FAULT INJECTION SYSTEM FOR FPGA-BASED SPACE COMPUTERS

by

Hezekiah Ajax Austin

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2023

ACKNOWLEDGEMENTS

I would like to acknowledge Dr. Brock J. LaMeres, my advisor, for purposing and funding my research into SEM fault injection on the RadPC computer architecture.

I would like to acknowledge Stottler-Henke Inc for funding the initial project, RadPC-AI, which formed the basis and major motivation for this further research. Their work and funding greatly assisted in the testing of RadPC on the International Space Station. The results of that experiment were the primary motivation for this research exploring better methods of fault injection.

I would like to acknowledge the Graduate Students, Chris Major, Justin Williams, and Colter Barney. All of whose work was stressed tested and/or broken via error injection in the creation of this Masters thesis.

I would also like acknowledge the Undergraduate Students, Zach Becker and Kenzie Smith, whose work on the proceeding RadPC-AI project and this project greatly helped in the completion of this research.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

NOMENCLATURE

Nomenclature

Bitstream Memory	Flash memory chip holding the partial and full bitstreams for the FPGA.
Configuration Memory	FPGA's internal configuration memory.
Data Memory	Flash memory chip holding the telemetry data packets.
Firmware	Implemented logic computer designs.
Hardware	Physical hardware for payloads and experiments, PCBs, LATTE, etc.
Lunar Board	RadPC PCB implementing RadPC-Lunar executing a counter program.
RadPC	Quad modular redundant computer architecture implemented with commercial-off-the-shelf components on an FPGA.
RadPC-AI	Partner Project with Stottler-Henke Inc for two payload experiment on the International Space Station.
RadPC-Lunar	Logic design developed for NASA project of a payload experiment on a lunar lander
RadPC-Memory	Project researching Soft Error Mitigation Controller fault injections into RadPC-Lunar configuration memory.
RadPC PCB	RadPC Rev. 4 Printed Circuit Board.
Software	Programs written in C, Python, and Bash for the payloads and ground station.
Stottler Board	RadPC PCB implementing RadPC-Lunar executing Stottler-Henke's computer algorithm.

Acronyms

AI	Artificial Intelligence
COTS	Commercial Off The Shelf
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
ECC	Error Correction Codes
FPGA	Field Programmable Gate Array
GEO	Geosynchronous Orbit
HDL	Hardware Description Language
HPSC	High Performance Space Computer
HPTC	High Performance Terrestrial Computer
IC	Integrated Circuit

NOMENCLATURE – CONTINUED

ISS	International Space Station
LATTE	Lander Analog Telemetry Test Emulator
LEO	Low Earth Orbit
LFA	Linear Frame Address
LSB	Least Significant Bit
LSITP	Lunar Surface Instruments and Technology Payloads
MCU	Micro Controller Unit
MIPS	Million Instructions per Second
MFLOPS	Million Floating Point Operations
ML	Machine Learning
MSP430	MSP 430 FR6989 Microcontroller
MSU	Montana State University
QMR	Quad Modular Redundancy
PCB	Printed Circuit Board
PFA	Physical Frame Address
PR	Partial Reconfiguration
Rad-Hard	Radiation-Hardened
RadPC PCB	RadPC Rev.4 PCB
RHBD	Radiation-Hardened by Design
RHBP	Radiation-Hardened by Process
RISC-V	Reduced Instruction Set Computer, Fifth Generation
SBC	Single Board Computer
SEE	Single Event Effect
SEFI	Single Event Functional Interrupt
SEM	Soft Error Mitigation
SEM Controller	Soft Error Mitigation Controller
SET	Single Event Transient
SEU	Single Event Upset
SOI	Silicon On Insulator
SPI	Serial Peripheral Interface
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

ABSTRACT

Abstract: Simulation of radiation effects in aerospace computers is a key testing and verification component to space operations. Contemporary computer architectures utilizing Field Programmable Gate Arrays (FPGA) requires particular focus in testing the configuration memory of the device for faults that cannot be recovered using traditional strategies. Faults in the configuration memory propagate to the hardware settings of the FPGA, changing the implemented logic circuit functionality. The effects of faults in the configuration memory are unpredictable, limiting the effectiveness of computer simulation and analysis. Therefore, designers of FPGA-based aerospace computers prefer to physically induce faults in the configuration memory to measure their impact. This allows the results of configuration memory fault injection used to classify faults occurring during space operation. The process is difficult to implement as the FPGA configuration memory is large, often undocumented, and the injection process is tedious when done manually. This paper presents the results of the deployment of two FPGA-based aerospace computers payloads to the International Space Station and the subsequently developed process for configuration memory fault injection. The injections are designed to simulate errors caused by radiation strikes to the computer hardware. These injections were performed on duplicate hardware to the RadPC payloads that operated on the ISS and was bombardment by real radiation. This provided the ability to see if the ground-based injection was correlated to real flight data. The developed process is able to inject single bit faults, which represents the majority of faults observed in configuration memory for space applications, and continuous injection, which stress tests the aerospace computer's recovery capability. Depending on the effects of the injected fault, the error is marked as either repairable, nonrepairable and propagating, or nonrepairable and nonpropagating. The result of this testing illustrates the key components in the implemented computer architecture which are vulnerable to faults in the configuration memory. Vulnerable components include the softcores, voter components, and the input logic. The process allows these key components to be isolated for further testing and the comparison of payload results to configuration memory testing on the ground.

1

INTRODUCTION

Advancements in computer capability has enabled new areas of space exploration and research. The current direction of development in aerospace computers is increased computer automation across the majority of spacecraft. Satellites, spacecraft, lunar landers, rovers, probes, and telescopes are using more sophisticated computers to accomplish their missions quickly and efficiently. Modern terrestrial computers are able to meet these developing and future computing needs, but are not capable of operation in space due to the harsh radiation environment. Terrestrial computers are protected from this issue by the Earth's atmosphere and magnetosphere, which greatly attenuate radiation before it reaches the Earth's surface. Current solutions to radiation induced faults in computers require the computers to be redesigned specifically for radiation resistance and hardware redundancy. Theses solutions to radiation effects are expensive because of the custom hardware and generally limit computer performance in favor of redundancy and survivability. Aerospace computers tend to lag behind terrestrial computers in development and overall capabilities because of these design limitations. Terrestrial computers have advanced capabilities in data processing, automation, machine learning, and artificial intelligence. Space missions would greatly benefit from these capabilities, especially if these computers were functional in space without requiring expensive radiation *hardening*. This line of research uses terrestrial computers built with commercially available parts in space with novel recovery methods for the radiation induced faults.

Radiation induced faults in aerospace computers fall into two general categories, total ionizing dose (TID) and single event effects (SEEs). TID gradually erodes the physical structure of the circuit as lower-energy particles are trapped inside the integrated circuit's

insulating features. The modern integrated circuit has become inherently resistance to TID due to much smaller feature sizes (<65 nm). These smaller sizes reduce the likelihood of a feature trapping a high-energy particle and being damaged. However, the smaller feature sizes are a compromise in computer resistance to radiation damage. While TID and the associated damage on an integrated circuit is greatly reduced, the smaller features are more susceptible to SEEs. With the smaller feature size, high-energy particles are likely to cause functional interrupts in the integrated circuit. Single event transients (SETs) occur when a logic line voltage is changed by a high energy particle or heavy ion strike to the line. Single event upsets (SEU) are logic-level transitions caused by high-energy particle or heavy ions strikes to the complementary metal oxide semiconductor (CMOS) device. In general, modern semiconductors are inherently vulnerable to SEEs due to the smaller feature sizes. This drives the need for a modern aerospace computer focused on SEE detection and recovery methods.

Field Programmable Gate Arrays (FPGAs) have emerged as a technology that can implement novel fault recovery procedures while maintaining high performance with commercial off the shelf (COTS) components. The main techniques used for FPGA-based, aerospace computers are redundancy, error correction codes (ECCs), Cyclic Redundancy Check (CRC) codes in memory, and configuration memory scrubbing. A FPGA based computer architecture has been developed to meet the demand for radiation tolerant, high performance, relatively inexpensive computers in space. This computer architecture, named RadPC, utilizes COTS components on a small form factor printed circuit board (PCB). RadPC implements Quad Modular Redundancy (QMR) with four softcores, CRCs and ECCs for memory checking, data memory scrubber, and a configuration memory scrubber. Two payloads were deployed to the International Space Station (ISS) as part of overall testing for this RadPC FPGA-based computer. Payload 1 following its return from the ISS is shown in

Figure 1.1: Payload 1 after return from ISS

A point of failure for FPGA-based computers is the configuration memory. Within an FPGA, the configuration memory is physically distributed across the device and actively controls the actual implementation of the logic design. Changes in the configuration memory will propagate to the logic design and induce faults in the outputs. Verification of the

reliability of FPGA-based computer is difficult as there are so many distinct locations where faults can occur. FPGAs have millions of locations in configuration memory which are capable of effecting the implemented logic circuit. For space operations, the lack of information on the computer response to faults in the configuration memory makes verification of faults causes during space operations difficult. Systematic testing of this configuration memory represents an important method of verification between ground and space results. Radiation chamber testing is generally too expensive for FPGA design verification and does not ensure that every location in the configuration memory will be faulted by a radiation strike. Furthermore, FPGA designs are created using Hardware Description Languages (HDLs) and are easily modifiable. Modern synthesis tools automatically convert the HDL into logic circuitry and map it into the resources of the FPGA. The designer has little control over how the logic is ultimately assigned, which creates a challenge in creating a consistent verification strategy. Slight changes in the HDL can cause an FPGA design to be implemented with a completely different placement on the board. The lack of a consistent verification strategy causes a major bottleneck in development due to the time and resources required for repeated, expensive, noncomprehensive testing. Therefore, FPGA-based aerospace computer development needs a comprehensive fault injection system capable of injecting faults into every bit of an FPGA's configuration memory and monitoring the resulting failures.

This paper proposes and details a comprehensive fault injection system to stress test FPGA-based computer's configuration memory scrubber and monitor the resulting faults in the computer architecture. The injections were performed on the RadPC, a FPGA-based computer developed for aerospace application. The purpose of this testing is to confirm the scrubber's detection/recovery capabilities for the FPGA's distributed configuration memory and provide a comparison point for the results from RadPC payloads deployed to space. This testing serves to identify critical injection locations that cause propagating or nonrepairable

faults in RadPC's implemented logic. These critical injection locations allow for precise stress testing of specific components in the logic circuit design. Furthermore, the configuration memory injection testing is used to analyze the data from the two ISS payloads.

BACKGROUND

Radiation Effects on Computer Systems in Space

Space is an inherently hostile environment for computing systems. On the surface of the Earth, computers are protected by the Earth's atmosphere and magnetic field from radiation. Personal computers, smartphones, servers, and micro-controllers are capable of functioning reliably because of these layers of protection. This allows computer systems to operate predictably without errors caused by energized particle strikes. In space, Earth's layers of protection are generally weakened and computers are bombarded with both low-energy and high-energy electrons, protons, and heavy ions resulting in material damage and faulted logic values. [4] With the increase in scientific research and commercial ventures into space, the need for computers capable of operating reliably in a radiation environment has risen. Most space operations occur inside the Van Allen radiation belts shown in Figure 2.1, which illustrates the most common radiation environment and the relative density of radiation. In addition, the need for reliably is also driven by the greater complexity of semiconductor based computers and the increased vulnerability to certain types of radiation strikes during operation.

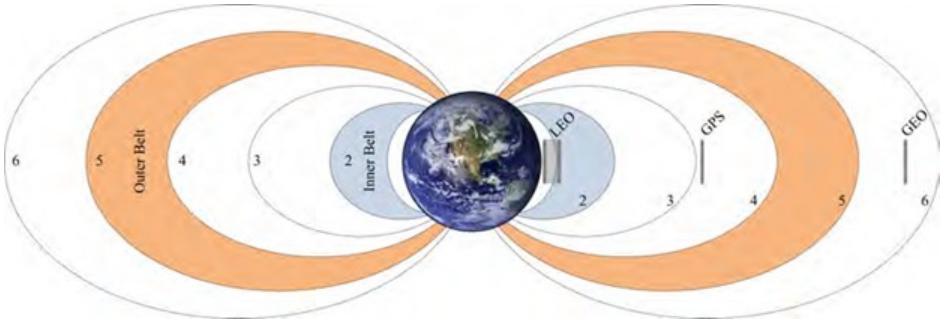


Figure 2.1: Van Allen Radiation Belts [21]

Within these regions, semiconductors are more likely to suffer radiation induced faults.

The effects of radiation exposure for modern semiconductors fall into two general categories, TID and SEE. TIDs are a gradual failure due to cumulative effects of lower energy particles strikes and SEEs are a temporary failure due to immediate effects of high energy particle strikes. [7] Figure 2.2 shows the effects of TIDs and SEEs in a complementary metal-oxide-semiconductor (CMOS) device. A TID is caused by lower energy particles depositing charge inside the semiconductor's installation layers. This deposited charge physically degrades the semiconductor's channel for current, leading to an uncontrolled current flow through the device. With longer exposure to radiation strikes, the semiconductor gradually becomes more damaged as the number of deposited charges increases over time and area. Failure occurs when the semiconductor becomes permanently biased, locking it indefinitely in an on or off state and allowing continuous uncontrolled current flow. This failure frequently leads to cascading failures as more semiconductors both from radiation strikes and damage of uncontrolled current flow.

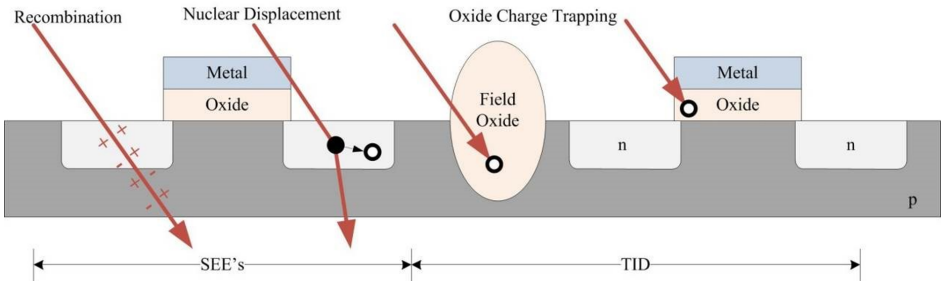


Figure 2.2: Effects of TIDs and SEEs in CMOS Cross Section [12]

Modern integrated circuit (IC) design size (<65 nm) makes TID more statistically unlikely due to reduced feature sizes lessening the likelihood of low energy particles being trapped in the insulating regions of the devices. [3] Smaller feature sizes are a compromise in computer resilience to radiation. Modern semiconductors are inherently more vulnerable to SEEs while having an increased resistance to TIDs. The smaller feature sizes cause particle strikes to be more likely to be captured, particularly in combinational logic circuits. [5] An

SEE occurs when a high energy particle or heavy ion strikes the device, causes a logic-level transition, and the transition is captured. Three subcategories of SEEs exist depending on the effect and location of the transition. Transitions in logic lines are single event transients (SETs). The SET occurs when a high energy particle changes the voltage in a line and therefore the logic value. A single event upset (SEU) occurs when a SET is stored in a memory device such as a D-flip-flop or memory cell. Conventional recovery strategies, such as error detecting codes, are generally successful in recovering from SEUs and SETs. The third subcategory is signal functional interrupt (SEFI) which occurs when the SET or SEU cannot be recovered from and the system fails from the fault. An SEFI generally requires a complete reset of the device in order to clear the fault.

Radiation Mitigation Methods

Mitigation techniques for radiation induced faults have been developed for aerospace computer systems. Different particles types require different mitigation techniques depending on the particles energy level and interactions with the materials of the spacecraft. Mitigation techniques generally focus on resistance to either TID caused faults or SEEs.

Shielding

Shielding is the addition of a material layer to physically block radiation from striking the computer system. This layer functions as barrier for low energy particles, alpha and beta particles, which will strike and scatter off the material layer. Shielding is primary effectively against TID, although it does have limited effectiveness vs high energy particles. However, shielding suffers from diminishing returns in thickness vs effectiveness as shown in Figure 2.3.

Shielding has relatively high mass and volume requirements since it uses a physical barrier on the spacecraft to block radiation. A compromise solution is to localize the shielding

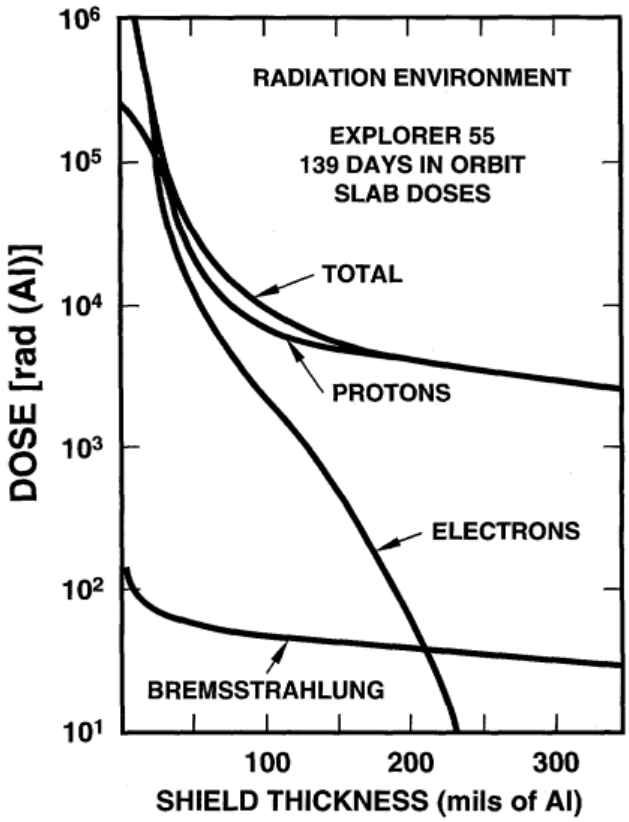


Figure 2.3: Total Radiation Dose vs Aluminium Shielding [17]

to only around the sensitive computer parts. [20] However, high energy particles, gamma particles, neutrons, and heavy ions, remain a problem as they are capable of passing through the shielding. This allows these high energy particles to be trapped by an IC feature inside the computer. Moreover, shielding can cause scattering of the incoming particles which creates secondary particles as the incoming particles pass through the shielding. [18] The low energy of the particles and the secondary scattering reduces the impact on the computer and spreads out the damage over a larger area. The shielding can be optimized to minimize this scattering. However, this places size and shape restraints on the shield. [6] Despite these limitations, shielding is one of the most effective methods of protecting computers outside of the Earth's atmosphere and magnetic fields. The main limitations of shielding are its mass

requirements and inability to block high energy particles.

