficient method of reconfiguration but is the most complete and reliable.

C.5.3 On-line Reconfiguration

On-line reconfiguration allows a device to be be dynamically reprogrammed while still running by providing selective access to the devices Configuration Memory. The speed of this dynamic reconfiguration is directly proportional to the number of configuration memory locations which need to be changed in order to implement the desired design modification. The smaller the amount of device resources that need to be reconfigured, the quicker the reconfiguration can take place leading to great speed improvements over the conventional method of full, off-line reconfiguration.