LA-UR- 08-6312

Approved for public release; distribution is unlimited.

Title:	A Primer for the Talk "Outside of Normal Operating Conditions: Using Commercial Hardware in Space Computing Platforms for Ubiquitous Sensing"
Author(s):	Heather Quinn
Intended for:	The Grace Hopper Celebration of Women in Computing Conference Keystone, CO 10/1-4/2008

Los Alamos
NATIONAL LABORATORY
EST. 1943

Form 836 (7/06)

A Primer for the Talk "Outside of Normal Operating Conditions: Using Commercial Hardware in Space Computing Platforms for Ubiquitous Sensing"

Heather Quinn
ISR, Space Data Systems
Los Alamos National Laboratory
Los Alamos, NM 87544
hquinn@lanl.gov

Abstract— **Over the past decade field-programmable gate arrays (FPGAs) have been useful in speeding up digital signal processing (DSP) algorithms, and FPGA implementations can be orders of magnitude faster than microprocessor implementations. As many commercial and national security satellites are DSP-oriented, many organizations have started using commercial FPGAs to process data closer to the sensor. Using commercial technology successfully in this environment has lead to new research into fault tolerance and resilience.**

I. Introduction

Ubiquitous sensing is an important aspect of the national security program. More and more, this data collection is handled by satellites designed to monitor global events. The satellite data is currently sent to ground stations for further processing, such as removing sensor artifacts from the data or doing feature extraction, so that policy analysts can make decisions based on this data. Over time the number of sensors and the amount of data these sensors produce has increased. As the *telemetry pipeline* responsible for communicating the data to the ground station is small, this situation has led to what is often called the *telemetry bottleneck*, which limits the amount of data that can be transmitted.

In an attempt to optimize the telemetry, several organizations have been researching the use of commercial electronics to provide on-board processing. Commercial static random access memory (SRAM) field-programmable gate arrays (FPGAs) has made in roads into space-based computational platforms for the past decade [1], [2], as these devices provide custom hardware speedups without the cost of fabricating custom silicon. Furthermore, the device's ability to be reprogrammed while on orbit could increase the usable lifetime of spacecrafts by allowing the FPGA's circuit to be modified to meet changing mission and sensor needs. Unfortunately, commercial FPGAs are not radiation-hardened and radiation-induced faults, called single-event upsets (SEUs), can cause the user circuit to output bad data. This research has shown that a variation of triple-modular redundancy (TMR) can be used effectively to mask radiation-induced errors [2] and automated tools for applying .have been created [3].

The first demonstration of this research at Los Alamos National Laboratory was launched on the Cibola Flight Experiment (CFE) satellite in March 2007, which had nine Xilinx Virtex-I field programmable gate arrays (FPGAs) for a software-defined radio application. Since its launch we have been monitoring CFE's hardware for SEUs. While there have been more than 100 SEUs since launch, the FPGAs have been able to continue processing fault tolerantly.

This primer will provide a quick introduction into FPGA hardware (Section II), how radiation affects FPGAs (Section III), and how to compute fault tolerantly in space with commercial FPGAs (Section IV).

II. Field-Programmable Gate Arrays

FPGAs are a type of programmable logic device that consists of a fabric of programmable logic and routing tiles. A user circuit is *programmed* onto the FPGA so that data can be processed. Because the unprogrammed device is completely blank, the user circuit is responsible for determining how the FPGA inputs/outputs data, how data is processed, and how data is internally routed. User circuits are *described* textually in a *hardware description language*, such as VHDL or Verilog, and then translated through synthesis and design flow tools into a bitstream that can be used to program the FPGA. Many commercially-available FPGAs are designed to be *reprogramable* so that the user can change the FPGA user circuit as often as wanted or required. In

this manner, the user circuit can reflect the most up-to-date algorithm for processing a particular data set. Furthermore, a particular user can process many different types of data with different types of user circuits without buying new hardware for each data set.[1]

A. FPGA Architecture and Components

A diagram of the reconfigurable fabric for a Xilinx Virtex-II FPGA is shown in Figure 1. The architecture is broken into a number of programmable components: input/output blocks, logic, user memory, and routing. Programmable input/output blocks allow the designer to choose which pins to use on the device to transfer data to and from the microprocessor or memory. The basic logic units are programmable lookup tables and embedded multiplier cores. In the more recent devices, embedded microprocessors are also available. Because lookup tables make up the majority of the FPGA fabric, they will be described in greater detail below. Programmable user memory is also available from flip-flops and on-chip SRAM. Finally, there is a variety of programmable routing available so that logic units can be combined in a manner that minimizes both the area and the speed of the circuit.

