

RECONFIGURABLE HARDWARE ACCELERATORS FOR HIGH  
PERFORMANCE RADIATION TOLERANT COMPUTERS

by

Raymond Joseph Weber

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

of

Doctor of Philosophy

in

Engineering

MONTANA STATE UNIVERSITY  
Bozeman, Montana

May, 2014

© COPYRIGHT

by

Raymond Joseph Weber

2014

All Rights Reserved

## TABLE OF CONTENTS

1. MOTIVATION AND BACKGROUND .....	1
Current and Future Space Computing Requirements .....	1
Radiation Effects on Semiconductor-Based Electronics .....	2
Conventional Radiation Effects Mitigation Techniques.....	5
Problems with Conventional Radiation Effect Mitigation Strategies.....	9
Commercial Off-The-Shelf FPGAs as a Solution .....	10
Problems with FPGAs in Radiation Environments .....	11
2. MSU’S APPROACH TO FPGA RADIATION TOLERANCE .....	14
TMR with Scrubbing and Spares .....	14
Silicon Based Spatially Aware Radiation Sensor .....	15
TMR+Scrubbing+Spares with Radiation Sensor Feedback System.....	17
System and Sensor Testing .....	18
Next Steps.....	21
3. DIGITAL HARDWARE DEVELOPMENT .....	23
System Architecture.....	23
Camera Subsystem.....	25
PRAM Daughterboard .....	27
4. POWER SYSTEM DEVELOPMENT .....	29
Introduction .....	29
Theory of Operation .....	30
Linear Regulators.....	30
Switching Regulators.....	30
Buck Regulator .....	31
Boost Regulator .....	32
Loop Feedback .....	32
Closed Loop Margining .....	33
Power Conversion Architecture .....	34
Power Supply Controller Devices .....	35
Voltage and Current Monitoring Circuits .....	36
Enable, Voltage Select and G-Switch Functions.....	38
Power Supply Sequencing:.....	38
Testing and Verification Results.....	39
Efficiency vs Load .....	39
Output Voltage vs Load.....	41

## TABLE OF CONTENTS – CONTINUED

Efficiency vs Input Voltage .....	43
Output Voltage Ripple.....	43
5. FPGA HARDWARE TILES.....	44
Introduction .....	44
Partially Reconfigurable Tiles .....	44
MicroBlaze Tile .....	45
Agnostic Hardware Accelerator Tiles.....	46
Floating Point Unit .....	47
Basic Linear Algebra Subprogram Accelerator.....	49
Camera Accelerator.....	54
Bitstream File Concatenation GUI.....	55
6. PARTIALLY RECONFIGURABLE OPERATING SYSTEM .....	57
Introduction .....	57
Control and Monitoring Operating System (ControlOS) .....	59
Program Objective .....	59
Power-On-Self-Test (POST) .....	59
Scheduling Algorithm .....	60
Radiation Sensor and TMR Voter Monitoring Functionality.....	62
Virtex-6 Reconfiguration Controller Interface .....	62
System Health Monitoring .....	64
Partially Reconfigurable Tile Operating System (prOSe) .....	64
Program Objective .....	64
Floating Point Unit Acceleration Functions.....	65
Basic Linear Algebra Subprograms (BLAS) Acceleration Functions .....	66
7. TEST APPLICATIONS AND RESULTS .....	67
LINPACK Benchmark.....	67
Background.....	67
Implementation .....	67
Results .....	67
8. RADIATION TOLERANT HARDWARE ANALYSIS.....	70
Error Rate Analysis .....	70
CREME96 Simulations .....	70
Computing Orbital Parameters .....	71

## TABLE OF CONTENTS – CONTINUED

Parametric Device Error Rate Analysis .....	72
Markov Reliability Analysis .....	73
Mathematical Model.....	73
Markov Chain with No Radiation Hardening.....	75
Markov Chain with TMR .....	75
Markov Chain with TMR and Scrubbing .....	76
Markov Chain with TMR, Scrubbing and Spares.....	77
Results .....	78
Orbital Fault Injection Simulator .....	85
<b>9. SYSTEM POWER AND PERFORMANCE ANALYSIS .....</b>	<b>86</b>
Introduction .....	86
Simplex System .....	87
Design Layout.....	87
Partial Reconfiguration Timing and Power Consumption .....	90
System Power Analysis .....	92
Performance Analysis .....	93
System Analysis .....	94
Power Consumption vs Fault Rate.....	96
MicroBlaze System Analysis with Accelerators and TMR .....	97
Design Layout.....	97
System Power Analysis .....	99
Performance Analysis .....	100
System Analysis .....	101
<b>10. CONCLUSION AND FUTURE WORK.....</b>	<b>104</b>
<b>REFERENCES CITED.....</b>	<b>107</b>
<b>APPENDICES .....</b>	<b>114</b>
APPENDIX A: TCM8240 Camera Initialization .....	115
APPENDIX B: Star Tracking Gyroscope .....	118
APPENDIX C: MATLAB Code for Markov Reliability Models .....	136

## LIST OF TABLES

Table	Page
3.1 Pinout between the Camera PCB and the FPGA .....	27
3.2 Pinout between the PRAM PCB and the FPGA .....	28
4.1 Power Supply Requirements.....	29
4.2 Required voltage regulators and usage .....	34
4.3 Digital Supply Configuration Parameters .....	36
4.4 Analog Supply Pin Configuration Parameters.....	36
4.5 Digital Power Supply Controller Voltage and Current Ratios.....	37
4.6 Analog Power Supply Controller Voltage and Current Ratios .....	37
4.7 Voltage regulation at the board output .....	41
4.8 Calculated regulation due to ripple voltage (regulation at the regulator circuit output) .....	43
5.1 MicroBlaze vs common CubeSat microprocessors .....	46
5.2 Implemented Double FPU Instructions .....	48
5.3 Theoretical FPU Accelerator Performance Limits .....	49
5.4 Implemented BLAS Level 1 Instructions .....	51
5.5 Theoretical BLAS Throughput.....	52
5.6 Theoretical elementwise vector throughput with bandwidth limited memory .....	52
5.7 Theoretical BLAS Throughput with Bandwidth Limited Memory.....	53
5.8 Theoretical Vector Operations with Bandwidth Limited Memory .....	53
6.1 Floating point instructions and their overloaded alternatives.....	65
6.2 Implemented BLAS Level 1 Functions .....	66
9.1 Power and Time for FPGA Configuration .....	91
9.2 Power Consumed to Perform Full Reconfiguration .....	91
9.3 Power and Time for FPGA Configuration .....	92
9.4 Power consumed by the tiles based on activity .....	92

## LIST OF TABLES – CONTINUED

Table	Page
9.5 Potential Speedup vs Tile Type.....	93
9.6 Speedup vs tile type with partial reconfiguration time .....	94
9.7 Power consumed by the tiles based on activity for the TMR system .....	100
9.8 Potential Speedup vs Tile Type for the TMR system.....	101
9.9 Speedup vs tile type with partial reconfiguration time for the TMR system .....	101
10.1 Performance comparison between the designed MicroBlaze based system, the RAD750 and common CubeSat processor boards .....	105
A.1 Toshiba TCM8240 JPEG Mode Initialization Sequence.....	116
A.2 Toshiba TCM8240 Bitmap Mode Initialization Sequence.....	117

## LIST OF FIGURES

Figure	Page
1.1 Comparison of the performance of commercial and radiation hardened microprocessors vs year of introduction [1] .....	2
1.2 Figure showing the approximate locations of the Van Allen radiation belts compared to common satellite orbit altitudes measured in earth radii.....	4
1.3 Cross section of charge paths that cause SEEs.....	6
1.4 Timing diagram comparison of the different types of SEEs.....	6
1.5 Shielding thickness required to shield devices in low earth orbit [2] .....	7
1.6 Comparison of a conventionally fabricated transistor with a SOI device [3] .....	7
1.7 Example of a radiation hardened by design inverter utilizing isolated transistors with guard rings [3].....	8
1.8 Example of a simple TMR system .....	9
1.9 Simplified FPGA logic and fault conditions.....	12
1.10 Levels of failure that can be caused by radiation events in an FPGA .....	13
2.1 Tiled Counter System with TMR+Spares+Scrubbing .....	15
2.2 Radiation Sensor - Cross section of an idealized radiation strike .....	16
2.3 Radiation Sensor - Two Dimensional Spatially Aware Sensor Layout .....	17
2.4 MSU's Proof-of-Concept Radiation Hardened FPGA System.....	18
2.5 Hardware stack mounted in a cyclotron beam .....	19
2.6 Research Balloon Launches .....	20
3.1 System architecture for the experimental FPGA system.....	24
3.2 The custom hardware stack developed for this research.....	25
3.3 The camera module designed for use on the hardware stack. ....	26
3.4 The camera module connected to the experiment board.....	26
3.5 Phase Ram Daughtercard designed for the hardware stack. ....	28



## LIST OF FIGURES – CONTINUED

Figure	Page
4.1 Simple Zener Regulator Circuit .....	30
4.2 Switching Power Supply Buck Regulator Circuit .....	31
4.3 Switching Power Supply Boost Regulator Circuit .....	32
4.4 Power Supply Feedback Circuit .....	33
4.5 Closed Loop Margining Circuit .....	33
4.6 Power Conversion Architecture for the switching power supply board .....	35
4.7 Fabricated Power Supply Board .....	39
4.8 Efficiency of the power supplies vs the output load .....	40
4.9 Output voltage of the power supplies vs the output load .....	42
5.1 Conventional FPGA Resource Requirements in time vs Partially Reconfiguration Resource utilization .....	45
5.2 MicroBlaze Tile Architecture .....	46
5.3 FPU Tile Architecture .....	47
5.4 FPU Tile State Machine .....	48
5.5 BLAS Tile Architecture .....	50
5.6 BLAS Tile State Machine .....	51
5.7 Camera Interface Tile Architecture .....	54
5.8 GUI for testing and configuring camera register settings .....	55
5.9 GUI for building SD Card images with multiple bitstreams .....	56
6.1 Functions performed by each operating systems, and the function name prefixes for the operations .....	58
6.2 Operating System Abstraction Levels .....	58
6.3 ControlOS Power on Self-Test (POST) .....	60
6.4 Output voltage of the power supplies vs the output load .....	60
6.5 Example Process Timing using the YEILD() Function .....	61
6.6 Example Process Timing using the WAIT() Function .....	61

## LIST OF FIGURES – CONTINUED

Figure	Page
6.7 GUI Interface for monitoring scheduler activity .....	62
6.8 GUI interface for monitoring tile activity .....	63
6.9 GUI interface for real time power monitoring .....	64
6.10 Decision Trees for FPU Overloaded Operations .....	65
6.11 Decision Trees for BLAS Overloaded Operations .....	66
7.1 LINPACK Results vs Matrix Size and Enabled Accelerator.....	68
7.2 LINPACK Results vs Matrix Size and Enabled Accelerator with and without data movement .....	69
8.1 Ideal Dipole Model for Earths Magnetic Field Lines and L-Shells .....	71
8.2 Impact of Solar Wind on Earths Magnetic Field Lines .....	72
8.3 Markov State Transition Diagram - Simplex System .....	75
8.4 Markov State Transition Diagram - TMR System without scrubbing .....	76
8.5 Markov State Transition Diagram - TMR System with scrubbing.....	76
8.6 Markov State Transition Diagram - TMR with Spares and Scrubbing .....	79
8.7 Mean time between failures for the system in a LEO orbit during average solar conditions.....	80
8.8 Mean time between failures for the system passing through the SAA in a LEO orbit during average solar conditions.....	80
8.9 Mean time between failures for the system in the worst orbital section during the worst week of a solar maximum in an ISS orbit.....	81
8.10 Mean time between failures for the system in the worst orbital section during the worst 5 minutes of a solar flare in an ISS orbit.....	82
8.11 Simulation of the MTBF for the different systems studied vs fault rate .....	83
8.12 Comparison of system reliability vs fault rate.....	84
8.13 Simulation showing estimated fault rate vs reliability.....	84

## LIST OF FIGURES – CONTINUED

Figure	Page
8.14 Orbital Radiation Fault Injection GUI with radiation sensors for a nine tile MicroBlaze System .....	85
9.1 Expected power consumption with only a microprocessor.....	86
9.2 Expected power consumption with only a partially reconfigurable hardware accelerator.....	87
9.3 Block Diagram of a partially reconfigurable system with hardware accelerators .....	88
9.4 Floorplan of a partially reconfigurable system with hardware accelerators implemented on the Virtex-6 FPGA.....	88
9.5 Measured power consumption showing power consumed by the PR operation and power saving by eliminating unnecessary acceleration hardware .....	89
9.6 Simulation of the power consumption showing power consumed by the PR operation and power saved by eliminating unnecessary acceleration hardware .....	90
9.7 Speedup achieved vs the MicroBlaze using accelerators based on number of floating point operations with partial reconfiguration time .....	94
9.8 Power consumed for an operation on the simplex system.....	95
9.9 Power consumed for an operation normalized to the MicroBlaze processor time without the microprocessor idle power during PR counted in the total operation power consumption.....	95
9.10 Power consumed for an operation normalized to the MicroBlaze processor .....	96
9.11 Power efficiency for the TMR system vs the size of the operation in MFLOPS .....	97
9.12 Power consumed by the system based on partial reconfiguration rates.....	97
9.13 Block Diagram of a partially reconfigurable TMR system with hardware accelerators .....	98

## LIST OF FIGURES – CONTINUED

Figure	Page
9.14 Floorplan of a TMR MicroBlaze partially reconfigurable system implemented on the Virtex-6 FPGA .....	98
9.15 Measured power consumption showing power consumed by the PR operation and power saving by eliminating unnecessary acceleration hardware on a TMR system.....	99
9.16 Simulation of the power consumption showing power consumed by the PR operation and power saving by eliminating unnecessary acceleration hardware in a triplicated system .....	100
9.17 Speedup over the MicroBlaze using accelerators based on number of floating point operations with partial reconfiguration time for the TMR system.....	102
9.18 Total power consumed for an operation vs the acceleration hardware available on the TMR system .....	102
9.19 Power consumed for an operation normalized to the MicroBlaze processor for the TMR System .....	103
9.20 Power efficiency for the TMR system vs the size of the operation in MFLOPS .....	103
10.1 Solidworks Rendering of preliminary 3U Cubesat Design.....	106
B.1 Startracker procedure vs accelerator type.....	119
B.2 Star tracker showing a mis-centered center of rotation .....	122
B.3 Star tracking algorithm with displacements to normalize the rotation around the z axis.....	123
B.4 MATLAB Simulated Star Tracking Algorithm.....	125
B.5 MATLAB Simulated Star Tracker roll angle after Kalman filtering.....	125

## ABSTRACT

Computers play an important role in spaceflight and with ever more complex mission goals and sensors, current devices are not sufficient to meet the requirements of planned missions. These challenges are complicated by memory corruption caused by high energy radiation inherent in the space environment. We propose the use of commercial field programmable gate arrays using partial reconfiguration, triple modular redundancy with spares and memory scrubbing to achieve a radiation hard, high performance system. This strategy is leveraged on modern fabrication process nodes largely eliminating long term effects of radiation on silicon devices and shifting the focus strictly on memory corruption errors. This dissertation improves the performance of Montana State University's (MSU) existing CubeSat computing research platform through the addition of hardware accelerator tiles, a reliability analysis and analysis of the power consumption vs performance tradeoffs allowing for the development of a metric for the use of accelerator functions.

## MOTIVATION AND BACKGROUND

### Current and Future Space Computing Requirements

One of NASA's research areas in recent years has been the study of improving both the radiation hardness of computers and computational efficiency [4–7]. In the most recent decadal survey, NASA projects that by 2020 missions will require computers capable of achieving 2000MIPS and 200MIPS/Watt to meet mission requirements. This far exceeds the current state-of-the-art processors, which range from 10s to 100s of MIPS and consume several watts. With the prevalence of high resolution sensors and increased reliance on small satellites, these power efficiencies and computational speeds will be required to meet mission goals [8].

NASA has been plagued by radiation issues on spacecraft since its earliest days [9, 10]. In more recent years, modern devices have continued to experience difficulties from high energy radiation effects [11]. Most recently the Mars Curiosity Rover experienced a memory corruption event requiring the spacecraft to utilize a radiation hardened fallback routine to recover from the radiation strike [12, 13]. This event and others like it emphasize the continuing requirement to study and develop techniques to manage these radiation effects.

Commercial devices are fully capable of reaching performance and power goals. Radiation tolerant microprocessors, however, are more limited due to the specialized fabrication procedures and limited demand. These factors have led to a lag in the performance of radiation tolerant systems versus commercial-off-the-shelf (COTS) variants by approximately a decade (Figure 1.1). As such, early 2000s era computers (Power PC 750s and Pentium equivalents) running at several hundred MHz are considered current high-end processors for satellite applications [14–16].

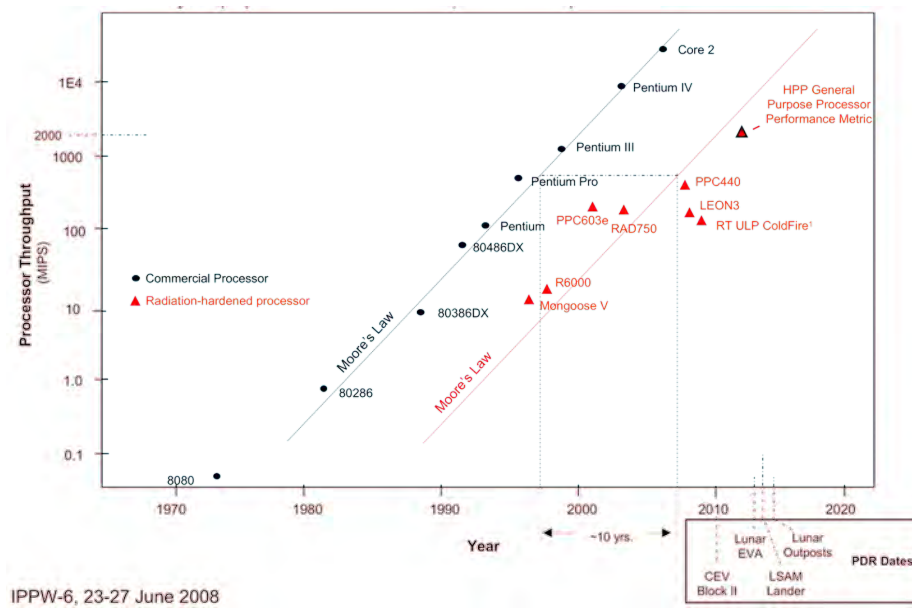


Figure 1.1: Comparison of the performance of commercial and radiation hardened microprocessors vs year of introduction [1]

An additional consideration when developing computer systems is cost. Towards this goal, COTS components are desirable over radiation hardened devices. These components benefit from the economy of scale, while utilizing cutting edge fabrication techniques. Modern fabrication processes also have a side effect of improved immunity to long term degradation due to radiation.

### Radiation Effects on Semiconductor-Based Electronics

The need for specialized radiation hardened computers is predicated on the abundance of ionizing radiation when outside the protection of Earth's magnetic fields and the atmosphere. Ionizing radiation is classified as radiation having sufficient energy to break electrons free from their chemical bonds. As this ionizing effect can

be achieved through numerous physical processes, radiation is categorized based on the particles and processes involved [17–20].

Alpha particles are the nuclei of helium atoms accelerated to relativistic speeds. These particles make up around 10% of the space radiation environment, but can be stopped with a piece of paper making shielding trivial.

Beta particles are electrons accelerated to relativistic speeds and are only slightly more difficult to shield, generally metal foil is sufficient and is inherently present in a spacecraft's structure.

Gamma radiation is an electromagnetic field effect and much more difficult to stop with shielding. It generally lacks sufficient energy transfer to affect digital hardware, making it of little concern to spacecraft electronics.

Neutrons are uncharged subatomic particles traveling at relativistic speeds. Due to their lack of charge they are not deflected by magnetic fields and are also difficult to shield [7]. Neutrons vary from charged particles as they create charge paths through kinetic effects and can cause issues on silicon devices.

Cosmic radiation is a combination of primarily protons and a smaller quantity of heavy ions accelerated to relativistic speeds with sufficient energy to ionize silicon devices. These are the particles which are of concern to space electronics.

In Earth orbit, charged particles are trapped in greater concentrations in the Van Allen belts. The innermost of these belts consists primarily of protons, while the outer belt consists of electrons. These belts of radiation are of interest for spaceflight computers as they must be transited to achieve high orbits (Figure 1.2) [17].

Earth's magnetic field deviates from the ideal dipole model in several parameters which have a significant impact on the radiation environment of orbiting spacecraft. The combination of the magnetic poles being rotated approximately  $11^\circ$  from the rotational axis and the magnetic center being offset from the center of the planet



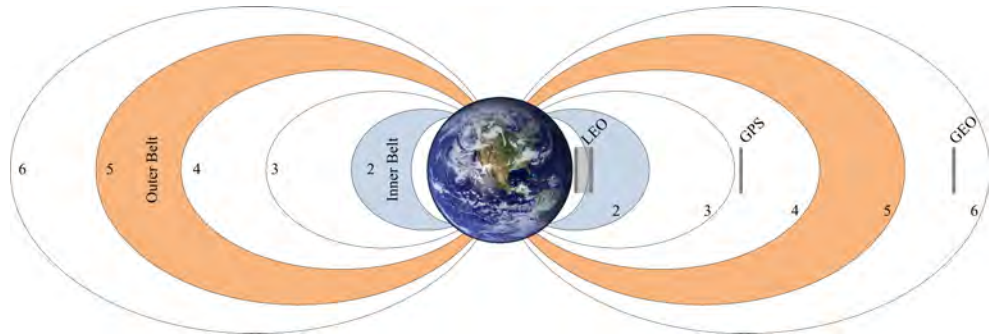


Figure 1.2: Figure showing the approximate locations of the Van Allen radiation belts compared to common satellite orbit altitudes measured in earth radii

towards the north pole result in the magnetic fields passing closer to the surface in the vicinity of South America. The result is greatly increased radiation for satellites in this portion of the globe and is referred to as the South Atlantic Anomaly (SAA).

Ionizing radiation effects for semiconductors fall into two distinct categories, long term damage to the semiconductor device and the immediate effects of a strike. Total Ionizing Dose (TID) degradation is a permanent long term effect of semiconductor devices operating in radiation rich environments. The cause is low energy particles impacting the device and remaining trapped in the semiconductor structures, modifying the material properties. The primary cause of failure is charge being trapped in the oxide layers, leading to oxide breakdown and leakage currents in either in the gate or field oxide. This causes increased power consumption and skews the threshold voltage resulting in timing failures (Figure 1.3) [18, 21]. TID degradation is generally non-repairable, though its effect may be reduced through annealing the component and reducing design level timing requirements.

The second major category of radiation effects is the Single Event Effects (SEEs). These are the result of a single strike on the device and can be repaired by restarting the system or by repairing corrupted memory. These errors are the result of a

radiation particle impacting a semiconductor device and creating a path of holes and electrons within the silicon substrate in a sensitive region (Figure 1.3). This can result in system glitches or memory devices changing states during system runtime, thereby corrupting data. The classification of these errors is further divided into more detailed sub-categories depending on the effect the strike causes (Figure 1.4).

Single event transients (SETs) are the least damaging of these errors and result in a momentary glitch in the device. As digital logic circuits generally process data on clock edges, these can often be ignored as they do not permanently corrupt memory. The exception is if they occur during a setup and hold period or on a clock edge, at which point they can manifest as single event upsets.

Single Event Upsets (SEUs) are of concern as they result in memory corruption. For an SEU to occur, either a SET occurs on a clock edge and is latched into memory or the memory itself is struck in a sensitive region causing it to change values. If an SEU occurs in the device's base functionality and halts the system operation, it is classified as a Single Event Function Interrupt (SEFI).

The most serious form of SEE is a Single Event Latchup (SEL). For this to occur, two transistors are struck in a manner that creates a short between power and ground. The resulting short is self-sustaining and can only be halted with a system restart. The current draw associated with the SEL can cause permanent device damage [1,22,23].

A comparison of the strike and the effect on timing of these different SEEs is shown in Figure 1.3 and 1.4.

### Conventional Radiation Effects Mitigation Techniques

The most obvious solution to radiation harden a system is to shield the electronic components. This is effective in protecting against alpha and beta particles where the

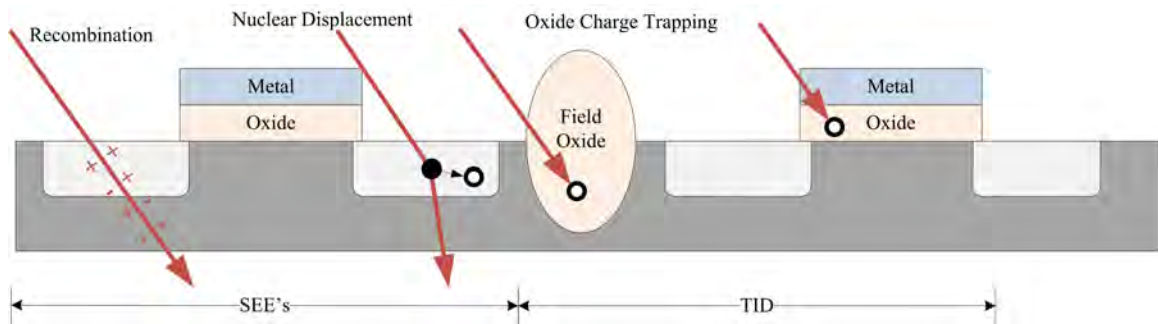


Figure 1.3: Cross section of charge paths that cause SEEs

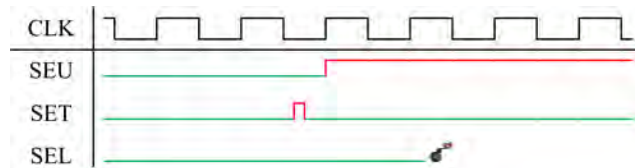


Figure 1.4: Timing diagram comparison of the different types of SEEs

shielding requirement is minimal and inherently included in the spacecraft structure. Shielding against gamma particles, neutrons and heavy ions is more difficult, requiring thick radiation shields. For the highest energy particles, shielding cannot be made effective and in this situation the shielding itself can cause multiple bit upsets to occur as shielding material creates secondary radiation (Figure 1.5).

The lowest level of silicon device hardening is Radiation Hardening by Process (RHBP). This involves specialized fabrication processes that reduce the influence of radiation on the semiconductor device. Included in these techniques are process changes such as Silicon on Insulator (SOI) or Silicon on Sapphire (SOS) transistors. In these processes transistors are fabricated on a thin layer of silicon deposited over an insulator. This reduction of silicon minimizes the length of the charge path and reduces the effect of the strike (Figure 1.6) [21, 24].

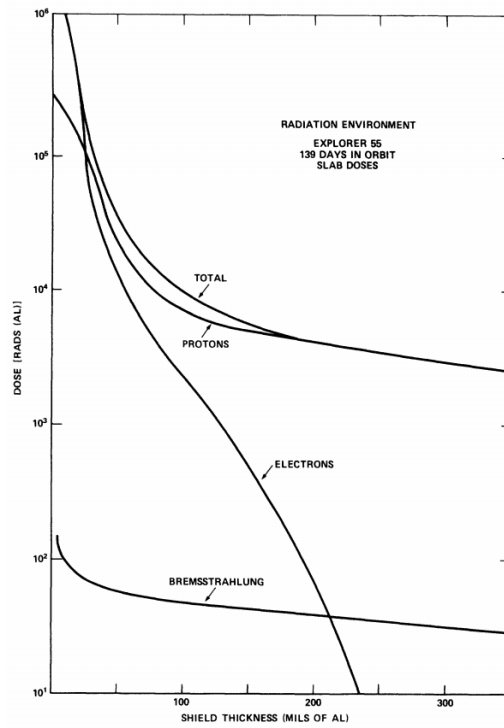


Figure 1.5: Shielding thickness required to shield devices in low earth orbit [2]

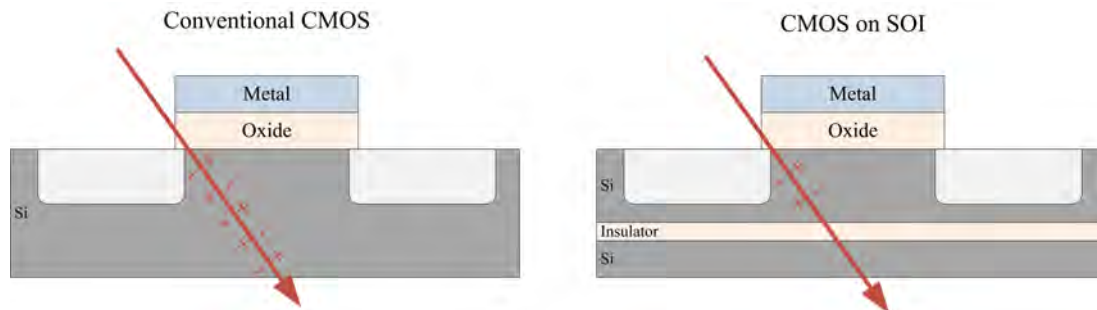


Figure 1.6: Comparison of a conventionally fabricated transistor with a SOI device [3]

Radiation Hardened by Design (RHBD) techniques include semiconductor layout strategies to reduce the effect of radiation on semiconductor devices. These strategies use larger transistors or BJTs, which are less susceptible to SEUs, and isolation of transistors with guard rings (Figure 1.7). These techniques can also be used to make devices latchup immune and eliminate the possibility of an SEE causing a destructive failure.

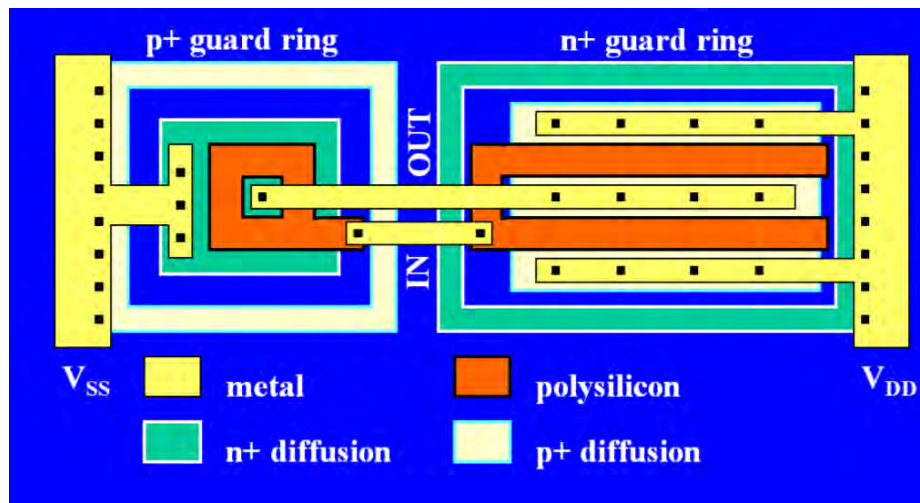


Figure 1.7: Example of a radiation hardened by design inverter utilizing isolated transistors with guard rings [3]

Radiation Hardened by Architecture (RHBA) is a third category of methods for radiation hardening. This includes Triple Modular Redundancy (TMR) for improving radiation tolerance of less reliable devices through use of redundant hardware [25,26]. By running two systems in parallel a fault in the system can be detected but the correct output cannot be determined. By including a third system (TMR) and assuming no multiple bit upsets, the correct result can be determined with a voting circuit and the system can be allowed to continue operating with valid outputs (Figure 1.8). Once a checkpoint is reached, the system can repair the error and resume

normal TMR operation. The space shuttle computer is an example of this type of system, utilizing multiple microprocessors running in parallel to validate computed results [27–29].

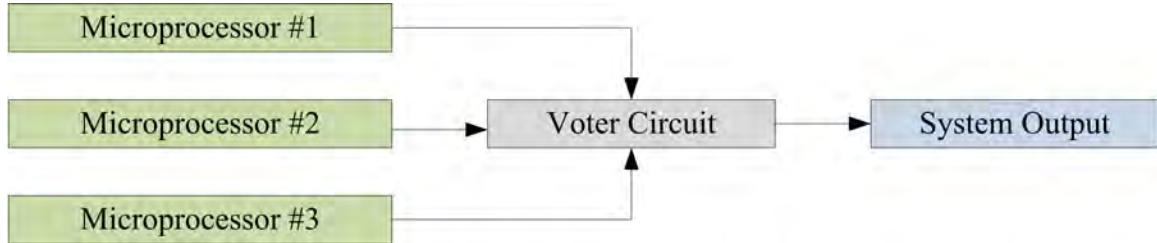


Figure 1.8: Example of a simple TMR system

### Problems with Conventional Radiation Effect Mitigation Strategies

All radiation effect mitigation methods have tradeoffs. Shielding is heavy and only effective against certain particles. The fabrication and design level RHBP/RHBD procedures are costly due to the low numbers fabricated. This cost for the currently available RAD6000 and RAD750 devices amounts to around \$200,000 per flight board, which exceeds the price of their commercially equivalent parts by several orders of magnitude. In addition, radiation hardened processes are generally based on older fabrication nodes, which are generally slower and less power efficient than their state-of-the-art commercial counterparts [1].

