HARDWARE, SOFTWARE AND DATA ANALYSIS TECHNIQUES FOR SRAM-BASED FIELD PROGRAMMABLE GATE ARRAY CIRCUITS

THESIS

Eugene B. Hockenberry, Captain, USAF

AFIT/GE/ENG/08-11

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GE/ENG/08-11

HARDWARE, SOFTWARE AND DATA ANALYSIS TECHNIQUES FOR SRAM-BASED FIELD PROGRAMMABLE GATE ARRAY CIRCUITS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Eugene B. Hockenberry, BS

Captain, USAF

June 2008

AFIT/GE/ENG/08-11

HARDWARE, SOFTWARE AND DATA ANALYSIS TECHNIQUES FOR SRAM-BASED FIELD PROGRAMMABLE GATE ARRAY CIRCUITS

Eugene B. Hockenberry, BS

Captain, USAF

Approved:

// SIGNED //	16 June 08
Yong Kim, PhD, (Chairman)	Date
// SIGNED //	16 June 08
James Petrosky, PhD, (Member)	Date
// SIGNED //	16 June 08
LaVern Starman, Maj, (Member)	Date

AFIT/GE/ENG/08-11

Abstract

This work presents a built, tested, and demonstrated test structure that is low-cost, flexible, and re-usable for robust radiation experimentation, primarily to investigate memory, in this case SRAMs and SRAM-based FPGAs.

The space environment can induce many kinds of failures due to radiation effects. These failures result in a loss of money, time, intelligence, and information. In order to evaluate technologies for potential failures, a detailed test methodology and associated structure are required. In this solution, an FPGA board was used as the controller platform, with multiple VHDL circuit controllers, data collection and reporting modules. The structure was demonstrated by programming an SRAM-based FPGA board as the device under test (DUT) with various types of adders, counters and RAM modules. The controllers, hardware, and data collection operations were tested and validated using gamma radiation from a Co-60 source at the Ohio State University Nuclear Reactor to irradiate the DUT.

The test structure is easily modified to allow for a broad range of experiments on the same DUT. In addition, this structure is easily adaptable for other memory types, such as DRAM, FlashRam, and MRAM. These additions will be discussed further in this document. The system fits in a backpack and costs less than $1000.

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr Yong Kim, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated. I would, also, like to thank my wife and children for their support during this endeavor.

Eugene B. Hockenberry

Table of Contents

Page

List of Figures

Page

List of Tables

Page

HARDWARE, SOFTWARE AND DATA ANALYSIS TECHNIQUES FOR SRAM-BASED FIELD PROGRAMMABLE GATE ARRAY CIRCUITS

I. Introduction

Failure of space assets can have high costs in terms of intelligence, communications, money and lives. The use of SRAM-based FPGAs is becoming more widespread since they are re-programmable, low-cost, and more versatile. With this motivation in mind, it naturally leads to the problem that there needs to be a methodology to evaluate these devices for potential failure modes and weak points in the physical structure of the circuit. Further, a test structure is required that can adequately implement this methodology to successfully determine weak points within these FPGAs that can be taken into account during the design phase of development. The solution proposed in this work is a combination of hardware and software that researchers can use to isolate areas of FPGAs, and potentially other memory devices like SRAM, MRAM, FlashRam, etc.

There is a need to develop a robust, re-usable, and stable test methodology for various types of memory technologies. Field Programmable Gate Arrays (FPGAs) can come in several varieties such as Static Random Access Memory (SRAM)-based and anti-fuse-based, use varying layers of metal, and low-power to name but a few. The type under investigation in this work based on SRAM. Further, once a methodology is established it must be implemented in a test platform. Preferably this platform would be easily adaptable to other memory technologies and applications that can employ a similar methodology.

Radiation experiments have been conducted on various memory technologies and circuits for decades. SRAM-based FPGAs have many advantages, as they are easily re-programmable, can allow for many design iterations in a short amount of time, and can be modified at any time after programming in the field by simply reloading a new bitstream. In recent years the use of FPGAs in space electronics has become significant due to their low cost and ability to be re-programmed to fix errors, or even add new functions and capabilities while in the field. This has important implications, since failures in these systems can cost millions of dollars, loss of communication capability, loss of intelligence, and in the worst case, cost lives.

An FPGA is comprised of many elements including switch blocks, SRAM cells, look up tables (LUTs), multiplexers, flip-flops, buffers, adders, and various other elements [6,7]. The transistors that make up these elements can vary in size, with some being larger or smaller leading to a variety of potential radiation effects. For example, the Virtex-4 FPGA has thirteen different transistor sizes, with three different thicknesses of gate oxide [23].

Many of these effects under investigation have relevance in more than just the space environment. Due to decreasing feature size, thinner oxides, higher packing density, and lower power supply voltages in next generation ICs, many of the effects that have previously been observed and studied in space systems will become more prevalent in terrestrial computer systems and circuits. Radiation effects can vary greatly, leading to completely different failure modes in different radiation environments.

Further complicating analysis is that transistors with gate oxides thicker than 20 nm [1], are typically susceptible to radiation-induced oxide and interface traps while those with thinner oxides are more susceptible to gate leakage and rupture [1]. All transistors can be susceptible to single event transients (SETs), which can become a single-event upset (SEU) and manifest themselves as logic errors. The Xilinx Virtex-4 FPGA is comprised of transistors that fall into both categories, making it uncertain which effect has the greatest consequence, and more importantly which region is the most susceptible to radiation and causes failure.

There are several techniques to mitigating the effects of radiation. These are investigated and discussed further in Chapters 2 and 3, in order to analyze a robust, programmable test setup for isolating the primary radiation failure mechanism in an FPGA. This single hardware setup can be used to experimentally investigate dozens of distinct effects in numerous modes and configurations. Radiation experiments need to be highly tuned and carefully planned to appropriately measure the feature of interest; for instance the clock (period, frequency, skew, rise-time, and fall-time), logic blocks, SEUs, latch-up, and a host of others. The test structure presented here is easily tunable to look for a variety of effects by modifying parameters in a centralized location. In many cases experimenters may want to test different chips from the same manufacturer, across multiple manufactures, across technology generations, etc. This test setup can provide for a common baseline and repeatability of experiments across multiple types of hardware. Test control, stimulus control, and data collection are additional variables that can be introduced. These variables make it hard to distinguish between real differences

and simply something the test setup influenced or reported differently or not at all. Standardizing the data collection parameters across various technologies and devices makes data analysis much less intensive and should result in better analysis overall.

One of the problems that researchers face in experimentation is that too often they must spend a majority of their time building a test setup that targets their specific project or chip, and not enough time organizing, conducting, and analyzing experiments to make useful contributions. With this structure, six months or more of labor and time are mitigated that can be re-directed to the real task, accomplishing experiments and doing data analysis.

This work demonstrates the viability of this setup, as well as presents results of its use in an experiment at a reactor facility. The code generated can easily be tuned to allow for a wide range of experiments that can provide a wealth of data. The tunable parameters include; clock frequency, clock pulse width, write frequency, read frequency, and perhaps most importantly, the data collection parameters. The capability of the setup is proven using a variety of adder, counter, and BlockRam modules. This work will mitigate six months or more of effort for researchers and provide a consistent experimental baseline to remove variability from test-to-test and device-to-device. This time can be re-directed to parameter setup, experimentation, and analysis.

II. Literature Review

2.1 Chapter Overview

This chapter will summarize the applicable literature regarding radiation effects on electronics, SRAM and FPGA function, FPGA vulnerabilities, and radiation hardening techniques for circuits and memories important to this work. This requires a background in normal operation of NMOS and PMOS transistors and the function of the SRAM cells that comprise the FPGAs as discussed in the following sections. Finally, radiation impacts on transistors and circuits, as well as various hardening techniques at the process and design levels will be discussed.

2.2 Normal transistor operation

A transistor can be thought of as a switch that either goes to ground or allows voltage to pass in order to transfer logic 0 or 1 respectively. It acts as a switch through application of a positive voltage to turn on an NMOS, or a negative voltage to turn on a PMOS.

Typically for an NMOS, the substrate and source is connected to ground. When a positive voltage greater than zero but less than the threshold voltage is applied to the gate, holes are pushed away from the interface between the substrate and the gate oxide creating a depletion region. As the voltage approaches the threshold voltage, electrons accumulate in the channel. This creates a conducting path for charge to flow through the channel. The speed with which the switch functions is dictated by channel length and

width, oxide thickness, gate insulator dielectric constant, carrier mobility, and threshold voltage (obtained from the threshold voltage equations) among others.

2.3 SRAM Function

Static Random Access Memory (SRAM) is a type of memory that retains its value without the need to refresh as long as power is applied to the cell. Once the value

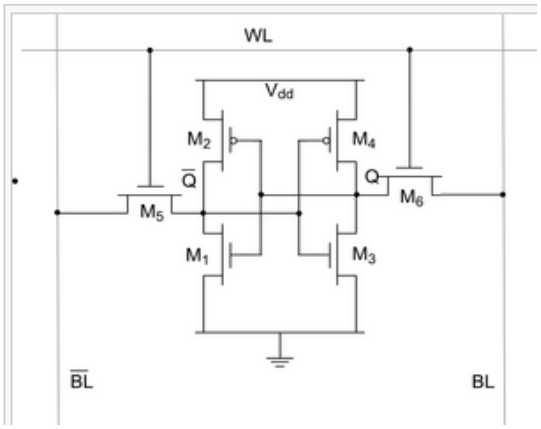

Figure 1: Basic Six-transistor SRAM cell. Two cross-coupled inverters keep the data stored as long as power is applied to the cell [22]. In the text, BL' is used to represent $\overline{BL}$.

is written to the cell, it is passed back and forth between two cross-coupled inverters, or where one inverter's output feeds the other inverter's input thereby retaining the value. SRAM is more expensive than Dynamic Random Access Memory (DRAM), but has several advantages. SRAM is faster, uses less power, and simpler to interface with since

the refresh interrupt logic is not required in order to retain the written value. The SRAM cell retains its value without refresh as long as power is applied to the system.

The basic SRAM memory cell consists of a minimum six transistors. Figure 1 shows the basic design, demonstrating two cross-coupled inverters that pass the value back and forth until the control lines are set to read from or write to it. By being cross-coupled, one inverter's output feeds the other inverter's input and vice-versa. Two other NMOS transistors control access to the cell when reading and writing. There are three modes of operation: read, write, and standby. To understand the operation, suppose a logic 1 was stored in the cell. First, the bit lines, BL and BL' are set high. On the following cycle, the WL line is set to 1 to allow the control transistors to latch the proper value. Reading a 0 results in setting the lines in the opposite manner.

During a write operation, the bit lines are loaded first. In the case of writing a 1, BL would have a 1 applied while a 0 would be applied to BL'. On the following timing pulse, WL is asserted which latches the data into the cell itself. The transistors are sized such that the values being latched by the bit lines are much stronger than the weaker voltage signals in the cell, enabling the value that is currently stored there to be overwritten [22].

During standby, the bit lines are disconnected from the internal cell. As long as power is applied, the value currently stored will simply be passed back and forth indefinitely.

2.4 FPGA Basics

SRAM-based FPGAs comprise a more significant portion of space system electronics every year. FPGAs are a flexible alternative to Application Specific Integrated Circuits (ASICs). ASICs typically yield faster, smaller, and lower power design than FPGAs. However an FPGA can be reconfigured to employ numerous circuit structures and logics. Instead of making a specific chip layout that meets a single requirement, software can use code that describes a vast variety of functions and applications and implement it using mapping technology in the FPGA.

There are distinct advantages to designing a system using FPGAs vice ASICs. These include quicker turn around times bypassing fabrication time, and requiring less time to test the circuit. In addition, a mask for a complicated ASIC can cost more than $1M, meaning that if a change is required due to a logic error after the mask had been completed, a completely new set would have to be manufactured at a significant cost. Inc contrast, an FPGA can be reconfigured quickly for a very low cost.

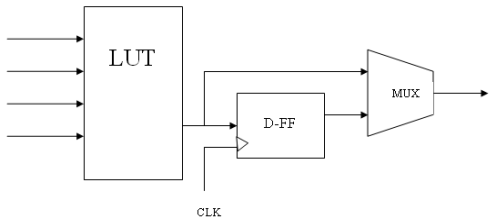

Figure 2: Basic Logic Element of an FPGA. The Basic Logic Element (BLE) is comprised of a look-up-table (LUT), Flip Flop (FF), and multiplexer (MUX)

FPGAs are made up of combinational logic blocks (CLBs) that are made of basic elements. These include look-up tables (LUTs), multiplexers, and flip-flops. In addition,

there is routing logic, pass transistors, and I/O pads. The CLB can implement any Boolean function of its inputs. The CLBs can be linked together via routing blocks to further implement more complex logic. A CLB with more inputs can implement more logic with fewer logic blocks. However, according to [6], the LUT complexity grows exponentially with the number of inputs.

The BLE has a D flip-flop to store the output from the LUT. The FF works with the LUT to form sequential circuits. A gated clock is used to keep power consumption low since unwanted FF transitions use additional power. The final element in the BLE is a multiplexer (MUX) that chooses between the LUT output and FF output to send to the routing network.

A fully connected FPGA is one in which every logic block can communicate with every other block. This configuration allows for very high logic utilization and flexibility, but routing between the blocks becomes extremely complex. In addition, the power dissipation from the wiring becomes the majority power dissipation mechanism, which is wasteful. Delays can also result as signals may have to travel back-and-forth to get to the appropriate logic block for additional processing. In order to address this issue, most FPGAs use an island cluster configuration that clusters CLBs into groups, to allow for greater processing and logic utilization while optimizing routing, power consumption, and delay. Figure 3 shows this configuration.