Radiation Hardening by Process

Another mitigation technique is radiation hardening by process (RHBP) which is also effective against TID caused faults. For RHBP devices, the radiation resistance is build directly into the IC during fabrication of the the device.

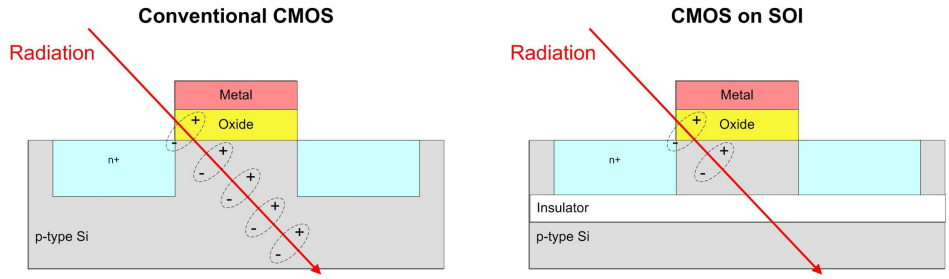


Figure 2.4: Conventional CMOS vs CMOS with RHBP [12]

An example of RHBP is silicon on insulator (SOI) where an insulating layer is included user the CMOS junction during fabrication. [11] The insulating layer reduces charge trapping and decreases the thickness of the gate oxide which isolates the CMOS. A diagram of conventional CMOS vs an SOI CMOS is shown in Figure 2.4. SOI is an effective method of hardening against TID caused faults. However, SOI transistors have a parasitic bipolar effect that amplifies the collected charge and decreases the SEU/SET immunity. [16] RHBP, at least SOI, should be paired with another mitigation technique for SEEs.

Radiation Hardening by Design

Radiation hardening by design (RHBD) builds the radiation resistance directly into the IC during fabrication. Unlike RHBP, RHBD allows radiation induced charges to flow through conducting paths instead of being trapped in the insulating layer. [10] This mitigation technique is very effective against TID caused faults as radiation induced charges are conducted away from the IC.

Radiation Hardening by Architecture

The final mitigation technique is radiation hardening by architecture (RHBA). This mitigation technique focuses on resistance to SEEs. The technique accepts there will be radiation induced faults and builds redundancies into the IC to resist and recovery from those faults.

Triple Modular Redundancy (TMR) is a common RHBA technique. For TMR, three copies of the circuit are implemented and a voter circuit is placed on the output. When an SEE caused fault occurs in one copy, the other two copies report the correct output and the voter uses the major output as the output for the circuit. This has the advantage of ensuring the circuit is resistance to SEEs. In TMR, the voter is a single point of failure for the entire system. [14]. Another limitation is that TMR requires a large amount of resources compared to the original circuit since three copies plus a voter circuit are implemented.

12

MISSION OVERVIEW

The ISS Project delivered two payloads to the ISS to test an FPGA-based computer system in a space radiation environment. After payload deployment to the ISS, ground testing was preformed on duplicate hardware. This provided the ability to check if the ground-based injections were correlated to real flight data. The ISS Project was made possible through the collaborations and contributions of three partners, Montana State University (MSU), Stottler Henke Inc. (Stottler-Henke), and NanoRacks Inc (NanoRacks).

Mission Partners and Partner Roles

Contributions of Montana State University

For the last fifteen years, RadPC has been researched and developed by MSU as an FPGA-based computer architecture designed to mitigate radiation induced faults in aerospace computers. This technology has been developed and tested through a number of high altitude balloon missions, sounding rocket missions, CubeSat missions, and ISS missions. For the ISS Project and Memory Project, the RadPC Rev. 4 PCB was used in ISS Payload 1, ISS Payload 2, and the Memory experiment. This PCB functioned as the basis for the ISS payloads. Each payload had two boards as part of the experiment. One board provided the data and the second board analysed the data using an algorithm developed by Stottler-Henke. RadPC Rev. 4 PCB was also used for the SEM memory injection testing discussed in later sections. The engineering model for the ISS Payloads is shown in Figure 3.1.

Contributions of Stottler-Henke

Stottler-Henke partnered with MSU for the ISS Project which developed, designed, and programmed two internal payloads for the ISS. Stottler-Henke focused on the development

Figure 3.1: Engineering Model for ISS Payloads

of a computer program to run on the RadPC computer developed by MSU. This functioned as both a capability test and a usability test. RadPC previously used a counter program for testing and flight payloads and was only programmed by the MSU team. The program was kept simple for development and verification purposes. The programming procedure was very complicated, requiring resynthesizing the VHDL logic design, and generating a new bitstream for any new C program implementation. Stottler-Henke developed a computer program which analysed data from a second RadPC board. This program tested RadPC's capability to run complex computer programs while running QMR and scrubber recovery systems. It also tested RadPC's ease of use by having an external developer write the code

for RadPC, without having previously worked on RadPC's VHDL development.

Contributions of NanoRacks

NanoRacks handled the payload interface with the ISS, payload delivery to NASA, payload testing to NASA safety requirements, and liaison with NASA for flight operations. ISS Payload 1 and Payload 2 were deployed to the NanoRacks ExPRESS RACK on the ISS. This rack functioned as the payload interface between the ISS and the two experiment payloads. The payloads were provided power and communication from the NanoRacks ExPESS RACK. To properly interface with the RACK, the two payloads were designed as 1U NanoLabs to interface with this laboratory experiment rack as shown in Figure 3.2. The 1U NanoLabs is a 4" by 4" by 4" size standard used for payloads built to interface with the NanoRacks' ExPRESS RACK.

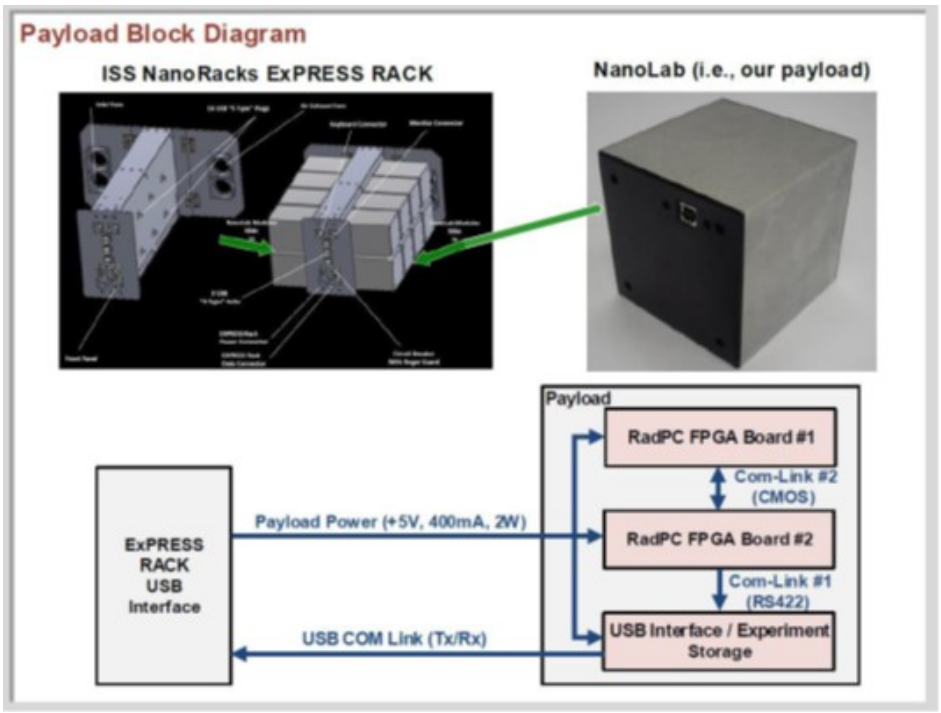


Figure 3.2: NanoRacks ExPRESS Rack and Payload Diagram

After payload delivery from MSU to NanoRacks, NanoRacks performed operational

testing on the payloads and verified the payloads were functional with NASA safety requirements before being delivered for launch. During operations with the payloads in place on the ISS, NanoRacks controlled flight operations such as data transfer from the ISS to ground stations. NanoRacks also handled the return of Payload 1 to MSU on mission completion. Payload 2 is still in operation on the ISS as of March 21st, 2023 with NanoRacks providing flight operations support.

Mission Payloads

The ISS mission consisted of two payloads designed to test the RadPC computer architecture in space radiation conditions. These payloads were developed in partnership with Stottler Henke Associates Inc (Stottler-Henke). For the experiment, Stottler-Henke developed the algorithm and MSU developed and provided the RadPC computer architecture for implementation. NanoRacks also provided the liaison with NASA for the payloads launch, deployment to the ISS, and return to Earth. Payload 1 and Payload 2 were deployed to the ISS and remained aboard for 5 months and 13 months respectively at time of writing. The installation of Payload 2 by US Astronaut Kayla Barron on February 22, 2023 is shown in Figure 3.3.

This chapter discusses the design, operation, experimental data, and analysis of these two payloads. It also provides the motivation and research basis of the configuration memory testing process discussed in following chapters.

Mission Objectives

The objectives of the ISS Project were threefold. The first objective was testing the latest iteration of the RadPC computer architecture in a natural space radiation environment. This objective was accomplished with the deployment of first and second payloads on the ISS.

Figure 3.3: Payload 2 Installation on ISS [13]

Analysis of the data and results is included in the following sections. The second objective was coding and executing an algorithm on the RadPC computer that utilized both inputs and outputs. Partnering with a external company for the algorithm development allowed the RadPC computer to be tested for ease of use. This objective was accomplished by a partnership with Stottler-Henke for the development of a algorithm and testing the ease of programming of the RadPC computer. Stottler-Henke developed the algorithm run on the RadPC Rev. 4 printed circuit board (PCB), shown in Figure 3.4, for both payloads and the MSU team used the level of support require for algorithm development and Stottler-Henke's feedback to evaluate RadPC's ease of programming. The third and final objective of the ISS Project was the development of a reusable design to interface with NanoRacks' ISS internal payload system. This objective was accomplished with the design of a PCB which provides power regulation and communication with the NanoRacks' internal payload system on the ISS. The PCB for this interface will allow future projects to focus more on experiment development instead of interface development.

ISS Experiment Design

The ISS Payloads were build around RadPC, developed by MSU. RadPC consists of a PCB with a microcontroller, an FPGA, two serial flash memory chips, and power regulation circuits for 1.0 Volt, 1.8 Volt, 2.5 Volt, 3.3 Volt, and 5.0 Volt rails.

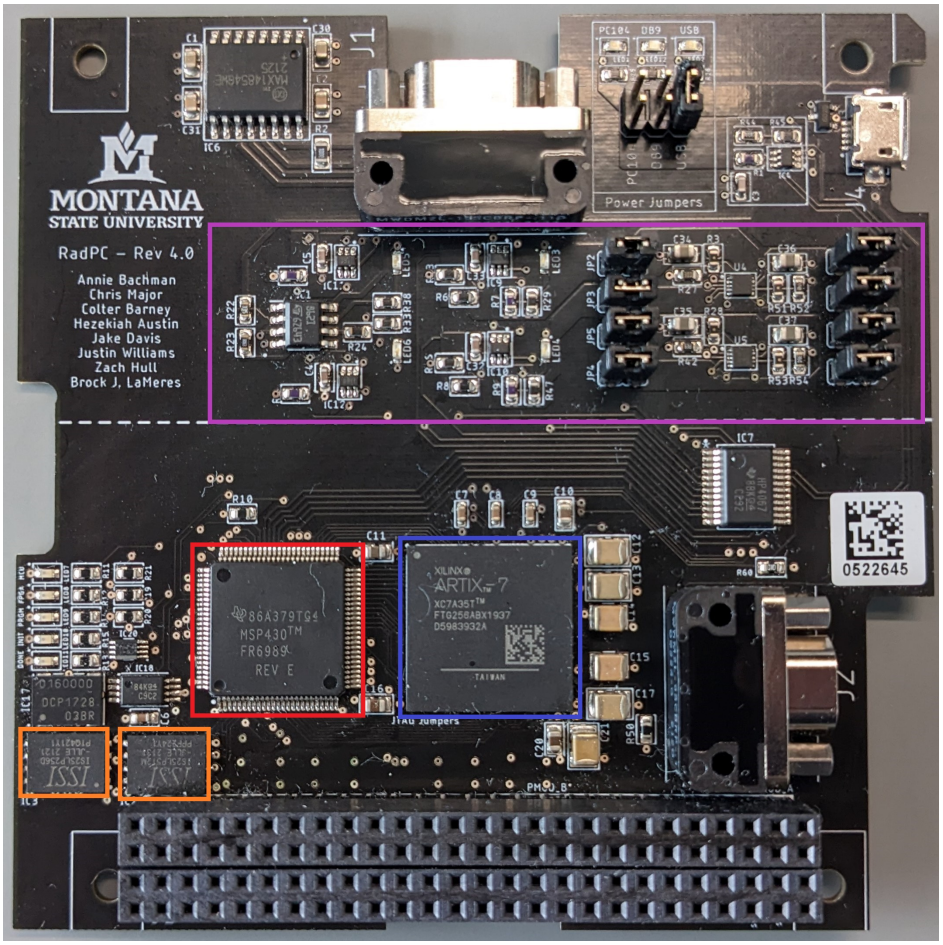


Figure 3.4: RadPC Rev. 4 PCB

For clarity, the following convention will be used throughout this paper. RadPC refers to the system as a whole. RadPC PCB refers to physical hardware that RadPC is implemented on. RadPC-Lunar and RadPC-Stottler refer to the firmware on the FPGA. RadPC-Stottler was a slightly modified version of RadPC-Lunar. The only modification was to allow RadPC-

Stottler to be able to receive data packets. Microcontroller refers to the microcontroller on the RadPC PCB. External memory refers to memory chips on the PCB which are external to both the microcontroller and FPGA. Since parts of RadPC PCB are specifically referenced, a picture of the RadPC Rev. 4 PCB is included in Figure 3.4. The power regulation is boxed in purple. The MSP430 microcontroller is boxed in red. The bitstreams memory chip and data packet memory chip are boxed in orange, and the Xilinx Artix-7 FPGA is boxed in blue.

PCB Stack

The hardware for ISS Project was a PCB stack consisting of three boards, Interface Board, Lunar Board, and Stottler Board, shown in Figure 3.5. Within this PCB stack, each board focused on one or more of the three major objectives for the ISS Project. The Lunar Board focused on implementation and testing of the latest firmware of RadPC computer architecture in a natural space radiation environment. This firmware consisted of the latest VHSIC Hardware Description Language (VHDL) design for RadPC as of September 2021, named RadPC-Lunar. The Stottler Board focused on the implementation of the Stottler-Henke developed algorithm for RadPC. This software algorithm was a C program developed to run on a modified version of the RadPC-Lunar firmware, named RadPC-AI. Onboard the ISS, RadPC-AI was used to test the RadPC computer architecture's capability to function in a radiation environment while execute a complex algorithm to be tested. Finally, the Interface Board focused on power regulation and communication with the NanoRacks' internal payload system on the ISS. Communication was handled by a Raspberry Pi Zero mounted on the Interface Board. Power regulation was handled by a circuit using three buck-boost switching regulators to provide 2.5 volts, 3.3 volts, and 5.0 volts to the Raspberry Pi and RadPC PCBs.

The innermost PCB of the stack is Lunar Board labeled as RadPC_std (Lunar) in

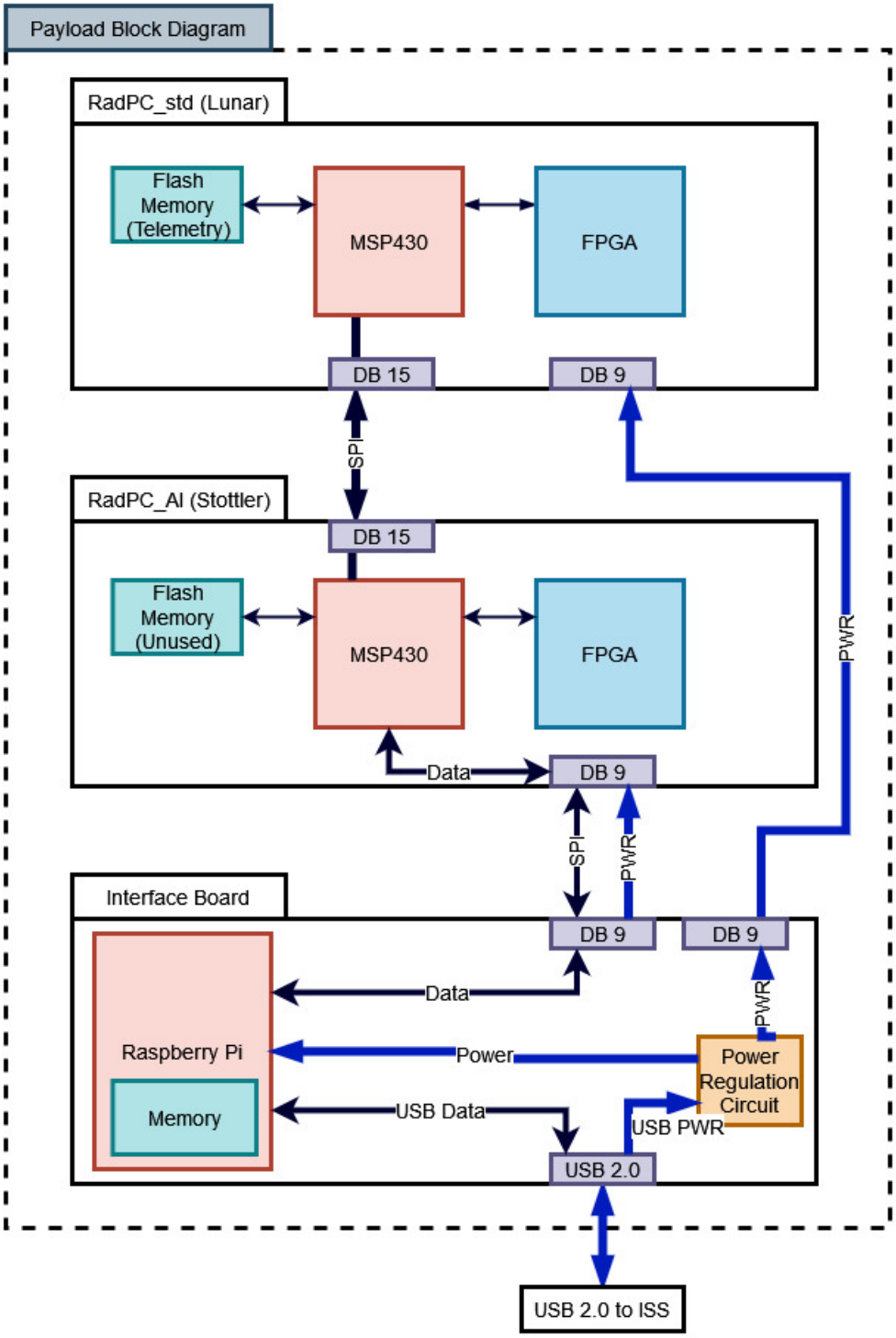


Figure 3.5: ISS Payload PCB Stack Diagram

Figure 3.5. The hardware for this board is the RadPC Rev. 4 PCB. It utilizes the RadPC-Lunar computer architecture written in VHDL for its firmware. The software for this board is a counter program written in the C Programming Language. RadPC-Lunar was used in the ISS Project as both a method of developing VHDL for RadPC computer architecture and providing real world testing of both the RadPC Rev.4 PCB and RadPC-Lunar computer architecture. During operation, the Lunar Board runs the counter program and builds data packets which are transmitted to the Stottler Board for analysis. The primary purpose of the Lunar Board was the objective of testing the latest iteration of the RadPC computer architecture in a space radiation environment.