TMR can be made to have high performance characteristics, but brings additional issues from the triplication of hardware. TMR increases the sensitive areas of the design by 300%, as well as the overall fault rate, but has the advantage that when faults occur they can be detected and corrected. The redundant hardware also has far greater power consumption and voting circuits leave a small area where a radiation error will still result in a system failure.

The current drive to improve radiation hardening and performance in spaceflight computers by using COTS parts is driven by an effort to reduce cost and take advantage of the high performance commercial offerings. In particular, the interest in using COTS field programmable gate arrays is due to their ability to process large amounts of data and the ability to be modified in flight based on the phase of the mission or running experiment.

### Commercial Off-The-Shelf FPGAs as a Solution

Field Programmable Gate Arrays (FPGAs) are programmable logic devices that can be viewed as a programmable application specific integrated circuit (ASIC) device. This is achieved by having programmable interconnects and logic blocks, which allows the hardware to be modified with software-like design times [30]. With this flexibility, the circuitry of the FPGA can be configured to meet the application requirements using soft processors and custom parallelized hardware. With this, near ASIC level performance can be achieved with design times and engineering costs more similar to that of a software project.

Another advantage of modern FPGAs is the ability to utilize Partial Reconfiguration (PR). This allows for small blocks on the FPGAs to be reconfigured while the device is operating. This improves resource utilization, as hardware can be multiplexed temporally. It can also improve timing by reducing interconnect lengths and reduce power by eliminating temporally unnecessary circuitry [31]. This is also advantageous for repairing radiation faults in the configuration RAM as adjacent sections of the FPGA can be allowed to continue operating while the fault is repaired, thus improving system availability [32].

The use of high-end commercial FPGAs brings further advantages to space computer platforms. As total ionizing dose degradation is due to oxide charge trapping, a device's susceptibility to this effect can be reduced through the use of a thinner gate oxide layer which is inherent in modern fabrication nodes. Tests have shown that commercial 32nm non-radiation hardened processors can exceed many of the metrics for previous generations of radiation hardened processors. This feature size compares favorably with the 40nm process of the Xilinx Virtex-6 and the 28nm process of the current Virtex-7 device families. In the case of Virtex-6 FPGAs, accelerated tests have been performed at up to 2 MRAD doses with minimal timing performance loss. This small feature size does, however, increase the probability of SEEs, as the small size of the diffusion region decreases the charge required to cause a data error [4, 29, 33, 34].

### Problems with FPGAs in Radiation Environments

Modern FPGAs may be largely immune to TID, but they are uniquely susceptible to SEEs. This is due to the presence of an additional large block of configuration RAM used to store the device's logic element configurations and interconnect paths. An error in this region has the effect of re-wiring or changing the functionality of the device and making the underlying circuitry not operate as desired. FPGAs, like all other semiconductor microprocessors, are also susceptible to SEEs in data and program memory (Figure 1.9).

All of these sources of system errors must be managed in a radiation tolerant FPGA system. Errors in configuration memory must be repaired through partial or full reconfiguration. A SEE in the program memory or registers can cause the program to crash and require reloading the software program and errors in the data



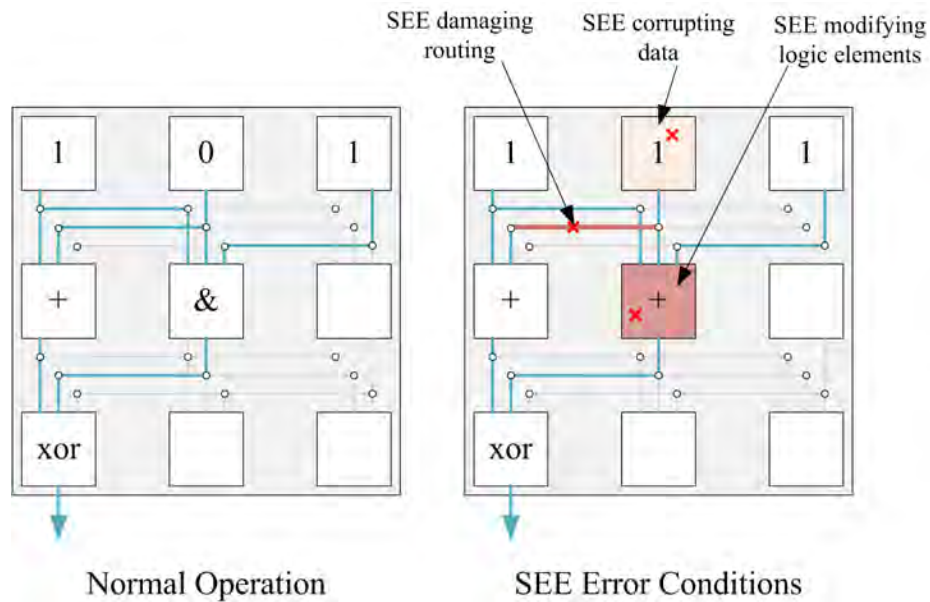


Figure 1.9: Simplified FPGA logic and fault conditions

memory can cause corrupted data to be output. FPGAs are only unique in that the FPGA fabric adds the uncertainty of the configuration of the FPGA fabric. These errors are of differing levels of severity and result in different amounts of data needing to be discarded to bring the system back to a known good state (Figure 1.10). For example, an error in the FPGA fabric could cause data corruption and the running program to fail, necessitating both those memory blocks be repaired along with the configuration memory; whereas an error in data would only require that good data be loaded back into the system.

To manage the possibility of data and program errors, TMR is the most prevalent and efficient method. By comparing three copies of the hardware, errors can be detected and repaired by copying known good data to a spare processor tile and allowing the system to resume operation.

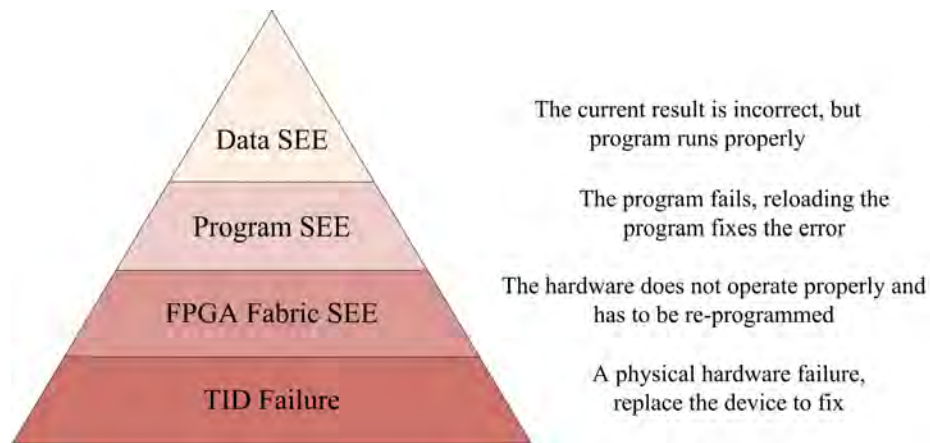


Figure 1.10: Levels of failure that can be caused by radiation events in an FPGA

For FPGA Configuration RAM, TMR may be effective, but other techniques can potentially detect these errors more rapidly and efficiently. Blind Scrubbing involves partially reconfiguring the unused portions of the FPGA to overwrite SEEs before the logic is brought online. Readback scrubbing is more efficient by reading the bitstream out of configuration memory and comparing it to a golden copy to detect errors. Only if an error is detected is the FPGA reconfigured to repair the error. Also, since readback does not overwrite active logic regions, readback scrubbing can be run on the entire configuration memory region [29, 35, 36].

Routing data between tiles must also be protected to avoid a single point of failure. This proves to complicate the design as the routing is not enclosed within a PR tile but spread across the FPGA. This research uses readback scrubbing or full reconfiguration to detect and repair these errors.

## MSU'S APPROACH TO FPGA RADIATION TOLERANCE

TMR with Scrubbing and Spares

The Many-Tile TMR+Scrubbing+Spares FPGA architecture is the base system for MSU's radiation tolerant computing research. In this architecture the control system chooses three partially reconfigurable tiles to run in a TMR configuration to provide fault tolerance. In the case of a TMR error, the system is halted, the faulted processor is taken offline while valid data is copied to a spare processor and the system is allowed to continue. The faulted tile is then repaired in the background and added back into the pool of spares. By switching to a new tile rather than simply repairing the damage, the system increases availability as the time to copy the data and start a new tile is faster than repairing the underlying fault.

Multiple systems based on this architecture have been developed. The simplest was a counter system of 256 simple VHDL counters. This system proved the system scalability and basic functionality of the architecture.

A second system consisted of 64 PicoBlaze processor tiles. This system allowed for processors to be implemented without the memory complexities of the MicroBlaze core.

Finally, a 16 tile MicroBlaze processor system was developed. This system proved the most difficult due to the complexity of peripheral initialization and copying existing data to a new processor core when a fault occurred.

In addition to running TMR, the system scrubs the configuration memory in the background to ensure the spare MicroBlaze tiles kept as spares are clean. This reduces the probability that a faulted tile will be brought online and immediately fail [37].

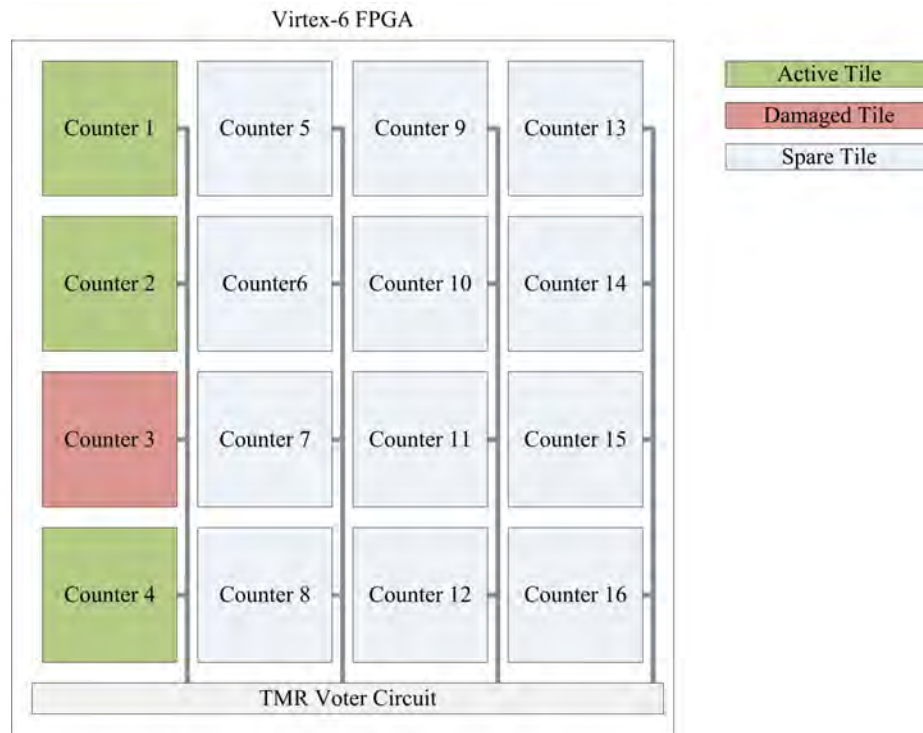


Figure 2.1: Tiled Counter System with TMR+Spares+Scrubbing

### Silicon Based Spatially Aware Radiation Sensor

In addition to the TMR+Scrubbing+Spares architecture, Montana State University developed a two-dimensional silicon based radiation sensor to provide environmental feedback for the system. Through knowledge of the spatial location of a radiation strike, a SEE can potentially be discovered prior to triggering faults, as well as provide knowledge as to the radiation conditions in which the system is operating.

The sensor is created with P type regions on the upper surface of the silicon, and N type regions on the bottom of the sensor creating a PN junction. When a radiation strike occurs, it then leaves a charge path of ionized particles in its path. If not

allowed to recombine in pace, these ions are attracted to opposite sides of the silicon region and result in a transient current [38].

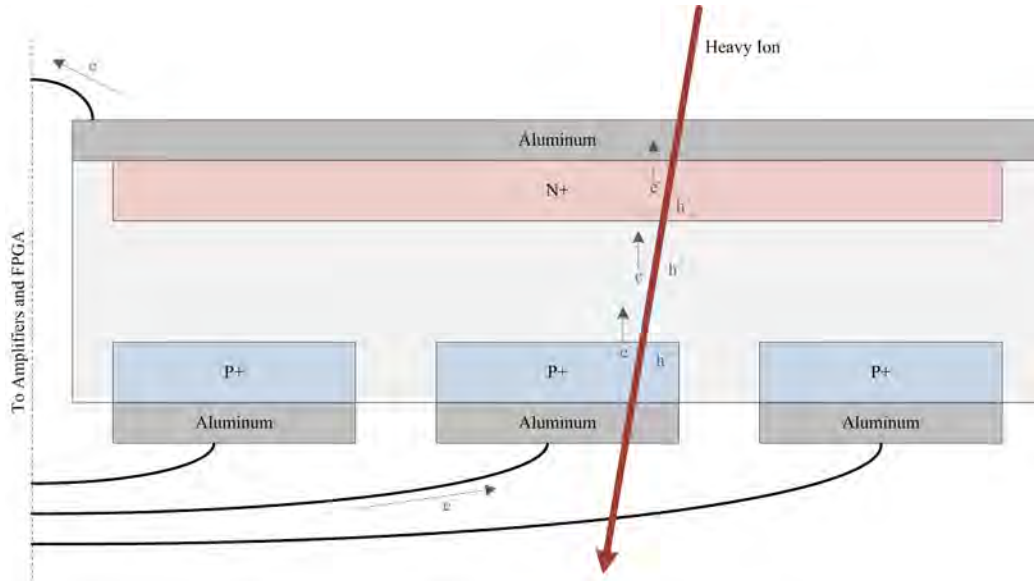


Figure 2.2: Radiation Sensor - Cross section of an idealized radiation strike

To increase the sensitivity of the sensor the PN junction is reverse biased, widening the depletion region. When fully depleted, as is ideal for a silicon based radiation sensor, the bias voltage is sufficient to increase the depletion region to its maximum width, minimizing recombination and maximizing sensor sensitivity.

The output current from the sensor is too small and short to be detected by the FPGA. To allow for the FPGA to detect these signals, an amplifier chain and pulse stretching circuit was designed to convert the signals to a 3.3V TTL signal of sufficient duration for FPGA sampling [39].

To create a two dimensional sensor and assuming single radiation strikes, the doped regions are created in strips on the top and bottom of the sensor (Figure 2.3). A strike that fully passes through the sensor then provides the X location of the strike

as a current on the top strip, and the bottom sensor returns the Y location of the strike [38, 39].

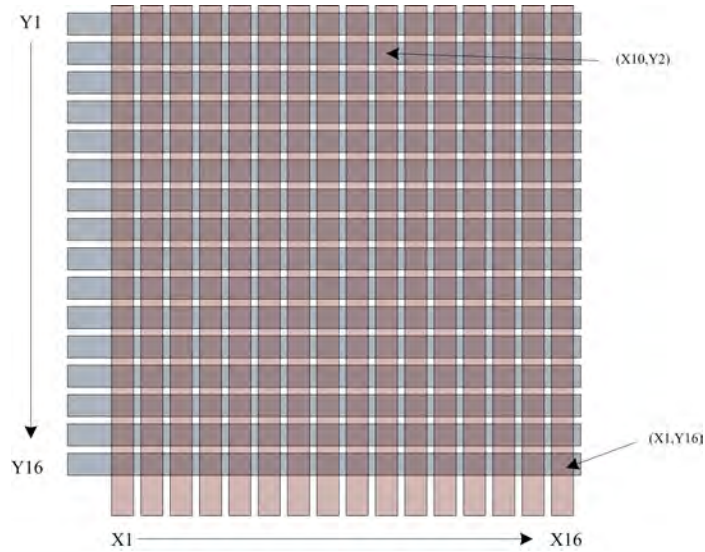


Figure 2.3: Radiation Sensor - Two Dimensional Spatially Aware Sensor Layout

This sensor could then be sampled by the FPGA and top and bottom detections correlated in time to determine intersections when applicable [39].

### TMR+Scrubbing+Spares with Radiation Sensor Feedback System

The architecture was implemented and tested using a Xilinx ML605 evaluation board using a single Virtex-6 FPGA. This system proved the viability of the architecture [37], as well as to establish the system's interaction with a silicon based radiation sensor [39].

While this was valuable for initial testing, the size of the evaluation hardware and associated power supplies did not permit for this system to be used for in-situ testing.

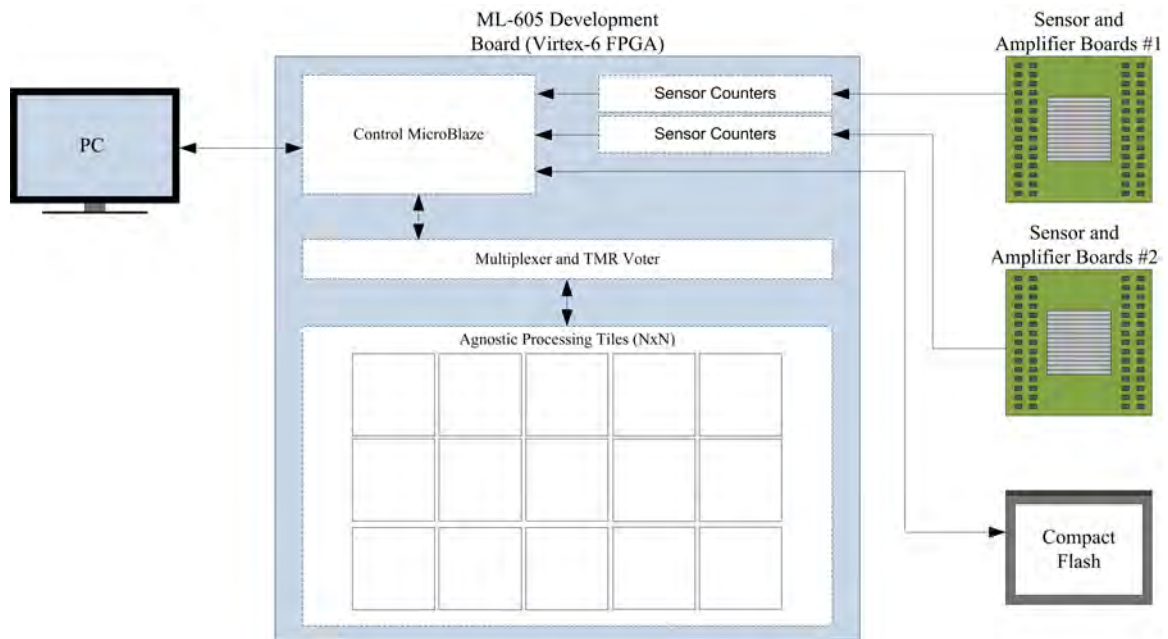


Figure 2.4: MSU's Proof-of-Concept Radiation Hardened FPGA System

Also, with a single FPGA, the system lacked dedicated control hardware which may have reduced the hardness of the system in a radiation environment.

### System and Sensor Testing

During the course of this research, the system has been tested using numerous platforms including cyclotron testing, high altitude balloon flights and is being prepared for a sounding rocket test flight. These tests have been used to validate the operation of the radiation sensor and the ability of the payload to operate under the constraints of a zero pressure environment.

Cyclotron testing of the system was performed at the Texas Cyclotron Facility using 25 MeV Krypton and Argon beams. This testing utilized a custom aperture that allowed for the beam to be isolated to a specific portion of the sensor and test



the sensor's ability to detect radiation and the FPGA system's ability to respond to the simulated radiation damage.

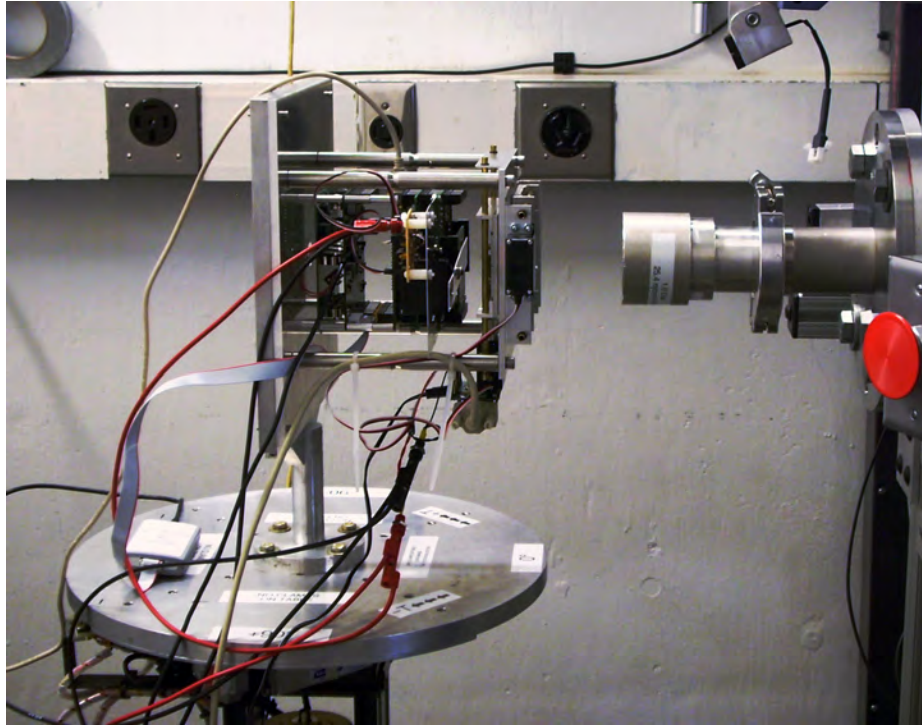


Figure 2.5: Hardware stack mounted in a cyclotron beam

Montana Space Grant Consortium Borealis research balloons were utilized in 2012 and 2013 to test the thermal properties and design of the payload in preparation for longer duration, zero pressure research balloon flights. These balloons verified the groups design of an enclosure, thermal simulations and the basic system functionality in a realistic environment. Radiation was logged on these missions, but due to limited flight time and altitude only a handful of potential strikes were identified.

The Borealis balloon flights were followed each year by a flight on NASA/LSU HASP zero pressure balloons. These balloons verified system performance and duration testing on 10-14hr float times at 120,000 ft. Several potential radiation



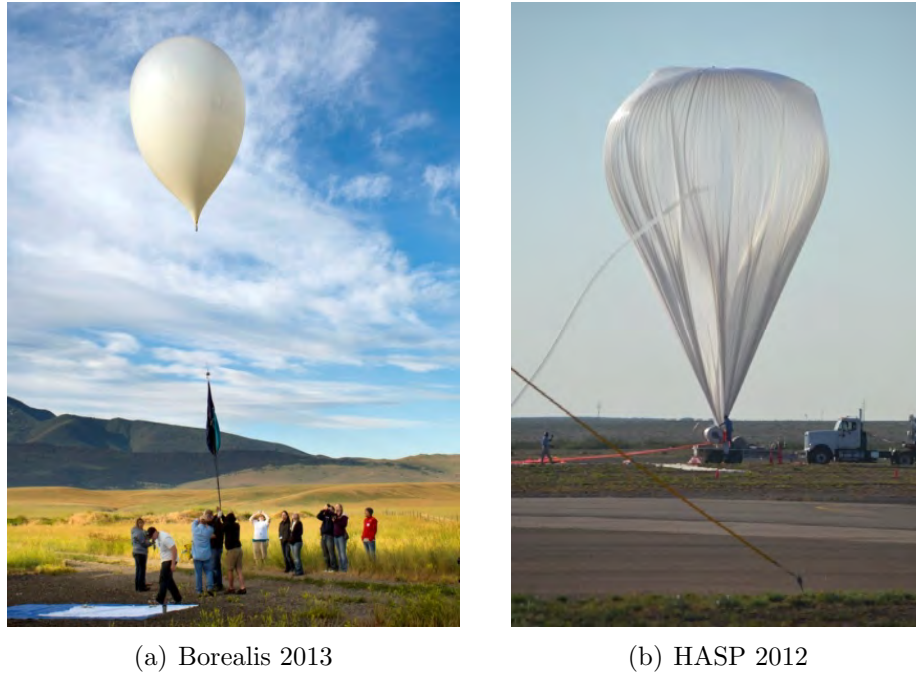


Figure 2.6: Research Balloon Launches

strikes were gathered during these flights due to the extended time spent at the float altitude. The 2012 flight results were impeded by an insufficient power system, where radiation strikes caused overall system crashes, while the 2013 flight was able to detect and record several promising radiation sensor strikes. The FPGA system did not experience any radiation induced faults due to the low probability of such strikes in the upper atmosphere being much lower than would be experienced in orbit.

These tests showed that the overall system was capable of detecting radiation and operating in a low pressure environment with the temperatures inherent in the near-space environment.

### Next Steps

The goal of this research is to develop, test and analyze partial reconfiguration as a technique for facilitating higher performance computers in radiation prone environments. This research will reduce the cost while increasing the performance of radiation hardened computers over the currently accepted systems.

The system hardware for a nearly CubeSat formfactor research platform was developed and tested. My contribution to this system was in the design of a power system capable of supplying the voltages required for the hardware stack as well as providing for real-time monitoring of system parameters.

Once the hardware system was present, partially reconfigurable regions were designed and tested to accelerate tasks. This included microprocessors, floating point accelerators, and hardware interface tiles.

To ease software development, an operating system was designed for the hardware stack. This improved the ability to monitor the system and facilitated partial reconfiguration tasks.

Once the system was operational it was tested with benchmarking routines and a star tracking gyroscope to evaluate the performance of the system with realistic workloads.

The system was analyzed analytically using Markov chains to determine how the design would react to different radiation environments. This testing showed the system is capable of operating reliably and with high availability under a variety of solar conditions and orbital locations.

Finally, an analysis of the system was performed to determine the tradeoff between power, performance and time for the use of accelerators in partially reconfigurable

FPGA designs. This work shows that partial reconfiguration can be useful to increase system performance as long as the task consumes sufficient time.

## DIGITAL HARDWARE DEVELOPMENT

### System Architecture

The research performed in this project was conducted on a custom CubeSat formfactor FPGA system designed at MSU. The platform consists of two silicon radiation sensors with amplifiers, a FPGA board with a Xilinx Spartan-6 75k, a Virtex-6 75k device and a power board to supply the system voltages. In addition, a camera board and battery board were developed to facilitate demonstration and test applications (Figure 3.1).

The system was built with a series of 4 inch square printed circuit boards connected by 100 mil headers. The radiation sensors are located at the top of the stack followed by the FPGA board, the optional experiment board, and the power supply occupies the bottom. If standalone power is required, a battery board is placed under the power supply (Figure 3.2).

This digital system is based around two FPGAs. A Spartan-6 operates as a control FPGA. This FPGA communicates with the SD card, configures the Virtex-6 on startup and manages the partial reconfiguration operations. It is also responsible for communication with the radiation sensors and the PC, which can be achieved through numerous UARTs. Lastly, the Spartan-6 monitors temperature and power status for the stack. Radiation hardness is achieved through the Xilinx Soft Error Mitigation (SEM) controller.

The Virtex-6 FPGA is the compute fabric for the system. This FPGA contains higher performance MicroBlaze soft core processors and hardware acceleration tiles. The results computed by this FPGA are output to the Spartan-6 for TMR voting. This FPGA also contains a 16 bit GPIO bus to interface with experiment hardware.

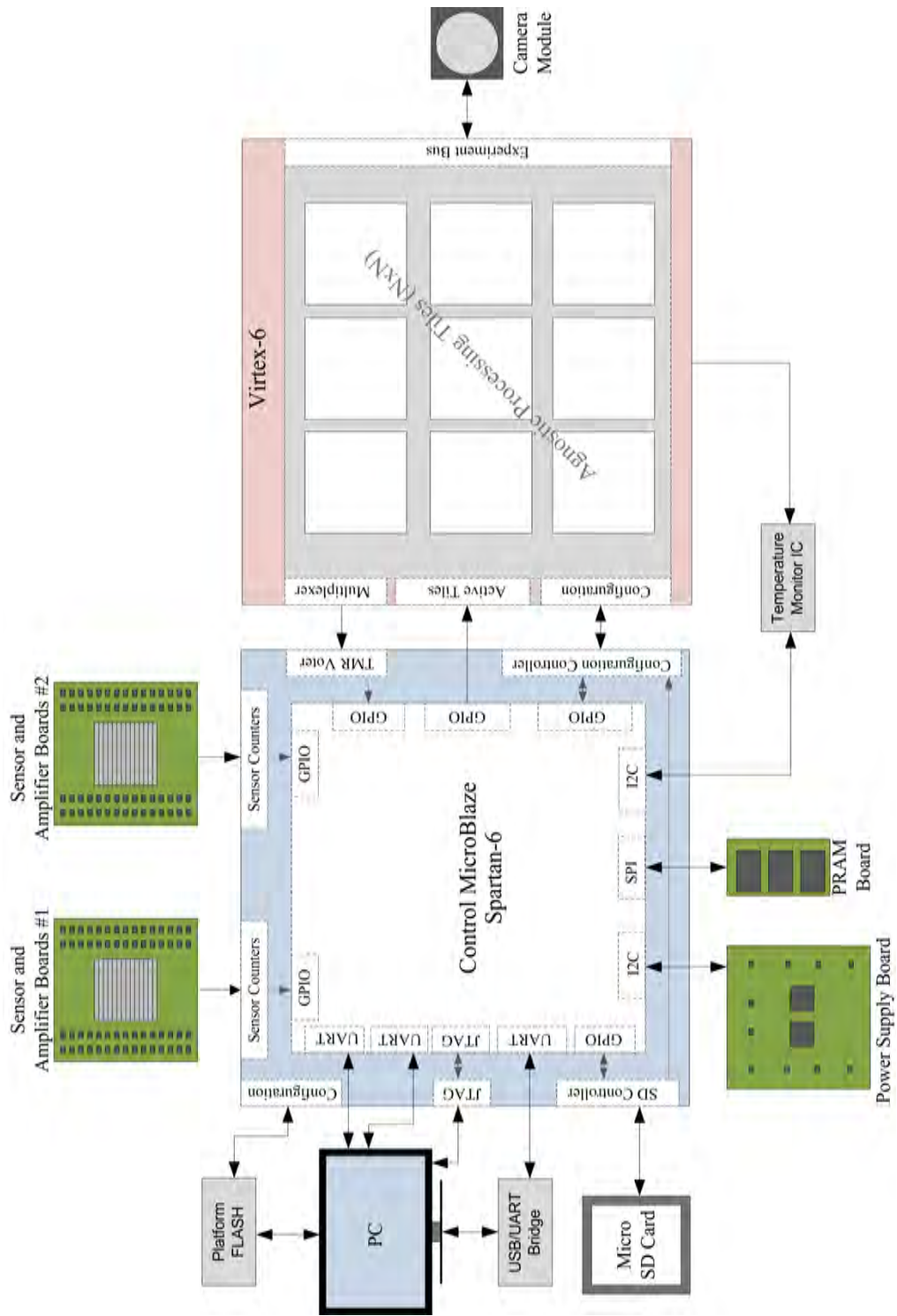


Figure 3.1: System architecture for the experimental FPGA system



Figure 3.2: The custom hardware stack developed for this research

The design of the FPGA board is discussed in detail in [40].

### Camera Subsystem

To develop imaging applications, a TCM8240 camera board was designed for the system. This allowed the system to be used for imaging applications. The camera module can collect images up to 1.3MP and could be configured to return either JPG or BMP images.

The camera is connected to the camera interface board through a header for power and flying leads to the experiment bus for data. This consists of an I2C bus (2 wire) for camera setup and control as well as an 8 bit parallel bus with clock and frame synchronization signals. The camera also requires a clock signal for internal logic. Power supply requirements are both 1.8V and 2.5V rails [41].

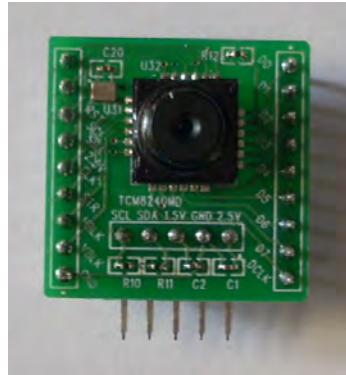


Figure 3.3: The camera module designed for use on the hardware stack.