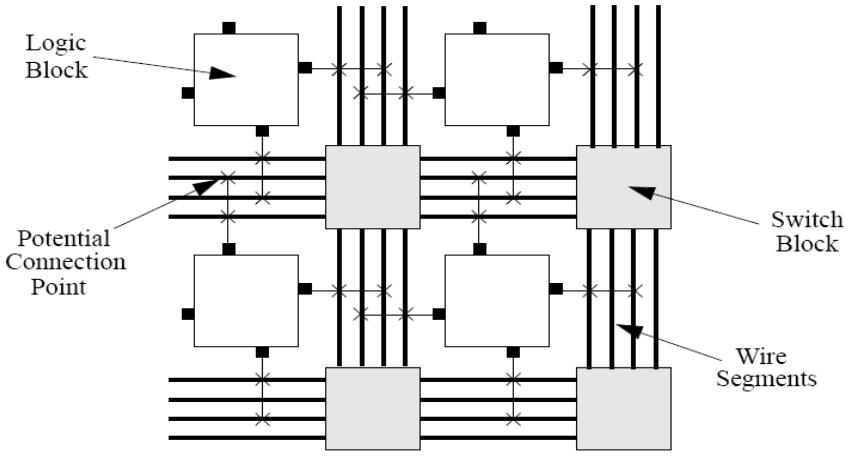

Figure 3: An island style FPGA. Local logic blocks are connected together for more complex processing [1]

2.5 FPGA Configuration Bitstream

The configuration bitstream determines how the SRAM switches are set when the FPGA is programmed. A fault here can cause the FPGA to operate incorrectly since the hardware is programmed incorrectly. The error or errors will continue until the bitstream is re-programmed correct configuration.

The normal flow for programming an FPGA starts with writing a file in VHSIC Hardware Description Language (VHDL). A software program called the synthesizer then converts that code to a hardware equivalent latching configuration, which is then mapped onto the FPGA. Finally the bitstream is downloaded to the device. These configuration bits determine the location of every structure on the device such as MUXes, LUTs, etc. If one of these bits is impacted, it will result in incorrect function until that portion of the bitstream is re-loaded.

2.6 Power MOSFETs

Until recently, bi-polar junction transistors (BJTs) were the power transfer device of choice. However, problems arose in that a base current up to 1/5 of the collector current was required to keep the device turned on [15], and a large reverse base current was required for fast turn-off. Power MOSFETs were invented to deal with this issue. Power MOSFETs use most of the same processing techniques as typical MOSFETs used in VLSI applications, but there are significant differences in the geometry, voltage, and currents. Figure 4 shows a diagram of a typical power MOSFET.

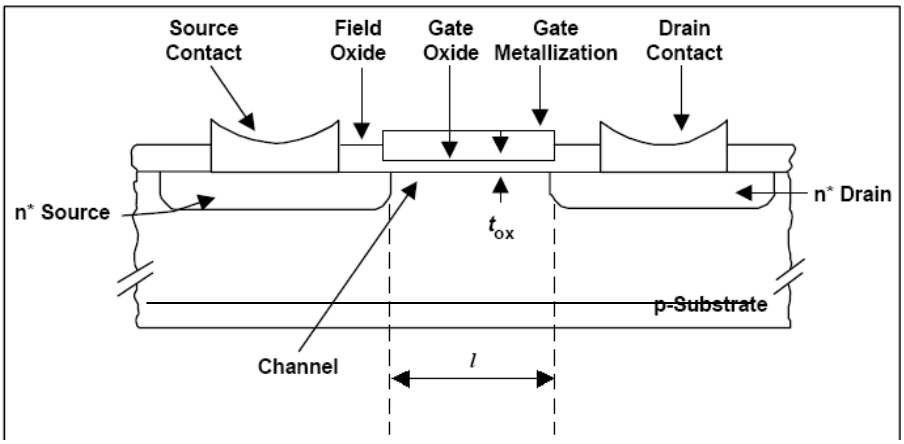

Figure 4: Power MOSFET Layout [15]

The power MOSFET provides another source for radiation sensitivity that can result in permanent failure of the device. If the power MOSFET fails, power is removed from the board and the entire configuration is lost. If the device has a hard failure, the FPGA will not be programmable anymore.

2.7 Radiation Background

Table 1 [17] summarizes the various types of particles in the space environment. These include gammas (or photons), electrons, neutrons, protons, alphas (He+ ions), heavy ions such as iron, and galactic cosmic rays. Depending on the orbit or location in space, time of day, solar activity, and terrestrial location, different particles become the radiation of interest.

Table 1 - Summary of Radiation Types with Primary and Secondary Effects [17]

Category	Particle	Mass (Kg)	Charge	Ionizing	Non-Ionizing
Photons	x-ray	n/a	n/a	Primary &	Secondary
				Secondary	
	Gamma	n/a	n/a	Primary &	Secondary
				Secondary	
Charged Particles	Electrons	9.11x10-31	-q	Energy Dependent	Energy Dependent
	Protons	1.67x10-27	+q	Energy Dependent	Energy Dependent
	Alphas	6.64x10-27	+2q	Energy Dependent	Energy Dependent
	Heavy Ions	Z Dependent	Z Dependent	Primary	Secondary
Uncharged Particles	Neutrons	1.68x10-27	n/a	Secondary	Primary

Photons travel at the speed of light, while velocities can vary significantly from a few meters per second up to close to the speed of light for particles with mass. Further discussion will be limited to gamma interactions, since that is the validation method used for this work.

Gammas have no mass or charge, and as such do not directly interact cause radiation effects. The photoelectric effect, Comption scattering, and pair production [17]

are all mechanisms that result in energetic free electrons. These electrons then impact the various circuit structures, causing the radiation effects of concern.

2.8 Radiation Effects on Transistors

There are a number of effects that radiation has on transistors. These include latchup, parasitic thyristor, oxide and interface trapping, threshold voltage shifts, gate leakage, displacement damage, transient upsets, and gate rupture. Each of these will be discussed briefly to explain the effect.

Latch-up is a high current, low voltage condition, where a p-n-p-n parasitic structure can create two self-feeding bi-polar junction transistors (BJTs). Normally, both parasitic transistors are turned off. However, ionizing radiation can cause avalanche breakdown, and use positive feedback to draw current to the point where the device won't function, and eventually lead to burning out the circuit. There are a few methods of dealing with latch-up, with the most common being to cycle power so that the positive feedback loop is broken. Another method can be to reduce the minority carrier mobility using neutron injection, but this is typically not practical in a functional circuit. A more common manufacturing method involves using Silicon-on-Insulator fabrication techniques. This effectively decouples the two side-by-side MOSFETs by inserting an insulator between them, eliminating the possibility of the parasitic p-n-p-n structure.

2.9 Vulnerabilities of SRAM

The basic SRAM cell is comprised of two, cross-coupled inverters that constantly pass the value stored back and forth. The transistors are thus "switching" at their

maximum frequency at all times during operation, limited only by the physical properties of the silicon. The sensitive nodes of a memory cell are typically the drains of the off-state transistors. When a charged particle strikes one of these nodes, the gate of the opposing transistor can be turned on, producing a bit flip. During normal cell operation, two transistors are always turned on to store either the 0 or 1.

Another effect is called the single transient effect. This is the case in which a charged particle strike can generate a transient current pulse. Depending on the circuit structure, this pulse may dissipate or be logically masked, but if it reaches a memory element with sufficient voltage and satisfies the setup and hold timing parameters of the device, the pulse can be interpreted as a valid signal and be passed along. Logical masking occurs when a SET cannot be propagated to the outputs due to controlled values in the data path.

The logic structure and operating frequency of the device will dictate the probability of a SET becoming a SEU [5]. For smaller device technologies the supply voltage is greatly reduced, and less charge is stored at each node of the device. This means that less energy is required to cause an anomaly decreases. Therefore, a transient pulse has a better chance of exceeding the required nodal charge to be read as a valid signal. Electrical masking, or having the transient pulse die out prior to reaching a memory element, also has less of an effect since the packing density increases with advanced technology and there are shorter distances between elements and logic. This increased density implies a higher operating frequency, which increases the probability of

an SEU. As devices features continue to be reduced, they are more susceptible to SEU events.

2.10 Triple Modular Redundancy (TMR)

Mitigation of SEU events in SRAM-based FPGAs includes several popular techniques. The first is Triple Modular Redundancy (TMR), where either parts or the entire circuit have outputs put into a majority voter to determine the correct output. Basically, as long as two out of the three agree on the correct value, the circuit will function as expected. There are some limitations however. First, TMR can be very expensive from both a processing and footprint standpoint. Further, it is impractical to fully triplicate every single latch and memory element in the system.

2.11 Scrubbing

The configuration bitstream is a vulnerable area, such that if a configuration bit gets changed, the circuit will not operate as expected until the design, or at least the effected portion, is reloaded. This can be mitigated by a technique known as scrubbing. Basically, the configuration bitstream is checked periodically to detect errors. Should an error be found, the relevant portion of the bitstream is re-loaded, usually from an off-chip ROM.

2.12 Error Correction Coding

In memory elements, reliability can be greatly enhanced by the use of data encoding. For example, in the code used in this work, a Hamming Code encoder and decoder was used. This effectively takes an 8-bit piece of data and produces 13-bit data.

Appendix A contains a detailed example showing how the Hamming code is implemented on an 8-bit piece of data. These additional bits are calculated by XOR-ing (even parity) or XNOR-ing (odd parity) certain bits together to produce five parity bits that are encoded with the original data. Upon reading it, the 13-bit data is passed through a decoder that checks the parity bits to determine if a flip has occurred. If an error is detected, it is automatically corrected and sent to its destination.

Some algorithms also provide feedback to automatically re-write the memory cells. This scheme can also detect multiple bit upsets, though it does not correct them. In an environment where multiple bit upsets are a primary concern there are other encoding schemes that can be employed. Reed-Solomon encoding is a commonly used technique, but beyond the scope of this work and will not be discussed further.

The major drawbacks to this approach are the additional memory storage space required and the time required to accomplish the encoding and decoding. Some of these are combinational, but many are implemented to be synchronous, requiring clock cycles and therefore limiting the speed of the system. For n-bit data, the encoded data takes approximately ln (2n), though this is not necessarily the worst impact. The encoder and decoder modules use XOR and XNOR gates, which are very bulky and area intensive. Together, it can be a serious addition to the chip area. It is important to note that these data encoding techniques are not just used for radiation or space applications, but high speed communications and memory in a wide array of applications.

2.13 MBUs

MBUs are becoming more and more common in SRAM. Ionizing radiation can cause a fault in two adjacent memory elements due to the particle depositing energy in both. This can be significantly dependent upon the angle of incidence. With decreased transistor size, it takes less energy to cause an upset, and things are packed more tightly together. As devices scale further and further down, it would seem that MBUs would defeat the encoding scheme. However, another mitigation strategy can be employed; bit interleaving. Bit interleaving requires that bits in the same piece of data are geographically separated from each other by a certain number of memory elements. This does not impact the ability to store data, just changes the index used to locate all the bits in the desired data. Many modern memory elements employ a combination of all these strategies as part of their internal function, and are invisible to the user.

2.14 Radiation impacts at the transistor level

Two gate-dielectric driven failure mechanisms are prevalent in transistors; oxide and interface trap mechanisms, and gate leakage. No additional knowledge on these topics are required for this work, but further information on the details of these mechanisms can be obtained in [12, 15, 17, and 22].

Radiation effects can be generally described as ionizing and non-ionizing. Ionizing radiation generates electron-hole pairs by depositing energy to elevate valence band electrons to the conduction band. Non-ionizing radiation is a result of elastic atomic collisions which typically results in crystal lattice defects. This causes basic

parameters that govern the basic physical characteristics of the device to change such as carrier mobility, capacitance, etc.

The effects of radiation can be mitigated in some ways by adding additional transistors in series, since then even if one transistor is always on, the other can still act as the switch. The worst case is when there are only NMOS transistors in parallel. The NOR gate is a good example of this configuration. Figure 5 shows this configuration. A failure in either transistor will result in failure of the circuit.

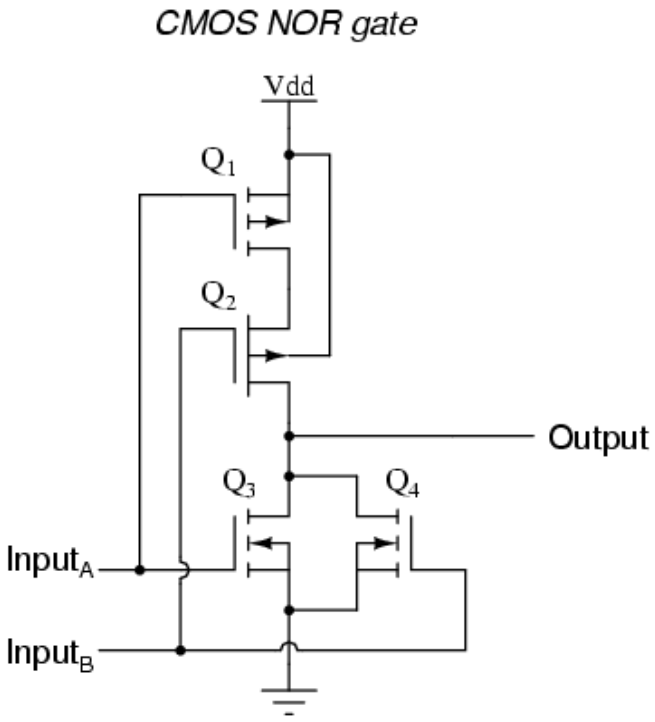

Figure 5 - A CMOS NOR Gate. A failure in either NMOS transistor will result in ground being latched to the output [10]

III. Methodology

3.1 Chapter Overview

The problem, as stated in Chapter 1, was two-fold. First, a test methodology was required to try to isolate areas more susceptible to failure. This methodology would require that data be collected to allow the user to pinpoint failure points and mechanisms. Second, a test platform needed to be constructed that would collect this data, and allow the device under test to be tested according to the methodology developed to address the first part of the problem.

Since this work is geared toward space electronics and reliability analysis of SRAM-based FPGAs, a way was needed to accomplish radiation testing on these devices. However, it was desired that this system be expandable to accommodate other memory technologies. In general, the following requirements were set forth for the test structure; it should be compact, versatile, adaptable, and low-cost.

In this work, I provide a useable, expandable test structure, consisting of a combination of hardware and software. The system is re-usable, flexible, and low-cost. Multiple parameters can be easily modified in a single parameter file to modify the system clock speed, effective clock speed, stimulus frequency, and reporting frequency to name some of the significant ones. By varying these parameters, many experiments can be accomplished that can yield significantly differing results and insights when the DUT is exposed to radiation. I validate the test structure's operation and data collection scheme by exposing a Xilinx Virtex-4 FPGA to gamma radiation from a Co-60 source.