The center PCB of the stack is the Stottler Board. This board uses the same RadPC Rev. 4 PCB and modified version of RadPC-Lunar firmware and a C program developed by Stottler-Henke for its software. The only difference between RadPC-Stottler and RadPC-Lunar is that RadPC-Stottler's was changed to allow the telemetry data packets to be received from the Lunar Board, the enforcement of software checkpoints for synchronization, and the limitation of no serial communications. This allowed the algorithm to run on the RadPC-Lunar with the full Quad Modular Redundant (QMR) system and tile software checkpoints while receiving data from the Lunar Board. During operation on the ISS, the Stottler Board used its algorithm to evaluate the Lunar Board's data packets and appended the result to the data packet before passing data on to the Interface Board. The PCB stack can be see in Figure 3.6. Top PCB is the Stottler Board. The Lunar Board is the center PCB. The Interface Board is the purple PCB visible on the bottom of the PCB Stack.

The Interface Board controls power regulation and data storage for the payload as a whole. It functions as the interface between the experiment portion of the Payloads, the Lunar and Stottler Boards, and the NanoRacks ISS internal payload system. Interfacing with NanoRacks was accomplished with a power regulation circuit and a Raspberry Pi Zero mounted on a PCB. The power regulation circuit provided protection against voltage or

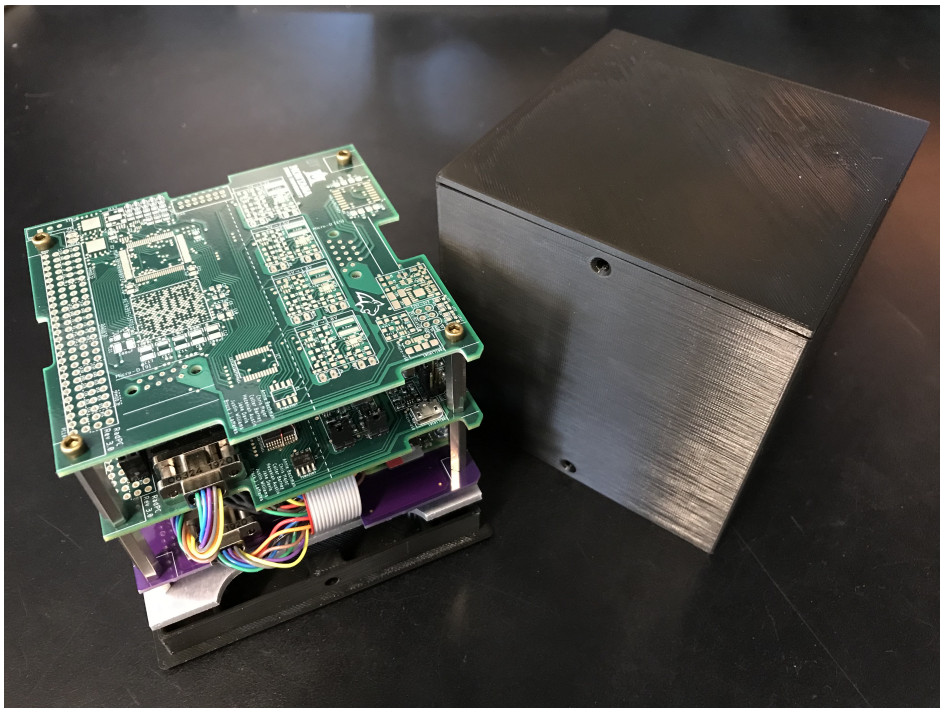


Figure 3.6: ISS Engineering Model with 3D Printed Case Removed

current spikes and regulated the incoming power supplied by NanoRacks' internal payload system to a clean 5 volts for the RadPC boards and the Raspberry Pi Zero. This Raspberry Pi Zero functioned as data storage and communication node between the experiment and NanoRacks' system. For the Stottler Board, the Raspberry Pi Zero functioned as a serial port using the Serial Peripheral Interface (SPI) protocol. To the ISS NanoRacks' system, the Raspberry Pi Zero functioned as USB flash drive which would update itself with new data files on a daily basis.

1U Case Shielding and ISS Radiation Environment

The PCB stack was encased in a 1U NanoLab's aluminum case with anodized sides and 3D printed endcaps. This provided the mechanical support for the PCB stacks and the bottom endcap included power and data plugins for the payload. The aluminum case increased the amount of shielding on the payload. However, SEEs are caused by high energy

particles which are capable of passing through the space station, aluminum case, and the IC. As shown in Figure 3.7, the radiation which primarily causes SEEs, Gamma Rays, X-Rays, and Neutron Rays are all capable of penetrating the aluminum. Since aluminum is the main material used for the ISS' hull, the NanoRacks' internal payload case, and the 1U NanoLab payload case, the shielding will only reduce the radiation by a small amount.

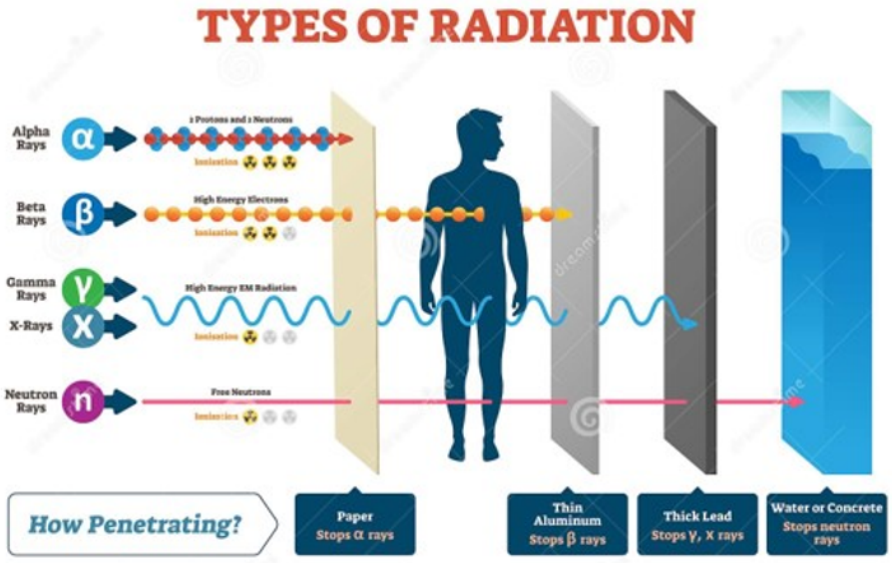


Figure 3.7: Radiation Types

The radiation environments for the ISS was modeled using the Cosmic Ray Effects on Micro-Electronics code (CREME96), the dimensions of the Artix-7 XC7A35T chip, and the Vivado software suite's essential bit count for RadPC. [15] Payload operation on the ISS occurred during a Quiet Solar cycle. While deployment of the experiment to the ISS during this Quiet Solar cycle is not ideal, essential bit flips would still average 8.7 per device per day.

Not all essential bit flips will trigger a softcore tile fault. However, RadPC is estimated to have approximately 3-4 softcore tile faults occur per device per day during payload operation.

ISS Radiation Environment			
Essential bits analysis (2,793,730)		Total bits analysis (14,663,584)	
Stormy Solar	Device/day	Stormy Solar	Device/day
Average	353.910	Average	1,857.690
Peak	669.849	Peak	3,516.510
Quiet Solar	Device/day	Quiet Solar	Device/day
Average	8.714	Average	45.741
Peak	24.809	Peak	130.233
Worst Case Scenarios	Device/day	Worst Case Scenarios	Device/day
Worst week	2,981.997	Worst week	15,654.750
Worst day	16,313.250	Worst day	85,640.400
Worst 5 min	62,247.000	Worst 5 min	326,781.000

Table 3.1: CREME96 modeling of ISS-orbit radiation environments 30% microprocessor derating [15]

Design of RadPC

The RadPC computer architecture was developed by Montana State University for aerospace applications. It consists of an FPGA with a QMR computer architecture that implements a juggling architecture, memory scrubbers to recover from SEEs, and a microcontroller to control data transmission, resets, FPGA reconfiguration, etc. The firmware implemented on the FPGA for the ISS payloads was the latest iteration of the RadPC Computer Architecture as of September 2021. The development objective of RadPC is providing radiation tolerance using off the shelf components via redundancy, scrubbing, and a recovery system. RadPC's computer architecture utilizes redundant, synchronized

cores with an output voter, data memory scrubber, and configuration memory monitor. This computer architecture uses a QMR computer system implemented on an FPGA. The inherent resistance to TID in modern FPGAs from circuit design size allows RadPC to focus mainly on detection and recovery from SEEs. The architecture of RadPC combines the strategies of a juggling architecture with an expansion of TMR and the implementation of memory scrubbing on an FPGA. [15] With TMR, the individual system is triplicated and the output is voted on with the majority vote passing through as the overall system output. However, TMR approach is not sufficient to effectively mitigate the effects of radiation on an FPGA's implemented logic circuit or configuration memory. [19] RadPC counters the vulnerability of TMR and expands on it by adding a fourth tile for QMR. Figure 3.8 shows the block diagram of RadPC at the board level.

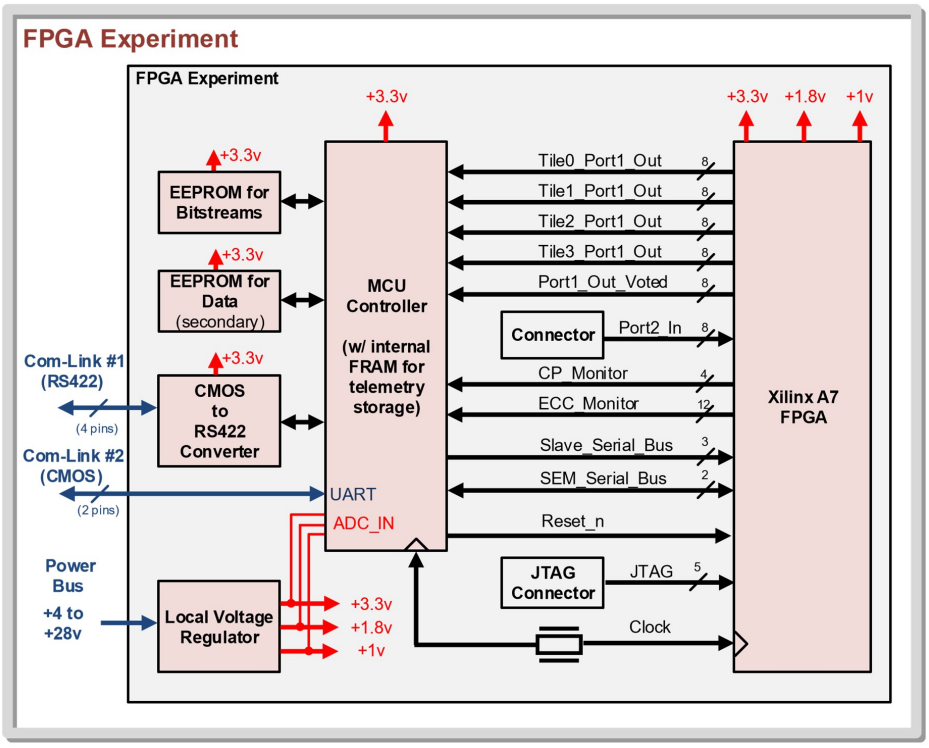


Figure 3.8: RadPC: Individual Tile [12]

The four tiles are implemented on a Xilinx FPGA. Each tile is a MicroBlaze Softcore,

a blackbox processor developed by Xilinx for their FPGAs. The inclusion of the fourth tile increases the likelihood of recovery in the event of multiple SEEs in two separate tiles occurring simultaneously or successively during voting. An MSP430 microcontroller functions as the Micro Controller Unit (MCU) of the board. The MCU controls the serial flash memory chips containing the bitstreams and the telemetry data packet storage.

A voter is used to implement the QMR system. Upon reaching a software checkpoint, the voter pauses the tiles, compares the tile outputs, and votes for the majority. If a tile disagrees with the majority, recovery procedures are started. The juggling architecture uses the undamaged tiles to reset the faulted tile. [8] Depending on the number of tiles being faulted, the voter triggers partial reconfiguration (PR) and memory scrubbing, or full reconfiguration (FR). Table 3.2 below shows the recovery procedures for all possible voted tile outputs.

Agrees	Disagrees	Majority	PR	Scrub	FR
4	0	✓	X	X	X
3	1	✓	✓	✓	X
2	1 - 1	✓	✓	✓	X
Anything Else		X	X	X	✓

Table 3.2: Recovery Procedures for Voted Outputs

The following paragraph outlines RadPC's response for each case in the above table. Figure 3.9 provides a visual block diagram of tiles and components triggering and undergoing recovery procedures.

In the first case given in the table, all tile outputs agree and the voter achieves a majority. There is no fault to recover from and the voter allows the tiles to resume operation. In the second case, three tile outputs agree and the voter achieves a majority with the fourth

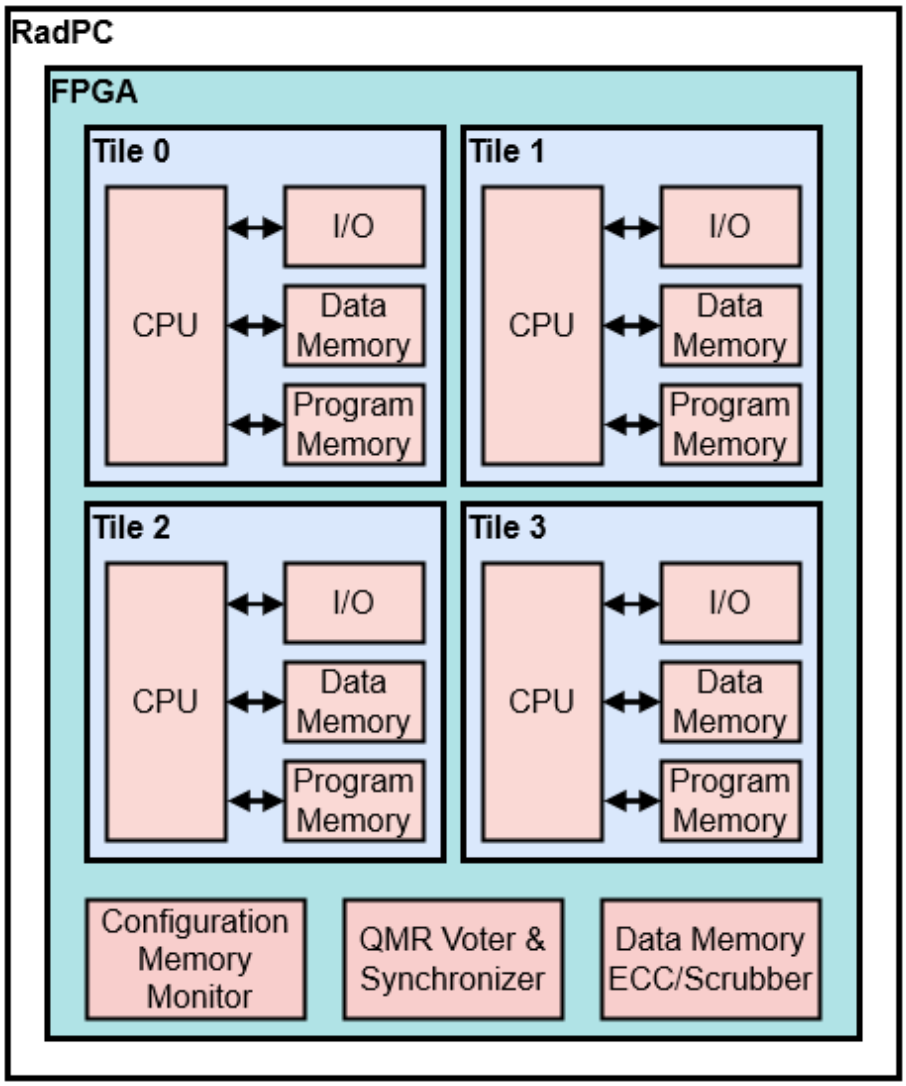


Figure 3.9: RadPC: Quad Modular Redundant Computer Architecture

tile undergoing recovery procedures. This consists of the Soft Error Mitigation Controller (SEM Controller) for the configuration memory being paused, the faulted tile being partially reconfigured, and the Data Memory Scrubber checking the data memory of all four tiles. For PR, the voter sends a request to the MCU to rebuild the faulted tile. The MCU pulls the tile's partial bitstream from the serial flash memory chip and loads it into the configuration memory. Once in the configuration memory, the current logic architecture of the FPGA is

wiped, including the tile's data memory, and the new architecture described by the partial bitstream is implemented. The Data Memory Scrubber sits outside the tile alongside the voter. Each individual tile has its own data memory implemented inside the FPGA's memory blocks. To bring the faulted tile back into synchronization with the other tiles, the Data Memory Scrubber compares the data memory from each tiles, votes for a majority, and overwrites incorrect entries with entries from the memory of a correct tile. The Data Memory Scrubber walks through all memory addresses until the data memory of all tiles match. Once the memory recovery is completed, the voter allows the tiles to resume operation. For the third case, two tile outputs agree and the voter achieves a majority with those tiles. The other two tiles disagree with both either other and the majority. These tiles undergo the same recovery procedures outlined in the second case for the faulted tile. In all other cases, the voter is not able to determine which tiles are faulted and triggers an FR of the FPGA. This follows the same procedure as PR with a request being sent to the MCU for an FR. The primary difference is a PR uses the Data Memory Scrubber to bring a faulted tile up to speed so no data is lost during recovery. An FR forces any program running on RadPC-Lunar on the FPGA to completely restart.