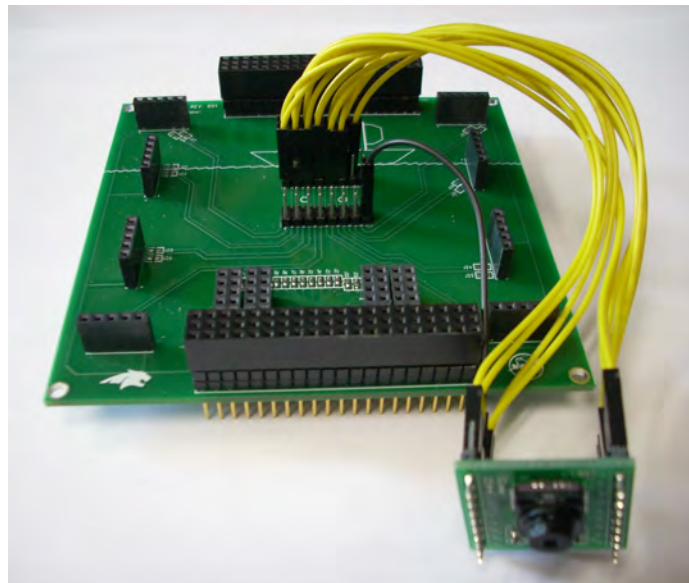


Figure 3.4: The camera module connected to the experiment board

The pinout between the connector and the FPGA was arbitrarily defined in Table 3.1:

Table 3.1: Pinout between the Camera PCB and the FPGA

Experiment Bus Pin	V6 Pin	Camera Pin	Function
1	C7	SCL	I2C Clock
2	D7	SDA	I2C Data
3	E7	DOUT0	Parallel Data Bus
4	C8	DOUT1	
5	D8	DOUT2	
6	D9	DOUT3	
7	A9	DOUT4	
8	B8	DOUT5	
9	B10	DOUT6	
10	C10	DOUT7	
11	A11	HBLK	Horizontal Blank Strobe
12	B11	VBLK	Vertical Blank Strobe
13	C11	DCLK	Data Clock
14	D10	RST	Camera Reset
15	F9		
16	G9	CLK	Clock Input

### PRAM Daughterboard

Initial development of the camera module showed that the Virtex-6 did not contain sufficient RAM to capture full resolution bitmap images, limiting its use. In an attempt to resolve this issue, a phase RAM (PRAM) daughterboard was developed to add additional memory to the system [42]. This daughterboard contains three 16MB PRAM modules and communication is achieved over a SPI communication bus.

Data for this card was passed from the Virtex-6 through unused pins on the Spartan-6 due to I/O limitations. The module can be used in different jumper





Figure 3.5: Phase Ram Daughtercard designed for the hardware stack.

selectable modes allowing either all the banks to share a SPI bus with different chip selects or each module to have an individual SPI bus.

Table 3.2: Pinout between the PRAM PCB and the FPGA

Spartan-6 Pin	Pin Name	Description
AA10	MEM1_S	Bank 1 Chip Select
P6	MEM1_W	Bank 1 Write Protect
AB10	MEM1_Q	Bank 1 Data Out of PRAM
P8	MEM1_H	Bank 1 Hold
P7	MEM1_D	Bank 1 Data Into PRAM
B12	MEM1_C	Bank 1 Serial Clock

In future revisions of the FPGA board, this memory solution will be replaced with triplicated DDR SRAM modules to ease memory limits and increase read/write speeds.

## POWER SYSTEM DEVELOPMENT

Introduction

Design of power regulation circuits is a vital component of designing systems to be power efficient and requires much care in circuit design and layout to permit the circuit to operate optimally.

The requirements for the power board for this project were a flexible, efficient and lightweight part that could be used in a wide variety of laboratory and in-situ environments. Of importance was the board have a wide enough input range to be flown both on the HASP Balloon ( $30\pm 2V$ ), or be self-powered with AA batteries (9V) for other launches or laboratory demonstrations. It was also decided that it would be desirable for the board to have internal monitoring capabilities. Once design began, fault protection and closed loop margining circuitry were found to be minimally invasive to add and extremely valuable. The fault protection has eliminated the danger of shorting the board during bench testing while the closed loop margining has increased voltage stability vs temperature or load and allows for in-situ adjustment of the analog regulators.

Table 4.1: Power Supply Requirements

	Minimum	Typical	Maximum
Input Voltage Range	4V	9V or 30V	42V
FPGA Voltages	-5%	0%	+5%
Analog Voltages	-25%	0	+25%

## Theory of Operation

### Linear Regulators

The simplest power supply circuit is the linear regulator. These supplies function through use of a Zener diode to set the desired voltage, such as in the circuit shown in Figure 4.1, while dissipating the excess as heat across a resistor. These regulators are simple to design, have no EMI issues and provide clean DC power rails. However, their conversion efficiency can be very low and is calculated by  $Eff = V_{out}/V_{in}$ .

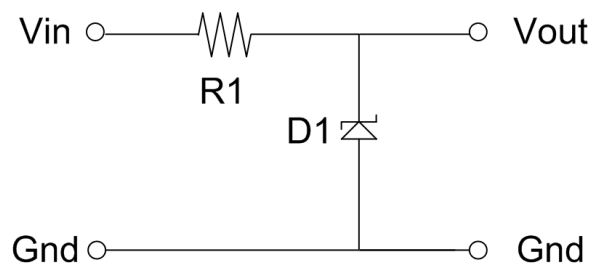


Figure 4.1: Simple Zener Regulator Circuit

This results in unacceptably low efficiencies when running with a  $V_{in}$  of 30V for this design. When outputting voltages between 1V and 3.3V a linear regulator circuit would only achieve 3-11% efficiency, which was not acceptable.

### Switching Regulators

Switching regulators operate by rapidly connecting and disconnecting the input power line from an energy storage element. These regulators do not contain resistive elements to dissipate the undesired energy, but simply disconnect the supply in a PWM-like fashion. As such, they can achieve much greater efficiencies as losses are limited to the non-ideal nature of the components, rather than being inherent in the

design itself. The design of these regulators is more complex and the switching effect results in output ripple and layout complexities, but result in conversion efficiencies that can exceed 90%. This makes them ideal for situations where power efficiency or heat are of concern.

The supplies used in this design fall into two of the common switching regulator architectures, Buck and Boost.

Buck Regulator: A Buck regulator is a simple mode for step-down switching voltage regulation. In this design, the input supply is switched to charge the inductor  $I$ , which is the energy storage element of the design and the switching occurs based on the voltage present at  $V_{out}$  to maintain the charge. The flyback diode presents an ideal path for the current to flow when the switch is open and prevents reverse current flow while the inductor is charging. The output waveform is filtered using the load resistance and the output capacitor to achieve a DC output. This regulator design has the advantage that since the current is sourced directly from  $V_{in}$  through the inductor, the maximum current is not inherently limited by the regulator architecture.

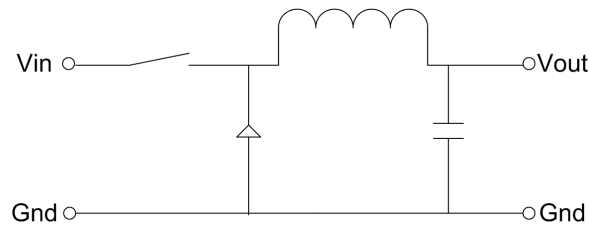


Figure 4.2: Switching Power Supply Buck Regulator Circuit

Except the bias voltage regulator, all of the regulators in this design are either Buck converters or minor variants.

Boost Regulator: Boost regulators are simple step-up regulators and can only supply voltages greater than the input voltage. In this architecture, when the switch is closed, the inductor has the greatest current and thus energy is stored in its magnetic field. Upon opening the switch, the inductor will continue to supply the high current across the load resistance causing a voltage spike above  $V_{in}$ . In this regulator circuit, the output resistance and load resistance is also used to filter the output voltage to a DC output.

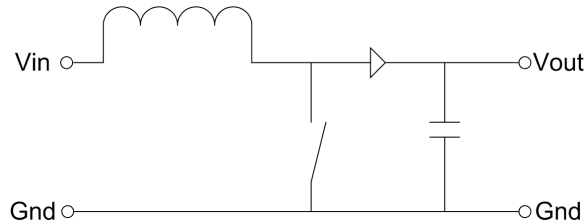


Figure 4.3: Switching Power Supply Boost Regulator Circuit

This form of regulator is used to provide the 15V bias in this design.

Loop Feedback: In all designs, the switching circuits are developed around a voltage comparator contained within the regulator that selects when to open and close the MOSFET switch to maintain the desired output voltage. This control circuit is of critical importance to set and maintain a stable output voltage.

To set the output voltage, a voltage divider is created by  $R1$  and  $R2$  to provide the reference voltage specified in the regulator datasheet when the desired  $V_{out}$  is present.

To increase the stability of the feedback circuit, a feed-through capacitor and resistor can be added across the inductor. This can serve to prevent the regulator from responding to transients and reduce ripple, or if improperly chosen it can make

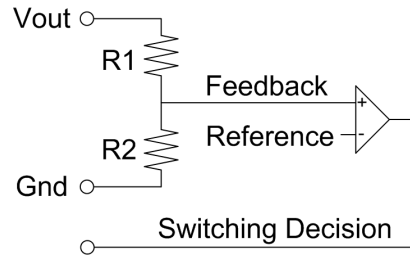


Figure 4.4: Power Supply Feedback Circuit

the regulator respond slower to voltage changes and voltage droops. These capacitors, if used, tend to have values on the order of pF.

Closed Loop Margining: To change the voltage of the regulator at runtime, the effective values of the resistors in the feedback voltage divider circuit must be modified. This can be accomplished by adding an additional resistor with an induced voltage.

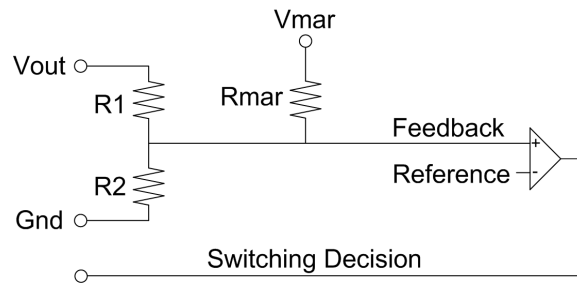


Figure 4.5: Closed Loop Margining Circuit

The value of  $R_{mar}$  can be computed by  $R_{mar} = \frac{R_1 R_2 V_{ref} V_{mar} * R_1 * R_2}{R_2 * V_{outmax} - R_2 V_{ref} R_1 V_{ref}}$  where  $V_{outmax}$  is the maximum desired output voltage and  $V_{ref}$  is the switching supplies comparator voltage reference. The resulting  $R_{mar}$  will then allow margining symmetrically  $\pm\Delta V$  where  $\Delta V = V_{outmax} V_{out}$ .

In the simplest mode the  $V_{mar}$  is either Gnd (0V) or Vcc (3.3V), allowing for a margin high and margin low setting in addition to the unmargined standard value. The resulting values can be used to compensate for the voltage drop that occurs when the power supply is loaded.

In active trimming mode  $V_{mar}$  is generated by a PWM with an RC filtering circuit. Reference equations for choosing these values can be found in [43] and [44]. Combined with the voltage monitoring circuitry, this allows the power controller to automatically vary  $V_{mar}$  to maintain a set voltage at the output pin of the power board independent of the load current.

### Power Conversion Architecture

This power system required the design and implementation of 12 power supply rails.

Table 4.2: Required voltage regulators and usage

Voltage	Component	Description
10V	TI LM3150	Intermediate Voltage Regulator
15V	TI LMR62014	Low Current, Sensor Bias Voltage
3.3V	TI LMR14203	Power Board Digital Components
3.3V	TI TPS62130	FPGA I/O Banks
2.5V	TI TPS62130	FPGA I/O Banks
1.8V	TI TPS62130	FPGA Configuration Voltage
1.0 or 1.2V	TI TPS62130	Spartan-6 Core Voltage
0.9 or 1.0V	TI TPS62130	Virtex-6 Core Voltage
2.5V	TI TPS62130	Radiation Sensor Front-side Comparator Voltage
2.5V	TI TPS62130	Radiation Sensor Backside Comparator Voltage
3.0V	TI TPS62130	Radiation Sensor Amplifier Vdd Voltage
-3.0V	TI TPS62130	Radiation Sensor Amplifier Vss Voltage

These rails were organized to either drive power from the connector directly into the 2nd stage, or regulate higher voltages to a constant 10V for the 2nd stage regulation circuits.

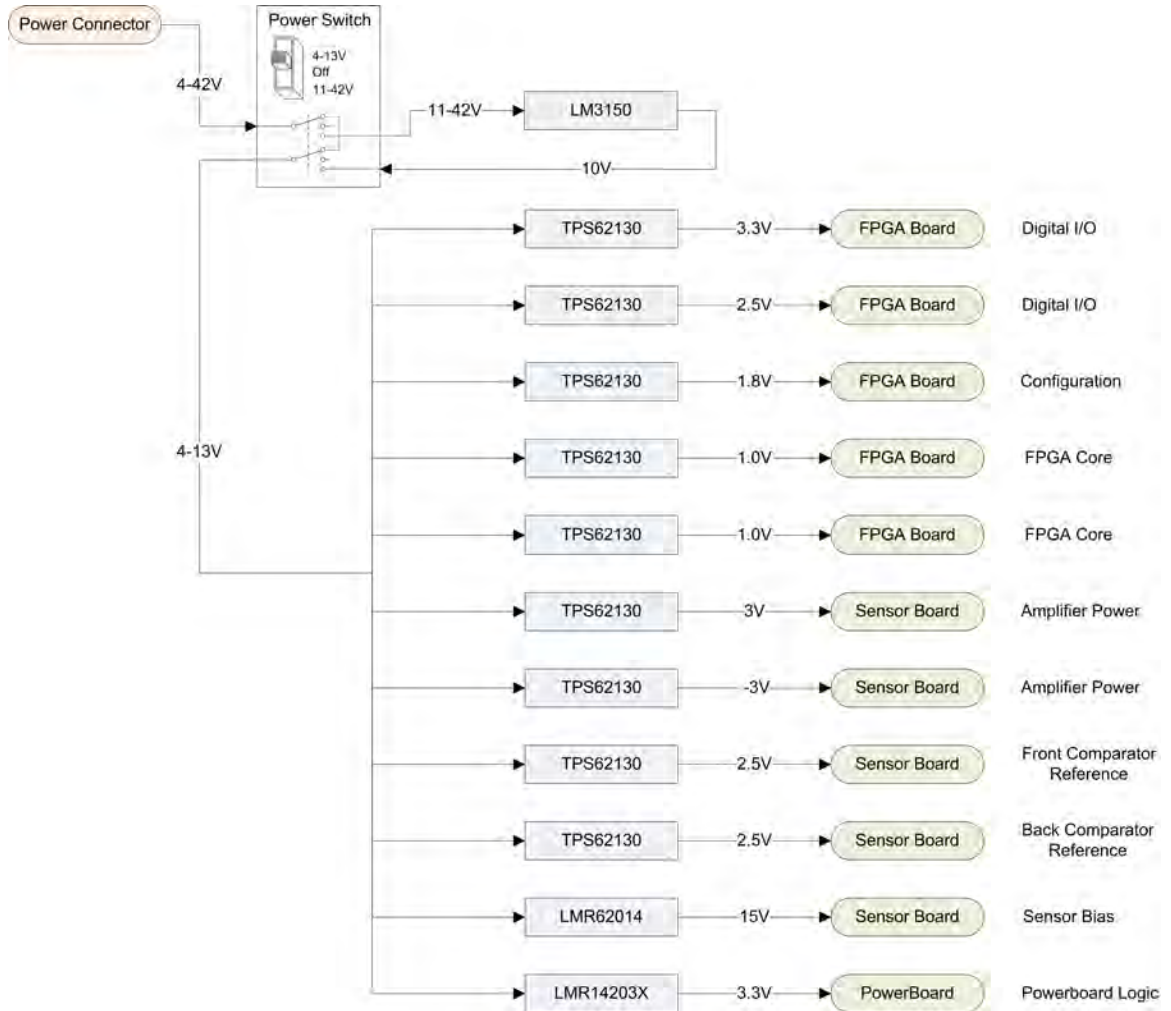


Figure 4.6: Power Conversion Architecture for the switching power supply board

### Power Supply Controller Devices

Power monitoring, power sequencing, fault detection, and margining are managed by two TI UCD90124A devices. Each contains twelve, 12 bit analog to digital lines



for voltage, current and temperature monitoring. They also have GPIO outputs for supply enables and PWM outputs to facilitate closed loop margining [45].

Communication with the modules is achieved with the PMBUS protocol over I2C, with either the Spartan-6 or a TI USB-GPIO operating as the master device. The modules can be configured using the TI USB-GPIO dongle and the TI Fusion GUI interface software. The I2C addresses are configured with resistors and are set such that the digital supply voltages are contained on the controller with an I2C Address of 0x52, and the analog voltages are on the controller with the I2C address 0x53.

Table 4.3: Digital Supply Configuration Parameters

Rail	#	Voltage	Current	Enable	Margin	Led Ext.	Temp
10V	7	Pin 6	Pin 5			Pin 11	Pin 62
3.3V	3	Pin 59	Pin 63	Pin 33	Pin 17	Pin 12	
2.5V	4	Pin 56	Pin 1	Pin 34	Pin 18	Pin 13	
1.8V	1	Pin 54	Pin 2	Pin 35	Pin 19	Pin 14	
1.0V S6	2	Pin 50	Pin 4	Pin 37	Pin 21	Pin 29	
1.0V V6	6	Pin 52	Pin 3	Pin 36	Pin 20	Pin 25	

Table 4.4: Analog Supply Pin Configuration Parameters

Rail	#	Voltage	Current	Enable	Margin	Led Ext.	Temp
10V	6	Pin 56		Pin 11			Pin 63
15.0V	4	Pin 6	Pin 59	Pin 33	Pin 17	Pin 12	
2.5V Fr	3	Pin 5	Pin 1	Pin 35	Pin 19	Pin 25	
2.5V Bk	2	Pin 4	Pin 62	Pin 34	Pin 18	Pin 29	
3.0V	1	Pin 50	Pin 2	Pin 36	Pin 20	Pin 14	
-3.0V	5	(Pin 52)	Pin 3	(Pin 37)	(Pin 21)	Pin 30	

### Voltage and Current Monitoring Circuits

Voltage monitoring is performed by ADCs on the power controller modules, however, the ADCs are limited to a maximum voltage of 2.5V. For any voltage that may exceed this, a voltage divider was used. These measurements were taken at the output power connector pin in an attempt to increase the accuracy of these

measurements and reduce the error at the load due to ohmic losses that might occur within the power board itself.

The current measurements were taken at the output of the power supply block. This circuit consisted of an inline 50 mOhm current sense resistor with a differential op-amp measuring the voltage across it. This op-amp (TI INA196) amplified the differential voltage by a ratio of 20x and passes the signal to an ADC on the power controller for measurement. Some of the UCD90124 monitor pins have a minimum voltage of 0.2V, so care was taken not to use these pins for current monitoring. This results in an effective value for the current sense resistor of  $R_{cs\_effective} = 20 * 50m\Omega = 1000m\Omega$ . Use of an INA197 (50V/V) or INA198 (100V/V) can be used on lower current rails to increase measurement accuracy if desired.

Table 4.5: Digital Power Supply Controller Voltage and Current Ratios

Rail	Voltage Ratio [V/V]	Current Ratio [mOhm]
Input	0.0625	
3.3V	0.5714	1000
2.5V	0.6667	1000
1.8V	1	1000
1.0V S6	1	1000
1.0V V6	1	1000

Table 4.6: Analog Power Supply Controller Voltage and Current Ratios

Rail	Voltage Ratio [V/V]	Current Ratio [mOhm]
10.0V	0.1668	1000
15.0V	0.1333	1000
2.5V Fr	0.6667	1000
2.5V Bk	0.6667	1000
3.0V	0.5714	1000
-3.0V	0.5	1000

### Enable, Voltage Select and G-Switch Functions

To allow for this circuit to be used for a possible RockOn sounding rocket launch, the safety circuitry for program was designed into this power supply board. This includes:

- Ready for Flight Jumper: This jumper can be toggled once the payload is added to the vehicle and must be connected to allow the payload to be powered. This allows the payload design team to prevent the payload from powering prior to the vehicle being moved to the Launchpad. Once the Read-For-Flight jumper is powered, the input voltage and current circuitry are enabled and the device will draw a small current.
- G-Switch / G-Switch Bypass Jumper: The g-switch is an empty header which was designed such that a vertical board could be connected into it with a pushbutton switch and a self-regenerating MOSFET circuit to power the circuit when a vertical acceleration was observed by the payload, and thus prevent the payload from fully powering while sitting on the Launchpad. Alternatively, this can be replaced by a simple jumper to enable to power supply and operates as a second enable jumper.
- Power Switch: The power switch is third in the line. This switch allows the supplies to be turned on and off and the voltage range to be selected. This has the effect of either adding the high voltage regulator to the circuit or bypassing it and passing the input voltage directly to the primary voltage power plane.

### Power Supply Sequencing:

Power supply sequencing is performed by the power sequencer devices primarily as a means of reducing inrush currents. However, the Virtex-6 FPGA does have a

preferred power sequencing which is respected through the sequencing pattern. As the -3V rail is always enabled in the current design due to a faulty level shifter circuit, the sensor rails are brought online first, followed by the Virtex-6 and Spartan-6 FPGAs.

### Testing and Verification Results

The power supply was designed and built as a 4 square board, and tested to verify its functionality and system operating parameters.

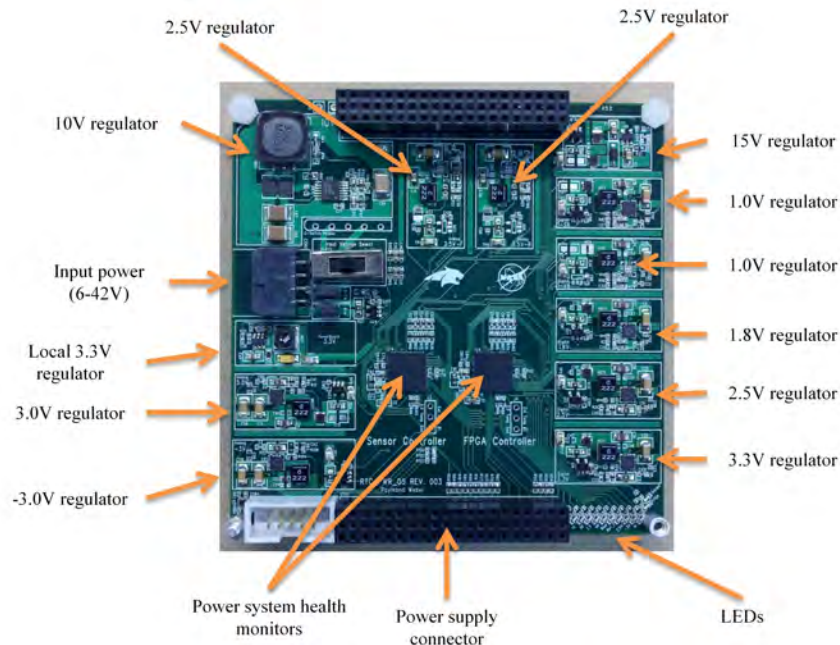


Figure 4.7: Fabricated Power Supply Board

### Efficiency vs Load

Power supply efficiency vs load for the largest voltage rails was measured with constant DC resistive loads and  $V_{in} = 9V$ . The power supply's digital logic circuitry also draws a constant current of approximately 50mW to power the power supply controllers and monitoring circuitry.

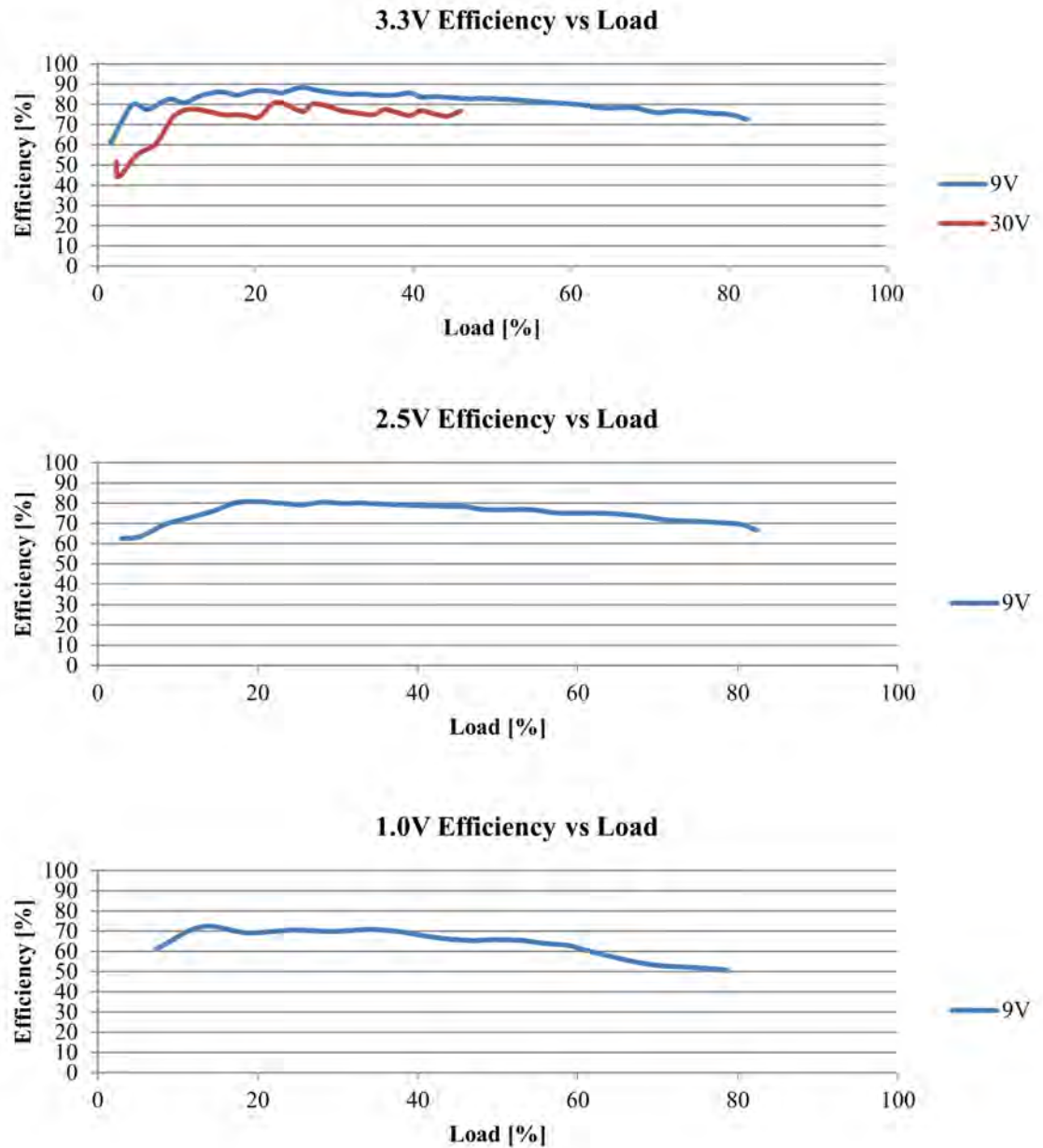


Figure 4.8: Efficiency of the power supplies vs the output load

The high voltage regulator as designed became unstable when sinking over 2A, and will require additional tuning to operate properly at higher currents.

### Output Voltage vs Load

The power supply was also tested across the voltage range to determine the effect of output load on the output voltage and voltage regulation parameter at the output pin of the power supply board.

The linear drop apparent in these plots is due to the resistance in the traces between the output pin of the regulator and the power connector.

These measurements lead to the output regulations at the power connector, where the regulation is defined as:

$$regulation_{load} = \frac{|V_{0\%} - V_{100\%}|}{V_{100\%}} * 100 \quad (4.1)$$

Results in the calculated regulations of:

Table 4.7: Voltage regulation at the board output

Voltage Line	Unmargined Load Regulation	Margined Load Regulation
3.3V	17.9%	0%
2.5V	28.2%	0%
1.0V	117.4%	26%

These results show that the voltage regulators provided the desired voltage across the entire range for 2.5V and 3.3V regulators when margined. The 1.0V regulator provided the desired voltage up to about 60% of full load (1.8A) after which the regulation voltage rapidly drops and becomes unusable before reaching full load.

The regulators with margining should be configured to use active trimming to achieve the optimal output voltage across the load range.

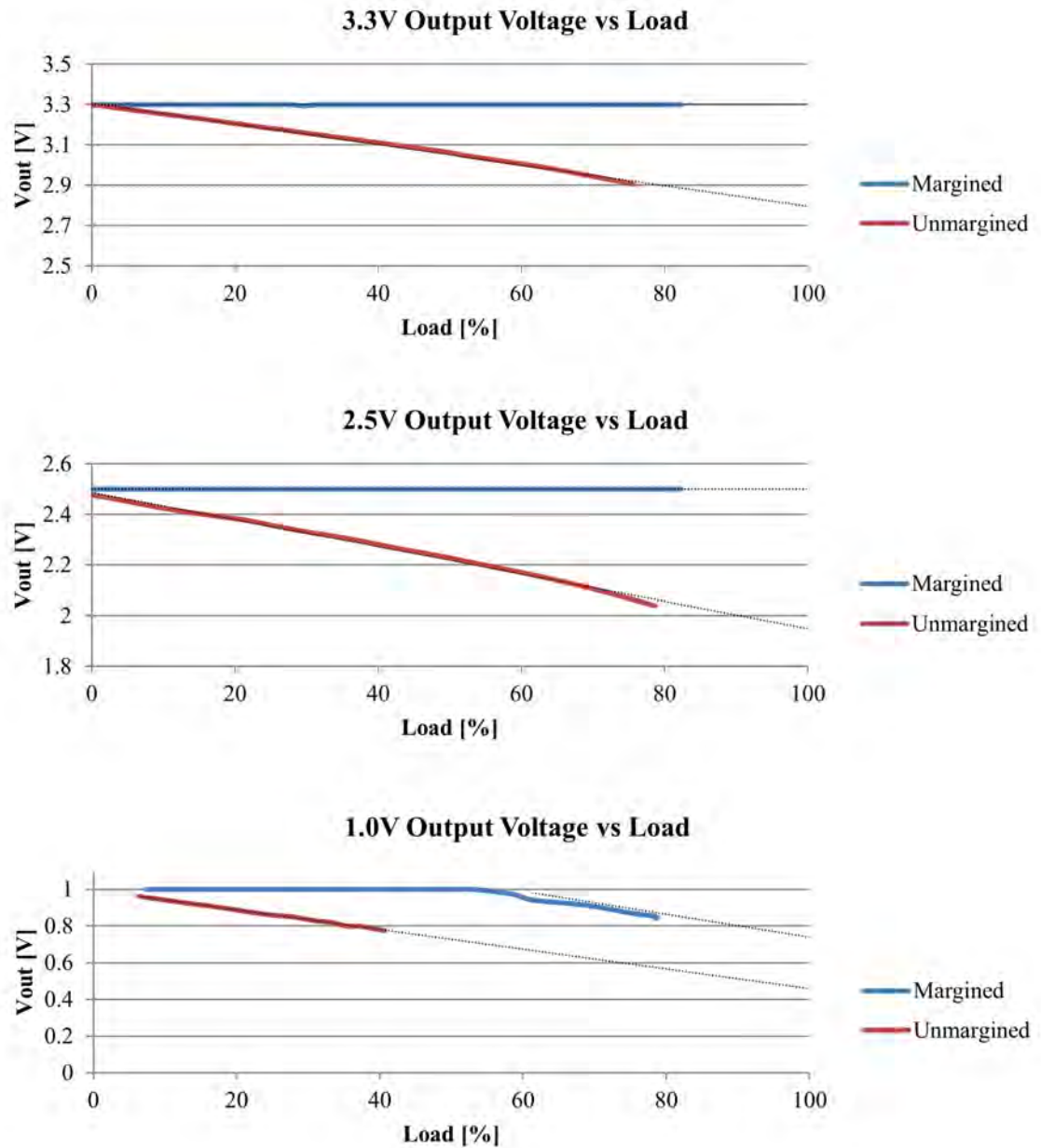


Figure 4.9: Output voltage of the power supplies vs the output load

### Efficiency vs Input Voltage

The power supply efficiencies were measured vs the input voltage when loaded with a 0.5A DC load.

This test showed that the circuit was most efficient with lower voltages and the greater efficiency of the high voltage regulator improved the efficiency of the circuit when operating at voltages greater than the intermediate 10V rail.

### Output Voltage Ripple

Ripple was determined to vary as the load varied but followed no long term trend in the amount of ripple. Numerous measurements were recorded at different load levels and the minimum and maximum recorded in Table 4.8.

Table 4.8: Calculated regulation due to ripple voltage (regulation at the regulator circuit output)

Voltage	Best Measurement [mVpp]		Worst Measurement [mVpp]	
	mVpp	%regulation	mVpp	%regulation
3.3V	15.4	0.47	35.2	1.07
2.5V	20.2	0.81	33.8	1.35
1.8V	13.2	0.73	24.2	1.34
1.0V	7.2	0.72	21.8	2.14

In all cases the regulation was better than the 5% recommended for the FPGA for proper operation. Should quieter lines be required, the power board contains an output ripple LC filter circuit, of which the capacitors were not populated on the tested boards.



## FPGA HARDWARE TILES

### Introduction

Microprocessors, while simple to utilize and program, are not without limitations. For this research, lack of parallelism when combined with the slower clock speeds inherent in FPGA designs results in lower performance systems. Softcore processors like the Xilinx MicroBlaze are unique in their ability to be expanded with custom instructions and hardware which allows for the strength of the FPGA to be exploited while maintaining simplicity of design to the user. We seek to use this ability by creating a hardware accelerated routines to improve the performance of common scientific computing applications.

### Partially Reconfigurable Tiles

In conventional FPGA design the device bitstream is loaded into the FPGA at startup and remains static. This results in all the device functionality and digital hardware to be present at all times. As an alternative, partial reconfiguration (PR) can be used to divide the FPGA into smaller regions which can be reconfigured during system operation. Each then contains a subset of the total functionality required for the complete design (Figure 5.1).

By combining tile systems and temporal separation between events, the system can utilize more specialized hardware blocks to increase performance. With this, it is possible create a design that would conventionally use more than 100% of the FPGA resources while eliminating unnecessary static power consumption.