This work looked particularly at the Xilinx Virtex-4 FPGA. These FPGAs are advertised as 90 nm technology, however different regions of the FPGA utilize varying thicknesses of gate insulator [23]. For functions that need to operate at maximum speed, such as the combinational logic blocks (CLBs) and switching network, the thinnest oxides are used. The I/O pads use the thickest oxide thicknesses, while the configuration bits in general use the middle thickness of oxide [23].

In order to test for weak points in the FPGA, various types of circuits were needed to pinpoint which part of a Virtex-4 FPGA is more susceptible to failure by stressing certain key areas. Based on how these circuits demonstrate glitches and when, the vulnerable part of the FPGA can be identified. This can be used to ascertain which parts of the circuit need to have increased reliability built in to the design. In addition, the system allows for the user to quantify how much additional protection is afforded by two reliability-enhancing schemes, triple modular redundancy (TMR) and error correction coding (ECC). Using this analysis, one should be able to ascertain which radiation effect is dominant given the technology used in the Virtex-4.

Three primary test controllers are implemented on a controller board that communicates with the FPGA on the DUT, consisting of an adder controller, counter controller, and BlockRam controller. The adder, counter, and BlockRam controllers provide all the stimuli to the circuits on the DUT, receive the raw data results back, analyze the data, and send data to a PC connected to the controller board through a Universal Asynchronous Receive and Transmit (UART) connection to a PC. The three

primary controllers communicate with the UART using a fourth controller, a UART controller.

3.2 Hardware

The setup consists of a controller board connected to a test FPGA board utilizing I/O break-out-boxes, and laptop for data transmission as shown in Figure 6. The ML405,

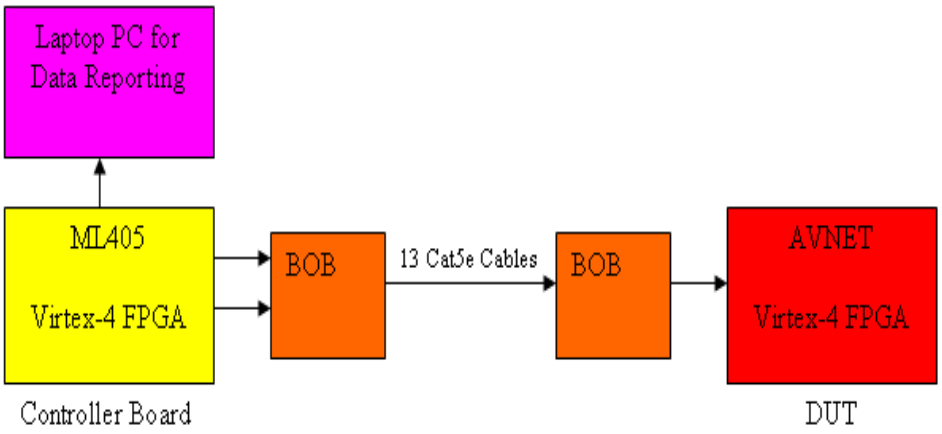

Figure 6 - Block Diagram of hardware setup. The ML405 has a Virtex-4 FPGA that sends control signals to another Virtex-4 FPGA on the DUT using the break-out-boxes (BOBs). The controller board receives the results and sends data to the PC.

The two boards were connected via ribbon cable and thirteen Cat5e cables from ML405's two 64 pin headers (only half of which were actual signals while the other half was grounded) to the 50 pin header on the AVNET FPGA board. Due to the number of control and data signals the counter and adder circuits had to be separated from the

BlockRam circuits for two separate bitstreams. A picture of the hardware setup is shown in Figure 7.

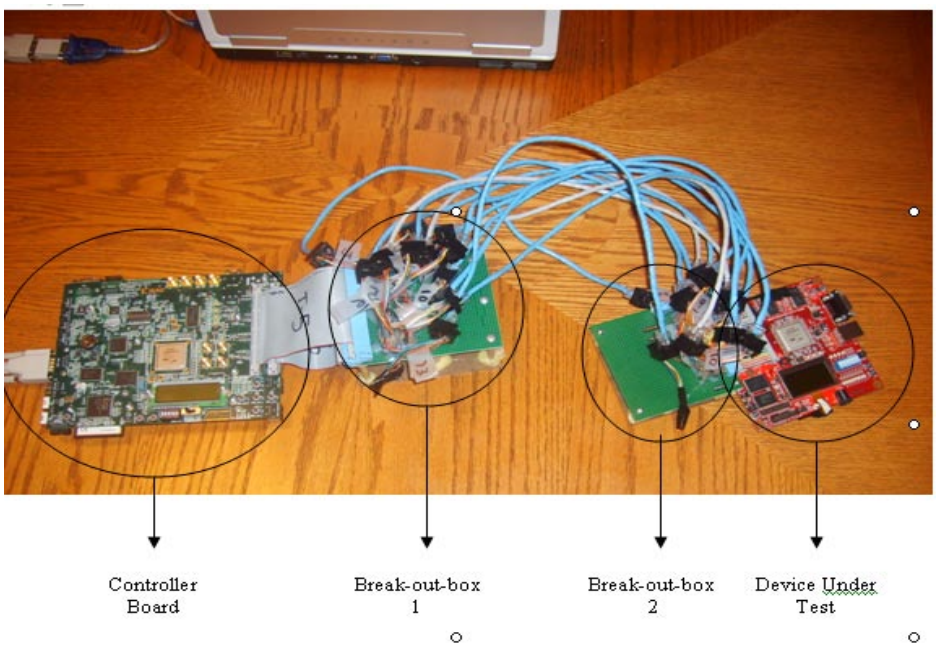

Figure 7 - Photo of the test hardware in functional configuration. The controller board and test board are connected via two breakout boxes to allow cat5e cables to run between them.

3.3 Code Structure

The controller board has four primary controller modules: Adder_Control, Counter_Control, BlockRAM_Control and UART_Control. A clock divider module, a First-in-First-out (FIFO) buffer, a transmitter module, and two finite state machines are also contained on the controller FPGA. The test FPGA board has four versions of adder circuits, four versions of counter circuits, two BlockRAM modules, and the Error

Correction Coding (ECC) modules. These are described in further detail below. For clarity, blocks in yellow are resident on the controller board. Blocks in red are code modules on the device under test, and green blocks are sub-modules.

3.3.1 Controller Board

Figure 8 shows the code modules resident on the controller board. The digital clock manager (DCM) clock module is created using Xilinx's Core Generator.

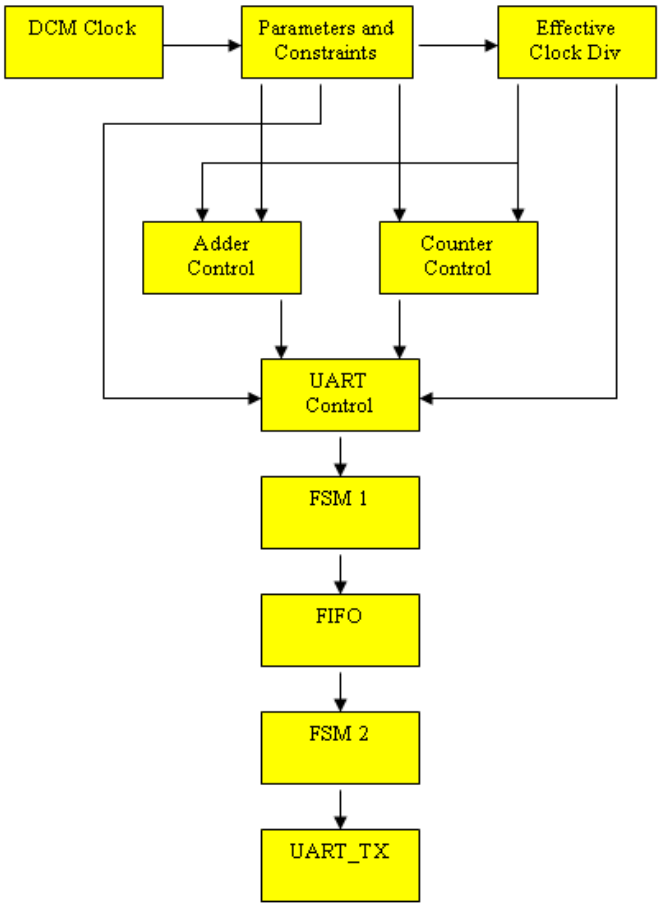

Figure 8 - Code Modules on Controller FPGA

It takes in the system board's crystal 100MHz system clock, and outputs true 200MHz, 100MHz, and 10MHz clocks. By true, it means that the clock pulse width matches the frequency, or 5 ns, 10 ns, and 100 ns respectively. These clocks are sent to the Parameters and Constraints module, where the user can determine which clocks should go to the test modules and the UART system. The Parameters and Constraints file sets the frequency for which the Adder, Counter, and BlockRam controllers send stimuli to the circuits on the test board. This file also sets the baud rate for the UART, and various data collection and reporting features.

The effective clock module takes the system clock from the Parameters and Constraints file and enables the system to run at a slower clock frequency. The Adder and Counter controller modules use the effective clock frequency to run the corresponding circuits on the test board. The UART controller, two finite state machines (FSMs), First-in-First-out buffer (FIFO), and UART_TX comprise the data reporting system to the PC.

3.3.1.1 Clock Management

3.3.1.1.1 System Clock vice Effective Clock

First, some terminology clarification when it comes to clocks. There is the main system clock, which is the clock coming directly from the Xilinx Digital Clock Manager or DCM. These include a 200MHz, 100MHz, and 10MHz clocks. Then there is an effective system clock. Suppose a user wanted to run the system at 100KHz. Given that Xilinx does not have a main system clock of this frequency we include an enable statement following every clock statement. The effective clock counter then counts the

number of cycles and only execute the code when the number of cycles has been met. An example is as follows:

```
If (clk'event and clk = '1') then
        If (clk_div_enable = '1') then
                Rest of code here
```

So if there is a main system clock of 10MHz, and it is desired to run execute the code at an effective clock rate of 100KHz, the counter would be set to count 100 clock cycles before setting the enable to 1. Therefore, the code following the enable would only be executed 100,000 times a second, or effectively a 100KHz clock. Simply using a counter, as some other architectures attempt to do, will not work since the clock pulse for a simple counter does not have the

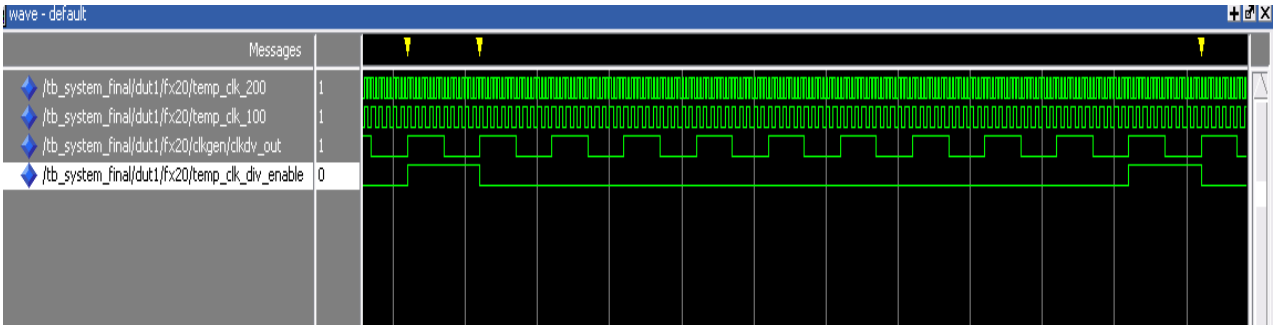

Figure 9 - Waveform demonstrating the various clocking differences. This shows 200MHz, 100MHz, 10MHz true clocks, and an enable signals that results in a 1MHz effective clock

proper period for a Xilinx FPGA. A waveform is shown in Figure 9 to demonstrate this methodology. An important thing to note, though, is that while the system is effectively

operating at 100KHz, the clock period for each pulse is still 100 ns, corresponding to the 10MHz clock and not 10,000 ns which would correspond to a true 100KHz clock. This has important ramifications when it comes to issues of cable lengths and clock control over various boards. Simply stepping down the effective clock does not improve the likelihood of the pulse being recognizable over a long cable run. In fact, testing showed

3.3.1.1.2 Stimulus Frequency

Often, when people discuss clock frequency, they are talking about the speed the circuits are being exercised at. In this case it would be how many times per second new inputs are provided to the adders and counters. For instance, suppose there is a 10MHz system clock and a 100KHz effective clock. Further, suppose the counters are to count up 10 times per second. Then, one could say the clock frequency of the counter is 10Hz, which is separate from the actual system clocks. A variable is available in the system to allow the user to dictate how many times a new stimulus is supplied to the individual circuits. This variable is set in the Constants and Parameters file, and is done by defining the number of clock cycles between stimuli based on the effective clock. So in the case above, the effective clock is 100KHz. For a stimulus frequency of 10Hz, the number of cycles would be set to 10,000 (100,000/10,000 = 10). With a system that is traveling over a long length of cabling, this can be important so that additional clock cycles can be injected to allow time and cycles for data to travel over the lines.

The maximum speed is limited by the length of the cables required to connect the two boards together. The intended strategy was to run the board at the maximum frequency during irradiation, then compare to irradiation using a slower clock frequency

to see if the allowable dose increased. The expectation was that a slower clock would result in the test circuits able to take more radiation and still function.

3.3.1.1.3 UART Clocking

For ease of re-use and to limit data errors for future users, de-coupling the UART clock from the main system clock was the best implementation method. This allows the baud rate and the data transfer mechanism to remain untouched, regardless what clock speed or rate of stimulus the experimenter decides appropriate for that particular run. The UART parameters can be modified in the Parameters and Constraints file. In this file, the UART can be set to run at any clock speed available. The UART transmitter (UART_TX) takes two clocks, the main clock and basically an enable clock that is 16 times slower than the main clock. This port, 16_x_Enable, is fed by a counter that counts to a certain number of cycles then sends a single pulse. The number of cycles is calculated as follows: Suppose the main UART is 100MHz and I want to run at 9600 baud. Then the number of clock cycles would be:

Clock speed / 16 / Baud Rate = 100MHz/16/9600 = 651.2.

