Measuring Soft Error Sensitivity of FPGA Soft Processor Designs Using Fault Injection

Nathan Arthur Harward
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Electrical and Computer Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/5699

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Measuring Soft Error Sensitivity of FPGA Soft Processor Designs Using Fault Injection

Nathan Arthur Harward

A thesis submitted to the faculty of Brigham Young University in partial fulfillment of the requirements for the degree of Master of Science

Michael J. Wirthlin, Chair
Doran K. Wilde
James K. Archibald

Department of Electrical and Computer Engineering Brigham Young University March 2016

Copyright © 2016 Nathan Arthur Harward All Rights Reserved
Increasingly, soft processors are being considered for use within FPGA-based reliable computing systems. In an environment in which radiation is a concern, such as space, the logic and routing (configuration memory) of soft processors are sensitive to radiation effects, including single event upsets (SEUs). Thus, effective tools are needed to evaluate and estimate how sensitive the configuration memories of soft processors are in high-radiation environments.

A high-speed FPGA fault injection system and methodology were created using the Xilinx Radiation Test Consortium’s (XRTC’s) Virtex-5 radiation test hardware to conduct exhaustive tests of the SEU sensitivity of a design within an FPGA’s configuration memory. This tool was used to show that the sensitivity of the configuration memory of a soft processor depends on several variables, including its microarchitecture, its customizations and features, and the software instructions that are executed.

The fault injection experiments described in this thesis were performed on five different soft processors, i.e., MicroBlaze, LEON3, Arm Cortex-M0 DesignStart, OpenRISC 1200, and PicoBlaze. Emphasis was placed on characterizing the sensitivity of the MicroBlaze soft processor and the dependence of the sensitivity on various modifications. Seven benchmarks were executed through the various experiments and used to determine the SEU sensitivity of the soft processor’s configuration memory to the instructions that were executed. In this thesis, a wide variety of soft processor fault injection results are presented to show the differences in sensitivity between multiple soft processors and the software they run.

Keywords: FPGA, reliability, fault injection, soft processor, softcore processor, MicroBlaze, LEON3, Cortex-M0, OpenRISC, PicoBlaze, Xilinx, Virtex-5, XRTC, XRTC-V5FI
ACKNOWLEDGMENTS

I would like to share my appreciation for my research advisor, Dr. Michael Wirthlin, for giving me an excellent research opportunity and guiding me through the exploration and writing of this thesis work and multiple papers. He has been very patient and encouraging. I also wish to thank Drs. Doran Wilde and James Archibald for being continually supportive and for their service on my thesis committee.

I wish to recognize research assistants who have participated in building the fault injector used for this research: Luke Hsiao, Shaelyn Meyer, and Jeremy Butler. I also would like to thank Jon-Paul Anderson for his helpful feedback and advice for this research and Aaron Stoddard for suggestions for improving my thesis draft. I thank and recognize a fellow researcher, collaborator, and friend, Michael Gardiner, for providing several soft processor designs and benchmarks that were used for various experiments in this research.

I am very grateful for the love and patience of my wife Caitlin, and my two daughters Jeri, and Isla, who allowed me to pursue my educational goals. I also thank my parents who fueled my fascination with electronics and discovery.

I also acknowledge the Xilinx Radiation Test Consortium (XRTC), Gary Swift, and other consortium members for insight and for providing our lab with the use of XRTC hardware. This work was funded by members of the NSF Center for High-Performance Reconfigurable Computing (CHREC), an I/UCRC Program of the National Science Foundation under Grant No. 1265957.
# TABLE OF CONTENTS

| LIST OF TABLES | vi |
| LIST OF FIGURES | vii |

## Chapter 1  Introduction

### Chapter 2  FPGA Radiation Testing and Infrastructure

2.1 Radiation Sensitivity of Digital Circuits and FPGAs

2.2 XRTC Test Infrastructure

   2.2.1 Xilinx Radiation Test Consortium
   2.2.2 XRTC Hardware System
   2.2.3 External Connections

2.3 Previous Experiments Performed with XRTC Hardware

## Chapter 3  Fault Injection System Based on XRTC Radiation Test Hardware

3.1 Background on Fault Injection Systems

   3.1.1 Background on Emulation-Based FPGA Fault Injection
   3.1.2 Previous FPGA Fault Injection Systems

3.2 Fault Injection on the XRTC Platform

   3.2.1 Benefits of Using XRTC Hardware for Fault Injection
   3.2.2 Supporting Fault Injection on XRTC Hardware

3.3 Fault Injection Methodology

   3.3.1 Sequence for an Individual Injection of a Bit Fault
   3.3.2 Running a Fault Injection Campaign
   3.3.3 Variations in Methodology

3.4 Comparison with Previous Works on Fault Injection

## Chapter 4  MicroBlaze Soft Error Sensitivity Experiments

4.1 Soft Processor Background

4.2 Background on the MicroBlaze Soft Processor

4.3 MicroBlaze Design Sensitivity

   4.3.1 MicroBlaze Raw Sensitivity Results
   4.3.2 Placement Dependencies
   4.3.3 Experiments with Additional MicroBlaze Variations

4.4 MicroBlaze Software Sensitivity

   4.4.1 Overview of Benchmarks Used
   4.4.2 Results of Benchmark Experiments
   4.4.3 Sensitivity Cost of an Unutilized FPU

## Chapter 5  Fault Injection Experiments on Multiple Soft Processors

5.1 Processor Examples
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Sensitive Bits Measured over Multiple Runs of a Single MicroBlaze Design</td>
<td>31</td>
</tr>
<tr>
<td>4.2</td>
<td>Configuration Bits Common to Multiple Runs</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>Variation in Placement and Routing</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Sensitivity Results and Utilization for MicroBlaze with and without LUTRAM Primitives</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Sensitivity Results and Utilization for MicroBlaze with Different Memory Sizes</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Execution Time Dependence on MicroBlaze and Towers of Hanoi Results</td>
<td>37</td>
</tr>
<tr>
<td>4.7</td>
<td>All Benchmark Software Programs</td>
<td>40</td>
</tr>
<tr>
<td>4.8</td>
<td>MicroBlaze FPU Comparison for Integer Benchmarks</td>
<td>42</td>
</tr>
<tr>
<td>5.1</td>
<td>Utilization for Minimalistic Soft Processor Design Implementations</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>The Sensitivity Results for Minimalist Soft Processor Design Implementations</td>
<td>47</td>
</tr>
<tr>
<td>5.3</td>
<td>Reset Recovery Results for Various Minimalistic Soft Processors</td>
<td>49</td>
</tr>
<tr>
<td>5.4</td>
<td>Resource Utilization for Performance Processor Design Implementations</td>
<td>50</td>
</tr>
<tr>
<td>5.5</td>
<td>Sensitivity Results for Performance Processor Designs Implementations Running on Multiple Benchmarks</td>
<td>51</td>
</tr>
<tr>
<td>5.6</td>
<td>Normalized Sensitivity Results for Performance Processor Designs Implementations Running on Multiple Benchmarks</td>
<td>53</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Configuration Memory</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Resulting Circuit</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Example of an SEU Effect on Routing</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td>Example of an SEU Effect on Logic</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>Picture of (a) XRTC motherboard, (b) V5QV daughtercard, and (c) PROM memory card</td>
<td>9</td>
</tr>
<tr>
<td>2.6</td>
<td>Picture of XRTC BrainBox</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>High Level View of Fault Injector Components without the BrainBox</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Both processors are placed on the DUT FPGA. Their outputs are compared with each other on the FuncMon service FPGA.</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Diagram Showing the Fault Injection Procedure</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>The placement and route layout images of each MicroBlaze where placement was generated from random seeds (a) 22, (b) 49, and (c) 75, and (d) using an area constraint.</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Sensitive bits and size in slices for different MicroBlaze placements</td>
<td>34</td>
</tr>
<tr>
<td>4.3</td>
<td>Placement Screenshot for MicroBlaze with (a) and without (b) LUTRAM Primitives</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>Placement Screenshot for MicroBlaze (a) with and (b) without Added Memory</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>Execution Time Dependence on MicroBlaze and Towers of Hanoi Results</td>
<td>38</td>
</tr>
<tr>
<td>4.6</td>
<td>MicroBlaze FPU Comparison for Integer Benchmarks</td>
<td>41</td>
</tr>
<tr>
<td>4.7</td>
<td>Placement Screenshot for MicroBlaze (a) with and (b) without FPU</td>
<td>42</td>
</tr>
<tr>
<td>5.1</td>
<td>Placement Screenshots for Minimalistic Soft Process in Top Left to Bottom Right Order: MicroBlaze, LEON 3, ARM Cortex-M0, OpenRISC, and PicoBlaze</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>Minimalistic Soft Processor Sensitive Bits and Normalized Sensitivity</td>
<td>48</td>
</tr>
<tr>
<td>5.3</td>
<td>Sensitivity Results for Performance Processor Designs Implementations Running on Multiple Benchmarks (Grouped by Processor)</td>
<td>51</td>
</tr>
<tr>
<td>5.4</td>
<td>Sensitivity Results for Performance Processor Designs Implementations Running on Multiple Benchmarks (Grouped by Benchmark)</td>
<td>52</td>
</tr>
<tr>
<td>5.5</td>
<td>Normalized Sensitivity Results for Performance Processor Designs Implementations Running on Multiple Benchmarks (Grouped by Processor)</td>
<td>54</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION

Aerospace applications require the use of microprocessors in harsh radiation environments, such as space. Examples of this use are telecommunication, media broadcasting, and intelligence gathering performed using processors in satellites. In addition, microprocessors are needed for control, telemetry, and processing sensor data for space exploration.

Microelectronic devices that operate in these radiation environments are susceptible to upsets within their internal flip-flop circuits and memory cells. When energized ions collide with sensitive logic regions within a microelectronic device, the state of the circuit can be changed [1].

Microprocessors must be designed to operate in high-radiation environments [1]–[3]. Traditionally, processors are protected from the effects of radiation by either a special fabrication process called radiation hardening [4] or by radiation-hardened by design (RHBD) techniques [5]. These techniques successfully reduce the sensitivity of the devices to the effects of radiation, but microprocessors protected by these methods are larger, slower, and more expensive than commercial off-the-shelf (COTS) microprocessors [6].

Currently, the use of soft processors is another computing approach that is being investigated for use in harsh radiation environments; these processors are implemented in field programmable gate array (FPGA) fabric. Reliability techniques for protecting FPGA-based systems are well understood, and designs can benefit from the ability to modify soft processor hardware and implement logic to help detect and mask errors caused by radiation effects. Protecting the device against radiation becomes even more important with an FPGA device than for application-specific integrated circuit (ASIC) designs, since single event upsets (SEUs) can affect both the structure of a design and its data [7].

Static random access memory (SRAM) is used to hold the FPGA configuration memory in SRAM-based FPGAs, and an SEU in a configuration memory bit can affect the logic and routing of a soft processor design on the FPGA. One method for determining the FPGA’s tolerance to
radiation is radiation testing. In a radiation test, the FPGA is bombarded with ionized particles to simulate a radiation environment. This testing is useful when preparing FPGA designs for use in harsh environments, but it can be prohibitive when multiple designs or design variations must be compared.

The reliability of soft processors when SEUs occur is not well understood, and the current research is limited. To better understand the sensitivity of soft processors to SEUs, many experiments must be performed, and ‘emulated fault injection’ is a method in which multiple tests can be conducted quickly. Fault injection provides a means of evaluating and comparing the radiation sensitivities of soft processor designs by testing each memory bit for its sensitivity to radiation effects.

Presented in this thesis is a high-speed FPGA fault injection system and methodology that was developed using the Xilinx Radiation Test Consortium’s (XRTC’s) Virtex-5 radiation test hardware to exhaustively test the sensitivity to SEU of a soft processor within an FPGA’s configuration memory. In this thesis, the experiments performed with this system are described to show that the sensitivity of the configuration memory of a soft processor depends on its microarchitecture, its hardware features, and the software instructions that are executed. The soft processor’s fault injection results are shown and analyzed.

One of the contributions of this work is a novel FPGA fault injection platform that can exhaustively test all configuration bits at a high rate and provide the flexibility required to deal with the challenges of soft processor fault injection. The fault injection platform also can be used to experiment with radiation-hardened FPGAs, since it was built using hardware that can be used in a radiation test.