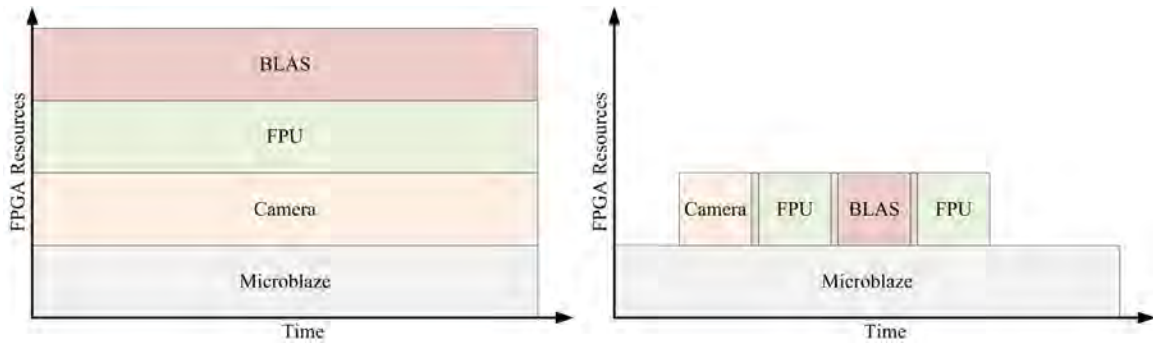


Figure 5.1: Conventional FPGA Resource Requirements in time vs Partially Reconfiguration Resource utilization

Partial reconfiguration also provides the basis for FPGA memory scrubbing. If the damage caused by a radiation strike can be isolated to a specific tiled region on the FPGA, that tile can be partially reconfigured to restore it to proper functionality. This eliminates the need to halt all operation and configure the entire FPGA.

### MicroBlaze Tile

The MicroBlaze tile is the primary tile type for this research system and is the only tile which has additional I/O beyond a BRAM interface. This tile contains the microprocessor itself, a compiled program in BRAM and I/O interfaces needed for the system. It is also the only tile whose outputs are protected by the TMR voter.

Compared to commonly used CubeSat processors, the implemented processor has far greater performance, and is only lacking in memory present on the prototype hardware (Table 5.1).

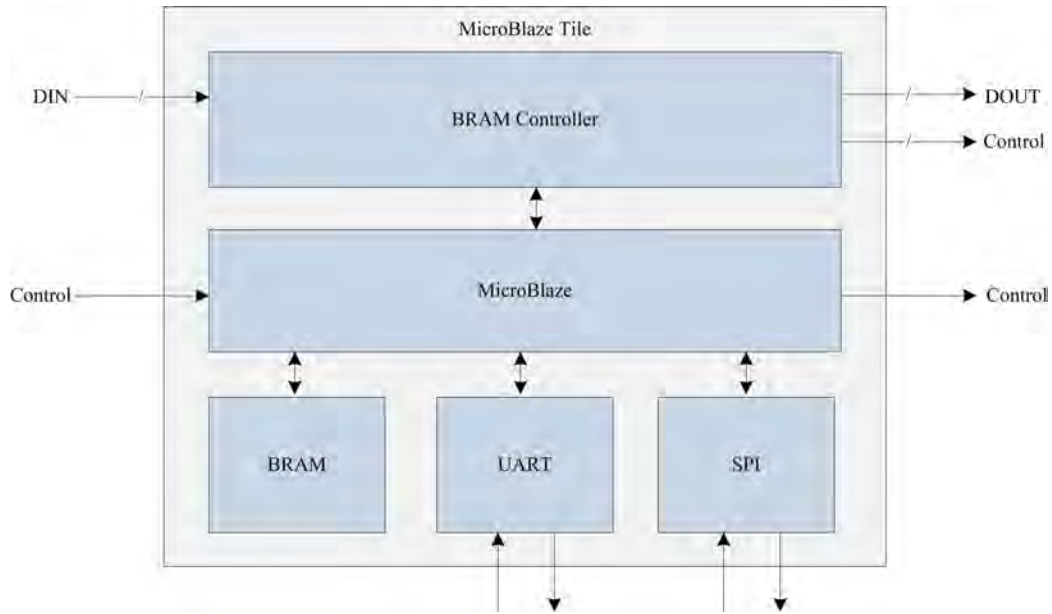


Figure 5.2: MicroBlaze Tile Architecture

Table 5.1: MicroBlaze vs common CubeSat microprocessors

	MicroBlaze	SI 8151	PIC 24F	dsPIC33 TI	MSP430
Speed	180MHz	100 MHz	16 MHz	80 MHz	25 MHz
Width	32 bit	8 bit	16 bit	16 bit	16 bit
FLASH		128 KB	256 KB	256 KB	116 KB
RAM	64 KB	8KB	16 KB	30 KB	8 KB
DDR	1 GB				
DMIPS	234 (uP)	20	24	92	25
Power	1.5W	195 mW	66 mW	297 mW	9.9 mW

### Agnostic Hardware Accelerator Tiles

To interface the accelerator tiles with the MicroBlaze processor, it was determined that the native block RAM (BRAM) interface would provide a simple interface with high throughput. A high speed clock and reset logic was passed into the accelerator allowing for them to operate at higher speeds.

## Floating Point Unit

Floating point math operations are multistage procedures in software which results in basic numerical operations requiring 50-500 clock cycles to be performed in software. As such, it is best performed in dedicated hardware floating point units (FPUs) utilizing parallelism [46]. The addition of the designed floating point hardware reduces the performance penalty of floating point operations to 20-100 clock cycles.

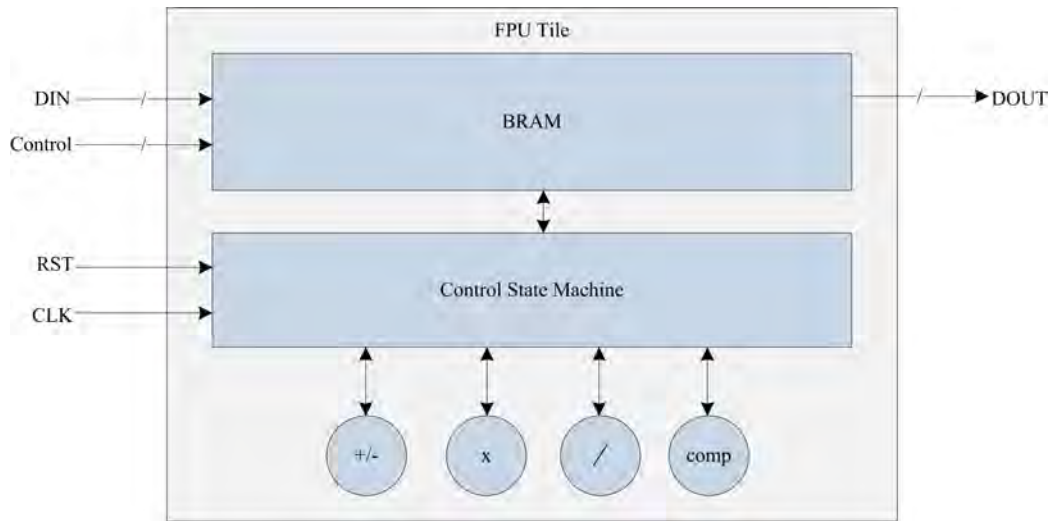


Figure 5.3: FPU Tile Architecture

The operations implemented in this accelerator are the basic mathematical operations: addition, subtraction, multiplication, division and comparisons. These operations were designed to replace the default double precision soft float library through function overloading.

The state machine for the FPU is shown in Figure 5.4. When active, the state machine loads data and the operation flags. The floating point cores run at 1/3 the clock rate so the state machine checks for the data ready flag every 3 clock cycles and stores the data to the output memory location when the operation is complete.

After storing the data, the system sets a done flag for the MicroBlaze and returns to an idle state.

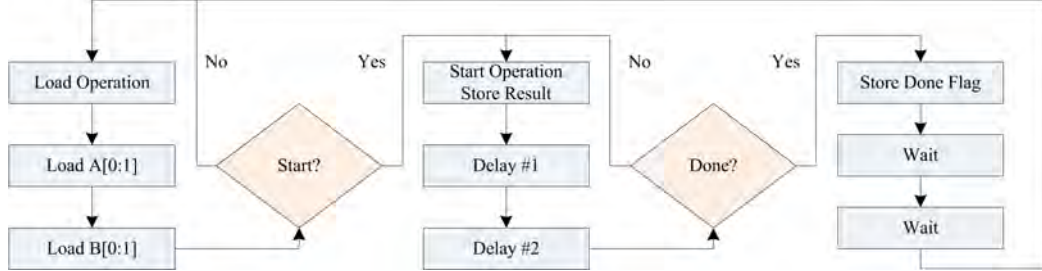


Figure 5.4: FPU Tile State Machine

The theoretical performance of this FPU, assuming no data movement was required is:

$$MFLOPS = \left( \frac{Cycles_{FPU}}{Clock_{FPU}} \right)^{-1} \quad (5.1)$$

For the design implemented, the parameters  $uP_{clk} = 150MHz$ , and  $FPU_{clk} = 400MHz$  were used to determine the peak computational bounds once data is stored in the accelerator:

Table 5.2: Implemented Double FPU Instructions

Operation	Description
ADDD	Double Addition
SUBD	Double Subtraction
MULD	Double Multiplication
DIVD	Double Division
Misc	Double Compare Functions

The memory overhead is a significant limiting factor on the system. This accounts for an additional 16 clocks of CPU time at 150MHz to be added to the overall

calculation time (4 clocks for each input and output plus 4 clocks for the operation command). This expands above equation to:

$$MFLOPS = \left( \frac{Cycles_{FPU}}{Clock_{FPU}} + \frac{Cycles_{CPU}}{Clock_{CPU}} \right)^{-1} \quad (5.2)$$

and results in the following theoretical performance bounds:

Table 5.3: Theoretical FPU Accelerator Performance Limits

Operation	$FPU_{procCycles}$	$MFLOPS_{theoretical}$	$MFLOPS_{memLimited}$
Addition	21	19.05	6.28
Subtraction	21	19.05	6.28
Multiplication	30	13.33	5.51
Division	93	4.04	2.82
Compares	15	26.67	6.94

These performance metrics are a still a significant improvement over using a soft float library, but show that reducing data movement when possible would greatly improve performance.

### Basic Linear Algebra Subprogram Accelerator

Despite this improvement, single data operations are not efficient on the FPGA. The FPGA fabric is much better suited to operating on larger blocks of data such as vectors and matrices which inherently contain large parallelizable data sections. The Basic Linear Algebra Subprograms (BLAS) is a library of rudimentary functions commonly used in low level vector and matrix operations. This library was first proposed in 1979 and has become the standard API for linear algebra routines. The BLAS library is used in many scientific computation packages including LINPACK and high level languages such as MATLAB [47]. By building specialized hardware

to perform these functions greater parallelism and greater performance computations can be achieved [48–52].

The BLAS library consists of three levels of matrix abstraction. Level 1 routines are basic scalar/vector operations. These include routines mostly follow the form  $C = a * B$ , such as dot products, scalar products and vector normals. Level 2 routines are vector/matrix operations that can be in the form of  $C = A * B$ , such as vector matrix multiplications. Level 3 operations are matrix/matrix operations, of which the most commonly used is the general matrix multiply, but also includes matrix transformations such as Gaussian elimination. In general, the higher level operations provide higher performance as they allow increased parallelism and pipelining while reducing function overhead [53].

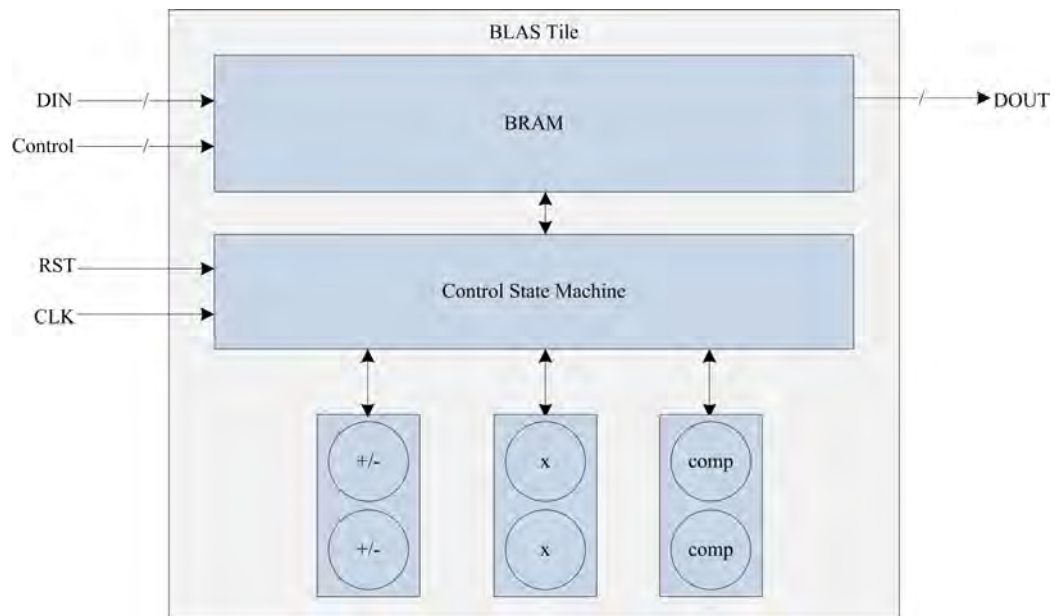


Figure 5.5: BLAS Tile Architecture

The basic computational instructions from the BLAS level 1 standard were implemented in a partially reconfigurable FPGA tile. The operations included are:

Table 5.4: Implemented BLAS Level 1 Instructions

Operation	Description
DAXPY	Double Scaler Vector Product
DASUM	Double Product of Magnitudes
DDOT	Double Dot Product
DSCL	Double Vector Scaler Product
IDAMAX	Index of Vector Maximum
IDAMIN	Index of Vector Minimum

The instructions for DAXPY, DDOT and DSCL as well as element-wise operations in the BLAS state machine follow the form in Figure 5.6. In this state machine, data is brought in two doubles per vector per clock and operated on and continues until the entire vector operation is complete. Upon completion the state machine toggles a done flag and returns to its idle state.

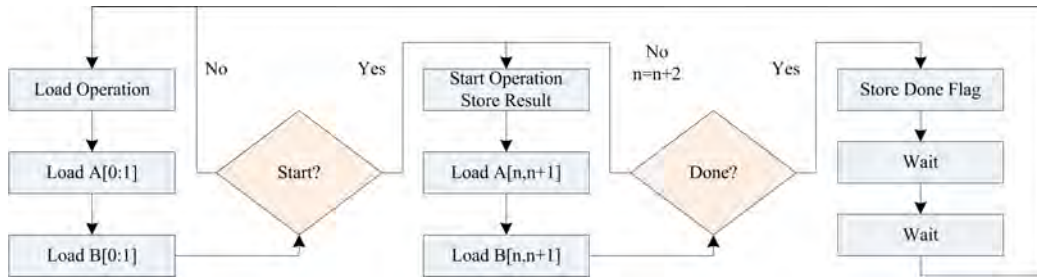


Figure 5.6: BLAS Tile State Machine

IDAMAX, IDAMIN and DASUM follow the same basic structure but due to data dependence form two results, one for the even and one for the odd indices, at which point an additional state is needed to provide the final result.

The theoretical performance of the BLAS Level 1 Accelerator can be found using the equation:

$$FLOPS = \left( \frac{Cycles_{BLAS}}{Clock_{BLAS}} \right)^{-1} \quad (5.3)$$



The basic accelerator architecture is the same as the FPU; the constants  $uP_{memCycles} = 12$ ,  $BLAS_{memReadCycles} = 6$ , and  $DSP_{clk} = BLAS_{clk}/3$  are unchanged.

Table 5.5: Theoretical BLAS Throughput

Operation	Cycles	MFLOPS (n=10)	MFLOPS (n=100)
DAXPY	$27+3n/2$	95.24	238.10
DASUM	$27+3n/2$	95.24	238.10
DDOT	$27+3n/2$	95.24	238.10
DSCL	$27+3n/2$	95.24	238.10
IDAMAX	$18+3n/2$	121.21	238.10
IDAMIN	$18+3n/2$	121.21	238.10

In addition, simple vectorized instructions are present to expand upon the FPU when data is known to be vectorizable. For these instructions, if the data is known to be independent, performance can be increased by grouping multiple operations into a single instruction operating on all the data in the same hardware call, and reducing overhead.

Table 5.6: Theoretical elementwise vector throughput with bandwidth limited memory

Operation	FPU Cycles	MFLOPS (n=10)	MFLOPS (n=100)
Element Addition	$3n/2+18$	121.21	238.10
Element Subtraction	$3n/2+18$	121.21	238.10
Element Multiplication	$3n/2+27$	95.24	225.99
Element Comparison	$3n/2+12$	148.15	246.91

Data transfers have a major impact on the performance which can be modeled by this accelerator by adding 12 clocks for each vector position and 4 for the operation and status. This reduces the overall theoretical performance dramatically and shows the memory throughput limitation of the architecture. Since the instructions operate on vectors, which are passed as addresses, it is feasible to check the input and output

array addresses and verify if the data can be operated on in place or if it must be copied to the accelerator.

In the case where the data must be moved, the CPU cycles to copy the vector into the accelerator memory must be added to the operation time:

$$FLOPS = \left( \frac{Cycles_{BLAS}}{Clock_{BLAS}} + \frac{Cycles_{CPU}}{Clock_{CPU}} \right)^{-1} \quad (5.4)$$

This results in the following theoretical performance bounds for the BLAS instructions (Table 5.7):

Table 5.7: Theoretical BLAS Throughput with Bandwidth Limited Memory

Operation	MFLOPS (n=10)	MFLOPS (n=100)
DAXPY	11.05	11.85
DASUM	11.05	11.85
DDOT	11.05	11.85
DSCL	11.05	11.85
IDAMAX	11.33	11.88
IDAMIN	11.33	11.88

For the vectorized instructions this modifies the peak theoretical performance to (Table 5.8):

Table 5.8: Theoretical Vector Operations with Bandwidth Limited Memory

Operation	MFLOPS (n=10)	MFLOPS (n=100)
Vector Addition	11.00	11.84
Vector Subtraction	11.00	11.84
Element Multiplication	10.73	11.80
Element Comparison	11.18	11.86

In all cases, if data needs to be copied into the accelerator, the performance is reduced to roughly double that of the FPU.

### Camera Accelerator

The camera module used in this project required more rapid data communication than could be provided with the microprocessor interface. A hardware accelerator was developed to buffer the data from the camera module before writing it to PRAM as a solution. This required that this accelerator contain additional pins for the camera input lines.

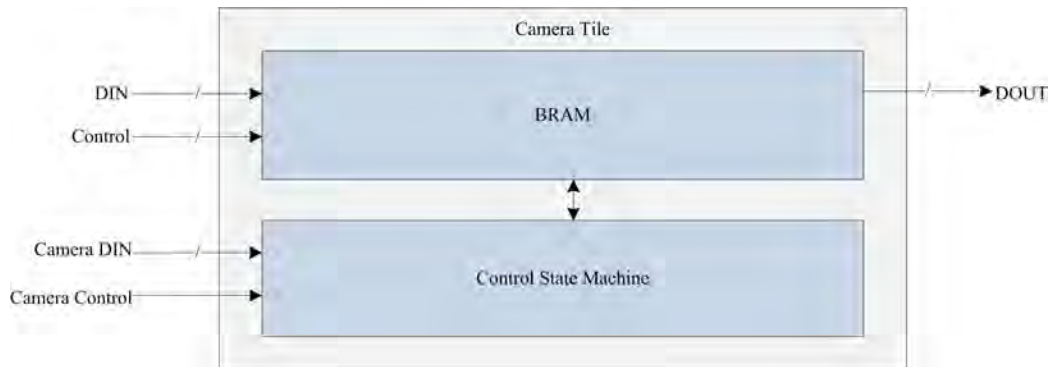


Figure 5.7: Camera Interface Tile Architecture

To maintain uniformity with the other acceleration tiles and complexity issues, the camera control and setup interface is part of the MicroBlaze tile. This interface is only needed for the initialization, after which image acquisition can be done entirely through the camera accelerator.

With this accelerator, a GUI interface to the registers and camera image was created to allow for the camera interface to be tested and the correct registers to be determined. The proper initialization of the camera required setting undocumented registers found through testing with the GUI interface and documented in Appendix A.

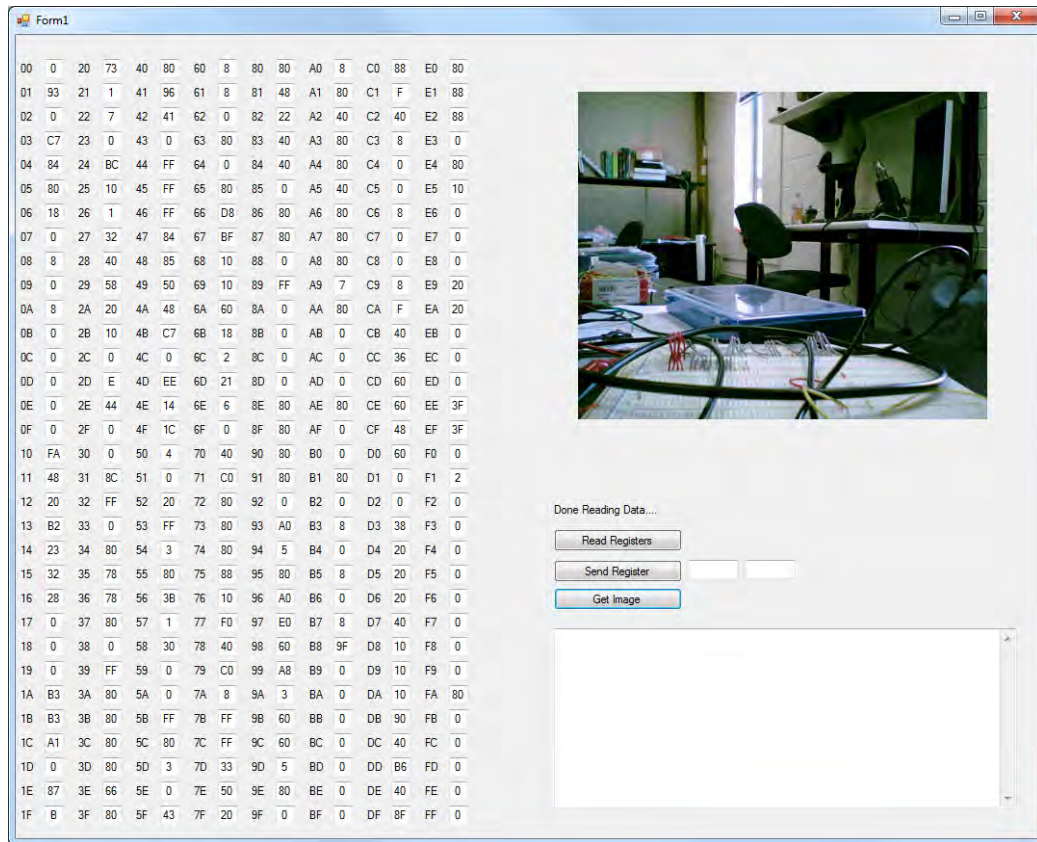


Figure 5.8: GUI for testing and configuring camera register settings

With this information the camera could be fully integrated into the system and could produce either JPG compressed images, or raw bitmap images for processing, storage or downlink.

### Bitstream File Concatenation GUI

The MicroBlaze system as designed reads the bitstreams from an unformatted SD card. To place multiple bitstreams on the same cards, the bitstream header must be removed and the files placed starting at a 512 byte block boundaries. As the complexity of this operation increased, it became necessary to automate this

procedure. This resulted in a software application to concatenate these hex files and to provide the information for the MicroBlaze system as to the file locations and sizes.

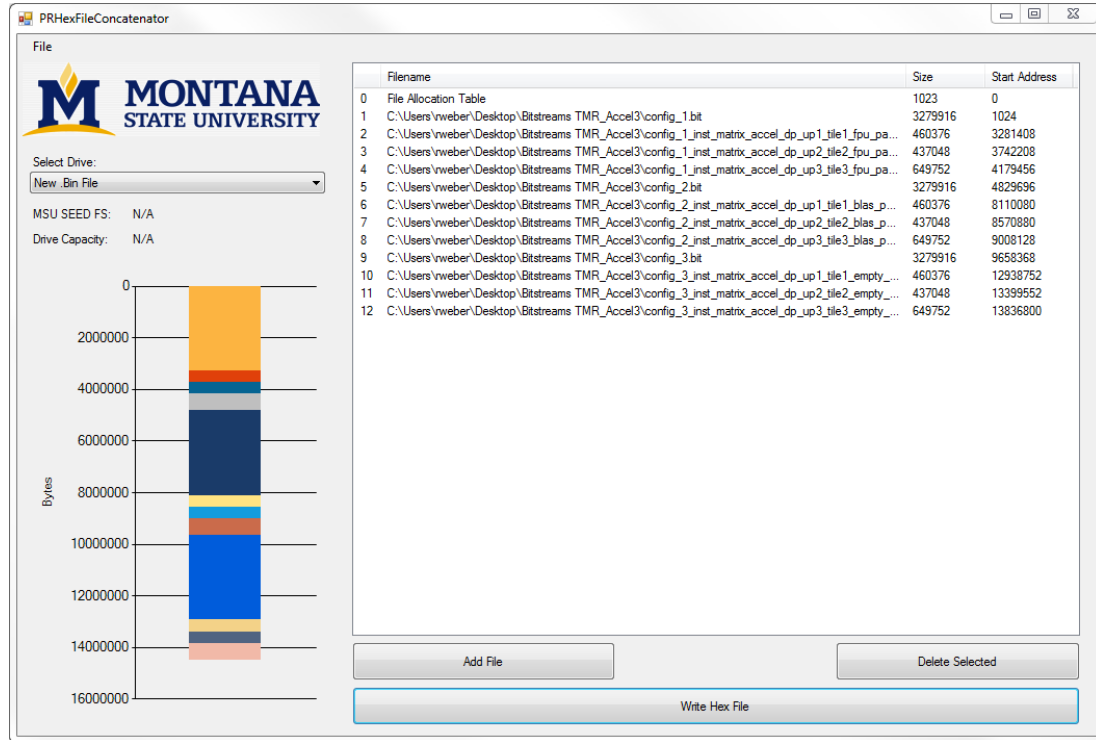


Figure 5.9: GUI for building SD Card images with multiple bitstreams

## PARTIALLY RECONFIGURABLE OPERATING SYSTEM

Introduction

During the development of the hardware accelerators, it was discovered that the software for the research platform was becoming increasingly complex due to the number of system peripherals and configuration options available (Figure 6.1). The programming overhead to utilize the monitoring the performance, radiation hardening and partial reconfiguration features could be greatly decreased by implementing an underlying operating system.

The design was split in two separate programs, one to manage the system monitoring tasks and manage the partial reconfiguration, while the other was designed to utilize the available reconfiguration resources in the tiles themselves. These two program act in concert with each other to perform the required desired tasks.

This permits a layered approach to be taken for application development (Figure 6.2). The applications each have access to a standardized library of functions, as well as to the base hardware functionality. This abstraction and access to higher level functionality includes specialized terminal function calls and partial reconfiguration options, as well as providing standard timing functions for benchmarking routines.

The two sides of the the operating system were named Control Operating System (ControlOS) for the Spartan-6 side and Partial Reconfiguration Operating System (prOSe) for the Virtex-6 MicroBlaze tiles.

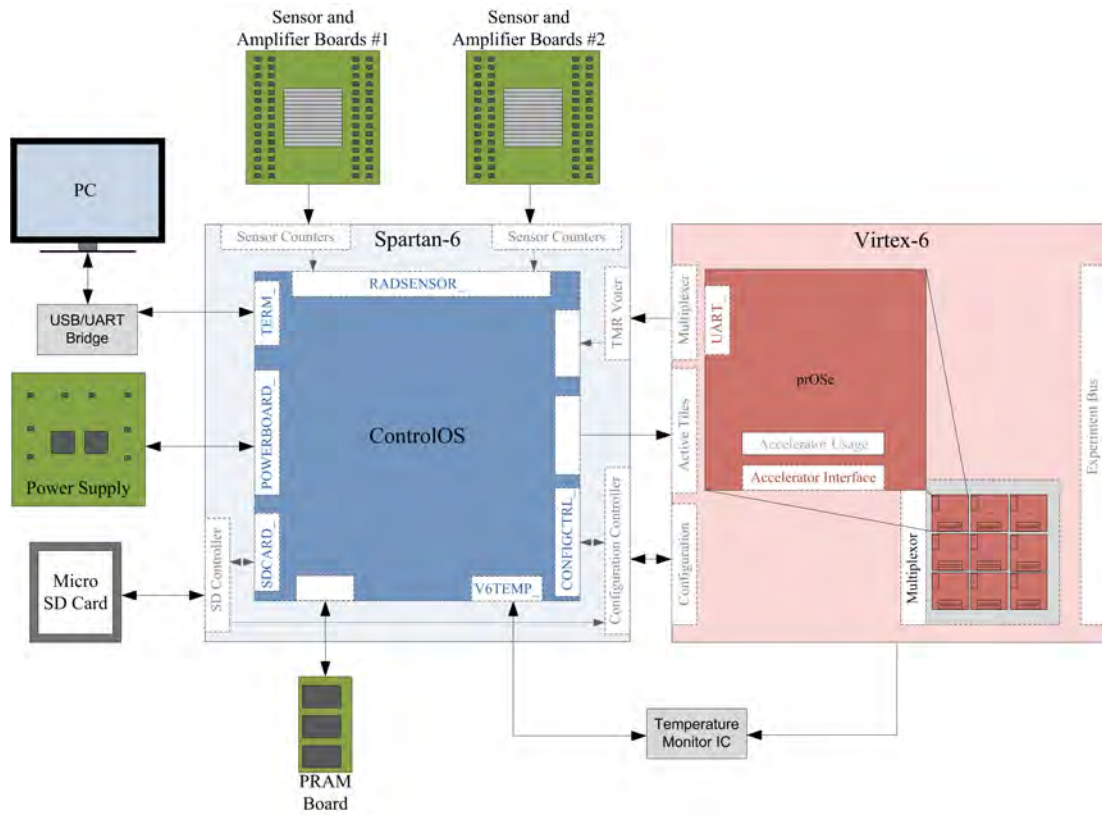


Figure 6.1: Functions performed by each operating systems, and the function name prefixes for the operations

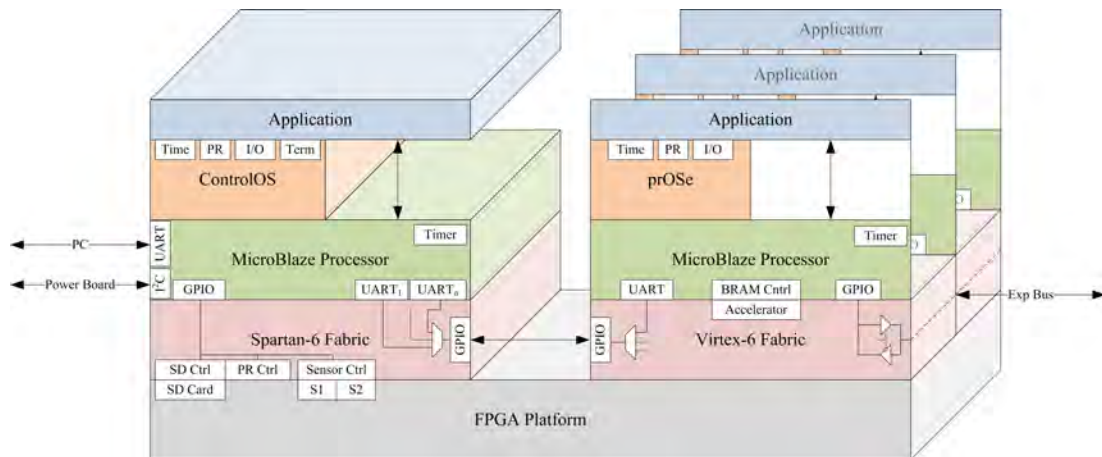


Figure 6.2: Operating System Abstraction Levels

## Control and Monitoring Operating System (ControlOS)

### Program Objective

The control side of the operating system was designed to monitor the system parameters, perform the radiation tolerant functions, oversee partial reconfiguration and provide a user interface to monitor the operation of the system.

As this design is not triplicated but fault mitigated using the Xilinx SEM controller, the system has access to more BRAM resources than the TMR'ed Virtex-6 processing tiles. This allows this program to have more complexity than its counterpart and less emphasis on performance.

To allow for ease of use to monitor and change system parameters, the system was configured to run as a terminal application. This allows for the operating system parameters to be monitored and adjusted using the mouse and keyboard in a terminal emulator.

### Power-On-Self-Test (POST)

Upon startup the operating system first tests initializes and tests all the hardware functions to ensure the system is working properly. These test results are stored in a SYSTEM\_POST\_struct that can be stored, downlinked or accessed during system operation to help debug the system.

At the conclusion of the POST, the Spartan-6 and Virtex-6 have been initialized and any initialization errors are logged. The Spartan-6 then enters a default non-interactive program mode in which streams data out the serial port. The Virtex-6 is configured and running its default program.



