2019-12-01

A Soft-Error Reliability Testing Platform for FPGA-Based Network Systems

Hayden Cole Rowberry
Brigham Young University

A Soft-Error Reliability Testing Platform

for FPGA-Based Network Systems

Hayden Cole Rowberry

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
Brad L. Hutchings
Jeffrey B. Goeders

Department of Electrical and Computer Engineering

Brigham Young University

ABSTRACT

A Soft-Error Reliability Testing Platform
for FPGA-Based Network Systems

Hayden Cole Rowberry
Department of Electrical and Computer Engineering, BYU
Master of Science

FPGAs are frequently used in network systems to provide the performance and flexibility that is required of modern computer networks while allowing network vendors to bring products to market quickly. Like all electronic devices, FPGAs are vulnerable to ionizing radiation which can cause applications operating on an FPGA to fail. These low-level failures can have a wide range of negative effects on the performance of a network system. As computer networks play a larger role in modern society, it becomes increasingly important that these soft errors are addressed in the design of network systems.

This work presents a framework for testing the soft-error reliability of FPGA-based networking systems. The framework consists of the NetFPGA development board, a custom traffic generator, and a custom high-speed JTAG configuration device. The NetFPGA development board is versatile and can be used to implement a wide range of network applications. The traffic generator is used to exercise the network system on the NetFPGA and to determine the health of that system. The JTAG configuration device is used to manage reliability experiments, to perform fault injection into the FPGA, and to monitor the NetFPGA during radiation tests.

This thesis includes soft-error reliability tests that were performed on an Ethernet switch network system. Using both fault injection and accelerate radiation testing, the soft error sensitivity of the Ethernet switch was measured. The Ethernet switch design was then mitigated using triple module redundancy and duplication with compare. These mitigated designs were also tested and compared against the baseline design. Radiation testing shows that TMR provides a 5.05× improvement in reliability over the baseline design. DWC provides a 5.22× improvement in detectability over the baseline design without reducing the reliability of the system.

Keywords: FPGA, computer networking, soft-error reliability, single-event upset, traffic generator, triple modular redundancy, duplication with compare, neutron radiation testing, fault injection, NetFPGA, Ethernet switch

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who have assisted me and made this thesis possible. In particular, I would like to thank my advisor, Dr. Michael Wirthlin, for granting me the opportunity to perform research and for encouraging me along the way. He has sacrificed countless hours guiding me in my research and and assisting with my writing. I am also grateful to the other members of my committee, Dr. Jeff Goeders and Dr. Brad Hutchings, for their support and for the knowledge and experiences they have helped me to obtain.

My family has been a great support throughout this journey. I am so grateful for the love and support of my wife, Elizabeth. She has always been so patient and understanding whether I'm traveling for beam tests or working late on projects. My parents and siblings have also been a great support.

I am grateful for my friends and colleagues in the BYU Configurable Computing Lab. In particular I'd like to recognize Andrew Keller, Jordan Anderson, Corbin Thurlow, and Jared Anderson. They have always been willing to lend a sympathetic ear and to offer whatever assistance they could. I have learned so much from being able to work alongside them.

This work was supported by the I/UCRC Program of the National Science Foundation under Grant No. 1738550 through the NSF Center for Space, High-Performance, and Resilient Computing (SHREC), by LANSCE proposal NS-2018-8031-A, and by Cisco Systems, Inc.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Computer networks have played a significant role in increasing human productivity and shaping modern society. At its core, a computer network is used to facilitate the communication and sharing of resources between computers. Within the past couple decades, there has been remarkable growth in the capabilities of computer networks. These networks are now used to connect individuals across the world and have enabled commonplace long-distance communication and the sharing of large amounts of data. As these networks play a larger role within organizations and between individuals, the demand on these networks also increases significantly.

To support this demand, there are three primary requirements for a modern computer network: high bandwidth, low latency, and high reliability. Bandwidth and latency are indicators of network speed and are key to providing high performance. As networks play a growing role in society, it is also important to maintain high levels of reliability. Financial trading service centers, telecommunication services, and data centers are examples of a few critical industries that require high levels of reliability [1]. Any prolonged network downtime could result in significant loss of productivity and money. Therefore, great care is taken in the design of network systems and architectures to minimize the occurrence of network failure.

Networked communication is backed by network systems that interconnect through cables, fiber-optic connections, and wireless links. These network systems primarily consist of routers and switches which are responsible for routing traffic between networks and hosts. Switches direct traffic within a local network while routers are used to forward packets between different networks. These two different systems provide similar capabilities, but use different addressing schemes and routing techniques to ensure that traffic successfully arrives at its destination. Other network systems include firewalls, for blocking dangerous or unwanted network traffic, and bridges, which connects two network segments together.

Depending upon where a network system is deployed, it may be necessary to handle traffic in excess of 100 Gbps. To achieve the high-performance demands of these networks, networking hardware vendors frequently turn to field-programmable gate arrays (FPGAs) for use in their network systems [2]. FPGAs are programmable integrated circuits that provide high-bandwidth connections and customizeable high-speed data processing logic. Because FPGAs can be easily reprogrammed, the product development time of an FPGA is relatively short. Additionally, an FPGA-based system provides the flexibility that makes it possible to correct hardware bugs even after a product has been deployed. Because network standards and requirements evolve rapidly, it is important for a network vendors to be able to bring products to market quickly in order to stay competitive. Although FPGAs are often more expensive than other technologies, they are frequently chosen because they allow faster product development.

Like all electronic devices, FPGAs are susceptible to failure due to ionizing radiation. While this radiation rarely causes any permanent damage to a hardware device, it has the potential to corrupt the data or programming of an FPGA. These upsets are rare and may only occur once within a 10 to 20 year period for a single device. However, in large deployments of FPGA-based devices, these upsets are encountered more frequently and could become problematic.

While the effects of radiation on an FPGA have been studied and are well understood, it is not as clear how these failures affect a network system as a whole. Radiation-induced FPGA failure has been observed within network systems, but the results are often unpredictable and often cannot be detected using conventional means [3]. More research is needed to understand the effects of radiation failure on network systems and to determine what techniques are effective in mitigating these failures.

To facilitate the study of FPGA reliability in networking systems, this thesis presents a framework for performing soft-error reliability tests on FPGA-based networking applications. This framework consists of a network-development platform which can be programmed to implement a network system. Additionally, this thesis introduces a novel traffic generator which can be used to exercise the experimental network system to detect and categorize the slightest network failure. And finally, the framework uses a custom device to monitor the network system and to simulate the effects of radiation-induced upsets.

This framework is then used to collect reliability measurements for an Ethernet switch design using both simulated fault injection and an accelerated radiation environment. The purpose of these tests are two-fold: first, to identify the failure modes of radiation-induced upsets on a network system, and second, to estimate the rate at which these failures would actually occur. Additionally, two common forms of error mitigation will be applied to a test design to determine the effectiveness of these techniques.

This thesis will first present background on the role of FPGAs within network applications and the soft-error reliability issues of FPGAs in Chapter 2. Chapter 3 introduces the soft-error reliability testing framework and provides an overview of its components. This chapter also shows how each of these components works together to test the soft-error sensitivity of a network application. The architecture of an Ethernet switch, which is used as the test design for this work, is described in Chapter 4. Then, Chapter 5 shows how two common soft-error mitigation techniques, triple modular redundancy (TMR) and duplication with compare (DWC), are applied to this test design. Chapter 6 describes the architecture of the traffic generator that is used to exercise and determine the health of the network system as it is being tested. The methodology for testing the reliability of the Ethernet switch will be discussed in Chapter 7. This will describe how both fault injection and neutron irradiation experiments are conducted. Additionally, this chapter will discuss the results from these experiments. This chapter will categorize the failure modes that were observed during the tests and will show how TMR and DWC improve soft-error reliability and detectability, respectively. This thesis concludes in Chapter 8.

CHAPTER 2. BACKGROUND

Modern computer networks are expected to process an enormous amount of service requests while maintaining high levels of reliability to satisfy customer expectations. Network vendors often turn to FPGAs when designing network systems in order to provide high levels of performance and reduce development time. However, SRAM-based FPGAs are susceptible to radiation effects which can cause significant downtime for these complex networking systems. While radiation effects are traditionally associated with aerospace applications, terrestrial radiation still provides a significant threat to the wide-scale deployment of high-availability network devices. The purpose of this chapter is to motivate the need for soft-error mitigation and testing in FPGA-based network systems.

2.1 Network Systems

A computer network is a collection of electronic devices that are interconnected for the purpose of sharing resources. These electronic devices are known as network nodes, and they can transmit, receive, and relay data depending on their role and location within the network. Nodes are connected together through links which can take many different forms including twisted pair Ethernet cables, wireless transmission, and fiber optic cables. Nodes and links can be arranged into a number of different topologies to meet performance, reliability, and physical requirements.

The most common network nodes are network interface cards (NICs), Ethernet switches, and Internet Protocol (IP) routers. NICs are used to connect computers to networks and they handle low-level signal transmission. Additionally, NICs allow a processor to offload network-related tasks to reduce processor overhead and provide increased network performance. Switches are then used to connect multiple hosts together within the same local network. Ethernet switches use media access control (MAC) addresses to identify individual hosts to provide packet routing on a local scale. Routers are used to connect multiple networks together and are concerned with

determining the best route for network traffic. Routers use IP addresses to identify network hosts and utilize routing protocols to determine how to relay data to its destination. As data is transmitted between hosts, packets may pass through many network nodes and links.

It is important that each of these components provides sufficient performance and reliability to support the computer network as a whole. Reliability ensures that data is relayed between hosts correctly. Even with reliability-enhancing protocols, frequent data loss or corruption will degrade the quality of a network. Network performance is primarily measured using two metrics: bandwidth and latency. Bandwidth refers to the amount of data that can pass through a connection within a given period of time (reported in bits-per-seconds). Low bandwidth will result in increased transmission time when transferring large files between hosts. Latency refers to the amount of time that is required for data to traverse the length of connection. High latency could present challenges to real-time applications (video conferencing, for example). A failure in any of these areas could degrade the quality of a computer network.

2.2 Network System Hardware Design

When designing a network system, engineers need to balance many factors when selecting components for that product. These factors typically include performance, development time, flexibility, and cost. There are three types of technology that designers traditionally use when designing a network system: application specific integrated circuits (ASICs), microprocessors, and FPGAs [4]. Each of these technologies offer a unique combination of advantages and weaknesses that must be carefully matched to the needs of the product. Frequently, a product will use all of these technologies at the same time to meet its requirements.

An ASIC is a customized integrated circuit that is designed for a specific purpose, as opposed to a general integrated circuit that can perform a wide variety of tasks. These chips consist of thousands of logic circuits that have been arranged to allow data to be pipelined and for many operations to occur in parallel and at high speeds. These benefits result in superior processing performance, making them attractive candidates for network applications that require the greatest performance [4]. However, some of the challenges associated with using ASICs in network systems is their long development cycle and lack of flexibility. These challenges are problematic in

an industry that is rapidly evolving [5]. Given the expense and time associated with ASIC development, ASICs are best suited to applications that are primarily static and can be widely deployed.

Microprocessors, on the other hand, excel in general purpose computation that can be utilized by software-based network applications. Software is far easier and faster to develop than hardware and can be updated to support new technologies. Additionally, microprocessors are readily accessible and are available at a relatively low cost. However, this flexibility comes at the expense of performance. A CPU's packet processing functionality is severely limited by the sequential nature of software processing [4]. Link speeds are increasing faster than the packet processing capabilities of microprocessors. This widening gap presents significant issues for using microprocessors in the datapath of high-end network systems [6].

An FPGA is a programmable integrated circuit that can be used to implement any logic function. They are often seen as a product that falls between a CPU and an ASIC with regard to computation power, flexibility, and development difficulty [7]. Like an ASIC, an FPGA can benefit from circuit specialization and data pipelining that provide improved performance over a CPU. However, an FPGA will not achieve the same levels of performance as an ASIC. FPGAs are reconfigurable which would allow upgrades or bug-fixes to be made to a network system even after deployment, just like a CPU-based product. While programming FPGAs is generally more difficult than programming a processor, an FPGA design can be created and tested much faster than an equivalent ASIC design. This provides an FPGA-based system with a reduced time to market, which is important in a fast-moving industry [8]. This balance of performance and flexibility makes FPGAs an appealing technology for designing network systems [4].

An FPGA's high-bandwidth connections and large programmable fabric provide it with ample processing capabilities to handle the high-bandwidth low-latency processing demands of high-end networking systems. FPGAs provide numerous traditional I/O connections in addition to high-speed serial I/O transceivers that are beneficial for high-bandwidth applications. For example, one of the largest parts in the Vertex UltraScale+ product family (XCVU13P) offers 128 multi-gigabit serial transceivers, providing a peak bidirectional bandwidth of 8,384 Gb/s. Additionally, this part also provides 832 traditional I/O ports for low speed connections [9]. This is enough bandwidth to support multiple 100 Gb/s or 400 Gb/s Ethernet connections. Being able to provide multiple high-bandwidth connections is an important requirement for most network systems.

Not only do FPGAs excel in providing high-bandwidth connections for moving data to and from the chip, they also provide sufficient resources for processing within the device. While the operations of a network system are device-specific, it is important that the network system be able to process traffic at high speeds in order to minimize network latency. The processing resources aboard the FPGA provide sufficient logic for inspecting and processing multiple streams of data in parallel and at line rate. FPGAs contain programmable logic cells along with internal memories that can be used together to implement any logic circuit. For example, the Virtex UltraScale+ part mentioned earlier offers 455 Mbit of high-speed internal memories and 3.78 million logic cells [9]. Assuming the FPGA is operating at 200 MHz, this device offers a raw memory bandwidth in excess of 70 Tbps. These logic cells can be used to create custom datapaths that are piplined and parallelized and that are capable of processing high-bandwidth traffic while providing minimal latency.

Because the logic on an FPGA is reprogrammable, network systems that use FPGAs enjoy flexibility that ASIC-based systems lack. This provides network systems with a certain degree of future-proofing, meaning that an already-launched product could support a new feature with a relatively simple upgrade [10]. Additionally, if an error is discovered in the system design after the product is launched, the FPGA design can be updated to correct the error. This same level of configurability would not be achievable with a custom silicon device.

FPGAs have been used in network systems extensively within academic settings [11]–[13]. The NetFPGA project is an example of an FPGA-based network system framework that was designed to facilitate the prototyping of high bandwidth devices [14]. This project contains several open-source reference and contributed network system designs ranging from NICs to Ethernet switches and software-defined network routers [11]. These designs demonstrate the role that FPGAs can play within network systems.

An example NetFPGA design, the 10G Ethernet reference switch, is diagrammed in Figure 2.1. In this design, the FPGA has been programmed to handle both the control plane (to determine how data should be routed) and the data plane (the actual routing of network traffic). The FPGA design includes Ethernet MACs to provide data framing and error detection, routing tables to control how traffic is directed, and a PCI Express controller for host system integration. The network system is capable of operating in a stand-alone mode or as a peripheral component in

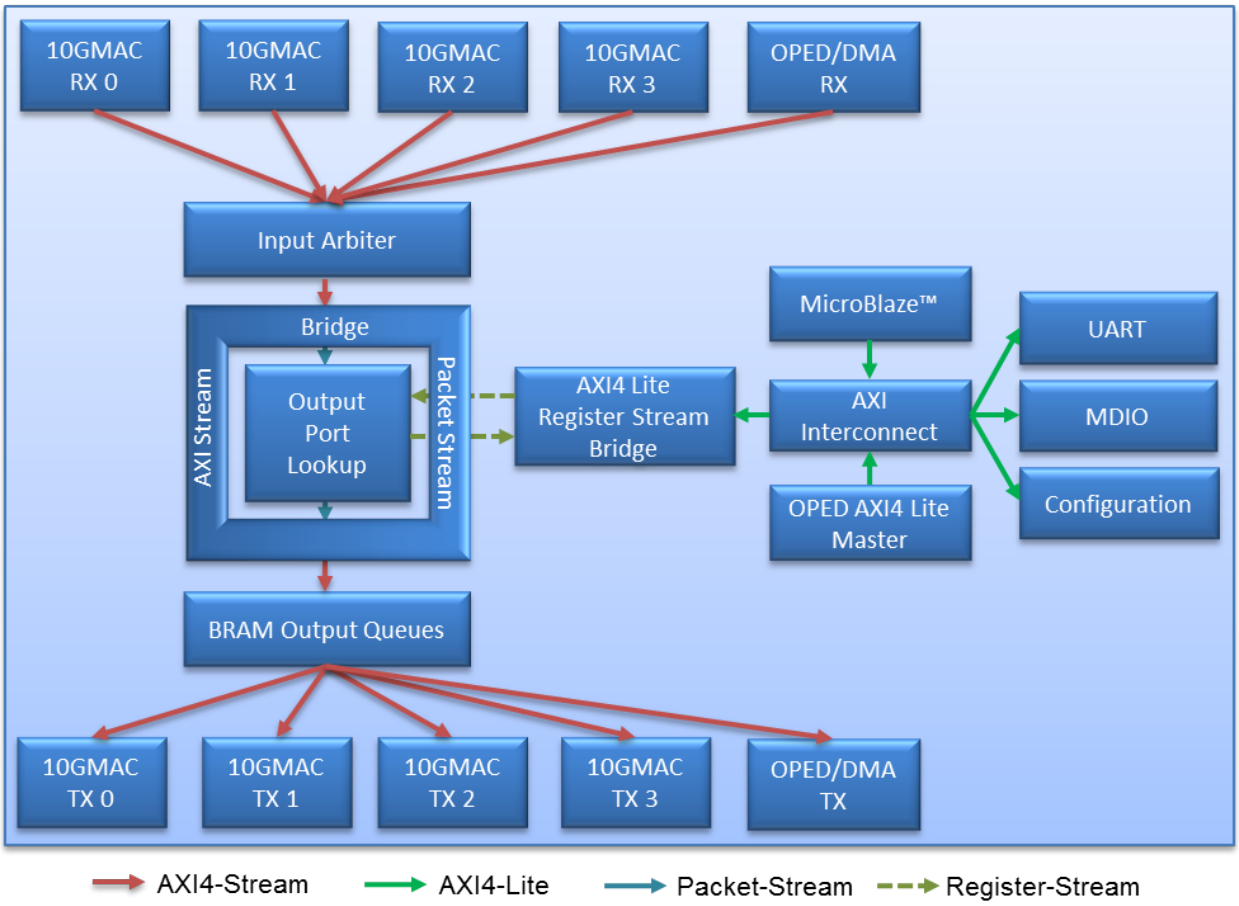

Figure 2.1: NetFPGA 10G Ethernet Switch Block Diagram [15]

a host computer system. This switch offers high-performance packet processing that approaches line rate (40 Gbps across all four ports). This example design shows the capabilities of FPGAs in implementing complete high-performance network systems at the data link, network, or even upper-level layers.

Cisco Systems, Inc. is a large network vendor that uses FPGAs in their high-end networking systems, including their Catalyst series switches and their Aggregation Services Routers. Although Cisco does not publicly disclose the internal designs of the products, they state in [16] that "[FPGAs] are crucial for the function of route processors, line cards, shared port adapters (SPAs), SPA Interface Processors (SIPs), and fan trays." They periodically release upgrades for the FPGAs in their products which contain bug fixes and new features [16]. These statements suggest that FPGAs play an important role in Cisco's products and that these products benefit from the reconfigurability of FPGAs.

Flexible hardware does not only benefit the network vendor, but it can also be utilized by the end user. Domain-specific languages such as PX and P4 have been developed to allow a network engineer to easily describe the functionality of their network at a high level. P4 code would then be compiled to create a bitstream for the FPGAs contained in the network appliance [17]. This allows the data plane of the network system to be customized from ingress to egress, with all data processing occurring at line rate. These firmware changes can even occur while the device stay online, maintaining a high-availability network [18].

A few hardware vendors are producing network systems that allow an end-user to access the system's FPGAs. The Arista 7130 FPGA-enabled network switches allow a developer to run their performance critical applications directly within the switch. These designs are given access to "Layer 1+" traffic and can use this to generate statistics, mirror traffic, perform data conversion, and perform link patching, just to provide a few examples [19]. Juniper has released the QFX5100 Series application acceleration switch which was designed for the financial industry. Financial firms can customize the FPGA for their own applications that require low latency, such as high frequency trading [20]. As the demand for high-bandwidth low-latency network applications increase, the use of FPGAs in network system will likely continue to grow.

Unfortunately, the high-end FPGAs that are most capable of high-bandwidth traffic processing are more expensive per-unit than ASICs or microprocessors. Despite this expense, several large network vendors use FPGAs in their products, suggesting that FPGAs are an effective choice for high-end network systems. Because network standards evolve rapidly [5] and there is demand for improved network performance [21], network system vendors must iterate quickly to stay competitive. FPGA-based products can be brought to market faster than ASICs due to reduced verification time and production delay [8]. This improved time-to-market likely provides economic incentive that make FPGAs worthwhile for use in high-end network systems.

2.3 Terrestrial Radiation

Like all semiconductor devices, FPGAs are susceptible to ionizing radiation. This radiation effects the memories of electronic devices and can cause the device to fail. Radiation-induced failure modes can range from data corruption and temporary disruption of service to complete system failure [3]. FPGAs require a large amount of configuration memory that other technologies

do not need. Because this configuration memory is susceptible to ionizing radiation, FPGAs are more likely to experience radiation-induced upsets than other technologies.

While the atmosphere does provide protection against most high-energy radiation, it is not a perfect barrier [22]. This makes terrestrial radiation a concern for applications that require high levels of reliability. Additionally, as semiconductor feature size shrinks and operating voltages decrease, the capacitance of a memory cell decreases. This makes memory cells vulnerable to more types of particles and particles of lower energy [23], causing the effects of radiation induced failure to become more pronounced [24].

The primary source of terrestrial radiation is a byproduct of galactic cosmic rays striking the atmosphere. When these rays interact with the atmosphere, a complex cascade of particles is produced. While many of these particles (such as muons and pions) are short lived, high energy neutrons do reach ground level and are the most likely form of cosmic radiation to cause upsets in terrestrial devices [22]. The neutron flux of New York City at sea level has been measured to be $12-14$ neutrons$/\text{cm}^2 \cdot \text{hr}$ [25]. This means that for a surface that is one square centimeter in size, a high-energy neutron will pass through every four to five minutes on average. Neutron flux is highly dependent upon a number of factors, particularly elevation. Devices deployed at higher elevations will receive more neutron radiation than those at lower elevations.

The interaction between neutrons and silicon is complex, and ionization is caused through induced silicon recoil [26]. Neutrons collide with silicon nuclei, causing them to break into smaller fragments, which then generate change and can induce faults within the device. As neutrons lack electrical charge, it is extremely challenging to provide effective shielding. Concrete has been shown to slow high-energy neutrons and reduce the probability of neutron induced upsets [26]. However, as physical shielding is not a practical solution for most devices, terrestrial systems need to anticipate the effects of ionizing radiation.