This work also provided experimental results that aid in understanding and designing less-sensitive soft processor systems and that aid in choosing a soft processor for a computer system intended for use in harsh environments. This thesis also presents original research that provided the sensitivity impact of software benchmarks on soft processors using fault injection results.

The contributions of the research described in this thesis are included in two full peer-reviewed papers. One paper is a chapter in a book entitled, “FPGAs and Parallel Architectures for Aerospace Applications” [8], and the other paper was presented at the IEEE’s 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) [9].
The thesis consists of six chapters. Chapter 2 provides background information on circuit reliability, FPGA radiation testing, and the XRTC radiation test infrastructure that was used to build a fault injector. Chapter 3 provides background information on fault injection and describes the fault injection system that was built for this research. Chapter 4 presents a set of MicroBlaze-specific soft processor fault injection experiments and the results that were obtained. Chapter 5 presents additional fault injection results for experiments using a variety of soft processor architectures. Chapter 6 presents the conclusions based on the experimental results.
CHAPTER 2. FPGA RADIATION TESTING AND INFRASTRUCTURE

The fault injection system created for this work uses the Xilinx Radiation Test Consortium (XRTC) radiation test hardware as a platform. Originally, the XRTC radiation test hardware was developed to evaluate reconfigurable devices under radiation environments because of a growing interest in using field-programmable gate arrays (FPGAs) for aerospace applications. This chapter provides the necessary background material on FPGAs and radiation effects and gives an overview of the XRTC radiation test hardware in order to understand how and why it is used for the fault injection platform.

2.1 Radiation Sensitivity of Digital Circuits and FPGAs

This work is motivated by the need to use microelectronics in high-radiation environments such as required for space applications and the nuclear industry. Given the costs and high stakes associated with nuclear and aerospace applications, it is necessary to measure and understand system vulnerabilities in these environments.

Microelectronic devices operating in space and other high-radiation environments are susceptible to upsets within their internal memory cells. Energized ions colliding with sensitive logic regions within a microelectronic device can change the state of the circuit [1]. When a collision event modifies the state of a memory bit or flip-flop, this is known as a soft error or an SEU.

This SEU occurrence especially affects SRAM-based FPGAs. FPGAs are integrated circuits that contain programmable logic blocks. SRAM FPGAs use SRAM to hold the FPGA configuration, which is the memory that defines all the programmable look-up tables (LUTs), registers, routing channels, and other programmable circuits. Modern FPGAs contain a very large amount of internal configuration memory (upwards of a billion bits for modern FPGAs [10]). Figure 2.1 shows the configuration memory that defines a simple circuit within an FPGA (Figure 2.2).
During operation, a single bit experiencing an upset could potentially cause a functional failure in the user design. Figures 2.3 and 2.4 depict how upsets in the configuration memory bits can affect operation of the user design by modifying the function of a LUT (Figure 2.4) or the routing between nodes (Figure 2.3), causing the user design to malfunction. Figure 2.3 shows how...
an SEU in the routing region of an FPGA can disconnect a route and leave an input to the LUT open. Figure 2.4 demonstrates how an SEU in the LUT may change the logic of the circuit by altering the LUT to behave like an XOR gate instead of an AND gate.

These configuration memory upsets can be detected with techniques like readback and bit-stream cyclic redundancy check (CRC) [11]. Upsets in the configuration memory can be repaired from their upset state by using techniques such as configuration scrubbing [12] and additional error detection and correction (EDAC) techniques [13]. The effects of configuration upsets can be mitigated by employing fault-tolerant design techniques such as triple modular redundancy (TMR) [14]–[16], or temporal majority voting (TMV) [17]. There is an active research community investigating techniques for tolerating temporary upsets within an FPGA. It can require many resources and special hardware in order to perform experiments on FPGA reliability. The next section will introduce an organization and test infrastructure that were created for this purpose.
2.2 XRTC Test Infrastructure

Specialized hardware is necessary to perform reliability experiments on FPGAs, evaluate FPGA design sensitivity, and measure the effectiveness of reliability techniques. The test hardware needs to have the ability to reconfigure the design under test (DUT) FPGA and read back the configuration data. FPGAs have a large number of pins that can be used as inputs and outputs for the designs that run on them. It may require an additional FPGA to provide input stimulus and capture outputs for a high number of pins. The hardware might support a variety of DUT FPGAs for evaluation which would require an interface for DUT daughtercards to connect to the test hardware. The resulting hardware required for this process could involve multiple FPGAs and be very complicated.

Because specialized test hardware is used as a test fixture and not as an end product, the costs of engineering and manufacturing a test fixture can be very prohibitive. To absorb these costs, an organization consisting of manufacturers and researchers can collaborate to create a test infrastructure that can be used by multiple researchers. This section shows an example of such a collaboration where a test infrastructure was made, describes their test hardware, and provides some examples of research conducted using this hardware.
2.2.1 Xilinx Radiation Test Consortium

The test infrastructure used for the experiments in this thesis was created by the Xilinx Radiation Test Consortium (XRTC) which is a commercial consortium originally organized by Xilinx Inc. (FPGA Manufacturer) and Jet Propulsion Laboratory (JPL). The XRTC combines voluntary researchers from universities, government laboratories, and companies in the aerospace industry to evaluate FPGAs for use in aerospace applications.

XRTC efforts include characterizing the Xilinx FPGA sensitivity to radiation environments, testing SEU mitigation techniques (e.g., scrubbing and TMR), and testing specific FPGA features and blocks. They release reports of their findings [18] and their members present their findings at workshops and other meetings. Most of their research experiments are performed with particle accelerators for radiation testing.

2.2.2 XRTC Hardware System

The XRTC created a special hardware test platform to facilitate radiation testing of various Xilinx FPGAs. The Xilinx Radiation test hardware was created as a test fixture for radiation beam testing and for other test-related activities. It was built to support multiple FPGAs and a wide variety of FPGA-related experiments. The main components are shown in Figure 2.5 and include the XRTC motherboard (Figure 2.5a), a DUT FPGA daughtercard (Figure 2.5b), and the non-volatile programmable read-only memory (PROM) card (Figure 2.5c). A piece of hardware called the BrainBox (Figure 2.6) also plays a role in many experiments.

The motherboard was designed by SEAKR Engineering and contains two FPGAs: the ConfigMon and the FuncMon (hidden by the daughtercard in Figure 2.5). The ConfigMon FPGA is responsible for the high-speed memory configuration and scrubbing of the design operating on the daughtercard, and the FuncMon FPGA is responsible for monitoring the functionality of the design while undergoing testing. The FuncMon can also optionally provide design stimulus, clock, and resets. The XRTC motherboard is designed to operate with any number of test daughtercards to support radiation testing and fault injection for a variety of FPGAs. The daughtercard used in these experiments contains a single Virtex-5QV (V5QV) FPGA. The V5QV FPGA is seen on the top of the daughtercard in Figure 2.5. The package lid has been removed and the die is visible to support
radiation testing with heavy ions. This device is a 65-nm RHBD FPGA manufactured by Xilinx that has been qualified for space applications [19]. The V5QV is comparable to the commercial Xilinx V5 FX130T and contains 20,480 logic slices, 320 DSP48Es, 10,728Kb of Block RAM (BRAM), and 20 GTX transceivers.

The PROM memory card contains a golden copy of the DUT FPGA bitstream and is attached locally to support high-speed, parallel configuration through the SelectMAP configuration port. The daughtercard contains a single FPGA that is programmed with the DUT. The test design data is held on a PROM card which is a non-volatile memory storage extension plugged directly into the motherboard. This card contains both the DUT bitstream file and the DUT mask file. The mask file is used to differentiate configuration bits that are used for function and routing from the configurable logic block (CLB) bits used for shift register LUTs (SRLs) and look-up table RAM (LUTRAM).
The BrainBox (Figure 2.6) provides an interface between a service FPGA (ConfigMon or FuncMon) and the host computer. On the motherboard, there are several connections between the components. The ConfigMon connects to the DUT SelectMAP pins and has an 8-bit wide SelectMAP data port. The ConfigMon and FuncMon FPGAs have a 16-bit wide common interconnect bus (CI-Bus) to share status and send commands with each other. The inputs and outputs of the DUT include a 145-pin single-ended bus, a 47-pin TMV bus, and a 20-pair differential bus. The motherboard relies on communication to and control from an external PC. These connections and software will be described in the next section.

2.2.3 External Connections

There are three Joint Test Action Group (JTAG) chains that are used for to configure the system’s FPGAs. The JTAG chain has the FuncMon, ConfigMon, and four PROM memory chips that can hold bitstreams for automatic loading of the two service FPGAs. The PROM card is programmable via JTAG, and the DUT can optionally be programmed via JTAG. An experiment may use the ConfigMon to program the DUT with a bitstream saved on the PROM card. There are three RS232 ports that can be used for serial communication directly to a PC, one for each FPGA
(ConfigMon, FuncMon, and DUT). The motherboard and daughtercard each have an multi-gigabit transceiver (MGT) which is a high-speed serial communication interface.

The BrainBoxes connect to the motherboard with wide parallel ports and connect to the host computers via universal serial bus (USB). Older BrainBoxes connected to the computer via a parallel port. The ConfigMon has a communication protocol between it and the BrainBox. The FuncMon similarly communicates with the BrainBox. Some researchers implement their own communication connectors and/or designs for ConfigMon. The BrainBox has a USB to universal asynchronous receiver/transmitter (UART) chip on board, so that when the BrainBox is connected to the host PC, it will have a COM port for serial communication.

2.3 Previous Experiments Performed with XRTC Hardware

The XRTC test infrastructure has been used for a variety of experiments ranging from static tests [20]–[22] on FPGA parts, to evaluating mitigation techniques on FPGA designs [23]–[25] and testing reliability of data transmission protocols [26]. The XRTC test hardware has evolved over a space of more than ten years — it has been used heavily for the testing of three space-grade Xilinx FPGAs: the Virtex-II QV [20], the Virtex-4 QV [21], and the Virtex-5 QV [22].

The XRTC radiation test hardware has also been used to study FPGAs that are using a combination of configuration memory scrubbing and TMR [24] techniques using an older test fixture for Virtex-4 radiation testing and fault injection. Radiation testing of scrubbing techniques has also been performed on the Virtex-5 [27]. TMR has been radiation tested on the Virtex-4 in [23]–[25] and also tested with Virtex-4 fault injection in [25]. Harding et al. used the XRTC infrastructure to set up an experiment to validate improvements to the high-speed Aurora protocol [26].

This chapter presented a radiation test infrastructure used for conducting FPGA reliability experiments. The experiments that are possible with this hardware can enable researchers to understand how FPGAs operate in harsh space environments. In addition to radiation testing, this hardware is also used for the fault injection used in this research. The fault injection system and methodologies used in this research are described in Chapter 3.
An FPGA fault injection system is a valuable research tool for identifying which bits in the FPGA’s configuration memory are sensitive to soft errors. Fault injection can be used to characterize a soft processor’s configuration memory reliability and to compare the sensitivity of multiple soft processor designs. With some modifications, the XRTC radiation test hardware is well-suited for use as a base for the FPGA fault injector used in this research.

This chapter describes the fault injection system used to perform the experiments in this thesis work. First, background information is provided for readers to understand how fault injection works and to know what other fault injection systems have been created. Second, the modifications made to the XRTC radiation test hardware are described that create the XRTC Virtex-5 Fault Injector (XRTC-V5FI) system. Third, the fault injection methodology is explained and the sequence of commands and events that are used to perform fault injection are described. Finally, the XRTC fault injection system is compared with other fault injection platforms.

3.1 Background on Fault Injection Systems

One of the contributions of this thesis work is to evaluate and compare multiple soft processor configurations and architectures. Because emulated FPGA fault injection can be used to determine which FPGA SRAM configuration memory bits are sensitive to SEUs, it is an excellent method to use for performing these soft processor experiments, but it is not the only method available for experimenting with soft processor design reliability. This section will show several methods of evaluating circuit reliability, provide background on fault injection, and review previous fault injectors. Subsequent sections will discuss the fault injection system built for this research.
Other methods for determining design sensitivity exist and they all require the application of SEU detection techniques (mentioned in Section 2.1) which are used to identify SEUs for logging. Radiation testing was introduced in Section 2.2. This method has a circuit placed in an artificial radiation environment to measure its tolerance to radiation upsets. Radiation testing can use a variety of energized particle beams and resembles a real radiation environment more closely than other testing methods, but it does not perfectly imitate a real space environment. Radiation testing provides a wide array of radiation effects and is not limited to just SEUs in configuration memory. Radiation testing can be expensive, require travel and can be impractical when a designer needs to quickly compare a large variety of FPGA designs and techniques.