For each of these different architectural components, some aspect of the component is programmed through setting configuration memory bits. Configuration memory bits are used to populate the values in a lookup table, initialize ROMs, select a wire to connect two logic units, and determine which input/output pin to use. The configuration memory is one of the strengths for commercial FPGAs and allows the user to change the user circuit programmed at will. Unfortunately, as discussed in Section III, the configuration memory bits are susceptible to radiation-induced effects when used in space.

Lookup tables are a hardware data structure that is analogous to an array in software. Table I shows an example of a 4:1 lookup table that takes four 1-bit inputs and returns a 1-bit output. In this particular example, the lookup table shows the logic function for a 4-input AND gate, where only the time a one is returned is when all of the inputs are one. For Xilinx FPGAs, lookup tables are made from configuration memory bits and decoding logic. The configuration memory cells store the output values. The decoding logic takes the input values and determines which memory location should be used as the output. Lookup tables can also be programmed to act as RAMs, ROMs, and shift registers. While this type of logic unit might seem tiny in comparison to microprocessors, the synthesis tools are adept at translating very complex arithmetic operations into lookup tables so the designer can focus on the algorithm at a much higher level of abstraction.

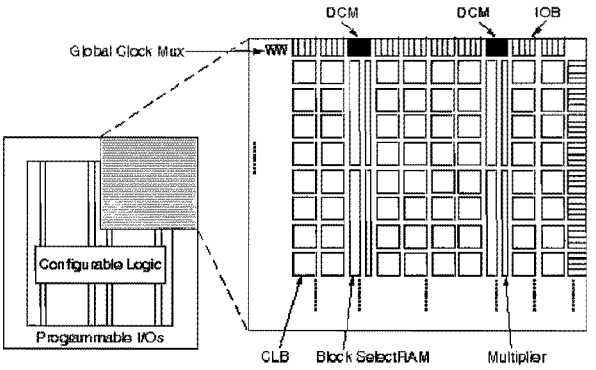


Fig. 1. A Diagram of the Xilinx Virtex-II FPGA Fabric, used courtesy of The Xilinx Corporation

TABLE I
A 4:1 LOOKUP TABLE FOR AN AND GATE

IN_3	IN_2	IN_1	IN_0	OUT
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

B. Uses for FPGAs

Originally, FPGAs were used as a platform for debugging and simulating designs that were intended to be fabricated into application-specific integrated circuits (ASICs), because using FPGAs were cheaper than fabricating a test ASIC device and faster than software-based debugging. For the past 10-15 years, though, FPGAs have been widely adopted as a platform for image and digital signal processing. These types of algorithms can take advantage of the low-level architecture of the FPGA and the flexibility of the FPGA fabric to implement

[1]For more information, the reader is directed to the book "Reconfigurable Computing: The Theory and Practice of FPGA-Based Computations" [4].

hardware algorithms that are 10-1000 times faster than microprocessor-based algorithms and cheaper than an ASIC to develop.

For example, in image processing, most image processing algorithms do channel-level processing on each pixel in the image. For the case of standard color image, this means that each each pixel is broken into the 8-bit red, green, blue, and transparency channels, each channel is processed separately, and finally the channels are recombined into pixels. For the case of a hyperspectral satellite image, though, each image could have thousands of channels. While splitting and recombining pixels in software requires bit-level mathematical operations, in hardware the same operations can be done through routing. Unlike microprocessors that might not have enough resources to perform the same operation on all of the channels at one time, FPGAs usually have enough resources to allow the channel-level and pixel-level operations to be performed in parallel. Finally, since the basic unit of computation is the lookup table, the user circuit can be optimized to work on irregularly-sized data, such as a 10-bit channel. By using all of the possible parallelization possible in the algorithm and by optimizing the operations for the size of the data, FPGAs can often process images much faster than a microprocessor.