A key point is that RadPC has two memory scrubbers, one that is part of the tile recovery procedures and a separate one for the FPGA's configuration memory. The Data Memory Scrubber is an integrated part of the voter and is only operational during recovery procedures after a PR. The second memory scrubber is the SEM Controller, which is completely separate from the voter and scrubs only the configuration memory on the FPGA. The SEM Controller is continuously operational except during the PR and FR recovery procedures. During PR and FR, the SEM Controller is paused for the duration to avoid write/read conflicts between the SEM Controller and MCU. The MCU will write either a full or partial bitstream of the serial flash memory chip to the configuration memory. The configuration memory of the FPGA is used to hold the full bitstream for RadPC and

partial bitstreams for individual tiles. During normal operations, the SEM Controller scans the configuration memory, repairs faulted bits, and injects a fault into the configuration memory.

FPGA Configuration Memory

Simulation of radiation induced faults was done via the SEM Controller used by Xilinx FPGAs. This system injects faults by executing a NOT operation on a single bit in configuration memory. Changes in the configuration memory propagate to the implemented computer architecture on the FPGA. This makes the configuration memory particularly vulnerable to SEEs as a single bit flip can completely change a logic circuit on the FPGA fabric.

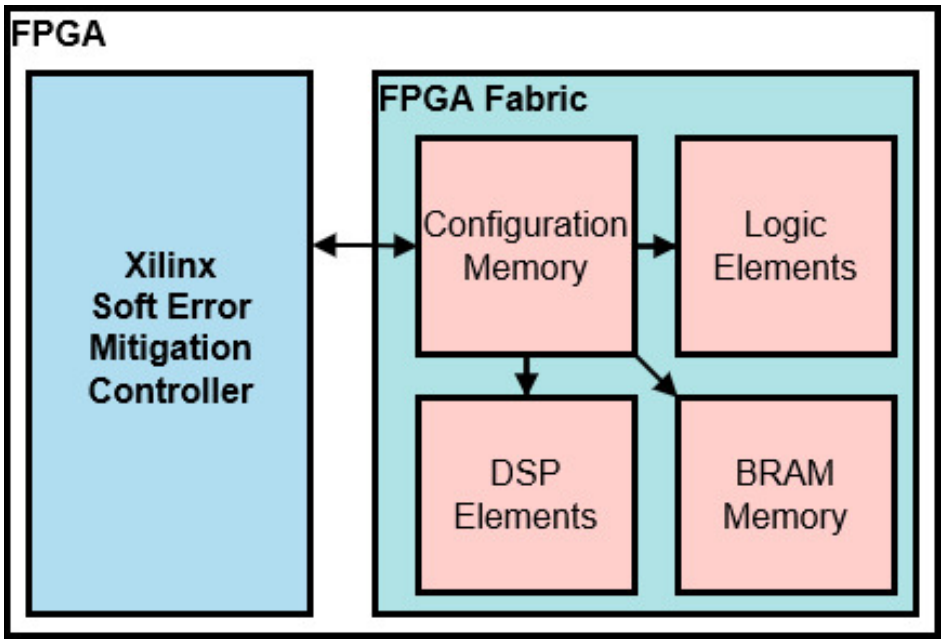


Figure 3.10: SEM Controller: Injection into Configuration Memory

During normal operation, the SEM Controller is continuously scanning the Error Checking and Correcting (ECC) codes and Cyclic Redundancy Check (CRC) codes for the configuration memory. Sets of memory addresses in the configuration memory are divided

into frames with an ECC code. This allows the SEM Controller to find and repair frames that have suffered faults. In addition to the ECC codes on the frames, CRC codes are implemented on arrays of frames. This allows the SEM Controller to identify and rapidly repair faults that cross frame boundaries. A block diagram of the SEM Controller and its interaction with the FPGA fabric is shown in Figure 3.10. For conceptual purposes, the configuration memory, logic elements, DSP elements, and BRAM memory of the FPGA are shown as blocks. On the hardware, these elements are physically scattered across the FPGA. The purpose of this distribution is due to signal path integration and timing constraints. Since configuration memory is the largest element and most physically distributed element on the FPGA, it is more susceptible to radiation induced faults.

During fault injections, the SEM Controller flips a bit in the FPGA configuration memory. For the ISS Payloads, the bit was arbitrarily selected, Linear Frame Address 0x0000011, and the injection operated on a cadence of an injection every 10 seconds. After the injection, the SEM Controller scans the ECC and CRC to find the fault. Upon the fault being found, the SEM Controller attempts to correct the error using algorithmic methods. [2] This consists of an active partial reconfiguration by the SEM Controller on the frame with the fault to rewrite it with the corrected contents.

Stottler-Henke Algorithm and RadPC Programmability

Onboard the ISS, RadPC-Stottler was used to prove RadPC computer architecture's capability to function in a radiation environment while executing a complex algorithm . For the ISS Payloads, an algorithm was developed by Stottler-Henke to run on the RadPC-Lunar system. This algorithm was implemented on the Stottler Board and consisted of a C program to designed to receive and evaluate telemetry data packets from the Lunar Board. A snippet from the Stottler-Henke *main.c* with the header files is shown below to illustrate how the RadPC computer architecture was used on a software level. Within C, the RadPC

is accessed through header files which provide access to input/output, memory locations, and voter.

```
// LIBRARIES
#include "sleep.h"
#include "xgpio.h"
#include "conditional_util.h"
#include "datamemorylocations.h"
#include "radpc.h"
#include "sensor_faults.h"
#include "stateestimator.h"
#include "voterrecovery.h"
```

Stottler-Henke's algorithm ran a series of evaluations on the Lunar Board's telemetry data packets. These consisted of checking the tile outputs for faults, estimating the power used by the Lunar Board from voltage and current readings, and tracking if the Lunar Board caught a fault and successfully recovered. After the algorithm completed the evaluation, a hex encoded output was appended to Lunar Board data packet under evaluation and the data was stored for later retrieval. This process was repeated with the next data packet.

The main limitations in the development of the above algorithm was no optimization inside the C program compilation, no serial communication, and the software checkpoint were strictly enforced. Optimization inside the C compiler caused issues because it removed the software checkpoints from the C program. Without the software checkpoints, RadPC did not have a place to pause the tiles and run its voter and recovery procedures. The lack of serial communication was due to RadPC lacking the hardware support for it. RadPC uses a parallel 8 line communication port for communication with the FPGA. Given the parallel nature of the FPGA, the parallel communication port was an efficient method to transmit telemetry data packets to the MCU. However, the lack of a serial port greatly limits amount

and type of data the C program can receive and transmit. Finally, software checkpoints are strictly enforced as each of the four tiles in the QMR system needed to maintain its synchronisation with the other tiles. After a major operation in the C program, RadPC needs to pause, synchronizes the tiles and voter on the tiles' outputs. Failure to do so will trigger a partial reconfiguration (PR) on the slowest tile.

The development of the algorithm by Stottler-Henke was used by the MSU team to evaluate the programmability of the RadPC system as a whole. During the development of RadPC, the MSU team focused on the firmware of the FPGA, the software on the MCU, and the PCB hardware for the system. The software implemented on the RadPC computer architecture was a counter program which allowed the tile outputs to be easily compared by the voter. The more complex program allowed the MSU team to evaluate how a program would be developed for RadPC. Overall, Stottler-Henke's development showed that RadPC was difficult to develop for. Programming in C for RadPC required knowledge of the underlying VHDL code and physical hardware. However, the algorithm was successfully implemented. RadPC is limited by the programming environment not being abstracted from the hardware. Because of the software checkpoints, parallel ports, and inability for code optimization in the compiler, there is a serious limitation in the speed of external users learning to program RadPC.

Operations and Results

Payloads were developed and deployed individually to the ISS, shown in 3.11. The first payload, Payload 1, was delivered to NanoRacks in October. Following the delivery of Payload 1, but before its deployment, the decision was made to build and deploy a second payload. The purpose was to increase the amount of time in a space radiation environment by doubling the hardware under test. The payloads were physically and functionally identical. The internal hardware for Payload 2 was originally the backup flight PCB stack for Payload

1. The external NanoLab's case was supplied by NanoRacks and Payload 2 was programmed, tested, and delivered to NanoRacks in December 2021. A few weeks later, Payload 1 flew on SpaceX CRS24 and installed on the ISS in December 2021. Payload 1 payload returned to Earth in May of 2022.

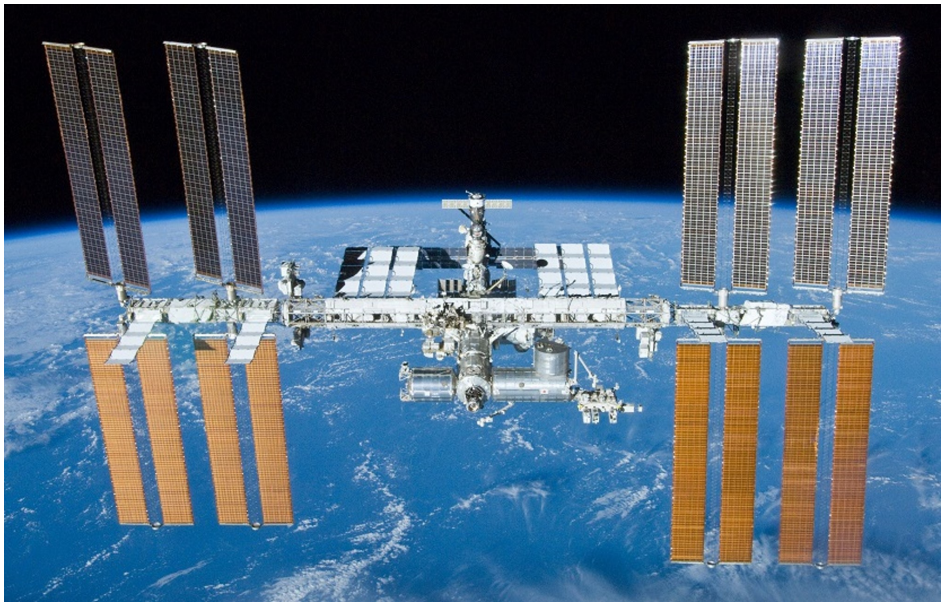


Figure 3.11: International Space Station

Payload 2 flew on NG-17, shown in 3.12, and was successfully installed on the ISS in February 2022. Payload 2 is currently still in operation on the ISS as of March 2023 with a scheduled return on the next available resupply mission.

As of the time of writing, Payload 1 was operational on the ISS for a total of 5 months and returned to Earth. Payload 2 was operational on the ISS for a total of 13 months and is still onboard the ISS in operation. This section discusses the operation and experimental data of the ISS payloads.

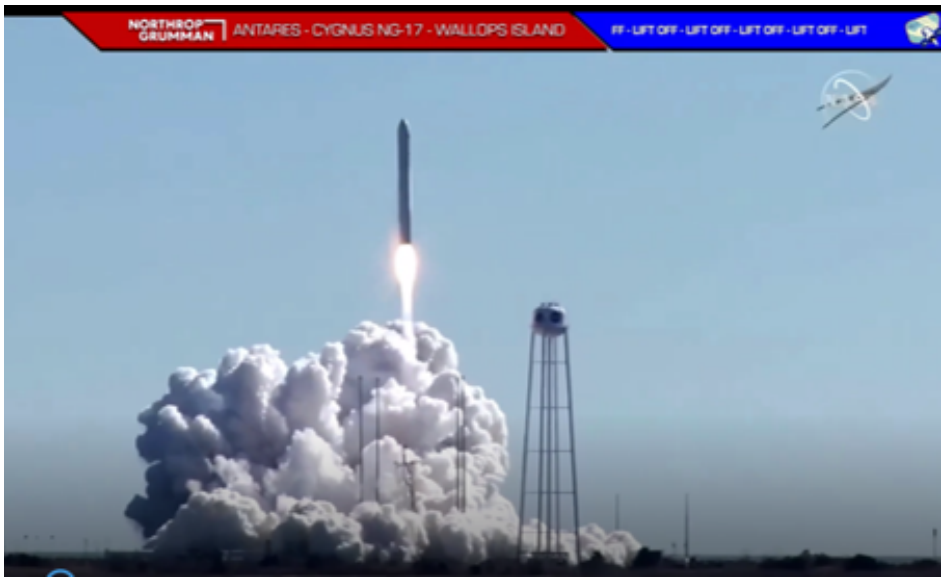


Figure 3.12: NG-17 Launch Carrying RadPC Payload 2 to the ISS

Experimental Data Fields

Each payload consists of two separate, but related, experiments. The Lunar Board experiment which is focused on testing the RadPC computer architecture in a space radiation environment and the Stottler Board which focused on testing the same architecture while running an analysis program on data from the Lunar Board. Thus, the two payloads generated a total of four telemetry data sets. Each payload generated a data set from its Stottler Board and a data set from its Lunar Board. The Stottler Board data packets were marked with the STLR header and the Lunar Board data packets were marked with the FPGA header. Since Payload 1 and Payload 2 are functionally identical, the data was analyzed in groups of the STLR and FPGA packets. For the STLR telemetry data packets, the key fields are the packet identification numbers, reset counts, power data, and tile fault counts. For the FPGA telemetry data packets, the fields are packet identification numbers, reset counts, power data, SEM Fault Counts, SEM Injection Counts, and tile fault counts.

Packet Identification Numbers

The MCU generates an incrementing number that is assigned to the start of each packet when the data packet is built. The packet number functions as an identifier for the packet and allows quick error checking of the packets. The Unparsed Packet Numbers from Payload 2's Stottler Board are shown in Figure 3.13. These packet numbers should be a line counting up to the latest packet number.

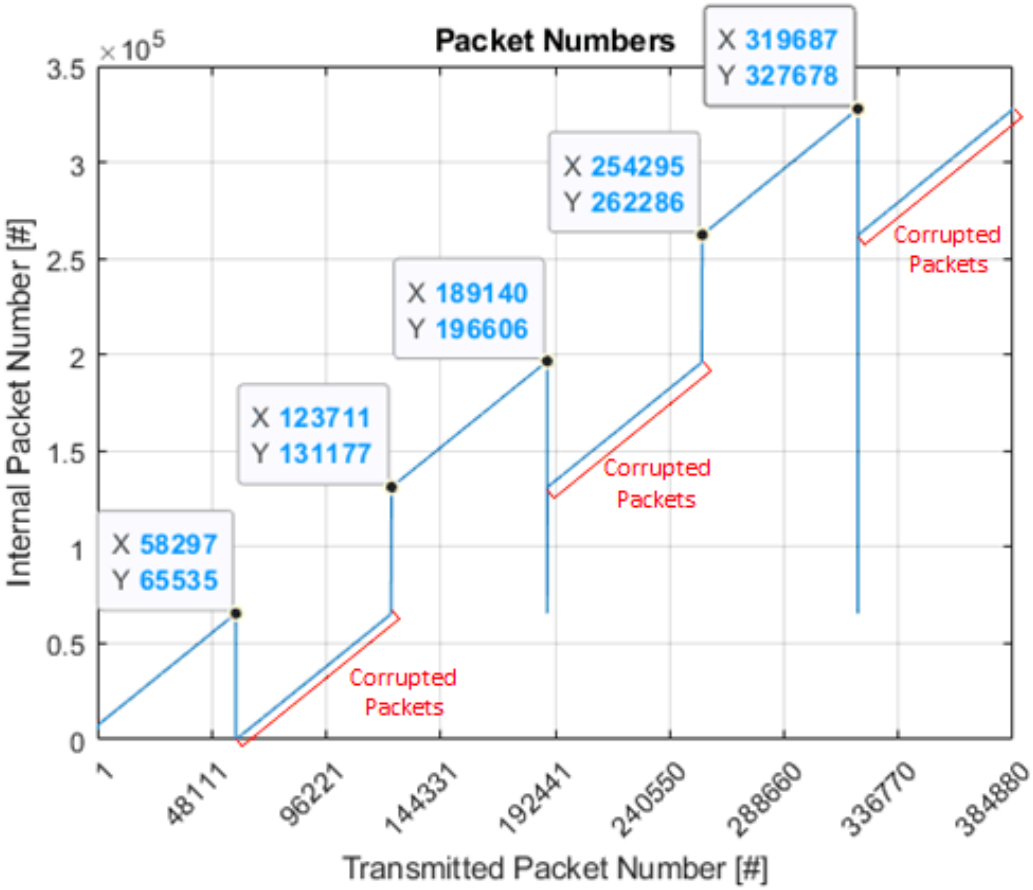


Figure 3.13: Payload 2 STLR: Unparsed Packet Numbers

However, there are two sections where the packet numbers drop and restart the count up. This issue was found after Payload 1 was deployed and Payload 2 had been delivered. In addition to the packet numbers, all packets include a CRC for all the data fields. A CRC

check and filtering out damaged packets produced the following graph shown in Figure 3.14.