When FPGAs are placed in real environments, methods are applied to track SEUs. This was done for the Cibola Flight Experiment Satellite (CFESaT) [28]. The CFESaT detected errors with a CRC on each frame in the configuration bitstream. Detected SEUs are logged and then repaired via memory scrubbing. The CFESaT additionally ran experiments to evaluate other SEU detection techniques. The CHREC Space Processor (CSP) [29] is another FPGA-based computer system that will evaluate next generation space computing systems in a real radiation environment. Real environment experiments can very accurately show how a design operates in a space environment, but can be a very prohibitive method of experimentation.

Another technique for evaluating design reliability is through simulated fault injection. Simulated fault injection uses a software-based simulation model to force faults into a design netlist and check the operation of the circuit for correct behavior. This technique offers a considerable amount of control over the insertion of the fault and allows for a wide variety of faults that can be simulated (e.g. stuck-at faults, bridging faults, state change etc.). This technique, however, is slow and can be difficult to use on large complex designs like large processor designs.

Radiation testing, real environment experimentation, and simulated fault injection can be powerful methods of evaluating microchips and circuit designs, but each have their own strengths and limitations. Multiple methodologies have been used to complement and validate each other’s results [25], [30], [31]. For certain kinds of experimentation, emulated FPGA fault injection can be an excellent and convenient alternative to many of these methods, especially when assessing the SEU sensitivity of a user design, and comparing to the SEU sensitivity of multiple designs. Emulated fault injection is not a complete replacement for these other methods, and other methods
may be desired when additional radiation effects should to be observed, including failures outside
the configuration memory. Emulated fault injection experimental results are very useful for testing
and comparing soft processors (work done as part of this thesis). The following subsection will
describe emulated fault injection in more detail.

3.1.1 Background on Emulation-Based FPGA Fault Injection

With an emulation-based fault injection system, faults are introduced into the device being
tested. It is faster than simulation and it does not require a detailed software model of the device
being tested. Emulated fault injection is less flexible than simulation because there are limited
number of possible upsets that can be emulated. Emulated fault injection is very beneficial on
FPGAs because the internal structures and state designs held on the FPGA fabric are defined in the
FPGA’s configuration memory and the configuration memory can be modified through an external
interface like SelectMAP or JTAG.

With an emulation-based FPGA fault injection, individual configuration bits in a design
can be tested for SEU sensitivity. To determine which bits are sensitive, individual configuration
bits are toggled and restored one by one while the design outputs are compared with the outputs
from a correct (golden) model or set of expected outputs. When an output mismatch is observed,
the fault injection system detects and logs the changed bit as sensitive. After determining which
configuration bits are sensitive, one can estimate the reliability of a design in greater detail than is
possible with simple modeling techniques.

A modification to an FPGA configuration memory bit may affect a net, a flip-flop, or a
memory bit within user design. While the design is running, the error can propagate and alter
the state of other parts of the design and corrupt the function of the design as a whole. The error
can also be isolated to a part of the logic that is not being used by the design and might leave
the function of the whole design unaffected. A design’s sensitivity can depend on how general
or specific the design is to a specific application. If the majority of the design’s logic is used in
calculating the output result, it can be more sensitive than a soft processor design where more
components may have less activity during execution.

Emulated FPGA fault injection can vary by method. The DUT can be duplicated where
only one design undergoes fault injection and the outputs can be compared with each other. The
DUT can also have a golden model counterpart, or a list of expect outputs that it can be compared with. Design outputs can be selected from nets and buses within the design, or they can be limited to the design’s external outputs. Inputs to the DUT can be randomly chosen, or they can be defined as a list of inputs. Some designs require only a clock and a reset input, like a soft processor design that has been loaded already with instruction memory to run on. One final methodology variation involves the selection of configuration bits to inject. The fault injection campaign can exhaustively inject all configuration memory bits. This may take too much time for a fault injection system so a campaign may choose only the bits that are expected to be significant to a design’s implementation. Bits can also be chosen randomly, or randomly within a location, and then statistical methods can be applied to estimated the design’s sensitivity.

There are several metrics used in this research to compare design sensitivity. After a fault injection campaign is over, the number of bits that are sensitive \((\text{Sensitive Bits})\) and the number of bits injected \((\text{Total Bits})\) will be known. The first metric is the design Sensitivity. This is calculated by dividing the count of Sensitive Bits by the number of Total Bits. The second metric used is the design Utilization which is calculated by dividing the number of Slices Used by the Total Slices on an FPGA. The third metric is the design’s Normalized Sensitivity. Normalized Sensitivity represents the percent of a design that is detected as sensitive. This is calculated by dividing design Sensitivity by the design Utilization. The normalized sensitivity equation can be customized for designs using this device by setting the device parameters. If the device’s Total Slices was 20,480 and the Total Bits was 34,087,072 like they are for the Virtex-5QV, the normalized sensitivity equation would be:

\[
\text{Normalized Sensitivity} = \frac{\text{Sensitive Bits}}{34,087,072} \times \frac{20,480}{\text{Design Slices}} \tag{3.1}
\]

\[
= 6.01 \times 10^{-4} \frac{\text{Sensitive Bits}}{\text{Design Slices}}. \tag{3.2}
\]
3.1.2 Previous FPGA Fault Injection Systems

The need for reliable FPGAs in space environments has motivated the development of FPGA fault injection platforms [13]. Over the years, many notable fault injection tools and platforms were created [32], [33], [15]. Each successive implementation made improvements in either speed, portability, or accuracy.

Johnson et al. used the Los Alamos National Laboratories (LANL) SLAAC-1V testbed which housed three Virtex (XCV1000) FPGAs. Two of these FPGAs were loaded with identical designs and sent identical clock and circuit inputs. Their outputs were compared in the third FPGA while SEUs were injected into one of the first two FPGAs to see if the fault injection resulted in erroneous output. Although the SLAAC-1V injector was able to test all configuration bits at high speeds, it could not be utilized with newer FPGA families because the design used a custom board [34].

Alderighi et al. took a different approach with the FLIPPER fault-injection platform. The FLIPPER used a single Virtex 2 Pro (XC2VP20) motherboard that could connect to a daughterboard. The FLIPPER used hardware that could also be used for radiation tests and featured customizable fault injection campaigns. However, rather than testing all configuration bits, the FLIPPER used partial re-configuration in randomly chosen configuration memory locations to apply stimuli. It would iterate this procedure, accumulating bit flips until a functional fault was detected before logging the fault. Although the FLIPPER was fast and more user friendly, it did not test all of the configuration bits, nor did it test bits individually [35].

In 2007, Sterpone et al. proposed yet another approach that used a single board that held a Virtex II-Pro (XCV2P30) FPGA with an embedded PowerPC microprocessor. The FPGA used its internal configuration access port (ICAP), and a timing unit which drove the clock and reset signals. It connected the FPGA, PowerPC, timing unit, and the ICAP to the same On-chip Peripheral Bus. With these connections, the PowerPC was responsible for fault injection and collecting results. Because all of these components were on-chip, this platform operated at very high speeds. However, because the fault injection solution uses a design internal to the test FPGA, the design limits the ability to test large designs or standalone systems, and the test vectors are limited to the resources available on the FPGA [36].
In more recent years, Cieslewski et al. at the University of Florida, have worked to improve fault injector portability with their Simple Portable FPGA Fault Injector (SPFFI) [37]. In their approach, the popular JTAG interface is used to program the FPGAs. They have also compensated for the speed bottleneck of JTAG by designing SPFFI to only fault inject bits that are representative of a region of interest and/or fault inject random locations. Using the sample of sensitive bits gathered with their fault injector, they calculate a confidence interval for estimating the overall soft error rate of the design. By using a Xilinx ML-401 development board with a Xilinx Virtex-4 LX25 FPGA and JTAG, they have successfully maintained high portability by avoiding the need for specialized hardware, while sacrificing some thoroughness and reconfiguration speed. In 2014, a multibit fault injection (MBFI) methodology was been integrated into Simple Portable FPGA Fault Injector (SPFFI) to achieve an even higher injection rate [38].

Guzman-Miranda et al. [39] have designed their FT-UNSHADES2 fault injection platform to obtain high-speed fault injection and full coverage. They use a standard Xilinx motherboard: the ML-510 with a Virtex-5 (XC5VFX130T). They can test custom-made daughtercards, which interface with the motherboard via PCI-Express. To maximize fault injection speed, FT-UNSHADES2 utilizes the SelectMAP interface. If other FPGAs are to be tested, new daughter boards would need to be designed to interface with their system [39].

Starting with the Virtex-5 FPGA, Xilinx has released a proprietary IP core called Soft Error Mitigation (SEM) Core [40]. The SEM Core is instantiated within the user design and uses the ICAP to detect, correct, and classify soft errors in the Configuration Memory of an FPGA device. Xilinx has released a whitepaper showing an example SEU emulation environment that fault-injects random bit locations of the configuration bits (or only the essential bits if desired). The tool will summarize bits found and output an approximate derating factor [41].

While these fault injectors vary in technologies and methods used, they all have offered valuable insight on how FPGA designs can be protected from SEUs. The fault injection platform that was created for this research is described in Sections 3.2 and 3.3, and is contrasted with these previous fault injectors in Section 3.4.
3.2 Fault Injection on the XRTC Platform

This section describes the fault injection system hardware used for the experiments reported in this thesis. This fault injector was built using the XRTC infrastructure developed for the radiation testing of Xilinx FPGAs that was introduced in Section 2.2. This fault injection platform is named the XRTC-V5FI [8]. This section will also describe the hardware modifications made to support fault injection.

This work requires a large amount of fault injection experiments to be completed. Several of the experiments in this work require that the system be flexible enough to support alternative detection techniques and simple fault recovery methods. It is also necessary that these experiments be performed in a radiation test to validate or complement an experiment’s result if needed. These requirements motivate the construction of a high-speed fault injection platform that is accurate, flexible, and capable of being used for radiation testing.

3.2.1 Benefits of Using XRTC Hardware for Fault Injection

The XRTC test infrastructure described in Section 2.2 is ideal for use as a fault injection platform for a number of reasons. First, it has already been designed with all the components necessary to support radiation testing. Many of the same hardware and modules are similar or identical to their function when used in a radiation test. The ConfigMon has access to the DUT’s SelectMAP interface which enables high-speed configuration and modification of configuration memory bits. The FuncMon can provide all the design inputs and observe the design outputs at a high rate. Second, the XRTC test infrastructure supports multiple different DUT FPGAs, and these DUT FPGA daughtercards are already built. For a chosen DUT FPGA daughtercard, the same bitstream used for a fault injection test could be used for a radiation test. Third, the XRTC motherboard supports a variety of experiments and peripherals that could be used for special purpose fault injection experiments.

Although the XRTC infrastructure has a number of advantages for fault injection, it does have a few limitations. It is a very large and complicated system that has limited documentation intended for a small group of individuals. This makes it very challenging to develop hardware used on the service FPGAs. Also the XRTC motherboard and supporting hardware are expensive and are
manufactured in limited quantities. Finally, the XRTC infrastructure is also not portable, meaning that designs used for fault injection experiments must run on the daughtercard DUT FPGA rather than the system that the design is intended to run on. Designs that require interaction with other hardware components not provided by the test infrastructure can not undergo fault injection easily.

### 3.2.2 Supporting Fault Injection on XRTC Hardware

This section will describe the modifications made to the XRTC hardware to support fault injection on the XRTC test infrastructure. The resulting fault injector is called the XRTC-V5FI. The fault injector is built using three XTRC components: the XRTC motherboard, a test FPGA daughtercard, and a non-volatile PROM card (shown in Figure 3.1). The software for high level control and setup of the fault injection campaign was created for a host computer. The BrainBox that connects the host with the fault injector was replaced with a more reliable, faster, and flexible solution as part of this work.