III. How Radiation Affects FPGAs

Space-based electronics must be able to withstand the radiation environment and still be able to process reliably over the usable lifetime of the mission. While there are number of radiation-induced faults that could beset space-based hardware, most designers are concerned about total ionizing dose (TID) and single-event effects (SEEs). SEE can take many forms, such as single-event latchup (SEL), single-event transients (SETs), single-event upsets (SEUs), and single-event functional interrupts (SEFIs). In this section, these effects and how they affect FPGAs are discussed in greater detail.[2]

A. Radiation Effects in SRAM-based Devices

While deployed, the voltage and switching characteristics of transistors can change gradually with long-term exposure to protons and electrons [6]. Space-bound electronics are tested for the maximum amount of radiation the device can accumulate before it cannot be used reliably. The amount of radiation a deployed system will endure is dependent on the orbit and the mission duration. For a low earth orbit (LEO), 100 kRads of total ionizing dose should be sufficient for several years of reliable operation, which is a requirement that the Xilinx Virtex family devices meet.

There are three primary forms of SEE that SRAM-based devices are concerned about: SEL, SEU, and SEFI. While there are a handful of SEE types that can damage a device, SEL is the predominant concern. Latchup is an issue that semiconductor manufacturers are already concerned about for terrestrial electronics reliability since it can destroy semiconductor devices through excessive current draw. Complementary metal oxide semiconductor (CMOS) technology is prone to latchup due to the parasitic transistors that result from integrating PMOS and NMOS transistors. SEL is a radiation-induced version of this destructive mechanism, where the charge implanted from the ionizing particle causes current to flow in the parasitic transistors. Designers generally avoid devices that are prone to SEL since many systems cannot tolerate the risk of having a damaged device while on orbit. For this reason, Altera's product lines have been avoided [7]. Xilinx products are free from latchup in the presence of protons and heavy ions.

The most common radiation-induced faults in SRAM-based FPGAs are SEUs (or *upsets*). SEUs affect memory devices by changing the stored values in memory bits, which could change the implemented circuit or the circuit's state in SRAM-based FPGAs. Specific failure states caused by SEUs are discussed in Section III-B. SEFIs are SEUs that cause more global functional effects and may require a device reset for device functionality to return. In SRAM-based FPGAs, SEFIs are often caused by SEUs in the control logic of the device, such as the internal control registers or the configuration interfaces (JTAG or SelectMap). Detecting and mitigating SEFIs is a challenge since the affected state can not be easily observed or fixed from the user standpoint. Specific SEFIs are discussed in Section III-B.

B. Failure modes in FPGAs from SEUs and SEFIs

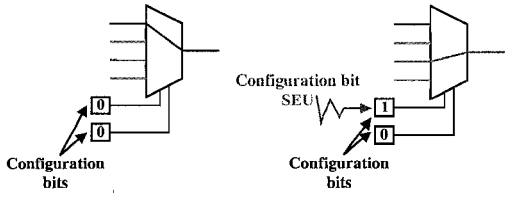
FPGAs have many SEU-induced failure modes that conventional ASICs circuits do not have. For example, by changing one configuration bit, a lookup table resource may no longer operate as a simple lookup table, a wire might not connect the same two endpoints, or an input may suddenly be coming from somewhere else. For this paper, we will classify errors in this manner: failure modes that affect the circuit functionality, failure modes that affect the circuit's state, and failure modes that affect the device's functionality.

[2]For a more in-depth discussion of radiation effects, the authors suggest "The Radiation Effects Handbook" [5].

1) Failures that Affect Circuit Functionality: For SRAM-based FPGAs the circuit functionality is vulnerable to three types of changes: routing, lookup tables, and tie offs. While maintaining the lookup table functionality is of the utmost priority for fault tolerant computing, the routing network and the tie offs are equally as important, if not more so, to maintaining circuit functionality. SEUs in the routing network can sever wires, making the transmission of the intermediate data values or the clock impossible. Furthermore, SEUs in tie offs can cause incorrect values to be injected into adders and multipliers. The vulnerabilities of these components is discussed in detail below.

Routing Vulnerabilities: In the Xilinx Virtex family FPGAs, the routing network largely consists of multiplexers, programmable interconnect points (PIPs), and buffers. In the older devices wires were connected using pass transistors (PIPs), whereas the newer devices use multiplexers. Finally, buffers are used where wires need to be actively and selectively driven by a few sources. These three resources are discussed below.

The select lines for routing multiplexers control which route is configured. These select line values are stored in configuration memory. An SEU in the select line configuration bits will cause a different routing configuration to be used. An example multiplexer select failure is shown in Figure 2. In practice, this could cause an input to float if the new input is not driven.