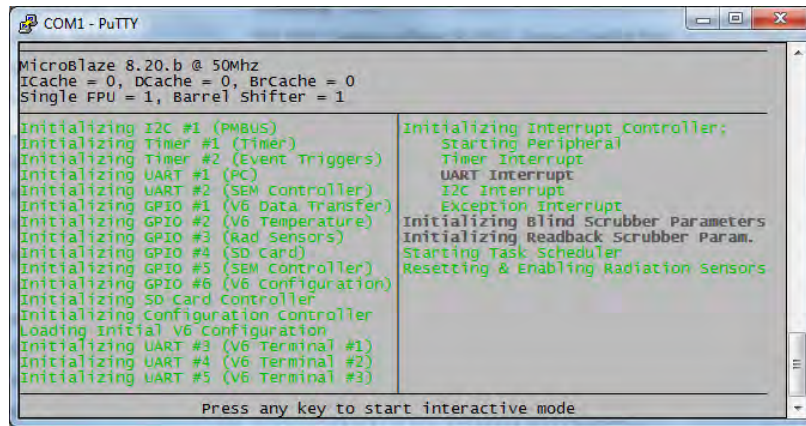


Figure 6.3: ControlOS Power on Self-Test (POST)

Scheduling Algorithm

This operating system was implemented as a first-in-first-out (FIFO) cooperative scheduling kernel with 16 task slots in a circular queue structure. Processes are added to a scheduled task list by a periodic timer interrupt function. This scheduling algorithm was chosen as it was simple to implement with the low overhead [54, 55].

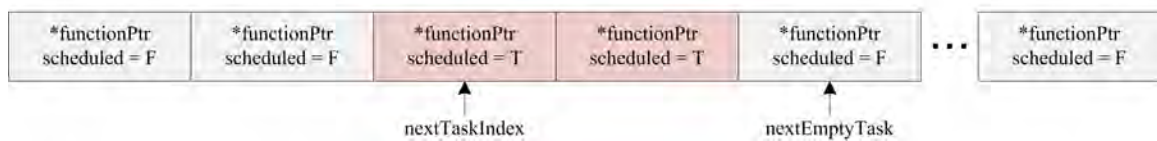


Figure 6.4: Ouput voltage of the power supplies vs the output load

This YEILD() function checks the scheduled task list for if any scheduled tasks are present and ready and if so runs the next scheduled task. Upon task completion the idle function marks the task as no longer scheduled and increments the scheduled task pointer.

The GUI is given a time slot between every YEILD() operation and is allowed to utilize any time unused by other tasks to improve the responsiveness of the GUI interface. This has the effect of increasing the performance of the GUI at the cost of higher latencies for the scheduled tasks.

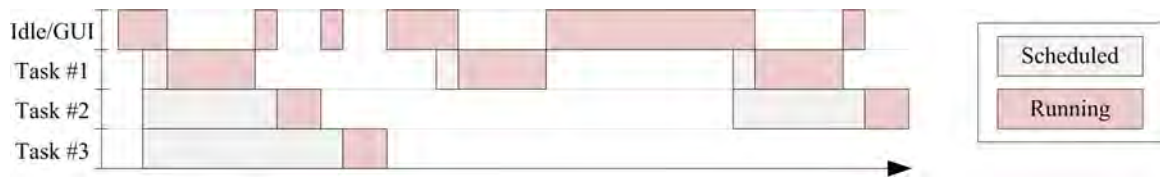


Figure 6.5: Example Process Timing using the YEILD() Function

The WAIT\_Keypress() function builds on the YEILD() function by completely yielding time to the scheduler until a keypress is registered. This function can be used to decrease the latency of the events when periodic GUI updates are not required.

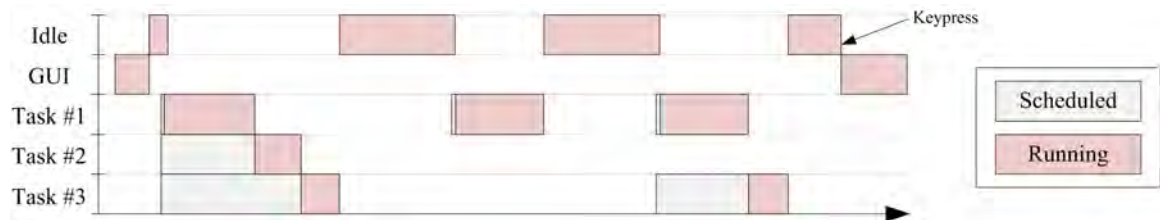


Figure 6.6: Example Process Timing using the WAIT() Function

For both functions, equivalent YEILD\_GUI() and WAIT\_Keypress\_GUI() wrapper functions were created for use in the terminal programs.

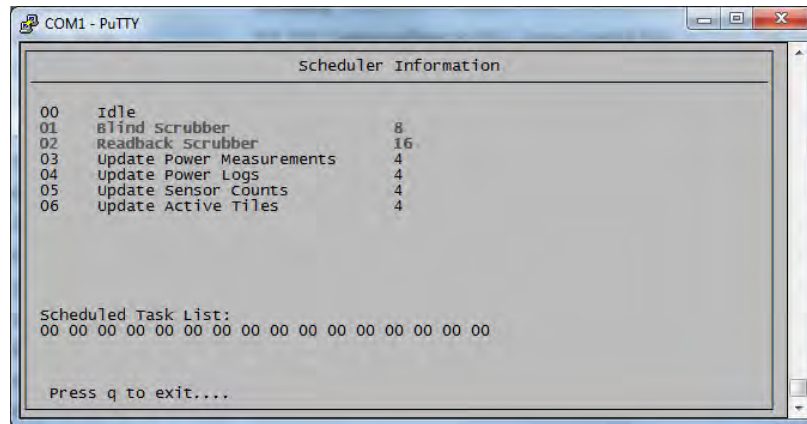


Figure 6.7: GUI Interface for monitoring scheduler activity

### Radiation Sensor and TMR Voter Monitoring Functionality

To collect the data required for this research, the outputs of all three tiles operating in TMR on the tile FPGA were monitored. This was simplified by having the control FPGA control the three tile UARTs. The system monitored voltages and temperatures periodically.

To provide radiation hardness for the Virtex-6, the control operating system monitored the radiation sensor as well as the TMR voter circuit to determine if and when errors occurred. In the event of an error, the system would mark the tile as bad, call for a context switch to a spare tile and then partially reconfigure the faulted tile and add it back to the spare list.

### Virtex-6 Reconfiguration Controller Interface

The control operating system will also perform partial reconfiguration at the request of the tile microprocessors to bring accelerators online as well as perform background scrubbing to ensure tile health.



Figure 6.8: GUI interface for monitoring tile activity

Prior to a partial reconfiguration being performed, the SD card must be initialized with `SDCARD_init()` and the reconfiguration control state machine with `PRCONTROL_init()`. These functions configure the SD card and state machine into an initial known state.

To configure the Virtex-6 with a full bitstream, the `PRCONTROL_ConfigureFull()` function is used. This function takes the start address of the bitfile on the SD card and the bitstream length as required parameters. This function blocks until the Virtex-6 is programmed and the configuration done pin is asserted.

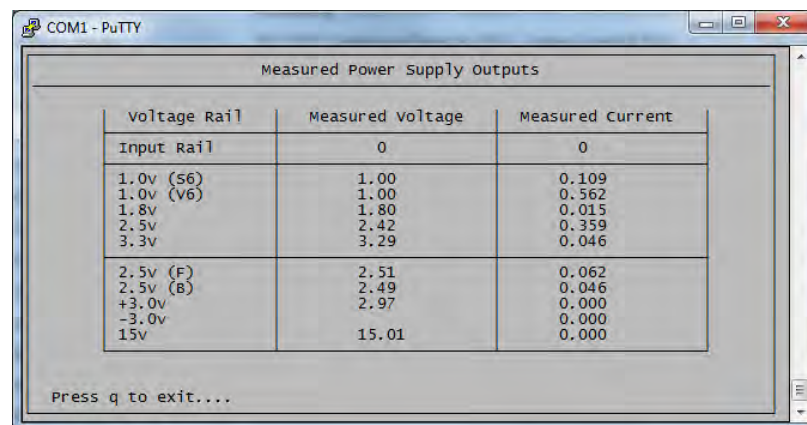
Partial reconfiguration is performed using much the same method, but using the `PRCONTROL_ConfigureTile()` function. The program blocks until the configuration controller responds that it has configured the full tile, but does not monitor a Virtex-6's response.

With this functionality, the blind scrubbing task is trivial to implement. The `SCRUBBER_Blind_Init()` function initializes a structure with the desired variables and current scrubber location. The `SCRUBBER_ScrubBlindTile()` is then scheduled

at periodic intervals and performs the blind scrubbing and increments the next tile to scrub counter.

### System Health Monitoring

The operating system communicates with the power controller modules to record current voltages, currents and temperatures throughout the system, which can be logged or viewed in real-time using the GUI interface.



voltage Rail	Measured voltage	Measured Current
Input Rail	0	0
1.0v (S6)	1.00	0.109
1.0v (V6)	1.00	0.562
1.8v	1.80	0.015
2.5v	2.42	0.359
3.3v	3.29	0.046
2.5v (F)	2.51	0.062
2.5v (B)	2.49	0.046
+3.0v	2.97	0.000
-3.0v		0.000
15v	15.01	0.000

Press q to exit....

Figure 6.9: GUI interface for real time power monitoring

### Partially Reconfigurable Tile Operating System (prOSe)

#### Program Objective

The operating system on the processing tiles was designed to make the accelerator functions transparent to the programmer and provide common functionality, such as timer events. This was accomplished by overloading functions based on available hardware and keeping metrics of function utilization to allow the software to determine the best hardware accelerator for its current application needs.

## Floating Point Unit Acceleration Functions

For the FPU functions, the double function call is overloaded. If an accelerator is available it is used with priority given to the faster FPU function over the higher latency BLAS versions. If no accelerator is present a software version of the floating point accelerated functions is called. This introduces some overhead but is offset by the additional functionality provided.

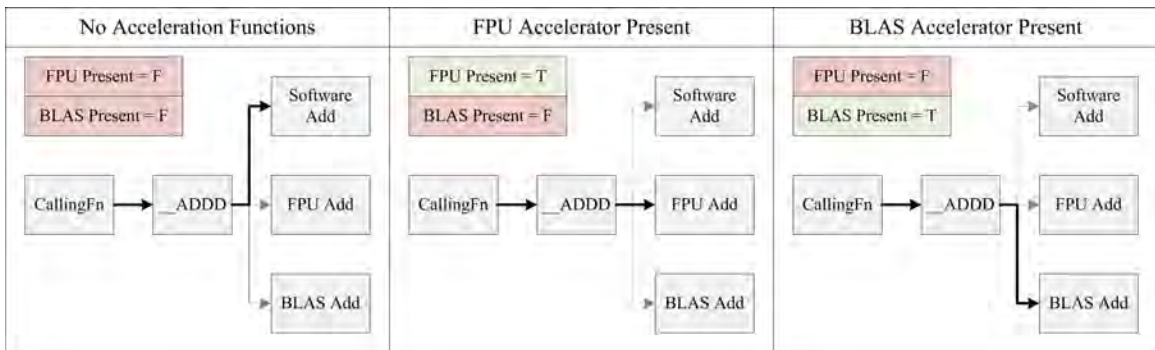


Figure 6.10: Decision Trees for FPU Overloaded Operations

Table 6.1: Floating point instructions and their overloaded alternatives

	Called Function	Software Function	FPU Function	BLAS Function
$A + B$	<code>__adddf3</code>	<code>_adddf3</code>	<code>FPU_adddf3</code>	<code>BLAS_adddf3</code>
$A - B$	<code>__subdf3</code>	<code>_subdf3</code>	<code>FPU_subdf3</code>	<code>BLAS_subdf3</code>
$A * B$	<code>__muldf3</code>	<code>_muldf3</code>	<code>FPU_muldf3</code>	<code>BLAS_muldf3</code>
$A/B$	<code>__divdf3</code>	<code>_divdf3</code>	<code>FPU_divdf3</code>	
$A > B$	<code>__gtdf2</code>	<code>_gtdf2</code>	<code>FPU_gtdf2</code>	
$A = B$	<code>__eqdf2</code>	<code>_eqdf2</code>	<code>FPU_eqdf2</code>	
$A < B$	<code>__ltdf2</code>	<code>_ltdf2</code>	<code>FPU_ltdf2</code>	
$A \geq B$	<code>__gedf2</code>	<code>_gedf2</code>	<code>FPU_gedf2</code>	
$A \leq B$	<code>__ledf2</code>	<code>_ledf2</code>	<code>FPU_ledf2</code>	
$A \neq 0$	<code>__nedf2</code>	<code>_nedf2</code>	<code>FPU_nedf2</code>	

## Basic Linear Algebra Subprograms (BLAS) Acceleration Functions

For the BLAS functions, a very similar process is carried out. The BLAS functions check to see if an accelerator is present. If so that accelerator is used immediately. If not a looped version of the function is used. This looped version then calls FPU functions above and uses the FPU if available.

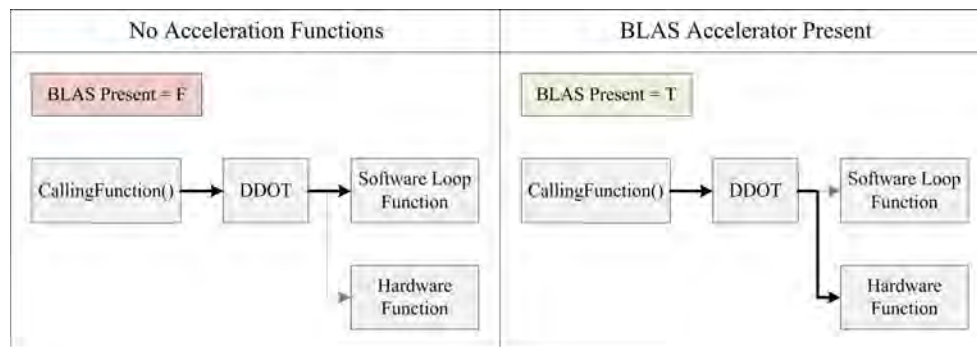


Figure 6.11: Decision Trees for BLAS Overloaded Operations

The functions implemented in the accelerator encompass most of the commonly used BLAS level 1 instructions.

Table 6.2: Implemented BLAS Level 1 Functions

Operation	Function	Description
$\text{argmax} \vec{A}$	idamax	Index of the largest vector element
$\text{argmin} \vec{A}$	idamin	Index of the smallest vector element
$\sum  A_i $	dasum	Sum of the vector magnitudes
$a\vec{B}$	daxpy	Scaler vector product
$\vec{A} \cdot \vec{B}$	ddot	Dot product
$\vec{A}b$	dscal	Vector scaler product

## TEST APPLICATIONS AND RESULTS

LINPACK BenchmarkBackground

The system was benchmarked using conventional tools to provide an indication of system performance and compare the platform with existing systems. The LINPACK benchmark was chosen as it allowed for testing floating point performance of the system with software floating point, floating point accelerator and the BLAS accelerator tiles.

The LINPACK benchmark performs a Gaussian elimination with partial pivoting on an  $n \times n$  matrix. This results in an operation count of  $2/3n^3 + n^2$  floating point operations. By modifying  $n$ , the memory requirements can be modified and the asymptotic performance of the architecture can be found. For this system, the asymptotic bound is beyond the memory the current hardware. Traditionally, 100 and 1000 are used as standard sizes for  $n$  [53].

Implementation

The system was implemented on the MicroBlaze utilizing the hardware accelerator hardware. The accelerations could be chosen through software flags and the performance results were printed over the UART terminal.

Results

During benchmark operation, the power consumption was recorded in addition to the performance to allow for performance and performance per watt to be determined.



The results of this test show that having an accelerator in all cases improves the performance of the system. Since the LINPACK benchmark heavily relies upon vector operations, use of the BLAS accelerator significantly improved performance over the basic FPU.

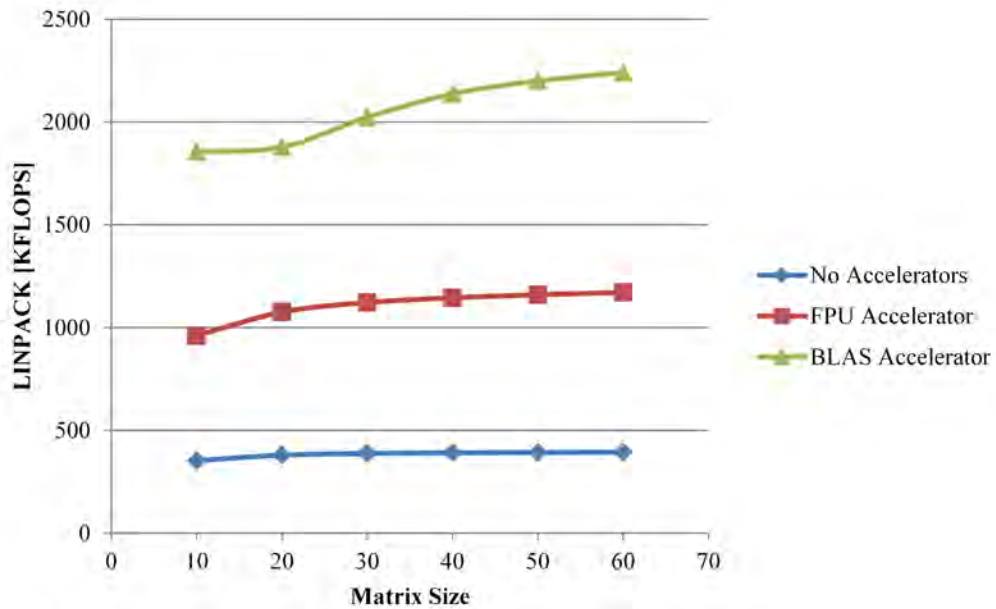


Figure 7.1: LINPACK Results vs Matrix Size and Enabled Accelerator

Much of the occupied time in these tests is spent moving data between the accelerator and the microprocessor. By placing the entirety of the matrix on the accelerator and operating on it in-place, the performance is increased dramatically. Due to the structure of the functions, this improvement is only achievable with the BLAS tile as the output memory location is a variable of the input function.

Performance is extrapolated so that  $n = 100$  to allow for comparison with published results. With these results, the system can be estimated to have

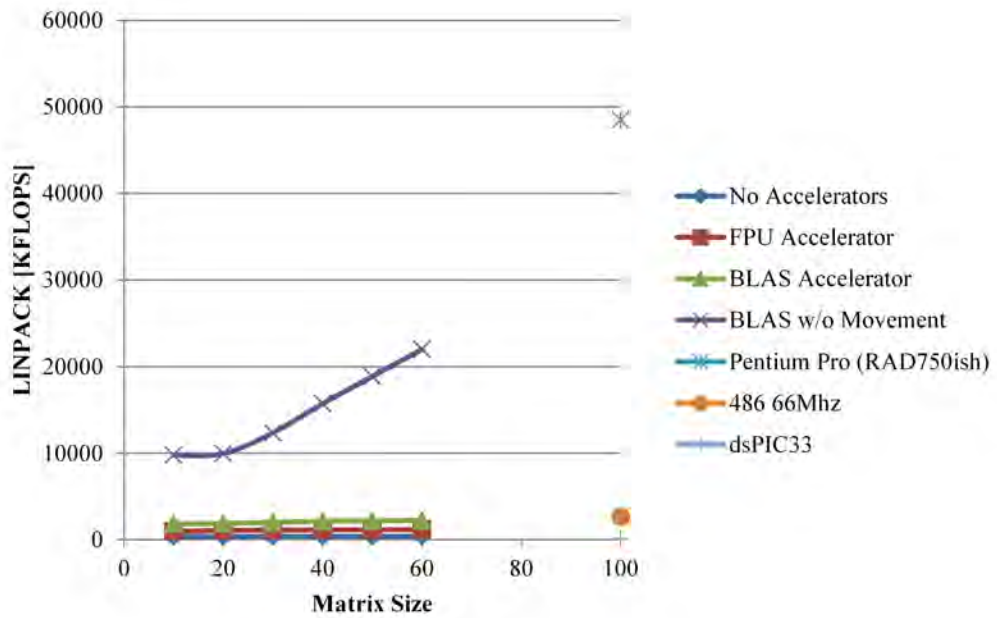


Figure 7.2: LINPACK Results vs Matrix Size and Enabled Accelerator with and without data movement

approximately the performance of a Pentium Pro, which is roughly comparable to a RAD750 in overall performance.

## RADIATION TOLERANT HARDWARE ANALYSIS

The system architecture was studied to allow for estimates to be calculated as to the availability and mean time to failure that the general system architecture would achieve in a space radiation environment. With these results, the different techniques to achieve radiation hardness in FPGAs could be quantitatively compared.

### Error Rate Analysis

Error rate for silicon based systems can be predicted through the use of an exponential distribution. For this analysis the radiation strike rate is assumed to be a Poisson process where  $\lambda$  can be computed with CREME96 and device fabrication parameters.

### CREME96 Simulations

CREME96 is a simulation tool designed by Vanderbilt University and the United States Navy. This tool is commonly used for simulating the radiation environment inside Earth's magnetosphere. While the tool does contain some data on deep space, it is less accurate in this environment.

Environmental inputs for CREME96 include solar activity parameters (Kp or solar wind/activity) and the orbital location. Device parameters input include the device size, and the number of memory elements. Spacecraft shielding parameters are also input into the simulation routines.

The error rate can then be calculated for the configuration memory and data memory and the device error rates extracted in terms of either *deviceErrors/deviceDay* or *bitErrors/deviceDay*.

### Computing Orbital Parameters

CREME96 orbital parameters are coded using McIlwain L-Parameters (also called L-Shells). As the orbital environment is reliant on the Earth's magnetic field, the radiation environment in orbit is the same along the same magnetic field line. These lines are named depending on their radius from Earth's center using an ideal dipole model (Figure 8.1).

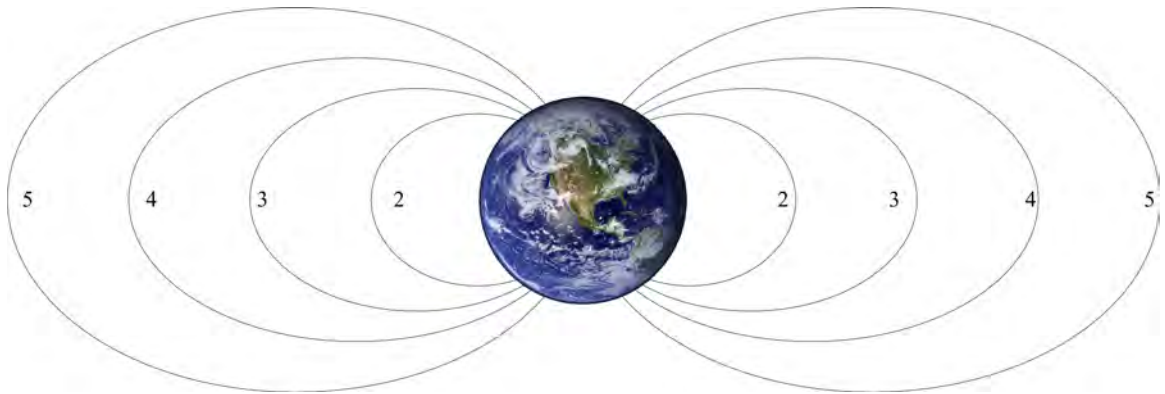


Figure 8.1: Ideal Dipole Model for Earth's Magnetic Field Lines and L-Shells

As Earth is not an ideal dipole, this method has been expanded to utilize the same concept with the International Geomagnetic Reference Field model (IGRF). This model takes into account the magnetic fields dipping lower at the South Atlantic Anomaly and models the magnetic field based on measured data.

In practice the higher of shells are compressed on the sun-facing side due to the effect of solar wind. This is added to the L-shell calculated with an external field model which adds solar activity and day/night to the conversion between altitude and L-Shell (Figure 8.2).

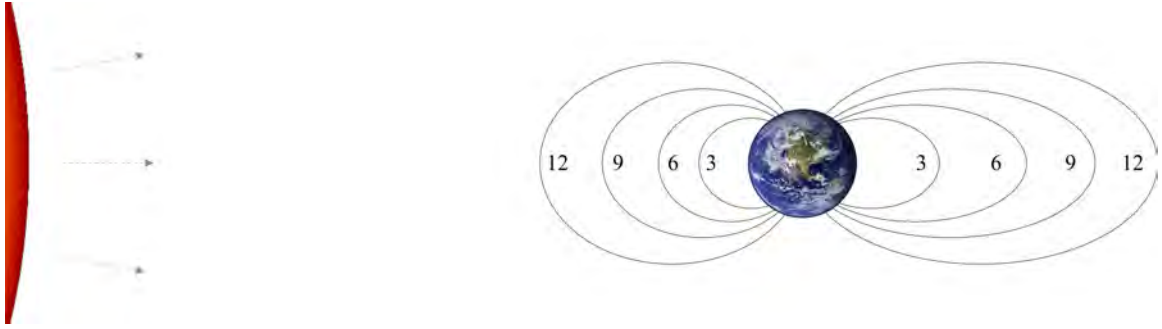


Figure 8.2: Impact of Solar Wind on Earth's Magnetic Field Lines

These parameters were computed for this research using an IGRF library, and the satellite positions were computed using a Simplified General Perturbation (SGP) library and publicly available two line element (TLE) files.

### Parametric Device Error Rate Analysis

For systems lacking repair, the error rate and Mean Time between Failures (MTBF) can be computed analytically using an exponential distribution. This assumes that the errors are distributed according to a Poisson process which is commonly accepted as a valid distribution for radiation induced faults.

The exponential distribution is defined as having the probability density function:

$$f(t, \lambda) = \begin{cases} \lambda e^{-\lambda t} & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (8.1)$$

The probability of an error occurring in time  $t$  is then modeled as the cumulative probability function:

$$F(t, \lambda) = \begin{cases} 1 - \lambda e^{-\lambda t} & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (8.2)$$

The mean time to an error is the value  $t$  where  $F(t, \lambda) = \frac{1}{2}$  which simplifies to:

$$MTBF = \frac{\ln(2)}{\lambda} \quad (8.3)$$

For repairable cases, Markov chain models were developed. The exponential distribution was used to validate the chains for non-repairable cases. This provided a second technique to ensure the validity of the more advanced techniques.

### Markov Reliability Analysis

#### Mathematical Model

To determine the failure rate for the system, a Markov chain was developed for a simplex system, a TMR system, a TMR system with scrubbing as well as the proposed TMR+scrubbing+spares system. The model was then run to determine the probability of failure based on fault rates.

The Markov Chain was developed where the state transition probabilities were entered in a square matrix  $Pr$  where [56]:

$$Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2 \dots) \quad (8.4)$$

An initial state vector ( $x$ ) was created, assuming the system always started in the good state:

$$x^{(0)} = [1, 0, 0, \dots, 0]^T \quad (8.5)$$

The state probabilities could then be computed for each time step with the iterative procedure:

$$x^{(n+1)} = x^{(n)} * P \quad (8.6)$$

which is equivalent to:

$$x^{(n)} = x^{(0)} * P^n \quad (8.7)$$

Mean time to system failure was modeled as the point where:

$$Pr(X_n | FAILED) = 0.5 \quad (8.8)$$

To determine the availability, the long term state probabilities were then approximated using:

$$x^{(n)} = \lim_{n \rightarrow \infty} x^{(0)} * P^n \quad (8.9)$$

However, for this system at infinite time the probability of failure is 100%. To more realistically model availability an external system watchdog circuit was simulated by added a path in the  $Pr$  matrix from failed back to a good state (a watchdog timer) with an associated probability of restart at each time step.

To estimate the fault rates for orbits, the fabrication node information and orbital locations were found and CREME96 was used to compute the fault rates and entered into this chain.

The system assumptions used in this simulation study were that the contact switch to a spare tile was 2 ms, the scrubbing time was 100 ms and that the system was a nine tile MicroBlaze system. In addition it was assumed that 70% of the area in a tile was occupied by logic.

### Markov Chain with No Radiation Hardening

A simplex system model was the simplest Markov chain studied. In this simulation, a MicroBlaze nine tile system was operated with a single tile and without TMR or scrubbing activated so in the case of a fault the system transitions from State 0 (Active) to State 1 (Faulted) [57].

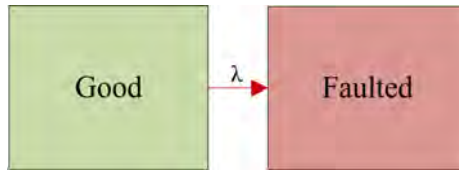


Figure 8.3: Markov State Transition Diagram - Simplex System

$Pr(X_n + 1)$  can then be generated where  $\lambda$  is the FPGA device fault rate.

$$Pr(X_n + 1) = \begin{bmatrix} 1 - \lambda A \Delta T & \lambda A \Delta T \\ 0 & 1 \end{bmatrix} \quad (8.10)$$

### Markov Chain with TMR

This simulation implements the tiled system with a TMR. The system operates normally until such time as two errors have occurred. This should result in an improved reliability of the system but will not improve the MTBF due to the increased area occupied by this system [57].

The transition matrix for this configuration is modeled as:

$$Pr(X_n + 1) = \begin{bmatrix} 1 - 3\lambda A \Delta T & 3\lambda A \Delta T & 0 \\ 0 & 1 - 2\lambda A \Delta T & 2\lambda A \Delta T \\ 0 & 0 & 1 \end{bmatrix} \quad (8.11)$$





Figure 8.4: Markov State Transition Diagram - TMR System without scrubbing

### Markov Chain with TMR and Scrubbing

For the third simulation the system was operated in TMR mode with a memory scrubber. To fail, the system must experience two failures prior to a scrub event completing. This scrubbing activity causes MTBF to increase as the system will continue to operate as long as two faults do not occur during the repair time, rather than two faults during overall system operation.



Figure 8.5: Markov State Transition Diagram - TMR System with scrubbing

With this  $Pr(X_n + 1)$  can be generated where State 0 = Active, State 1 = One Tile Faulted and State 2 = Failure [57]. The variable  $R$  is the scrubber repair rate.

$$Pr(X_n + 1) = \begin{bmatrix} 1 - 3\lambda A \Delta T & 3\lambda A \Delta T & 0 \\ \mu \Delta T & 1 - 2\lambda A \Delta T - \mu \Delta T & 2\lambda A \Delta T \\ 0 & 0 & 1 \end{bmatrix} \quad (8.12)$$

Due to the limited FPGA resources available in the development hardware, this was the radiation hardening technique used for the implemented system.

### Markov Chain with TMR, Scrubbing and Spares

The final Markov chain was a TMR+Scrubbing+Spares system with nine total tiles. In the case of a fault, the system rapidly switches the faulted tile to a spare and reduces the number of spares. The system must either experience two faults prior to processor swap completing or run out of spares to result in the failure state being observed. This further improves MTBF by decreasing the effective repair time while the system has spares available.

This Markov chain adds the concept of a fast context switch between tiles noted by the variable  $\nu$ , measured in *ContextSwitches*/ $\Delta T$ .

The FPGA used for this research lacked sufficient resources to implement this technique. The system was designed such that future hardware revisions can use this technique instead of TMR+Scrubbing to further improve reliability.

The initial good state is modeled as [57]:

$$Pr(X_n + 1, good) = \begin{bmatrix} 1 - 3\lambda A \Delta T & 3\lambda A \Delta T & 0 \\ \mu \Delta T & 1 - 2\lambda A \Delta T - \mu \Delta T & 2\lambda A \Delta T \\ 0 & 0 & 1 \end{bmatrix} \quad (8.13)$$

From a spare fault state, the transition to repair the fault is modeled as:

$$\mu \Delta T \quad (8.14)$$

The probability of transitioning to a fault on an active tile is:

$$\#Active\lambda A\Delta T \quad (8.15)$$

And the probability of encountering a strike on an additional spare tile is modeled as:

$$\#Spares\lambda A\Delta T \quad (8.16)$$

From an active tile fault, the probability of a context switch to a spare tile is:

$$\nu\Delta T \quad (8.17)$$

While the probability of an additional fault on an active tile failing the system is:

$$2\lambda A\Delta T \quad (8.18)$$

The probability of a strike on a spare tile prior to a context switch is also modeled as:

$$\#Spares\lambda A\Delta T \quad (8.19)$$

For all states, the probability of remaining in the same state is defined such that the sum of all probabilities of the state is equal to 1.

## Results

The Markov models were run with the CREME96 fault rates for a number of orbital conditions to estimate the MTBF and availability of the systems.