2.4 Radiations Effects in FPGAs

Ionizing radiation can affect semiconductor devices through both long-term exposure and single event effects (SEE). Long-term exposure resulting in total ionizing dose is typically a non-issue for terrestrial applications due to generally low radiation exposure levels. SEEs, on the other hand, are still a notable concern for terrestrial devices and are defined as any noticeable change

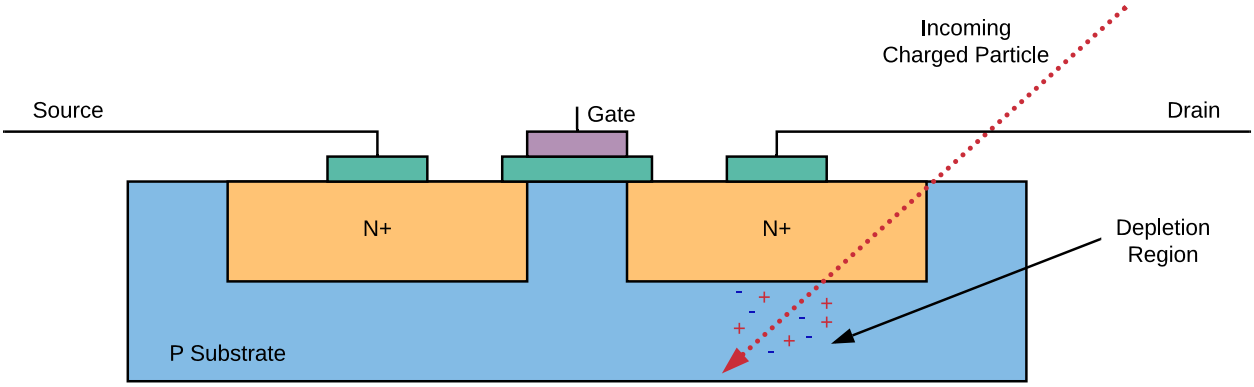

Figure 2.2: Single Event Effect Resulting from a High-Energy Particle

in state or operation of a microelectronic device resulting from a single energetic particle strike. As an energetic particle traverses through the sensitive region of a transistor, it generates are large number of electron-hole pairs (as shown in Figure 2.2). These charges may accumulate at the node of the transistor and create a spurious current spike within the device. This current spike translates to a voltage spike, known as a single event transient (SET). SETs are short lived and are generally harmless to the device. However, similar to an electronic glitch, an SET could be latched into a memory element and corrupt application or control data [22].

When an SEE affects a memory element, the error is categorized as a single event upset (SEU), which is colloquially known as a "bit flip." SEUs are further categorized into single-bit upsets (SBU) and multi-bit upsets (MBU) depending on the number of memory elements that are upset by the single particle. An SEU does not damage the device and has no lasting consequence. However, an SEU-induced error may persist within the affected device until the corrupted data is either propagated out of the system or is somehow repaired. In an FPGA, an SEU could corrupt user block memory, flip-flop state, and configuration memory.

Configuration memory provides an FPGA with its ability to be reprogrammed and is used to store all of the necessary information to define how the FPGA behaves. An FPGA consists of a large array of configurable logic elements interconnected by programmable connections as shown in Figure 2.3. The logic elements typically consist of programmable lookup tables (LUT) capable of implementing any given logic function in addition to flip-flops for storing sequential logic. These logic elements can be tied together to build large and complex circuits. Routing between logic elements sn programmable and is controlled by switch blocks. These switch blocks sit at the

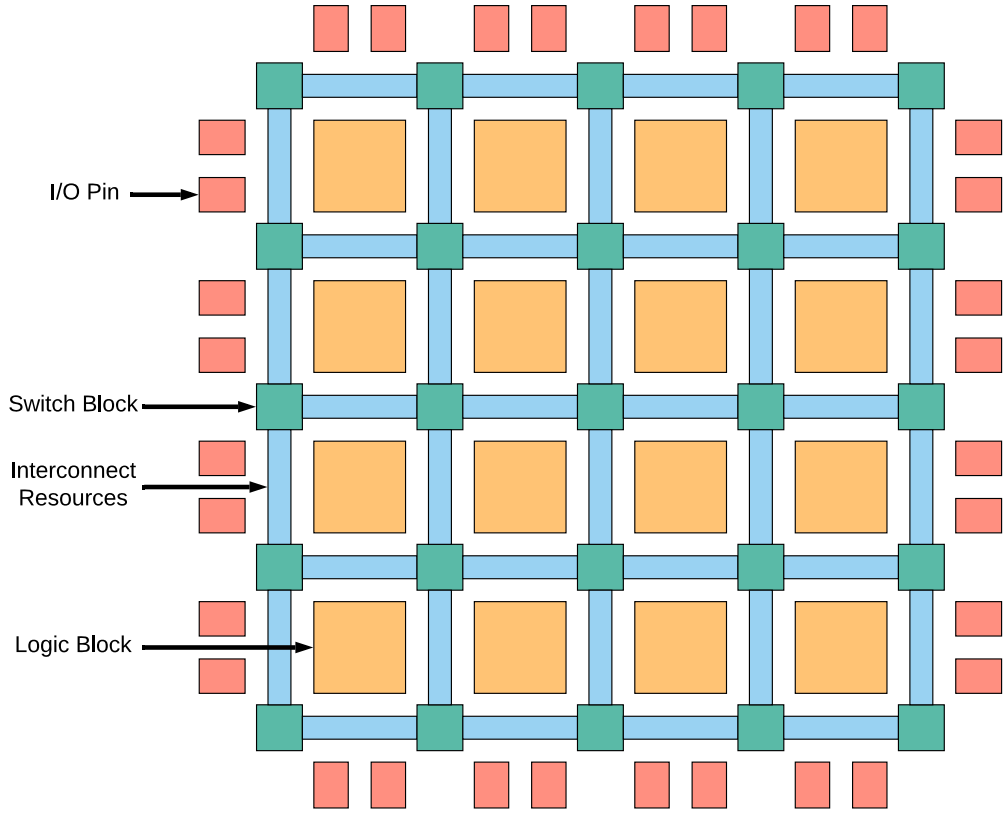

Figure 2.3: Traditional Island-Style FPGA Architecture

intersections between connections and are used to determine which wires are to be connected or remain separated.

When an upset occurs within an FPGA's configuration memory, a corresponding functional change occurs within the FPGA's programmable fabric. Upsetting a bit corresponding with a logic element would potentially change the logic function of the LUT, modify the behavior of a flip-flop, or affect internal routing. A routing bit upset could disconnect an essential signal, or introduce connections between signals that ought to remain independent. Although only a small portion configuration bits are likely to affect the operation of a design, the detrimental impact of these upsets can cause grants importance to their study. Unlike data upsets, which have a chance of naturally propagate out of the system or can be reset, upsets in configuration persist until they are repaired.

These upsets in configuration memory can have a significant impact on the operation of FPGA-based network systems. While the likelihood of an upset impacting a single system is small, the effects of radiation must be considered for large-scale deployment. As computer networks grow

and more network systems are deployed, the likelihood of an upset occurring within any system also grows. Soft-error mitigation is needed ensure that FPGA-based network systems can provide the high levels of availability that network operators demand.

CHAPTER 3. NETWORK RELIABILITY TESTING FRAMEWORK

Although the effects of SEUs on electronic devices is well documented and understood, it is not as clear how FPGA-based network systems fail when upset by a soft error. This thesis presents a framework to facilitate the experimentation of SEUs in FPGA-based network systems to better understand the effect that upsets have on these systems. This framework provides a controlled environment for observing the failure modes of a network system and determining what mechanisms are required for its recovery. Additionally, the framework can be used in both fault injection and accelerated radiation experiments to estimate the soft-error sensitivity of an FPGA-based network design. This chapter will introduce the network reliability testing framework by providing a high-level overview of its components and discussing how they each interact.

3.1 Framework Overview

The objective of the network reliability testing framework is to enable the observation of soft-error effects on FPGA-based network systems. To achieve this objective, the framework provides a network system, a way to inject upsets into that system, and a mechanism for measuring the functional effects of the upset. The experimental network system for this framework is implemented on the NetFPGA hardware development platform. The BYU JTAG Configuration Manager (JCM) is used to perform fault injection in the network system while a custom traffic generator is used to observe the health of the network system. These components are interconnected, as shown in Figure 3.1, and work together to provide in-depth data collection on the effects of soft errors on a FPGA-based network system.

At the core of the framework is the FPGA-based network system to be tested, which will be referred to as the device under test (DUT). The NetFPGA development platform is flexible and can be programmed to implement a wide range of network systems, including network interface cards, switches, routers, and hardware firewalls. This thesis, however, will focus exclusively on an

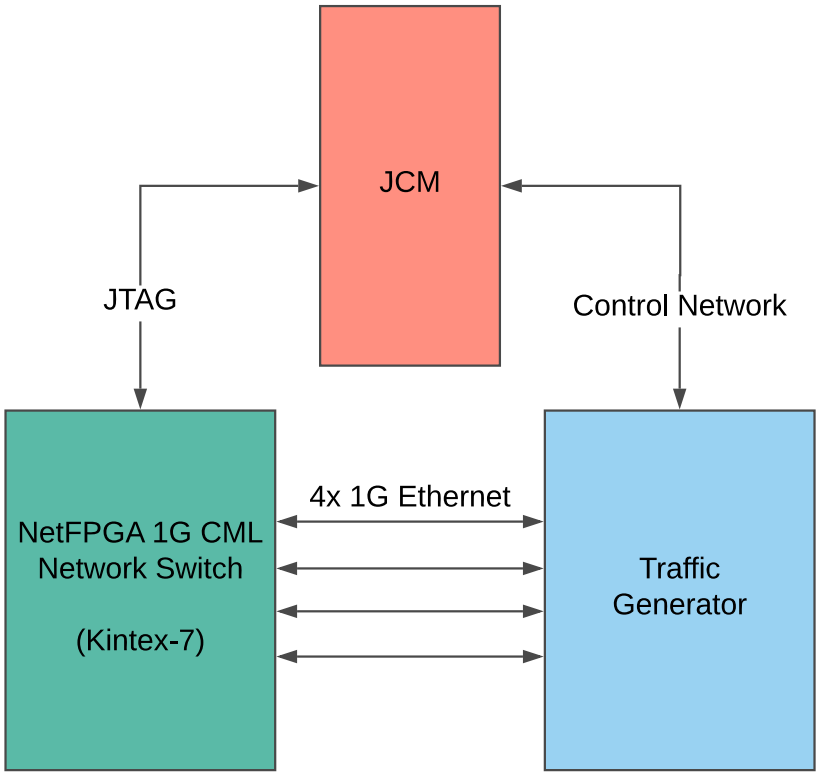

Figure 3.1: Reliability Test Framework Block Diagram

Ethernet switch design which is described in detail in Chapter 4. An Ethernet switch is a network system that is responsible for directing network traffic at a data-link layer, or in other words, between hosts within the same local network. In a typical deployment of an Ethernet switch, each Ethernet port would connect to a different network host. However, in the network reliability testing framework, each port of the Ethernet switch is connected to a port of the traffic generator.

The traffic generator is a custom-designed system that is used to monitor the health of the network system DUT during reliability testing. Details on the design of the traffic generator are provided in Chapter 6. The traffic generator is responsible for creating network packets to be processed by the Ethernet switch. Additionally, the traffic generator serves as a packet checker to ensure that the Ethernet switch is functioning properly. Any data corruption, misrouting, or packet loss will be detected by the traffic generator and will be reported as a failure condition. Status reports are provided to the JCM over a secondary control network that is independent of the network being tested.

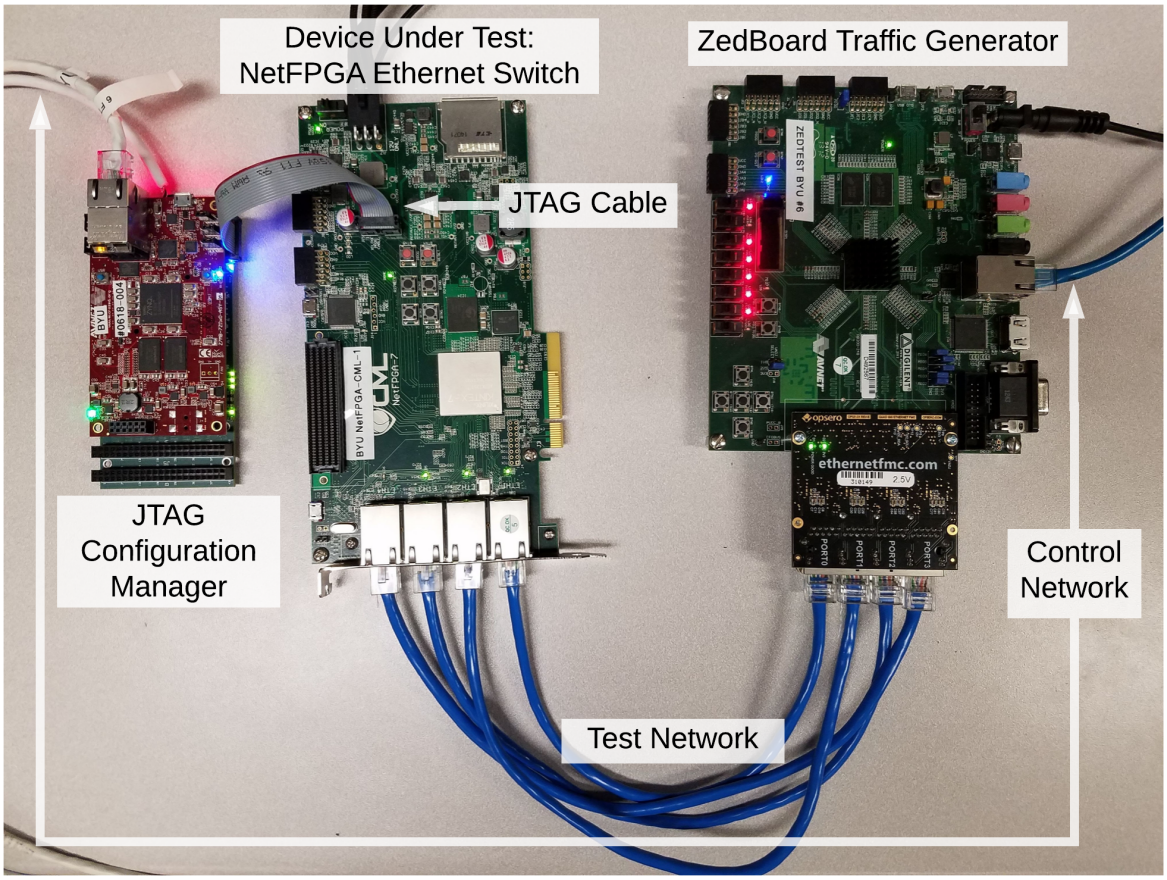

Figure 3.2: Network Reliability Testing Framework

The JCM is responsible for managing the reliability experiment. It communicates over the control network with the traffic generator to start and stop the generator as needed and to collect statistics. Additionally, the JCM is connected to the NetFPGA through JTAG. This connection can be used to access the configuration memory of the NetFPGA and to access statistic registers of the network system. The JCM uses this connection to insert upsets into the configuration memory for fault injection, or to provide configuration scrubbing during an accelerated radiation test.

Peripheral components can also be added to the network reliability testing framework, depending upon the needs of the specific experiment. A networked power supply is often used with the framework to allow the NetFPGA or traffic generator to be rebooted automatically if they were to crash during an experiment. During neutron irradiation experiments, a beam counter device is

often used to provide neutron fluence statistics. These devices are added to the control network and will be accessed by the JCM as needed during the experiment.

3.2 NetFPGA

NetFPGA is an open source project that was developed to support the research of FPGA-based network systems. This project has produced several FPGA rapid prototyping boards that are particularly well suited for developing network designs [14]. These development boards provide the I/O that is needed for network systems and memories that are helpful for buffering packets and storing network routes.

This framework uses the NetFPGA-1G-CML development board to run the experimental network designs. At the heart of the board is a Kintex-7 325T FPGA (xc7k325tffg676-1) which provides 326,080 logic cells, 16,020 Kb of block RAM, and up to 4,000 Kb of distributed RAM. Additionally, the board provides 4.5 MB of external SRAM which is suitable for storing forwarding table information and 512 MB of DDR3 DRAM which can be used for packet buffering. Rapid boot configuration is supported by a 128 MB BPI flash memory chip.

The board provides four Gigabit Ethernet ports, each connected to a 10/100/1000 Ethernet transceiver (PHY) that interfaces with the FPGA through a reduced gigabit media-independent interface (RGMII) connection. The development board is capable of interfacing with a host computer through a PCI Express connection, offering a bandwidth of 5 Gbps per lane and up to 4 total lanes. Additional expansion is enabled through a fully compliant high pin count FPGA Mezzanine Carrier (FMC) connector with GPIO and high-speed serial connections. The FMC port is capable of high-speed communication and could be used to provide additional communication, measurement, and control features. For example, this FMC expansion port could be used to provide small form-factor pluggable (SFP) transceiver ports for fiber optic connections, HDMI connectors for a video streaming application, or SATA connectors for a network-attached storage system.

The physical capabilities of this development board make it a suitable platform for developing and testing FPGA-based network designs. The NetFPGA community maintains an open-source repository of common network application hardware modules to assist in the development of a network system. The NetFPGA project provides provides a NIC, Ethernet switch, and IPv4 router as reference designs for these development boards [14]. Other, community-developed design include

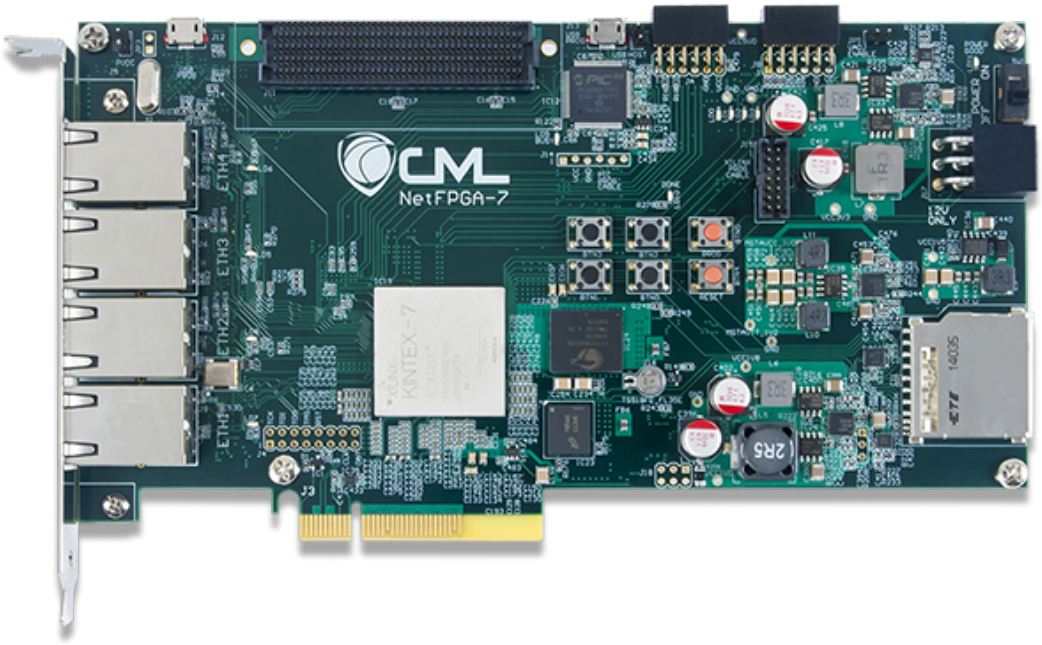

Figure 3.3: NetFPGA-1G-CML Development Board [27]

a traffic generator [28], a network intrusion detection system [29], a heavy-hitter detector [30], and a traffic monitoring system [31]. The NetFPGA platform provides system designers with the tools that are needed to develop complex network systems.

Using the open-source repository, an Ethernet switch was developed as an experimental design for this thesis. The switch works by inspecting the MAC addresses of Ethernet frames, looking up the address in a routing table, and forwarding the data to the correct port. This switch is self-learning which means that routes do not have to be programmed statically and that the routing table will be updated automatically as hosts come and go from the network. This design presents the potential for several failure-modes would be straight-forward to monitor. Details on the architecture of the Ethernet switch design are provided in Chapter 4.

3.3 Traffic Generator

The traffic generator is responsible for providing a load to be processed by the network system DUT. This network traffic is important for determining when the network system is operating properly and when it has failed. The traffic generator is configurable, allowing a user to specify

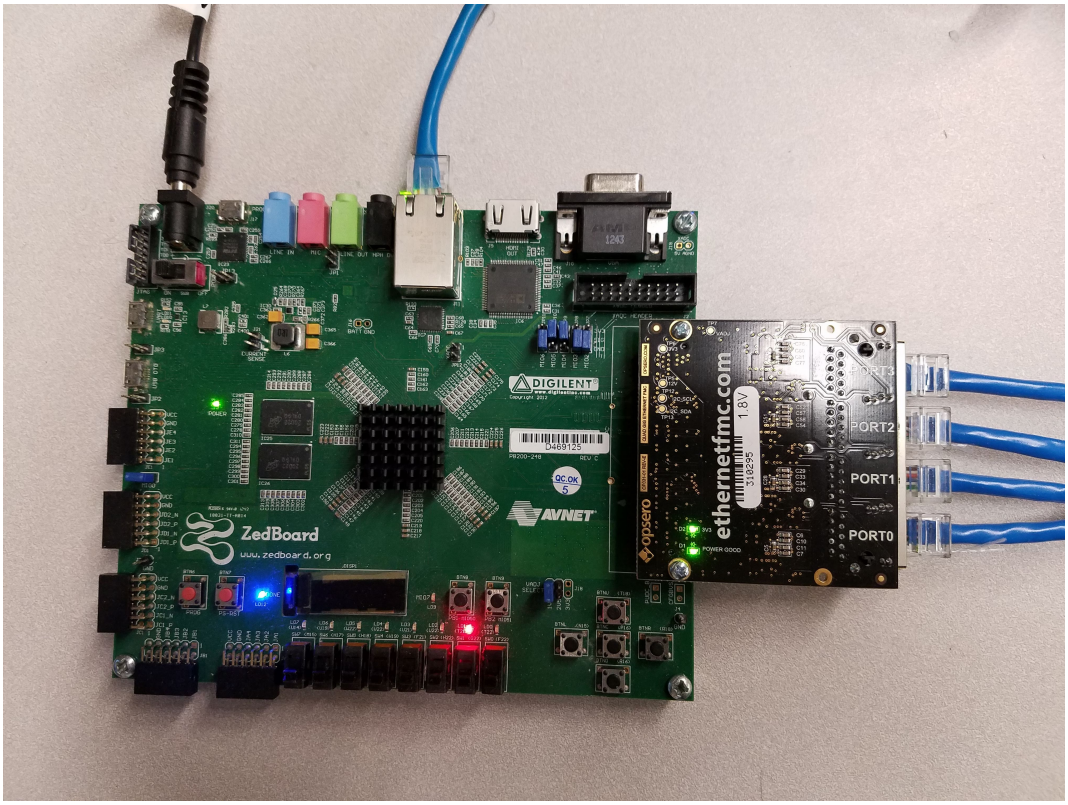

Figure 3.4: Traffic Generator Hardware

the headers, size, and frequency of packets being generated. This allows the network system to be tested under a variety of conditions. Additionally, the traffic generator has been designed to operate at line rates, providing a peak bandwidth of 4 Gbps.

The traffic generator also serves as a traffic checker which allows for failure detection. As the traffic checker receives return traffic from the network system, it attempts to match every packet that is received to one that had previously been transmitted by the generator. This allows the traffic checker to detect packets that have corrupt headers or data, packets that have been misrouted, and traffic that has been dropped. Details on the design and functionality of the traffic generator and checker are provided in Chapter 6.