The closest number is 651, which is within the transmitter's tolerance of 1-2%. Any other baud rate is supportable by modifying the number of cycles between pulses going to the 16_x_Enable port. The UART transmitter only transmits a character when a pulse goes to the 16_x_Enable port.

3.3.2 Adder Circuitry

Figure 10 shows a block diagram of the adder circuitry. The adder controller sends stimuli to the adder circuits on the test board, reads the results, and sends any errors to the UART.

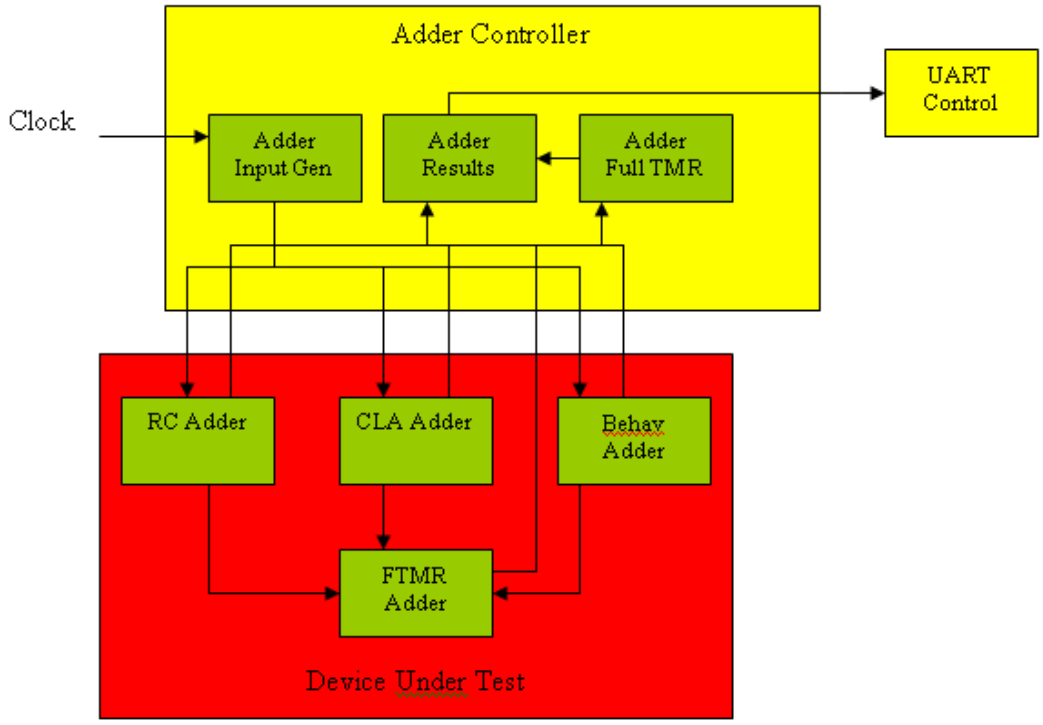

Figure 10 - Adder Control Module. The Adder Controller provides inputs to 3 adder circuits on the device under test. It then takes the results, checks them, and sends any error to the PC.

The FPGA under test contains three separate adder modules; a ripple carry adder (adder_RC), carry-look-a-head adder (adder_cla), and behavior adder (adder_behav). The purpose of looking at these adders was to see if one structure was significantly more

susceptible to radiation than the others. Each adder is implemented in a different manner in the FPGA due to the differences in how they operate. Suppose one wanted to add two 4-bit inputs. The ripple-carry adder would only calculate a bit after it had received the carry from the previous bit calculation, making it the slowest. The carry-look-ahead adder takes one clock cycle by looking at the inputs to determine what the carries will be ahead of time. The behavioral circuit allows the Xilinx synthesizer to determine the best configuration, though this may not correspond to the best configuration in a radiation environment.

The Adder controller supplies two 4-bit numbers, A and B, to all three adder modules on the test board. The B input is hard coded to be 0001 in order to save I/O pins, while the A input starts at 0 and incremented by one at the user-defined stimulus clock frequency. The discrete 5-bit (four sum bits and a carry bit) results from the three adder modules are also directed to a voting logic module on the test board to determine a functional TMR (FTMR) solution. The three adder results and FTMR solutions are reported back to the controller module on the controller board, where the non-TMR results are put through voting logic again to determine a full TMR solution, for a total of five, 5-bit adder results to be checked against the expected, or true, 5-bit result. This is accomplished every clock cycle, so even though new inputs may not have been applied to the adders, if a bit flip occurs while computing the same inputs over clock cycles, it will be captured and show up on the output.

The UART processes data serially, so all data is converted to hexadecimal to reduce the number of characters required to be transmitted by a factor of 4. This is

accomplished in the individual controllers, such that a properly formatted bitstream is provided as the input to the UART controller. The Adder UART messages are in the following format:

Add_Data Read_Data True_Adder Code_TMR Code_System Time

An example of the logic mapping methodology is in Appendix B.

3.3.3 Counter Circuitry

The counter control and circuitry structure is identical to the adder controller. The controller sends an enable signal to a One Hot state encoded counter and two Up-Down counters on the DUT. The controller receives the results, and outputs any errors to the PC.

The counter controller supplies the enable signal to the three counter modules and receives back the 4-bit results from each of the three modules. Once again, the results are put through voting logic on the test chip to generate a functional TMR solution. Finally, the non-TMR results are put through voting logic on the controller board to generate a full TMR solution.

The counter controller sends an enable signal to a one hot state encoded counter (Counter_OneHot) and a normal up-down counter that just incrementes the previous value by one (Counter_UpDown). The enable signal is calculated based on the number of cycles between pulses as defined in the Constants and Parameters file. Two instances of the Counter_UpDown and one instance of Counter_OneHot are used to implement a functional TMR on the test board. Finally, the results of the three discrete counters are sent through voting logic on the controller board to give a fully-triplicate TMR solution.

These results are sent to the counter results module for analysis, and anomalies sent to the UART.

These values are checked against the true counter value. The counter UART message is in the following format:

CTR_Data Read_Data True_Counter Code_TMR Code_System Time

The codes are contained in Appendix C for a complete description.

3.3.4 BlockRam Circuitry

BlockRam is Xilinx's name for larger memory storage elements. It is effectively chunks of SRAM on the FPGA. The BlockRam controller uses a finite state machine to write patterns of 1's and 0's to two BlockRam modules on the test board. The first BlockRam module simply stores the 8-bit data coming from the controller. The second generates additional parity bits to be stored in the data stream using Hamming code to encode the data for error correction coding. For more on the Hamming code, refer to Chapter 2.12. Figure 11 shows the block diagram for the BlockRam function.

The BlockRam controller is connected to two separate BlockRams. The BlockRams were created using Xilinx's Core Generator. The first was an 8-bit data ram block with 2048 addressable locations. The Hamming encoder and decoder modules, were implemented as shown in paragraph 2.12 to accomplish single bit error correction and double bit error detection. If a single error is found, it is automatically corrected and an appropriate error message generated specifying the bit location of the error. If a double bit upset occurs no correction is accomplished, but an error code is generated for output through the UART to indicate a double event occurred.

A state machine with ten states was implemented to accomplish the controlling functions. A diagram of the state machine is shown in the Figure 12. In State 0, a reset has just occurred and the variables are set to begin the first write pattern. State 1 writes a checkerboard pattern to memory of 1's and 0's.

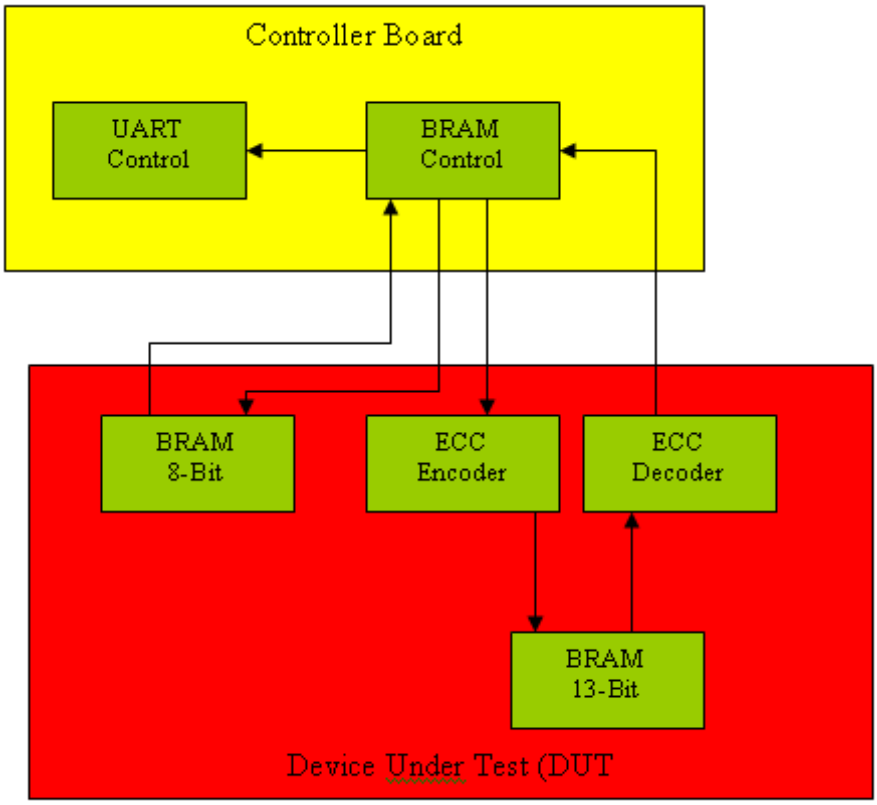

Figure 11 - BlockRam Configuration. There are two BlockRams that are controlled from the Virtex-4 on the controller board. The second BlockRam utilizes a 5-bit Hamming code for single error correction and double error detection.

The user defines, in the Parameters and Constraints file, the number of consecutive addresses to write the 1's and 0's to. This feature is to enable the user to

look for imprinting and other effects associated with clustering of 1's and 0's. The BlockRam modules created using CoreGen also implement an automatic read-back feature.

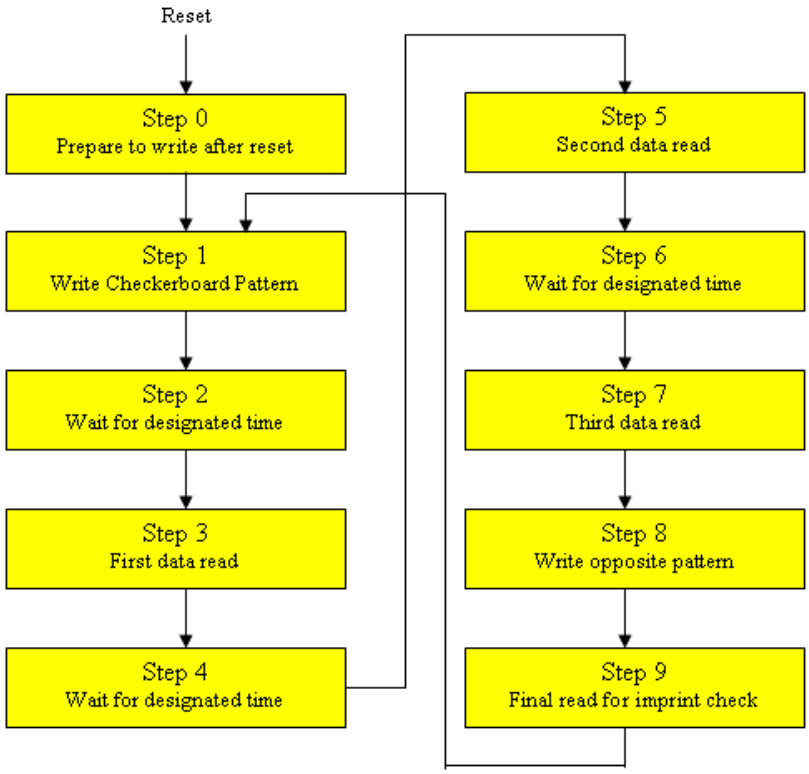

Figure 12 - BlockRAM Controller State machine.

That is, when something is written to memory it is automatically echoed on the read line. In this manner, the data written is verified at the time of write to ensure that what was intended was really written into memory.

State 2 takes another parameter from the Parameter and Constraints file. The user defines a set number of clock cycles to wait. This enables the user to define how long the data is under radiation before reading to verify the contents. These reads can occur just a

few clock cycles after writing, or tens of seconds or longer as desired. The number of cycles in the file is based on the effective clock.

State 3 reads all the addresses and checks them to verify data integrity. Both the 8-bit and 13-bit blockRams are checked simultaneously. This is important, so that hopefully any errors will be shown to be corrected. Given the ECC overhead, it is useful to know how much additional protection is offered in the Virtex-4 FPGA vs using only a stand-alone BlockRam. It would be useful to quantify this for military designers as space and performance can be at a premium. Experimental data would help tremendously in the final cost-performance trade-off.

State 4 performs a second wait, while State 5 performs the second data read. State 6 does a third wait cycle, and State 7 does a third read cycle. State 8 writes the opposing checkerboard pattern to memory. For instance, if memory addresses zero through nine contained “00000000” and addresses ten through nineteen were written with “11111111” on the first write in State 1, State 8 would write “11111111” to addresses zero through nine and “00000000” to addresses ten through nineteen. The purpose of this state is to check for hard imprinting of the address. This write and subsequent read happen very quickly with no wait interval between them. The vast majority of the time under irradiation will be spent with the initial checkerboard pattern. Finally, State 9 does a final read of the data that was just written in State 8. Following this, the controller returns to State 1 to re-write the initial checkerboard pattern, and the entire cycle repeats until the user removes power.

At this point, the user should re-apply power and re-program the device. Since the BlockRams are not initialized to any value, it is unknown what value may be written. Imprinting is the effect that a transistor or element, especially memory, is pre-disposed to be one value consistently over another. This does not mean it cannot be written, but the data that was previously written may be accessible by reading the memory even after power has been removed. To check for imprinting of the device, the BlockRam addresses should be read to see if their contents match what was previously written there following irradiation. Should the data show significant retention, it would be very useful for reverse-engineering or anti-tampering work.