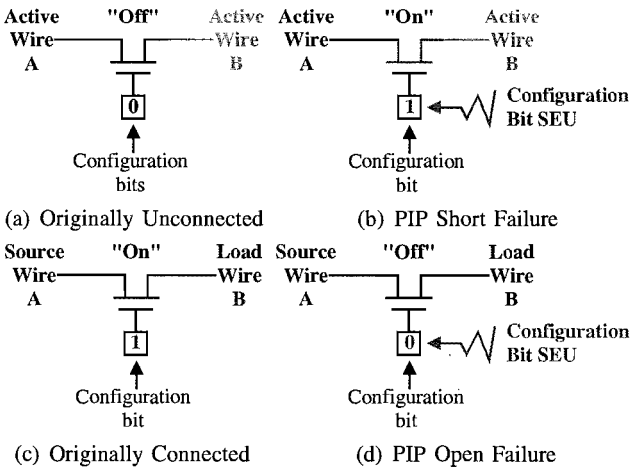


(a) Original (b) After Upset

Fig. 2. Multiple xor Select Failure Example

PIPs have two kinds of SEU-induced failures: shorts and opens. Figure 3(b) depicts 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. Figure 3(d) shows a PIP open failure. This failure effectively breaks a wire into two pieces and interrupts the flow of information from one part of the design to another.

Buffers, like PIPs, either short or open when they fail. The main difference between buffers and PIPs is that a buffer failure is 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



(a) Originally Unconnected (b) PIP Short Failure

(c) Originally Connected (d) PIP Open Failure

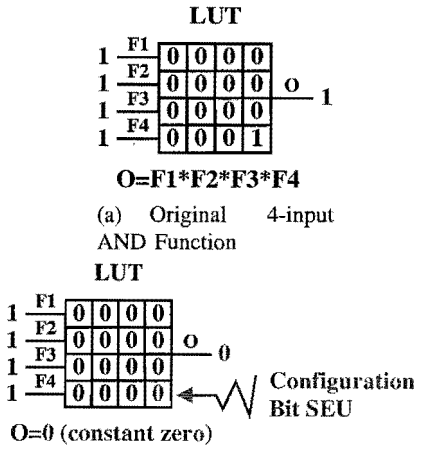
Fig. 3. PIP Failure Mode Examples

PIP, but with a buffer failure only the output is affected. As buffers usually are placed on the outputs of some multiplexers and on bi-directional wires, an SEU could cause a wire to be undriven.

Logic Vulnerabilities: There are two types of logic vulnerabilities: lookup table value changes and control bit changes. The Xilinx Virtex family FPGAs use lookup tables to generate most logic functions, so a change in the values stored in a lookup table 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 of a lookup table value change is shown in Figure 4. Here the lookup table 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.

In contrast to lookup table value changes, control bit changes generally cause errors for all, or almost all, possible circuit inputs. Many of the architectural components use quite a few control bits to determine miscellaneous functionality. Some of these control bits are programmable inversion bits that can wrongly cause the value carried on that particular wire to be inverted, if affected by an SEU. There are also control bits that determine whether the lookup table performs as a lookup table, a 16x1 dual-ported RAM, a part of a 32x1 RAM, or as a programmable shift register. If a lookup table suddenly turns into a shift register, output errors will likely result.

Tie Offs: Tie offs are needed to generate constant zero and one logic values used internally by FPGA designs [8]. "Implicit" logical constants are widely used in



(a) Original 4-input AND Function

(b) Upset lookup table Function (Constant "0")

Fig. 4. lookup table Upset Example

designs to drive inputs to I/O, logic, RAM, clocking, and other resources. "Explicit" logical constants are needed for the zeroth bit of the carry chain for adders and unused multiplier/DSP inputs. Each device in the Virtex line has designed different approaches to tie offs. In all of the devices, the implicit logical constants were implemented with half-latches. In earlier devices, upsets in a half-latch could not be fixed through on-line configuration and would remain "stuck" in the upset value. In current generation, the half-latch device leaks off the extra charge within a second, returning the circuit to normal operation. Explicit tie offs are often generated using lookup tables that have been programmed to implement a constant one. These explicit tie-offs are susceptible to all of the usual radiation-induced errors with lookup tables.

2) Failures that Affect Circuit State: Maintaining a circuit's state can be difficult on orbit, as the state is vulnerable to SEUs that affect circuit functionality or SEUs in the user memory. In particular, when a circuit's functionality is affected by an SEU, incorrect intermediate data values could be generated. After the circuit's functionality is repaired through reprogramming, the bad state data generated during the error state will remain until it either naturally flows out of the system in feed forward circuits or the circuit is reset under more pathological conditions.