The traffic generator was constructed using a ZedBoard and an EthernetFMC mezzanine card. The ZedBoard is a development board for the Xilinx Zynq-7000 SoC (XC7Z020-CLG484-1), which is a dual-core ARM Cortex-A9 processor coupled with programmable logic. The FPGA fabric provides 85,000 logic cells, 4.9 Mb of block RAM, and 200 I/O pins. The traffic generator and checker are implemented on the programmable logic in order to provide line-rate performance.

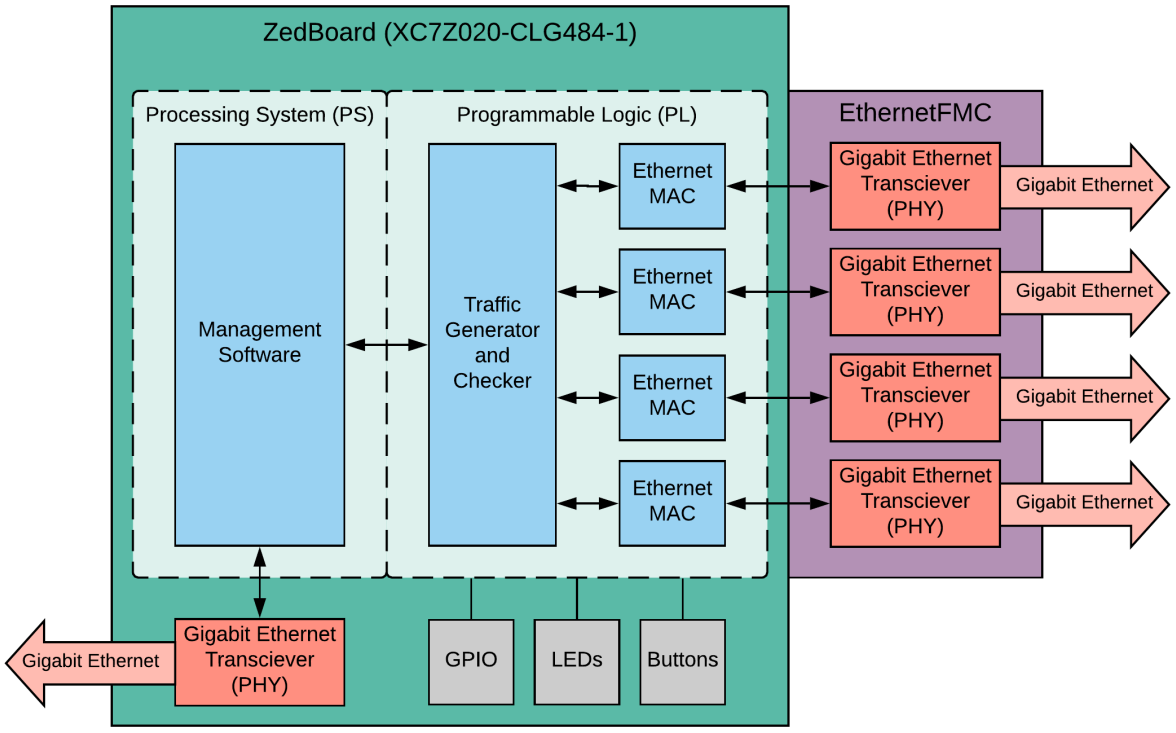

Figure 3.5: Traffic Generator Block Diagram

Additionally, the FPGA contains Ethernet MACs which are responsible for establishing a data-link with an adjacent network host. The processing system runs management software that configures the traffic generator and obtains status from the traffic checker.

The ZedBoard provides many different interfaces for supporting a wide variety of applications. The two most important interfaces used in this application are the Gigabit Ethernet port and low pin count FMC connector. The Gigabit Ethernet port is used to connect the ARM processor to the control network, allowing the traffic generator's management software to interact with the JCM. The FMC connector is used to connect peripheral devices requiring a large number of I/O to the programmable logic of the SoC. The traffic generator uses the FMC port to interface with the EthernetFMC card.

The EthernetFMC card from Opsero [32] is a Vita 57 standard compliant low pin count FMC card that is designed to provide an FPGA development board with four Gigabit Ethernet ports. Each of the Gigabit Ethernet ports is connected to a Marvell Ethernet transciever which then communicates with an FPGA through the reduced gigabit media-independent interface (RGMII)

protocol Additionally, the board includes a 125 MHz oscillator to supply the FPGA with a clock needed for Gigabit communication.

These features enable a system designer to add networking capabilities to any FMC-compatible development board. EthernetFMC has similar goals to the NetFPGA project (facilitating network system research and development), but takes a slightly different approach. EthernetFMC only provides an expansion card for providing networking interfaces as opposed to a complete network hardware platform. The EthernetFMC project provides example designs for using the card as a traffic generator and a four-port network interface card (NIC) for the Zynq system. In academic projects, the EthernetFMC has been used in a deep packet inspection system [33] and in distributed computation offloading [34].

3.4 JTAG Configuration Manager (JCM)

The BYU JCM tool is used in this system to inject faults in to the FPGA configuration memory and to provide configuration memory scrubbing during radiation experiments. The JCM was developed to facilitate high-speed JTAG communication with FPGAs while providing high-level JTAG programability [35]. The JCM is composed of a MicroZed development board and a custom carrier card that provides JTAG connections to the SoC. The processor on the MicroZed runs a full installation of Linux to allow reliability test applications to be programmed using high-level languages. The FPGA portion of the MicroZed has been programmed with custom hardware that has been designed for communication with JTAG devices.

In this framework, the JCM is responsible for managing the reliability test. This involves performing fault injection, configuration scrubbing, monitoring status registers of the DUT, controlling the traffic generator, and interacting with peripheral devices. An application running on the JCM will direct each of these actions and log the results of the experiment locally.

The flow of a reliability testing application will depend on the requirements of the experiment being performed, but there are some generic phases that are common among the tests performed in this thesis. At first, the JCM will initialize the test by programming the experimental design to the NetFPGA board and starting the traffic generator. Then a configuration upset is introduce into the design through fault injection or neutron irradiation. The JCM then requests a status report from the traffic generator to determine if the design has failed. The JCM will record

the configuration upset along with the functional impact of that upset. Finally, the JCM will recover the DUT to a working state and repeat the process until sufficient data has been collected. Additional details on testing methodology are provided in Chapter 7.

3.5 Summary

The network reliability testing framework provides a controlled environment for observing and measuring the effects of soft errors in FPGA-based network systems. The JCM manages the experiment and provides fault injection for the network system. The traffic generator provides a load for the network system and is used to determine when failures have occurred withing the network system. This framework will be used to perform a soft-error sensitivity test on a FPGA-based Ethernet switch design. The following chapters will provide design details for each component of the network reliability testing framework.

CHAPTER 4. ETHERNET SWITCH ARCHITECTURE

An Ethernet switch is a networking system that operates at the data link layer and is designed to connect network hosts within a local network. To accomplish this task, the switch accepts packets from its clients, inspects the contents of those packets, and then forwards the data to its correct destination. Commercial switches often include additional features which provide packet prioritization, queue management, and device isolation.

This chapter will describe the details of the Ethernet switch design, which is to be used in reliability testing later in this thesis. It will outline the design and operation of each module used to form the design. Additionally, this chapter will show how the switch operates as a whole by tracing the path of a single packet as it is processed by the switch.

4.1 Ethernet Switch Design Overview

The Ethernet switch design is implemented on the NetFPGA-1G-CML as described in Chapter 3. While the development board provides many hardware components that are beneficial for implementing a network system, the actual data processing occurs in the Kintex-7 FPGA.

The Ethernet switch design can be categorized into three groups: interfaces, processing core, and the monitoring system. In Figure 4.1, interface modules are shown in red, the processing core is shown in blue, and the monitoring system is shown in yellow. The interface modules communicate with the Ethernet transceivers to move network traffic to and from the Ethernet switch processing core. The processing core modules determine how to route the data. The monitoring system supervises the entire system to provide error detection.

The Ethernet switch design has four interface modules, one for every physical Ethernet port. The interface modules on the left side of Figure 4.1 are receiving ingress data from the Ethernet transceivers. Once data has been received by an interface module, it will be read by the input arbiter. The arbiter consolidates the four data streams that are produced by the interface modules

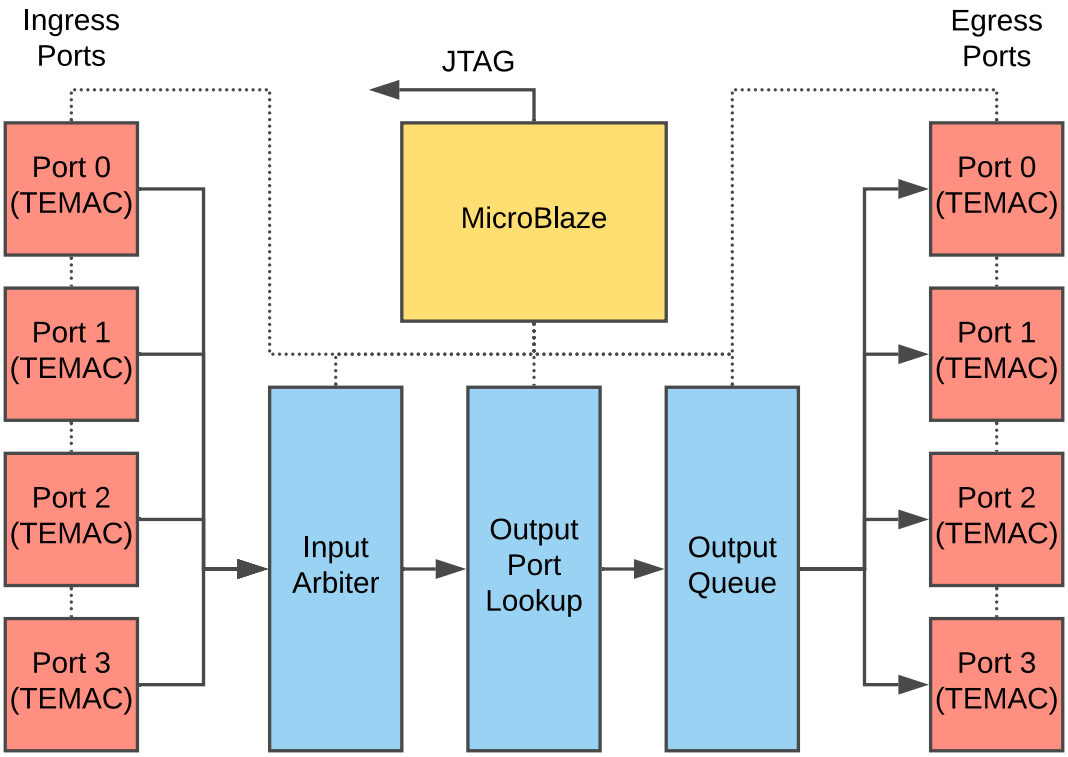

Figure 4.1: Ethernet Switch Block Diagram

into a single stream for processing. The network traffic is then processed by the output port lookup module to determine how packets should be routed. Then the output queue module distributes the single data stream to the interface modules to be transmitted by the Ethernet transceivers. Although the ingress and egress ports are shown as separate modules in Figure 4.1, a single interface module handles both ingress and egress data and each module is shown twice to simplify the diagram.

The monitoring system operates independently of the traffic processing system. This system consists of a MicroBlaze soft microprocessor core and a series of counters distributed throughout the Ethernet switch design. The MicroBlaze periodically reads the values of these counters to determine if anomalies have occurred in the data processing path. By comparing the values of adjacent counters, the MicroBlaze can determine if packets are moving properly throughout the switch. The results of these tests are reported over JTAG to the JCM.

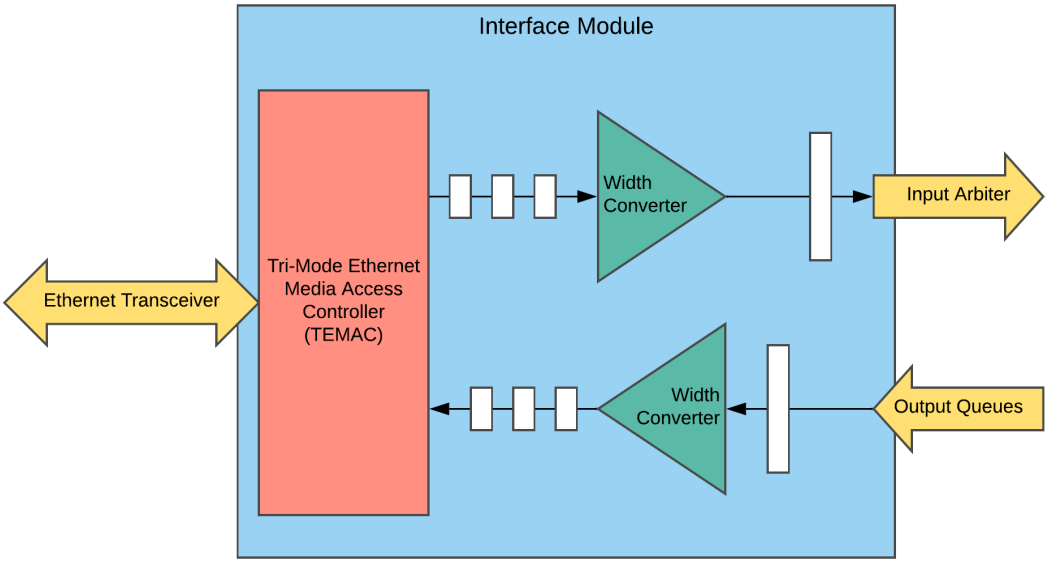

Figure 4.2: Interface Module Block Diagram

4.2 Interface Module

The interface module resides at the periphery of the Ethernet switch design and is responsible for facilitating communication between the Gigabit Ethernet transceivers and the core processing modules. This module is a wrapper for the proprietary Xilinx Tri-Mode Ethernet Media Access Controller (TEMAC) and acts as a protocol converter between the TEMAC and the traffic processing modules. NetFPGA's AXI-Stream width converter provides the necessary conversion to and from the TEMAC. The organization of these submodules is shown in Figure 4.2.

The TEMAC IP core implements the data link layer of the network stack which provides a logical link between directly-connected network hosts. This logical link is established atop the physical analog connection between hosts that is managed by the PHY. This means that when a network application needs to transmit data, the application will assemble an Ethernet frame to provide to the MAC. The MAC will then prepend a synchronization sequence and append a CRC checksum to the Ethernet frame before transmitting the frame to the PHY byte-by-byte. The PHY will handle encoding each byte into electrical symbols to be transmitted across the physical connection. On the receiving end, a MAC will accept the Ethernet frame byte-by-byte from the PHY. This MAC will

be responsible for detecting the start of the Ethernet frame through the synchronization sequence and validating the accuracy of the transmission using the CRC checksum.

The proper implementation of an Ethernet MAC is challenging because Ethernet supports a wide range of speeds and features that must be handled by the MAC. The TEMAC supports the three most common consumer data rates: 10 Mbps, 100 Mbps, and 1 Gbps. Each of these data rates offers different capabilities and calls for different clock rates and slightly modified interfacing protocols (see Table 4.1). The TEMAC must support data rate negotiation and adapting to the interfacing protocol associated with that data rate. Additionally, because the clock rate varies according to transmission rate, the TEMAC must also support operating on a variable clock rate. The TEMAC must also support full-duplex and half-duplex communication depending on the physical connection. While these are the core features that the TEMAC provides, there are many more complex features that allow the TEMAC to be used in a wide range of applications.

Table 4.1: RGMII Transmission Rates

Transmission Rate [Mbps]	Clock Rate [MHz]	Bits per clock cycle
10	2.5	4
100	25	4
1,000	125	8

For the Ethernet switch design, the TEMAC is configured to provide a small handful of standard features: PHY interfacing, data framing, and CRC checksum validation and insertion. The TEMAC interfaces with the Ethernet PHY through the RGMII protocol while interacting with the rest of the Ethernet switch design through an receive (RX) and a transmit (TX) AXI4-Stream interface. These three interfaces are shown in Figure 4.2. When the TEMAC receives data from the PHY, it will process the Ethernet frame and transmit it over the RX data stream. Similarly, the switch design will provide Ethernet frames to the TEMAC through the TX data stream interface. The TEMAC will process the outgoing data and provide it to the PHY to be transmitted. The TEMAC's AXI4-Streams are 8 bits wide and are clocked independently based on negotiated transmission rate and transfer direction. The TX clock is generated locally, using an oscillator, while the RX clock is recovered by the PHY and brought on chip to then be used by the design.

To simplify the Ethernet switch design, the processing core uses a common system clock (100 MHz) while the RX and TX clock domains for each port are contained within their respective interface module. Using a slower system clock provides greater flexibility in FPGA design implementation, but also means that fewer processing cycles will occur in any given period of time. To ensure that the core processing modules can adequately handle the bandwidth of multiple traffic streams, the processing modules use a 256 bit wide AXI4-Stream interface.

Each interface module includes two data-width converter sub-modules to handle stream width conversion in addition to clock domain crossing. Along the RX path, the width-converter module is configured to convert an eight-bit stream to a 256-bit stream while handling the clock domain crossing from the RX clock to the system clock. Additionally, the RX data width converter will tag the packet stream with metadata (packet length and source port number) to be used when processing the packet. Along the TX path, the width-converter module will reduce the 256-bit stream into an eight-bit stream and facilitate the clock domain crossing from the system clock to the TX clock. This width converter will also strip the metadata fields from the packet as this information is no longer needed.

The width-converter module consists of a BRAM FIFO with some supporting logic. On the RX data stream, eight-bit data transfers are received by the width converter module and assembled into a wider 256-bit word. Once this word is filled (or if the last byte of the packet is received), the 256-bit word is then written into the FIFO. Additionally, this module will count each byte that it receives in order to record the size of the packet (see Section 4.7). The packet can then be streamed out of the FIFO at its wider width, synchronous to the system clock. The TX stream operates in a similar manner as the RX data stream, but in reverse. 256-bit word transfers are received from core processing modules and are written into a FIFO. On the reading end of the FIFO, the converter core reads a 256-bit word and breaks it down into 8-bit transfers. These transfers are synchronous to the transmission clock and are received by the TEMAC to be processed for transmission.

In summary, the interface module acts as a wrapper around the TEMAC module. The interface module handles all of the clock domain crossing and protocol conversions necessary for the core processing modules to communicate with the Ethernet transceivers. In this design, the interface module receives transmission data from the output queue module. When the interface module receives an Ethernet frame, it passes this data along to the input arbiter module.

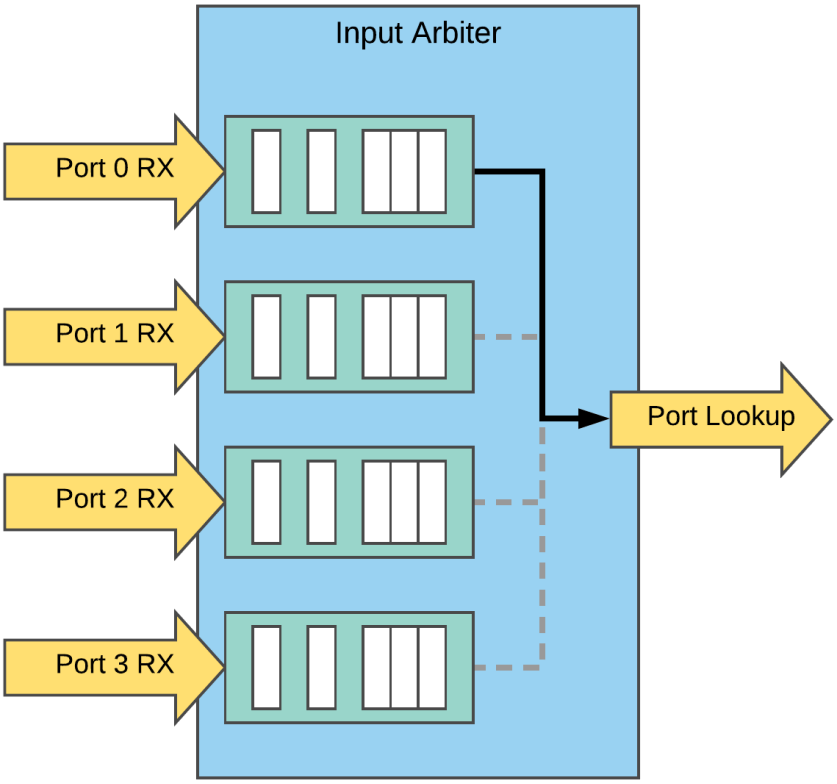

Figure 4.3: Input Arbiter Block Diagram

4.3 Input Arbiter

The input arbiter is responsible for consolidating the RX streams from each interface module into a single data stream for processing. By consolidating these streams, routing and processing information can be centralized and the resource utilization of the FPGA can be minimized. In terms of complexity, a centralized design is preferable to a distributed system as a distributed system needs to consider data coherency and consistency. A centralized design shares resources between all streams of data, allowing configuration to reside in a single location. This makes updating and processing much simpler.

The input arbiter consists of four input interfaces and a single output interface. The module consists of four FIFOs and some routing logic. A FIFO is placed at each of the arbiter's input ports and is deep enough to accept an full packet. These FIFOs allow the operation of the input arbiter to be decoupled from interfacing and handshaking with other modules. As soon as a packet

becomes available within an interface module, it can be transferred to the input arbiter, assuming that the arbiter's input FIFO has sufficient space for the packet. The arbiter iterates through these FIFOs in a round robin fashion, connecting the output of the selected FIFO to the arbiter's AXI4-Stream output. The arbiter permits a single packet to be transmitted per port before switching to another input FIFO. The status signals on the input FIFOs are used determine which port should be considered next. This prevent time from being wasted on input ports that don't have data to be processed.

4.4 Output Port Lookup

The output port lookup module processes the output stream of the input arbiter and is responsible for determining a packet's destination. As a packet is streamed into the module, its contents are parsed to extract source and destination MAC addresses. The destination MAC address is used in conjunction with a content addressable memory (CAM) table and a standard routing table to determine its destination.

A routing table is a block of memory that maps a destination address to a physical port number. While some advanced routing tables may consider traffic priority, network segregation, and other packet metadata in determining a route, this routing table only maps Ethernet MAC addresses to ports. This Ethernet switch design provides a self-learning feature, which means that routes do not have to be programmed statically. Instead, the switch will automatically learn the route to network hosts as they join the network. When packets are received by the output port lookup module, their ingress port and source MAC address will be noted and added to the routing table. Then, when any other network hosts attempts to contact the newly added host, the switch will already know how to route traffic to its destination.