The simulation shown in Figure 8.7 shows that for an orbit during average solar conditions in a low earth orbit, the use of either TMR+scrubbing or

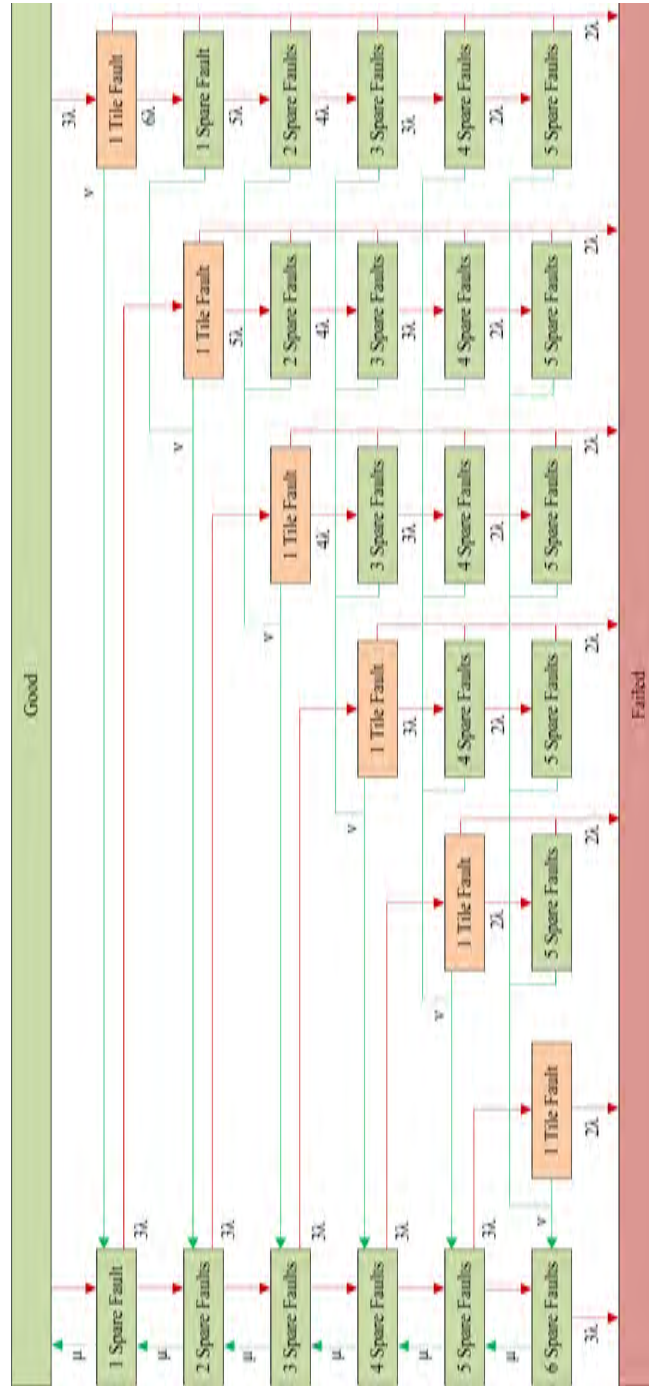


Figure 8.6: Markov State Transition Diagram - TMR with Spares and Scrubbing

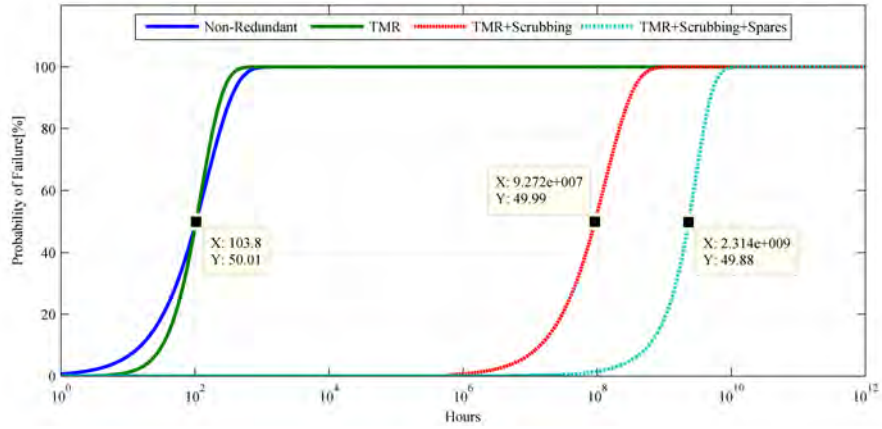


Figure 8.7: Mean time between failures for the system in a LEO orbit during average solar conditions

TMR+scrubbing+spares is sufficient to result in a near infinite mean time to failures. Use of any system lacking scrubbing results in a MTBF which would not be appropriate for any mission as the system would require restarts on average of twice per day.

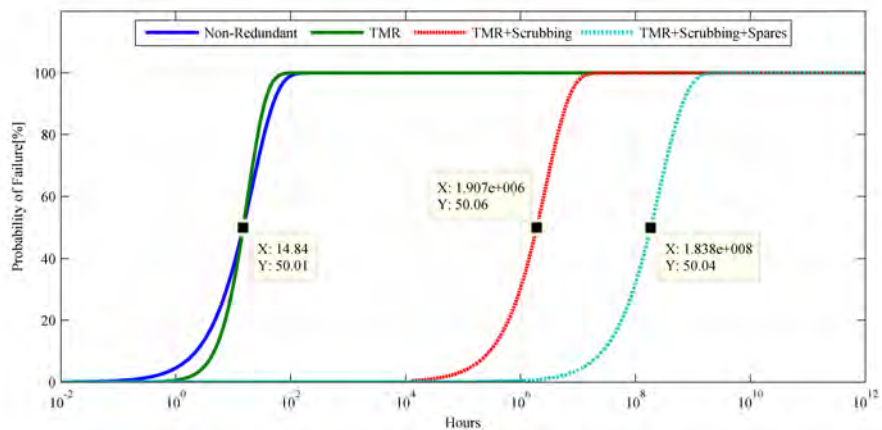


Figure 8.8: Mean time between failures for the system passing through the SAA in a LEO orbit during average solar conditions

Figure 8.8 shows the predicted mean time to failure for FPGA systems operating in the South Atlantic Anomaly during average solar conditions.

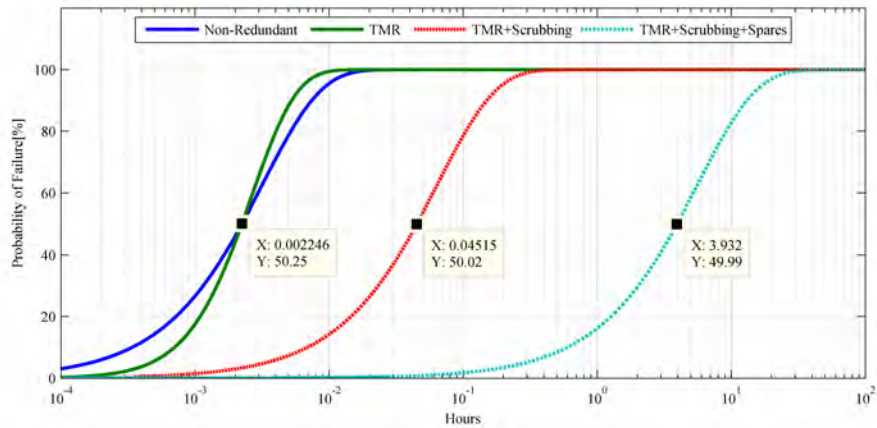


Figure 8.9: Mean time between failures for the system in the worst orbital section during the worst week of a solar maximum in an ISS orbit

During worst week of the solar max conditions (Figure 8.9) MTBF values are decreased dramatically. In this situation the systems without scrubbers fail instantly, while TMR+Scrubbing only survives 1.35 minutes. TMR+Scrubbing+Spares continues to operate for over an hour resulting in a far greater likelihood of the computer surviving the flare condition.

In the simulation shown in Figure 8.10, the system is being subjected to an intense flare while in the SAA. None of the systems are capable of operating under these conditions and the system would need to be restarted by a harder watchdog circuit after the event.

These results show that during average solar conditions a simple TMR+Scrubbing system is capable of providing the necessary radiation hardening for operation in a LEO orbit. During worst case situations, the TMR+Scrubbing+Spares has a greater lifetime and is more likely to survive the transit through the worst case portions of

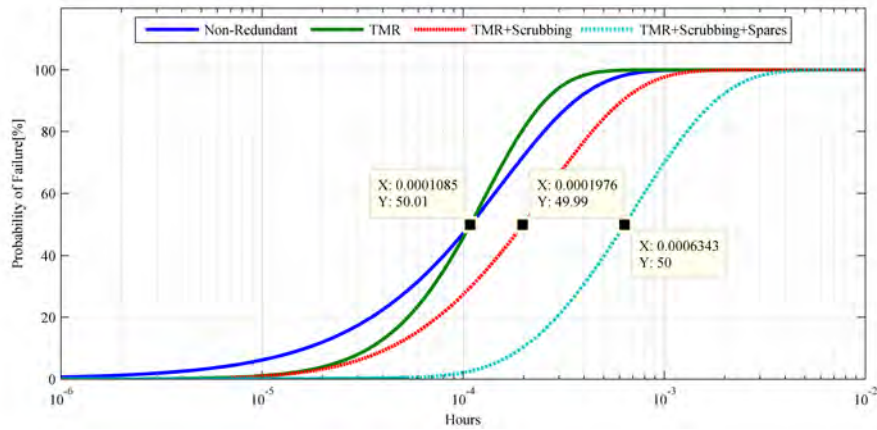


Figure 8.10: Mean time between failures for the system in the worst orbital section during the worst 5 minutes of a solar flare in an ISS orbit.

the orbit without failing. The addition of the fast context switches is a significant improvement and allows the system to respond as though the partial reconfiguration operation was much faster.

The Markov chains were also run to determine the operating availability of the systems. For these simulations a two minute watchdog counter was added to the model to allow for recovery in the case of complete system failure, and long term values were found for the states in which the system is processing data.

Figure 8.12 shows that the decrease in time involved to switch to a new tile in the TMR+Spares+Scrubbing system results in the system having much greater availability during periods of moderately high solar activity and would likely be able to meet the computational needs of the mission.

The simulation shown in Figure 8.13 shows that the non-redundant system very loses its high availability when any significant SEE rate is present. The TMR+Scrubbing system maintains availability until SEE rates exceed 1 SEE/ms.

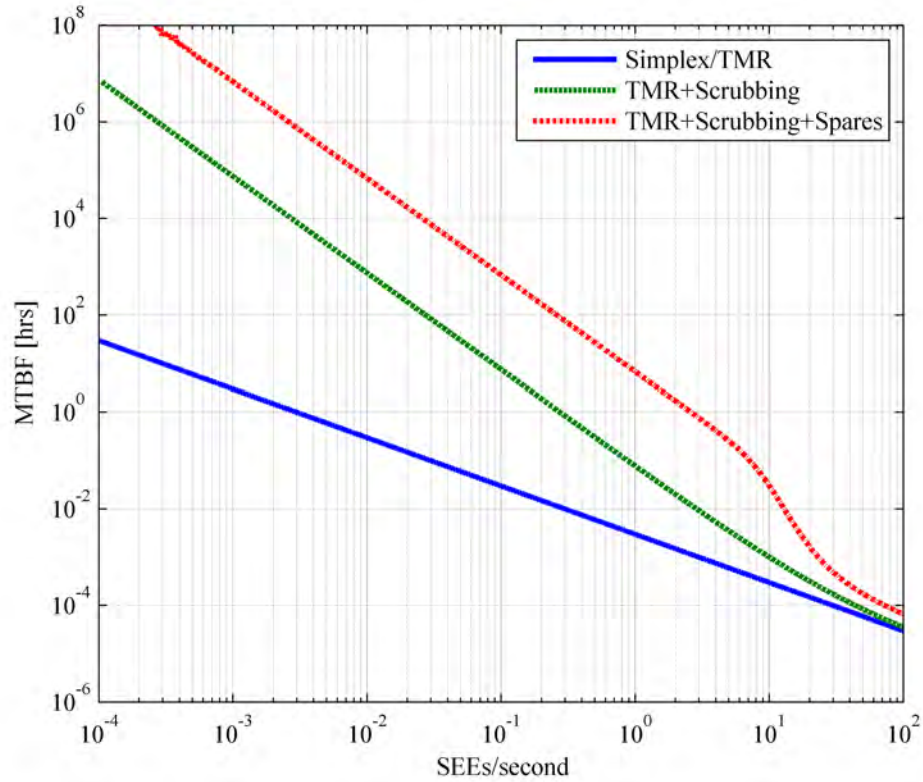


Figure 8.11: Simulation of the MTBF for the different systems studied vs fault rate

The TMR+Scrubbing+Spares system by comparison, achieves high availability until SEE rates exceed 20 SEEs/ms.

The availability of the system is improved over the alternative options due to the processing being continued after a short context switch rather than waiting for a PR operation to complete. This effect is especially dramatic as fault rates increase. Use of a non-redundant system is shown to not be a viable solution to operate in any radiation prone environment.



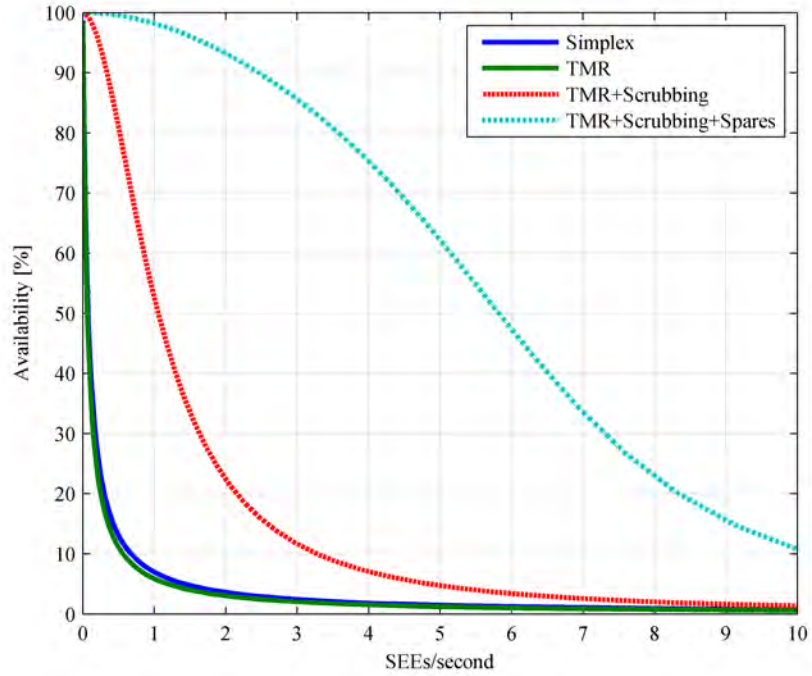


Figure 8.12: Comparison of system reliability vs fault rate

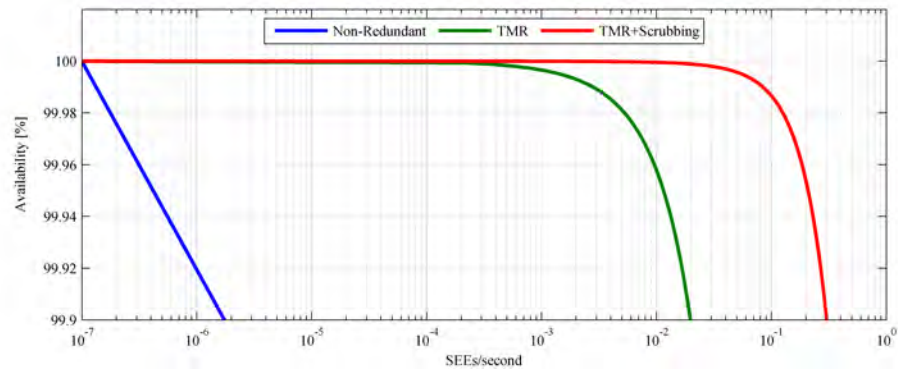


Figure 8.13: Simulation showing estimated fault rate vs reliability

The TMR+Scrubber+Spares architecture has higher availability and reliability than conventional FPGA TMR techniques and make the system a viable solution for allowing the use of the computational abilities of FPGAs in harsh environments.

## Orbital Fault Injection Simulator

To bench test the reaction of the system for realistic orbits a test routine was developed to model the error rate and frequency of radiation strikes in earth orbits. Using CREME96 data, orbital simulations, and magnetic field line solvers we were able to both compute satellite orbits and approximate the error rate at that location in orbit. By running this simulation connected to a hardware fault injector the system's ability to respond to radiation can be demonstrated in realistic environmental conditions.

The CREME96 data and the orbital mechanics were programmed into a PC based interface to inject faults into the tile FPGA (Figure 8.14). In the case of an error the system communicated the presence of a fault to the Spartan-6 control FPGA through a serial link. This system could also induce FPGA faults through the FPGA sensors and monitor system parameters such as voltage and temperature.

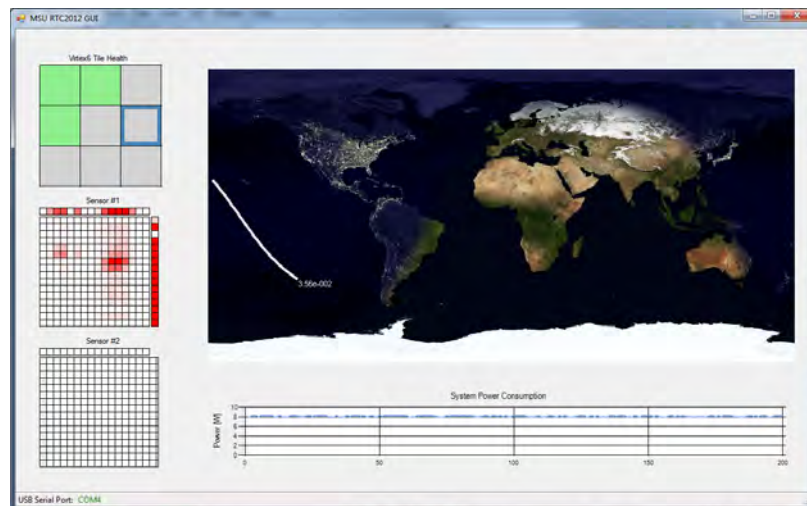


Figure 8.14: Orbital Radiation Fault Injection GUI with radiation sensors for a nine tile MicroBlaze System

## SYSTEM POWER AND PERFORMANCE ANALYSIS

Introduction

Performance and power are broken into two sections, the steady state performance of the microprocessor with a chosen accelerator and the time and power required to configure the accelerator. Should the system consume more power than is saved by using acceleration hardware, or if the hardware consumes more time than is saved, the acceleration hardware is not advantageous. This section provides an analysis of this power/time/hardware tradeoff.

Ideally, the system can perform any operation with the microprocessor given enough time and the total power consumption is represented by the area under the curve (Figure 9.1). When the microprocessor has completed its operation it returns to a low power state.

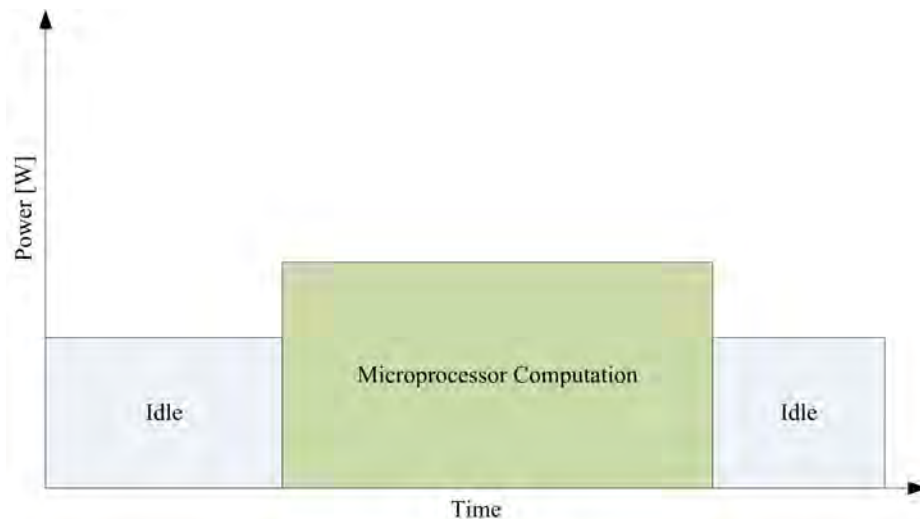


Figure 9.1: Expected power consumption with only a microprocessor

When using acceleration hardware, this analysis is modified to have a partial reconfiguration time followed by a shorter, higher power computation time and the system returning to an idle state (Figure 9.2). As long as the area under the partial reconfiguration and computation section is smaller than the microprocessor-only section, the partially reconfigurable hardware is advantageous in the total system operations.

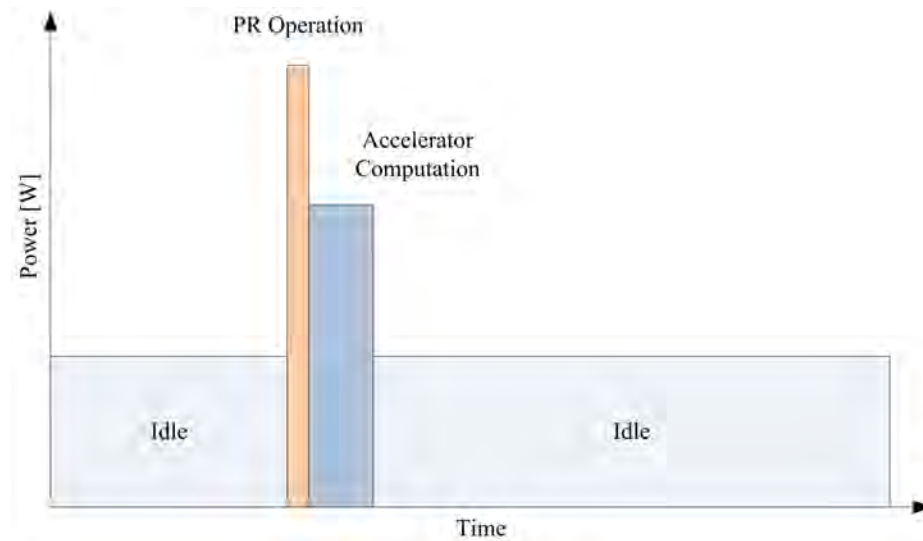


Figure 9.2: Expected power consumption with only a partially reconfigurable hardware accelerator

### Simplex System

#### Design Layout

The system and accelerators were placed on the FPGA fabric in separate tiles to allow for the tiles to be configured independently. This system layout has the potential for the greatest performance and resource utilization, but has limited radiation hardening due to the lack of TMR and no resource sharing.

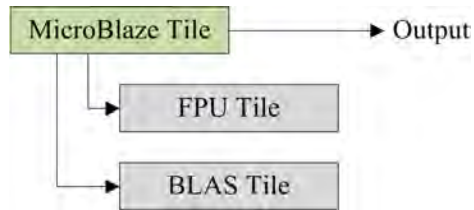


Figure 9.3: Block Diagram of a partially reconfigurable system with hardware accelerators

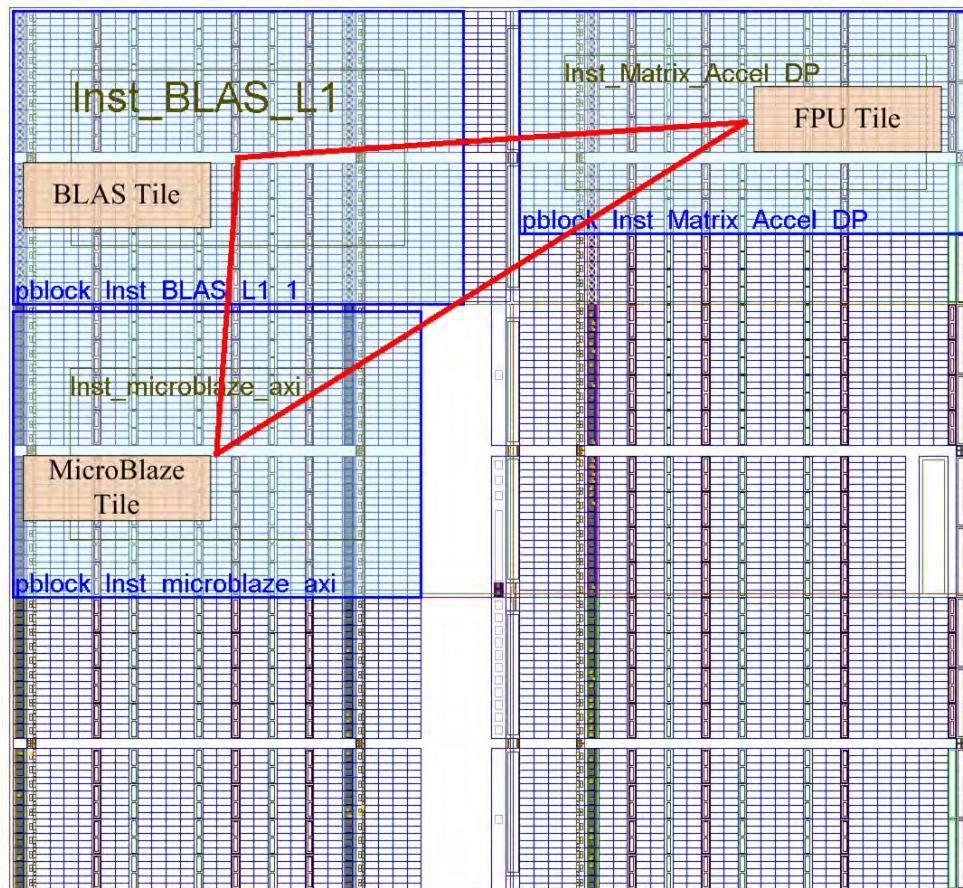


Figure 9.4: Floorplan of a partially reconfigurable system with hardware accelerators implemented on the Virtex-6 FPGA



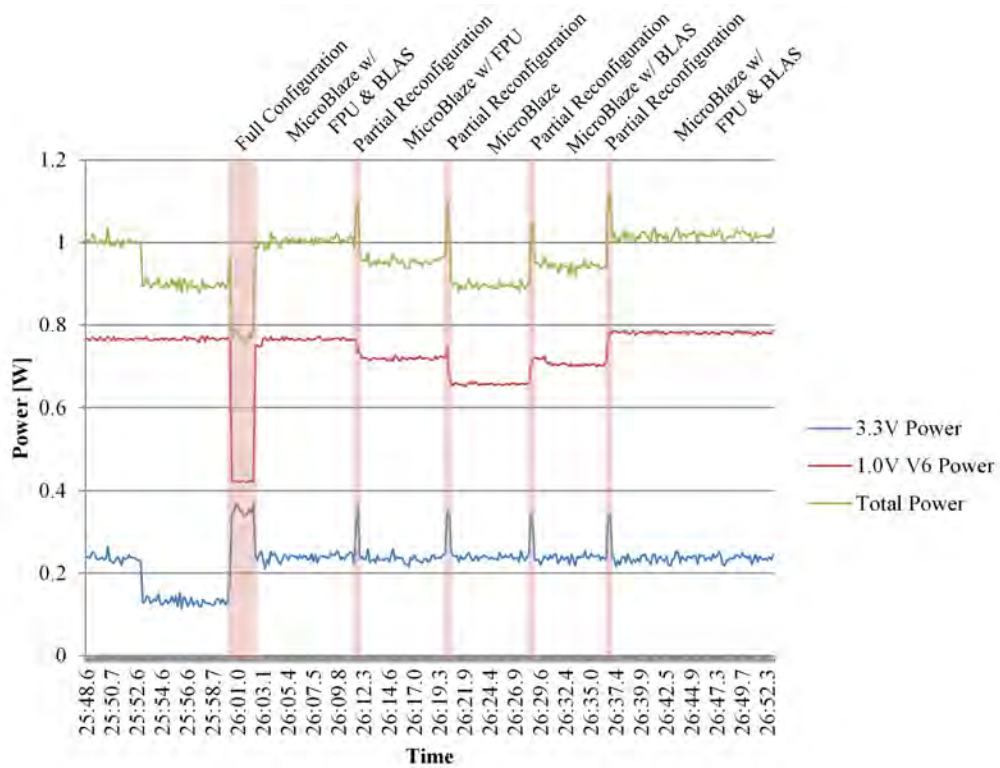


Figure 9.5: Measured power consumption showing power consumed by the PR operation and power saving by eliminating unnecessary acceleration hardware

This data was analyzed and added to a MATLAB simulation. By running this model, the power consumption of the system under different operating systems can be quickly analyzed. To initially verify the model, the same operating conditions as the test conditions was simulated. This allows for average power to be quickly estimated and for the power consumption to be determined without requiring extensive lab tests.

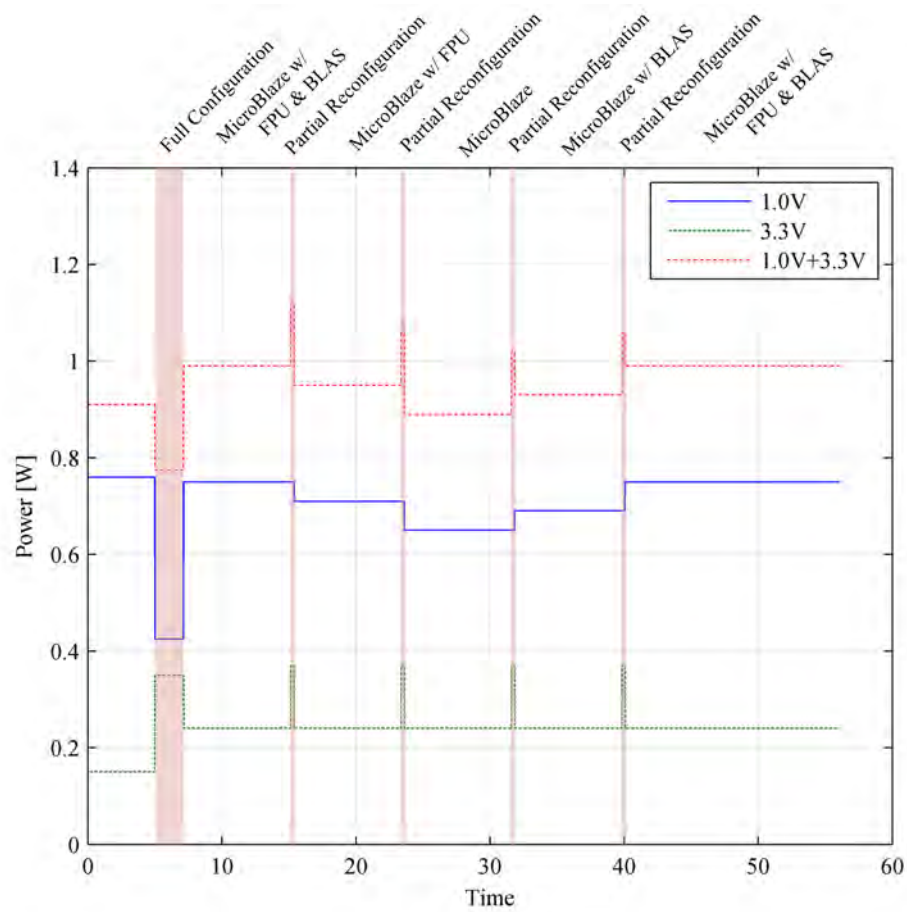


Figure 9.6: Simulation of the power consumption showing power consumed by the PR operation and power saved by eliminating unnecessary acceleration hardware

### Partial Reconfiguration Timing and Power Consumption

The initial analysis was of the time and power consumed by the partial reconfiguration task. The reconfiguration time can be modeled by computing the time to read the bitstream from for the Spartan-6 to read the bitstream from the SD card and the time to write the bitstream from the Spartan-6 to the Virtex-6 configuration memory.

The average power consumption for the configuration is a product of the SD card read time and power as well as the configuration interface write time and power:

$$P_{config} = P_{config} * T_{writeSelectMap} + P_{config} * T_{readSD} \quad (9.1)$$

The reconfiguration timings were measured to program the full bitstream and partial bitstreams for the Virtex-6 and these values were measured and recorded in Table 9.1:

Table 9.1: Power and Time for FPGA Configuration

	Time [sec]	Average Power [W]
Full Reconfiguration	2.139	0.119
Partial Reconfiguration	0.233	0.089

In addition the power consumed for the system with the un-programmed Virtex-6 and the different programming modes was recorded.

Table 9.2: Power Consumed to Perform Full Reconfiguration

	Un-Programmed		Full Reconfiguration		Difference	
	mA	mW	mA	mW	mA	mW
3.3V	90	297	90	297	0	0
2.5V	30	75	60	150	30	75
1.8V	80	144	60	108	-20	-36
1.0V S6	80	80	120	120	40	40
1.0V V6	390	390	410	410	20	20
Total		986		1085		119

As the power consumption on the Virtex-6 during partial reconfiguration is dependent on the design running on the remainder of the chip, the difference introduced by this operation was measured.



Table 9.3: Power and Time for FPGA Configuration

	$\Delta$ Current [mA]	$\Delta$ Power [mW]
3.3V	0	0
2.5V	30	75
1.8V	-20	-36
1.0V S6	30	30
1.0V V6	20	20
Total		89

From these tests, performing a full FPGA reconfiguration consumes 119mW of additional power over an idle Virtex-6, and performing a partial reconfiguration consumes an additional 89 mW.

### System Power Analysis

During the operation of the FPGA design, the power consumption can be modeled on a tile by tile basis. Average power consumption was measured for each tile type when in idle or active states.

Table 9.4: Power consumed by the tiles based on activity

Tile Type	Average Power
Unconfigured	410 mW
MicroBlaze Tile	+490 mW
FPU Tile	+50 mW
BLAS Tile	+70 mW

With this data having been collected, the system power consumption during runtime can be approximated by summing the respective tiles and tile states where system supports where  $n = 9$  for the implemented system and  $P_{other}$  includes the non-tile related FPGA logic, including the DCM and tile interconnects.

$$P_{operation} = t_{operation} * P_{other} + \sum_{tiles=1}^n t * P_{tileState} \quad (9.2)$$

### Performance Analysis

To model the timing performance improvements of the system, an average performance improvement for each tile was found and averaged from Chapter 5.