SEUs can also directly affect user memory, such as user flip-flops or user SRAM, that are used to store the circuit's state. While it is possible to discern upsets to any user-specified ROMs in the bitstream, the state of most user memory tends to be very dynamic, changing on a clock-cycle basis in some cases, and it is, therefore, hard to distinguish an error state from normal operation. Furthermore, it is not generally possible to read the contents of the memory while it is actively being used in a circuit without the possibility of affecting its content.

3) Failures that Affect the Device's Functionality: The device has a handful of configuration and control registers that result in SEFIs. The most common SEFIs are SEUs in the configuration ports of the device – SelectMAP and JTAG – that interrupt the functioning of these ports. Because these ports are used to change and repair the circuit functionality, malfunctioning configuration ports mean that the design cannot be repaired or errors in the programming cannot be detected until the configuration port is reset. SEFIs to the configuration control logic is also possible and cause the device to wipe the data from configuration memory, leaving the device unprogrammed. While the effect of SEFIs is far more detrimental than SEUs, SEFIs happen far less frequently than SEUs.

IV. Fault-Tolerant Computing with FPGAs in Space

Without proper mitigation, there is no guarantee that output data will not be corrupted by SEUs. Accumulating SEUs increases the likelihood that output data corruption occurs. Accumulating upsets will also cause the device to draw more current, which could critically affect the device or the battery resources on the spacecraft. Therefore, mitigation and repair of SEUs is essential for reliable computation. To date, the best option found for mitigating SEUs [9] is to mask SEUs through the use of redundancy-based methods, such as triple-modular redundancy (TMR). On-line reconfiguration, called *scrubbing*, can be used to remove SEUs from devices. Previous research has shown that TMR with scrubbing is an effective method of masking the effects of SEUs [10]. As stated before, detecting SEFIs internally on the device is impossible and SEFI detection/mitigation is usually handled by an external scrubbing circuit. These two concepts will be described in better detail in the following two sections.

A. Triple Modular Redundancy (TMR)

The concept of a redundancy-based masking scheme was first introduced by von Neumann in 1956 [11]. While redundancy-based masking schemes can have any number of redundant copies of the circuit, the minimum number for masking is three copies. Voters compare the modules' outputs and output the majority value. Internal voters on SRAM-based FPGAs are susceptible to SEUs and are often triplicated, as shown in Figure 5, to remove the possibility that the data can be corrupted

in the voters. For those applications that require maximum reliability for the FPGA hardware, we strongly recommend that TMR be applied to all aspects of the design, including I/O, global signals, logic and voters. Our recent work has shown that not triplicating global signals can make TMR-protected circuits less reliable than unprotected circuits [12].

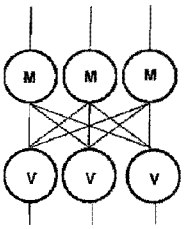


Fig. 5. The recommended implementation of TMR functional with triplicated modules (M) and voters (V).

The robustness of TMR masking schemes is dependent on failures existing in only one module at any give time. Previous research shows that fully TMR-protected FPGAs circuits have had the single-bit SEU cross-section completely removed from a design, leaving the design vulnerable to only SEFIs and non-single-bit SEUs. When faults exist in different redundant modules, masking cannot be guaranteed. Therefore, scrubbing is necessary to maintain the assumption that at most one modules is broken at a time. As long as the repaired module can resynchronize it's state before the next SEU occurs, there should only be one fault in the system at any given time. In larger, newer systems, the chance that either multiple-independent upsets (MIUs) or a multiple-bit upset (MBU) [13], where a single-ionized particle causes multiple upsets, occurs has become increasingly more likely. We have shown in previous work that designs that are able to mask all single-bit upsets might not be able to mask all MBUs [14]. Currently, there is no method to mask all MBUs. MIUs can be mitigated more effectively when the design is broken into several partitions and the smaller partitions are voted on. With this method, the TMR-protected design should be able to withstand more upsets, as long the upsets do not cause two or more modules in the same partition fail.