Once again, in order to save serial processing time all of the data is in hex. The BlockRAM UART message is in the following format:

BRM_Address_8-Bit Data_13-Bit Data_ECC Code_System Time

The BlockRAM tables describing the appropriate codes are contained in Appendix D.

3.3.5 UART Printing

The UART print function accepts fully mapped, formatted data streams from each of the three primary controllers. The adder, counter, and BlockRAM controllers convert the values to be printed to hexadecimal, then maps those characters to their ASCII equivalents for output through the UART. A diagram is shown in Figure 13.

This module was much more complicated than first anticipated. The UART controller prioritizes the data for printing by assigning the adder module with first priority followed by the counter. When data is ready, a flag is set, causing the controller to send data to the first finite state machine (FSM). The purpose of the first FSM module

is to keep the same error message from being printed repeatedly over numerous clock cycles due to differences in the effective system clock and the UART clock.

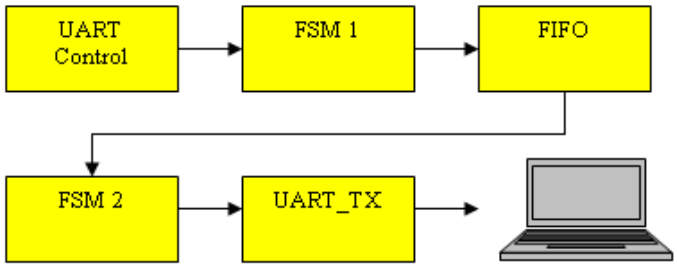

Figure 13 - UART Controller and Printing Modules. Data from the controller goes to the FIFO, and then to the transmitter to be sent to the UART

Suppose the UART clock is running at 100MHz. If the effective clock were 1MHz, there would be one hundred clock cycles where data would be sent from the controller instead of just one. This would result in the same error message being erroneously written numerous times, and overload the buffers. It would also make data analysis more difficult.

3.3.6 Detailed discussion of programmable parameters

One of the goals in this work was to make a test setup that was very extendable and applicable for a wide range of experiments. To accomplish this, vital parameters were made part of a top-level file called Parameters and Constraints. This file contains the parameters required for:

- System Clock
- Effective Clock

- Baud Rate

- Frequency of Adder Stimulus

- Frequency of Counter Stimulus

- Frequency of Auto-reporting current results

- Number of 1's and 0's for BlockRam checkerboard pattern

- Time between BlockRam read and write cycles

3.3.6.1 System Clock

The Parameters_Constraints.vhd file contains the following code for clock ports:

```
inClk_200MHz:   in std_logic; -- Input clocks
inClk_100MHz:   in std_logic;
inClk_10MHz:    in std_logic;
outSystem_Clk:  out std_logic – output system clock
```

There are three main system clock frequencies available, 200MHz, 100MHz, and 10MHz. These are true clocks with proper clock periods of 5 ns, 10 ns, and 100 ns respectively, with half of the period being high and half the period being low. Any of these can be assigned to the outSystem_Clk port, which feeds the clock ports of the adder controller, counter controller, blockRam controller, and UART controller.

3.3.6.2 Effective Clock

By setting the clk_frequency_EffectiveCycles parameter in the Parameter and Constraint file to a certain value, the system will only execute its primary code modules after a set number of cycles. This means that with a 10MHz system clock, a 100KHz effective clock rate can be obtained by setting the value above to 100 (10MHz/100 =

100KHz). The clock period will still be 100 ns, which is why this is effective and not a true clock since a 100 KHz clock would have a true period of 10,000 ns. The system will run at the true system clock speed by setting this parameter to zero. There are constants provided in the Parameter and Constraints file for the user's convenience to automatically calculate these frequencies, however if a custom value is needed, it can simply be added to the constants declaration.

3.3.6.3 Baud Rate

The following lines are contained in the Constraints and Parameters file:

```
outUART_Clk:   out std_logic;
Baud_Calc_16_x_Enable_Cycles: out std_logic_vector (10 downto 0)
```

outUART_Clk is the main UART system clock. This value can be set to any of the main system clocks available in the Constraints and Parameters file.

Baud_Calc_16_x_Enable_Cycles is a constant that is passed to the baud rate divider module. The purpose of this constant is to provide a counter with the proper number of cycles in order to provide a pulse at the right time for UART_TX. Every time one of the pulses from this module is sent, a single serial character is transmitted. Baud_Calc_16_x_Enable_Cycles is calculated as follows:

UART Clock Frequency/16/Desired Baud Rate

Suppose the UART clock is 100MHz and the user wanted to run at 9600 Baud. Then the above parameter would be set to 651. For a 19,200 baud rate, the variable would be set to 325.

9600 Baud: 100MHz/16/9600 = 651

19200 Baud: 100MHz/16/19200 = 325

The appropriate constants were provided in the file for baud rates up to 57,600 using either the 100MHz or 200MHz clock. To modify the baud rate, the user must simply assign one of the following constants to the signal in the Constraints and Parameters file.

3.3.6.4 Frequency of Adder Stimulus

The variable experiment_Adder_Stimulus_Frequency_Cycles controls how many clock cycles occur between stimuli to the adders. The number of counts is based on the effective clock. Suppose the main system clock is 10MHz, and the effective clock is 100 KHz, and the user wanted to provide new inputs 100 times per second. Then the above port would have a num_1_Thousand assigned to it.

100KHz/1000 = 100

3.3.6.5 Frequency of Counter Stimulus

The parameter experiment_Counter_Enable_Frequency_Cycles controls how many clock cycles occur between enable pulses to the counters. The number of counts is based on the effective clock. As in the adders, suppose the main system clock is 10MHz, and the effective clock is 100 KHz, and the user wanted to provide new inputs 100 times per second. Then the above port would have a num_1_Thousand assigned to it.

100KHz/1000 = 100

3.3.6.6 Frequency of Auto-reporting current results

The parameters autoReportStatus_Cycles_Adder and autoReportStatus_Cycles_Counter allow the user to have the software report the current

data being processed at a given, recurring time. The message follows the same format as an error message. This feature is primarily to ensure the circuit is still operating properly, and to automate regular data collection. The number of cycles for the parameters is based on the effective clock. Suppose the system clock is 10MHz and the effective clock is 100KHz, and the user wanted the system to report the current data every 30 seconds. Then the parameter would have assigned to it:

30 * num_100_Thousand

Also implemented is a user, on-demand reporting switch. At any time the user wants data to print to the UART, a switch must simply be hit and the current contents will be printed. Further, a switch to suspend data reporting is also implemented. This was necessary in the case where a cascading failure occurred and filled the buffer. This stops the error from reporting and stifling any other error messages being reported by other modules. Another use for this feature is after an experimental run is accomplished, the other buffers can be emptied by turning off the higher prioritized modules.

3.3.6.7 Number of 1's and 0's for BlockRam checkerboard pattern

The variable num_1_0_Checkerboard allows the user to define the number of alternating 1's and 0's. For instance, suppose the user wanted to have alternating blocks of one hundred 1's and 0's. Then the above parameter would be set to num_1_Hundred. Then, following irradiation it was determined that instead of blocks of 100, every address should alternate. Then the parameter could be easily reset to num_Zero without having to modify the blockRam controller in any way.

3.3.6.8 Time between BlockRam Read and Write Cycles

The parameter blockRam_Read_Cycles allows the user to define how heavily utilized the blockRam modules are. Suppose the user wanted to have the data write, then wait 10 seconds between the subsequent reads. With the same clocking parameters laid out above, the number of cycles would be set to:

10 * num_100_Thousand

In a subsequent or different experiment, the user may want to exercise the BlockRam more heavily. In this case the value could be set to say, num_Ten such that only ten clock cycles would go between reading all the data in the BlockRam modules.

3.4 Mapping of Codes to FPGA Elements and Failure Analysis

The types of circuits and adders that werebuilt for this test were specifically chosen to target certain areas of the FPGA. Diagrams of the Virtex-4 sliceM and sliceL are shown below in Figures 16 and 17, with the fast-carry-look-ahead logic in Figure 18. Using a logical analysis of the data results, one should be able to make reasonable assumptions about which areas of the FPGA or types of elements need to have attention paid to them during design.

The Virtex-4 uses a triple oxide process, where transistors that have to operate faster (CLB transistors, pass transistors) have the thinnest oxides while the I/O pads have the thickest oxides. The configuration bit transistors make up the middle thickness transistors. I expect that more effects will be seen within the CLBs, though Xilinx claims the Virtex-4 FPGAs are immune to latchup. First, the general area of failure can be determined.

The structure of a Virtex-4 FPGA slice [25] is shown in Figure 14 and 15. Each

is stored differently within the slice.

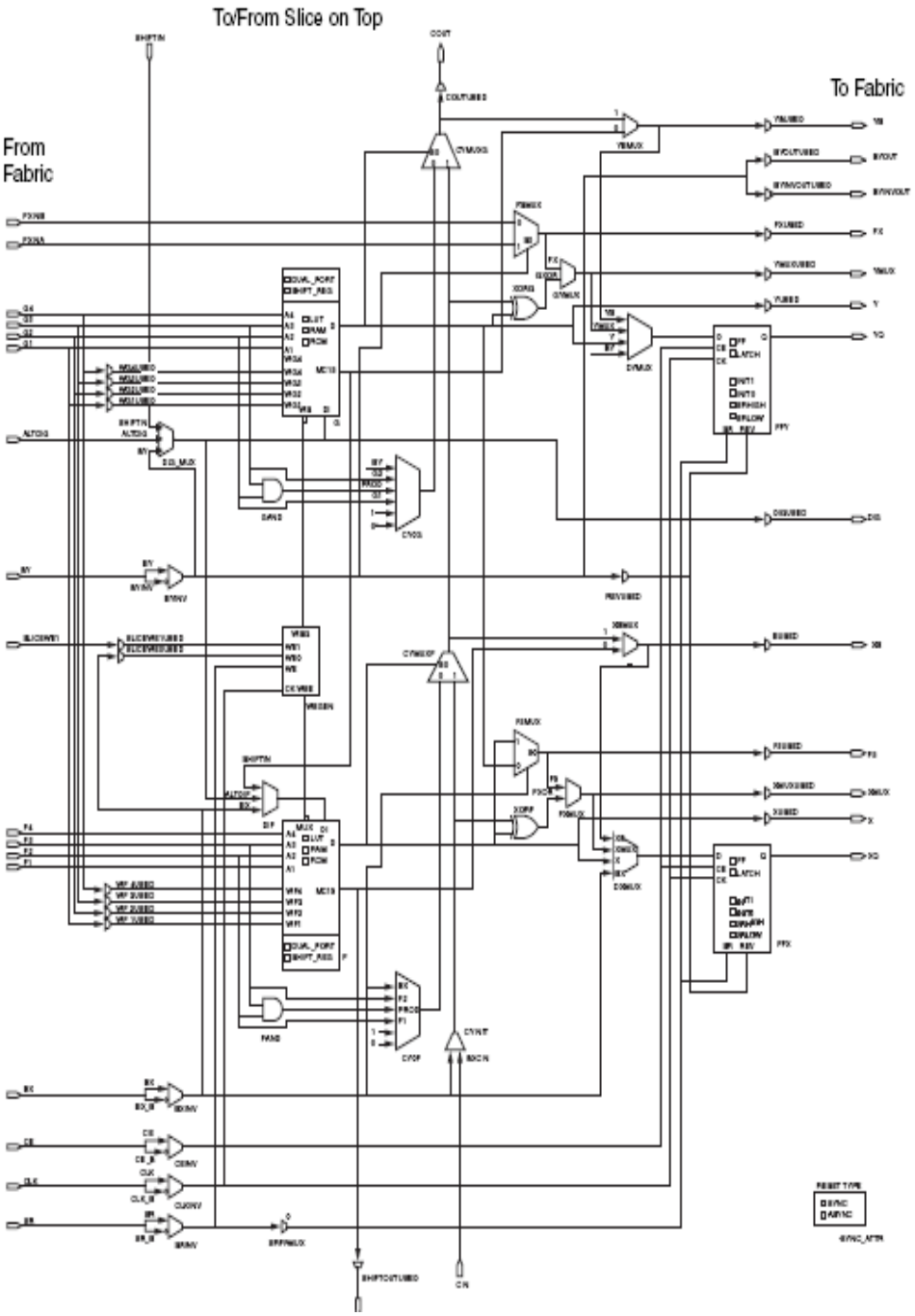

Figure 14 - Virtex-4 Slice M. Slice M can be used for logic or memory [25]