![Figure 3.1: High Level View of Fault Injector Components without the BrainBox](image)

The first modification was the removal of the BrainBox (introduced in Section 2.2). The BrainBox limited the communication from the host computer to the ConfigMon with an inflexible protocol. Synchronizing the state between the BrainBox and the ConfigMon limited the speed of
communication. Also a method for dumping ConfigMon logs was unavailable and need to be created. By replacing the BrainBox with a UART interface over the RS232 port, this communication became more reliable, faster, and flexible. The logic required to communicate over the new serial interface was placed in the ConfigMon FPGA and utilized a PicoBlaze soft processor. Log dumps and new commands where now possible and added to the new system. A high level illustration of the system without a BrainBox is shown in Figure 3.1.

With radiation experiments, logging detailed fault information is done by the host. The host computer observes the error counters and saved information relevant to the status and address of the fault. This method is tolerated for radiation testing because faults are usually observed at a slower rate than fault injection. When this method was used for fault injection, there was a huge loss of time from the frequent interaction between the ConfigMon and FuncMon FPGAs when recording information about each discovered fault. While data was transmitted across the UART, fault injection was stalled to avoid fault detection at a rate faster than could be recorded. To alleviate this bottleneck, a first-in-first-out (FIFO) was created within the ConfigMon to log fault injection faults. Faults could then be logged while fault injection was still running and then be dumped to the host computer at a steady rate.

The FuncMon design was custom built to provide stimulus to the DUT and observe DUT outputs. The outputs of the DUT are compared to a golden reference for a mismatch (Figure 3.2) and, if detected, a fault would be registered for the current fault injection. The fault injection process at a bit-by-bit level (described in next section) is run with a program on a PicoBlaze, which is controlled externally through a serial interface (similar to what was done for the ConfigMon). By reducing dependence on the host computer and controlling the bit-by-bit fault injection within the same FPGA (FuncMon) that was used for fault detection, fault injection rate improved by over 200x.

3.3 Fault Injection Methodology

A methodology was designed to achieve the fault injection goals of this research and to run experiments on any FPGA circuit including soft processors. While the methodology is not specific to soft processor designs, several decisions were geared towards overcoming the challenges of performing fault injection on soft processors. These challenges include providing enough DUT
execution time to achieve acceptable coverage, and addressing faults that have propagated to non-volatile memories and that stick around after the fault was repaired and the DUT was reset. This methodology interrelates with hardware decisions made in Section 3.2 with the common goal of achieving high speed fault injection.

There are two important details to note about the fault injector: First, two copies of the soft processor are placed on the DUT FPGA (shown in Figure 3.2). The outputs of each copy are assigned to 72 bits of the 145-bit signal that is received by the FuncMon for error detection. Second, the fault injector is heavily dependent on asynchronous control from a host computer which introduces non-determinism into the system. The results can vary slightly between identical runs on the same design.

![Figure 3.2: Both processors are placed on the DUT FPGA. Their outputs are compared with each other on the FuncMon service FPGA.](image)

3.3.1 **Sequence for an Individual Injection of a Bit Fault**

Each bit is injected one at a time, and is later restored as the next bit is injected. For each individual fault injection, the test design reset signal is asserted to reset logic state before turning on the clock signal. If a fault was detected in injection of the prior bit, the design is reconfigured before injecting the next fault to ensure the is no old state in design memories. The time to inject
a single fault is at minimum 49.1 $\mu S$ (empirically derived by dividing the total campaign time by the total number of configuration bits injected).

The actual injection of the fault is performed by the ConfigMon by reading back a frame (memory chunk) from the DUT configuration memory, inverting the target bit, and writing the frame back into the DUT through the SelectMAP port (8-bits wide bus with 33 MHz clock). The XRTC-V5FI was designed to accurately measure configuration sensitivity by completely covering all 34.1 million configuration bits that control function and routing.

The list below is the fault injection loop procedure used for our experiments. This procedure is depicted as a diagram in Figure 3.3.

1. Inject fault: The ConfigMon FPGA toggles the target bit in the DUT FPGA’s configuration memory (this performs a single fault injection).

2. Assert reset: The FuncMon issues a reset signal to the DUT design and enables the clock on the DUT. The DUT is given time to load memories, execute software, and allow any errors to propagate through to DUT outputs.

3. Compare outputs: The DUT’s clock is stopped, and the outputs from both copies of the test design are compared with each other. Section 3.3.3 describes methods for detecting the fault.

4. Handle error: If an error was detected, the FuncMon signals the ConfigMon to record and log the error with the error’s location and type.
   
   (a) SEFI: If a single-event functional interrupt (SEFI) error (functional error independent of the test design) [42] is detected, it is recorded and the DUT is fully reconfigured. Fault injection resumes at the next bit.
   
   (b) Reconfigure: If the design contains a soft processor, we fully reconfigure the DUT after each detected output error to ensure full recovery of user-design memories.
   
   (c) Repeat: For reset recovery experiments (Section 3.3.3), the configuration memory bit is restored and this process is repeated to determine if the error remained. The error is recorded as either recovered or unrecovered.

5. Restore bit: The injected bit is restored as the next target bit is toggled by the ConfigMon.
3.3.2 Running a Fault Injection Campaign

Each campaign requires a design that is developed for the DUT FPGA that is duplicated and set to receive its clock, reset signal, and inputs from the FuncMon FPGA. Internal signals are outputted to the FuncMon FPGA for comparison. After the design is generated, fault injection campaign parameters are defined and set in the host software and the ConfigMon. The host then initiates fault injection with the given parameters, monitor the progress of the test, and receives log dumps.

With the fault injection procedure running, the host commands the FuncMon to sequentially initiate fault injection for a user-specified number of bits. The FuncMon will then perform a single fault injection for each bit by issuing commands to the ConfigMon, waiting for a user-specified software execution time, and reporting the result. The FuncMon reports the number of injected bits while the host ensures that all errors are recovered. The host also retrieves logged faults from the ConfigMon and keeps a database of errors with location and type, allowing for later analysis of the data.

Afterwards, the host computer parses the fault injection logs and the design utilization “map report” to generate a summary report file. The host will the insert the report into a database and display a web page with a full description of the experiment parameters and results.
3.3.3 Variations in Methodology

Different experiments can be performed on the same design with differences in methodology. Optional modifications to the methodology made in the software code that is loaded into the FuncMon’s PicoBlaze memory. We vary this methodology for multiple special-purpose experiments that help to further characterize design sensitivity. The three optional modifications that will be described are the reset recovery, the fault capture, and the additional DUT monitoring.

A system-wide reset can be a simple recovery technique for FPGA designs; however, it does not always successfully recover a soft processor design to the non-erroneous state. This happens when errors propagate into design memories and persist after a system reset. The goal of the reset-recovery experiments is to identify which configuration bits errors cannot be recovered from. This experiment requires an additional step in the fault injection procedure where the fault-injected bit is corrected, the test design is reset, and the design outputs are again checked for errors. Reset recovery results for multiple soft processors are shown in Section 5.2.3.

Another methodology change modifies when comparisons are made for determining faults and enables experiments that find how many faults go away on their own. The “fault capture” option was created to improve fault detection coverage and determine how many bits can be detected during some operations and not detected during other operations. When the design outputs (buses and nets) are compared only at the end of a single fault injection execution time, a previously visible output mismatch may depart and not be detected. The fault capture circuit will capture the faults that appear during execution, not just after execution. The fault capture implementation used for the system begins the capture halfway through the execution time.

One final modification was implemented to provide additional logging information about the kind of faults that are detected. This was accomplished by having the FuncMon fault detection logic observe specific DUT outputs and prioritize their logging over the rest of the DUT outputs. This could be useful for advanced tests that include mitigation strategies like TMR or checkpointing where masked faults could be reported in addition to the outputted faults. Another use for this modification would be to monitor faults that are detected within the DUT FPGA in addition to the faults detected in the FuncMon. This can identify how many bits are sensitive due to the detection and output logic (output registers, input/output block (IOB) slices). The type of event detected is encoded into the log message.
The reset recovery, fault capture, and additional DUT monitoring options were developed to allow more advanced experiments to be conducted with the XRTC-V5FI fault injection system. These modifications will be referred to in later chapters as they are used for experiments. The test infrastructure has two large service FPGAs where much flexibility is possible for many kinds of alternative methods and experiments that could be created in future work. The PicoBlaze module in the FuncMon serves as the fault injection controller, and the program executable can be loaded into the PicoBlaze BRAMs without having to reconfigure the FuncMon FPGA. The variations discussed in this section are written into this program executable and are enabled or disabled with simple UART commands from the Host to the FuncMon. Other characteristics of the XRTC-V5FI system are highlighted later in this section.

3.4 Comparison with Previous Works on Fault Injection

The fault injection system used in this thesis work (XRTC-V5FI) differs from the fault injection systems that were introduced in Section 3.1.2. This section will contrast the XRTC-V5FI with other fault injection systems.

As mentioned in earlier sections, the XRTC-V5FI system is based on a radiation test infrastructure. This attribute enables radiation test results and fault injection test results to validate and complement each other. There is one other fault injector that was also based on a radiation test infrastructure built by Brian Pratt [43]. Pratt’s fault injector uses an older version of the XRTC infrastructure that has two Xilinx Virtex-II Pro FPGAs on the motherboard and a Virtex-4 FPGA for the daughtercard. One difference between Pratt’s fault injector and the XRTC-V5FI is that Pratt’s fault injector retains the DUT configuration and mask data (target bits) on the host computer and uses a USB2.0 interface to receive this data from the host when needed. The XRTC-V5FI uses a PROM card to hold the DUT design configuration data and mask data and does not require interaction with the host to receive this data. Both approaches are fast, but the second approach is more suitable for soft-processor fault injection because soft processor designs need to be reconfigured or scrubbed more thoroughly than application-specific designs in order to recover completely from faults. A second difference is that the XRTC-V5FI uses the radiation-hardened V5QV FPGA daughtercard. The V5QV or its commercial counterpart would be more likely to be used in a space computing system.
The XRTC-V5FI uses the Xilinx SelectMAP interface for high-speed configuration and fault injection. A high-speed configuration interface like SelectMAP is critical for timely fault injection of all configuration bits and fast full fault recovery of soft processor design. Three other fault injection systems use the SelectMap interface: the SLAAC-1V [44], FT-UNSHADES2 [39], [45], and Pratt’s VII-Pro XRTC fault injector [43]. There are several partial reconfiguration fault injection systems that use the ICAP which is in essence an internal version of the SelectMAP interface [46], which are Sterpone and Violante’s partial reconfiguration-based fault injection system [36] and Xilinx’s SEM controller IP [40]. Other fault injectors (SPFFI [37] and FLIPPER [35]) use the slower JTAG external interface. Fault injection systems that use an external configuration interface can use the full DUT FPGA for the experiment design, whereas systems that use an internal interface can partially reconfigure areas of the FPGA that are unused by the fault injection controller. Some systems that use the JTAG interface for configuration have a portability advantage [37] of performing fault injection with a larger set of FPGAs and even FPGAs that are in their application circuit systems already.

This chapter describes a unique fault injection platform that offers exhaustive high-speed fault injection of FPGA configuration memory. This platform can be used for radiation tests and is flexible enough to handle the challenges of performing fault injection on soft processor designs and alternate fault detection and recovery experiments. The next chapter demonstrate work performed with the platform by introducing MicroBlaze soft processor fault injection experiments and results.
CHAPTER 4. MICROBLAZE SOFT ERROR SENSITIVITY EXPERIMENTS

Processors implemented in FPGA logic are known as soft processors, or softcore processors. The MicroBlaze is a common soft processor used exclusively on Xilinx FPGAs. It is supported by, and is proprietary to Xilinx. Due to well-supported tools and ease of customization, the MicroBlaze was chosen to be the first soft processor used in the experiments for this thesis. The experiments described here will show that software executables and customizations to the MicroBlaze will affect the configuration memory sensitivity of the MicroBlaze. Furthermore, the results shown here could be useful to other practitioners exploring trade-offs when building MicroBlaze designs on reliable computing systems.

This chapter introduces soft processors and explains why there is a need to perform fault injection on them. The MicroBlaze processor will be described in more detail, and experimental results on MicroBlaze sensitivity will be presented. Experiments were also performed on variations of the MicroBlaze design that include the design placement, memory size, and use of distributed LUTRAM. Finally, this chapter will present a MicroBlaze experiment where multiple benchmarks are tested using fault injection to understand the impact of software executables on SEU sensitivity.