Because design tools perform many steps of translation and optimization on SRAM-based FPGA circuit designs, even the most careful descriptions of TMR-protected circuits in hardware description languages are often undermined by the synthesis tools removing all or some of the redundant logic. Furthermore, tying off all of the tristate buffers and unused inputs is difficult at the VHDL level. To circumvent these problems, we strongly recommend using one of the two tools that automatically apply TMR to post-synthesis circuit representations. One tool, called the BL-TMR tool, was developed by BYU and LANL and specializes in automatically applying partial TMR to designs when the fully triplicated design will not fit on the device [3]. The second tool is from Xilinx and is called the TMRtool [15]. Both of these tools will automatically apply TMR to the design properly.

B. Scrubbing

Scrubbing uses on-line reconfiguration — a feature unique to Xilinx SRAM FPGAs — to reload the FPGA's configuration bitstream during circuit operation, removing any SEUs that may have accumulated in the bitstream between scrubs. While Xilinx provides some guidance [16], Xilinx should be engaged to guarantee scrubbing is done properly. In the past, "blind scrubbing", where the programming data is continually rewritten without reading the data back to ensure data integrity, was used frequently. Over the years, though, the control logic and registers necessary for scrubbing have grown larger and SEUs in these areas during scrubbing have been observed to cause the Scrub SEFI that causes a high current state on the device [17].

One recommended scrubbing algorithm is as follows:

1) Readback the configuration data.
2) Complete a CRC check for each configuration data frame.
3) If the CRC value does not match, scrub the frame.

Since the device is not scrubbed end-to-end, the Scrub SEFI should only affect one frame instead of multiple frames. Often times, when the device has suffered a SEFI, the number of frames that need to be scrubbed in a single scrub cycle will increase dramatically. Therefore, an effective method of detecting SEFIs is to keep track of how many frames are scrubbed in a scrub cycle. If this number exceeds some threshold, then a complete reconfiguration of the device is done and the circuit state is reset to fix the SEFI-affected logic.

V. Conclusions

In conclusion, we have presented a brief introduction to how commercial electronics are being used as part of the national security ubiquitous sensing program in spacecraft. Since these devices were not intended for the harsh radiation environment, their sensitivity to radiation must be measured and fault tolerance techniques must be studied. LANL's first demonstration of this hardware has been working successfully on orbit since March 2007.

REFERENCES

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

[2] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A Fault Injection Analysis of Virtex FPGA TMR Design Methodology," in *Proceedings of the 6th European Conference on Radiation and its Effects on Components and Systems (RADECS 2001)*, 2001.

[3] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-Induced Persistent Error Propagation in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2438 – 45, 2005.

[4] S. Hauck and A. DeHon, Eds., *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing*. Morgan Kaufmann, 2007.

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

[6] H. Barnaby, *Evolving Issues for the Application of Microelectronics in Space*, ser. Short Course Notebook for the Nuclear and Radiation Effects Conference. IEEE, 2005, ch. Total Dose Effects in Modern Integrated Circuit Technology.

[7] S. L. Clark, K. Avery, and R. Parker, "TID and SEE testing results of the Altera Cyclone Field Programmable Gate Array," *IEEE Radiation Effects Data Workshop*, pp. 88 – 90, 2004.

[8] P. Graham, M. Caffrey, M. Wirthlin, D. E. Johnson, and N. Rollins, "SEU mitigation for half-latches in Xilinx Virtex FPGAs," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2139–2146, December 2003.

[9] K. Morgan, D. McMurtrey, B. Pratt, and M. Wirthlin, "A comparison of TMR with alternative fault-tolerant design techniques for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2065 – 2072, 2007.

[10] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR Techniques in the Presence of Single Event Upsets," in *Proceedings fo the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*. Washington, D.C.: NASA Office of Logic Design, AIAA, September 2003, p. P63.

[11] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies*, C. Shannon and J. McCarthy, Eds. Princeton University Press, 1956, pp. 43–98.

[12] H. Quinn, P. Graham, and B. Pratt, "An automated approach to estimating hardness assurance issues in triple-modular redundancy circuits in Xilinx FPGAs," accepted to the IEEE Nuclear and Space Radiation Effects Conference 2008.

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

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

[15] "Xilinx TMRTool User Guide," on web: http://www.xilinx.com/products/milaero/ug156.pdf.

[16] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration: Application Note 216," on web: http://www.xilinx.com, 2000.

[17] C. W. Tseng, C. Carmichael, and G. Swift, "Optimizing configuration management for SEU mitigation in Xilinx Virtex-4 FPGA and self-scrubbing," http://nepp.nasa.gov/mafa/talks/MAFA07_20_Allen.pdf.