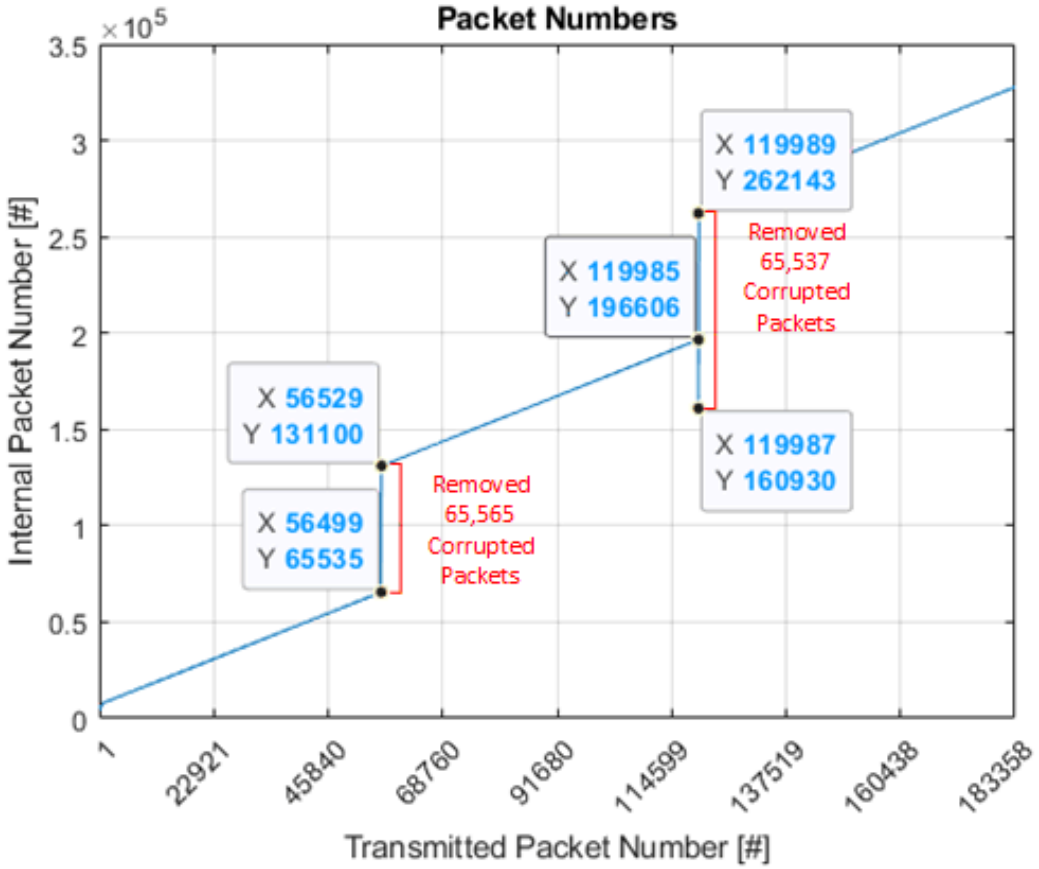


Figure 3.14: Payload 2 STLR: Parsed Packet Numbers

All of the packets in the two sections had been damaged and their CRCs did not match the packet's data. Further testing on the engineering model isolated the issue to the MCU's storage of the telemetry data packets in memory. When the packets are stored, MCU writes them to a serial flash memory chip, the IS25LP256D. This chip starts with all of its bits set to "1". Reading and writing is done on a byte basis with a single byte being read or written at a time. Rewriting is possible, but the chip does not support changing a bit from "0" to "1" during a write operation/reprogramming. Before a byte is reprogrammed, the sector or block that contains the byte must be erased, setting bits to "1" [9]. Therefore, a chip

is generally written to until completely full. The telemetry data packets are read out and transmitted to the Interface Board for later retrieval by NanoRacks, then the MCU erases the chip. A bug in the MCU's memory addressing incrementing loop reset the address early after only filling half the memory. This bug resulted in the MCU overwriting the previously stored packets with current packets. Since the previously stored packets were not pulled and transmitted beforehand nor was the memory erased, this write operation results in two packets undergoing an AND operation. The ANDed packets were transmitted with the undamaged packets, however the data was nonrecoverable. This caused half of the data from the ISS mission for both Payload 1 and Payload 2 to be lost. The data was lost on a two week cycle, one week of good data followed by a week of nonrecoverable data. All data used in the following graphs is parsed data that has passed its CRC check for integrity.

Reset Counts

As previously discussed, RadPC uses a microcontroller, MSP430, to receive data from the FPGA, build data packets, control memory chips, reconfigure the FPGA, and handle serial communication with off board devices. The MSP430 functions as the MCU for the entire board and is not a radiation hardened microcontroller. As such, the MCU is vulnerable to radiation induced faults. To counter this, the MSP430 was reset every twelve hours to clear out any potential faults. Figure 3.15 shows the MCU was successful reset for the valid CRC packets. The packets damaged by the memory overwrite and communication issues where removed from the graph and the discontinuities in the data marked. Without these discontinuities, the data would have a smooth slope showing that the MSP430 successfully reset every 12 hours with additional resets from the ISS power cycling the payload. Given the lack of issues found with the MCU, this implies that the MCU functioned within expectations for the duration of the ISS mission including for the duration of time in the corrupted data packets.

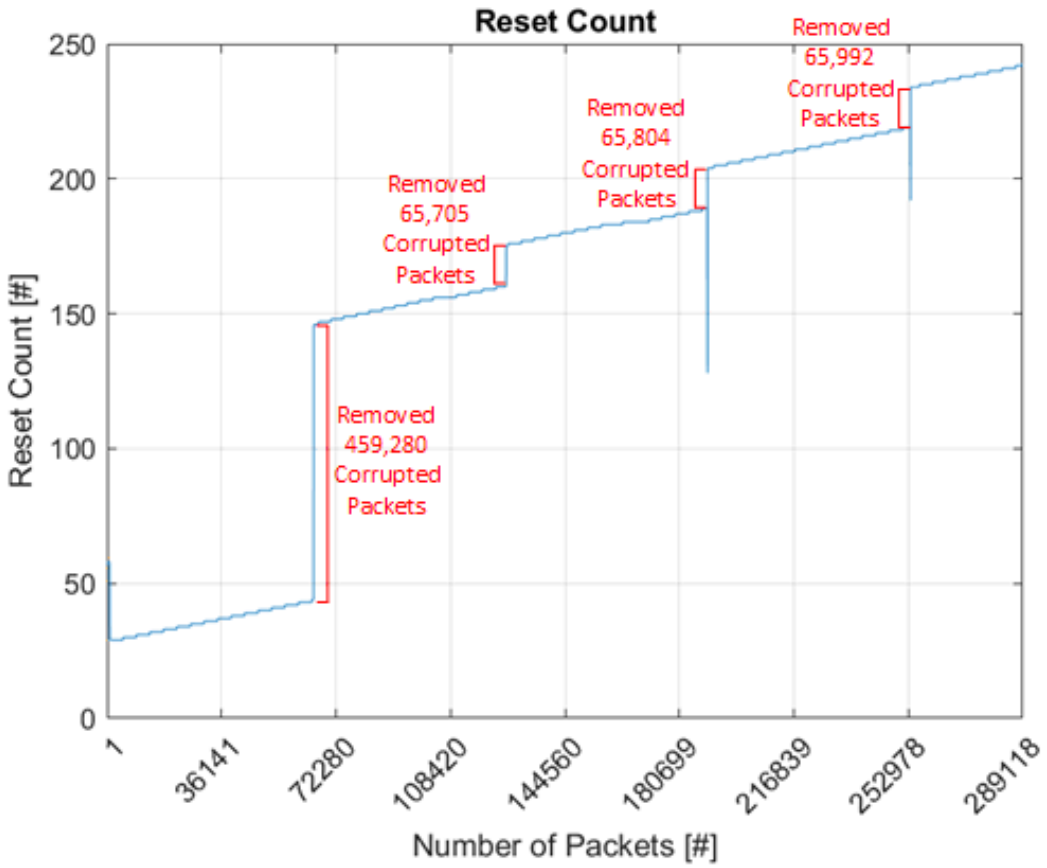


Figure 3.15: Payload 1 STLR: MCU Reset Count

Power Data

The main power supply for the RadPC Rev. 4 PCB was provided through a 5 Volt rail from the Interface Board. Once on the RadPC PCB, the 5 Volt rail is regulated down 3.3 Volt, 2.5 Volt, 1.8 Volt, and 1.0 Volt to supply power to the MSP430, the FPGA, the flash memory chips, and the RS422 communication chip. Overall, power data was stable across all four of RadPC PCB boards. Measurements are digital sampling from analog voltage and current sensors which leads to some noise in the power plotting. However, the power measurements were within an acceptable range (±2%) around the expected value of 0.5 Watts for the 5.0 Volt rail on all boards. A example for the 5.0 Volt rail is shown in Figure 3.16.

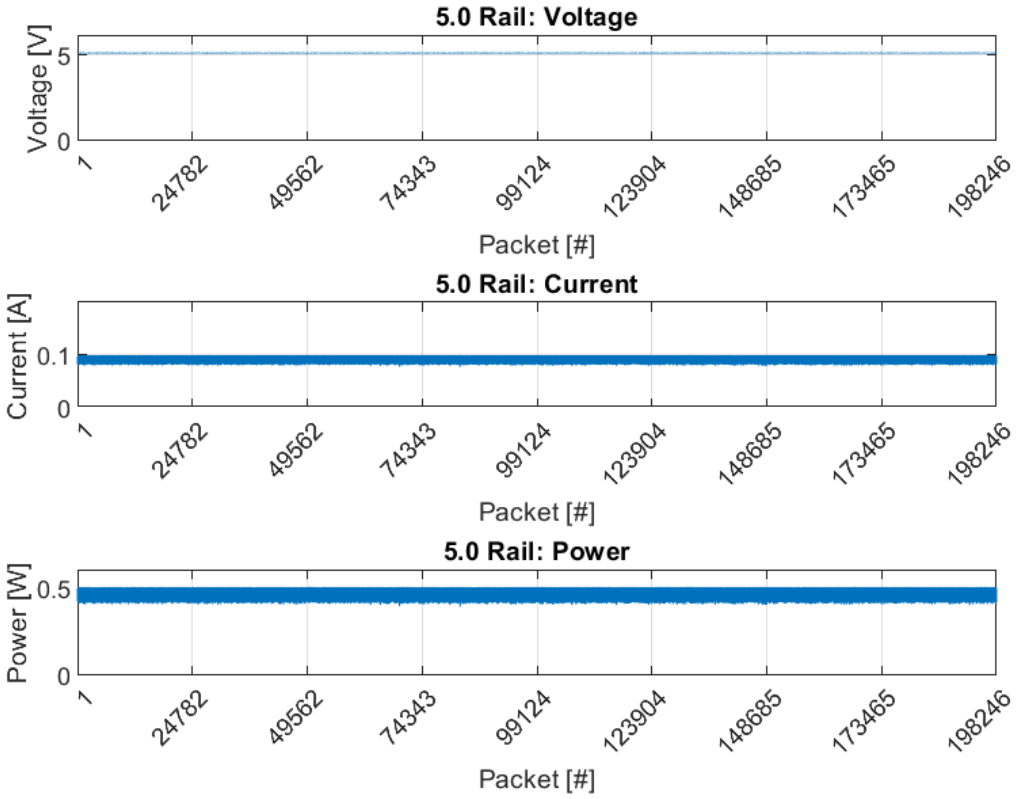


Figure 3.16: Payload 1 FPGA: 5 Volt Rail

SEM Data

During operation, the SEM Controller injected faults into the configuration memory on the FPGA. For the injection, the MCU will order an injection and the SEM Controller will respond by injecting a fault at Linear Frame Address 0x0000011 in the configuration memory. After the injection, the SEM Controller scans the configuration memory, repairs the fault if possible, and submits a report to the MCU. The MCU tracks the number of injections ordered and the number of fault reports submitted from the SEM Controller. These numbers are included that as the SEM Injection Count and SEM Fault Count respectively when building packets.

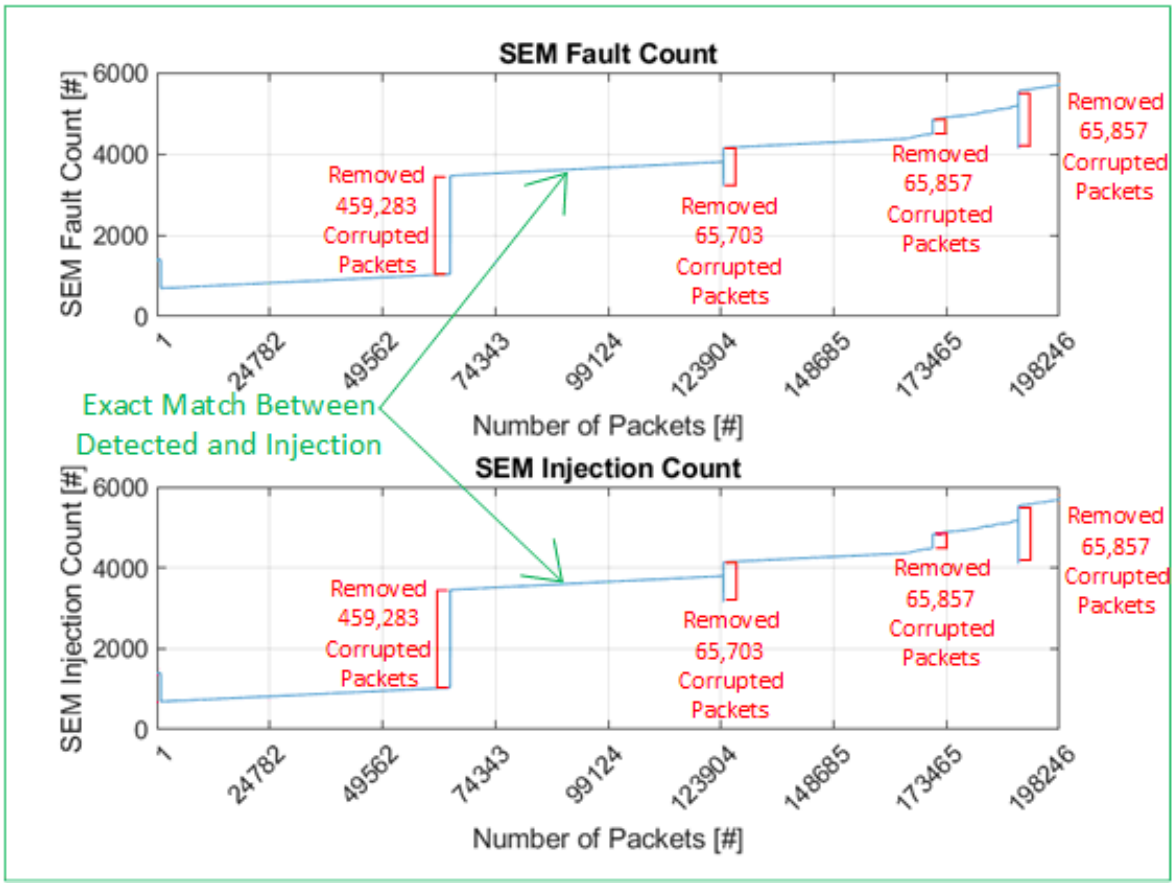


Figure 3.17: Payload 1 FPGA: SEM Fault and Injection Counts

The graph in Figure 3.17 shows the SEM Fault Count and SEM Injection Count. As shown, the injected faults and repaired faults are aligned in time and number. This suggests that the majority of faults detected in the configuration memory were the faults injected by the SEM Controller. RadPC was successful at detecting and recovering from these injected faults. The SEM Fault Count and SEM Injection Count are based on the MCU ordering an injection and the SEM Controller confirming that the injection was performed. While the above graphs suggest the majority of faults were injected faults, this conclusion cannot be confirmed. The SEM Controller has to run its detection and repair before faults can be verified as injected faults or radiation induced faults.

As shown, the SEM Correctable Tile Faults in Figure 3.18 corresponds with the SEM

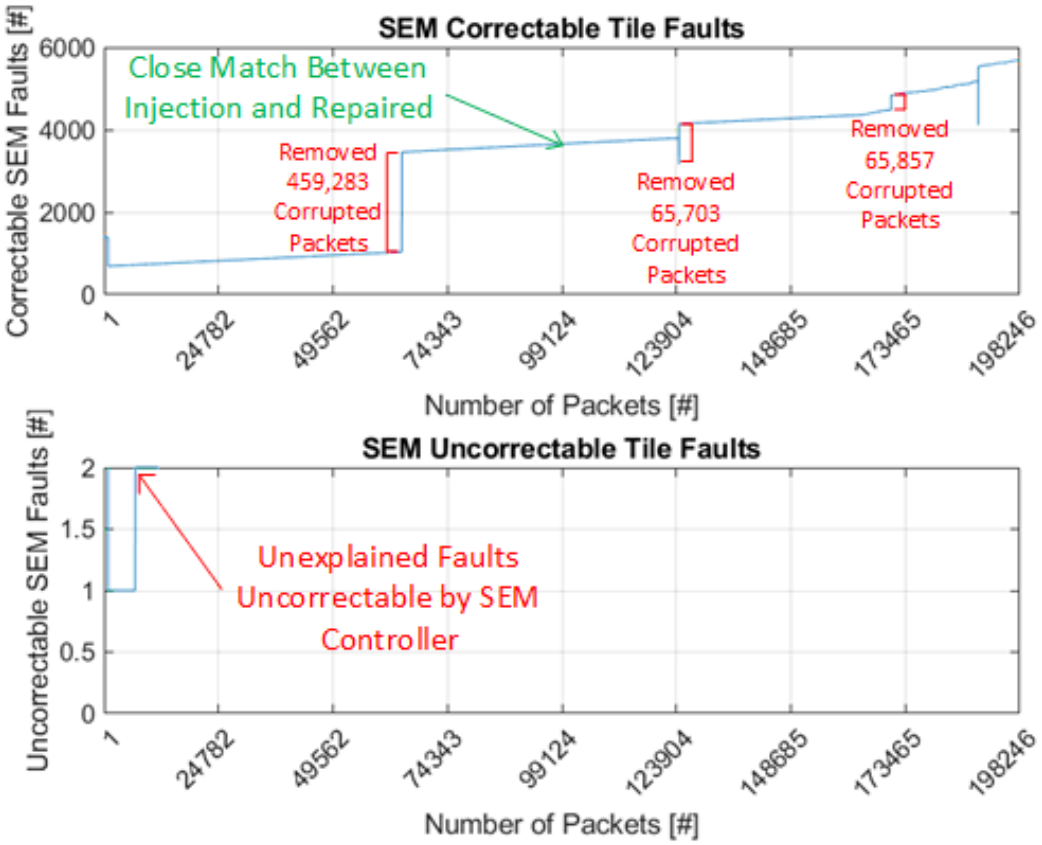


Figure 3.18: Payload 1 FPGA: SEM Correctable and Uncorrectable Tile Faults

Injection Tile Faults from Figure 3.17. This confirms that the SEM Controller injected faults and then performed a repair on those faults. Overall, the vast majority of faults detected in the configuration memory were injected faults and were successfully recovered from. However, a number of faults were SEM Uncorrectable faults. The SEM Controller detected the faults and was unable to repair them. These faults were not injected faults as all injected faults are SEM Correctable Faults. SEM Uncorrectable Faults were not expected and the SEM Controller is only able to detect and log this type of fault. Currently, RadPC is only able to recover from this type of fault, SEM Uncorrectable Faults, via a power cycle of the entire payload. This procedure restarts the entire system and the FPGA is reprogrammed

from the golden copy bitstream. This wipes out all the SEM Uncorrectable Faults as the SEM Controller is also restarted along with the implemented logic design.