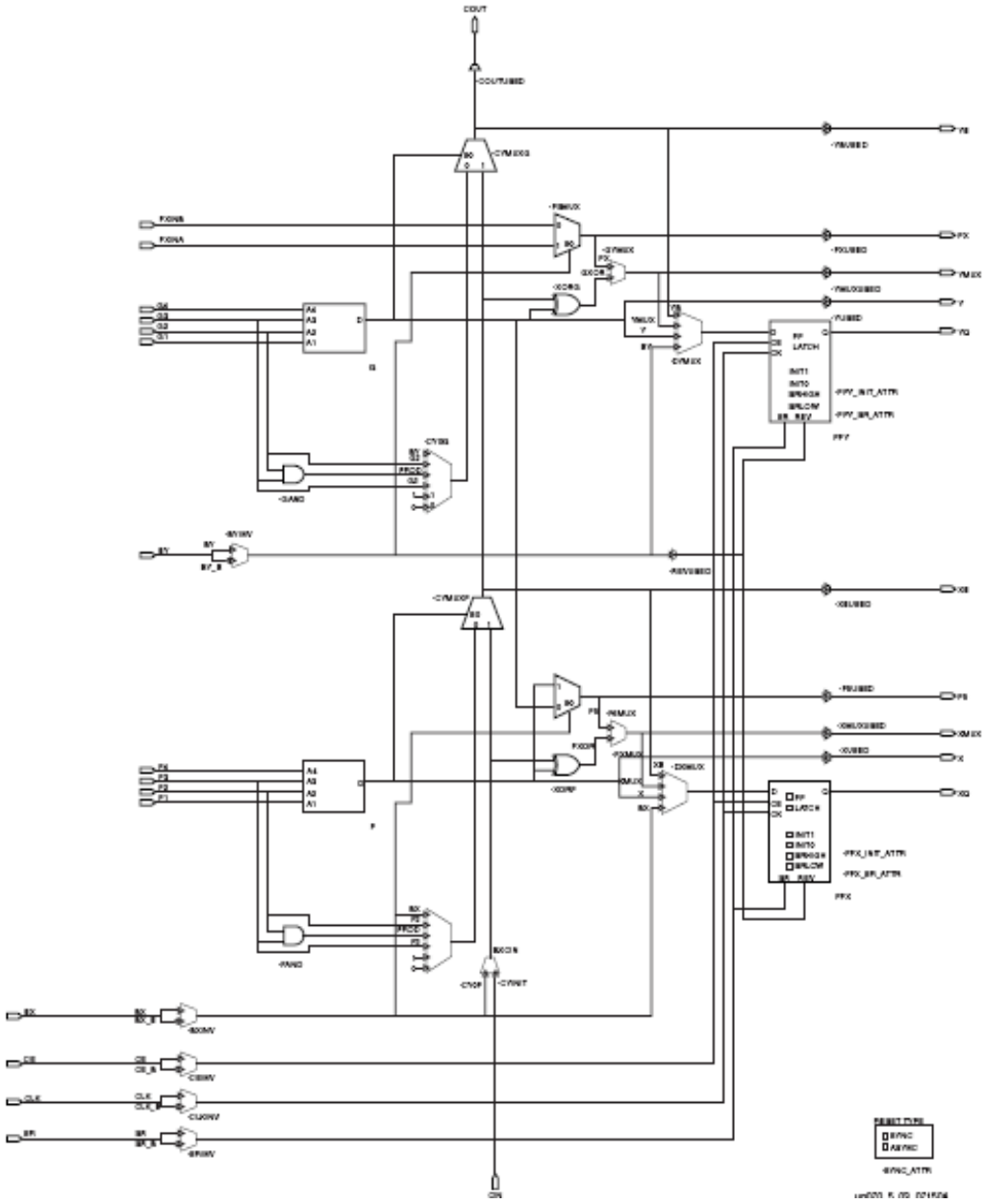

Figure 15 - Virtex-4 Slice L block diagram [25].

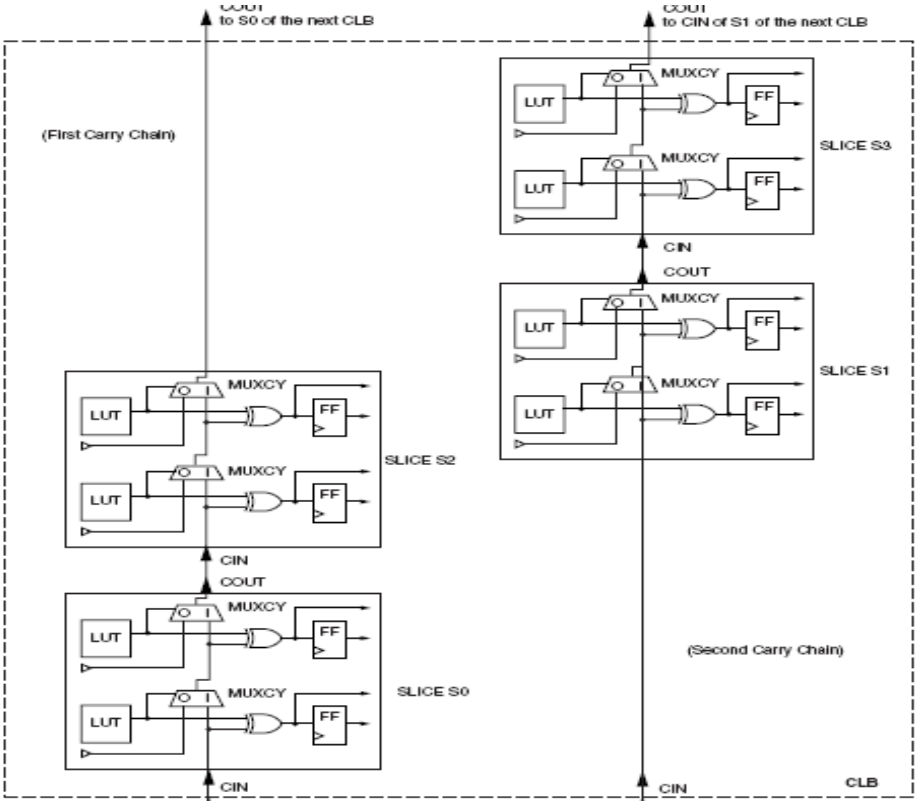

Figure 16 - Virtex-4 CLB Fast-Carry Lookahead logic structure [25]

The ripple carry adder will span multiple CLBs and utilize the carry chain for every stage. The CLA adder will be stored as a Boolean function within the look-up-table. Finally, the behavioral adder is implemented by the synthesizer in the best manner possible, which in this case should be the fast-carry-look-ahead logic within the CLB. This structure is highlighted in Figure 16. Therefore, if one of the adders would tend to show failure before the others, the specific area within the CLB could be identified as an area of concern. Also, since the outputs of the individual adders are also put through voting logic, which is also output, the Adder Controller can compare if the individual outputs match what was received by the FTMR module. This can help isolate errors

occurring in the routing logic and the I/O pads. Tables 7 through 10 summarize what kind of errors point to which areas. The intention is to treat this as any series of experiments. Run the first set to focus which area of interest is likely the problem area. Then, either refine or create new experiments to further isolate subcomponents within the major area of interest.

Table 2 - Summary of adder errors to isolate major area of failure

Case	Individual Adders	FTMR Circuit	FX20 Results Module	Implication
1	Individual Adder has an error	FTMR Module Produces Error Code	Error Read matches FTMR Code Error	Error occurred in CLB
2	Individual Adder has an error	FTMR Module Produces Error Code	No Error Read	Error occurred in switching network when data was passed to FTMR module. Case 4 should also be seen for routing logic to be primary failure area
3	Individual Adder has no error	FTMR Module Produces Error Code	No Error Read	Error occurred in FTMR voting logic
4	Individual Adder has no error	FTMR Module Produces No Error Code	Error Read on FX20	Error occurred in IO pad or routing logic. If same effect is consistently seen, IO pad is most likely area.

Table 3 - Summary of adder errors to isolate area within the CLB

Case	Adder Circuit	Targeted Area	Comment
1.1	Ripple Carry Adder	Carry chains	Error should consistently show on either the 3rd bit or last carry out
1.2	Carry-Look-a-head Adder	Look Up Table	If transient effect, then should only error one time. May also only error on the same inputs

		Configuration Bits	Error should manifest itself repeatedly. Possible if error only occurs for same inputs. Probable if multiple table entries error at the same time
1.3	Behavioral Adder	Arithmetic Logic	Assumes behavioral implimentation uses Fast Carry Look-a-head internal Logic

Table 4 - Summary of counter errors to isolate major failure areas

Case	Individual Counter	FTMR Circuit	Results Module	Implication
1	Individual Counter has an error	FTMR Module Produces Error Code	Error Read matches FTMR Code Error	Error occurred in CLB
2	Individual Counter has an error	FTMR Module Produces Error Code	No Error Read	Error occurred in switching network when data was passed to FTMR module. Case 4 should also be seen for routing logic to be primary failure area
3	Individual Counter has no error	FTMR Module Produces Error Code	No Error Read	Error occurred in FTMR voting logic
4	Individual Counter has no error	FTMR Module Produces No Error Code	Error Read on FX20	Error occurred in IO pad or routing logic. If same effect is consistently seen, IO pad is most likely area.

Table 5 - Summary of counter errors to isolate CLB components

Case	Counter Circuit	Targeted Area	Comment
1.1	UpDown Counter	Flip Flop	Error should consistently show on either the 3rd bit or last carry out
		CLB	Error should consistently occur on the same count
1.2	One Hot Counter	Flip Flop Chain	If transient effect, then should only error one time. May also only error on the same inputs

Suppose an error occurs, and the code for the FTMR module matches what the results module receives on the controller board. This would imply that the error occurred in the CLB, not the I/O pad or routing logic. The other expected options are laid out in the tables. This is not to imply these are all-inclusive, or that multiple kinds of errors will not occur. However, with enough data collected, the logic trail should be easy enough to isolate the problem component. There are other possibilities, such as there is no consistent error. In this case, one has to start looking at common structures to all the major areas. These could include multiplexers and tri-state buffers. In this manner it can be determined what areas need to be hardened to produce a more radiation tolerant design.

3.5 Summary

I have developed my test architecture and setup to isolate potential fault sites within an FPGA. This setup and code structure should work for any type of FPGA, with

the only difference being the clock , FIFO, and BRAM modules that were generated using Xilinx tools would have to be generated using the other manufacturer’s software.

IV. Analysis and Results

4.1 Chapter Overview

To validate the design and functionality of my setup in a legitimate environment, two FPGA boards were obtained to test during irradiation. The limited number did not impact the ability to demonstrate that the setup can be adjusted to run over a broad range of conditions while in a radiation environment. I successfully irradiated the FPGA boards, and showed the setup worked as expected and that data was collected as intended.

4.2 Test Strategy

To accomplish testing, the design was validated at the nuclear reactor using a gamma cell with a Co-60 source. The test unit was first put at the height corresponding to the highest exposure, since it could not be assumed that sufficient dose would be absorbed to see errors. As soon as glitches were reported, the board would be pulled out, and tested outside of the radiation chamber. The unit was baselined for normal operation prior to testing. Depending on the timeframe involved in seeing a problem, the test board would be irradiated at a lower dose rate. In doing this, the goal was to gather data over several runs and varying dose rates using just the single board. Also of interest was any compound effects due to multiple exposures to gamma radiation. This approach did pose the risk that permanent damage could occur on the first test, thereby eliminating any chance to do additional data collection.

4.3 Experiment Setup

The DUT was exposed in the gamma cell at the maximum dose rate of 86 krad (tissue)/hr, which is roughly equivalent to the equivalent dose rate for silicon. Pictures

showing the actual configuration are shown below. I operated the adder stimulus at both 1Hz and 10Hz, with no issues. The counter operated at 1Hz. The UART operated at 100 MHz and 9600 Baud. The system clock was set to 10MHz and the effective clock was varied from 100KHz to 1KHz by stepping down by powers of ten.

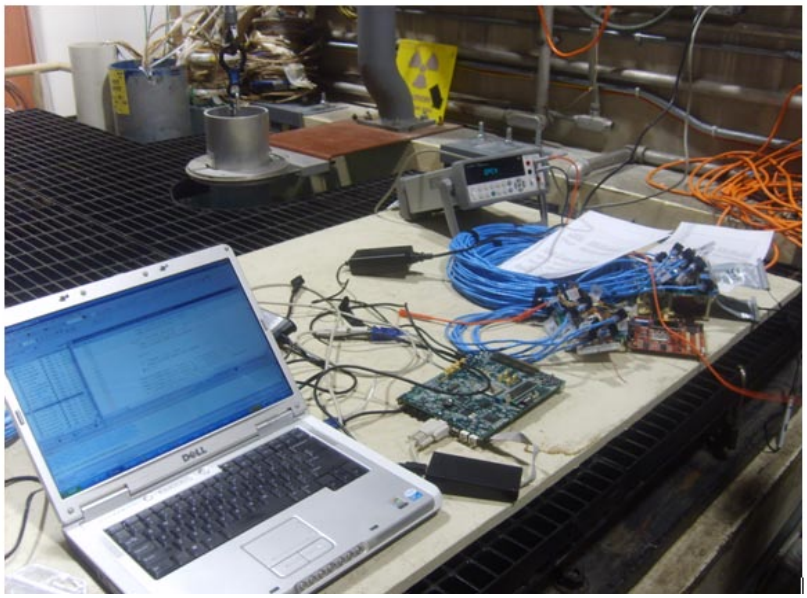

Figure 17 - Photo showing setup at reactor site

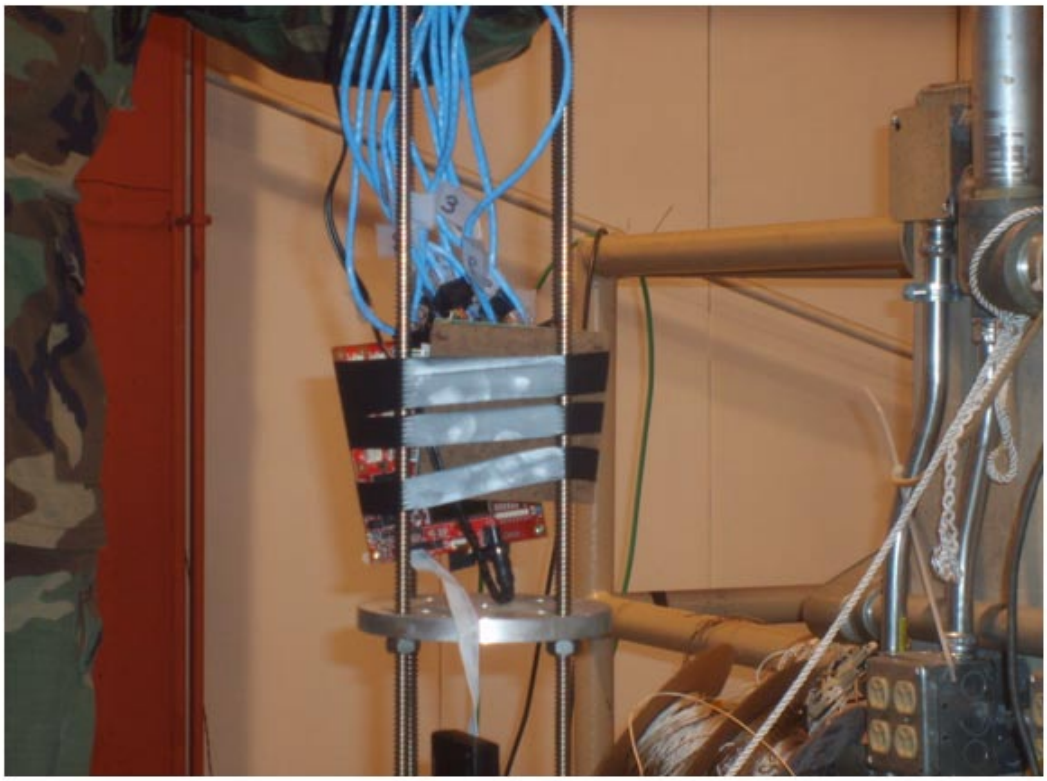

Figure 18 - Test board secured in the gamma cell elevator

I experienced several issues upon arriving, including a coding error that needed to be corrected, and problems with the UART port on the monitoring laptop. I attempted to measure the current, only to find that I could not get the ammeter between the DC converter and the board. I had intended to monitor this to look for power effects. Should the unit have drawn more current while irradiating, it would have indicated that either the power MOSFET was being affected, or possibly latchup. Multiple runs could give a better idea what was happening.