Routing tables are frequently used in conjunction with CAM tables in order to reduce the time needed to find the correct routing table entry. Unlike a traditional memory which uses an address to retrieve data, a CAM table uses data to lookup an address. The output port lookup module uses the Xilinx Parameterizable CAM [36] which is implemented using Block RAMs. Routing table addresses and stored in the body of these BRAM modules while MAC addresses are used to index into these memories. The CAM table is structured and encoded in such a way

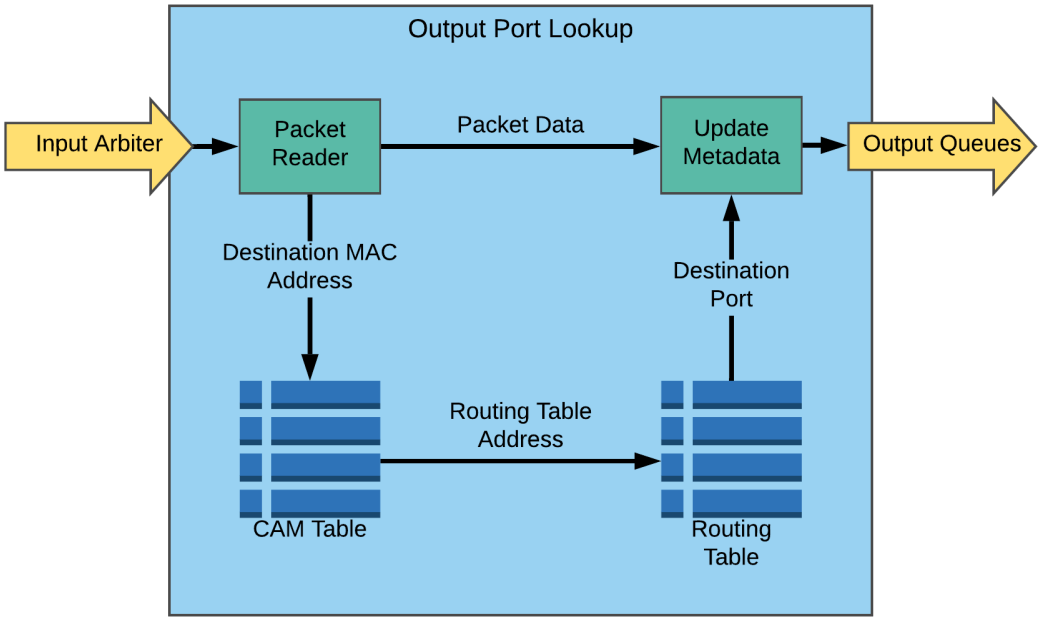

Figure 4.4: Output Port Lookup Block Diagram

that a fully populated 48-bit memory is not needed. Ultimately, this table provides a constant time mapping between MAC addresses and routing table entries.

When the output port lookup module receives a packet from the input arbiter, it will first decode the headers of the packet to determine the source and destination MAC addresses. The source MAC address will be stored in the routing table along with the packets ingress port if a routing entry doesn't already exist. The destination MAC address will simultaneously be sent to the CAM. The CAM will determine if an entry for the provided MAC address exists and will return the corresponding address within the routing table. The output port lookup module will then use this address to index into the routing table and determine the packet's destination port. The egress port number is stored in the packet metadata for future processing by the output queues module.

If the routing table does not have an entry for the destination MAC address, the output port lookup module will default to broadcasting the packet. This allows traffic to get to its destination even though the switch is unaware of the correct route. This broadcast behavior is achieved by marking all egress ports in the packet metadata. However, the packet will not be transmitted back to the ingress port as this would create additional unneeded traffic.

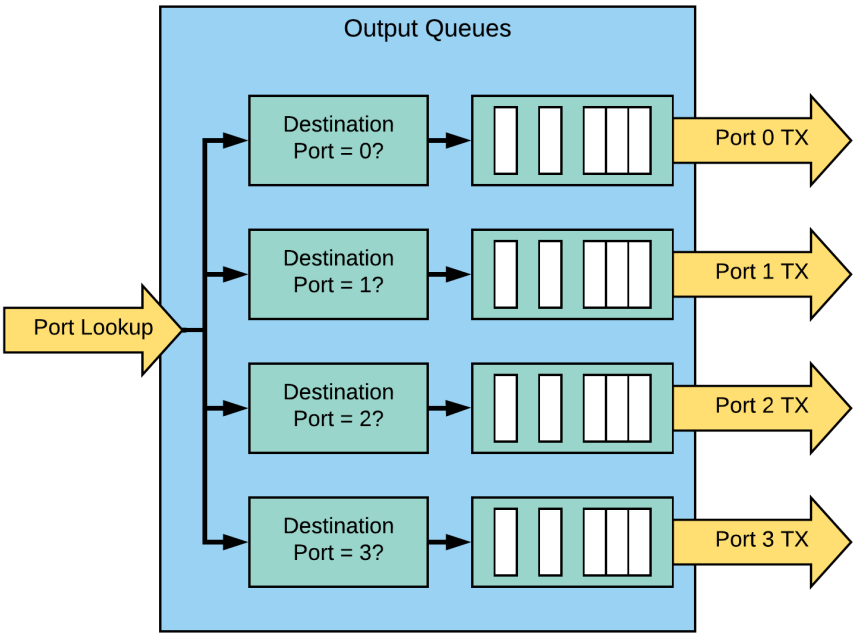

Figure 4.5: Output Queues Block Diagram

4.5 Output Queues

The purpose of the output queues module is to distribute packets from the central processing stream to individual streams for transmission. It accepts a single input packet stream from the output port lookup module and produces four output streams, one for each of the interface module. This module is also responsible for detecting congestion within any egress path and dropping packets where necessary.

This module consists of four output packet queues and some routing logic. When a packet enters the module, the routing logic will first inspect its metadata to determine which destination ports are needed. Assuming that all of the requested queues have sufficient space for the packet, the routing logic will assert the write enable signal for the appropriate output queues and allow the packet to be streamed into those queues. However, if any of the requested queues are full, the write enable signal will not be asserted for any queue. This will result in the packet being dropped.

The output of each queue is connected to the transmission port of an interface module. These output queues are controlled independently, which allows decoupling between packet pro-

cessing and data transmission. This allows each output interface to transmit at its desired rate without the need to coordinate precise timing requirements.

4.6 Monitoring System

In addition the the packet processing system, a monitoring system is also present within the Ethernet switch. This system is used to observe the health of the switch and detect any abnormal operation. The monitoring system consists of packet counters, a soft-core MicroBlaze processor, and a JTAG communication module. The location of these components is highlighted in Figure 4.6.

In each of the packet processing modules, counters are placed at both the input and the output ports to record the number of packets that pass into and out of the module. The output queues module also includes a counter to record the number of packets that were dropped due to excessive congestion. Each of these packet counters are accessible over an AXI4-Lite connection.

A soft-core processor is connected to each of these packet counters and periodically reads their values. The values of adjacent packet counters are compared and any discrepancies over a certain threshold value are reported as a failure. Discrepancies between the output port of one module and the input of the subsequent module indicate a failure in transmission between these modules, while a discrepancy between the input and output of the same module would indicate a failure within that module. As all input and output paths are being recorded, a counter discrepancy indicates that a packet has truly been lost and that some system failure has occurred.

Monitoring occurs in parallel with processing, so it is expected for some discrepancy to exist between modules as multiple packets will inevitably be in different phases of processing. Therefore it is important to set a threshold high enough to account for these expected differences, but low enough to catch actual errors. We found that using an error threshold of eight packets provided the design with enough leniency to avoid false alarms while still detecting actual failure events.

The results of the monitoring system are then reported to an external system using a custom JTAG communication module. The FPGA provides JTAG BSCAN user primitive elements that allow designs to be accessed through the JTAG chain. This port is frequently used for debugging and testing, but can also be used for serial communication. To use this core, the soft-processor would write its results to a a register file within the communication core. These registers would

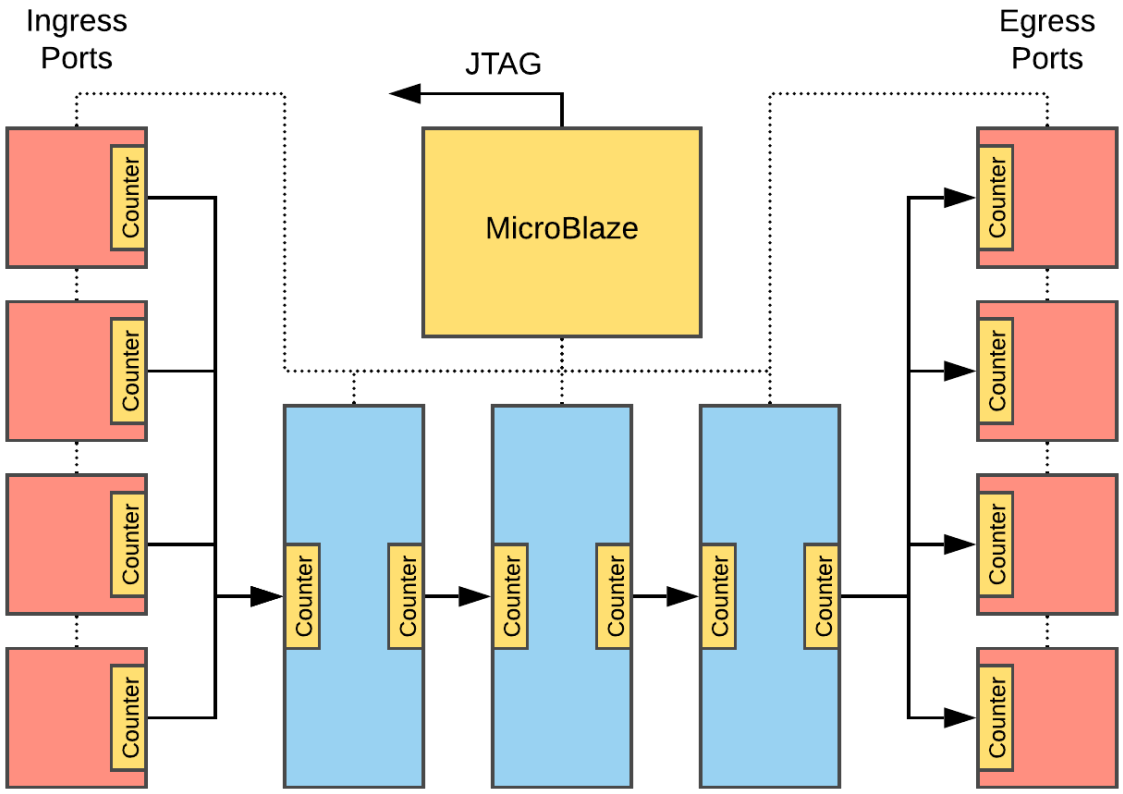

Figure 4.6: Monitoring System Block Diagram

then be serialized and transmitted when requested by the JTAG host. This same communication core is then also used to transmit commands from the JTAG host to the soft-core processor. While many other forms of communication could have been used to transmit device statistics, this form was used for its ease of incorporation with the rest of the test framework.

4.7 NetFPGA Standard Interfaces

The AXI4-Stream protocol is used extensively to interconnect these processing modules and to provide a datapath for network traffic. This protocol uses a point-to-point topology to connect a single master (data producer) to a single slave (data consumer). While this topology makes the protocol simple to implement, AXI4-Stream is sufficiently versatile to support a wide

variety of applications. As used in the NetFPGA project, a stream consists of 6 data and control signals: TREADY, TVALID, TDATA, TKEEP, TLAST, and TUSER.

TREADY and TVALID are each single bit signals used for handshaking between the master and slave modules. TREADY is controlled by the slave module and is used to indicate that the slave is prepared to receive data from the master. TVALID is asserted by the master to indicate that the values currently being presented on the stream are valid. A transfer occurs when both of these signals are asserted during the same clock cycle. Data transfers may occur back-to-back as long as both TREADY and TVALID signals remain asserted.

TDATA is the primary payload of the stream and is used to pass packet data between modules. The width of TDATA is parameterizeable depending on the needs of the application and available resources, but the width must be an integer number of bytes. This project uses an 8-bit TDATA stream within the interface module to support 1 Gbps transport and a 256-bit TDATA stream within the core, supporting data transport up to 40 Gbps. The AXI4-Stream specification requires that the first byte of a transfer is located in TDATA[7:0], the second byte is in TDATA[15:8], and so on.

As an Ethernet frame can be no smaller than 64 bytes, multiple transfers are required to successfully transport a packet between modules. The TLAST signal is asserted to mark the final transfer of a packet and consequently indicates that the next transfer will be the beginning of a new packet. Frequently the last transfer of a packet does not fill the entire TDATA signal. When this occurs, the data is consolidated into the lower order bytes of TDATA and the high order bytes are unused. TKEEP is used to indicate which bytes of TDATA are valid, and which should be ignored. TKEEP[n] is asserted to indicate that nth byte of TDATA is valid (TDATA[$8n-1:8(n-1)$]).

TUSER is a secondary payload within the stream and is used to convey metadata about the packet. This may contain information that is not directly available in the packet body but that is useful for packet processing. The NetFPGA standards requires that TUSER be valid during the first transfer of a packet and the value of TUSER is undefined during subsequent transfers. TUSER is 128 bits wide and consists of 9 fields, as shown in Table 4.2. The source and destination port fields are one-hot encoded, allowing a packet to be broadcasted on several interfaces by setting multiple bits in the destination field. The user-defined metadata slots are not used in this application, but all NetFPGA compatible cores must preserve and forward these fields. In this application, the

Table 4.2: NetFPGA AXI4-Stream TUSER Signal Definitions

Signal	Description
TUSER[15:0]	Packet length (bytes)
TUSER[23:16]	Source port (one-hot encoded)
TUSER[31:24]	Destination port (one-hot encoded)
TUSER[47:32]	User-definable metadata slot 0
TUSER[63:48]	User-definable metadata slot 1
TUSER[79:64]	User-definable metadata slot 2
TUSER[95:80]	User-definable metadata slot 3
TUSER[111:96]	User-definable metadata slot 4
TUSER[127:112]	User-definable metadata slot 5

packet length and source port fields are filled by the ingress interface while the destination port is determined by the output port lookup module.

4.8 Packet Processing Example

To illustrate how each of these components works together, we will follow the path that a single packet takes as it is transmitted between hosts and through this Ethernet switch. This path is shown in Figure 4.7. A network application on the transmitting host generates the data and sends the data to the host operating system to transmit on the network. The host will encapsulate this data within a transport layer segment, a network layer packet, and finally an link layer frame before it can transmit the data on the network. Each of these encapsulations provides crucial information to ensure that the data is able to arrive safely at its intended destination. Some of the most notable fields that are included in a packet are addressing information, port numbers, priority, and data checksums. Once prepared, the host will provide the Ethernet frame to its network interface card to transmit across its copper Ethernet connection (1000BASE-T).

The signal will propagate across the copper wires and arrive at the Ethernet switch's PHY. This chip will handle the intricacies of receiving a signal transmission, intricacies that include echo cancellation, clock recovery, DC-balancing, and signal decoding. The transceiver will use the transmission to reconstruct the Ethernet frame and will send the frame, along with a recovered clock, to the FPGA through an RGMII connection.

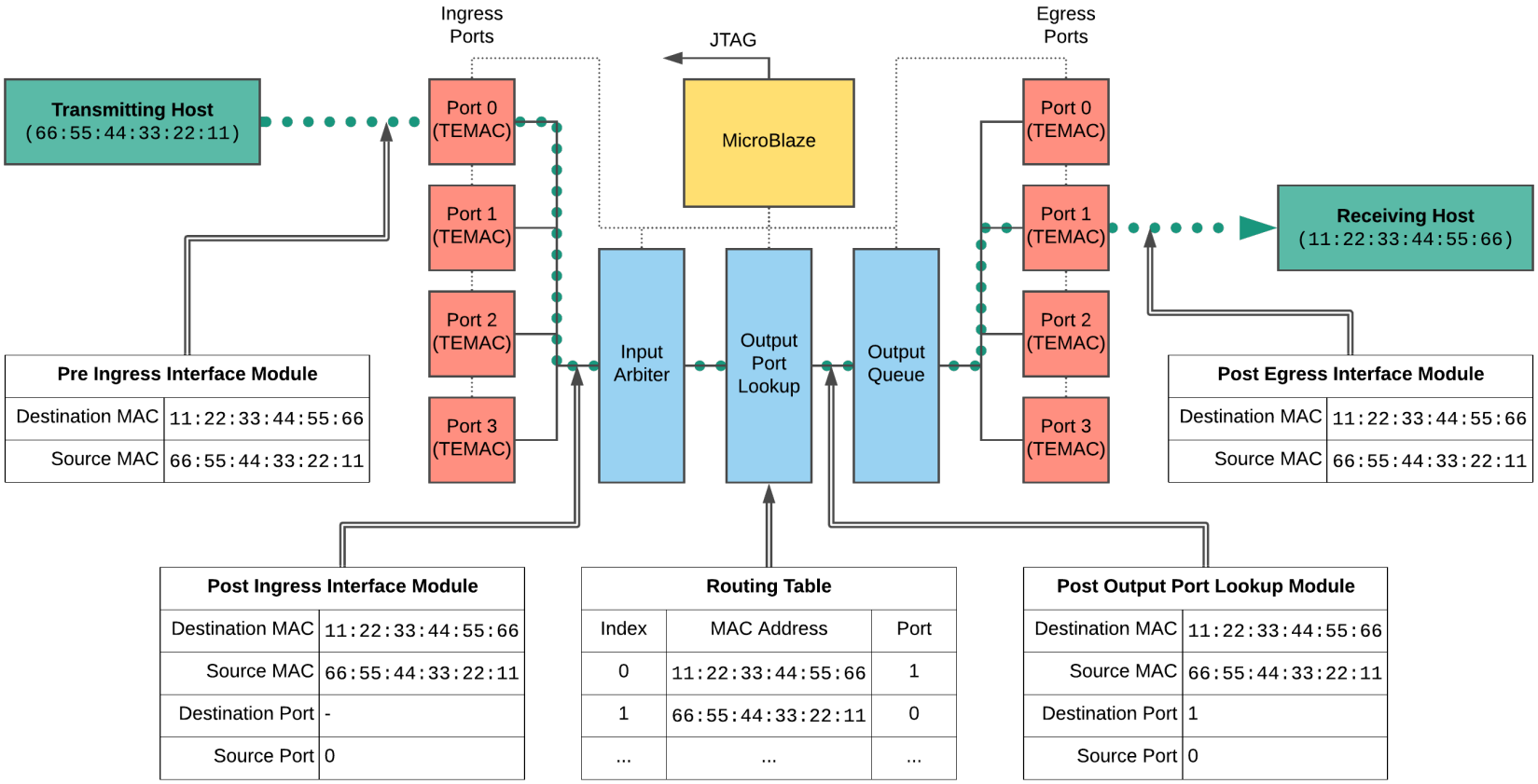

Figure 4.7: An example of a single packet being processed by the Ethernet switch. Important headers and metadata fields are shown at key stages in the switch.

This RGMII connection is received by the interface module within the FPGA logic. This module converts the RGMII data stream into an AXI4-Stream that is better suited for data processing within an FPGA. Additionally, the interface module handles the issues of clock domain crossing, packet tagging, CRC removal, and data stream widening. Once the packet has passed through all of these operations, it waits in a queue to be accepted by the input arbiter.

The input arbiter cycles between all of the interface modules in a round-robin manner to ensure that each interface has a fair chance to have its data processed. Upon detecting the arrival of a packet at one of the interfaces, the arbiter finish processing any scheduled queues and will eventually read the packet out of the interface queue and will pass it along the main processing stream.

The packet is then brought into the output port lookup module to determine its destination. Initially, the packet's source and destination MAC addresses are extracted from the packet. Then, the destination MAC address will be compared against a CAM routing table to determine the destination, and the source address will be recorded for future routing decisions. The destination port will be recorded in the metadata of the packet, and the packet will be passed along to the output queue module.

When the packet is received by the output queue module, the destination field will be read to determine which queues will be required. If all of the available queues are available, the packet will be copied into the output queues. However, if any of the required queues are full, the packet will be dropped.

Each output queue feeds the transmission stream of the interface module. When the interface is available, it will accept the packet from the queue, downsize the stream, convert the clock domain, calculate a new CRC value, and transmit the frame to the PHY over RGMII. The PHY will then perform all of the necessary analog signaling needed to transmit the packet over the wire to the receiving host.

4.9 Implementation

This Ethernet switch design was synthesized and implemented using Vivado 2018.1, targeting a Kintex-7 325T FPGA (xc7k325tffg676-1). The design required 27% of the FPGA's available block memories and 17% of the available slices. Block memories were used extensively through-

out the design to provide packet buffers. These buffers are used in separating the various stages of packet processing and for providing safe clock domain crossings within the interface modules. Slice utilization provides a measure for the amount of logic that is needed to implement the required functionality. Detailed resource utilization is shown in Table 4.3.

Table 4.3: Baseline Ethernet Switch Resource Utilization

Kintex 7 Device	Used		Total
Slices	8,456	(17%)	50,950
Registers	29,615	(7%)	407,600
LUTs	18,345	(9%)	203,800
Block Memories	120	(27%)	445
Global Clock Buffers	8	(25%)	32
I/O	53	(13%)	400

The design consists of eight different clock domains: a system clock domain, two TX clock domains, four RX clock domains, and a reference clock domain. The system clock domain operates at 100 MHz and contains the core processing modules, the monitoring system, and portions of the interface modules. Each interface module has its own RX clock that is received from an Ethernet transceiver. Physical resource constraints within the FPGA allow two interface modules to share a single TX clock which operates at 125 MHz to support Gigabit Ethernet. The reference clock is used in the recovery of each RX clock signal. The post-implementation timing report shows that the system clock domain has a minimum slack of 1.641 ns which translates to a maximum operating frequency of 120 MHz. The worst slack within any of the TX or RX clock domains is 1.801 ns, resulting in a maximum operating frequency of 161 MHz. These values indicate that the design is able to operate at its designated frequencies.

Vivado offers the ability to generate a report of essential bits to be used in estimating and improving the soft-error reliability of a design. An essential bit is any configuration bit that would effect the design circuitry if upset. Non-essential bits are guaranteed to have no impact on the functionality of a design, while essential bits may potentially impact functionality [37]. Even in worst-case scenarios, no more than 10% of configuration bits in a design are critical to its functionality [38]. This baseline Ethernet switch design has 5,449,425 essential bits out of 75,202,176 total configuration bits (7.25%). Using the essential bits data and the neutron upset

data presented in [38], this design has an estimated essential bit soft-error rate of 272 FIT[1]. This means that a radiation-induced failure is expected to occur no more than once every 419 years, per device. This estimation provides an upper-bound on the soft-error sensitivity of a design, but determining how many of these configuration bits are actually critical to the functionality of the design must determined experimentally. The details and results of this test will be discussed in Chapter 7.

[1]Failures in Time (FIT) is the expected number of failures per one billion device-hours of operation.

CHAPTER 5. SOFT-ERROR MITIGATION OF THE ETHERNET SWITCH DESIGN

There are a handful of commonly employed design techniques that are used to reduce the soft-error rate of FPGA applications. In contrast to device-hardening techniques which are applied at a physical level to reduce the occurrence of soft errors [39], these techniques are used by the digital logic designer to allow and application to better tolerate a soft error [40]. This chapter will introduce three common techniques used to provide soft-error mitigation: TMR, DWC, and configuration scrubbing. TMR is used to provide online fault tolerance, allowing a system to continue to operate in the presence of errors. DWC provides increased detection of faults which then allows the system to perform some recovery procedure. Configuration scrubbing corrects upsets within the design and can be used in conjunction with either of the previous two techniques.

The Ethernet switch, as described in Chapter 4, serves as the baseline design for applying these soft-error reliability techniques. Other than a series of packet counters, the baseline design offers no additional reliability or error detection features. Two experimental designs were derived from this baseline design: TMR and DWC. Both of these designs were created by performing netlist-level modifications to the fully synthesized baseline switch design. This chapter will also describe the steps taken to apply mitigation techniques to form these experimental designs.