Table 9.5: Potential Speedup vs Tile Type

Tile Type	MFLOPS	Speedup
MicroBlaze Tile	0.4	1
FPU Tile	5.6	14
BLAS Tile	10.8	27

The performance improvement provided by the tile can then be modeled using Amdahls law where [58]:

$$Speedup = \frac{ExecutionTime_{old}}{ExecutionTime_{new}} \quad (9.3)$$

The point where the new system exceeds the performance of the old system is then where the ratio is equal to 1. A speedup less than 1 indicates a loss of performance and a speedup ratio greater than 1 is an improvement in system performance.

With this data, the system speedup with partial reconfiguration can be found dependent on the number of floating point operations as:

$$Speedup = \frac{n * FLOPS_{noAccel}}{n * FLOPS_{accel} + T_{PR}} \quad (9.4)$$

This data shows that for small numbers of operations the partial reconfiguration overhead is not practical, but as  $n$  increases the speedup becomes substantial.

Table 9.6: Speedup vs tile type with partial reconfiguration time

Tile Type	Speedup			
	$n=1$	$n = 1e3$	$n = 1e6$	$n = 1e9$
MicroBlaze Tile	1	1	1	1
FPU Tile	1.235e-5	1.24e-2	1.26	1.34
BLAS Tile	1.235e-5	1.24e-2	2.22	2.70

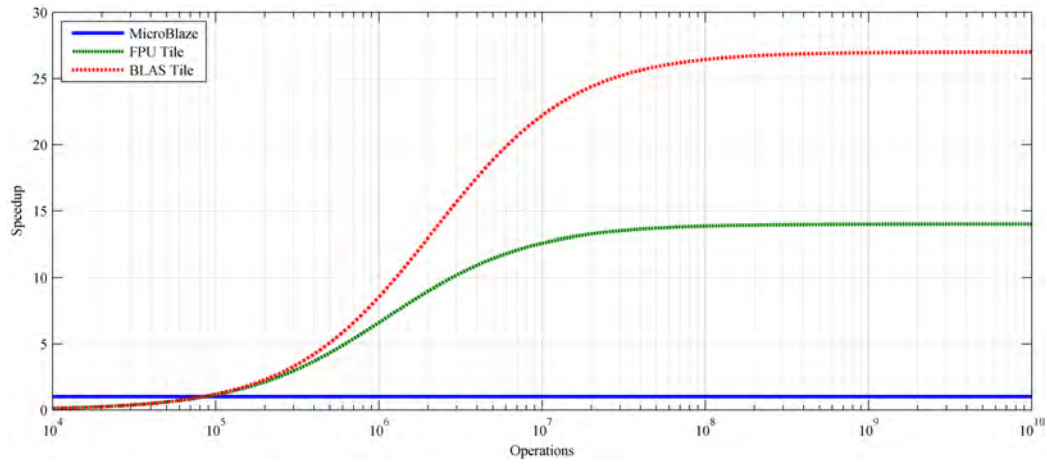


Figure 9.7: Speedup achieved vs the MicroBlaze using accelerators based on number of floating point operations with partial reconfiguration time

### System Analysis

Combining the processing time and state analysis with the performance analysis, Figure 9.8 can be generated to show the conditions under which tiles will result in improved performance, reduced power consumption or both. Depending on the implementation details of the hardware, this tradeoff point will vary. Once online, the FPU and BLAS tiles use dramatically less power to perform operations.

The assumption in figure 9.8 that the MicroBlaze is idle while the partial reconfiguration is occurring can be avoided in many situations by requesting the tile prior to it being required. In this case, the MicroBlaze can be occupied on another

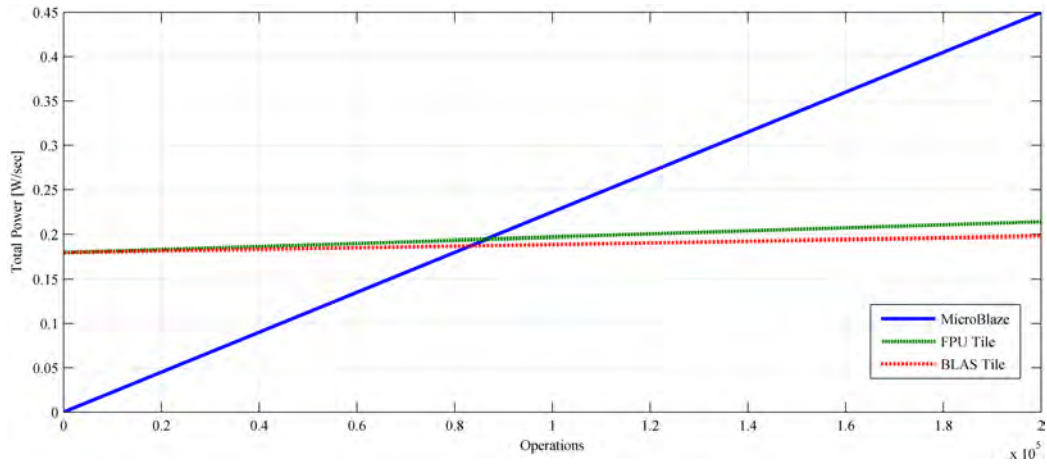


Figure 9.8: Power consumed for an operation on the simplex system

required task and the idle power from the time is not counted against the total power to complete this operation. If this can be achieved, the power penalty is greatly reduced (Figure 9.9).

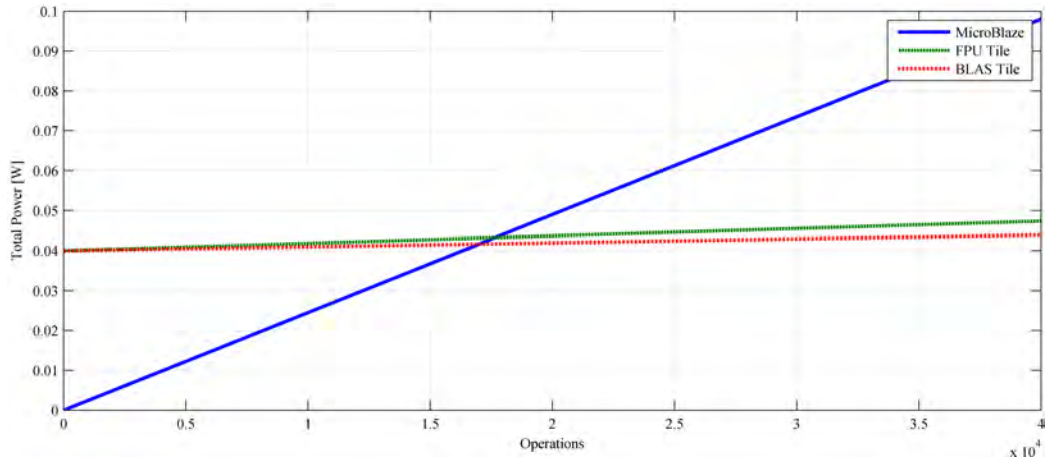


Figure 9.9: Power consumed for an operation normalized to the MicroBlaze processor time without the microprocessor idle power during PR counted in the total operation power consumption

The power required to perform the operation compared to the MicroBlaze is shown in Figure 9.10. This demonstrates that power can be conserved through use of the additional specialized hardware.

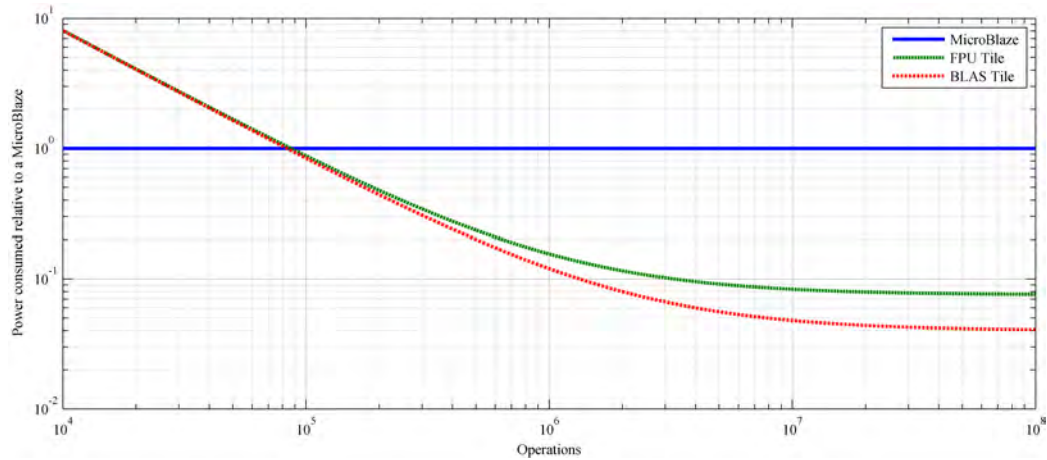


Figure 9.10: Power consumed for an operation normalized to the MicroBlaze processor

Finally, the power efficiency per MFLOP of computation was performed (Figure 9.11). This analysis shows that as the number of operation increases, the power efficiency increases for all but the MicroBlaze system. This is due to the increased performance of the accelerator tiles being more efficient per operation, but requiring a fixed amount of power to start.

#### Power Consumption vs Fault Rate

In the final analysis the simplex system was simulated to evaluate the power increase caused by performing partial reconfiguration on the tiles at varying rates. For reconfiguration rates sufficient to manage realistic space radiation environments the additional power required by the system is negligible.

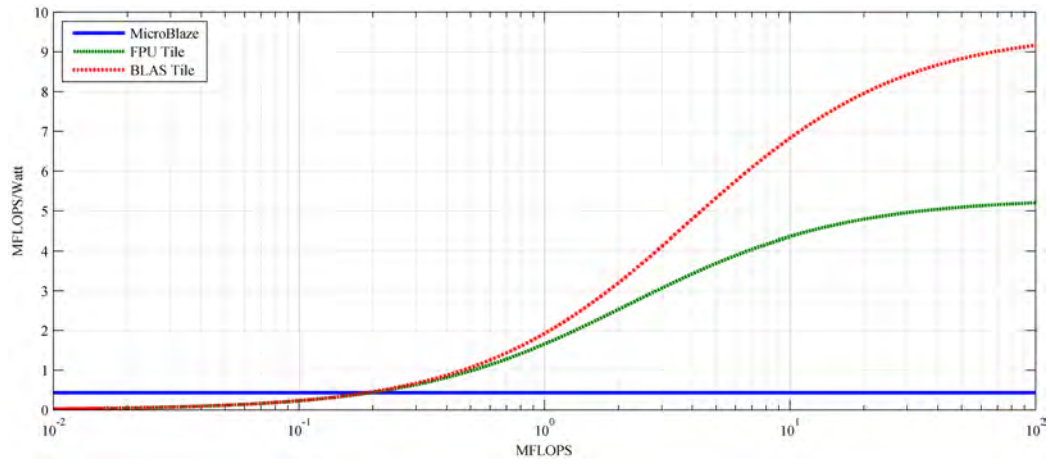


Figure 9.11: Power efficiency for the TMR system vs the size of the operation in MFLOPS

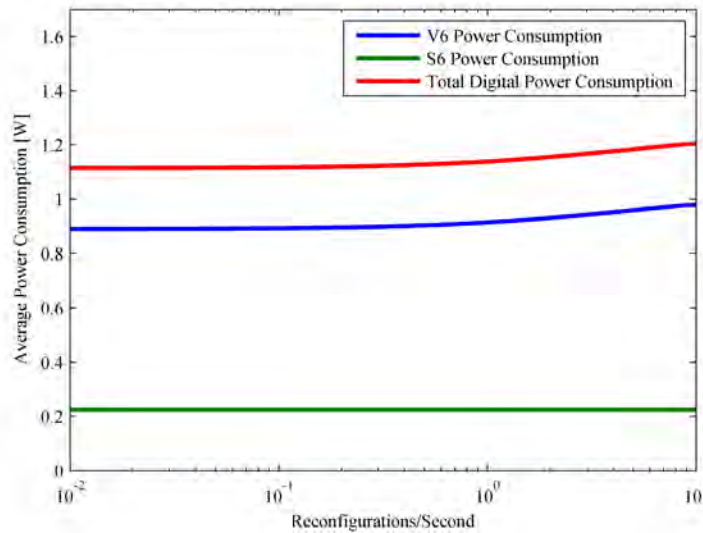


Figure 9.12: Power consumed by the system based on partial reconfiguration rates

## MicroBlaze System Analysis with Accelerators and TMR

### Design Layout

After initial analysis and development was completed on the simplex system, a TMR version of the design was developed using three MicroBlaze processors and agnostic acceleration tiles.

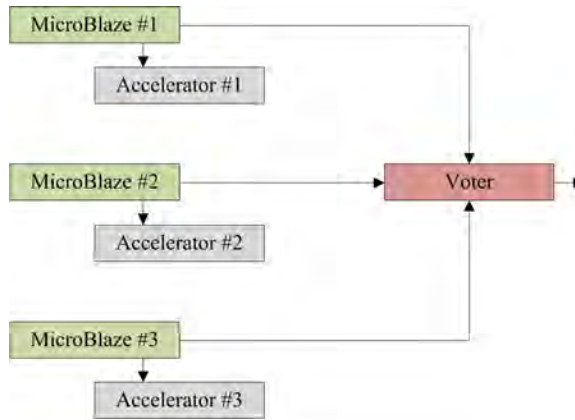


Figure 9.13: Block Diagram of a partially reconfigurable TMR system with hardware accelerators

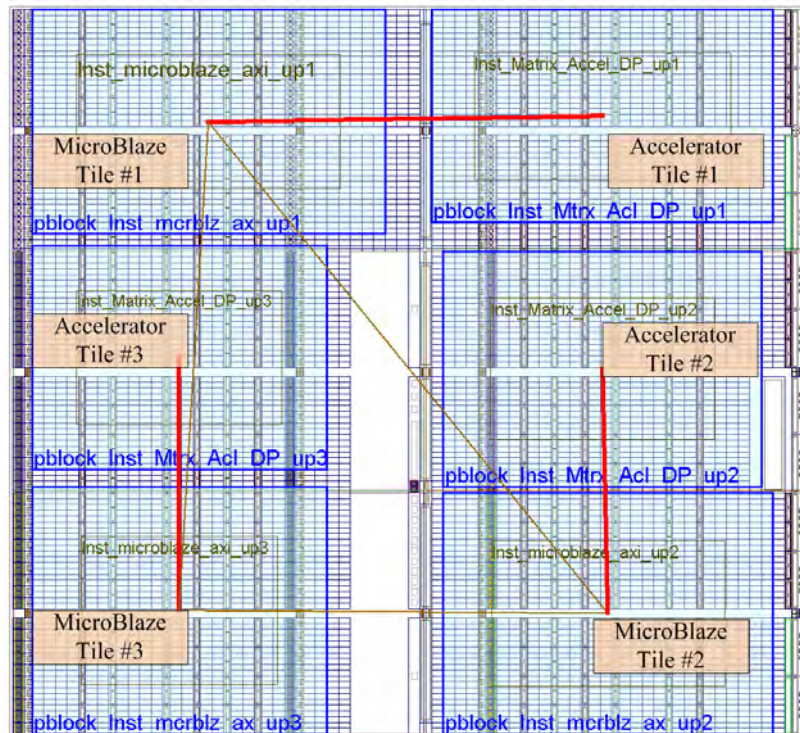


Figure 9.14: Floorplan of a TMR MicroBlaze partially reconfigurable system implemented on the Virtex-6 FPGA

While this system is functionally similar to simplex system, the increased device utilization required relaxed timings to be implemented, slowing the overall system performance.

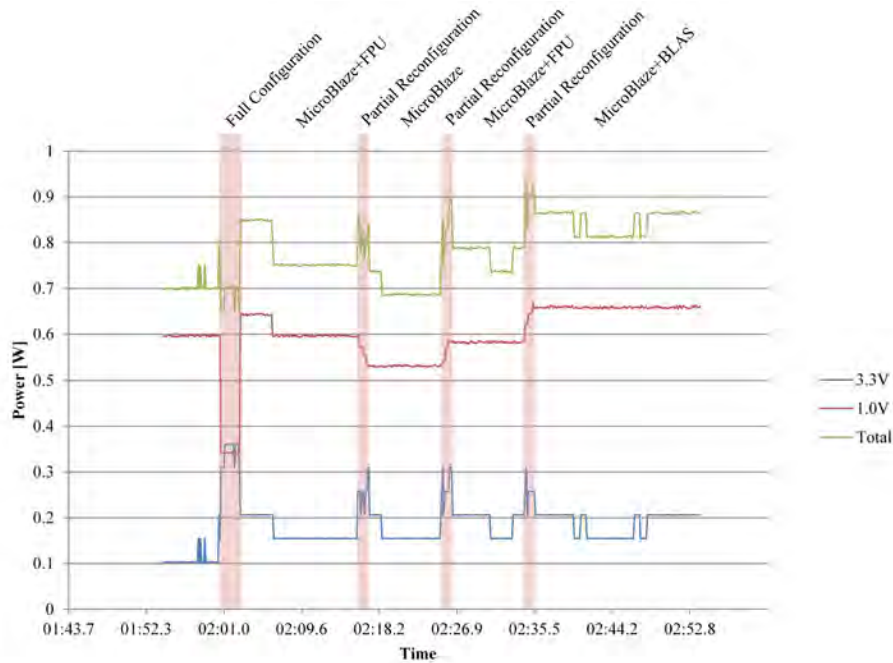


Figure 9.15: Measured power consumption showing power consumed by the PR operation and power saving by eliminating unnecessary acceleration hardware on a TMR system

### System Power Analysis

Due to the greater resource utilization of this device, timing constraints needed to be relaxed to meet timing. This leads to reduced power consumption on a tile by tile basis for the tested system.

The reconfiguration task consumed the same power as the simplex system, but due to the triplicated hardware required three times longer to perform the task.



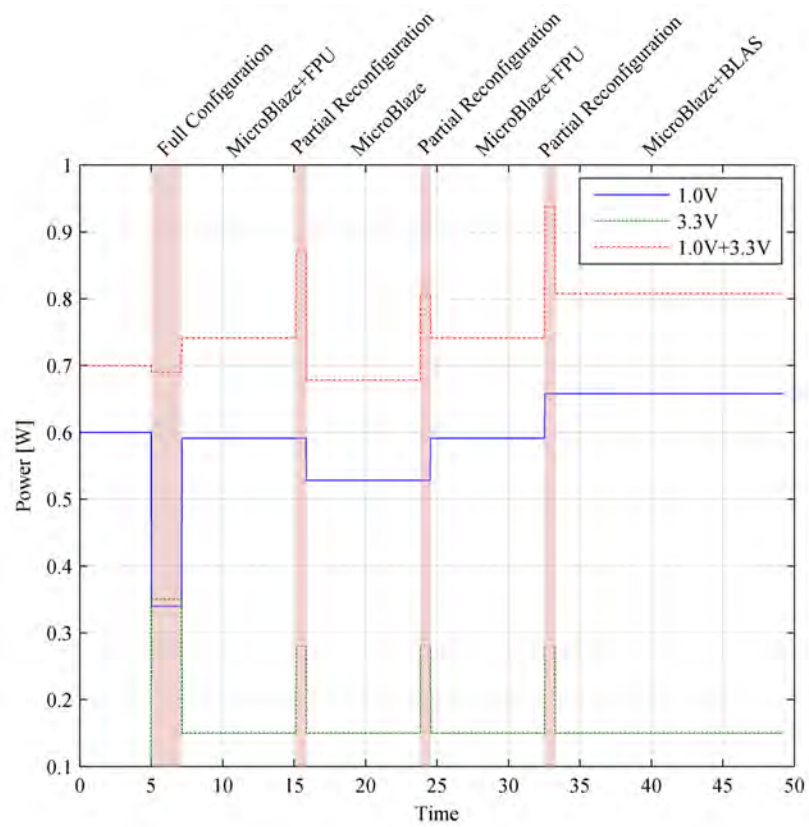


Figure 9.16: Simulation of the power consumption showing power consumed by the PR operation and power saving by eliminating unnecessary acceleration hardware in a triplicated system

Table 9.7: Power consumed by the tiles based on activity for the TMR system

Tile Type	Average Power
Unconfigured	410 mW
MicroBlaze Tile	+280 mW
FPU Tile	+100 mW
BLAS Tile	+160 mW

### Performance Analysis

The slower clock rates of this system necessitated recalculating the performance achievable by the tiles. The MicroBlaze was reduced to 125MHz while the FPU and BLAS clocks were reduced to 300MHz.

Table 9.8: Potential Speedup vs Tile Type for the TMR system

Tile Type	MFLOPS	Speedup
MicroBlaze Tile	0.3	1
FPU Tile	4.2	14
BLAS Tile	8.1	27

When longer partial reconfiguration times were added, this resulted in (Table 9.9):

Table 9.9: Speedup vs tile type with partial reconfiguration time for the TMR system

Tile Type	Speedup			
	n=1	$n = 1e3$	$n = 1e6$	$n = 1e9$
MicroBlaze Tile	1	1	1	1
FPU Tile	1.8e-5	1.78e-2	1.39	1.51
BLAS Tile	1.8e-5	1.24e-2	2.51	2.92

The result of this accelerator’s performance vs data size are very similar to the earlier case. The additional partial reconfiguration time required to reconfigure three tiles increases the data size required to offset the bitstream swap making it less advantageous unless large amounts of data are required.

### System Analysis

The tradeoff point for power is also greatly increased by the partial reconfiguration time. Once online the efficiency of the accelerator tiles uses far less power per floating point operation.

Compared as a ratio to the MicroBlaze’s power consumption, it is apparent that the accelerator systems can result in lower system power consumption in some instances.

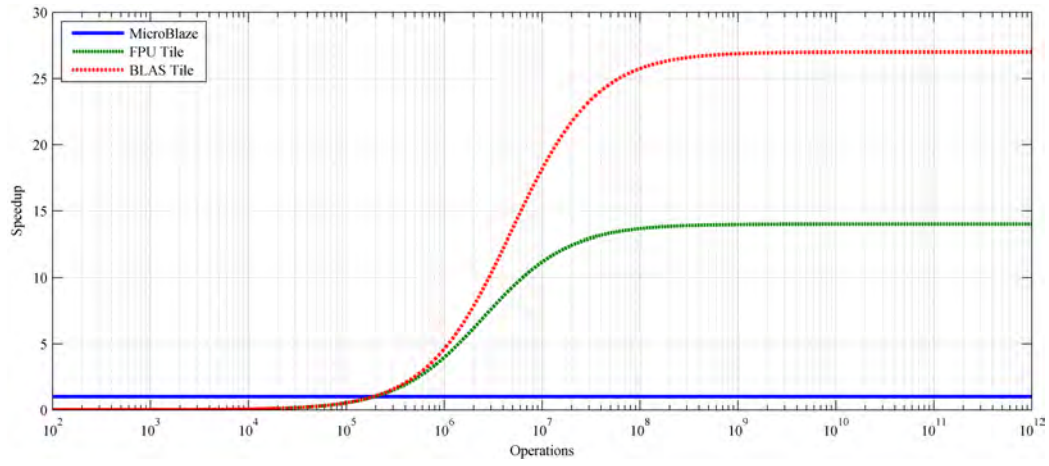


Figure 9.17: Speedup over the MicroBlaze using accelerators based on number of floating point operations with partial reconfiguration time for the TMR system

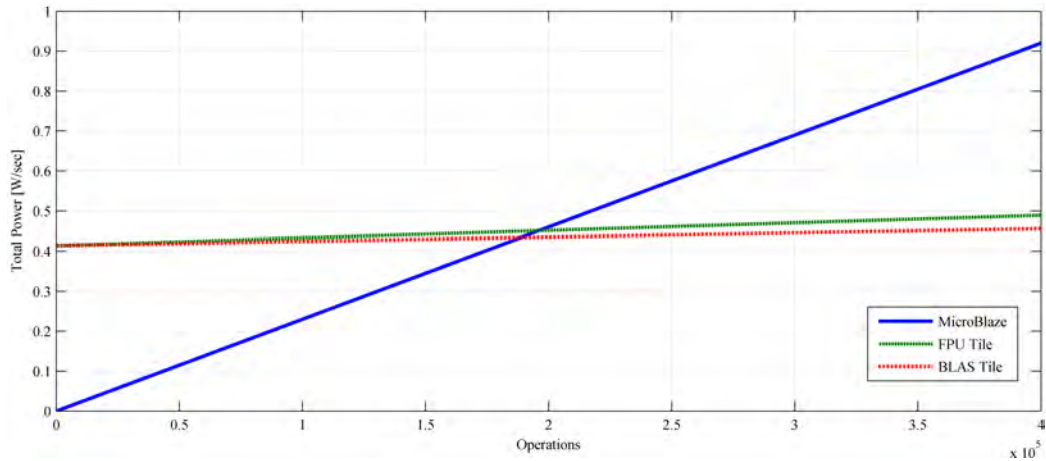


Figure 9.18: Total power consumed for an operation vs the acceleration hardware available on the TMR system

Lastly the efficiency per MFLOP was computed for the TMR system. This system like in the simplex case showed that computation efficiency is increased using the accelerators, but that efficiency is initially reduced due to the power consumed during the partial reconfiguration phase.

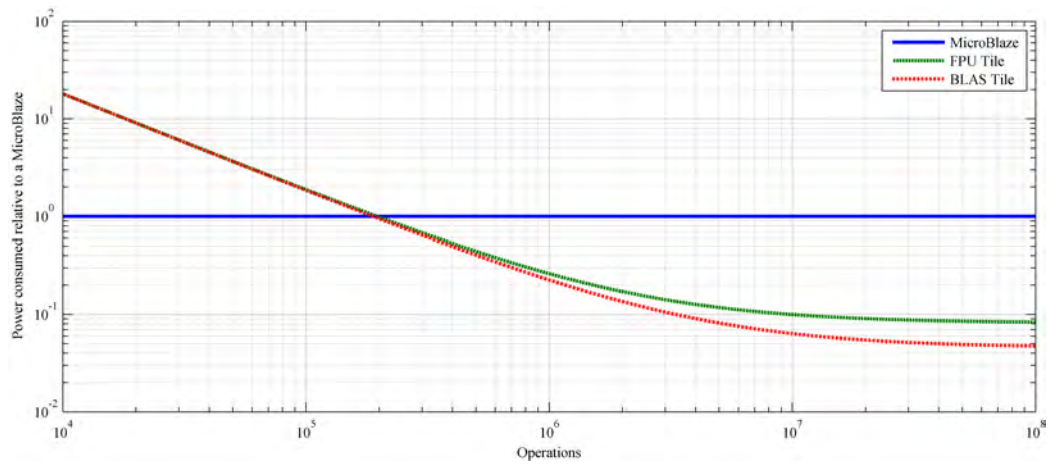


Figure 9.19: Power consumed for an operation normalized to the MicroBlaze processor for the TMR System

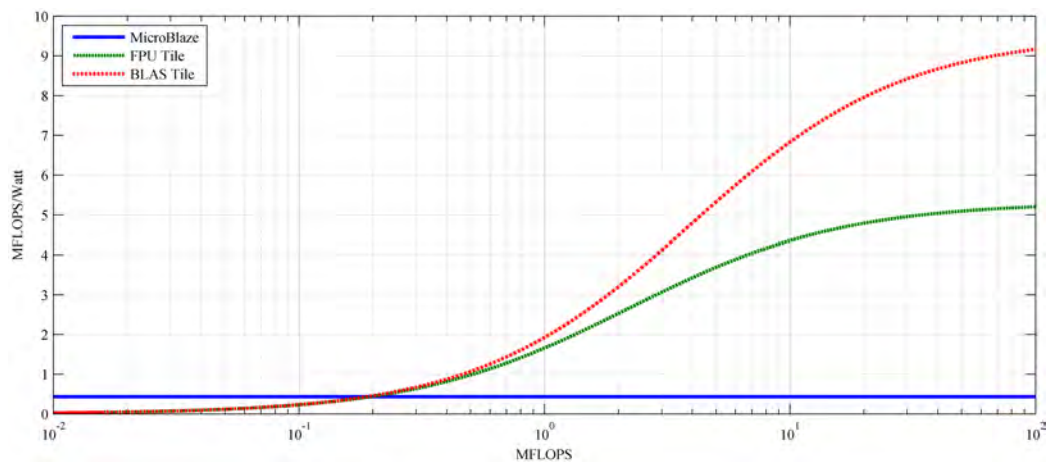


Figure 9.20: Power efficiency for the TMR system vs the size of the operation in MFLOPS

Both systems were helped by available accelerators. Should the system need to wait for the acceleration hardware to become available, the dataset required becomes large. The performance per watt characteristics, do show the validity of the concept if the program can be written to pre-configure the FPGA tiles.

## CONCLUSION AND FUTURE WORK

This research showed the design of a design and construction of a radiation hardened FPGA computer system utilizing partially reconfigurable accelerator hardware tiles. This fills a gap in currently available knowledge about using FPGAs in extreme environments. While FPGAs are a topic of interest for these situations, the practical implementation of such a system and the analysis of it is still an area of growing knowledge.

The system was further analyzed to mathematically verify failure rates in the harsh space radiation environment. In-situ and synthetic environmental testing done to date has verified the hardware can operate both mechanically and electrically in the desired environments.

In addition, this research shows an analysis of how partially reconfigurable accelerators impact system performance and power efficiency. This demonstrates and validates that FPGAs using partially reconfigurable accelerators can result in lower cost, high performance radiation computers with improved power efficiencies when compared to existing alternatives.

The hardware designed for this research greatly improves the amount of computational power available for small form factor CubeSats at the cost of increased overall power consumption over the common embedded, non-radiation hardened processors commonly used. The performance of the system also approaches that of the larger, high performance RAD750 processor, while achieves higher performance per watt (Table 10.1).

The FPGA MicroBlaze system performance metrics are under-estimated in Table 10.1, as these metrics do not include custom designed hardware accelerators which cannot be fairly applied to the Dhrystone benchmark.

Table 10.1: Performance comparison between the designed MicroBlaze based system, the RAD750 and common CubeSat processor boards

	RAD750	RAD750	MicroBlaze	SI 8151	PIC 24F	dsPIC33	MSP430
Core Speed	200MHz	132MHz	180MHz	100 MHz	16 MHz	80 MHz	25 MHz
Bus Width	32 bit	32 Bit	32 bit	8 bit	16 bit	16 bit	16 bit
RHBP	X						
RHBD		X					
RHBA			X				
FLASH/EEPROM	4MB	4MB		128KB	256KB	256KB	116KB
uP RAM/Cache	1MB	1MB	64 KB	8KB	16KB	30KB	8KB
External RAM	4-128MB	4-128MB	1GB				
DMIPS	400	260	234	20	24	92	25
Power	10-25W	10-25W	1.5-2W	195mW	66mW	297mW	9.9mW

The primary obstacle this research demonstrated for achieving power savings through PR and accelerator tiles is in hiding the reconfiguration time for the system. Only if this time can be hidden or in the case of large computations do the power and performance improve over a microprocessor system.

Future work for this research is largely dependent on additional RAM that will be present in the next revision of the hardware stack. This will allow for larger matrices to be computed and larger values of  $n$  for the LINPACK benchmarking. In addition, development was started on a startracking algorithm (Appendix B), but limited memory prevented its full development and testing.

This research is also being integrated into the development of a 3U CubeSat as a technology demonstration to show the techniques operating in the space environment (Figure 10.1).