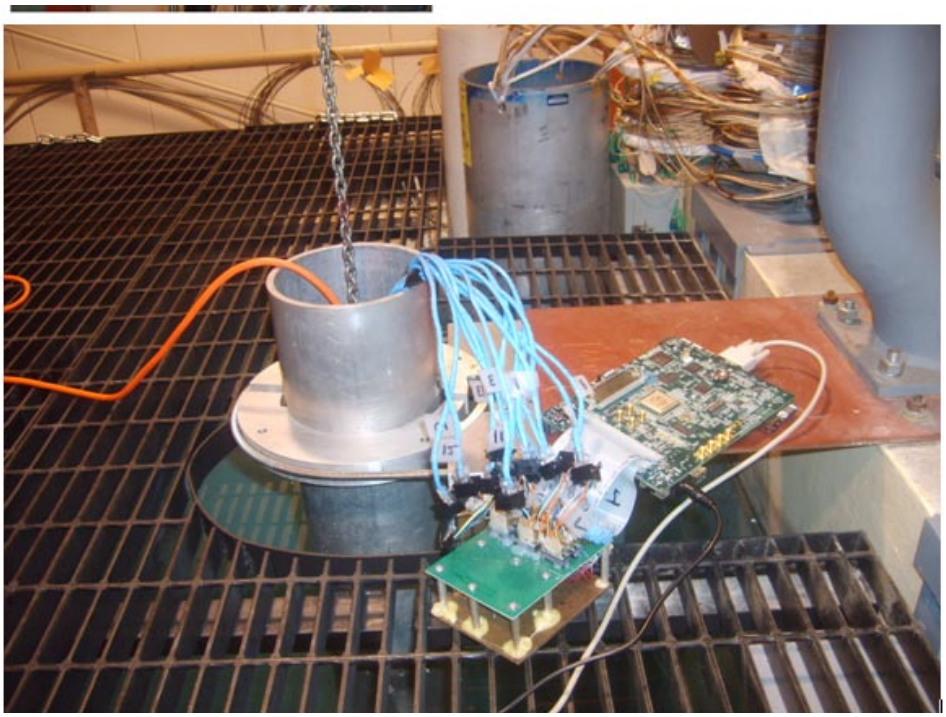

Figure 19 - Test board in the gamma cell under radiation

4.4 Results of Experimental Testing

The FPGA lost power after 00:59:32 (HH:MM:SS). The unit operated as intended over a fifteen foot cable run, though in the future some buffering of data may improve reliability. All parameters were validated while being irradiated, though the results were not as useful since the DUT failed due to power anomalies before any upsets or glitches were detected by the controllers. Upon removing the board from the chamber, the board clearly still had power since certain LEDs were illuminated. However, when attempting to re-program the FPGA immediately upon removal from the gamma cell, the FPGA was not detected by the software, indicating the FPGA was not powered.

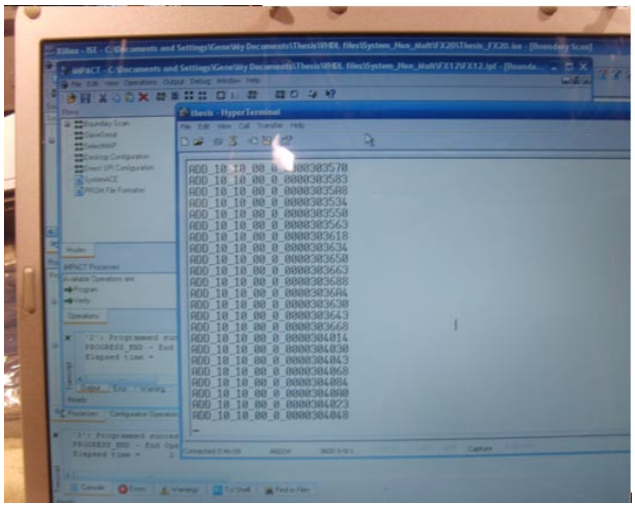

Figure 20 - Adder controller data during irradiation. Data Indicates normal operation.

The next day the board was programmed again. This time the FPGA programmed and ran correctly, but the external temperature of the chip was 35 degrees higher than prior to irradiation. This indicates two important things; first, more power was being consumed due to increased power dissipation, and second, that the effects of radiation self-annealed during the night. Further experimentation is required to more precisely determine what element of the FPGA, be it the CLBs, routing network, or the power MOSFET. The first indication would seem to be a problem with the power MOSFET, though this could easily turn out to be incorrect.

One effect that was noticed involved the clock issues mentioned before. Just stepping down the clock does not guarantee the ability of the counters on the test board to interpret the clock pulse as a valid pulse. Over shorter distances of a few feet no problems were seen, but over longer cable lengths the counter would show enough clock skew that the counter would start to count slower than the counter module anticipated, resulting in errors. Normally this occurred after a consistent period of time, making it

possible to look at the timing of the error to determine if it is a result of clock skew or a legitimate error. It is possible that clock skew could increase during irradiation, which would be exhibited by the consistency of the timing of errors changing to shorter time periods. This was not checked during experimentation.

This can be mitigated in several different ways. First, additional clock cycles can be inserted between when the stimulus is provided and when the result is analyzed for correctness. Second, the signal can be amplified with a buffer to increase the strength and make sure the clock pulse is read. This will also result in additional clock pulses needed between when the stimulus is applied and when read and verified.

4.5 Analysis of Range of Experiments

One of the primary goals in this work was to make a platform that would take one of the most significant variables between experiments out of play. The experimental setup and data collection can make a real "apples to apples" comparison of data quite tricky, and to this point no setup was readily available that multiple users could utilize to directly compare experimental data. The system ran as expected after repeatedly varying all aspects of the Parameters and Constraints file, resulting in a much expanded capability for a broad range of experiments. Some examples of these are as follows:

Clocking

- Differences between effective and true clocking
- Differences between true clocks
- Clock skew

Stimulus

- Differences in time between stimuli

BlockRam

- Susceptibility to SEUs
- Differences in clustering of 1's and 0's
- Differences in time values are allowed to sit in memory
- Susceptibility to data imprinting

Radiation testing has the potential to show very different effects depending on how these parameters are varied together. It is also possible that different areas may exhibit failure based on the varied values of the parameters. For instance, under high frequency counting, the clock may skew enough such that the flip flop fails. Under slower clocking conditions, possibly the error would occur first in the switch network or I/O pad.

A secondary application for the BlockRam controller is in the use of SRAM testing. The controller can be very easily modified to add the additional one or two control signals if required, but the patterning is still applicable for good comparison testing from SRAM to BlockRam, BlockRam to BlockRam, or SRAM to SRAM. It will also provide a stable setup to once again compare apples to apples across multiple chips, including radiation hard chips.

Finally, the code for this setup is completely portable to any other model of Xilinx FPGA, and any other manufacturer's FPGA. A couple of core-generated modules would have to be created under the software of that particular manufacturer. The only requirement is for enough pin connections to adequately pass data back and forth. This is important so that not only FPGAs with different technology sizes can be tested against

the same benchmark, but different manufacturer's FPGAs can also be measured against that same benchmark. One would reasonably expect FPGAs with different feature sizes to respond quite differently to a radiation environment. Further, different manufacturers lay out their circuits in completely different manners, resulting in quite probably a very different response to the same stimulus under testing. This flexibility also extends to SRAM chips.

Important too is the feature that new controller modules can easily be integrated into the controller chip. So for instance, one could write a DRAM or SDRAM controller that followed a similar methodology to the BRAM/SRAM controller. Previous work has identified that DRAM does have failure when put in a radiation environment, but no data was provided that could assist in defining where and how the error occurred, and more importantly how to mitigate it. This will be discussed in further detail in Chapter 5.

Finally, the cost of the system is a few hundred dollars. With the laptop, hardware, and FPGA boards, the total system cost less than $1000. Given that the laptop, controller board, and break-out-boxes can be re-used, the recurring cost of the system is approximately $400 for the specific board tested in this work. Other test boards may cost more or less, but the test setup effectively takes the place of very expensive logic analyzers, function generators, and monitoring devices. Additionally, the data parsing accomplished by the controller modules processes the vast majority of raw data that must be eliminated to find the few relevant and significant facts.

V. Conclusions and Recommendations

5.1 Chapter Overview

This work has provided a robust methodology for targeting areas of SRAM-based FPGAs, as well as a low-cost, versatile, and expandable test setup that can be re-used for many types of experiments on numerous types of chips. The setup has been utilized for experimentation, and validated that all data reporting systems functioned as expected. The system is easily modified and capable gathering the relevant data that can assist researchers isolate areas of weakness in the devices under test.

5.2 Conclusions of Research

Data was obtained that showed a loss of power to the FPGA under test before any errors were logged. However, by repeating the experiment with a larger sample size and varying the dose rate it is very possible errors will be seen in the areas being targeted.

5.3 Significance of Research

This work will allow future researchers to eliminate six months or more of software writing and hardware building. Using this apparatus, future experiments should be able to isolate vulnerable parts of FPGAs and other chips, over multiple technology generation and multiple manufacturers, using less equipment and requiring fewer hours of data analysis. Additional controllers for other types of memory such as SRAM, MRAM, DRAM, etc can be written and easily integrated without interfering with the controllers and reporting system already in place. This contribution will be invaluable to future users, as this will provide a common baseline over which to compare experimental

results, as well as common data collection, which typically makes it very difficult to compare data when different test setups and data parameters are used.

5.4 Recommendations for Action

Some immediate enhancements that are recommended are the placing of an additional parameter in the counter module to allow for automatic reset after a specified period of time. This should reduce errors related to cable length and clock skew.

In addition, the addition of a FIFO buffer for each of the individual controllers will allow more data to be buffered in preparation for sending to the PC. Currently, the buffer size is limited to five, and is stored in the UART controller. This will make the UART control truly independent, and even more easily facilitate adding additional controller modules for other types of memory and chips. The UART can then be used to simply define the print priority of the FIFOs, and send the control signals to notify when data is ready to print.

5.5 Recommendations for Future Research

5.5.1 DRAM

Many areas of future research can be investigated using this hardware and software suite as the initial starting point. The BlockRAM controller especially, should be expanded with interrupt logic to make it suitable for DRAM. This would be very relevant given past work by Capt Eric Lam and Maj David Arnold on DRAM memories. The data collection capability of this system can give great insight into the cause and location of the effects they observed in their simple testing setup.

5.5.2 SRAM

The BlockRam controller is extremely easy to adjust, if any adjustment is required at all, to use for SRAM testing. At most, an additional control signal may be required and a change in the addressing to account for more memory locations.

5.5.3 Magnetic RAM (MRAM)

MRAM needs a lot of experimentation performed. This can be easily accomplished using the established principles for data collection and control I have established in this work. A new controller should be written for this to properly interface with the chip. Using Chapter 3 of this thesis, a person with reasonable VHDL skill and basic background in controllers and radiation effects should be easily able to define the critical data parameters and program the controller to provide the required stimulus and report the results.

5.5.4 FPGAs from Multiple Technology Generations

This would require no modification of the test setup and controllers as they are currently written. Sufficient I/O capability and a form factor that will allow the DUT to fit in the chamber are the only requirements that must be carefully matched.

5.5.5 FPGAs from Multiple Manufacturers

Moving to multiple manufacturers would have the same requirements as above.

5.5.6 Radiation Effects on the Clock

In the current setup, the clock for the board under test is driven from the controller board. It would be interesting if the results of experiments were different if the clock was driven from the board being irradiated. Should different results be seen, it would quantify the susceptibility of the clock to inducing errors in the system. Since virtually

every major system is synchronous clock effects can have a dramatic effect on circuit reliability. This modification is easily accomplished by associating all clocking and synchronous logic on the board with the 100MHz clock resident on every Xilinx FPGA board.

5.5.7 Irradiation Using Different Sources

This test used gamma radiation from a Co-60 source. Other sources could be utilized to look for differing radiation effects. An electron gun can hit very specific areas with high energy electrons. However, some things have to be taken into account. The heat generated by the electrons necessitates the use of a nitrogen-cooled cold finger. In addition, there is a potentially large flux of X-rays given off with sufficiently high electron energies that could potentially corrupt the controller board making it more difficult to determine where an error occurred.

Neutron radiation can be used to test chips and boards for displacement damage, but neutron irradiation is made more difficult due to the fact that the elements under test will become activated. This includes the test structure the chip is mounted in, making it more challenging to do multiple tests with different kinds of stimuli.

5.6 Summary

In summary, the work demonstrated in this thesis a provides a greatly enhanced testing methodology, structure, and setup that will be of great use for current and future researchers. This platform will have a common data reporting methodology, and a common baseline with which to do true “apples to apples” comparison. This will enable future experimenters to spend a majority of their time on experimentation instead of test

setup construction. This work provides for an excellent jumping-off point for follow-on researchers to make a greater impact in future radiation experimentation and space systems reliability. In the past, very little has been passed from researcher to researcher at AFIT, such that each new student has had to start virtually from scratch. Hopefully this will not be the case in the future enabling the program to be much more relevant in the radiation effects and space systems world.

Appendix A: Hamming Code Example

This work used a Hamming code based on the following parity scheme. Suppose the following data word was to be stored:

(D7 : D0) = 01001110.