Potential Radiation Induced Faults

A total of six uncorrectable SEM faults were found in Payload 1 data. A example of two uncorrectable tile faults shown in Figure 3.18. These faults are believed to be radiation induced faults. Injections done by the SEM Controller during operation on the ISS are know not to cause SEM uncorrectable faults. The six faults found do not correspond to the injected faults timing and the injection address is known not to cause uncorrectable faults. This cannot be confirmed beyond a statement that the behavior displayed by RadPC does not match its response to a SEM Controller injected fault and potentially matches a response to a radiation induced fault. Three faults was the maximum of SEM Uncorrectable Faults in the system simultaneously. Overall, the six faults were flushed out of the system when RadPC was restarted after the ISS power cycled the NanoRacks' internal payload rack.

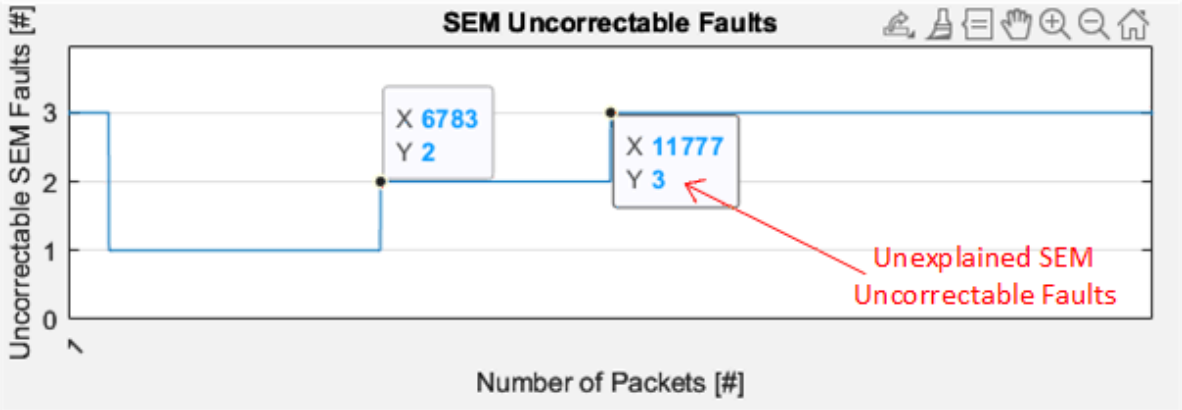


Figure 3.19: SEM Uncorrectable Faults with Transmitted Packet Numbers

Tile Fault Counts

The voter was also comprehensively tested using injection faults. Inside the RadPC-Lunar and RadPC-Stottler computer architecture design was an error injection circuit. This

circuit would inject faults into a tile within RadPC. These faults would be detected by the voter which would PR the faulted tile.

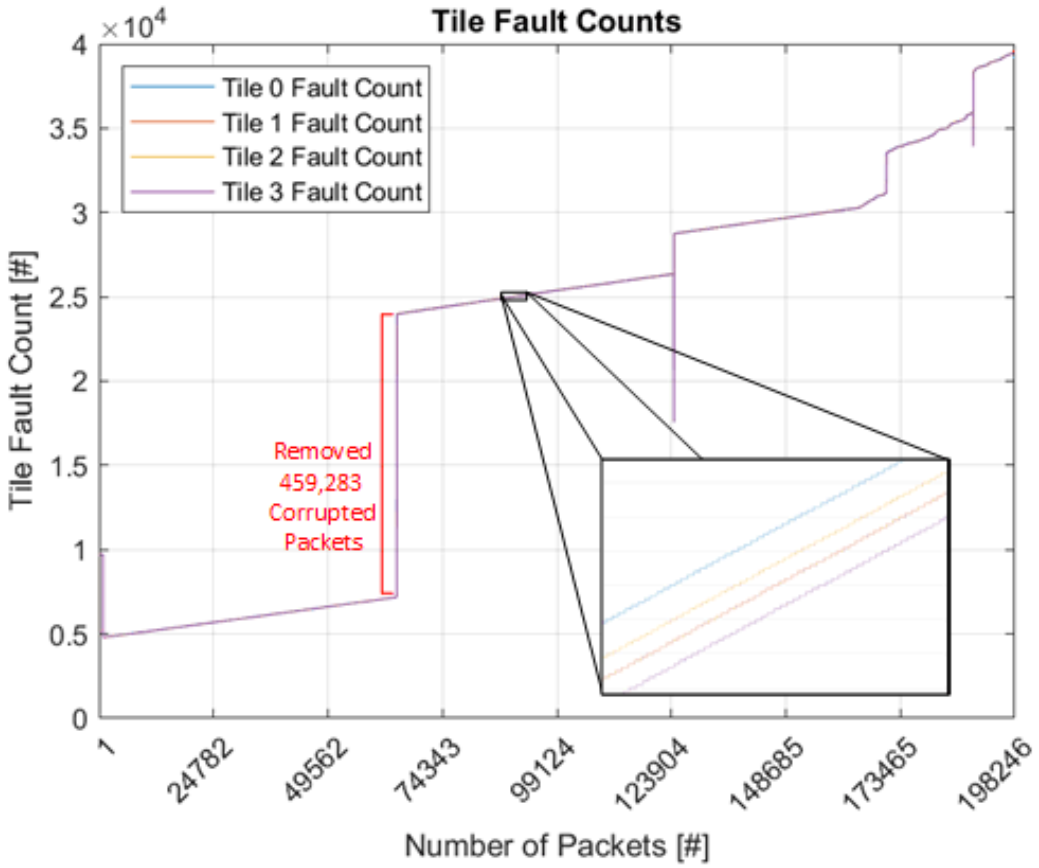


Figure 3.20: Payload 1 FPGA: Tile Fault Count

Tiles were faulted in a round robin pattern, starting with Tile 0 and ending with Tile 3. On the Lunar Board, the tile injections were consistently injected, detected, and repaired. As shown in Figure 3.20, this gives the graph a step pattern as the faulted tiles were successfully detected and repaired.

On the Stottler Board, tile fault injection was slowed. This was done to ensure that the tile fault injection would not effect Stottler's algorithm. Tile 0 was faulted a vast majority of the time. This was primarily due to the fault injections being done on a cycle controlled

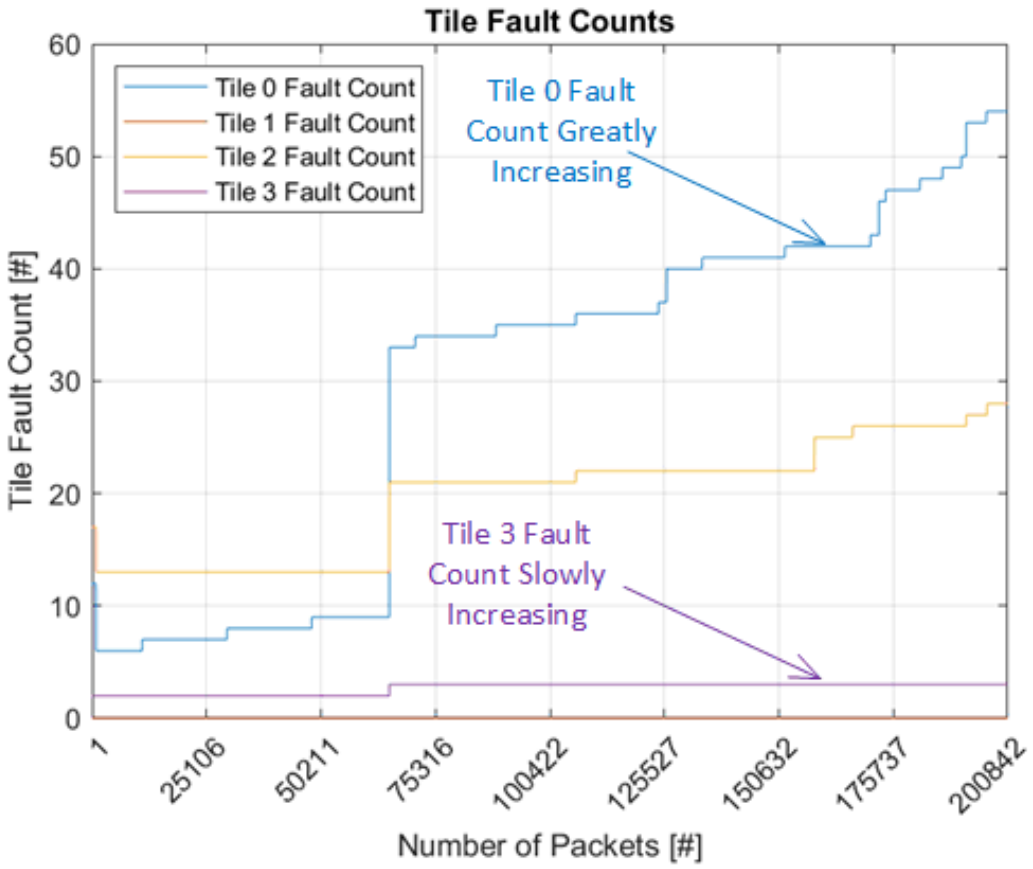


Figure 3.21: Payload 1 STLR: Tile Fault Count

by the MCU. The MCU would first fault Tile 0, then Tile 1, then Tile 2, then Tile 3, and finally the cycle restarted with Tile 0. Since MCU required a reset very 12 hours to ensure its functionality and the cadence of injections roughly matched the reset cadence, Tile 0 was faulted more often than all the other tiles combined. This does have the benefit that the detection of other tile faults are clearly visible in Figure 3.21. All of these detected faults were recovered from using either a PR or an FR. Overall, RadPC's voter was successful at detecting and recovering from injected tile faults while operating in a space radiation environment.

ISS Mission Conclusions

The two payloads were highly capable in recovering from injected faults while in a space radiation environment. A number of issues were found during the payloads' deployment to the ISS. However, these issues did not cause a mission failure. While the data was lost due the memory overwrite and the cadence of the SEM Controller's injections, the mission still generated more data than the previous missions testing RadPC on the ISS or using High Altitude Balloons. The data from the two payloads operating on the ISS shown RadPC was capable of recovering from faults injected to the configuration memory and the tiles. In total, RadPC operated on the ISS for a total of 18 months between two different payloads, Payload 1 and Payload 2. During that time, the payloads were fully operational and successful recovered from the injected and radiation induced faults.

Limitations of Fault Injection via SEM Controller

Analysis of the data from Payload 1 and Payload 2 showed that the current injection method had a number of limitations. The injections were done at a single nonessential, noncritical address. This address was arbitrarily selected during development as a method of testing the SEM Controller's memory scrubbing and recovery capability. As a method of rapid testing over multiple iterations of RadPC, a single address is an efficient method. However, this address does not cause errors in the tile outputs which limits detection of the faults to the SEM Controller. The voter cannot detect faults unless they propagated from the configuration memory to the implemented logic design and appear as errors in the tile outputs. This injection method is also limited as the cadence of the injected faults was too high for practical detection of radiation induced faults. The SEM Controller was configured to report multiple configuration memory faults to the MCU. However, the MCU was configured to track if a fault was reported, not where the fault was or if multiple faults

were detected. This functionality has not been developed for the MCU. Since faults are reported by address and the injection address was Linear Frame Address 0x0000011, the MCU would parse the SEM Error Report for only the fault detected message from the injected error. Additionally, the injections were performed every 10 seconds after which the the SEM Controller shifts to observation mode. In observation mode, the SEM Controller would scan the configuration memory and repair any detected faults, both injected and radiation induced. This configuration was effective for development purposes on the ground where recovery capability from fault injection was priority. In space, the radiation induced faults would be repaired along with the injected faults. Since radiation induced faults are rare relative to the injected faults, the high cadence of injected faults caused the radiation induced faults difficult to detect from the collected data. Estimates based on cadence and rate of radiation induced faults, 3-4 faults per device per day, suggested 90 days of operation on ISS before a radiation induced fault could be detected by RadPC.

Research Path

RadPC requires a more robust error injection system for simulating SEEs in the FPGA's configuration memory. This system would provide a systematic way of testing the configuration memory for critical locations that lead to tile output errors in FPGA's implemented logic design. If an injected fault is correlated with a tile fault, the SEM Controller's injection and the voter detection can be used to isolate injected faults from radiation induced faults. In order to accomplish this, the utilized configuration memory needs to be systematically tested for each implemented logic design and the tile outputs need to be cross referenced to injection locations to build a map of critical locations. Once this map is built, a single address that causes a known fault can be selected as the injection address for the RadPC computer architecture under testing. Furthermore, this system of testing can be applied to the data from the ISS Project to identify possible radiation induced

faults. Since the current injection method is known not to cause faulted tiles in RadPC, radiation induced faults approximate location in configuration memory can be identified. In the following chapters, a method of testing the configuration memory for critical locations which lead to tile output errors in RadPC is discussed.

CONFIGURATION MEMORY TESTING

On an FPGA, slight modifications in the logic design lead to large alterations in the configuration memory upon implementation. Even minor changes such as routing between components or moving components require that a completely new bitstream be generated and loaded into the configuration memory. Compared to the previous bitstream, the new bitstream will have different essential bits which actually cause changes in the implemented logic design. The locations of critical bits, essential bits that cause detectable faults in the implemented RadPC-Lunar computer architecture, are also altered with a new bitstream. Overall, RadPC requires a more robust fault injection system for simulating SEEs in the FPGA's configuration memory. Using the SEM Controller, RadPC requires a systematic way of testing the configuration memory for critical locations that lead to tile output errors in an FPGA's implemented logic design. Furthermore, this testing method should be repeatable for future iterations of RadPC. The development of this testing method is the Memory Project.

Experiment Design

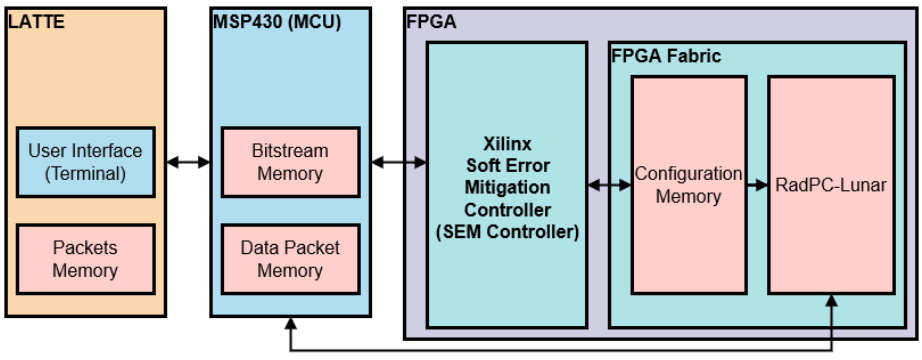


Figure 4.1: Block Diagram of Experiment Setup

The RadPC Rev.4 PCB (RadPC PCB) was used as the basis for this experiment. Configuration memory testing was conducted utilizing AMD Xilinx's Soft Error Mitigation Controller (SEM Controller) on an Artrix-7 FPGA with the RadPC-Lunar computer architecture implemented. The block diagram of the testing setup is shown in Figure 4.1.

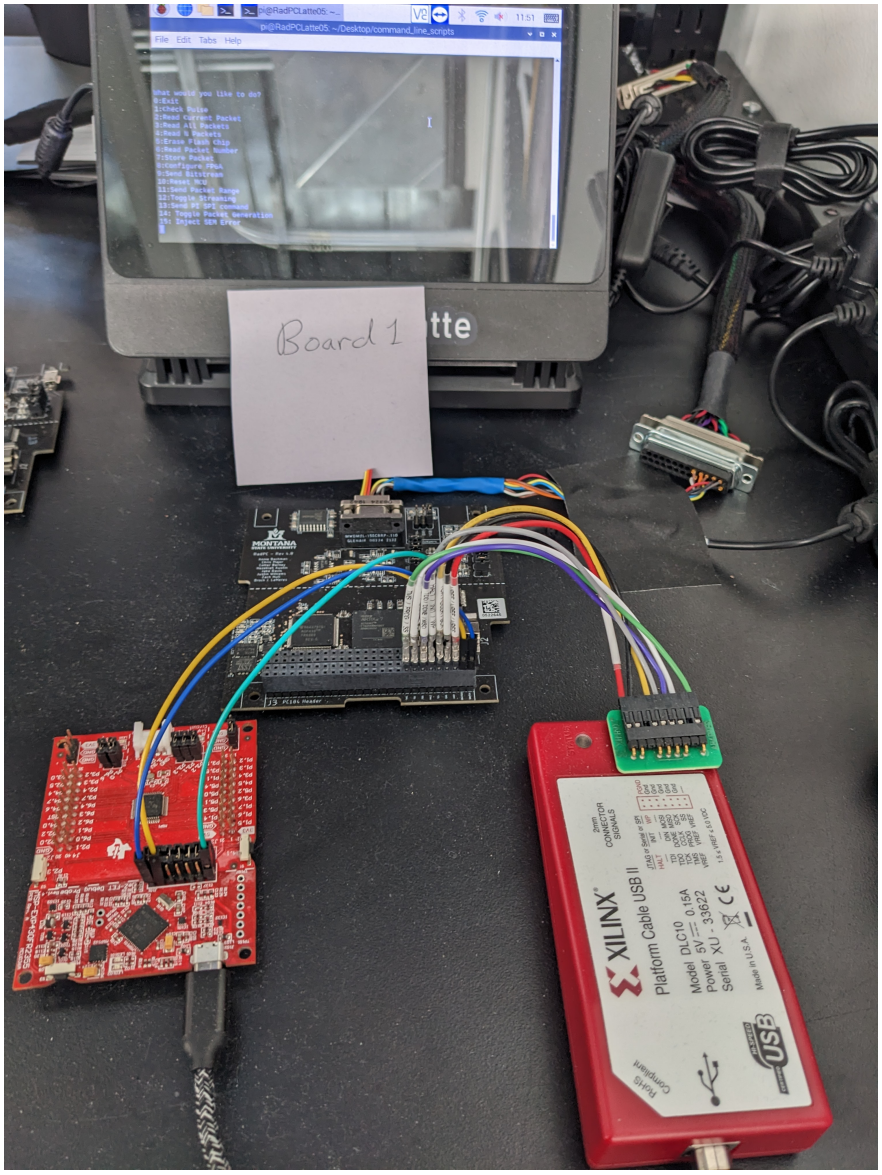


Figure 4.2: Configuration Memory Injection Testing Setup

A Lander Analog Telemetry Test Emulator (LATTE) provided the user interface for

sending commands and retrieving data packets from RadPC PCB. On RadPC PCB, the MSP430 microcontroller functioning as the Micro Controller Unit (MCU) received commands from the LATTE and would execute those commands. Also on RadPC PCB, RadPC-Lunar was implemented on the FPGA and its voter tracked tile outputs. A picture of the testing setup is shown in Figure 4.2. The MSP430 Launchpad and Xilinx Platform Cable which are used to program the MCU and FPGA, respectively, are visible at the bottom of the picture.