5.1 Triple Modular Redundancy (TMR)

A common approach to mitigating soft errors is through TMR which uses spatial circuit replication to increase reliability by masking errors. In single device spatial TMR, three redundant copies of the circuit design are placed on the same FPGA. These circuits operate in parallel, allowing the mitigated circuit to operate under the same time constraints as the original design (barring any timing violations resulting from the implementation of a larger design). Each copy of the circuit receives the same input value and, under normal operation, the output from these modules is identical.

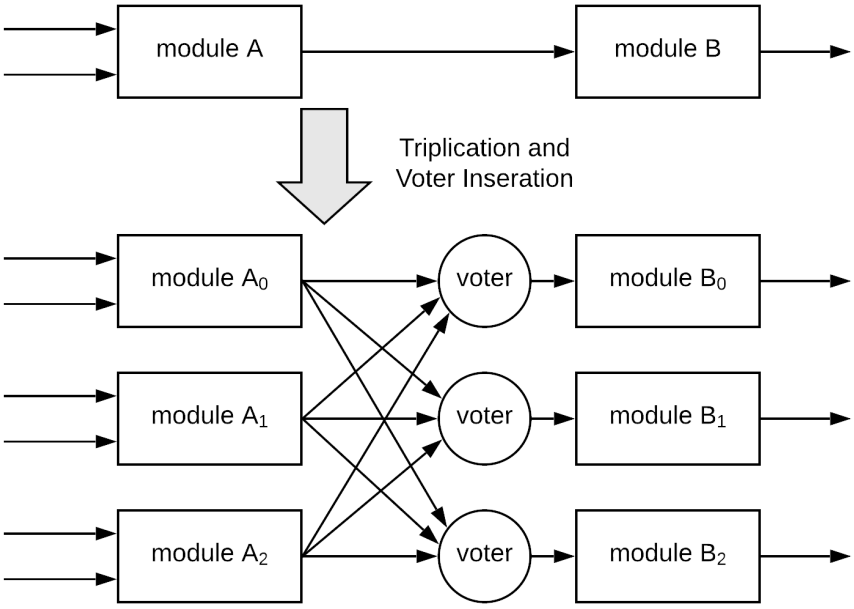

Figure 5.1: Triple Module Redundancy (TMR)

However, should an upset effect the operation of a single circuit, that circuit will produce an invalid result. To handle this, majority voters are inserted at the output of the triplicated modules to propagate the majority value of a signal between the redundant circuits. So long as at least two of the circuits are functioning correctly, the propagated value will be correct as well. This effectively masks the result of a faulty circuit and allows the design as a whole to continue to operate despite the presence of internal errors. To avoid single point failures, voters are often triplicated as well (see Figure 5.1) [41].

The TMR Ethernet switch design was created using BYU-LANL TMR tools [42] to automate the process of triplication and voter insertion. This tool accepts a synthesized Edif netlist file as an input, applies triplication to the design, and produces a new netlist file that can be used to complete design implementation. The process of triplicating a design involves three primary steps. First, each component instantiation of the design is triplicated. Then, the nets connecting these components are also triplicated [41]. The final step is voter insertion.

When a voter is added to a circuit, a net is cut and the output of the source module is used to drive the voter, and the output of the newly inserted voter is then used as the input to the nodes previously connected to the cut net. This step is particularly challenging as adding voters increases

latency and resource utilization, so it is desirable to obtain the most benefit possible using the fewest number of voters. Additionally, it is important to insert voters within feedback paths to provide self-synchronization to the circuit. The Highest Flip-Flop Fan-out algorithm, as described in [41], provides small timing and area impacts and was selected as the preferred voter insertion algorithm.

The BYU-LANL TMR tool has historically prohibited the use of hierarchy in netlists and would automatically flatten any violating design. A flat design reduces the complexity of design analysis and voter insertion. Unfortunately, this design could not be flattened effectively due to constraints and timing requirements. To overcome these challenges, the BYU-LANL TMR tool was modified to accept hierarchy. Each primitive element would be replicated within its own layer of hierarchy and then ports would be added as needed to each module to support connections between triplicated logic elements.

Additional challenges presented by this design include clock domain crossing and proprietary encrypted IP. While TMR has proven to be successful in synchronous circuits, it cannot be safely used across clock boundaries without modification [43]. To handle clock domain crossings, fan-in voters were inserted into the circuit prior to clock-domain crossing synchronizers. While this does provide single points of failure within the design, it does also ensure that there are no clock-domain crossing errors. Similarly, the Xilinx TEMAC core is a proprietary encrypted IP that must be treated as a black box. This core has many input signals across several clock domains, with a couple signals requiring precise timing constraints. To avoid interfering with its functionality, fan-in voters were inserted prior to any input port and output signals were split to feed triplicated logic.

To produce the TMR design, the synthesized EDIF netlist is extracted from the baseline Ethernet switch design. Then, the netlist is provided to the BYU-LANL TMR tool to apply TMR mitigation. Recognizing that mitigation cannot be safely applied to certain regions of the design, the tool is configured not to replicate these areas. Specifically, these regions include clock domain crossing areas, the encrypted portion of the TEMAC IP core, BSCAN primitive instantiations, and I/O buffers. Then, the tool will automatically apply the mitigation strategy as described previously and generate a new EDIF netlist. TMR was applied to 70% of circuit components in the Ethernet switch FPGA design.

Vivado 2018.1 was used to perform implementation of the TMR design using the new EDIF netlist and constraints exported from the baseline design. One of the anticipated challenges of TMR application is increased resource utilization. For a fully triplicated design, a 3× - 6× increase in area can be expected [44]. As the Ethernet switch design could only be partially triplicated, the increase in resource utilization was less than would be seen in a fully triplicated design. Resource utilization for the TMR Ethernet switch design are shown in Table 5.1.

Table 5.1: TMR Ethernet Switch Resource Utilization

Kintex 7 Device	Used	Total	Increase from Baseline
Slices	22,807 (45%)	50,950	2.70×
Registers	69,699 (17%)	407,600	2.35×
LUTs	57,160 (28%)	203,800	3.12×
Block Memories	344 (77%)	445	2.87×
Global Clock Buffers	8 (25%)	32	1.00×
I/O	53 (13%)	400	1.00×

This increase in resource utilization can also present challenges in meeting timing. The fine-grained application of TMR to an FPGA design requires that the triplicated modules are in relatively close proximity. Attempting to add more logic within a limited area will require all logic to spread out a little. This logic spreading increases propagation delay within the circuit. Additionally, voter insertion can impact the critical path of a circuit introducing more challenges to meeting timing specifications. The calculated maximum operating frequencies for the TMR design are 107 MHz and 140 MHz for the system clock domain and the interface clock domain, respectively. Both of these clock rates are notably slower than the baseline design. Although these clock rates meet the requirements of the design, it is easy to see how the application of TMR can present challenges for the FPGA design tools. In the most severe cases, it is not possible to meet the timing constraints of a TMR design. To resolve this issue, the scope of TMR application may need to be reduced.

Using Vivado's essential bit algorithm, it was determined that this design has 16,266,756 essential bits out of 75,202,176 total (21.63%). This is a 2.98× increase in essential bits and results in an essential bit soft-error rate of 811 FIT. This increase in FIT rate corresponds with the increase in resource utilization, but this does not mean that the design is more likely to experience

a functional failure. Determination of critical bits must be determined experimentally, as will be shown in Chapter 7.

5.2 Duplication With Compare (DWC)

DWC is designed to provide on-line error *detection*, as opposed to TMR which provides on-line error *correction*. Many systems can operate operate correctly despite the presence of an upset if there is adequate error detection [45]. Depending on the system, error detection can be used to trigger a repair routine, invalidate potentially corrupt data, or activate a redundant failover system. Network protocols are designed to compensate for occasional data loss to minimize the end-user impact of brief network failures. Highly-reliable network systems frequently use redundant hardware components to handle a wide variety of failure modes [46]. DWC could be utilized within a network system to activate a failover once an upset has been detected. While the backup system is handling the network traffic, the soft-error can be corrected in the primary system. The primary system will then be returned to a working state and can resume its role. While there are other ways that DWC could be utilized in a network system, this example illustrates how DWC enables network systems to achieve high availability by exploiting existing redundant hardware.

DWC aims to increase a circuit's ability to detect errors using spatial replication. DWC calls for two redundant copies of a circuit to be placed on the same FPGA. These two circuits then operate in lockstep and should produce identical results under normal operation. Detectors are placed at the outputs of the circuit to determine if the output from one circuit differs from the other, indicating that a fault has occurred within either circuit. Detectors are often duplicated to rule out false positives that occur when a fault is present within the detector rather than the actual circuit [40].

The BYU-LANL TMR tools were also used to generate the DWC design [40]. There are three main steps to generating a DWC design: replicate logic, replicate nets, and add detectors. Replication of logic and nets occurs in the same manner as the TMR process, but only two copies are inserted instead of three. And while detectors are similar to TMR's voters, they provide the benefit that they do not need to be placed within the data path. The detectors are placed strategically throughout the design to compare the output values of duplicated modules. However, the output

signals of detectors are not inherently used in the design itself, which minimizes the timing impact of detector insertion.

Like TMR, it is unsafe to cross clock-boundaries in DWC without taking special precautions. This results in several isolated regions of duplicated logic. The boundaries of these regions are marked by clock-domain boundaries, black box module I/O ports, and the FPGA's I/O pins. It is at these boundaries that duplicated detectors are placed. Duplicated detectors are used in order to reduce the occurrence of false positives. When both detectors report a discrepancy, then there is high probability that an error has actually occurred in the protected logic. However, if only one detector reports an error, then the fault likely exists within the detector and not within the system.

The output signals of the detectors are brought together into a central communication module. Similar to the communication system described in Section 4.6, this module serializes the detector outputs to be read over a JTAG connection. This allows an external system to be alerted to system failures.

An example of DWC application is shown in Figure 5.2. This figure shows how the original modules are duplicated in the mitigated design. A boundary of replication exists between module B and module C which could be attributed to clock domain crossing or a black-box module. At this boundary, two detectors are inserted into the design and are provided with the output signals from modules B_0 and B_1. The output signals from these detectors will be routed to some monitoring system not shown here. The output signal from module B_0 will be used to drive the inputs to both module C_0 and module C_1. The modules in replication domain 0 will be used to drive the actual outputs of the system, while the modules in replication domain 1 are used exclusively for comparison against the logic in domain 0.

Table 5.2: DWC Ethernet Switch Resource Utilization

Kintex 7 Device	Used		Total	Increase from Baseline
Slices	15,768	(31%)	50,950	1.86×
Registers	50,577	(12%)	407,600	1.71×
LUTs	33,755	(17%)	203,800	1.84×
Block Memories	232	(52%)	445	1.93×
Global Clock Buffers	8	(25%)	32	1.00×
I/O	53	(13%)	400	1.00×

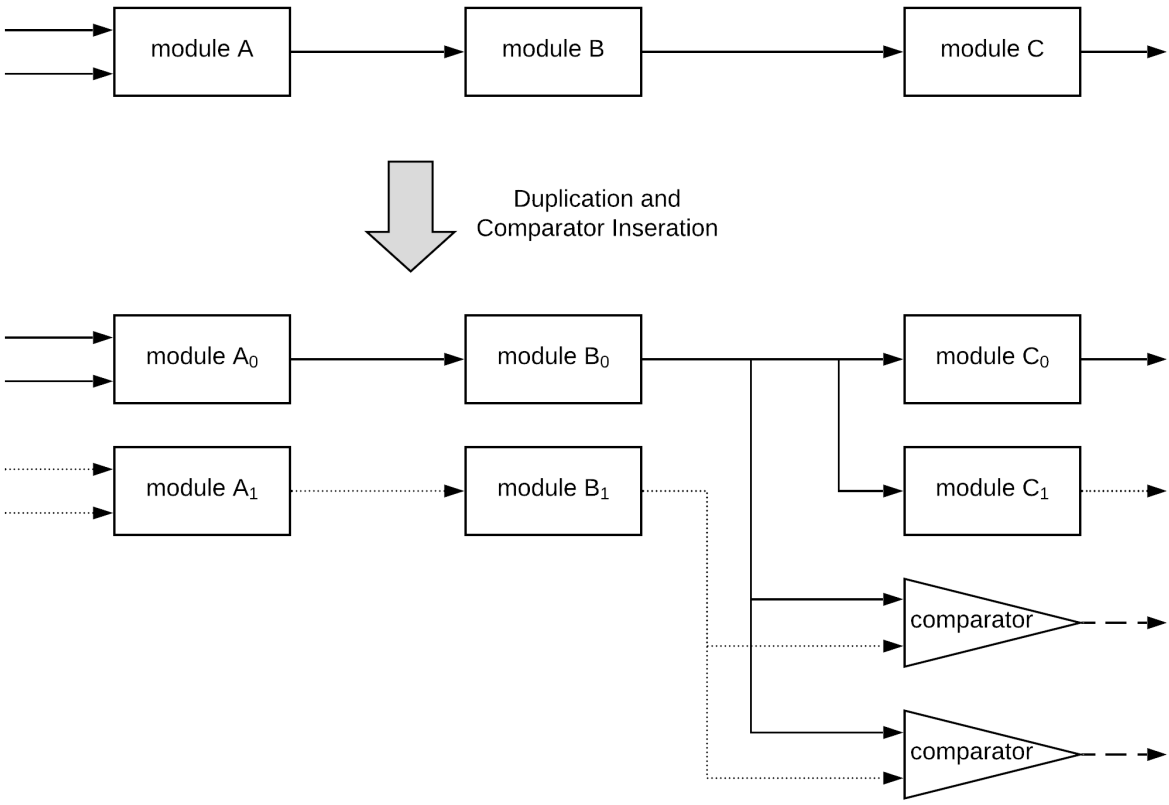

Figure 5.2: Duplication with Compare (DWC)

Like TMR, DWC also experiences increased resource utilization, but to a lesser degree. Table 5.2 shows that the DWC design uses no more than 2× any resource in the baseline design. DWC was applied to only 56% of the design, and the utilization would likely be slightly larger than 2× if the complete design could be duplicated. The DWC design has maximum operating frequencies of 114 MHz and 161 MHz for the system and interface clock domains, respectively. This is slightly slower than the baseline design and is a likely consequence of the logic spreading effect discussed previously. However, the impact of DWC on timing is much less than that of TMR.

This design has 9,375,669 essential bits out of 75,202,176 total (12.47%). When compared to the baseline design, the DWC design has 1.72× more essential bits. This results in an essential bit soft-error rate of 468 FIT. However, this metric does not account for the protection that DWC provides and cannot directly translate to an estimated system failure rate. Because a DWC design

contains two identical copies of the original circuit, it is expected that twice as many upsets will occur within the system. However, these upsets will be split evenly between the functional circuit and the redundant circuit. This will result in the functional circuit experiencing approximately the same failure rate as the original, unmitigated design. However, as the redundant copy will also experience failures, the system will report false positive events at a similar rate as actual functional failures. This false positive phenomenon is expected, but not necessarily problematic, and will be explored further in Chapter 7.

5.3 Configuration Scrubbing

Configuration scrubbing is a technique used to correct upsets that occur in an FPGA's configuration memory. Scrubbing is performed by periodically by reading the configuration memory of the FPGA and comparing against a known golden value. Any discrepancy between these two values indicates that an upset has occurred and that repair is needed. Upon detection of the upset, the scrubber overwrites the corrupted memory value with the golden value. The process of scrubbing the configuration memory can occur while the FPGA's application is running and does not introduce any interruption.

A scrubber can be internal or external to the FPGA [47]. An internal scrubber relies on memory protection mechanisms to detect when upsets occur. Configuration memory is frequently protected as a whole by CRC with individual frames protected by ECC [48]. These two schemes are complementary and provide robust and high resolution error detection. The scrubber periodically checks the validity of both mechanisms and in most cases can determine how to correct a potential upset.

External scrubbers rely on additional hardware to inspect the state of the FPGA. These scrubbers have their own copy of the configuration to provide comparison against the deployed FPGA. This separate copy of the configuration allows the scrubber to provide more robust error detection than an internal scrubber is usually able to provide [49]. However, this robustness comes at the expense of increased system complexity, additional reliability concerns, and potentially reduced response speed.

For the experiments performed in this thesis, the JCM will be used as an external scrubber. This scrubber will be used in both fault injection and neutron irradiation tests for all three variants

of the Ethernet switch design. Using an external scrubber will provide thoroughness and accuracy that will be beneficial to understanding the nature of SEUs in network systems. Although external scrubbers tend to be slower than internal scrubbers, repair speed is less of a priority for these experiments.

5.4 Summary

The mitigation strategies discussed in this chapter are frequently used by system designers to decrease SEU sensitivity of an FPGA design. TMR provides on-line error correction to allow a system to continue operating in the presence of an upset. DWC provides on-line error detection to alert the system of the presence of an upset to trigger some other repair or mitigation strategies. Both of these techniques were applied to the baseline Ethernet switch design using the BYU-LANL TMR tools. The soft-error sensitivity of these designs will be tested, and results shown in Chapter 7.

CHAPTER 6. TRAFFIC GENERATOR ARCHITECTURE

A traffic generator is a networking tool that is used to test the performance and reliability of networking system by generating random packets for that device to process. To effectively test the network device, the traffic generator must also monitor the output traffic to determine if it has been processed correctly. Traffic generators give system designers valuable information concerning the operation of their system under stress. Because these devices are so widely used in research and development, several commercial solutions exist, such as Ixia's IxNetwork [50] and Xena Network's Valkyrie product line [51]. Many academic traffic generators have also been developed to provide the flexibility and cost savings required by researchers [28], [52]–[54].

This work introduces a custom traffic generator that is designed for measuring the accuracy and reliability of a network system. This traffic generator is capable of managing eight independent traffic flows across four Gigabit Ethernet ports. The generator is a hybrid computing system consisting of custom FPGA hardware design with a managing application running on a CPU. The hardware design entirely handle the dataflow of the traffic generator while the processor application provides configuration for the hardware. This hybrid design allows the traffic generator to achieve line-rate speeds through the FPGA while facilitating simple system integration through software running on the processor. This chapter will describe the design and functionality of the traffic generator that was developed for testing the reliability of the Ethernet switch designs described in Chapter 4 and Chapter 5.

6.1 Traffic Generator Overview

The traffic generator consists of three primary systems: the management system, the packet generator, and the packet checker. The management system runs on the ARM processor and is responsible for configuring the traffic generator and interacting with the reliability testing framework. It is responsible for activating and deactivating the packet generator as requested and for collecting

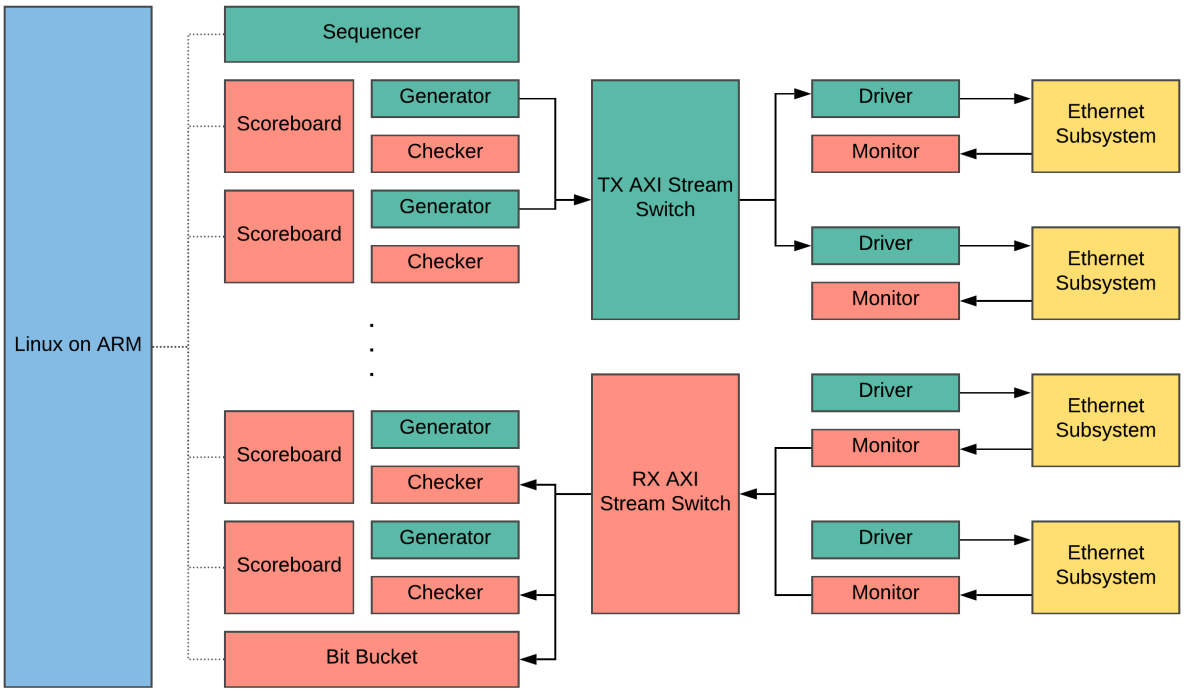

Figure 6.1: Traffic Generator Block Diagram

results from the packet checker. The packet generator is responsible for crafting and transmitting randomized Ethernet frames according to the management system's specification. The packet checker compares received packets to transmitted packets in order to detect a potential error. The traffic generator is diagrammed in Figure 6.1 with the monitoring system shown in blue, the packet generator system in green, and the packet checking system in red.

The configuration of the traffic generator is organized into traffic "flows." A flow refers to single unidirectional stream of related packets that originates from the traffic generator. Flows are configurable to support several upper-layer protocols and can be used to specify packet length, transmission frequency, and header values. The fields that can be specified for a flow are shown in Table 6.1. The flexibility provided by the network flow allow the system to imitate the network characteristics of actual network applications. This traffic generator supports eight simultaneous network flows, each of which can target any physical Ethernet port.

These flows are used to control both packet generation and packet checking. The traffic generator was designed to work in a loopback configuration in which every packet that is generated

Table 6.1: Mapping of packet headers between flow configuration, mini packets, and full packets

Layer	Flow Configuration	Mini Packet	Full Packet
Configuration	Duty Cycle	-	-
	Source Interface		-
	Destination Interface		-
Ethernet	Source MAC Address		
	Destination MAC Address		
	Ethertype		
Internet Protocol (IPv4)	-	-	Version
	-	-	Header Length (IHL)
	-	-	Service Type (DSCP)
	-	-	Congestion (ECN)
	-	-	Total Length
	-	-	Identification
	-	-	Flags
	-	-	Fragment Offset
	-	-	Time To Live
	Protocol		
	-	-	Header Checksum
	Source IP Address		
	Destination IP Address		
Transport Layer (TCP/UDP)	Source Port		
	Desination Port		
Payload	Max Length	Length	Body
	Min Length		
	-	Seed	
	Flow ID		
	-	Packet ID	
Ethernet	-	-	Checksum (CRC32)

by the traffic generator is expected to also return to the traffic generator for checking. Traffic flows define what type of packets are to be generated by the traffic generator, which port they should originate from, and what port they should return to after being processed. To use the traffic generator effectively, a test designer must first understand how their network system is configured to operate and then design flows appropriate for the network system. Flow configuration is stored in software within the management system and used to program the traffic generator hardware upon startup.