4.1 Soft Processor Background

A soft processor is an implementation of a processor architecture for use on an FPGA that can often be customized by the user. The key advantage soft processors offer to their users over standard microprocessors is the ability to optimize the hardware design for a particular application using FPGA resources. The reconfigurability of soft processors is also advantageous in that it allows the design to be updated whenever new features are desired, granting the processors relative immunity to obsolescence and enabling changes even when the FPGA has been deployed in a remote or harsh environment.
With a rise in the use of soft processors in radiation environments, a detailed understanding of soft processor reliability and failure modes is becoming indispensable. Using fault injection, we can test the configuration memory sensitivity of soft processors on FPGAs in an effort to understand their reliability and evaluate soft processor mitigation strategies and recovery methods. However, fault injection for soft processors involves grappling with a number of challenges that are unique to these soft processor designs.

First, the reliability of a soft processor system depends not only on the specific hardware modules and features of the processor included in the system, but also on the software application that the processor is executing. Since different software programs exercise a processor’s functional units and memory in different ways, one software program may result with a different configuration memory sensitivity than another.

A second challenge in soft processor fault injection is handling errors that propagate into memories. If an error from an injected fault propagates into an FPGA memory resource such as BRAM, LUTRAM, or an SRL, the error can persist in the memory after a full system reset. Without using a special memory scrubbing approach or applying full reconfiguration of the FPGA bitstream to repair the error, subsequent configuration bits may be deemed sensitive when the actual cause of the error was a fault that was injected on a previous configuration bit.

A third challenge in conducting fault injection experiments on soft processors is choosing a design runtime long enough to ensure that any bootloader code has completed and that the desired software application has a chance to execute. If the execution time is too long, the overall experiment time can be prohibitive. The execution time needs to be long enough to cover the software and be short enough to minimize overall test time.

There are many soft processors that run on FPGAs, five of which are used in this research. The Xilinx MicroBlaze processor is introduced in Section 4.2 and four more soft processors are introduced in Section 5.1 which are the Xilinx PicoBlaze, Aeroflex Gaisler LEON3, the OpenCores OpenRISC, and the ARM Cortex-M0 DesignStart. This work will compare them in terms of configuration memory sensitivity. This chapter focuses on the sensitivity of a single processor, the MicroBlaze, and background is given in the following section.
4.2 Background on the MicroBlaze Soft Processor

The MicroBlaze is a 32-bit reduced instruction set computer (RISC) soft processor that is built and optimized for use solely on Xilinx FPGAs [47]. Xilinx provides the Xilinx Platform Studio where the MicroBlaze can be customized, combined with a large set of peripherals, and synthesized into a netlist for use within an FPGA design.

The MicroBlaze has a full Harvard architecture with separate data and instruction memory buses. It is highly customizable, and Xilinx has produced many compatible IP modules and libraries to use with it. The MicroBlaze can use an AXI or PLB bus interface depending on the target FPGA. The pipeline for the processor typically has five stages, and the MicroBlaze can be configured to use instruction and data caches when an external memory is used. A branch target cache can be included to allow for branch prediction [47]. The MicroBlaze has 32 32-bit general purpose registers.

The MicroBlaze provides a wide array of options and features for customization. The system can be setup as a single-processor or a dual-processor system with shared peripherals. The processor pipeline can either be a three or a five stage pipeline. Optional components include a floating point unit (FPU), barrel shifter, integer multiplier, integer divider, and pattern comparator. Instruction and data caches, memory management, and exception handling can also be added to the MicroBlaze design. Optional peripherals include (but are not limited to) a timer module, watchdog timer, UART, debug modules, and a general purpose input output (GPIO).

4.3 MicroBlaze Design Sensitivity

The first set of MicroBlaze experiments measures the sensitivity of the MicroBlaze processor running a single benchmark. These experiments will provide a baseline sensitivity measure that will be used throughout the work as a comparison with other related sensitivity experiments. In addition to obtaining a sensitivity measure, this set of experiments will measure the variability in sensitivity from run to run and investigate the impact of placement and floor planning on the processor sensitivity.

A basic MicroBlaze processor was created with minimal features to run this benchmark. The features of the processor include: a 32-bit multiplier, barrel shifter, pattern comparator, and
two BRAMs for instruction and data memories (8KB total). It contains no divider, cache, FPU, exceptions, memory management unit (MMU), or debug units. The size of the processor for this experiment is 515 slices or 2.5% of the Xilinx V5QV FPGA (20,480 total slices). The minimum clock period for the implemented design is 9.934 ns.

The benchmark used for this first experiment runs the recursive Towers of Hanoi program. This is a relatively simple program that consumes 4.2KB of program memory and fits within in a single BRAM. The program is defined in 25 lines of code.

### 4.3.1 MicroBlaze Raw Sensitivity Results

The first set of experiments measures the sensitivity of the MicroBlaze processor while running a single benchmark. The experiment will provide a baseline sensitivity measure that will be used throughout the work as a comparison with other related experiments. In addition to obtaining a sensitivity measure, this set of experiments will measure the variability in sensitivity from run to run.

The sensitivity of the MicroBlaze processor running the Towers of Hanoi benchmark is estimated by performing the fault injection procedure described in Section 3.3 six different times. With each run requiring 1028 minutes, the six runs took over 102 hours to complete. The total number of sensitive bits measured in each run is shown in Table 4.1. The average sensitivity measured in these six runs is 52,003 configuration bits or 0.15% of the total configuration bitstream.

In addition to the average sensitivity, Table 4.1 provides the minimum, maximum, range, and standard deviation of the sensitivity measurements. It is surprising to see that the range in sensitivity between these six runs is very small – the difference between the lowest and highest sensitivity measure is only 41 out of an average of 52,003 bits (0.079% of the average). The standard deviation of the six runs is 17.11 or 0.033% of the average. These results suggest that the sensitivity measurements in a run do not vary much and that a single run is probably sufficient to obtain an accurate sensitivity estimate for most experiments.

Although the sensitivity results from run to run are statistically very close to each other, it is not clear from Table 4.1 whether the sensitive bits in each run are the same or not. To determine how many unique configuration bits are found in this set of runs, the union of all unique sensitive configuration bits identified in these six experiments was determined. The total number of unique
Table 4.1: Sensitive Bits Measured over Multiple Runs of a Single MicroBlaze Design

<table>
<thead>
<tr>
<th>Run</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51,982</td>
</tr>
<tr>
<td>2</td>
<td>52,023</td>
</tr>
<tr>
<td>3</td>
<td>52,001</td>
</tr>
<tr>
<td>4</td>
<td>51,996</td>
</tr>
<tr>
<td>5</td>
<td>51,990</td>
</tr>
<tr>
<td>6</td>
<td>52,024</td>
</tr>
<tr>
<td>Min</td>
<td>51,982</td>
</tr>
<tr>
<td>Max</td>
<td>52,023</td>
</tr>
<tr>
<td>Avg</td>
<td>52,003</td>
</tr>
<tr>
<td>Std. Dev. σ</td>
<td>17.11 (0.03%)</td>
</tr>
<tr>
<td>Max-Min Δ</td>
<td>41 (0.08%)</td>
</tr>
<tr>
<td>Set Union</td>
<td>52,430</td>
</tr>
</tbody>
</table>

bits found was 52,430 or 407 more configuration bits than identified in the sensitivity run with the maximum number of configuration bits. This result suggests that while a few new sensitive configuration bits are identified on subsequent runs, the vast majority of sensitive configuration bits are identified on a single run.

Table 4.2: Configuration Bits Common to Multiple Runs

<table>
<thead>
<tr>
<th>Bit Intersection</th>
<th>Number</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 of 6 Runs</td>
<td>50,867</td>
<td>97.019%</td>
</tr>
<tr>
<td>5 of 6 Runs</td>
<td>1,847</td>
<td>2.258%</td>
</tr>
<tr>
<td>4 of 6 Runs</td>
<td>96</td>
<td>0.183%</td>
</tr>
<tr>
<td>3 of 6 Runs</td>
<td>72</td>
<td>0.137%</td>
</tr>
<tr>
<td>2 of 6 Runs</td>
<td>88</td>
<td>0.168%</td>
</tr>
<tr>
<td>1 of 6 Runs</td>
<td>125</td>
<td>0.238%</td>
</tr>
</tbody>
</table>

As shown in Section 3.1.1, the sensitivity of the processor can be normalized by the size of the design. Using Equation 3.2, the normalized sensitivity of the MicroBlaze running this benchmark is 6.07%. This number suggests that roughly 6% of the configuration bits associated with the slices used by the MicroBlaze are sensitive to upsets. This is slightly lower than the 10% normalized sensitivity that is used as a rule of thumb by some practitioners.
4.3.2 Placement Dependencies

To understand the effect of placement on sensitivity, different placements of the same design were generated and the same sensitivity measurements were performed. To generate a unique placement in the Xilinx ISE tools (version 13.2), we modify the seed used in the “Starting Placer Cost Table”. Three MicroBlaze designs were generated with the seeds 22, 49, and 75 (the seed values were chosen randomly). A fourth placement was created by providing a user area constraint to a square area that provided 392 possible slices per processor, which reduced the design size in slices by 25%. The FPGA placement and route layout of each design is shown in Figure 4.1 where layouts A–C are the seeds 22, 49, and 75, and layout D is the constrained MicroBlaze. The layout representations show how different seeds can vary the placements and also how the area constraint differs from the three unconstrained layouts. As described in Section 3.3, each design in the sensitivity experiment has two processors for detecting processor faults.

Figure 4.1: The placement and route layout images of each MicroBlaze where placement was generated from random seeds (a) 22, (b) 49, and (c) 75, and (d) using an area constraint.

The results of each fault injection experiment are shown in Table 4.3. The average, range, and standard deviation of the three designs using a unique seed are also shown. Like the previous experiment involving multiple runs of the same mapped design, the sensitivity does not vary much
between designs with different placements. However, the range (Max-Min) of sensitivity in these experiments is 541 bits or $17 \times$ that of the previous experiment. The standard deviation is 294 bits, or 0.5305% of the mean average, which is $16 \times$ greater than the multiple run experiment. This result suggests that a unique placement has a measurable impact on sensitivity.

Table 4.3: Variation in Placement and Routing

<table>
<thead>
<tr>
<th>Design</th>
<th>Sensitive Bits</th>
<th>Slices</th>
<th>Normalized Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed 22</td>
<td>55,286</td>
<td>512</td>
<td>6.488%</td>
</tr>
<tr>
<td>Seed 49</td>
<td>55,754</td>
<td>463</td>
<td>7.235%</td>
</tr>
<tr>
<td>Seed 75</td>
<td>55,213</td>
<td>484</td>
<td>6.854%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>55,418</strong></td>
<td><strong>487</strong></td>
<td><strong>6.846%</strong></td>
</tr>
<tr>
<td>Max-Min Δ</td>
<td>541</td>
<td>49</td>
<td>0.747%</td>
</tr>
<tr>
<td>Std Dev σ</td>
<td>294</td>
<td>25</td>
<td>0.374%</td>
</tr>
<tr>
<td>Constrained</td>
<td>62,013</td>
<td>366</td>
<td>10.180%</td>
</tr>
</tbody>
</table>

The area-constrained MicroBlaze design exhibits a significantly higher number of sensitive bits than the non-constrained designs (62,013 sensitive bits). This is 11.9% more sensitive than the non-constrained designs. Because the design takes 25% fewer slices, the normalized sensitivity is 48.7% higher than the non-constrained designs. This result suggests that while user constraints can reduce the number of slices used by the design, it may significantly increase the configuration sensitivity. The relationship between design sensitivity and design size is shown in Figure 4.2.

### 4.3.3 Experiments with Additional MicroBlaze Variations

Additional fault injection experiments were conducted where changes are made to either the design or the experiment. These changes help to further characterize the MicroBlaze sensitivity. The first experiment tests a MicroBlaze where LUTRAM and shift register LUTs (SRLs) primitives are disallowed from the implementation. The second experiment has an increase in memory even though the software does not require it. The third experiment varies the software execution time.

There is an option in the MicroBlaze netlist generation tool that disallows the use of primitives (LUTRAM and SRLs). The MicroBlaze designs in this section only have LUTRAM so this option would disable the use of LUTRAM. This experiment explores what happens in MicroBlaze
sensitivity when the LUTRAM and SRL primitives are turned off in the MicroBlaze configuration. The utilization for the two designs are shown in Table 4.4 and the placement of each design is shown in Figure 4.3. The design with no LUTRAM is much larger than the design with LUTRAM. The sensitivity results are also shown in Table 4.4 and they show that the use of LUTRAM decreases the raw sensitivity of the design. When not using LUTRAM, the normalized sensitivity decreases which could be due to an increased design size and a reduction in the density of logic which is spread out across more slices.