With this testing setup, the LATTE controlled the timing between injections and total number of injections per run. The MCU controlled the locations of the injections via addressing in the SEM Controller's Linear Frame Addressing system. The MCU also controlled packet generation for both regular data packets used in the ISS Project, labelled FPGA, and data packets for the SEM Controller's report, labelled SEM. On the FPGA, the SEM Controller controlled the actual fault injection upon command from the MCU. Depending on the injected fault type, the SEM Controller scanned the configuration memory after an injection and generated an error report that was returned to the MCU for SEM data packet generation. Also on the FPGA, RadPC-Lunar was under normal operation. The voter in RadPC-Lunar was used to detect and report faults that propagated from the configuration memory into the implemented logic design. This setup allowed essential bits which caused changes in implemented logic to be differentiated from critical bits which caused detectable faults in the implemented logic design.

Error Types

Two types of errors were injected during testing. The first type was designed to simulate radiation induced faults inside the configuration memory. Single location injections performed a bitflip at a single address simulate radiation strikes that only acted on a single bit. The diagram of a single fault injection for Linear Frame Addressing is shown in Figure 4.3.

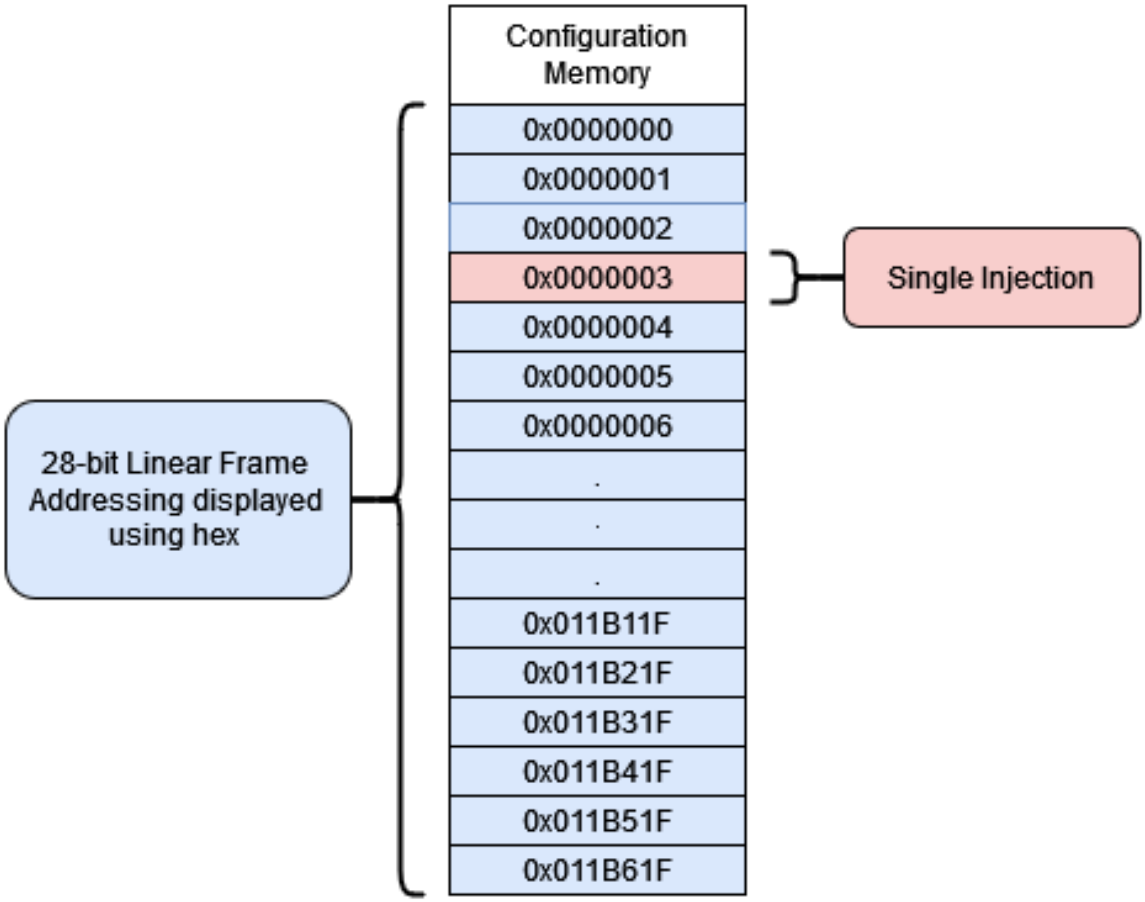


Figure 4.3: Single Fault Injection

After an injection, the SEM Controller is set to Observation Mode to detect and repair any faults in the configuration memory. The SEM Controller is capable of finding, repairing, and reporting faults within a maximum 150 milliseconds from the start of the scan for a fault. [2] With the single bit fault, the SEM Controller is capable of an average of 23 milliseconds depending on fault location. This particular type of fault were only a single bit is changed is the most likely occurrence in space operations.

The second fault type is continuous adjacent injection shown in Figure 4.4. This type of fault is unlikely to occur during space operations and was not used for simulating radiation induced faults in the configuration memory. The purpose is to deliberately overload the

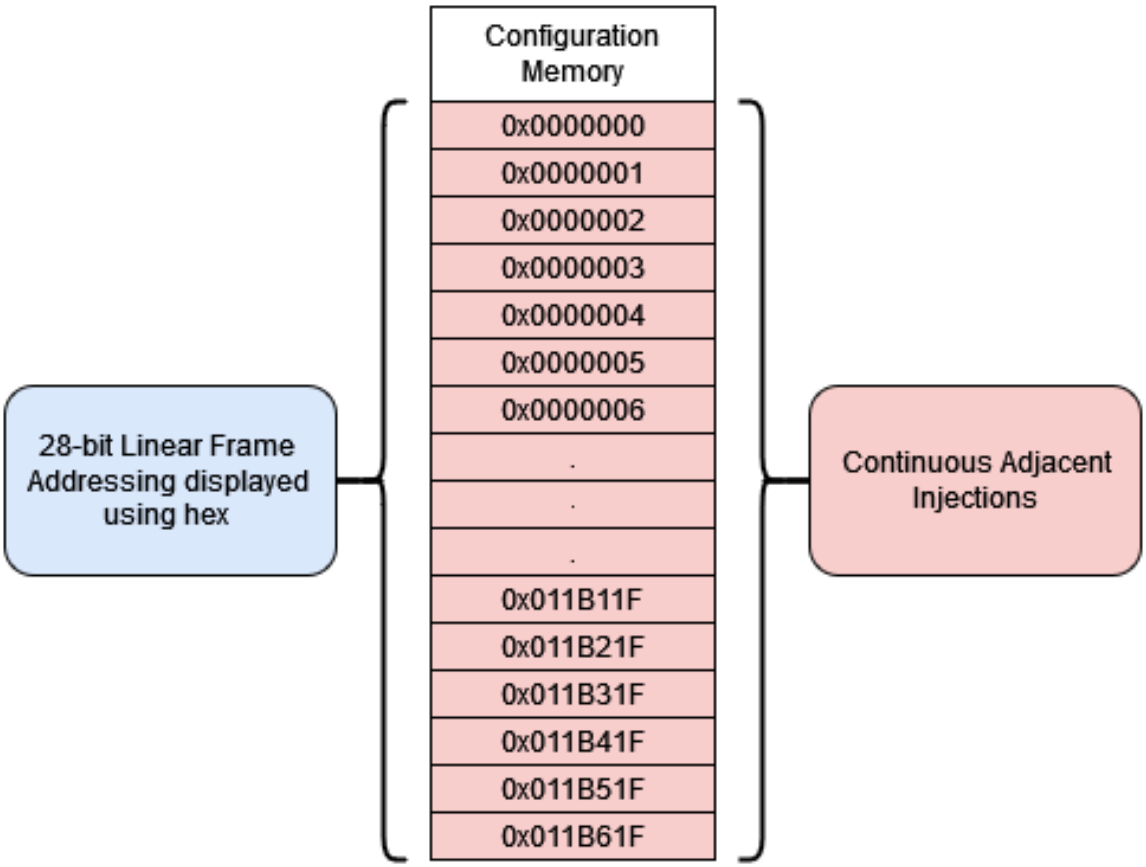


Figure 4.4: Continuous Fault Injection

configuration memory with faults which will propagate to the implemented logic design. By comparing the locations of the continuous injections and the tile output faults from RadPC-Lunar, the general location of critical fault addresses can be found. As previously discussed in double adjacent faults, the SEM Controller is not capable of more than a single injection at a time. Between injections, the SEM Controller was placed in Idle Mode to avoid it scrubbing the configuration memory and detecting the injected faults. During the time between injections, the MCU would also build data packets which shown if the latest continuous injection had triggered a detectable fault in RadPC-Lunar.

Procedure For Error Injections

Injection consisted of the following steps. The number, type, and timing of injections was set by the user on the external test unit, the LATTE. the fault injection addresses are set by the MCU on the RadPC Rev 4.0 Board. Found in the UART_A0_driver.c file, this code controls the fault injection address in linear frames addressing, the injection type, the SEM Controller's state, and processes the SEM Injection Report returned by the FPGA into a packet. Addresses are incremented by either bits or Linear Frame Addresses. The bits option injects at every bit location of the implemented logic system and is used for completeness in testing. The Linear Frame Addresses injects at the Least Significant Bit (LSB) of each frame of the implemented logic system and is used for speed in testing. After the FPGA receives the fault injection command from the MCU, the FPGA passes the command to its internal SEM Controller. The SEM Controller injects the fault into configuration memory via a bitflip at the given address. This fault in configuration memory determines the behavior of the design. It describes function block behavior and function block connectivity. [2] Fifth, the SEM Controller is set to either Idle or Observation. In Idle, the SEM Controller allows the fault to remain in the configuration memory and change the logic system implemented on the FPGA fabric. Depending on the location of the injection, this fault can induce faults in the output of the logic system. In Observation, the SEM Controller will check the Error Checking and Correcting (ECC) codes of the frames and the Cyclic Redundancy Check (CRC) codes of the arrays of frames for errors. If a fault is found, the SEM Controller will repair it and return an SEM Injection Report up the communication chain to the MCU to store until requested by the LATTE.

An example of a fault injection command is "IN C000000000O". This example would inject a single fault at Linear Frame Address "0000000", put the SEM Controller into Observation Mode to detect the fault, and have the SEM Controller repair any detected

errors in the FPGA's configuration memory. After the repair, the SEM Controller would generate an SEM Injection Report listing the classified errors by linear and Physical Frame Addressing.

Frame Addressing for Injections

There are two addressing schemes for fault injection into the configuration memory, Linear Frame Addressing and Physical Frame Addressing. Linear Frame Addressing utilizes arbitrary addresses from 0x0000 to 0x11B6 which are mapped to the Physical Frame Addressing of the implemented logic system.

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	0	S	S	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	W	W	W	W	W	W	W	B	B	B	B	B

Error Injection Command (Linear Frame Address)

Figure 4.5: Linear Frame Addressing [2]

The format for Linear Frame Addressing is shown in Figure 4.5. The "SS" is for the Hardware Super Logic Region (SLR) number and set to "00" and "LLLLLLLLLLLLLLLLL' is the Linear Frame Address (17-bit). [2] The "WWWWWWW" and "BBBBB" are the word address (7-bit) and bit address (5-bit) respectively. The word and bit address are the same between Linear Frame Addressing and Physical Frame Addressing.

Physical Frame Addressing utilizes the actual addresses of the implemented logic system in the FPGA's configuration memory and is shown in Figure 4.6. The "TT" is the block type (2-bit), 'H" is the half address (1-bit), "RRRRR" is row address (5-bit), "CCCCCCCCCC" is column address (10-bit), and "MMMMMMM" is minor address (7-bit). [2]

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	S	S	T	T	H	R	R	R	R	R	C	C	C	C	C	C	C	C	C	C	M	M	M	M	M	M	M	W	W	W	W	W	W	W	B	B	B	B	B

Error Injection Command (Physical Frame Address)

Figure 4.6: Physical Frame Addressing [2]

Physical Frame Addresses provide the information on the type and physical location, but are scattered across the configuration memory space to match with their physical locations in the FPGA fabric. Linear Frame Addressing is simpler to implement, however the addresses do not provide type and physical location of the frame under test. [2] For the testing of the configuration memory, all addresses for the SEM injections utilize Linear Frame Addressing. Physical Frame Addressing covers the entirety of the FPGA's configuration memory including the unutilized parts. A simulated fault may be injected into implemented elements or into the unutilized FPGA elements and Xilinx suggests using Linear Frame Addressing for this reason. [1] Since the total number of unutilized FPGA elements is approximately 33%, testing needs to be focused for time constraints. Runtimes for the SEM injections over the entirety of the mapped Linear Frame Addresses required 7 days at 1 injection per 0.25 seconds. Injecting over the entirety of the memory space would require 10.5 days of runtime. Since multiple runs were required for each type of fault injection, Linear Frame Addressing was chosen for its relative speed and focus on the implemented design. Additionally, this addressing mode maps the Linear Frame Addressing to the utilized Physical Frame Addresses. Its results are for simulated errors inside the design implemented on the FPGA. The Physical Frame Address is returned within the SEM Injection Report for a Linear Frame Address injection that was successfully detected and repaired. This allows Physical Frame Addresses and Linear Frame Addresses to be cross-referenced for analysis. Since the purpose of this research was to see the effects of simulated errors inside the implemented design, only Linear Frame Addressing was used. Physical Frame Addressing and testing the effects of fault injection on the entirety of the configuration memory space is left to future research.

Limitations of Error Injection

When interpreting the results from this experiment, there are a few key considerations. Xilinx does not support analysis of any specific customer design. [2] Tracing back the

fault location to the design is not supported nor are physical locations in configuration memory matched to components in the logic design. Furthermore, Xilinx specifically states that configuration memory injection can cause the implemented logic design to behave in unpredictable ways. [1] Correlation between the SEM Controller's injections and faults in the logic design are dependent on the voter detecting the fault. Since the voter is a part of the logic design, it shares the same vulnerability to configuration memory injections as the rest of RadPC-Lunar. Injections into the configuration memory defining the voter causes unpredictable multiple tile faults.

Linear Frame Addresses	Physical Addresses
00000026	00000026
00000027	00000027
0000002A	00000080
0000002B	00000081

Figure 4.7: Invalid Linear Frame Addresses

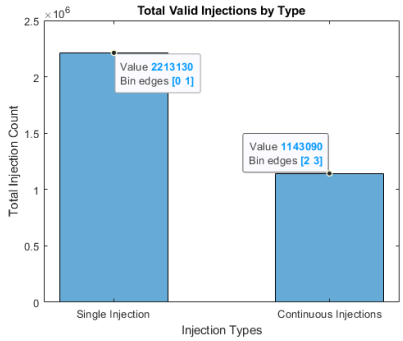
Furthermore, injections might not be detected because many of the configuration bit locations are masked or do not exist in the actual configuration memory For example with RadPC-Lunar, the Linear Frame Addresses 0x00000028 to 0x00000029 do not correspond to a valid injection address within the configuration memory. As shown in Figure 4.7, these Linear Frame Addresses are missing from the linear to physical address mapping. Injections at these locations return invalid injection reports. This gap appears at the edges of blocks in the configuration memory. The utilized configuration memory is divided into blocks in both Linear Frame and Physical Frame Addressing and the gaps appear between these blocks.

The final limitation is specific to multiple bit injections. There is an undocumented time delay between injection and propagation from configuration memory to the FPGA fabric. In single bit injections and double bit injections, this is not an issue as the injections are physical close together and the fault is localized. Multiple bit injections, such as those used for the continuous injection testing, can trigger short cascades of fault detection and correction. During these cascades, the SEM Controller is continuously detecting, repairing, and reporting faults, only for the sequence to restart with a new fault detection. These appear after a long series of injections and last for the next 4 or 5 reports. After this, the SEM Controller recovers and continues normal operation. The current theory is that the SEM Controller has an upper limit on the number of faults per report. On reaching that limit, the SEM Controller sends the report and starts building another report as quickly as possible. As the SEM Controller's programming is proprietary, this theory has not been verified.

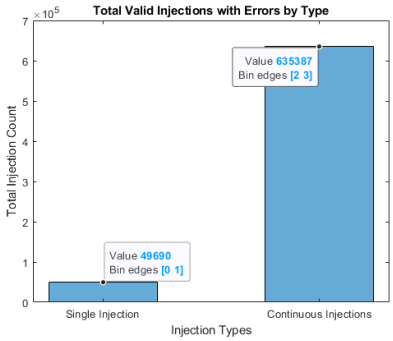
Configuration Memory Results

Overall, RadPC-Lunar is very resistant to injections into the configuration memory. A total of 2,213,130 valid single bit injections were performed on RadPC-Lunar over 4 months of testing as shown in Figure 4.8a. Only 2.25% of these injections caused a detectable fault in the tile outputs. All the detectable errors were recoverable either through a PR of the faulted tile or an FR of the whole RadPC-Lunar system.