The packet generator system is responsible for creating and transmitting packets into the network system under test. Packet transmission is directed by the sequencer module which signals to the generator modules when it is time to create a packet. There is one generator module for each traffic flow, and they are responsible for creating randomized "mini packets" based on their respective flow configurations (see the Flow Configuration and Mini Packet columns of Table 6.1). A mini packet is a condensed version of a regular Ethernet packet that is exclusively used within the traffic generator hardware. The mini packet contains all of the important information that is contained within the full-sized packet, but is smaller and easier to manage. For each mini packet that is created by the generator, a copy is sent to both the checker module (to be used in packet checking) and to the driver (by way of the TX AXI4-Stream Switch). There is one driver module for every physical Ethernet port of the traffic generator. This module serves to convert the mini packet into a full-sized packet and to transmit the packet through the TEMAC (see the Mini Packet and Full Packet columns of Table 6.1). The TEMAC handles communication with the Ethernet transceiver and performs the final steps of transmitting the packet.

Packet checking works in the opposite manner of packet generation. As the TEMAC receives an Ethernet frame, it will provide the full-sized packet to the monitor module. The monitor will strip out unneeded information to condense the full-sized packet into a mini packet. The mini packet is then passed to a checker module through the RX AXI4-Stream Switch. The checker module compares the mini packet that was received against the previously stored copy. If there were any discrepancies between these two versions of the mini packet, the checker will report a failure event to the scoreboard module. The scoreboard modules keep a record of packets that are sent and received and is used to determine the health of the network system under test. The bit bucket module is used to capture packets that not associated with any checker, which indicates that the packets did not originate from the traffic generator or have been severely corrupted.

6.2 Sequencer

The sequencer is a simple module that is responsible for controlling the transmission rate of the entire traffic generator. This module has two CPU-accessible registers that are programmable by the host application: enable and delay. Enable is used to determine whether the traffic generator is actively transmitting packets. By default, the sequencer is disabled to allow the traffic generator

to be properly configured upon startup. The sequencer is also disabled while checking status registers to allow all in-flight packets to return, and to prevent erroneous status reads. The delay register is used to insert a global delay within the packet generator system to reduce transmission rates.

When enabled, a counter within the sequencer will increment each clock cycle, from zero until reaching the configured delay value. Upon reaching the delay value, the sequencer will send an activation signal to each of the generator modules and restart the counter. The activation signal cannot be sent until all generators assert that they are ready to receive the signal, implying that all previously generated packets have cleared this system. If any generator is not ready for the signal, the sequencer will stall. This design prevents the traffic generator from becoming internally congested, which would result in packets being dropped within the traffic generator. This would introduce significant challenges in determining network statistics.

The activation signal additionally contains an activation code, which is a random number generated by a 16-bit linear feedback shift register[1] (LFSR). When a generator receives the activation signal, it compares the activation code to its configured duty cycle. If the activation code is smaller than the duty cycle value, the generator will produce a packet. Otherwise, the code will be discarded and the generator will prepare to receive the next activation signal.

The duty cycle configuration is used to control the relative transmission rate between generators. A duty cycle of 0x0000 will effectively disable a single generator, while a duty cycle of 0xFFFF will cause the generator to transmit every time that it receives an activation code (resulting in maximum bandwidth utilization). However, because physical ports can be used by multiple flows simultaneously, the duty cycle does not provide an absolute measurement of bandwidth. For example, if two flows are configured to transmit through Port 0 and they both have a duty cycle of 0xFFFF, each flow will transmit with a bandwidth of 500 Mbps. This reduced bandwidth occurs even though both flows have the highest duty cycle possible. Because both flows are activated with the same frequency, they must share the bandwidth equally. However, if Flow 0 has a duty cycle of 0x3000 and Flow 1 has 0x1000, then they would have bandwidths of 750 Mbps and 250 Mbps, respectively. This bandwidth allocation is a result of Flow 0 being activated three times more frequently than Flow 1.

[1] An LFSR is a digital circuit that is used to generate psuedo-random numbers.

6.3 Generator

The generator module is responsible for creating mini packets according to the configuration of its network flow. The traffic generator design consists of eight generator modules, one for each traffic flow. The generator contains CPU-accessible registers for each of the flow configuration values specified in the flow configuration column of Table 6.1. These values are typically written shortly after startup by the management application. However, these configuration values could be changed dynamically with immediately visible results.

When the generator receives an activation signal from the sequencer, it will compare the activation code against the duty cycle to determine whether a packet should be generated. If it is determined that a packet should be created, the generator creates a new mini packet from the values in the flow configuration. As shown in Table 6.1, most of the mini packet header values can copied directly from the flow configuration registers. The notable exceptions are the payload length, payload seed, and packet ID fields.

The flow configuration allows a user to specify a packet's minimum and maximum length. When the generator creates a mini packet, it must select a length between these values. To do this, the generator maintains a length counter that is incremented after every packet is sent. This counter is initialized to the flow configuration minimum length and resets upon hitting the maximum length. The value of this counter is used to determine the length field of the mini packet.

When a packet is transmitted, it will contain a randomized payload generated by the driver module. The generator module generates a seed value to be later used by the driver in payload generation. The generator creates a seed by appending the flow identification number (a constant value) to the activation code received from the sequencer. The role of this payload seed value will be discussed later in this chapter.

The packet identifier is used to help the checker match transmitted and received packets. The generator module maintains a counter of every packet that is transmitted. This counter is used to populate the packet identifier field of the mini packet. The packet counter is 16-bits wide which is sufficiently large to ensure that no two packets that are currently in-flight will have the same identifier.

Once all of the fields within the mini header have been filled, it is wrapped in an AXI4-Stream packet and transmitted to both the checker and the driver. Each generator is associated

with a single checker which is connected through a dedicated point-to-point connection. However, generators are permitted to transmit packets from any of the Gigabit Ethernet ports, so an AXI4-Stream switch is used to facilitate the many-to-many connection.

6.4 TX Stream Switch

The TX Stream Switch is used to facilitate communication from the generator modules to the drivers. This module is necessary because a generator is permitted to transmit packets through any driver module. This core is a proprietary Xilinx IP module that is provided as part of the AXI4-Stream Infrastructure IP Suite.

The stream switch is configured to accept eight input streams (one from each generator) and provide four output streams (one to each driver). The switch uses a crossbar design to allow simultaneous communication between non-competing resources, allowing multiple drivers and generators to communicate in parallel wherever possible. Additionally, the switch provides round-robin arbitration to ensure the generators have equal access to every driver when resource conflicts arise.

The switch works by inspecting the mini packet's source interface field (stored in the AXI4-Stream packet's TDEST field) and mapping the value to a driver module. If the driver is available, the switch will allow the mini packet to be transmitted. However, if the driver is not currently available, the mini packet will wait until the driver becomes available and ultimately transmitted to the driver.

6.5 Driver

The driver module has two primary responsibilities: to generate full-length packets and to interface with the Ethernet subsystem. When the driver receives a mini packet from the generator, it will initially send a control sequence to the Ethernet subsystem indicating that there is information to be transmitted. Then, the driver will begin to assemble a full-length packet from the information that is present in the mini packet.

The mini packet contains the essential fields needed for forming an IPv4 packet, including addresses for routing and protocol identifiers. As the driver forms the full-length packet, it will

copy these values from the mini packet's fields. Some fields, like explicit congestion notification (ECN), differentiated services (DSCP), and flags, are only used in specific circumstances. As the Ethernet switch does not handle these features, they have not be enabled in the traffic generator. So when recording these values to the full-length packet, the driver fills these fields with default values. Other fields (such as length and checksum fields) are derived from the content of the packet, and as such, can be calculated on the fly.

The body of the packet, also known as the payload, is generated dynamically by the driver. The Ethernet switch design does not inspect or modify the payload of its traffic, so the actual payload content is relatively unimportant. The driver uses an LFSR to generate data to fill the packet's body. The mini packet's seed field is used to initialize the LFSR, and the length field is used to dictate how many iterations of the LFSR are to be used. Using a repeatable sequence, like the output from an LFSR, will allow the checker module to easily determine if the payload has been altered in-flight. Finally, packet and flow identifiers are appended to the end of the transmitted packet to ensure that it can be easily reidentified by the monitor. As this packet is constructed, it is transmitted to the Ethernet Subsystem.

6.6 Ethernet Subsystem

The Ethernet Subsystem is responsible for providing a convenient method for interfacing with Ethernet transceivers. This module is proprietary Xilinx IP [55], and acts as a wrapper for the TEMAC module as shown in Figure 6.2. This subsystem consists of the TEMAC core that is described in Section 4.2 in addition to a few other modules that handle data flow, clock domain crossing, and data buffering. This module consists of five primary interfaces: TX control and data streams, RX control and data streams, and an RGMII connection to the PHY. As mentioned previously, to send a packet using the Ethernet Subsystem, an application must send the proper command sequence through the control stream before sending data over the data stream. Similarly, when the Ethernet Subsystem receives data from the PHY, it will notify the monitor module through the control stream before sending the packet through the RX data stream.

The Ethernet subsystem also provides some low-level error detection capabilities. This module is responsible for appending the CRC checksum to outgoing packets and verifying the

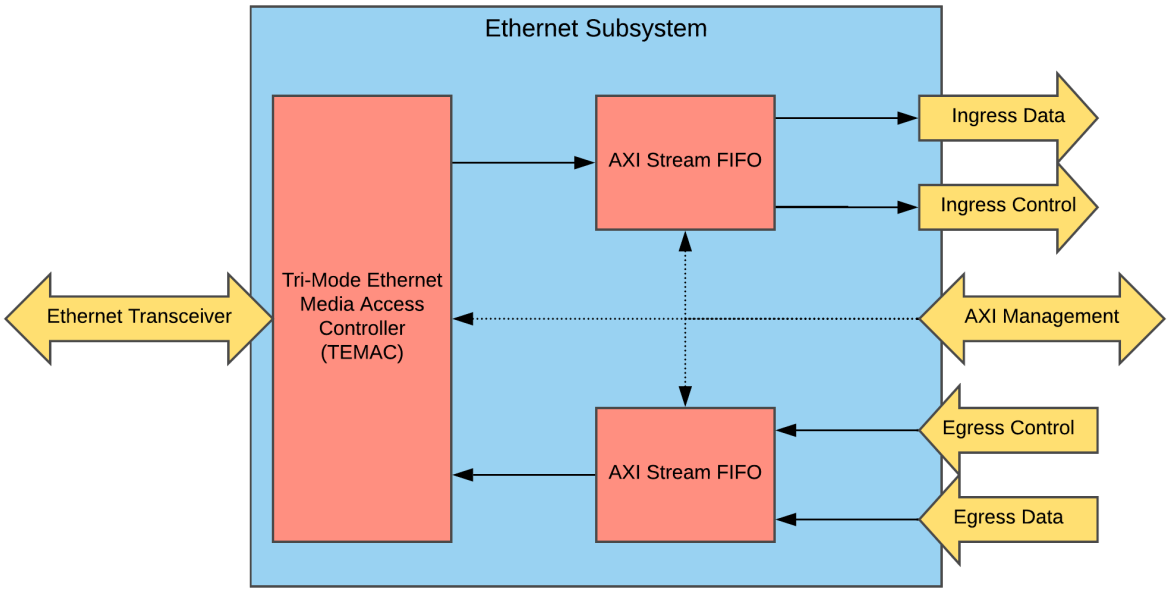

Figure 6.2: Ethernet Subsystem Block Diagram

checksum of incoming packets. Additionally, the Ethernet subsystem will detect if a received packet is excessively long or too short to be considered a valid packet.

6.7 Monitor

A monitor module interfaces with and receives full-length packets from an Ethernet Subsystem module. This module is the mirror of the driver, and is responsible for converting full-length packets into mini packets. These mini packets will later be transmitted to the checker module to be verified. Additionally, the monitor performs some error checking that would otherwise go undetected by the checker.

When the Ethernet Subsystem notifies the monitor of an incoming packet, the monitor will create a blank mini packet. The full-length packet is then streamed into the monitor, and the monitor will extract the relevant headers from the full-length packet to be written into the mini packet. These headers are at fixed locations within the packet, making it easy for the monitor to obtain the values that it needs.

However, not all headers in the full-length packet are used in the mini packet. To ensure that these headers have not been modified in-flight, they will be examined by the checker module.

Any header field that was filled with a constant value will be checked to ensure that the same constant value is still present. Calculated header fields will be recalculated by the monitor to verify that errors have not been introduced into these fields. Any mismatched detected by the monitor will be recorded as a failure.

Once all of the headers of the packet have been processed, the monitor will proceed to read and validate the body of the packet. The first word (four bytes) of the payload is the value of the seed that was used by the generator to fill the packet body. The monitor will extract this value to store in the mini packet and to seed its own LFSR. Because the driver and the monitor now have the same LFSR seed, they are guaranteed to produce the same data sequence. As each subsequent word is read out of the payload, it is compared to the value produced by the LFSR. Any discrepancies between the packet body and the LFSR are recorded as a payload error.

The final word of the packet contains the packet and flow identifiers. These values are extracted from the payload and will be stored in the mini packet. To complete the verification of the packet, the mini packet will be encapsulated in an AXI4-Stream packet. The flow id from the mini packet will be used as the TDEST field, and the mini will then be sent to the RX Stream Switch to be routed to the correct checker module.

6.8 RX Stream Switch

The RX Stream Switch is identical to the TX Stream Switch, but is just configured to connect monitors to checker modules. This module is necessary to allow a flow to be received by any port of the traffic generator, allowing flexibility in testing designs. This stream switch is configured to use flow identifiers to route mini packets to the appropriate checker. The routing of the stream switch is configured statically at design time.

Recognizing that the flow id field of a packet may be corrupted in-flight, the RX Stream Switch provides a catch-all route. Mini packets that have an invalid flow id will be routed to the bit bucket module rather than a checker. This allows all received packets to be counted by the traffic generator.

6.9 Checker

The checker is responsible for comparing received packets to their golden values. The module has two ports for receiving data, one from the generator and the other from the monitor. Internally, the checker consists of a large table that is used to store golden packets until they are ready to be checked. The checker also contains the circuitry needed to compare two packet headers.

When the checker receives a packet from the generator, it will first look at the packet identification number. This number is used to generate an address in the golden packet table. Due to memory constraints, only 128 packets can be in-flight per flow at an given moment before a collision occurs. When storing a packet, the table is first checked to ensure that no other packet is currently residing at that address. If there is a packet waiting, it is considered to have expired and is reported to the scoreboard. Once space is available, the new header is stored. Packet identifiers are incremented with each packet sent, so the table address is determined by taking the modulo of the packet identifier with the table width. The incremental nature allows the table to act as a circular buffer, forcing the oldest packet to expire first.

When receiving a mini packet from the monitor, the checker will use the received packet's identification number to request the golden header from the memory table. Assuming the correct header is found, each of the header values will be compared. Any discrepancies between the headers indicates in-flight corruption, and is reported to the scoreboard. The golden mini packet is then removed from the table.

6.10 Scoreboard

The scoreboard keeps track of statistics of received packets and is accessible to the CPU through the AXI bus. Each flow has its own scoreboard which records the number of packets transmitted, how many packets were received without errors, and how many packets were received with errors. For any packets received with errors, the nature of each error is recorded (for example, which header layer is corrupted). Comparing the number of packets transmitted to the number received provides a good image of the status of the device being tested. When reading statistics from the scoreboard, the management system will clear the counters after each read. This ensures that each read of the scoreboard is associated with a specific timeframe.

6.11 Bit Bucket

The bit bucket is a simplified version of the scoreboard that is used for recording corrupted and miscellaneous packets. As it is possible that a packet is corrupted beyond recognition, it would be impossible to determine the exact nature of the error. When a monitor receives a packet that has an unrecognized value in the flow identification field, the packet is routed to the bit bucket rather than a specific checker. This module collects these packets and reports the number of unidentifiable packets. As there is no way to match one of these packets to a golden value, this module does not provide failure mode statistics.

6.12 Management Software

The ZedBoard's ARM processor is configured with a Linux system that runs a management application for the traffic generator. This program is designed to make the traffic generator easily controllable through a network connection. The application is written in Python, using Flask [56] and uWSGI [57] to provide an HTTP server for receiving network commands. The application uses memory mapped I/O to read from and write to the traffic generator hardware registers to perform actions.

Upon startup, the ZedBoard's programmable logic will automatically be programmed with the traffic generator design through the bootloader. Once the Linux system comes online, the management software will start automatically as a daemon. The management software will initialize the flow configuration registers of the traffic generator using the values stored in a configuration file on the file system. After initializing the hardware, the management software will be prepared to receive network commands.

The management software accepts four primary commands through the network: start, stop, check status, and reboot. These commands are issued by sending GET and POST requests to specific HTTP endpoints. When a start or stop command is received, the management software will write a 1 or a 0 to the sequencer's enable register. This will cause the sequencer to immediately start or to stop sending packets.

The check status command is a little more involved. Initially, the software will disable the sequencer and sleep for 0.1 seconds to allow the network system to finish processing all traffic in

flight. Then the system will read the status registers of each scoreboard, bit bucket, and Ethernet subsystem module. The values are then inspected to determine if an error has occurred and then the values are then serialized into JSON[2] and returned to the requester. Afterward the sequencer is enabled again to resume packet transmission.

The reboot command will cause the management software to issue a reboot command to the operating system. This command can be used in reliability testing should the traffic generator ever stall and need to be restarted remotely or programmatically.

6.13 Implementation

The traffic generator hardware was designed in C++ to be used with High Level Synthesis (HLS). Vivado HLS 2018.1 was then used to compile the traffic generator modules into register transfer level (RTL) Verilog modules. These modules were then assembled into a top-level project to be synthesized and implemented using Vivado 2018.1.

The traffic generator was designed to support a variable number of traffic flows. A new flow can be supported by instantiating a new generator, checker, and scoreboard. The RX and TX stream switches would just then need to be modified to accept an additional connection. When targeting the ZedBoard, the programmable logic could support a maximum of eight flows due to hardware constraints.

Table 6.2: Traffic Generator Resource Utilization

Zynq-7000 Device	Used		Total
Slices	13,284	(99.9%)	13,300
Registers	62,592	(58.8%)	106,400
LUTs	38,480	(72.3%)	53,200
Block Memories	52	(37.1%)	140
Global Clock Buffers	9	(28.1%)	32
I/O	69	(34.5%)	200

Resource utilization for the traffic generator design is shown in Table 6.2. Slice utilization is the limiting factor that prevents this design from expanding. A larger FPGA or design

[2]JavaScript Object Notation (JSON) is lightweight data-interchange format that is frequently used in HTTP communication.

modifications would need to be employed in order to expand capabilities of the traffic generator. The AXI4-Stream switches and the Ethernet Subsystem modules are the largest consumer of slice resources in this design.

The traffic generator was designed specifically for collecting low-level reliability statistics of an Ethernet switch network system. However, with some modifications, this traffic generator could be used to test various metrics and different types of network systems. A network-level router could be tested by modifying the checker to ignore MAC addresses and to expect some header modifications (a TTL decrement, for example). Quality of Service (QoS) measurements could be obtained by enabling the DSCP field in the generator module and using a global clock to record round-trip time of packets. Network system throughput and performance could be tested by modifying the scoreboard to record RX and TX bandwidth rather than error types.

As is, the traffic generator has been configured to collect and classify low level failure statistics of network traffic at line rates. This information is useful in observing and measuring the failure modes of network systems. Using this traffic generator, we will exercise and monitor the operation of an FPGA-based network system when subjected to ionizing radiation.

CHAPTER 7. EXPERIMENTAL RESULTS

This chapter presents the results of our soft-error performance experiments. First, the methodology for testing these designs using both fault injection and neutron irradiation will be described. These two methods are frequently used in device reliability testing in order to reliably collect large amounts of upset data in a short period of time. Then, the metrics for measuring radiation and the results of these experiments will be described. Finally, the results of these experiments will be presented and analyzed. These results include error correction metrics showing how TMR can be used to increase the reliability of a design and error detection metrics to show that DWC can improve a system's ability to detect faults.

7.1 Test Methodology

There are two main methods used for testing soft-error reliability of FPGA designs: fault injection and neutron irradiation. Neutron irradiation is the gold standard for soft-error reliability testing as it provides the most accurate representation of a system's response to ionizing radiation. This test is performed by placing the test system in an accelerated radiation environment and observing its response. Given the high neutron-flux levels of these environments, years worth of neutron upset data can be collected in a very short time. Fault injection is often used because it is more accessible and cheaper than using neutron irradiation. This type of experiment is performed by simulating an SEU by modifying an FPGAs configuration memory while it is being tested [58]. These two types of test were used to measure the soft-error sensitivity of the baseline and mitigated Ethernet switch designs. This section will outline the methodology for both tests and show how the tests are used to measure how these FGPA designs respond to radiation-induced upsets.

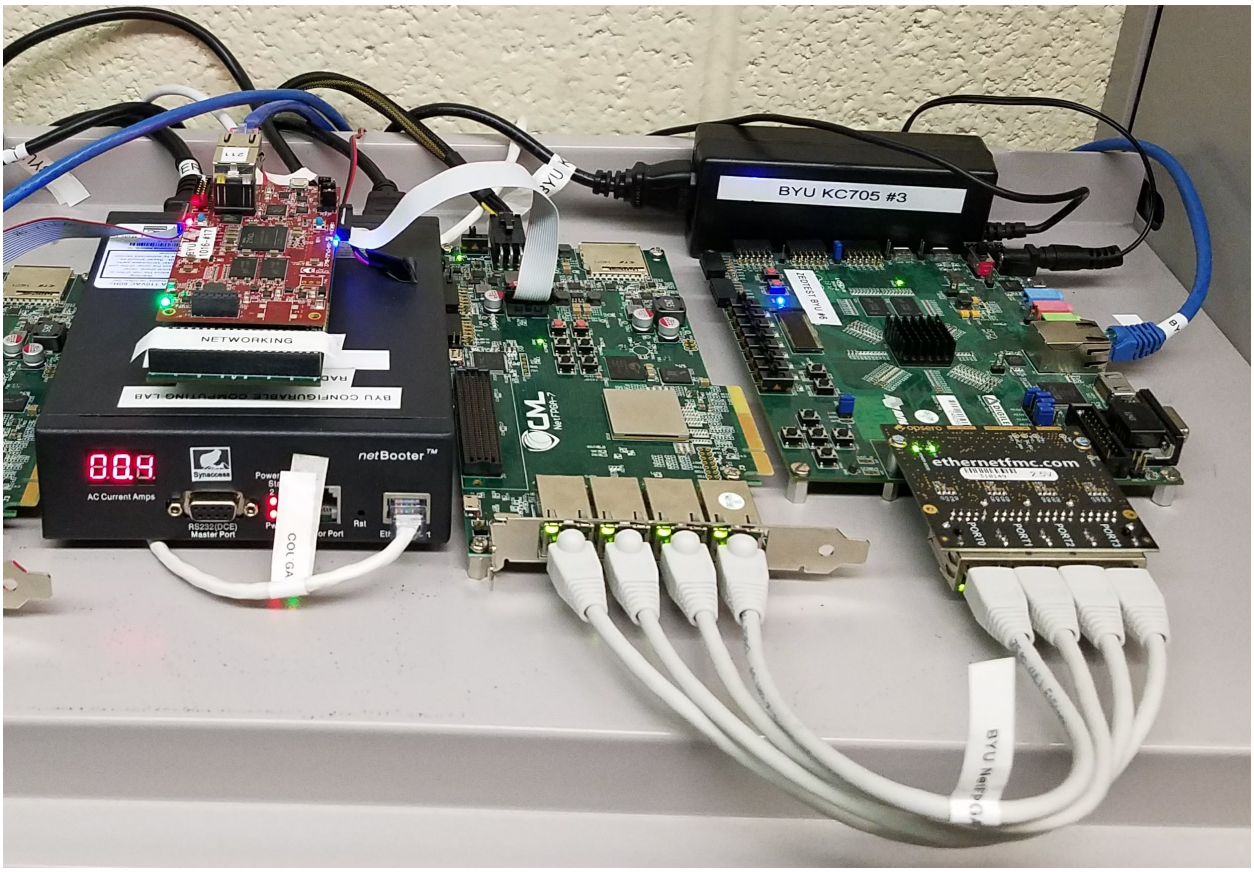

Figure 7.1: Fault Injection Reliability Test Framework

7.1.1 Fault Injection Testing

Hardware-based fault injection is a soft-error reliability testing methodology that aims to mirror the effects of radiation-induced upsets. The test is performed by injecting upsets into the configuration memory using partial reconfiguration. The location of the injected upset is selected randomly in order to mimic naturally-occurring SEUs. Using the JCM, a frame of configuration memory of a pre-programmed FPGA is read, a portion of the frame is inverted, and then the modified frame is rewritten to the FPGA's configuration memory. The FPGA application is then observed to determine the effect of this upset.