Table 4.4: Sensitivity Results and Utilization for MicroBlaze with and without LUTRAM Primitives

<table>
<thead>
<tr>
<th>Design</th>
<th>Sensitive Bits</th>
<th>Slices</th>
<th>LUTs as Logic</th>
<th>LUTs as Memory</th>
<th>Normalized Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze with LUTRAM</td>
<td>51,947</td>
<td>515</td>
<td>1,095</td>
<td>64</td>
<td>6.066%</td>
</tr>
<tr>
<td>MicroBlaze without LUTRAM</td>
<td>62,945</td>
<td>1,043</td>
<td>2,088</td>
<td>0</td>
<td>3.629%</td>
</tr>
<tr>
<td>Difference</td>
<td>10,998</td>
<td>528</td>
<td>993</td>
<td>(64)</td>
<td>2.437%</td>
</tr>
</tbody>
</table>

The previous experiments use a MicroBlaze with only 8KB of memory (4KB instruction memory and 4KB data memory) which is enough memory to run the recursive Towers of Hanoi benchmark. The MicroBlaze memory experiment has additional memory added (128KB total) to
Figure 4.3: Placement Screenshot for MicroBlaze with (a) and without (b) LUTRAM Primitives

the MicroBlaze design, and the executing software was recompiled for the new processor. The utilization for the two designs are shown in Table 4.5 and the placement of each design is shown in 4.4.

Table 4.5: Sensitivity Results and Utilization for MicroBlaze with Different Memory Sizes

<table>
<thead>
<tr>
<th>Design</th>
<th>Sensitive Bits</th>
<th>Slices</th>
<th>BRAMs</th>
<th>Normalized Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze 8KB</td>
<td>51,947</td>
<td>515</td>
<td>2</td>
<td>6.066%</td>
</tr>
<tr>
<td>MicroBlaze 128KB</td>
<td>64,731</td>
<td>483</td>
<td>32</td>
<td>8.061%</td>
</tr>
<tr>
<td>Difference</td>
<td>12,784</td>
<td>(32)</td>
<td>30</td>
<td>1.995%</td>
</tr>
</tbody>
</table>

The Execution Time Dependence Experiment is the final experiment in this section which modifies the amount of time for the processor to run during the fault injection experiment. The fault injection capture circuit from Section 3.3.3 is used (captures faults during final 50% of execution time) and this experiment show the number of faults detected as the execution time increases. One might think that the number of faults captured would increase as the execution time increases, but
because the capture circuit captures faults during only the last half of the execution, the capture
time window changes its start time as well as its end time. This means that if an earlier part of
the program is more sensitive than the a later part, a longer execution time window could have its
starting time of the window after the more sensitive program segment and count a smaller number
of sensitive bits.

The results for the Execution Time Dependence Experiment are shown in Table 4.6 and
Figure 4.5. The Execution Time Multiplier column is an execution time parameter in the settings for
the fault injection campaign. This parameter is used by the fault injection controller (a PicoBlaze)
inside the FuncMon. The execution times in clock cycles are listed as well as the cycle in which the
capture circuit was enabled. The sensitive bits captured average 53,435 and range from 53,167 bits
to 53,983 bits. The sensitive bits have a standard deviation of 215 bits which is 0.4% of the average
and is slightly higher than the Repeated MicroBlaze experiment in Section 4.3.1. The sensitive
bits have an increasing trend up to time multiplier 8 and then they drop for time multipliers 9–12.
Because variations are small, these results may reflect the indeterminism in the fault injector as
well as variations in detection coverage that depend on the execution time and the fault capture sample window.

Table 4.6: Execution Time Dependence on MicroBlaze and Towers of Hanoi Results

<table>
<thead>
<tr>
<th>Execution Time Multiplier</th>
<th>Execution Time in Clock Cycles</th>
<th>Start Capture Cycle</th>
<th>Cycles with Capture Enabled</th>
<th>Sensitive Bits Captured</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>772</td>
<td>387</td>
<td>385</td>
<td>53,167</td>
</tr>
<tr>
<td>2</td>
<td>1,541</td>
<td>772</td>
<td>769</td>
<td>53,315</td>
</tr>
<tr>
<td>3</td>
<td>2,310</td>
<td>1,156</td>
<td>1,154</td>
<td>53,437</td>
</tr>
<tr>
<td>4</td>
<td>3,079</td>
<td>1,541</td>
<td>1,538</td>
<td>53,425</td>
</tr>
<tr>
<td>5</td>
<td>3,848</td>
<td>1,925</td>
<td>1,923</td>
<td>53,561</td>
</tr>
<tr>
<td>6</td>
<td>4,617</td>
<td>2,310</td>
<td>2,307</td>
<td>53,536</td>
</tr>
<tr>
<td>7</td>
<td>5,386</td>
<td>2,694</td>
<td>2,692</td>
<td>53,564</td>
</tr>
<tr>
<td>8</td>
<td>6,155</td>
<td>3,079</td>
<td>3,076</td>
<td>53,983</td>
</tr>
<tr>
<td>9</td>
<td>6,924</td>
<td>3,463</td>
<td>3,461</td>
<td>53,294</td>
</tr>
<tr>
<td>10</td>
<td>7,693</td>
<td>3,848</td>
<td>3,845</td>
<td>53,409</td>
</tr>
<tr>
<td>11</td>
<td>8,462</td>
<td>4,232</td>
<td>4,230</td>
<td>53,286</td>
</tr>
<tr>
<td>12</td>
<td>9,231</td>
<td>4,617</td>
<td>4,614</td>
<td>53,242</td>
</tr>
</tbody>
</table>

| Minimum | 53,167 |
| Maximum | 53,983 |
| Maximum - Minimum Δ | 816 (1.527%) |
| Arithmetic Average | 53,435 |
| Standard Deviation σ | 215 (0.402%) |

The three MicroBlaze Variation Experiments suggest that the MicroBlaze sensitivity is increased when no LUTRAM is included in the design and when additional memory is added (even when it is not required). Execution Time Dependence Experiment tells us that the results can vary due to the sample time and duration.

All the MicroBlaze experiments in this section provide results that may help an engineer better understand how the MicroBlaze and its features have an impact on the reliability of a soft processor-based space computing system. The following section will experiment with different software benchmarks to further characterize the MicroBlaze soft processor.
4.4 MicroBlaze Software Sensitivity

The sensitivity of a processor design will depend on the areas of the processor that are exercised during the fault injection experiment. Because different software programs use a different composition of instructions, each benchmark program will exercise different parts of the processor. As such, the sensitivity measurements of a soft processor are expected to vary for different benchmarks. The sensitivity of several benchmarks was measured to determine this variation.

To support a larger range of benchmarks, a larger, more complete MicroBlaze processor with more features was used. The MicroBlaze used in this experiment has a larger BRAM internal memory (instruction and data memories total 128KB) and the processor includes hardware FPU support (a feature not included in the other experiments in this chapter). No caches are used and no external memory interface is added. This MicroBlaze utilizes 912 slices with 32 BRAMs and 5 DSP48Es. Also, 64 slices LUTs are used as LUTRAM, and 21 slices are used as SRLs.

The fault injection system parameters were modified for the purpose of the benchmark experiment. Each injection was given 16,921 clock cycles of run time after reset to give the program more cycles to execute instructions, which is more execution time than the previous experiments. For 34.1 million injections, the total run time can add up quickly. Each test took between 48 and 71 hours, which added up to 13 days of fault injection time for this experiment.
4.4.1 Overview of Benchmarks Used

Five benchmarks were chosen to test the variation in sensitivity on executed instructions. Each of the benchmarks execute out of the internal BRAM memories and use only internal memory during their execution. The benchmarks used in this experiment are summarized below:

- **Towers of Hanoi** is a recursive benchmark that computes the solution to the Towers of Hanoi puzzle involving the movement of disks of various sizes from one peg to another.

- **Dhrystone** is a synthetic benchmark for testing the integer performance of a processor. Dhrystone has been an industry standard for measuring integer performance for years [48]. Dhrystone scores are available for many different processors.

- **Whetstone** is a synthetic benchmark used for measuring a processor’s floating point performance. We performed our tests using the single-precision subset of the benchmarks [49].

- **CoreMark** is a synthetic benchmark that measures integer performance, developed by the Embedded Microprocessor Benchmark Consortium (EEMBC) in 2009. This benchmark is intended as a replacement for Dhrystone, and it executes four different subprograms: list searching and sorting, matrix operations, a state machine on a series of numerical inputs, and a computing a CRC [50].

- The **Dijkstra** benchmark implements Dijkstra’s algorithm for determining the shortest path in a graph [51]. It builds a large adjacency matrix and then it repeatedly calculates the shortest path for each pair of nodes in \(O(n^2)\) time.

4.4.2 Results of Benchmark Experiments

The results for the MicroBlaze software benchmark experiment are listed in Table 4.7. They range from Towers of Hanoi at 76,733 sensitive bits to Whetstone at 114,508 which is 49.2% greater than Towers of Hanoi. Note that the MicroBlaze design running Towers of Hanoi is much more sensitive than the MicroBlaze used in Section 4.3 due to a larger platform design with added memory, the FPU, and other features described at the beginning of this section.
As shown in Table 4.7, sensitive bits measure strongly depend on the software used. The integer synthetic benchmarks (Dhrystone and CoreMark) show roughly 39% more sensitive bits than the simpler kernel benchmarks (Towers of Hanoi, Dijkstra), and the floating point synthetic benchmark (Whetstone) has nearly 7% more sensitive bits than the integer synthetic benchmarks.

Table 4.7 also includes the arithmetic mean, the union of all the sensitive bit sets, and the intersection of all the sensitive bit sets. The set union for all detected sensitive bits is 150,531, meaning that the most sensitive benchmark, Whetstone, covers 76% of all detected sensitive bits, and the least sensitive benchmark, Towers of Hanoi, is 51% of all detected sensitive bits. The set intersection is 58,638 bits, showing that 39% of all detected sensitive bits are present in all five benchmarks.

The sizes of each compiled executable and its code memory segment are also shown in Table 4.7. FPU usage and the size of the compiled code memory segment both correlate well with the number of sensitive bits detected.

### 4.4.3 Sensitivity Cost of an Unutilized FPU

Most of the benchmarks described in Section 4.4.2 are integer benchmarks. The results shown in Table 4.7 also used the same MicroBlaze configuration and features, which includes an enabled FPU. The integer experiments (Hanoi, Dhrystone, CoreMark, Dijkstra) were run in order to see how an unused FPU affects the sensitivity of a MicroBlaze.
The comparison results are shown in Figure 4.6 and Table 4.8. The additional FPU hardware, even though it is not being used, adds as much at 9.5% additional sensitivity to the design. Figure 4.6 shows that the FPU and the non-FPU MicroBlaze designs have similar sensitivities and the FPU MicroBlaze is more sensitive by a small amount. Table 4.8 shows that the difference in sensitivities vary from an additional 4,925 (6.91%) bits to an additional 8,460 (9.49%) bits.

Even though the FPU is not utilized by the benchmarks in this experiment, it increases the design sensitivity because it is connected to and integrated with the rest of the design. The FPU MicroBlaze design takes up 912 slices and the non-FPU MicroBlaze takes 483 slices. The FPU MicroBlaze takes 429 (88.82%) slices more than the non-FPU MicroBlaze.