The continuous injections totals are also shown in Figure 4.8a. For this type of injection, RadPC-Lunar suffered errors for 59.52% of the total number of injections. This percentage is lower than expected as the continuous injections were specifically developed to fault RadPC-Lunar across as many components as possible.



(a) Single and Continuous Injection Totals



(b) Single and Continuous Error Totals

Single Bit Fault Injection

In the case of single injection, the SEM Controller will usually find, scrub, and report a fault without any of the tiles or voter being damaged. Figure 4.9 shows the tile outputs and voted output of four sequential FPGA data packets. An injection was done before each of these packets was generated. The injection had no effect on the tile outputs and was either repaired before damaging the logic design or the injection location was a nonessential bit.

TILE0 COUNT	TILE1 COUNT	TILE2 COUNT	TILE3 COUNT	VOTED OUTPUT
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Figure 4.9: Single Injection: Normal Operation

The most common fault for single injection testing is a single tile fault. Since testing was conducted incrementally from the Linear Address 0x0000000 upwards, the tile faults tended to cluster in Tile 0 and Tile 1 as lower addresses received more coverage. An example of a single tile fault is shown in Figure 4.10 where Tile 1 is faulted. This error was repaired by PR on Tile 1.

TILE0 COUNT	TILE1 COUNT	TILE2 COUNT	TILE3 COUNT	VOTED OUTPUT
f	e	f	f	bf
f	e	f	f	bf
f	e	f	f	bf
f	e	f	f	bf

Figure 4.10: Single Injection: Repairable Single Error

The most severe fault suffered during single bit injection testing was a propagating double error, shown in Figure 4.11. This fault was originally a single fault for Tile 1 which the SEM Controller successfully found and repaired. However, the following injection on the next address in the configuration memory triggered a double fault in Tile 0 and Tile 3. As RadPC-Lunar was unable to determine which set of tiles was correct, the entire system was FRed and all tile outputs reset to zero. This error is reproducible with single injections around Linear Frame Address 0x00001E2.

TILE0 COUNT	TILE1 COUNT	TILE2 COUNT	TILE3 COUNT	VOTED OUTPUT
b	a	b	b	eb
b	a	b	b	eb
c	b	b	c	ec
c	b	b	c	ec

Figure 4.11: Single Injection: Propagating Double Error

For single bit injections, the SEM Controller coupled with RadPC-Lunar's voter system was able to recover from the majority of faults through PR while maintain operation. The remaining faults triggered an FR to effectively restart the FPGA.

Continuous Adjacent Bit Fault Injection

This method of injection is an extreme test case used to identify critical bit addresses and build a Linear Frame Address to Physical Frame Address mapping. An example of the Linear to Physical mapping is shown in Figure 4.12. Since Linear Frame Addressing does not provide information on the physical location of logic design components, the mapping is used to convert to Physical Frame.

Linear Frame Addresses	Physical Addresses
00000C3B	0040039F
00000C3C	004003A0
00000C3D	004003A1
00000C3E	004003A2
00000C3F	004003A3

Figure 4.12: Linear to Physical Address

As discussed in Experiment Design, the purpose of this type of injection is not to test RadPC's recovery capability. Rather, this injection method is designed to overwhelm the SEM Controller and force repeated faults in RadPC-Lunar. These faults are biased towards the start of the address space. Continuous injections tend to cause faults are lower address numbers as entire blocks of configuration memory are overwritten.

Approximately 40% of injections using the continuous injection type did not cause faults. RadPC-Lunar maintained its normal operations during continuous injection testing.

TILE0 COUNT	TILE1 COUNT	TILE2 COUNT	TILE3 COUNT	VOTED OUTPUT
e	e	e	e	4e
e	e	e	e	4e
f	f	f	f	4f
f	f	f	f	4f

Figure 4.13: Continuous Injection: Normal Operation

An example of normal tile outputs during continuous injection testing is shown in Figure 4.13. The counter program is on its 15 loop incrementing from 0 to F.

TILE0 COUNT	TILE1 COUNT	TILE2 COUNT	TILE3 COUNT	VOTED OUTPUT
f	b	b	b	6b
f	b	b	b	6b
c	c	c	c	6c
c	c	c	c	6c

Figure 4.14: Continuous Injection: Repairable Single Error

In the event of a single tile fault caused by continuous injection, RadPC-Lunar was able to recover via PR of the affected tile. An example of faulted Tile 0 undergoing PR and being synchronized with the other tiles is shown in Figure 4.14. This fault was caused by a single injection at Linear Frame Address 0x000089E.

TILE0 COUNT	TILE1 COUNT	TILE2 COUNT	TILE3 COUNT	VOTED OUTPUT
d	1	1	1	71
d	1	1	1	71
e	2	2	2	72
e	2	2	2	72

Figure 4.15: Continuous Injection: Propagating Single Error

An example of a single propagating error shown in Figure 4.15. This fault was caused by an injection at Linear Frame Address 0x0000326. The fault propagated for 160 data packets and 12 counter increments before being detected by the voter and corrected via PR of Tile 0. Correlating Tile 0 output faults with the injected faults and using the Linear to Physical Address mapping puts Tile 0 between Physical Frame Addresses 0x00000980 and 0x00020D9D in the configuration memory. Injections inside this range cause faults within Tile 0.

TILE0 COUNT	TILE1 COUNT	TILE2 COU	TILE3 COUNT	VOTED OUTPUT
e	a	a	a	8a
f	b	b	b	8b
f	b	b	b	8b
0	0	0	0	0

Figure 4.16: Continuous Injection: Suspected Propagating Triple Error

The final result found using continuous injections was a multiple tile fault. This triple tile fault requires multiple, simultaneous faults at specific locations in configuration memory to trigger three simultaneous tile fault in RadPC-Lunar. The likelihood of natural radiation bombardment causing the precise conditions required for this triple tile fault is extremely low. Given the unlikelihood of the fault occurring under natural radiation bombardment,

RadPC-Lunar is not designed to recognize these type of faults. However, the fault in Figure 4.16 was confirmed as a triple tile fault by checking that Tile 0 was PRed and the error persisted. Since a PR retrieves a partial bitstream from a memory chip external to the FPGA, the bitstream cannot have been altered by the SEM Controller's injections. As part of the PR process, the Data Memory Scrubber uses the other three tiles' data memory to synchronizes the PRed tile. After PR, Tile 0 should have fully recovered. Yet on the next count, Tile 0's output disagrees with the other three tiles. Another PR is triggered. This cycle continued until the system underwent full reconfiguration and was restarted. This fault was caused by an injection which affected the voter inputs received from Tile 1, Tile 2, and Tile 3. Since the voter was unable to recognize a triple tile fault, it treated this fault as a single tile fault and PRed the uncompromised tile.

FUTURE WORK

Three main avenues of future research were identified during the deployment of the payloads to the ISS and the development of the SEM injection process. These are configuration memory injection testing on the latest RadPC computer architecture, triple tile fault identification and recovery, and injection testing on physical adjacent addresses to the implemented logic.

Configuration Memory Injection Testing on Latest RadPC Architecture

The primary purpose of the testing described in this paper was to determine the addresses in configuration memory where injected faults cause errors in the logic design output. This testing is specific to the logic design implemented on RadPC-AI Payload 1 and Payload 2. The RadPC-AI computer architecture used four Xilinx MicroBlaze Softcores as the individual tiles in a QMR system with a voter and memory scrubber. This design is limited as any faulty tile output requires the entire system to pause, reconfigure the faulted tile, synchronize all tiles via checkpoints, and resume operation. Individual MicroBlazes functioned as black boxes inside of the larger RadPC architecture. An inability to repair or interact with the MicroBlazes' internal logic design limits recovery methods and data acquisition. An injection which causes a fault in configuration memory can trigger an error in a MicroBlaze in the FPGA. Even after the fault is corrected, this error can propagate through the MicroBlaze, completely undetected, until it reaches the output voter. Currently, the main solution to repair a faulted tile is a partial reconfiguration of the affected MicroBlaze.

Since this testing was completed, RadPC has been further developed and moved away from the MicroBlaze to a new RISC-V computer architecture. This new RISC-V computer architecture allows the repair and recovery procedures to operate on a lower architectural level compared to previous MicroBlaze computer architecture. The RISC-V computer

architecture is capable of register level resets and recovery procedures. One of the potential avenues of research will be applying the developed configuration memory injection method to the new RISC-V architecture and developing new recovery techniques for the faulted tiles based on the greater control the RISC-V architecture provides.

Triple Tile Fault Identification and Recovery

One of the limitations of the testing in this paper was triple tile fault detection and recovery. Injections into configuration memory can affect more than a single tile in the RadPC architecture implemented on the FPGA. Injections into the voter's configuration memory block can cause an fringe case where three tiles are faulted, or at least appear faulted based on their outputs. In the event that all three faulted tiles have the same output, the faulted tiles will outvote the uncompromised tile. Since these tile outputs exactly matches the single faulted tile case, the voter will order a partial reconfiguration of the single uncompromised tile. The tile is partial reconfigured from the bitstream which is not stored in configuration memory and is therefore unaffected by the injections. The partial reconfiguration sets the tile to the correct value. This is compared to the faulted tiles and the process restarts by voting the corrected tile as faulted. Since the partial reconfiguration only affects the single correct tile and ignores the triple faulted tiles, the system is forced into a reconfiguration loop. This loop generally continues until the microcontroller's FPGA reset timer triggers and the FPGA is completely reconfigured.

A potential avenue of research is implementing a repair or recovery method for this specific fringe case. Possible solutions include utilizing the MCU to trigger full reconfiguration if the same tile is faulted and partially reconfigured multiple times or having a logic circuit that compares program counters from each tile to detect repeated errors.

Injection Testing on Physical Adjacent Logic Elements

During this testing, Linear Frame Addressing was used due to its focus on the implemented design and time requirements for Physical Frame Addressing. An avenue of future work is the expansion the error injection to include Physical Frame Addressing. Given the size of the memory space, the Linear Frame Addressing could be used to find the edges of the utilized configuration memory blocks and Physical Frame Addressing could inject at locations physically close to but not part of the implemented logic. This would allow for targeted testing around the edges of the implemented logic to observe if those logic elements interact with the implemented logic.

CONCLUSIONS

Potential Radiation Induced Faults in Configuration Memory

As discussed in the Mission Overview, RadPC functioned on the ISS for a total of 13 months between two payloads. Six uncorrectable faults were detected in the configuration memory of Payload 1's Lunar Board. These faults did not effect the payload's normal operation and were removed by the next power cycle. The SEM Controller is known not to cause uncorrectable fault with its injections at Linear Frame Address 0x0000011. The injected tile faults inside the implemented logic are also known not to cause this type of fault. This suggests the six uncorrectable faults in the FPGA's configuration memory were radiation induced faults. Memory overwrites, serial communication issues, and frequent power cycles make verification difficult. Additionally, verification of the six faults being radiation induced faults through on the ground testing failed as the configuration memory testing as unable to replicate this uncorrectable fault.

Development of Systematic Configuration Memory Injection

The testing on the ground was not able to replicate the uncorrectable faults. Injections were done at all valid Linear Frame Addresses. None of these addresses triggered a similar SEM Uncorrectable Fault. This suggests the SEM Uncorrectable Faults were caused by faults inside the configuration memory, but outside where the SEM Controller is allowed to injection. Since Xilinx's configuration memory design is proprietary, this hypothesis cannot be tested. However, the lack of a similar response to the permitted injections suggests the SEM Uncorrectable Faults were caused by radiation.

Furthermore, this research into the SEM Uncorrectable Faults lead to the development of a systematic configuration memory injection process for FPGA-based aerospace comput-

ers. The FPGA configuration memory on RadPC can now be systematically tested. Faults detected by the voter inside the implemented logic design can be correlated to SEM injections in the configuration memory. This development allows a new method of tile fault injection for RadPC. Currently, two separate fault injection methods are used in RadPC, one via the SEM Controller for the configuration memory and one via implemented logic on the FPGA. With a map of RadPC's responses to configuration memory injection locations, the two methods can be combined. A single injection in the configuration memory can cause a tile fault in the implemented logic design on the FPGA.

Conclusions

In conclusion, the ISS Project and Memory Project successfully further the development of the RadPC aerospace computer. Radiation tolerance is a major consideration for the future of aerospace computers. RadPC provides a cost-effective, radiation resistant system for small satellite, data processing applications. RadPC has been deployed to the ISS for an on-orbit test mission with two payloads. On the ISS, RadPC operated successfully for a total duration of 5 months for Payload 1 and 13 months for Payload 2. A number of issues, memory overwrite, irregular communication glitches, SEM Injection cadence, were identified and resolved for future missions. Power supplies, RadPC's recovery methods, payload response to power cycles, and payload response to natural radiation bombardment were successfully tested. At the end of the mission, RadPC Payload 1 returned to Earth and was fully functional upon return. RadPC Payload 2 is still fully functional and in operation on the ISS as of March 21st, 2023.

Furthermore, data of the on-orbit missions motivated research that developed a more robust fault injection system for configuration memory testing. The configuration memory testing showed that key components within the logic design, individual tiles and the voter, are particularly vulnerable to faults in the configuration memory. The voter is a particular

point of vulnerability. Since all the tile outputs are channeled through the voter, it is the single point of failure across the entirety of the system. The overall system was successful in the majority of cases. Single tile faults were successfully recovered from and propagating multiple tile were recoverable via full reconfiguration of the FPGA upon the MCU reset.

Overall, this project successfully launched two payloads into space, where they were deployed on the ISS. Data from the ISS led to the development of a configuration memory fault injection process for FPGA-based computers.

REFERENCES CITED

[1] AMD Xilinx. *66570 - UltraScale Architecture Soft Error Mitigation Controller - Guidance for testing with error injection*, February 2016.

[2] AMD Xilinx. *Soft Error Mitigation Controller v4.1: LogiCORE IP Product Guide*, May 2022.

[3] H. J. Barnaby. Total-ionizing-dose effects in modern cmos technologies. *IEEE Transactions on Nuclear Science*, 53(6):3103–3121, 12 2006.

[4] J.L. Barth, C.S. Dyer, and E.G. Stassinopoulos. Space, atmospheric, and terrestrial radiation environments. *IEEE Transactions on Nuclear Science*, 50(3):466–482, 6 2003.

[5] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage, and T. Turflinger. Digital single event transient trends with technology node scaling. *IEEE Transactions on Nuclear Science*, 53(6):3462–3465, Dec 2006.

[6] S. Caorsi and G. Cevini. Tdfem analysis of the scattering properties of shielding structures. In *2003 IEEE International Symposium on Electromagnetic Compatibility, 2003. EMC '03.*, volume 1, pages 288–291 Vol.1, May 2003.

[7] Cor Claeys and Eddy Simoen. *Radiation Effects in Advanced Semiconductor Materials and Devices*. Springer, Berlin, Heidelberg, 1 edition, 2002.

[8] M. Garvie and A. Thompson. Scrubbing away transients and jiggling around the permanent: long survival of fpga systems through evolutionary self-repair. In *Proceedings. 10th IEEE International On-Line Testing Symposium*, pages 155–160, July 2004.

[9] Integrated Silicon Solution, Inc. *IS25LP256D IS25WP256D 256Mb SERIAL FLASH MEMORY WITH 166/104MHZ MULTI I/O SPI & QUAD I/O QPI DTR INTERFACE, DATA SHEET*, July 2017.

[10] Connor R. Julien, Brock J. LaMeres, and Raymond J. Weber. An fpga-based radiation tolerant smallsat computer system. In *2017 IEEE Aerospace Conference*, pages 1–13, March 2017.

[11] Connor Russell Julien. A radiation tolerant computer mission to the international space station. *ScholarWorks*, 2017.

[12] Brock J. LaMeres. Fpga-based radiation tolerant computing. *Journal of Aerospace Information Systems*, 2012.

[13] Brock J. LaMeres. *Highly Integrateable AI Modules for Planning, Scheduling, Characterization, and Diagnosis (ISS Payload Installation)*. Montana State University, February 2023.

[14] Dina G. Mahmoud, Gehad I. Alkady, Hassanein H. Amer, Ramez M. Daoud, Ihab Adly, Youssef Essam, Hassan A. Ismail, and Kirollos N. Sorour. Fault secure fpga-based tmr voter. In *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, June 2018.

[15] Christopher M. Major, Annie Bachman, Colter Barney, Skylar Tamke, and Brock J. LaMeres. Radpc: A novel single-event upset mitigation strategy for field programmable gate array–based space computing. *Journal of Aerospace Information Systems*, 18(5):280–288, 2021.

[16] A. Makihara, M. Midorikawa, T. Yamaguchi, Y. Iide, T. Yokose, Y. Tsuchiya, T. Arimitsu, H. Asai, H. Shindou, S. Kuboyama, and S. Matsuda. Hardness-by-design approach for 0.15 /spl mu/m fully depleted cmos/soi digital logic devices with enhanced seu/set immunity. *IEEE Transactions on Nuclear Science*, 52(6):2524–2530, Dec 2005.

[17] J R Schwank. Space and military radiation effects in silicon-on-insulator devices. *IAEA*, 9 1996.

[18] James R. Schwank, Marty R. Shaneyfelt, and Paul E. Dodd. Radiation hardness assurance testing of microelectronic devices and integrated circuits: Radiation environments, physical mechanisms, and foundations for hardness assurance. *IEEE Transactions on Nuclear Science*, 60(3):2074–2100, June 2013.

[19] L. Sterpone and M. Violante. Analysis of the robustness of the tmr architecture in sram-based fpgas. *IEEE Transactions on Nuclear Science*, 52(5):1545–1549, 2005.

[20] Ramazan Uzel and Alime Özyildirim. A study on the local shielding protection of electronic components in space radiation environment. In *2017 8th International Conference on Recent Advances in Space Technologies (RAST)*, pages 295–299, June 2017.

[21] Raymond Joseph Weber. Radpc: A novel single-event upset mitigation strategy for field programmable gate array–based space computing. *Montana State University Library ScholarWorks*, 2014.