The steps of the fault injection reliability campaign are outlined in Figure 7.2. Initially, the JCM programs the NetFPGA with the bitstream associated with the DUT. Then the JCM instructs the traffic generator to start creating traffic for the switch to process. To ensure that the design was programmed correctly, the JCM will check the status registers on the Ethernet switch design and

on the traffic generator. If a status register reads bad, this initialization process will repeat until the switch is in a known good state.

Once the switch has been initialized successfully, a fault will be injected by flipping a single random configuration bit and waiting a fixed amount of time to ensure that a potential error propagates to the output. Faults are not permitted to accumulate in the design, so every fault that is injected must be corrected. Upsets are repaired by re-inverting the single bit that was previously upset. After this fast scrub, the JCM then checks the status of the traffic generator to determine if the NetFPGA has experienced a traffic failure. The status is recorded, and the JCM will attempt to automatically correct any failure with an incremental repair routine. This process repeats until the test concludes.

The incremental repair routine serves two primary purposes: reduce the time required to bring the DUT into a known good state, and to determine the effect that each injected fault has on the design. After each attempted repair, the JCM checks the system status to determine if the repair has been successful. If the repair succeeds, the fault injection campaign is allowed to continue. Otherwise, a more thorough repair strategy will be employed. Initially, a full device scrub is run to determine if any additional faults occurred in configuration that were not corrected by the fast scrub. This repair mode would also catch any transient failures that only exist while the fault is present. Should the full scrub fail, the Ethernet switch will be reprogrammed. This will reset BRAM memories and flip-flop values in addition to resetting the state of the configuration memory. This repair mode is useful for catching errors that have corrupted device state that is not automatically repaired using self-synchronization. Beyond this, the Ethernet switch will be rebooted and reprogrammed. In a fault-injection campaign, this recovery mode should only be required if the state of a peripheral device is corrupted. The final repair method is to reinitialize the entire test by rebooting the traffic generator. This approach is seldom required, but somewhat alarming as an upset in the Ethernet switch should not cause failure in the traffic generator. However, this approach may be useful for re-establishing links, resetting PHY states, and synchronizing the test.

7.1.2 Neutron Irradiation

The commonly accepted standard for testing terrestrial-radiation-induced soft errors in semiconductor devices is through the use of accelerated neutron and proton radiation sources [25],

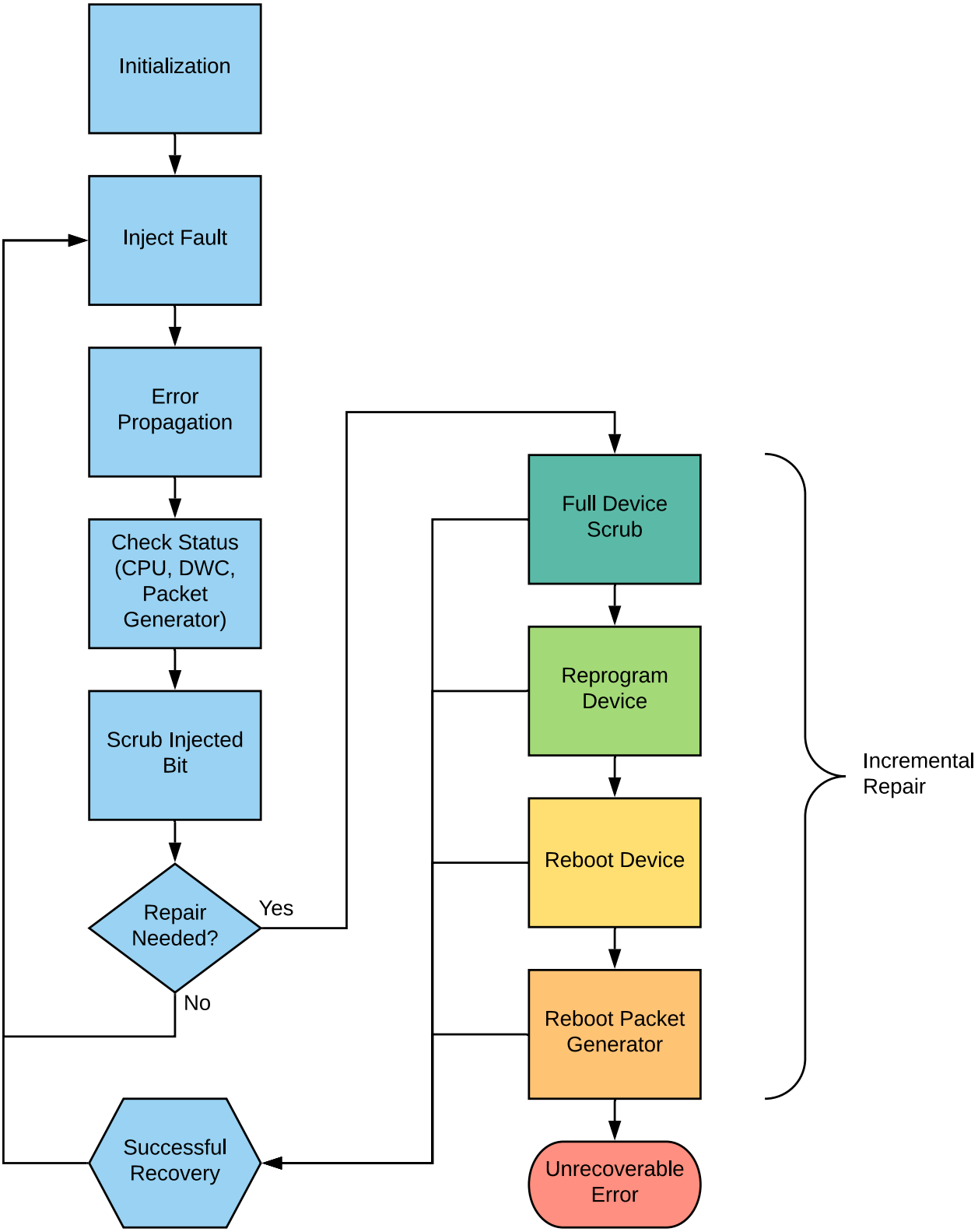

Figure 7.2: Fault Injection Reliability Test Flowchart

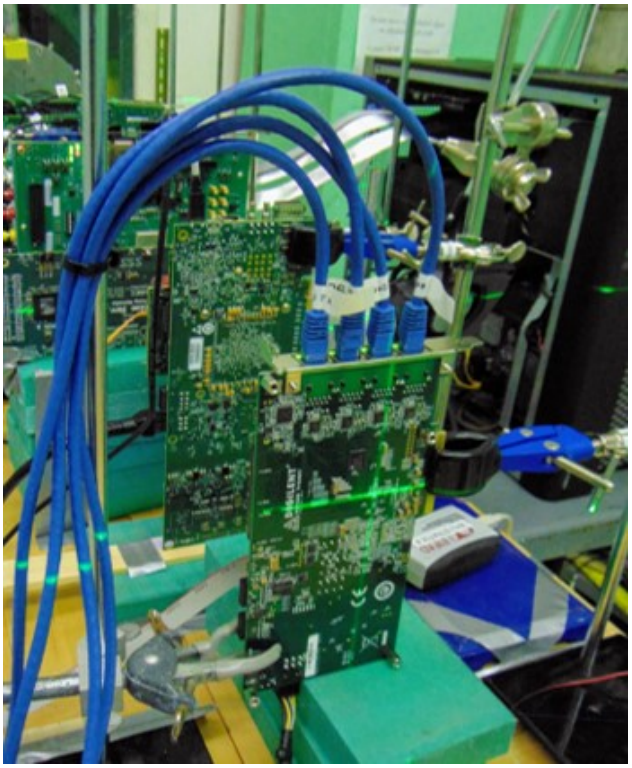

Figure 7.3: NetFPGA Positioned for Neutron Radiation Experiment

[59]. A spallation neutron source with a wide energy spectrum similar to that found in terrestrial environments is located at the Neutron and Nuclear Science (WNR) Facility at LANSCE. The high neutron flux available at this facility allows devices to be tested at accelerated rates. Radiation testing is costly, but provides important insight into the expected reliability of a design.

Neutron radiation testing was conducted on the Ethernet switch design in December of 2018 at LANSCE on the ICE II beam path. This beam path offers a neutron flux that is typically 5×10^7 times more intense than the flux seen at ground level [60]. This significantly accelerated rate allows for years worth of data to be collected in a short amount of time. The FPGA was aligned perpendicular to the two inch collimated beam (see Figure 7.3). The distance of the FPGA from the beam source was recorded and the flux of the beam was degraded in analysis to compensate for the distance [25]. All three versions of the design (i.e., baseline, TMR, and DWC) were tested for traffic failure mode SEU sensitivity.

The methodology for testing the reliability of a device in a neutron beam (shown in Figure 7.4) is similar to the process used in fault injection, but contains a few notable differences. The

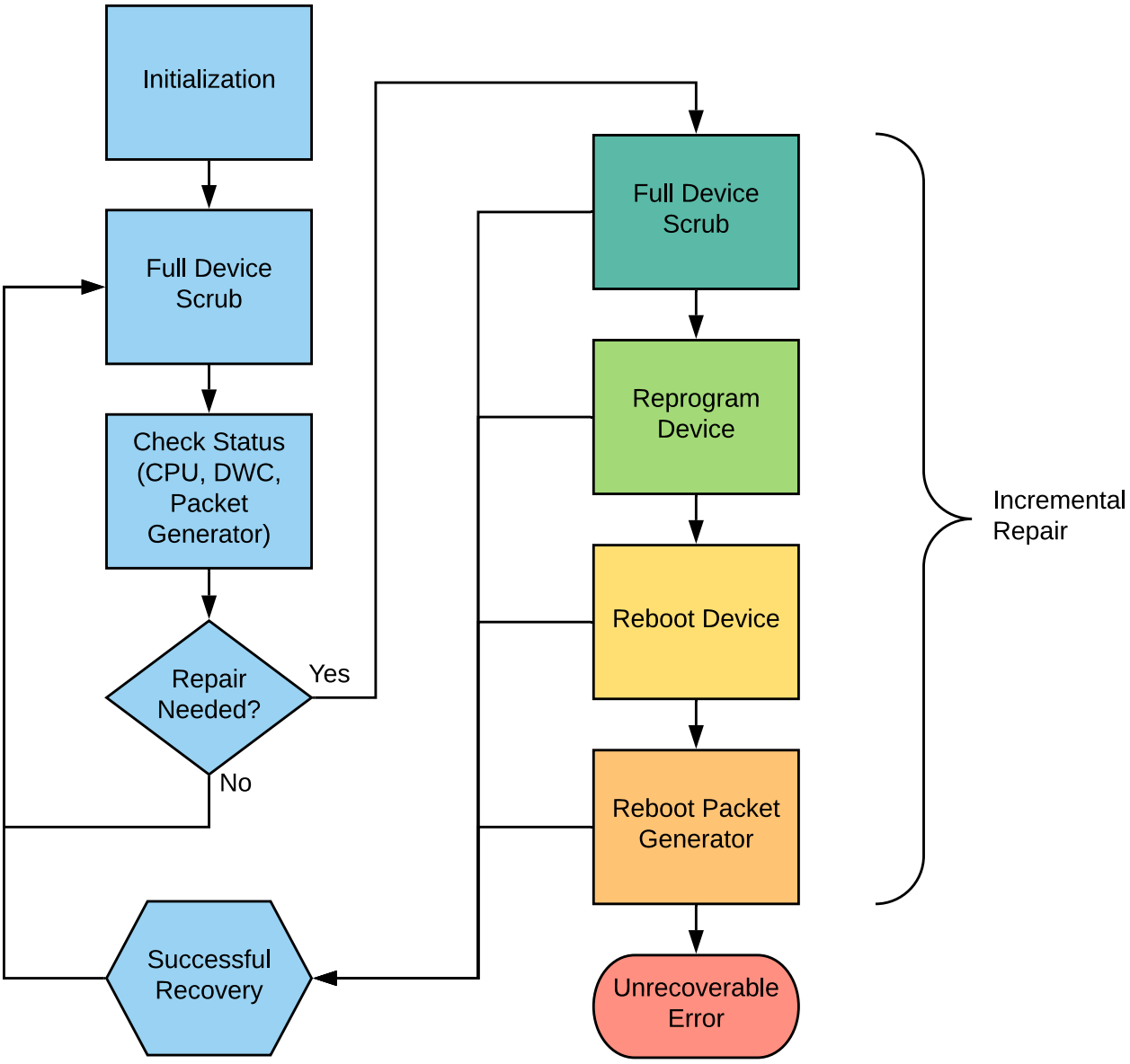

Figure 7.4: Neutron Beam Reliability Test Flowchart

most distinct difference is that the neutron beam will induce faults within the design, making it unnecessary for the JCM to introduce additional faults. Unlike fault injection, we do not know where or when a fault will occur within the configuration memory. This makes fast localized scrubbing unsuitable for this methodology and a full device scrub must be used instead. Because full device scrubbing takes approximately two seconds, it is unnecessary to artificially insert an error propagation delay. In standard operating mode, the JCM will simply alternate between scrubbing the FPGA memory and checking its status.

The neutron irradiation recovery mechanism is identical to that used for fault injection, but the differences between each recovery mode play more importance. Due to challenges in reading or modifying BRAM values of running applications through JTAG, the fault injection experiment exclusively targets configuration memory. A neutron does not distinguish between memory types and has no issues upsetting values within a BRAM. This upset will not be detected or corrected through a full device scrub and would only be recovered by reprogramming the device. An upset could potentially affect control registers within the FPGA which would need to be corrected by rebooting the entire development board. Having a robust repair strategy allows the experiment to handle a wide range of errors that result from radiation-induced upsets.

7.2 FPGA Network Failure Modes

The purpose of an Ethernet switch is to accurately relay Ethernet frames from a source port to its intended destination port. Any behavior that deviates from this expectation is considered a failure condition. The traffic generator has been configured to generate traffic that is relatively evenly distributed between egress ports. This means that all traffic should reach its destination under normal operation and that packet loss is not expected. To confirm this, the traffic generator was run for a week without performing any fault injection or neutron irradiation. The Ethernet switch was able to sustain the load throughout the duration of the test and not a single packet was dropped. Therefore, any failure conditions observed during reliability testing are assumed to be the direct result of an upset.

The failures observed during the reliability tests can be categorized into one of three classifications: data corruption, misrouting, and packet loss. A data corruption failure occurs when the packet that was received by the traffic generator is somehow modified from the packet that was transmitted. This category includes both payload and header data corruption, and likely indicates that an upset affected the datapath of the system. A packet is considered misrouted when it does not arrived at its intended port, but everything else about the packet has remained intact. This type of failure is most likely to result from an upset affecting routing tables or other control logic. Packet loss occurs when a packet enters the Ethernet switch but it is never forwarded to an external port. An error in either the data path or control logic could lead to traffic loss.

7.3 Metrics

In order to understand the significance of the results of this work, an understanding of radiation and reliability metrics is needed. These metrics will be used in reporting and analyzing the results of the soft-error reliability tests.

While there are many ways to measure radiation, *flux* and *fluence* are commonly used in conjunction with soft-error reliability. Flux is the rate at which radiation passes through a given area and is generally measured in particles/cm^{2}hr. Fluence is related to flux and refers to the total number of particles that pass through a given area within a set time frame. Fluence is measured in particles/cm^2 and can be calculated by integrating flux with respect to time or by simply multiplying average flux by the time frame. As high-energy neutrons are the primary cause of SEUs in terrestrial applications, this work will report neutron flux and fluence.

The primary measure of soft-error reliability in neutron irradiation experiments is *cross section*. Cross section is measured in cm^2 and refers to a hypothetical area that if a particle passes through, would cause an event. In the case of a failure cross section, the event would result in a functional failure of the device. A larger cross section signifies that the design is more sensitive to ionizing radiation. The failure cross section is estimated by diving the number of observed failures by the total fluence produced by the radiation beam:

$$\sigma = \frac{\text{Number of Failures}}{\text{Total Fluence}}. \tag{7.1}$$

Sensitivity is a measure of soft-error reliability that is often used to report fault injection results. This metric refers to the percentage of FPGA configuration bits that cause a design to fail when upset. The sensitivity of a design is determined by dividing the total number of observed failures by the total number of faults injected throughout the duration of test:

$$\text{Sensitivity} = \frac{\text{Number of Failures}}{\text{Number of Injections}}. \tag{7.2}$$

The sensitivity of a design is dependent upon FPGA size and could fluctuate significantly depending on device utilization. This makes sensitivity an adequate metric for making quick comparisons

between design variants implemented on the same device. However, this data alone is not sufficient for assessing the reliability of an FPGA design.

Failure in Time (*FIT*) is the standard industry value used in reliability engineering to express the failure rate of electronic devices. FIT rate refers to the expected number of failures occurring per one billion device-hours [38]. The FIT rate of an FPGA design can be derived from neutron irradiation data using the failure cross section and neutron flux:

$$\text{FIT} = \sigma \times \text{Neutron Flux} \times 10^9. \tag{7.3}$$

Neutron flux can be adapted to the environment where the device will be deployed in order to obtain an accurate FIT rate, but a flux of 13 neutrons/cm^{2}hr is typically used for terrestrial applications.

FIT can also be calculated from fault injection results by first estimating the cross section of a design using the sensitivity, total number of CRAM bits, and the neutron bit cross section [38]:

$$\sigma_{\text{est}} = \text{Sensitivity} \times \text{Number of CRAM Bits} \times \text{Neutron Bit Cross Section}. \tag{7.4}$$

Then the estimated cross section can be used with Equation (7.3) to estimate the FIT rate of the design.

As the measurements reported in this chapter are based on experimental observations, they are only estimates of an actual value. A *confidence interval* is used to quantify a range that the actual parameter is likely to fall within. The confidence interval for a cross section measurement is calculated using the following:

$$95\%\ \text{C.I.} = \frac{2 \times \sqrt{\text{Observed Failures}}}{\text{Total Fluence}}. \tag{7.5}$$

Additional information on confidence interval calculation can be found in [59].

Self detection is a metric used to indicate a system's ability to determine the presence of an fault. In this experiment, two methods are used to perform error detection: system status registers being monitored by a soft-core processor monitoring and DWC. A system's self detection rate is the percentage of system errors that are correctly detected using an internal mechanism and is

calculated using the following equation:

$$\text{Detectability} = \frac{\text{Number of Detected Errors}}{\text{Total Number of Failures}}. \tag{7.6}$$

All of these metrics will be used in the following section to compare and contrast the reliability and self detection rates of different designs.

7.4 Soft-Error Reliability

Soft-error reliability in FPGA-based systems refers to the capability of continuing proper operation despite the presence of configuration upsets. This section will present the soft-error reliability experiment results for the baseline and TMR Ethernet switch designs. The baseline design design offers no form of error mitigation while the TMR design has been protected by a partial application of TMR, limited by proprietary logic and clock domain crossing. In both fault injection and neutron irradiation testing, TMR shows significant improvements in reliability over the baseline design.

Table 7.1: Baseline and TMR Ethernet Switch
Fault Injection Results

Ethernet Switch Design	Baseline	TMR
CRAM Faults Injected	876,416	770,370
Total Failures	12,001	1,955
Equivalent Fluence (n/cm^2)	2.11E+12	1.86E+12
Sensitivity	1.37%	0.25%
Cross Section (cm^2) (95% Conf. Interval)	5.68E-9 (5.57E-9, 5.78E-9)	1.05E-9 (1.00E-9, 1.10E-9)
Failure Rate (FIT) (95% Conf. Interval)	73.8 (72.5, 75.1)	13.7 (13.1, 14.3)
Improvement (95% Conf. Interval)	1.00× (0.96×, 1.04×)	5.40× (5.07×, 5.75×)

A fault injection campaign was run on the baseline Ethernet switch and the TMR mitigated switch for 76 and 58 hours, respectively. The results of these experiments are shown in Table 7.1. After the test was run, the log files were parsed to extract the total number of faults injected and

the results of those injections. The equivalent fluence measure is included in order to provide the same metrics across both fault injection and neutron irradiation tests. While there is no actual fluence associated with a fault injection campaign, this value can be estimated using the neutron bit cross section [38] to determine the equivalent fluence for each injected fault. Ultimately, the TMR design achieves a 5.40× improvement in reliability over the baseline design. While this may not seem like a significant improvement alone, it is notable considering that only 70% of the design components were mitigated.

Table 7.2: Baseline and TMR Ethernet Switch
Fault Injection Failure Modes

Ethernet Switch Design		Baseline	TMR
Data Corruption	Failures	8,856	305
	Sensitivity	1.010%	0.040%
	Cross Section (cm^2)	4.19E-09	1.64E-10
	Failure Rate (FIT)	54.46	2.13
Misrouting	Failures	140	21
	Sensitivity	0.016%	0.003%
	Cross Section (cm^2)	6.62E-11	1.13E-11
	Failure Rate (FIT)	0.86	0.15
Packet Loss	Failures	3,005	1,629
	Sensitivity	1.010%	0.040%
	Cross Section (cm^2)	1.42E-09	8.77E-10
	Failure Rate (FIT)	18.48	11.40

For each failure that occurred in the fault injection test, the traffic generator logs were consulted to determine how the Ethernet switch failed. These distribution of failure modes is displayed in Table 7.2. From the Baseline design to the TMR design, there is a significant decline in occurrence of data corruption and misrouting failure modes. However, the packet loss failure mode does not see the same levels of improvement.

These two designs were also subjected to an accelerated radiation environment at Los Alamos National Laboratories. The baseline design received 13 hours of accelerated neutron exposure while the TMR design received 38 hours. The results from the neutron irradiation test are shown in Table 7.3. Unsurprisingly, the results are very similar to the fault injection experiment.

Table 7.3: Baseline and TMR Ethernet Switch
Neutron Irradiation Results

Ethernet Switch Design	Baseline	TMR
Fluence (n/cm^2)	3.43E+10	8.59E+10
CRAM Upsets	10,682	26,836
Total Failures	113	56
Sensitivity	1.06%	0.21%
Cross Section (cm^2) (95% Conf. Interval)	3.30E-09 (2.68E-09, 3.92E-09)	6.52E-10 (4.78E-10, 8.26E-10)
Failure Rate (FIT) (95% Conf. Interval)	42.8 (34.8, 50.9)	8.5 (6.2, 10.7)
Improvement (95% Conf. Interval)	1.00× (0.68×, 1.46×)	5.05× (3.24×, 8.20×)

The TMR shows a slightly decreased improvement, but this can be attributed to the neutron beam being able to upset more parts of the device that cannot be tested in fault injection.