![Figure 4.6: MicroBlaze FPU Comparison for Integer Benchmarks](image)

This chapter presented experiments that show how the MicroBlaze varies in sensitivity to different software programs and MicroBlaze features and layouts. With this information, a practitioner could make more meaningful estimates on the configuration memory sensitivity of a MicroBlaze from empirical results. If also given the static SEU rates on the target FPGA configuration memory, a designer could also estimate the design SEU upset rate and the mean time to
Table 4.8: MicroBlaze FPU Comparison for Integer Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>No FPU</th>
<th>FPU</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hanoi</td>
<td>64,731</td>
<td>70,874</td>
<td>6,143 (9.49%)</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>89,141</td>
<td>97,601</td>
<td>8,460 (9.49%)</td>
</tr>
<tr>
<td>Coremark</td>
<td>89,284</td>
<td>97,262</td>
<td>7,978 (8.94%)</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>71,312</td>
<td>76,237</td>
<td>4,925 (6.91%)</td>
</tr>
<tr>
<td>Average</td>
<td>78,617</td>
<td>85,494</td>
<td>6,877</td>
</tr>
<tr>
<td>Max</td>
<td>89,284</td>
<td>97,601</td>
<td>8,460</td>
</tr>
<tr>
<td>Min</td>
<td>64,731</td>
<td>70,874</td>
<td>4,925</td>
</tr>
<tr>
<td>Max - Min Δ</td>
<td>24,553</td>
<td>26,727</td>
<td>3,535</td>
</tr>
<tr>
<td>Std. Dev. σ</td>
<td>12,526</td>
<td>13,958</td>
<td>1,640</td>
</tr>
</tbody>
</table>

Figure 4.7: Placement Screenshot for MicroBlaze (a) with and (b) without FPU

SEU upset for a specific orbit in space. The next chapter will introduce experiments where fault injection was performed on additional soft processors.
CHAPTER 5. FAULT INJECTION EXPERIMENTS ON MULTIPLE SOFT PROCESSORS

There are a variety of soft processors cores that are available for FPGAs, each with their own trade-offs of size, performance, and reliability. While research has been done to investigate soft processor performance [52], there is no research (to our knowledge) that estimates the SEU sensitivity for a wide variety of processors. This chapter describes two sets of experiments where fault injection is performed on multiple soft processors. The first set of experiments use five different processors with a minimalistic configuration and a simple software benchmark. The second set of experiments uses three processors that are set with a higher performance configuration and are run with four software benchmarks. These experiments will help us understand the difference in SEU sensitivity of five different processors and aid in the selection and design of a soft processor to use in a radiation environment.

This chapter is organized to first provide background on the additional processors and benchmarks used, and then to describe the two experiments performed for this study. The first experiment compares the Xilinx MicroBlaze, the AeroFlex Gaisler LEON3, the ARM Cortex-M0, the OpenRISC, and the Xilinx PicoBlaze. Each processor is built with similar minimal configurations and runs the Towers of Hanoi benchmark. The second set of experiments compares only the MicroBlaze, LEON3, and the Cortex-M0. In the second set of experiments, the processors are built for performance and run multiple benchmarks during fault injection. The benchmarks used are Dhrystone, Whetstone, basicmath and qsort.

5.1 Processor Examples

This study will include the MicroBlaze, LEON3, Cortex-M0, OpenRISC, and PicoBlaze soft processor cores. The MicroBlaze was introduced in Section 4.2. The other processors will be introduced below.
The LEON3 is a technology independent 32-bit soft processor developed by Aeroflex Gaisler. It implements the SPARC V8 architecture and uses an AMBA 2.0 AHB bus. The seven-stage pipeline of the LEON3 is one the deepest of any FPGA-based processor. The LEON3 employs separate instruction and data caches whose size and associativity can be configured by the user. While the LEON3 does not have a branch target cache, branch prediction can be enabled using a branch-always strategy [53]. Fault-tolerant versions of the LEON3 and its FPU are available commercially.

The Cortex-M0 DesignStart is a 32-bit ARM soft processor that implements the ARMv6-M instruction set architecture. It has a Von Neumann architecture (instructions and data are stored in the same memory space), implements a three-stage pipeline, and is organized around an AMBA 3 AHB-Lite system bus [54]. This processor is a free version of a commercial processor, the Cortex-M0, with a more limited feature set. The Cortex-M0 DesignStart is delivered as obfuscated, gate-level, technology-independent Verilog code that prevents the user from changing the processor configuration.

The OpenRISC 1200 is an open-source soft processor developed and maintained by the OpenCores organization. The OpenRISC instruction set architecture (called the OpenRISC 1000 or OR1K architecture) is based on the DLX and MIPS architectures. The processor has a Harvard architecture, implements a five stage pipeline, and is built around a Wishbone system bus [55].

The Xilinx PicoBlaze processor is an efficient, 8-bit embedded controller for use in simple control applications on Xilinx FPGAs. It is very small and used in many applications requiring simple, low-bandwidth I/O processing and handling [56]. While not directly comparable to the other processors (all other processors are 32-bit), this processor provides an interesting alternative to the larger, more complete processors described above.

5.2 Fault Injection of Minimalistic Processors

For this first set of experiments, all five processors were configured with a minimal set of options and features, and run with only a single simple benchmark. This experiment allows us to compared the SEU sensitivity of the processors in their most basic form.
First, an overview of the experimental setup and the configurations of each processor will be explained. Then, the raw sensitivity results for each processor will be shown along with their normalized sensitivities. Finally, experimental variations will be described, along with their results.

5.2.1 Overview of Experiment

All the processors used in this study are user-configurable. For this set of experiments, each processor is built with a minimal amount of BRAM memory. No FPUs, MMUs, debug modules, or caches are enabled in any of the processors. The size of each processor is shown in Table 5.1.

The processors were clocked at 50MHz and were given 16,921 clock cycles of run time after reset to execute the Towers of Hanoi benchmark. Not all processors have the same boot sequence; the LEON3 design has a separate read-only memory (ROM) and a bootloader for loading the program memory into RAM. Additional execution time is required in order to test the processor software execution. Each processor fault injection experiments ranged from 12 hours (PicoBlaze) to 68 hours (Cortex-M0) totaling 10 days of fault injection run time over all five processors.

The signals used for error detection were selected from processor buses. In the case of the MicroBlaze, the dedicated local memory bus (LMB) was used. For the other processors, the system bus is used, which is attached to the memory controllers. The objective is to observe memory writes and reads along with other bus transactions.

Table 5.1: Utilization for Minimalistic Soft Processor Design Implementations

<table>
<thead>
<tr>
<th>Processor</th>
<th>Slices</th>
<th>BRAMs</th>
<th>DSP48Es</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze</td>
<td>515</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>LEON3</td>
<td>1,177</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>Cortex-M0</td>
<td>1,780</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>OpenRISC</td>
<td>828</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>PicoBlaze</td>
<td>70</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

As described in Section 3.3, each design in the sensitivity experiment has two processors for detecting processor faults. The Figure 5.1 shows the layouts of the various soft processors. They all vary in size and shape with the largest being the Cortex-M0 5.2.1 and the smallest being the PicoBlaze 5.2.1. Table 5.1 shows the FPGA resource utilization for each design.
Figure 5.1: Placement Screenshots for Minimalistic Soft Process in Top Left to Bottom Right Order: MicroBlaze, LEON 3, ARM Cortex-M0, OpenRISC, and PicoBlaze
5.2.2 Raw Sensitivity Results

These soft processor experiments apply the fault injection methodology described in Section 3.3 to each of the five processors while running the Towers of Hanoi benchmark. Although this benchmark is the least sensitive benchmark of the benchmark suite, it can be compiled on all five of the processors that were created (with minimal configurations). Even though each processor was customized to have similar features and memory sizes, their architectures are significantly different and the larger processors were more underutilized than the smaller processors.

The sensitivity results from the soft processor experiments are listed in Table 5.2 and shown in Figure 5.2. This table lists the size of the processor (in FPGA slices), the raw number of sensitive configuration bits identified in the fault injection experiment, and the normalized sensitivity (see Equation 3.2).

Several important observations should be made from these results. First, each processor has a unique normalized sensitivity. The Xilinx-specific processors (the MicroBlaze and PicoBlaze) have a higher normalized sensitivity, possibly because they are designed with Xilinx FPGA primitives and also have a higher logic density. Another observation is that the larger designs have lower normalized sensitivities. This result may be due to the simple program, which stresses fewer architectural features in the larger processors, resulting in a lower normalized sensitivity. Even though the PicoBlaze cannot handle large applications, it can run the Towers of Hanoi benchmark with a design that is 4% of the size and 6.3% of the sensitive bits of the Cortex-M0.

Table 5.2: The Sensitivity Results for Minimalist Soft Processor Design Implementations

<table>
<thead>
<tr>
<th>Processor</th>
<th>Slices</th>
<th>Sensitive Bits</th>
<th>Normalized Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze</td>
<td>515</td>
<td>51,947</td>
<td>6.06%</td>
</tr>
<tr>
<td>LEON3</td>
<td>1,177</td>
<td>94,189</td>
<td>4.81%</td>
</tr>
<tr>
<td>Cortex-M0</td>
<td>1,780</td>
<td>112,224</td>
<td>3.79%</td>
</tr>
<tr>
<td>OpenRISC</td>
<td>828</td>
<td>77,650</td>
<td>5.63%</td>
</tr>
<tr>
<td>PicoBlaze*</td>
<td>70</td>
<td>7,116</td>
<td>6.11%</td>
</tr>
<tr>
<td>Average</td>
<td>1,075</td>
<td>84,003</td>
<td>5.07%</td>
</tr>
<tr>
<td>Max-Min</td>
<td>1,265</td>
<td>60,277</td>
<td>2.32%</td>
</tr>
</tbody>
</table>
5.2.3 Reset Recovery Experiment

Additional experiments were performed on the processors in Section 5.2.2 to determine how well each processor recovers after a reset. The modification to the fault injection methodology that enabled this experiment is described in Section 3.3.3. After a fault is determined to be sensitive, the reset recovery modification removes the injected fault and resets the system to check if the system has fully recovered from the fault. Faults that are unrecoverable by reset have effects that propagate and persist in LUTRAM, SRLs, and the BRAMs that serve as read-only memories (ROMs). The recovery rates are shown for all five processors in Table 5.3. The “no LUTRAM” MicroBlaze from Section 4.3.3 is also included in the results.

The processor with the lowest reset recovery rate is the OpenRISC with 19.36% of the sensitive bits being unrecovered by reset. Soft processors with the highest reset recovery rate are the LEON3 and the PicoBlaze. The MicroBlaze and the Cortex-M0 have similar recovery rates. The LEON3 has a separate ROM memory where the program executable is stored and loaded into RAM on start-up by a bootloader. This bootloader sequence effectively scrubs most errors that could persist in processor memories (BRAM) after a reset. The remaining unrecovered bits (215) could be from errors related to LUTRAMs and SRLs.
Table 5.3: Reset Recovery Results for Various Minimalistic Soft Processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Sensitive Bits</th>
<th>Unrecovered Bits</th>
<th>Percentage Unrecovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze</td>
<td>52,001</td>
<td>7,136</td>
<td>13.72%</td>
</tr>
<tr>
<td>MicroBlaze (No LUTRAM)</td>
<td>73,653</td>
<td>9,954</td>
<td>13.51%</td>
</tr>
<tr>
<td>LEON3</td>
<td>94,327</td>
<td>215</td>
<td>0.23%</td>
</tr>
<tr>
<td>Cortex-M0</td>
<td>119,157</td>
<td>15,258</td>
<td>12.80%</td>
</tr>
<tr>
<td>OpenRISC</td>
<td>77,258</td>
<td>14,955</td>
<td>19.36%</td>
</tr>
<tr>
<td>PicoBlaze</td>
<td>7,072</td>
<td>18</td>
<td>0.25%</td>
</tr>
</tbody>
</table>

In this section, five different soft core processors were tested to contrast the sensitivity and normalized sensitivity of a variety of soft processors. These results suggest that the MicroBlaze had the lowest design sensitivity but also had the highest normalized sensitivity. The high normalized sensitivity is due to the optimization of the architecture to the Virtex-5 FPGA architecture.

5.3 Experiments Using Performance Level Processors on Multiple Benchmarks

This set of processor experiments will allow us to determine the SEU sensitivity of the processors in a configuration that is slightly more representative to their actual use. Three soft processors are used for this experiment and each processor is configured for performance and given larger memories to handle larger benchmarks. Each processor also is given an FPU. First, an overview of the experimental setup and the configurations of each processor will be explained. Then the sensitivity results for each processor and benchmark combination will be shown.

5.3.1 Overview of Experiment

The Performance-Processor Benchmark Experiments can be described as a cross product of the Minimalistic Processor Experiments (Section 5.2) and the MicroBlaze Benchmark Experiments (Section 4.4) but with different sets processor designs and benchmarks. The major differences from the Minimalistic Processor Experiments are that only three different processors are tested and these processors have a larger configuration with more features, memory, an FPU, and a UART TX module. The resource utilization for each processor is listed in Table 5.4 (there are two LEON3 processors used because two benchmarks required additional memory). The major differences
from the MicroBlaze Benchmark Experiments are that a different set of benchmarks are used. Similarly, all benchmarks used here are modified to loop infinitely.