To implement double error detection and single error correction, a total of thirteen bits will be required. The original bits are ordered as follows:

P4	D7	D6	D5	P3	D4	D3	D2	D1	P2	D0	P1	P0
	0	1	0		0	1	1	1		0		

The parity equations for encoding are calculated as follows:

P0 = D0 xor D1 xor D3 xor D4 xor D6

P1 = D0 xor D2 xor D3 xor D5 xor D6

P2 = D1 xor D2 xor D3 xor D7

P3 = D4 xor D5 xor D6 xor D7

P4 = P0 xor P1 xor D0 xor P2 xor D1 xor D2 xor D3 xor D4 xor P3 xor D5 xor D6 xor D7

The final parity bit is what allows for double error detection. If only single error correction were desired, then the final parity bit could be eliminated. The complete 13-bit encoded data is shown below.

P4	D7	D6	D5	P3	D4	D3	D2	D1	P2	D0	P1	P0
0	0	1	0	**1**	0	1	1	1	**1**	0	**1**	**1**

Upon decoding, five error bits are calculated by XOR-ing the original equations for the parity bits with their respective parity bit.

Error 0 = P0 xor D0 xor D1 xor D3 xor D4 xor D6

Error 1 = P1 xor D0 xor D2 xor D3 xor D5 xor D6

Error 2 = P2 xor D1 xor D2 xor D3 xor D7

Error 3 = P3 xor D4 xor D5 xor D6 xor D7

Error 4 = P4 xor P0 xor P1 xor D0 xor P2 xor D1 xor

D2 xor D3 xor D4 xor P3 xor D5 xor D6 xor D7

The error code is then translated using this relationships in Table 6.

Table 6 - Error codes following initial parity bit check to decode Hamming - encoded data

Error Code		**Implication**
00000		No Error
00001		P0 Incorrect
00010		P1 Incorrect
00011		D0 Incorrect
00100		P2 Incorrect
00101		D1 Incorrect
00110		D2 Incorrect
00111		D3 Incorrect
01000		P3 Incorrect
01001		D4 Incorrect
01010		D5 Incorrect
01011		D6 Incorrect
01100		D7 Incorrect
01101		Double Bit Error

Upon detection of a code other than 00000, the appropriate bit is inverted before being sent to the output. This scheme is very simple, yet powerful for protecting memory, registers, register files, etc. It is important to note that for the scheme to work correctly, bits that are protected or checked by the same parity bits should not be placed next to each other in memory. This could result in a Multiple Bit Upset (MBU) event that will defeat the parity check and have the system put out an incorrect value without detecting that an error occurred.

Appendix B: Adder Code Tables and Examples

This appendix provides an example utilizing the methodology to determine the failure mechanism in the FPGA. The adder data code is in the following format:

ADD_Data Read_Data True_Adder Code_TMR Code_System Time

The codes are in hexadecimal format to save processing time. Also, each TMR voting logic has a 2-bit code indicating whether any of the inputs did not match. These four bits are combined such that the FTMR code is the higher order bits and the TMR code comprises the lower order bits. Take the following example for an error in the combinational logic block.

Suppose the following code is reported by the test structure.

ADD_05_07_02_F_0000426AF1

The data read in binary is 00101. The true data is 00111. This indicates that the second bit has flipped from a 1 to a 0. The adder status code of 02 indicates the ripple carry adder produced the wrong data. In addition, the FTMR and TMR modules reported the correct result, and indicated a code of 11 for both the FTMR and TMR voting logics. This is also the code for the ripple carry adder to show that both voting logics received the incorrect input from the CLB where the result is processed. Taken together, this should guide the experimenter to the CLB as the failure location.

A second example will show a failure in the routing logic. Suppose the following code was received:

ADD_05_05_08_8_0000426AF1

Notice that the read result and the true result match. This indicates a failure was detected by the voting logic and corrected it. Next, the adder status code of 08 tells the

experimenter the FTMR voting logic reported this result code. Also, the TMR code of 8, or 1000 in binary, shows that the TMR voting logic reported no error, while the FTMR voting logic found a mismatch with the carry-look-ahead (CLA) adder. Since the adder status code reported no error in the CLA adder result, the TMR voting logic confirmed no error, and the FTMR reported a mismatch, the error most likely occurred in the switch logic prior to the result reaching the FTMR voting logic.

A similar methodology, using the tables in Appendices B through D, can be used to isolate potential failure points. In many cases, a cause may not be directly attributable to a single point, and instead trends must be analyzed to make the best determination for the most likely area of interest. Finally, additional experiments may need to be developed to provide more refined data for the researcher.

Appendix C: Counter Code Tables

Table 7 - Counter Status Codes

Comment	Counter Code
No Error	0
UpDown 1 Counter	1
One Hot Counter	2
OneHot, UpDown1 Counters	3
UpDown 2 Counter	4
UpDown 2, UpDown 1 Counter	5
UpDown 2, OneHot Counters	6
UpDown 2, OneHot, UpDown1 Counters	7
FTMR Counter	8
FTMR, UpDown1 Counters	9
FTMR, OneHot Counters	0A
FTMR, UpDown1, OneHot Counters	0B
FTMR, UpDown 2 Counters	0C
FTMR, UpDown2, UpDown 1 Counters	0D
FTMR, UpDown 2, OneHot Counters	0E
FTMR, UpDown2, OneHot, UpDown1 Counters	0F
TMR Counter	10
TMR, UpDown1 Counters	11
TMR, OneHot Counters	12
TMR, OneHot, UpDown1 Counters	13
TMR, UpDown2 Counters	14
TMR, UpDown2, UpDown1 Counters	15
TMR, UpDown 2, OneHot Counters	16
TMR, UpDown 2, OneHot, UpDown 1 Counters	17
TMR, FTMR Counters	18
TMR, FTMR, UpDown 1 Counters	19
TMR, FTMR,OneHot Counters	1A
TMR, FTMR, OneHot, UpDown 1 Counters	1B
TMR, FTMR, UpDown 2 Counters	1C
TMR, FTMR, UpDown 2, UpDown 1 Counters	1D
TMR, FTMR, UpDown 2, OneHot Counters	1E
TMR, FTMR, UpDown 2, OneHot, UpDown 1 Counters	1F

Table 8 - Counter TMR status codes

Comment	Counter TMR Code
No Mismatch	00
UpDown Counter 1	01
One Hot Counter	10
UpDown Cunter 2	11

Appendix D: BlockRam Code Tables

Table 9 - BlockRam Error Correction Code Indicators

Comment	ECC Code
Bit Zero Incorrect	11
Bit One Incorrect	12
Bit Two Incorrect	13
Bit Three Incorrect	14
Bit Four Incorrect	16
Bit Five Incorrect	17
Bit Six Incorrect	18
Bit Seven Incorrect	1A

Bibliography

[1] Alles, M, "Process Technology and Hardening", *2007 IEEE Nuclear and Space Radiation Effects Conference Short Course Notebook*. I-3 – I-66. New York: IEEE Press, 2007.

[2] Avnet Inc Design Services. Xilinx Virtex-4 FX12 Evaluation Kit Configuration Reference Manual ADS-005200, April 2006.

[3] Avnet Inc Design Services. Xilinx Virtex-4 FX12 Evaluation Board ADS-XLX-V4FX12EVL-SCH, February 2005.

[4] Avnet Inc Design Services. Xilinx Virtex-4 FX12 Evaluation Kit User Guide ADS-005204, April 2006.

[5] Berg, Melanie, "Fault Tolerance Implementation within SRAM-Based FPGA Designs based upon the Increased Level of Single Event Upset Susceptibility", *Proceedings of the 12the IEEE Internationa On-Line Testing Symposium*, (2006).

[6] Betz, V, Rose J, "How Much Logic Should Go Into an FPGA Logic Block",

[7] Betz, V, Rose J, "Cluster-Based Logic Blocks for FPGAs: Area Efficiency vs. Input Sharing and Size",

[8] Chapman, K, "200MHz UART with Internal 16-Byte Buffer," Xilinx XAPP223, June 2001.

[9] Chapman, K, "UART Transmitter and Receiver Macros," Xilinx Brief, January 2003.

[10] Dressendorfer, P, "Basic Mechanisms for the New Millennium", *1998 IEEE Nuclear and Space Radiation Effects Conference Short Course Notebook*. III-1 – III-117. New York: IEEE Press, 1998.

[11] Garrett, Henry, and others, "Radiation Environment Within Satellites," *1993 IEEE Nuclear and Space Radiation Effects Conference Short Course Notebook*. II-1 – II-160. New York: IEEE Press, 1993.

[12] Holmes-Siedle, Andrew, Adams, Len. *Handbook of Radiaiton Effects, 2nd Edition*. Oxford: Oxford University Press, 2002.

[13] Kastensmidt, F, "SEE Mitigation Strategies for Digital Circuit Design Applicable to ASIC and FPGAs," *2007 IEEE Nuclear and Space Radiation Effects Conference Short Course Notebook*. II-1 – II-84. New York: IEEE Press, 2007.

[14] Ladbury, R, "Radiation Hardening at the System Level," *2007 IEEE Nuclear and Space Radiation Effects Conference Short Course Notebook*. IV-1 – IV-94. New York: IEEE Press, 2007.

[15] Neamen, Donald A. *Semiconductor Physics & Devices, 2nd edition*. New York: McGraw-Hill, 1997.

[16] Peterson E, Single Effect Analysis and Prediction, *1997 IEEE Nuclear and Space Radiation Effects Conference Short Course Notebook*. III-1 – III-160. New York: IEEE Press, 1997.

[17] Petrosky, James. *Radiation Effects on Electonic Devices: Theory, Modeling and Experiment*.

[18] Pratt, Brian, and others, "Improving FPGA Design Robustness with Partial TMR"

[19] Rose, J, R. J. Francis, D. Lewis and P. Chow, "Architecture of Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid State Circuits*, 1217 – 1225 (Oct. 1990).

[20] Samudrala, P, "Selective Triple Modular Redundancy for SEU Mitigation in FPGAs",

[21] Schwank, J, "Basic Mechanisms of Radiation Effects in the Natural Space Environment", *1994 IEEE Nuclear and Space Radiation Effects Conference Short Course Notebook*. II-1 – II-109. New York: IEEE Press, 1994.

[22] Sze, S.M., Kwok, K. *Physics of Semiconductor Devices, 3rd Edition*. Hoboken: John Wiley & Sons, Inc. 2007.

[23] Telikepalli, A, "Power-Performance Inflection at 90 nm Process Node – FPGAs in Focus," *Chip Design Magazine*. https://www.chipdesignmag.com/print.php?articleId=261?issueId=0

[24] Xilinx. Schematic, ML405 Evaluation Platform Block Diagram,

[25] Xilinx. Virtex-4 User Guide UG070, March 2006.

[26] Xilinx. Virtex-4 Packaging and Pinout Specification UG075, September 2005

[27] Xilinx. Virtex-4 Data Sheet DC and Switching Characteristics DS302, September 2006

[28] Xilinx. Virtex-4 Configuration Guide UG071, September 2006

[29] Xilinx. "Xilinx FPGAs Overcome the Side Effects of Sub-90 nm Technology" WP256, March 2007

Vita

Capt Eugene Hockenberry was born in Camp Hill, PA. After graduating from Red Land High School, he went to Virginia Tech, where he received a Bachelor of Science degree in Computer Engineering. Capt Hockenberry was commissioned in May of 2000 to the United States Air Force. His first assignment was at Wright Patterson AFB, OH, where he worked as a Systems Engineer and Lead Engineer at Aeronautical Systems Center's (ASC) Training Systems Product Group (TSPG) on the C-141, AWACS, F-15, and F-16 programs. Following this, Capt Hockenberry was re-assigned to Kirtland AFB, NM, as mission manager for the Constellation Observing System for Meteorology, Ionosphere, and Climate (COSMIC) rocket launch program. Capt Hockenberry was accepted to the Air Force Institute of Technology (AFIT) in 2006, where he pursued a Masters of Science degree in Electrical Engineering.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
19-06-2008	Masters Thesis	August 2006 – June 2008

4. TITLE AND SUBTITLE

HARDWARE, SOFTWARE AND DATA ANALYSIS TECHNIQUES FOR SRAM-BASED FIELD PROGRAMMABLE GATE ARRAY CIRCUITS

5a. CONTRACT NUMBER N/A
5b. GRANT NUMBER N/A
5c. PROGRAM ELEMENT NUMBER N/A

6. AUTHOR(S)

Hockenberry, Eugene B, Captain, USAF

5d. PROJECT NUMBER N/A
5e. TASK NUMBER N/A
5f. WORK UNIT NUMBER N/A

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/08-11

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr James Lyke
AFRL/RVSE, Bldg 914
3550 Aberdeen Ave. SE,
Kirtland AFB, NM 87117-5776
James.lyke@kirtland.af.mil
DSN 246-5812, Comm 505-846-5812

10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/VSSE

11. SPONSOR/MONITOR'S REPORT NUMBER(S) N/A

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
None

14. ABSTRACT

The main objective of this research is to accomplish two objectives; first, develop a robust test methodology to successfully allow researchers to isolate errors occurring in SRAM-based FPGAs and other memory devices. Second, provide a test platform capable of exercising this methodology. Several requirements were made known for this platform. The system should be compact, versatile, flexible, and affordable.

To meet these objectives, a failure analysis tree was derived such that various combinations of circuits on a test FPGA, with subsequent data analysis being accomplished on a controller FPGA board. Using the data derived from various types of adder and counter circuits, major subsystems of the FPGA could be targeted. Next, a test structure was built using a controlling FPGA board and a laptop for data reporting. The controller board provided stimuli to the device under test (DUT), accepted the data produced, and automatically analyzed the data to generate data messages.

Additionally, Xilinx BlockRam modules were created to test their susceptibility to logic errors in a radiation environment. Two reliability-enhancing techniques were implemented for evaluation; triple modular redundancy (TMR) and error correction coding (ECC).

15. SUBJECT TERMS

Field Programmable Gate Array; radiation environment; test structure; Virtex-4; triple modular redundancy; error correction coding.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kim, Yong
U	U	U	UU	85	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, ext 4620 (yong.kim@afit.edu)