Table 7.4: Baseline and TMR Ethernet Switch
Neutron Irradiation Failure Modes

Ethernet Switch Design		Baseline	TMR
Data Corruption	Failures	93	15
	Sensitivity	0.871%	0.056%
	Cross Section (cm^2)	2.71E-09	1.75E-10
	Failure Rate (FIT)	35.26	2.27
Misrouting	Failures	5	5
	Sensitivity	0.047%	0.019%
	Cross Section (cm^2)	1.46E-10	5.28E-11
	Failure Rate (FIT)	1.90	0.76
Packet Loss	Failures	15	36
	Sensitivity	0.140%	0.134%
	Cross Section (cm^2)	4.38E-10	4.19E-10
	Failure Rate (FIT)	5.69	5.45

The failures for the radiation test are categorized in Table 7.4. Again, the results of this test are very similar to the fault injection test. The TMR design shows significant improvement in the data corruption and misrouting failure modes, but even less improvement is seen in the packet loss failure mode.

7.5 Soft-Error Detection

In contrast with soft-error reliability, detection is less concerned with a system's ability to tolerate an error and is more focused on accurately determining the presence of an error. This error detection could then be used to trigger a repair routine, flag the current data as invalid, or activate a failover system. The actual response to a detected failure is dependent upon the application and is unimportant for this experiment.

To demonstrate the effectiveness of DWC in providing soft-error detection, the DWC design will be tested alongside the baseline Ethernet switch design. The baseline design uses a CPU-based packet counter technique to detect errors. The DWC design uses a partial application of DWC in addition to the CPU-based technique to detect the presence of errors. These designs will be tested to determine how accurately the systems can detect the presence of fault through fault injection and neutron irradiation. This section presents the results from these experiments along with some analysis.

Table 7.5: Baseline and DWC Ethernet Switch Fault Injection Results

Ethernet Switch Design	Baseline	DWC
CRAM Faults Injected	876,416	834,569
Total Failures	12,001	10,562
Equivalent Fluence (n/cm^2)	2.11E+12	2.01E+12
Sensitivity	1.37%	1.27%
Cross Section (cm^2)	5.68E-9	5.25E-09
Failure Rate (FIT)	73.8	68.2
Improvement	1.00×	1.08×
Self-Detected Failures	1,840	9,612
Percentage (95% Conf. Interval)	15.33% (14.62%, 16.05%)	91.01% (89.15%, 92.86%)
Undetected Failure Rate (FIT) (95% Conf. Interval)	62.5 (61.2, 63.7)	6.1 (5.7, 6.5)
Improvement (95% Conf. Interval)	1.00× (0.98×, 1.02×)	9.41× (7.74×, 11.96×)
False Positive Events	440	9,958
Cross Section (cm^2) (95% Conf. Interval)	2.08E-10 (1.88E-10, 2.28E-10)	4.95E-09 (4.85E-09, 5.05E-09)

The fault injection results for the baseline and DWC experiments are shown in Table 7.5. The baseline design was tested for 96 hours while the DWC design was tested for 94 hours. Like the soft-error reliability experiment, this table also shows equivalent fluence for use in generating useful and comparable statistics. Although the aim of DWC is not to provide increased reliability, these metrics are included to reveal how the application of DWC impacts the reliability of a device. Between the baseline and DWC designs, there is no significant change in reliability measurements. This is to be expected as DWC does not add any features to improve reliability, nor does it increase the target size of the operating circuit.

However, the DWC design does offer significantly improved error detection. The DWC design is able to correctly detect 91% of errors while the baseline design only detects 15% of errors. This is a notable improvement in error detection, especially considering that DWC could only be applied to 56% of the design. As expected, approximately 51% of the events detected by DWC were actually false positives. These false positives are the result of upsets that occur in the duplicated logic that do not affect the output of the system. False positives are not necessarily an issue for DWC designs, but the information is included here for informational purposes.

Table 7.6: Baseline and DWC Ethernet Switch Fault Injection Error-Detection Distribution

Ethernet Switch Design	Baseline	DWC
CPU Only	1,840 (15.3%)	559 (5.3%)
DWC Only	-	7,931 (75.1%)
Both CPU and DWC	-	1,122 (10.6%)
Undetected	10,161 (84.7%)	950 (9.0%)

The distribution of error detection by system is shown in Table 7.6. This is useful for determining the effectiveness of each error detection system. In both the baseline and DWC designs, the CPU correctly detects approximately 15% of all errors. However, there is a 10% overlap in error detection between the DWC and CPU systems. This means that the DWC would assumedly detect 85% of errors on its own, showing just how effective DWC is in providing accurate error detection.

Table 7.7: Baseline and DWC Ethernet Switch
Neutron Irradiation Results

Ethernet Switch Design	Baseline	DWC
Fluence (n/cm^2)	3.43E+10	1.15E+11
CRAM Upsets	10,682	34,390
Total Failures	113	402
Sensitivity	1.06%	1.17%
Cross Section (cm^2)	3.30E-09	3.50E-9
Failure Rate (FIT)	42.8	45.5
Improvement	1.00×	0.94×
Self-Detected Failures	22	340
Percentage (95% Conf. Interval)	19.47% (11.17%, 27.77%)	84.85% (75.40%, 93.75%)
Undetected Failure Rate (FIT) (95% Conf. Interval)	34.5 (27.3, 41.7)	7.0 (5.2, 8.8)
Improvement (95% Conf. Interval)	1.00× (0.81×, 1.23×)	5.22× (2.94×, 14.22×)
False Positive Events	16	359
Cross Section (cm^2) (95% Conf. Interval)	4.67E-10 (2.33E-10, 7.00E-10)	3.12E-09 (2.80E-09, 3.45E-09)

The baseline and DWC designs were tested in an accelerated neutron irradiation environment for 13 hours and 43 hours, respectively. The results of this experiment are shown in Table 7.7. The neutron radiation results are very similar to the fault injection results. There is very little difference between the sensitivity of and DWC design and the baseline design, as expected. The DWC design shows significant improvement in detectability by detecting an additional 66% of errors that would otherwise be undetected using the CPU alone. We also see a 51% false positive rate in the neutron beam test, which is similar to the rate seen in fault injection.

Table 7.8: Baseline and DWC Ethernet Switch Neutron Irradiation
Error-Detection Distribution

Ethernet Switch Design	Baseline	DWC
CPU Only	22 (19.5%)	18 (4.5%)
DWC Only	-	285 (70.9%)
Both CPU and DWC	-	37 (9.2%)
Undetected	91 (80.5%)	62 (15.4%)

Error detection for the neutron radiation test is categorized by system in Table 7.8. These results are similar to the results seen in fault injection, but show slightly degraded performance in both the CPU and DWC. This is not unexpected as radiation tests typically yield slightly worse results than fault injection. In this case, the degraded performance results in more errors going undetected in both designs.

The results from both the error correction and the error detection tests are very promising. In the error correction test, the TMR design shows a 5.05× improvement in reliability over the baseline design. For the error detection test, the DWC design is able to detect 5.22× as many errors as the baseline design. These radiation results are corroborated by fault injection data, providing a high degree of confidence in their accuracy. These results are very promising considering that each of the mitigation techniques could only be applied to a portion of the Ethernet switch design. It is likely that a larger application of mitigation technique would result in even better results.

CHAPTER 8. CONCLUSION

This thesis presents a framework for monitoring and measuring the effects of soft errors in FPGA-based network systems. This framework consists of a network development board and a traffic generator that can be used in both fault injection and irradiation experiments. The network development framework provides the necessary hardware for implementing a wide variety of network systems.

To demonstrate the effectiveness of this framework, the open-source NetFPGA code repository was used to design an Ethernet switch. Two common soft-error mitigation techniques, DWC and TMR, were applied to this design to create mitigated systems. These network systems were then tested using fault injection and neutron irradiation and observed to determine how the network systems respond to soft errors and the effectiveness of these mitigation techniques at a system level.

The results from these tests provide great insight into understanding how SEUs cause FPGA-based network systems to fail, how frequently these failures can be expected in the field, and how common mitigation techniques can be used to improve these systems. The baseline Ethernet switch design has an estimated failure rate of 42.8 FIT for neutron irradiation at sea level. When mitigated with TMR, the design improves by 5.05×, reducing the estimated failure rate to 8.5 FIT. Applying DWC makes no impact on the reliability of the design, but it does improve the design's ability to detect failures by 5.22×. This reduces the estimated undetected failure rate of the Ethernet switch to 7.0 FIT. These results are very promising, especially considering the limited application of these mitigation techniques.

This framework, along with these test results, have the potential to be of great value to a network system designer. These results reveal the soft-error failure modes of network systems and their failure rates. This information can be used in evaluating to what degree soft errors are a concern, and what kind of failures can be expected. The framework itself can be adapted to test commercial products and determine the soft-error sensitivity of those designs to provide more

accurate failure rate estimates. This can be used to experiment with different mitigation techniques and to develop a system that is sufficiently reliable.

As networking standards and technology continue to develop to meet the increasing demands of high-bandwidth and low-latency connections, FPGAs will continue to play an important role in network systems. But to ensure that these systems meet reliability requirements when deployed at large scales, soft-error reliability needs to be considered in the design process. The framework presented in this thesis serves as a capable resource for understanding soft-error effects in network systems, increasing the reliability of these systems, and consequently, improving the quality of computer networks as a whole.

8.1 Future Work

The framework and results presented in this thesis serve as a good starting point for researching soft-error effects in network systems, but there is much to still be understood in how to best improve the reliability of network systems. The experiments of this thesis were performed on a relatively simple Ethernet switch design; however, commercial high-performance systems typically offer many more features, which would result in a more complex design. These designs would have more complex failure modes that would not have been encountered in this work. Most notably, this system does not provide any quality of service (QoS) guarantees which would introduce a new class of failure modes. A more featureful network system may more accurately reflect high-end commercial products.

To accompany a more complex system, the traffic generator would also benefit from allowing more fine-grained control of flow transmission rates and in collecting statistics. By collecting packet processing delay times, the traffic generator would be able to provide more helpful statistics. These statistics could assist in measuring QoS compliance, providing per-flow bandwidth information, and packet processing latency statistics. These features would allow the traffic generator to be used for general performance testing rather than strictly providing reliability metrics.

The application of TMR and DWC in this design is limited by design constraints (proprietary IP and clock domain crossing). However, commercial designs are often constrained by physical resources and may only be able to protect limited portions of their design. It would be beneficial to experiment with different levels of mitigation to determine the effectiveness of limited

application and which portions of the designs benefit most from mitigation. Modifying the current design to use an open source MAC would grant greater flexibility in selecting which regions of the design to mitigate.

REFERENCES

[1] C. T. Tsai, R. H. Jan, and K. Wang, "Design and implementation of high availability OSPF router," *Journal of Information Science and Engineering*, vol. 26, no. 6, pp. 2173–2198, 2010. 1

[2] M. Santarini, "FPGAs Take Central Role in Wired Communications," *Xcell Journal*, no. 67, pp. 8–13, 2008. [Online]. Available: https://www.xilinx.com/publications/archives/xcell/Xcell67.pdf 2

[3] A. L. Silburt, A. Evans, I. Perryman, S. Wen, and D. Alexandrescu, "Design for Soft Error Resiliency in Internet Core Routers," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3551–3555, dec 2009. 2, 9

[4] J. W. Lockwood, "Evolvable Internet hardware platforms," *Proceedings - NASA/DoD Conference on Evolvable Hardware, EH*, vol. 2001-Janua, pp. 271–279, 2001. 5, 6

[5] D. Comer, "Network Processors: Programmable Technology for Building Network Systems," 2004. [Online]. Available: https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-30/network-processors.html 6, 9

[6] T. Wolf and J. Turner, "Design issues for high-performance active routers," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 3, pp. 404–409, mar 2001. [Online]. Available: http://ieeexplore.ieee.org/document/917702/ 6

[7] D. F. Bacon, R. Rabbah, and S. Shukla, "FPGA Programming for the Masses," *Queue*, vol. 11, no. February 2013, pp. 1–13, 2013. [Online]. Available: http://doi.acm.org/10.1145/2436696.2443836 6

[8] M. Thompson, "FPGAs accelerate time to market for industrial designs," 2004. [Online]. Available: https://www.eetimes.com/document.asp?doc{_}id=1150608 6, 9

[9] Xilinx Inc., "High Speed Serial." [Online]. Available: https://www.xilinx.com/products/technology/high-speed-serial.html 6, 7

[10] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, "Reconfigurable Network Systems and Software-Defined Networking," *Proceedings of the IEEE*, vol. 103, no. 7, pp. 1102–1124, jul 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7122247/ 7

[11] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as Research Commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, sep 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6866035/ 7

[12] S. Saponara, E. Petri, M. Tonarelli, I. del Corona, and L. Fanucci, "FPGA-based Networking Systems for High Data-rate and Reliable In-vehicle Communications," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, apr 2007, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/4211844/ 7

[13] M. Mansour and A. Kayssi, "FPGA-based Internet Protocol Version 6 router," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*. IEEE Comput. Soc, 2002, pp. 334–339. [Online]. Available: http://ieeexplore.ieee.org/document/727071/ 7

[14] N. Zilberman, Y. Audzevich, G. Kalogeridou, N. M. Bojan, J. Zhang, and A. W. Moore, "NetFPGA - rapid prototyping of high bandwidth devices in open source," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, sep 2015, pp. 1–1. [Online]. Available: http://ieeexplore.ieee.org/document/7293966/ 7, 17

[15] G. Kalogeridou, "NetFPGA 1G Ported Switch 10G," 2015. [Online]. Available: https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-1G-Ported-Switch-10G 8

[16] Cisco Systems Inc., "Field Programmable Device (FPD) Upgrades on Cisco IOS XR - Cisco." [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-xr-software/116859-technote-fpd-00.html 8

[17] P. Morreale and J. Anderson, *Software Defined Networking*. CRC Press, nov 2014. [Online]. Available: https://www.taylorfrancis.com/books/9781482238648 9

[18] Xilinx Inc., "Software Defined Specification Environment for Networking (SDNet)," 2014. [Online]. Available: http://www.xilinx.com/publications/prod{_}mktg/sdnet/backgrounder.pdf 9

[19] Arista Networks Inc., "7130 FPGA-enabled Switches," Tech. Rep. [Online]. Available: https://www.arista.com/assets/data/pdf/Datasheets/7130-FPGA-Switches-Quick-Look.pdf 9

[20] Juniper Networks Inc, "Breakthrough Technology Moves the Financial Services Industry into a New Era," 2015. [Online]. Available: https://www.juniper.net/us/en/insights/switchingbreakthroughforfinance/ 9

[21] S. Lawson, "Facebook sees need for Terabit Ethernet," 2010. [Online]. Available: https://www.computerworld.com/article/2520675/facebook-sees-need-for-terabit-ethernet.html 9

[22] F. Wang and V. D. Agrawal, "Single Event Upset: An Embedded Tutorial," in *21st International Conference on VLSI Design (VLSID 2008)*. IEEE, 2008, pp. 429–434. [Online]. Available: http://ieeexplore.ieee.org/document/4450538/ 10, 11

[23] Microsemi Corporation, "Neutron-Induced Single Event Upset (SEU) FAQ," Tech. Rep., 2011. [Online]. Available: https://www.microsemi.com/document-portal/doc{_}view/130760-neutron-seu-faq 10

[24] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proceedings International*

Conference on Dependable Systems and Networks. IEEE Comput. Soc, 2002, pp. 389–398. [Online]. Available: http://ieeexplore.ieee.org/document/1028924/ 10

[25] "Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices," 2006. [Online]. Available: https://www.jedec.org/sites/default/files/docs/JESD89A.pdf 10, 65, 67

[26] R. C. Baumann, "Soft errors in advanced semiconductor devices-part I: the three radiation sources," *IEEE Transactions on Device and Materials Reliability*, vol. 1, no. 1, pp. 17–22, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=946456 10

[27] Digilent Inc., "NetFPGA-1G-CML Reference Manual," 2018. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/netfpga-1g-cml/reference-manual 18

[28] T. Groléat, M. Arzel, S. Vaton, A. Bourge, Y. Le Balch, H. Bougdal, and M. Aranaz Padron, "Flexible, Extensible, Open-source and Affordable FPGA-based Traffic Generator," New York, NY, USA, pp. 23–30, 2013. [Online]. Available: http://doi.acm.org/10.1145/2465839.2465843 18, 49

[29] R. Al-Dalky, K. Salah, H. Otrok, and M. Al-Qutayri, "Accelerating snort NIDS using NetFPGA-based Bloom filter," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, aug 2014, pp. 869–874. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6906470 18

[30] T. Wellem, Y. Lai, C. Cheng, Y. Liao, L. Chen, and C. Huang, "Implementing a heavy hitter detection on the NetFPGA OpenFlow switch," in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, vol. 2017-June. IEEE, jun 2017, pp. 1–2. [Online]. Available: http://ieeexplore.ieee.org/document/7972136/ 18

[31] T. Wellem, Y. Lai, and W. Chung, "A software defined sketch system for traffic monitoring," in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, may 2015, pp. 197–198. [Online]. Available: http://ieeexplore.ieee.org/document/7110138/ 18

[32] Opsero Electronic Design, "EthernetFMC - Quad Port Gigabit Ethernet FMC." [Online]. Available: http://ethernetfmc.com/ 20

[33] A. Dominguez, P. P. Carballo, and A. Nunez, "Programmable SoC platform for deep packet inspection using enhanced Boyer-Moore algorithm," *12th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip, ReCoSoC 2017 - Proceedings*, 2017. 21

[34] O. Arap, L. R. Brasilino, E. Kissel, A. Shroyer, and M. Swany, "Offloading Collective Operations to Programmable Logic," *IEEE Micro*, vol. 37, no. 5, pp. 52–60, sep 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8065002/ 21

[35] A. Gruwell, P. Zabriskie, and M. Wirthlin, "High-speed FPGA configuration and testing through JTAG," pp. 1–8, 2016. 21

[36] L. Kyle, "Parameterizable Content-Addressable Memory," Xilinx, Inc, Tech. Rep., 2011. [Online]. Available: http://www.xilinx.com/support/documentation/application{_}notes/xapp1151{_}Param{_}CAM.pdf 29

[37] R. Le, "Soft Error Mitigation Using Prioritized Essential Bits," Xilinx, Inc., Tech. Rep., 2012. 38

[38] Xilinx Inc., "Device Reliability Report (UG116)," 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user{_}guides/ug116.pdf 38, 39, 71, 73

[39] L. R. Rockett, "Designing CMOS data cells for space systems," *Microelectronics Journal*, vol. 35, no. 12, pp. 953–967, dec 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026269204001156 40

[40] J. Johnson, W. Howes, M. Wirthlin, D. L. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using Duplication with Compare for On-line Error Detection in FPGA-based Designs," in *2008 IEEE Aerospace Conference*. IEEE, mar 2008, pp. 1–11. [Online]. Available: http://ieeexplore.ieee.org/document/4526470/ 40, 44

[41] J. Johnson and M. Wirthlin, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '10*. New York, New York, USA: ACM Press, 2010, p. 249. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1723112.1723154 41, 42

[42] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA Design Robustness with Partial TMR," in *2006 IEEE International Reliability Physics Symposium Proceedings*. IEEE, mar 2006, pp. 226–232. [Online]. Available: http://ieeexplore.ieee.org/document/4017162/ 41

[43] Y. Li, B. Nelson, and M. Wirthlin, "Synchronization Techniques for Crossing Multiple Clock Domains in FPGA-Based TMR Circuits," *IEEE Transactions on Nuclear Science*, dec 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5658028/ 42

[44] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR Techniques in the Presence of Single Event Upsets," *International Conference of Military and Aerospace Programmable Logic Devices (MAPLD)*, vol. 836, pp. P63–P63, 2003. 43

[45] D. L. McMurtrey, "Using duplication with compare for on-line error detection in FPGA-based designs," Ph.D. dissertation, Brigham Young University, 2006. 44

[46] Cisco Systems Inc., "Cisco IT Best Practices: Cisco High Availability LAN," Tech. Rep., 2008. [Online]. Available: https://www.cisco.com/c/dam/en{_}us/about/ciscoitatwork/downloads/ciscoitatwork/pdf/How{_}Cisco{_}IT{_}Achieved{_}Highly{_}Available{_}Local{_}Area{_}Network.pdf 44

[47] A. G. Stoddard, "Configuration Scrubbing Architectures for HighReliability FPGA Systems," Ph.D. dissertation, Brigham Young University, 2015. 47

[48] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *2009 International Conference on Field Programmable Logic and Applications*. IEEE, aug 2009, pp. 99–104. [Online]. Available: http://ieeexplore.ieee.org/document/5272543/ 47

[49] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. A. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2259–2266, aug 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4636940/ 47

[50] Ixia, "Network Performance - IxNetwork." [Online]. Available: https://www.ixiacom.com/products/ixnetwork 49

[51] Xena Networks, "Valkyrie - Stateless Ethernet Traffic Generation and Analysis Platform." [Online]. Available: https://xenanetworks.com/valkyrie/ 49

[52] J. Sommers, H. Kim, and P. Barford, "Harpoon," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, p. 392, jun 2004. [Online]. Available: https://www.jstor.org/stable/375417?origin=crossrefhttp://portal.acm.org/citation.cfm?doid=1012888.1005733 49

[53] M. Ghobadi, M. Labrecque, G. Salmon, K. Aasaraai, S. Hassas Yeganeh, Y. Ganjali, and J. G. Steffan, "Caliper," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, p. 445, aug 2010. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1851275.1851255 49

[54] J. Sommers and P. Barford, "Self-configuring network traffic generation," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*. New York, New York, USA: ACM Press, 2004, p. 68. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1028788.1028798 49

[55] Xilinx Inc., "AXI 1G/2.5G Ethernet Subsystem (PG138)," Tech. Rep. [Online]. Available: https://www.xilinx.com/support/documentation/ip{_}documentation/axi{_}ethernet/v7{_}0/pg138-axi-ethernet.pdf 56

[56] A. Ronacher, "Flask," 2019. [Online]. Available: http://flask.pocoo.org/ 60

[57] UWSGI, "The uWSGI project," 2016. [Online]. Available: https://uwsgi-docs.readthedocs.io/en/latest/ 60

[58] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of Virtex FPGA TMR design methodology," in *RADECS 2001. 2001 6th European Conference on Radiation and Its Effects on Components and Systems (Cat. No.01TH8605)*, vol. 00, no. C. IEEE, 2011, pp. 275–282. [Online]. Available: http://ieeexplore.ieee.org/document/1159293/ 63

[59] H. M. Quinn, D. A. Black, W. H. Robinson, and S. P. Buchner, "Fault Simulation and Emulation Tools to Augment Radiation-Hardness Assurance Testing," *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 2119–2142, jun 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6519339/ 65, 71

[60] S. F. Nowicki, S. A. Wender, and M. Mocko, "The Los Alamos Neutron Science Center Spallation Neutron Sources," *Physics Procedia*, vol. 90, no. November 2016, pp. 374–380, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.phpro.2017.09.035 67