Table 5.4: Resource Utilization for Performance Processor Design Implementations

<table>
<thead>
<tr>
<th>Processor</th>
<th>Slices</th>
<th>BRAMs</th>
<th>DSP48Es</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze</td>
<td>1,866</td>
<td>32</td>
<td>20</td>
</tr>
<tr>
<td>LEON3 (Dhrystone &amp; Whetstone)</td>
<td>6,501</td>
<td>62</td>
<td>6</td>
</tr>
<tr>
<td>LEON3 (basicmath &amp; qsort)</td>
<td>6,435</td>
<td>59</td>
<td>6</td>
</tr>
<tr>
<td>Cortex-M0</td>
<td>1,690</td>
<td>33</td>
<td>0</td>
</tr>
</tbody>
</table>

The benchmarks used for this experiment are Dhrystone, Whetstone, basicmath, and qsort. Dhrystone and Whetstone were already introduced in 4.4.1. The basicmath benchmark executes a number of floating point math functions that embedded processors typically do not have hardware support for. Some examples are “cubic function solving, integer square root and angle conversions from degrees to radians” [51]. The inputs values are fixed constants.

The final benchmark is qsort, the common Quicksort algorithm. Quicksort is a recursive sorting algorithm which on average runs $O(n\log(n))$ comparisons. The data set used is an array of strings that are compared and sorted when executed [51].

5.3.2 Results and Analysis

A total of twelve fault injections runs were performed, each a combination of three processors and four benchmarks. The raw sensitivity of these experiments is shown in Table 5.5 and Figures 5.3 and 5.4. Table 5.5 lays out the sensitive bits across two dimensions where the rows are the processors and the columns are the benchmarks. Simple statistics are provided for each row and each column as well (arithmetic mean and difference between maximum and minimum results). Figures 5.3 and 5.4 are shown as two separate plots to show provide two perspectives of the results: grouped by processor (Figure 5.3) and grouped by benchmark (Figure 5.4). Like all previous experiment results, each sensitive bit count is halved to represent a single processor on a DUT (each DUT contains two processors as described in Section 3.3).
Table 5.5: Sensitivity Results for Performance Processor Designs Implementations Running on Multiple Benchmarks

<table>
<thead>
<tr>
<th>Sensitive Bits</th>
<th>Benchmarks</th>
<th>Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>Dhrystone</td>
<td>qsort</td>
</tr>
<tr>
<td>MicroBlaze</td>
<td>107,115</td>
<td>109,120</td>
</tr>
<tr>
<td>LEON3</td>
<td>176,440</td>
<td>183,076</td>
</tr>
<tr>
<td>Cortex-M0</td>
<td>164,419</td>
<td>94,511</td>
</tr>
<tr>
<td>Average</td>
<td>149,325</td>
<td>128,902</td>
</tr>
<tr>
<td>Max-Min Δ</td>
<td>69,325</td>
<td>88,565</td>
</tr>
</tbody>
</table>

Figure 5.3: Sensitivity Results for Performance Processor Designs Implementations Running on Multiple Benchmarks (Grouped by Processor)

Over all twelve runs, the average result is 149,542 sensitive bits with the maximum result being the Cortex-M0 running `basicmath` at 187,500 sensitive bits and the minimum result also being the Cortex-M0 running `qsort` at 94,511 sensitive bits. The Cortex-M0 shows the highest variance of all four processors where the lowest result is nearly half of the maximum result (visible on Cortex-M0 on Figure 5.3). The LEON3 shows the least variance with the maximum result only being 7,142 bits greater than the minimum result (“Max-Min” column in Table 5.5). The LEON3 may have the least variance because the processor runs a bootloader sequence before it
begins executing the benchmark. If there was no bootloader sequence, it is likely that the LEON3 would show additional sensitivity because it would spend more time running the benchmark. The MicroBlaze is the least sensitive soft processor with an average of 112,207 sensitive bits and the most sensitive soft processor is the LEON3 with an average of 179,771 bits. By comparing the LEON3 and the Cortex-M0 on Figure 5.3 one can see that their sensitivities are similar except for the qsort benchmark. The FPU benchmarks indicate that the Cortex-M0 is more sensitive than the LEON3, but the integer-only benchmarks indicate that the LEON3 is more sensitive than the Cortex-M0. These differences could be due to sensitivity differences of the processor FPUs.

A different perspective for viewing the results is seen when looking at the column results in Table 5.5 and benchmark groups in Figure 5.4. Benchmark basicmath records an average of 165,996 sensitive bits and is more sensitive that the other benchmarks, possibly due to the variety of operations used, and the use of the FPU. The Whetstone benchmark is the next most sensitive benchmark and it also uses the FPU. Dhrystone and qsort are both integer-only benchmarks and the are the least sensitive with qsort averaging the least at 128,902 sensitive bits and Dhrystone having the smaller “signature” (Figure 5.4) when not including the Cortex-M0 result. Measurements of
variance (difference between maximum and minimum) for the different benchmarks are similar to each other with basicmath having the lowest variance with 60,365 sensitive bits (with 165,996 average) and quicksort having the greatest variance at 88,656 sensitive bits (with 128,902 average).

The results of this experiment can be compared to previous experiments. When compared with the minimalistic design results of Section 5.2, the designs in this experiment show more sensitivity due to a larger processor size and benchmark complexity. The MicroBlaze experiments from Section 4.4 have the Dhrystone and Whetstone benchmarks in common with this experiment and show more variance in sensitivity between the benchmarks. The processors for these experiments show less benchmark variance from previous designs because they have additional modules including communication UART.

The normalized sensitivities of the processors are shown in Table 5.6 and Figure 5.5. The processor with the highest normalized sensitivity is the Cortex-M0 which measured as high as 6.67%. The processor with the lowest normalized sensitivity is the LEON3 which measured as high as 1.71%. The MicroBlaze measurement is between the other processors with a maximum normalized sensitivity of 4.09%. The LEON3 could have the lowest sensitivity because it is the largest design. The Cortex-M0 and the MicroBlaze are similar-sized designs but the MicroBlaze uses DSP48E slices for integer and float multipliers which are less sensitive than the LUT-implemented multipliers used in the Cortex-M0.

<table>
<thead>
<tr>
<th>Processors</th>
<th>Dhrystone</th>
<th>qsort</th>
<th>Whetstone</th>
<th>basicmath</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroBlaze</td>
<td>3.45%</td>
<td>3.51%</td>
<td>3.40%</td>
<td>4.09%</td>
</tr>
<tr>
<td>LEON3</td>
<td>1.63%</td>
<td>1.71%</td>
<td>1.63%</td>
<td>1.71%</td>
</tr>
<tr>
<td>Cortex-M0</td>
<td>5.85%</td>
<td>3.36%</td>
<td>6.41%</td>
<td>6.67%</td>
</tr>
</tbody>
</table>

This section combined multiple processors and multiple benchmarks in an experiment where processor configurations used were larger than the configurations for previous experiments. The Cortex-M0 had the highest number of sensitive bits as well as the highest normalized sensitivity. The MicroBlaze had the lowest raw sensitivity and the LEON3 had the lowest normalized
sensitivity. Soft processors were most sensitive when running the basicmath benchmark which had the highest number of sensitive bits for all three processors. When compared to other soft processors, the MicroBlaze appears to be the best candidate for use in a high-radiation environment because of its ability to execute the benchmarks with a smaller design size and a lower sensitivity.

In this chapter, five different soft processors were tested to contrast their sensitivity and normalized sensitivity. The first set of experiments tested minimal configurations of the five processors and found that the MicroBlaze had the lowest design sensitivity (even though it had the highest normalized sensitivity due to its design). The second set of experiments test only three processors with larger configurations and found that the MicroBlaze is also the least sensitive design over the benchmarks tested except for the qsort benchmark.
CHAPTER 6. CONCLUSION AND FUTURE WORK

In this thesis, a high-speed FPGA fault injection system is presented along with a methodology that can exhaustively test the sensitivity of a design’s bitstream to SEUs and, optionally, the system can be used for radiation testing. While other fault injectors may share similar attributes or advantages, this fault injector is unique and well-suited for soft processor fault injection. The fault injector can continue to support future research using the latest space-grade FPGAs and can be adapted for future XRTC motherboards and daughtercards. Software written for the host machine can create, schedule, and analyze fault injection campaigns and can be adapted for use in future systems.

Also, the results of fault injection experiments on multiple soft processor designs are presented in the thesis, and it was determined that soft processor SEU sensitivities can vary significantly depending on the type of processor, its features, and the programs it runs. Of the four major soft processors that were tested, the MicroBlaze had the lowest sensitivity and the smallest size. The MicroBlaze soft processor was tested in a variety of configurations, and it was found that its features, such as an FPU and additional memory, added to the sensitivity of a soft processor’s design, irrespective of whether it was required by the executing software. Variations in MicroBlaze placements and placement constraints also can improve or worsen the processor’s sensitivity. In addition, multiple benchmarks were run during fault injection, and it was found that they can lower the sensitivity of the soft processor by as much as 50% because they exercise different components.

Over six billion configuration bits were injected in experiments, which required thousands of hours of testing. These experiments provided empirical data that can be used by practitioners in making their own estimates of a soft processor’s SEU sensitivity and for comparing trade-offs when choosing a soft processor or deciding the options that would best suit their purposes. A list of recommended guidelines for someone who is using soft processors and would like to minimize configuration memory sensitivity is provided below:
• The Xilinx MicroBlaze appears to be the least sensitive of all the soft processors used in this research.

• Pick the processor for the application. Over-sized soft processors add additional sensitivity to the FPGA design. If a PicoBlaze can be used, do not use a MicroBlaze.

• Configure the soft processor with the minimum set of features required for the application. Subsystems and components, such as FPU or UART, can impair design sensitivity when they are included but not utilized.

• Adding more BRAM to the design than is necessary also can increase design sensitivity.

• Be cautious with area constraints because they can reduce the design to a smaller number of slices, but they also can result in longer routed wire paths and potentially increase the total number of sensitive bits.

In the future, additional fault injection experiments may be designed and conducted to evaluate new soft processor and software fault mitigation techniques. Additional processors and peripherals can be tested to further characterize possible soft processor systems in the design space. Testing may be performed with soft processors running real-time operating systems (RTOS) or Linux. The fault injector used in this work can be extended to scrub only relevant BRAMs, SRLs, and LUTRAMs when recovering from a fault, which would speed up fault injection time considerably. The fault injector also can be extended to include support for newer FPGA daughtercards.

The usage and applicability of FPGAs and soft processors for space computing systems may be enhanced in the future. Fault injection enables rapid design cycle iterations for evaluating soft processor designs in which engineers apply fault tolerance techniques and compare designs. I expect that the demand for fast fault injection systems will increase. They will be used as a tool, not only by researchers, but also by government and commercial entities that are working to build the next wave of high-performance space computing systems.
REFERENCES


[37] G. Cieslewski and A. D. George, “SPFFI: Simple portable FPGA fault injector,” in Pro-
cedings of Military and Aerospace Programmable Logic Devices Conference (MAPLD),
Aug./Sep. 2009. 17, 26

[38] G. G. Cieslewski, A. Jacobs, A. D. George, and A. Gordon-Ross, “Multibit fault
injection for field-programmable gate arrays with simple, portable fault injector,” Jour-
http://dx.doi.org/10.2514/1.I010200 17

SHADES2: A platform for early evaluation of ASIC and FPGA dependability using par-
tial reconfiguration,” La Sociedad de Arquitectura y Tecnología de Computadores, 2012. 17, 26

v2_1/ug764_sem.pdf 17, 26

pdf 17

[42] D. White, “Considerations surrounding single event effects in FPGAs, ASICs, and
support/documentation/white_papers/wp402_SEE_Considerations.pdf 22

Ph.D. dissertation, Brigham Young University, April 2011. 25, 26


SHADES2: A novel platform for early evaluation of robustness against SEE,” in Radiation
and Its Effects on Components and Systems (RADECS), 2011 12th European Conference on,
Sep. 2011, pp. 169–174. 26


documentation/sw_manuals/xilinx13_2/mb_ref_guide.pdf 29

//doi.acm.org/10.1145/358274.358283 39

content/19/1/43.abstract 39