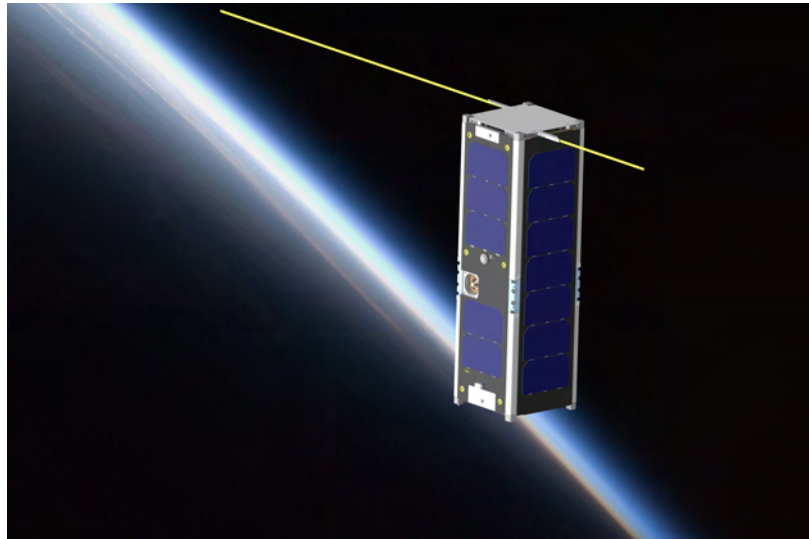


Figure 10.1: Solidworks Rendering of preliminary 3U Cubesat Design

REFERENCES CITED



- [1] J. H. Keys, Andrew and; Adams, J. D. Cressler, M. C. Patrick, M. A. Johnson, and R. C. Darty, "Radiation hardened electronics for space environments (rhese) project overview," in *Extreme Environments: Sixth International Planetary Probe Workshop*, 6 2008.
- [2] R. Cliff, V. Danchenko, E. Stassinopoulos, M. Sing, G. Brucker, and R. S. Ohanian, "Prediction and measurement of radiation damage to cmos devices on board spacecraft," *Nuclear Science, IEEE Transactions on*, vol. 23, no. 6, pp. 1781–1788, 1976.
- [3] B. J. LaMeres. (2012) Fpga-based radiation tolerant computing.
- [4] M. Berg, H. Kim, C. Perez, and A. Phan, "Xilinx virtex 5 proton accelerated radiation test," NASA Goddard, Test Report, January 2010. [Online]. Available: [http://radhome.gsfc.nasa.gov/radhome/papers/D062209\\_XCVLX30T.pdf](http://radhome.gsfc.nasa.gov/radhome/papers/D062209_XCVLX30T.pdf)
- [5] T. Flatley, "Advanced hybrid on-board science data processor - spacecube 2.0," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2012)*, 2012. [Online]. Available: [http://esto.nasa.gov/conferences/estf2011/papers/Flatley\\_ESTF2011.pdf](http://esto.nasa.gov/conferences/estf2011/papers/Flatley_ESTF2011.pdf)
- [6] M. Lin, T. Flatley, A. Geist, and D. Petrick, "Nasa gsfc development of the spacecube mini," in *25th Annual AIAA/USU Conference on Small Satellites*, 2011. [Online]. Available: [http://esto.nasa.gov/conferences/estf2011/papers/Flatley\\_ESTF2011.pdf](http://esto.nasa.gov/conferences/estf2011/papers/Flatley_ESTF2011.pdf)
- [7] A. S. Keys, J. H. Adams, M. C. Patrick, M. A. Johnson, and J. D. Cressler, "A review of nasa's radiation-hardened electronics for space environments project," in *SPSC 2010*, September 2008.
- [8] J. E. Tomayko, *Computers in space: The NASA experience*. NASA, 1988. [Online]. Available: [http://www.archive.org/details/nasa\\_techdoc\\_19880069935](http://www.archive.org/details/nasa_techdoc_19880069935)
- [9] D. Binder, E. Smith, and A. B. Holman, "Satellite anomalies from galactic cosmic rays," *Nuclear Science, IEEE Transactions on*, vol. 22, no. 6, pp. 2675–2680, 1975.
- [10] K. L. Bedingfield, R. D. Leach, and M. B. Alexander, *Spacecraft system failures and anomalies attributed to the natural space environment*. National Aeronautics and Space Administration, Marshall Space Flight Center, 1996.
- [11] L. Wang, X. Qing, Y. Liu, and K. Li, "Anomaly attributed to low earth orbit plasma environment," in *Proceedings of the 7th Spacecraft Charging Technology Conference*, 2001.

- [12] J. Matijevic and E. Dewell, *Anomaly recovery and the mars exploration rovers*. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2006.
- [13] G. Reeves and T. Neilson, "The mars rover spirit flash anomaly," in *Aerospace Conference, 2005 IEEE*, March 2005, pp. 4186–4199.
- [14] J. Marshall, N. Wood, M. Milliser, R. Ferguson, and E. Maher, "Higher performance bae systems processors and interconnects enabling spacecraft applications," in *Aerospace conference, 2009 IEEE*, 2009, pp. 1–10.
- [15] N. Haddad, R. Brown, R. Ferguson, A. Kelly, R. Lawrence, D. Pirkl, and J. Rodgers, "Second generation (200mhz) rad750 microprocessor radiation evaluation," in *Radiation and Its Effects on Components and Systems (RADECS), 2011 12th European Conference on*, 2011, pp. 877–880.
- [16] R. Berger, D. Bayles, R. Brown, S. Doyle, A. Kazemzadeh, K. Knowles, D. Moser, J. Rodgers, B. Saari, D. Stanley, and B. Grant, "The rad750tm-a radiation hardened powerpctm processor for high performance spaceborne applications," in *Aerospace Conference, 2001, IEEE Proceedings.*, vol. 5, 2001, pp. 2263–2272 vol.5.
- [17] J. H. Adams, Jr., R. Silberberg, and C. H. Tsao, "Cosmic ray effects on microelectronics. part 1: The near-earth particle environment," *NASA STI/Recon Technical Report*, vol. 81, p. 34134, Aug. 1981. [Online]. Available: <http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA103897>
- [18] J. Adams, "The natural radiation environment inside spacecraft," *Nuclear Science, IEEE Transactions on*, vol. 29, no. 6, pp. 2095–2100, 1982.
- [19] J. Barth, *Applying Computer Simulation Tools to Radiation Effects Problems*, ser. 1997 IEEE Space and Radiation Effects Conference: Short Course. IEEE Publishing Services, 1997.
- [20] A. Tylka, J. Adams, P. Boberg, B. Brownstein, W. Dietrich, E. Flueckiger, E. Petersen, M. Shea, D. Smart, and E. Smith, "Creme96: A revision of the cosmic ray effects on micro-electronics code," *Nuclear Science, IEEE Transactions on*, vol. 44, no. 6, pp. 2150–2160, 1997.
- [21] J. R. Schwank, "Space and military radiation effects in silicon-on-insulator devices." Sandia National Labs, Albuquerque, NM., Tech. Rep., Sep 1996. [Online]. Available: <http://www.osti.gov/energycitations/servlets/purl/279707-Khmy0T/webviewable/279707.pdf>

- [22] E. Petersen, J. Pickel, E. Smith, P. Rudeck, and J. Letaw, “Geometrical factors in see rate calculations,” *Nuclear Science, IEEE Transactions on*, vol. 40, no. 6, pp. 1888–1909, 1993.
- [23] D. White, “Considerations surrounding single event effects in fpgas, asics, and processors,” Xilinx, White Paper 402, March 2012. [Online]. Available: [http://www.xilinx.com/support/documentation/white\\_papers/wp402\\_SEE\\_Considerations.pdf](http://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf)
- [24] S. Kerns, B. Shafer, J. Rockett, L.R., J. Pridmore, D. Berndt, N. van Vonno, and F. Barber, “The design of radiation-hardened ics for space: a compendium of approaches,” *Proceedings of the IEEE*, vol. 76, no. 11, pp. 1470–1509, 1988.
- [25] E. F. Moore and C. E. Shannon, “Reliable circuits using less reliable relays. i-ii,” *Journal of the Franklin Institute*, vol. 262, no. 43, pp. 191–208, 1956. [Online]. Available: [http://mriedel.ece.umn.edu/wiki/images/3/30/Moore\\_Shannon\\_Reliable\\_Circuits\\_Using\\_Less\\_Reliable\\_Relays.pdf](http://mriedel.ece.umn.edu/wiki/images/3/30/Moore_Shannon_Reliable_Circuits_Using_Less_Reliable_Relays.pdf)
- [26] J. von Neumann, “Probabilistic logics and the synthesis of reliable organisms from unreliable components,” *Automata Studies*, pp. 43–98, 1956. [Online]. Available: [http://www.mriedel.ece.umn.edu/wiki/images/a/af/Von\\_Neumann\\_Probabilistic\\_Logics\\_and\\_the\\_Synthesis\\_of\\_Reliable\\_Organisms\\_from\\_Unreliable\\_Components.pdf](http://www.mriedel.ece.umn.edu/wiki/images/a/af/Von_Neumann_Probabilistic_Logics_and_the_Synthesis_of_Reliable_Organisms_from_Unreliable_Components.pdf)
- [27] N. H. Rollins, “Hardware and software fault-tolerance of softcore processors implimented in sram-based fpgas,” Ph.D. dissertation, Brigham Young University, April 2012. [Online]. Available: <http://nrollins.com/files/dissertation.pdf>
- [28] J. Wakerly, “Transient failures in triple modular redundancy systems with sequential modules,” *IEEE Transactions on Computers*, vol. 24, no. 5, pp. 570–573, 1975. [Online]. Available: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1672856>
- [29] P. Ostler, M. Caffrey, D. Gibelyou, P. Graham, K. Morgan, B. Pratt, H. Quinn, and M. Wirthlin, “Sram fpga reliability analysis for harsh radiation environments,” *Nuclear Science, IEEE Transactions on*, vol. 56, no. 6, pp. 3519–3526, 2009.
- [30] E. Fong, M. Converse, and P. Denham, “An electrically reconfigurable programmable logic array using a cmos/dmos technology,” *Solid-State Circuits, IEEE Journal of*, vol. 19, no. 6, pp. 1041–1043, 1984.
- [31] El-Araby, Esam, I. Gonzalez, and T. El-Ghazawi, “Exploiting partial runtime reconfiguration for high-performance reconfigurable computing,” *ACM Trans.*

- Reconfigurable Technol. Syst.*, vol. 1, no. 4, pp. 21:1–21:23, Jan. 2009. [Online]. Available: <http://doi.acm.org/10.1145/1462586.1462590>
- [32] P. Graham, M. Caffery, M. Wirthlin, D. E. Johnson, and N. H. Rollins, “Reconfigurable computing in space: From current technology to reconfigurable systems-on-a-chip,” in <http://nhrollins.com/files/publications/second/ieeaero03.pdf>, 2003. [Online]. Available: <http://nhrollins.com/files/publications/second/ieeaero03.pdf>
- [33] M. Berg, “Field programmable gate array (fpga) single event effect (see) radiation testing,” MEI Technologies in support of NASA/Goddard Space Flight Center, Tech. Rep., February 2012. [Online]. Available: [https://nepp.nasa.gov/files/23779/FPGA\\_Radiation\\_Test\\_Guidelines\\_2012.pdf](https://nepp.nasa.gov/files/23779/FPGA_Radiation_Test_Guidelines_2012.pdf)
- [34] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, and J. Fabula, “Radiation testing update, seu mitigation, and availability analysis of the virtex fpga for space re-configurable computing,” in *presented at the IEEE Nuclear and Space Radiation Effects Conference in Proc. International Conference on Military and Aerospace Programmable Logic Devices*, 2000. [Online]. Available: <http://nhrollins.com/files/publications/second/ieeaero03.pdf>
- [35] M. Berg, C. Poivey, D. Petrick, K. LaBel, M. Friendlich, S. Stansberry, and H. Kim, “Risk reduction for use of complex devices in space projects,” *Nuclear Science, IEEE Transactions on*, vol. 54, no. 6, pp. 2137–2140, 2007.
- [36] J. M. Johnson and M. J. Wirthlin, “Voter insertion algorithms for fpga designs using triple modular redundancy,” in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA ’10. New York, NY, USA: ACM, 2010, pp. 249–258. [Online]. Available: <http://doi.acm.org/10.1145/1723112.1723154>
- [37] J. S. Hane, “A fault-tolerant computer architecture for space vehicle applications,” Master’s thesis, Montana State University, May 2012.
- [38] E. C. Gowens, “Two dimensional radiation sensor development for use in space bound reconfigurable computers,” Master’s thesis, Montana State University, May 2011.
- [39] T. M. Buerkle, “Ionizing radiation detector for enviromental awareness in fpga-based flight computers,” Master’s thesis, Montana State University, May 2012.
- [40] J. A. Hogan, “Reliability analysis of radiation induced fault mitigation strategies in field programmable gate arrays,” Ph.D. dissertation, Montana State University, May 2014.

- [41] “Tcm8240md datasheet v1.3,” Toshiba, April 2004. [Online]. Available: [https://www.sparkfun.com/datasheets/Sensors/Imaging/TCM8240MD\\_E150405\\_REV13.pdf](https://www.sparkfun.com/datasheets/Sensors/Imaging/TCM8240MD_E150405_REV13.pdf)
- [42] “P5q serial phase change memory (pcm) datasheet,” Micron, January 2012. [Online]. Available: [http://www.micron.com/~/media/Documents/Products/Data%20Sheet/PCM/p5q\\_32\\_64\\_128Mb\\_serial\\_pcm\\_ds.pdf](http://www.micron.com/~/media/Documents/Products/Data%20Sheet/PCM/p5q_32_64_128Mb_serial_pcm_ds.pdf)
- [43] “Voltage margining using the tps62130,” Application Note, Texas Instruments, July 2010. [Online]. Available: <http://www.ti.com/lit/an/slva375b/slva375b.pdf>
- [44] “Voltage margining using the tps62130,” Application Note, Texas Instruments, November 2011. [Online]. Available: <http://www.ti.com/lit/an/slva489/slva489.pdf>
- [45] “Ucd90124 datasheet,” Texas Instruments, September 2010. [Online]. Available: <http://www.ti.com/lit/ds/slvs29b/slvs29b.pdf>
- [46] Z. Jin, R. N. Pittman, and A. Forin, “Reconfigurable custom floating-point instructions,” Microsoft Resesarch, Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA 98052, Tech. Rep. MSR-TR-2009-157, August 2009. [Online]. Available: [http://research.microsoft.com/pubs/115374/Reconfigurable%20Custom%20Floating-Point%20Instructions\\_v2.pdf](http://research.microsoft.com/pubs/115374/Reconfigurable%20Custom%20Floating-Point%20Instructions_v2.pdf)
- [47] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, “Basic linear algebra subprograms for fortran usage,” *ACM Trans. Math. Softw.*, vol. 5, no. 3, pp. 308–323, Sep. 1979, bLAS paper - leads into LINPACK. [Online]. Available: <http://doi.acm.org/10.1145/355841.355847>
- [48] L. Zhuo and V. Prasanna, “Design tradeoffs for blas operations on reconfigurable hardware,” in *Parallel Processing, 2005. ICPP 2005. International Conference on*, 2005, pp. 78–86.
- [49] W. Zhang, V. Betz, and J. Rose, “Portable and scalable fpga-based acceleration of a direct linear system solver,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 1, pp. 6:1–6:26, Mar. 2012. [Online]. Available: <http://doi.acm.org/10.1145/2133352.2133358>
- [50] K. Underwood and K. Hemmert, “Closing the gap: Cpu and fpga trends in sustainable floating-point blas performance,” in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 2004, pp. 219–228.
- [51] S. Kestur, J. D. Davis, and O. Williams, “Blas comparison on fpga, cpu and gpu,” in *Proceedings of the 2010 IEEE Annual Symposium on VLSI*, ser. ISVLSI

- '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 288–293. [Online]. Available: <http://dx.doi.org/10.1109/ISVLSI.2010.84>
- [52] L. Guedria, “High performance fpga based blas accelerator,” November 2005, centre of Excellence in Information and Communication Technologies. [Online]. Available: [http://rssi.ncsa.illinois.edu/2007/docs/academic/Guedria\\_presentation.pdf](http://rssi.ncsa.illinois.edu/2007/docs/academic/Guedria_presentation.pdf)
- [53] J. J. Dongarra, *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard*, BLAST Forum Std., January 2001. [Online]. Available: <http://www.compsci.wm.edu/sciclone/documentation/software/math/ATLAS/blas-report.pdf>
- [54] H. M. Deitel, P. J. Deitel, and D. R. Choffnes, *Operating Systems*, 3rd ed. Practice Hall, 2004.
- [55] I. D. Craig, *Formal Models of Operating System Kernels*. Springer, 2007.
- [56] S. Karlin and H. M. Taylor, *An Introduction to Stochastic Modeling, Third Edition*. Academic Press, 1998.
- [57] K. Trivedi, *Probability and Statistics With Reliability, Queuing and Computer Science Applications*. Prentice Hall PTR, 1982.
- [58] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*, 4th ed. Morgan Kaufmann, 2006.

APPENDICES

APPENDIX A

TCM8240 CAMERA INITIALIZATION



Table A.1: Toshiba TCM8240 JPEG Mode Initialization Sequence

Address	Value	Operation
0x02	0x00	Camera on
0x02	0x40	Reset camera
0x02	0x00	Camera on
0x03	0xC7	Enable PLL Mode 7
0xE9	0x10	Q Table Gain (Image Quality)
0xEA	0x10	Q Table Gain (Image Quality)
0x24	0x5E	Unknown, but necessary
0x01	0x00	Unknown, but necessary
0xF1	0x02	DRI
0x04	0xC0	Set the image size
0x0B	0x00	Disable the whitebar
0x52	0x50	Set the exposure mode
0x58	0x30	Set the exposure time
0x32	0x8C	Automatic picture correction
0x05	0x80	Set the Frame Speed
0x1A	0xB3	HCNT (low byte)
0x1B	0x3B	HCNT (high nibble) VCNT (low)
0x1C	0x21	VCNT (high byte)
0x11	0x48	Unknown, but necessary
0x14	0x23	Reduces image noise
0x1E	0x87	SPCOUNT (low byte)
0x1F	0x0B	SPCOUNT (high byte)
0xE6	0x00	Unknown, but necessary
0x0E	0x00	Center image horizontally
0x48	0x85	Auto Level Control Enable
0x4A	0x48	Auto Level Control Mode
0x6D	0x48	Auto White Balance Enable and Mode
0x38	0x00	AGMIN PPED
0x39	0xFF	AGMAX PPED
0x5A	0x00	AGMIN
0x5B	0xFF	AGMAX
0x23	0x00	Changes BGR to RGB mode
0x81	0x48	Red-Green Balance
0x82	0x22	Blue-Green Balance
0xB8	0x9F	Gamma Correction
0xEF	0x3F	JPG Encoder Options
0xEE	0x3F	JPG Encoder Options

Table A.2: Toshiba TCM8240 Bitmap Mode Initialization Sequence

Address	Value	Operation
0x02	0x00	Camera on
0x02	0x40	Reset camera
0x02	0x00	Camera on
0x03	0xC1	Enable PLL Mode 1
0x04	0x8A	Set the image size
0x0B	0x00	Disable the whitebar
0x52	0x50	Set the exposure mode
0x58	0x30	Set the exposure time
0x32	0x8C	Automatic picture correction
0x05	0x80	Set the Frame Speed
0x1A	0xFF	HCNT (low byte)
0x1B	0xB3	HCNT (high nibble) VCNT (low)
0x1C	0xA1	VCNT (high byte)
0x11	0x4A	QVGA Setting
0x14	0x23	Reduces image noise
0x1E	0x23	SPCOUNT (low byte)
0x1F	0x0C	SPCOUNT (high byte)
0x0E	0x00	Center image horizontally
0x48	0x85	Auto Level Control Enable
0x4A	0x48	Auto Level Control Mode
0x6D	0x48	Auto White Balance Enable and Mode
0x38	0x00	AGMIN PPED
0x39	0xFF	AGMAX PPED
0x5A	0x00	AGMIN
0x5B	0xFF	AGMAX
0x23	0x00	Changes BGR to RGB mode
0x81	0x48	Red-Green Balance
0x82	0x22	Blue-Green Balance
0xB8	0x9F	Gamma Correction

APPENDIX B

STAR TRACKING GYROSCOPE

## Background

To test the system with a more realistic application, a star tracking gyroscope algorithm was developed and tested for the system. This consisted of the system collecting an image, identifying stars and the relationships between them, and using subsequent images to determine the transition of the system compared to its initial location. This application utilizes the camera, floating point operations, and matrix operations in rapid succession allowing for testing of all three system accelerators with a single commonly required software application.

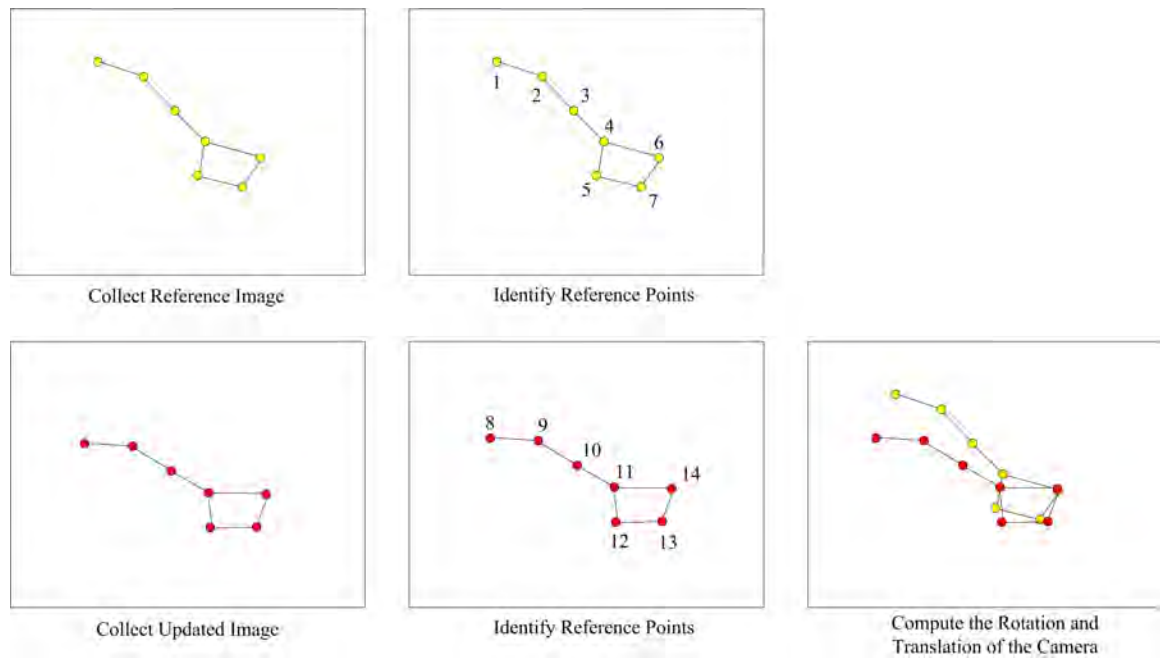


Figure B.1: Startracker procedure vs accelerator type

The initial step in star tracking is to collect an image from the camera on the hardware stack using the camera tile.

Once the initial image has been collected the image is filtered with a threshold detector to filter noise, dust and small stars.

After the image has been threshold filtered, the starts are located by sweeping the image with a filter to detect the center of the stars, which appear as white circles in the image. These xy coordinates are then processed in the remainder of the procedure.

After a second image is collected and processed, the several stars are picked at random from the image. These are corralled with the previous image by finding the xy with the minimum vector distance between the images and the gyroscopic motion is computed.

### Mathematical Model

The image is collected by the camera as a grayscale bitmap stored in an array *IMG*.

The threshold image filter if run on the image is defined as:

$$IMG = IMG > threshold \quad (B.1)$$

The star centers are then located and the xy coordinates are stored in the matrix *star* where each row of *star* contains [ *starX<sub>n</sub>* *starY<sub>n</sub>* ].

To correlate the new star's position with its previous location the minimum vector distance between the desired star's previous image location and the current location of all stars is computed.

$$starCorr_n^+ = \arg \min_n (\|star - star_n\|_2) \quad (B.2)$$

Once the stars have been located and correlated between images, the slope of the tangent line associated with a line between the correlated stars are computed by:

$$m_{star} = \frac{\Delta X}{\Delta Y} \quad (\text{B.3})$$

An overdefined system of equations is then created to locate the intersection of the tangent lines which is equivalent to the center of rotation for the image:

$$\begin{bmatrix} Y_1 + m_1 * X_1 \\ Y_1 + m_1 * X_1 \\ \vdots \\ Y_n + m_n * X_n \end{bmatrix} = \begin{bmatrix} m_1 & 1 \\ m_1 & 1 \\ \vdots & \vdots \\ m_n & 1 \end{bmatrix} \begin{bmatrix} X_{center} \\ Y_{center} \end{bmatrix} \quad (\text{B.4})$$

The result of solving this system of equations is the location of the center of rotation movement observed between the frames.

The rotation of the image is then normalized through solving the system of equation to find the intersection, and a finite difference approach, translating the image in x and y (pitch and yaw) to center to observed rotation in the center of the image.

The result of this process allows the yaw to be computed with:

$$\phi = \arctan \left( \frac{\Delta X}{viewfield_{X.rad}} \right) \quad (\text{B.5})$$

The pitch is computed with:

$$\theta = \arctan \left( \frac{\Delta Y}{viewfield_{Y.rad}} \right) \quad (\text{B.6})$$

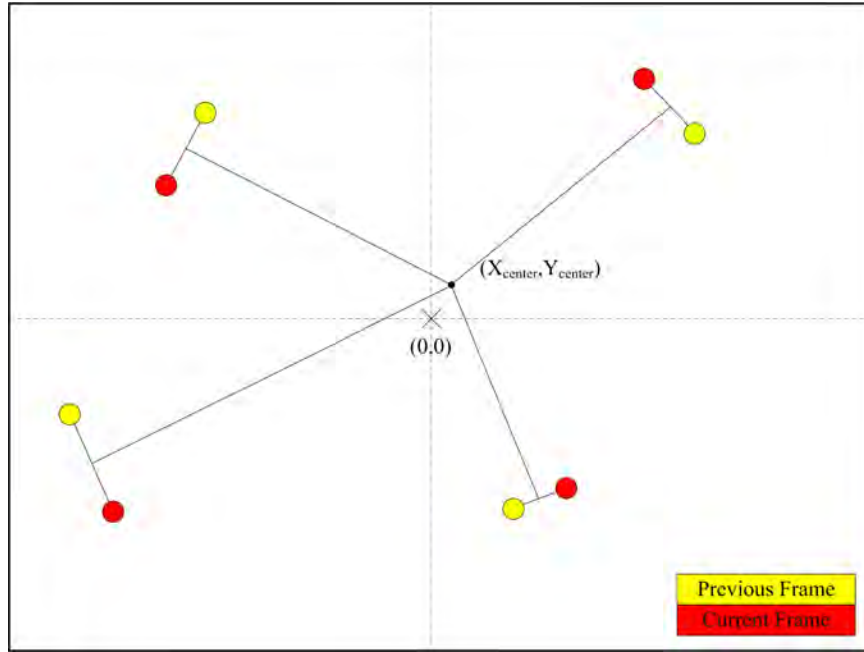


Figure B.2: Star tracker showing a mis-centered center of rotation

And the roll can be computed with:

$$\psi = 2 * \arctan \left( \frac{X}{Y} \right) \quad (\text{B.7})$$

Once differences are computed, the data is feed into a Kalman filter to reduce the noise and increase the accuracy of the tracking data. This also serves to eliminate noise issues where stars are detected improperly in one set of frames and allow the long term observations to correct for this error. In addition, once initialized the Kalman filters output can be used to guess roll, pitch, and yaw, reducing the chance mis-correlating the observed stars.

The mathematical model for the traditional Kalman Filter is used where the state estimate is defined as:

$$x_{k|k-1} = F_k x_{k-k|k-1} + B_k U_{k-1} \quad (\text{B.8})$$

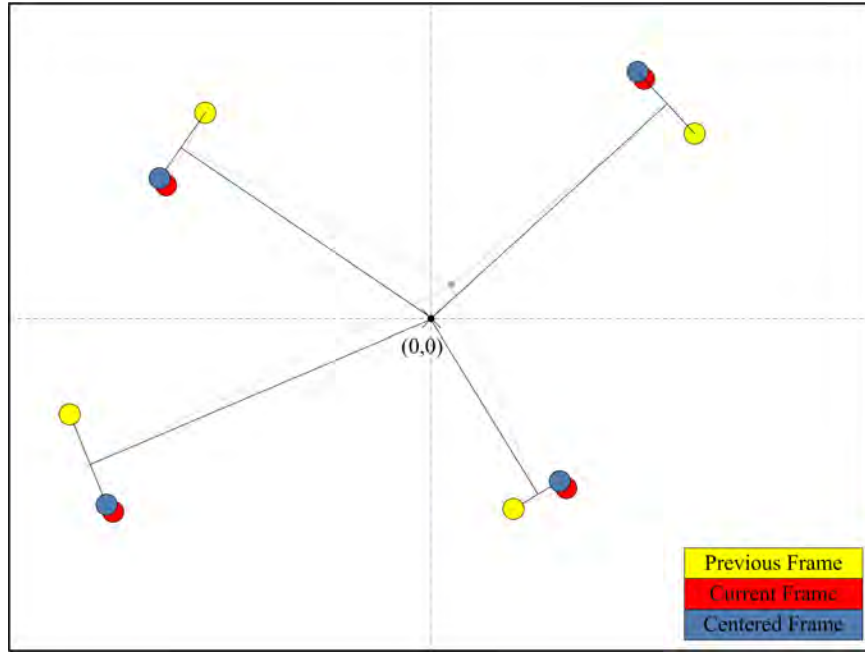


Figure B.3: Star tracking algorithm with displacements to normalize the rotation around the z axis

and the predicted covariance is:

$$P_k|k-1 = F_k P_{k-1|k-1} F_k^T + Q_k \quad (\text{B.9})$$

The update procedure is then to update the innovation parameter:

$$y_k = z_k - H_k x_{k|k-1} \quad (\text{B.10})$$

and the innovation covariance:

$$S_k = H_k P_{k|k-1} H_k^T + R_k \quad (\text{B.11})$$



The optimal Kalman gain is then computed by:

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \quad (\text{B.12})$$

The state estimate is then updated with:

$$x_{k|k-1} + K_k y_k \quad (\text{B.13})$$

and the covariance estimate is updated by:

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \quad (\text{B.14})$$

### MATLAB Implementation

The algorithm was initially developed and tested in a MATLAB simulation. A simulated starfield was generated with randomly placed stars. Each star was represented as a 2-dimensional Gaussian distribution. The script then rotated the simulated starfield by a known amount and the rotation and translation was computed.

Yellow circles are the previous frame location of the star, the red is the current frames location of the detected star, and the blue circle denotes the location of the star after the center of rotation is fixed to the center of the image frame.

With this filtering, the Kalman output reduced the noise and converged towards the simulated rotation rate. This output could then be integrated into the next time step to reduce the search space.

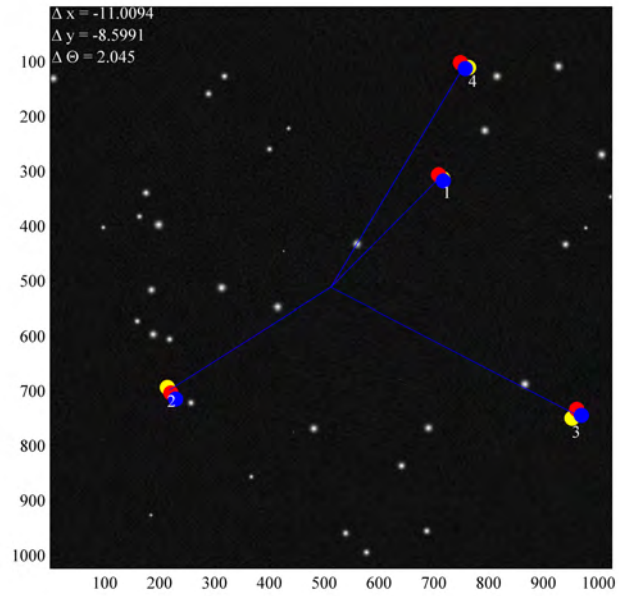


Figure B.4: MATLAB Simulated Star Tracking Algorithm

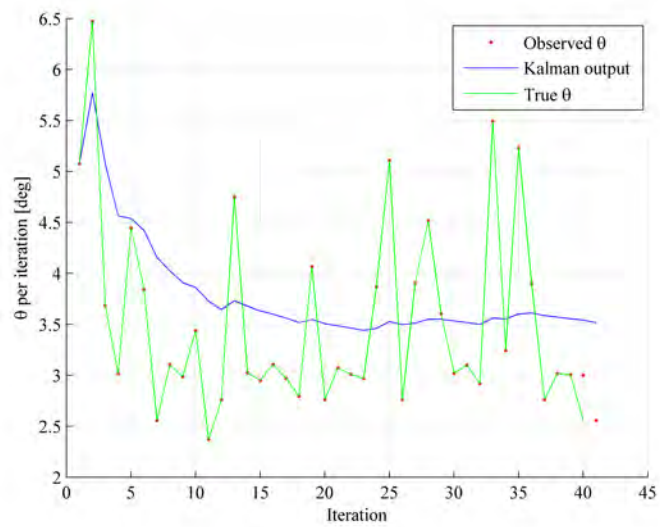


Figure B.5: MATLAB Simulated Star Tracker roll angle after Kalman filtering

Matlab Code

```
%% Global Variables

xlim = 1024;
ylim = 1024;
maxStars = 40;

m = 7;

%% Generate a starfield
img = zeros(ylim , xlim);

%random noise in the image
img = 1.2*rand(ylim , xlim);

%generate the stars
for ind = 1:maxStars
    %random star location
    x = round(xlim*rand);
    y = round(ylim*rand);

    %random star size
    intensity = rand;

    %random star size
    sigma = 20*rand;
```

```

%generate the star
for xind = (x-12):(x+12)
    for yind = (y-12):(y+12)
        try
            img(yind , xind) = img(yind , xind) + m*exp(-((
                xind - x)^2 / (2*sigma)+((yind-y)^2/(2*
                sigma)))));
        end
    end
end
end

%show the resulting starfield
figure(1)
imagesc(img)
colormap(gray);
axis square

%% find the stars....

%Threshold the data so I'm only looking at bright objects
thImg = img>5;
imagesc(thImg)

```

```

axis square
xCenter = zeros(ylim , xlim);
yCenter = zeros(ylim , xlim);

xStarWidth=0;
yStarWidth=0;

%Then step through the image and find the centers
for yind = 1:ylim
    for xind = 1:xlim
        %x centroid
        if (thImg(yind , xind) == 1)
            xStarWidth = xStarWidth+1;
        else
            if (xStarWidth>0)
                try
                    xCenter(yind , xind - round((xStarWidth)/2)
                        )=1;
                end
            end
            xStarWidth = 0;
        end
    end

    %y centroid
    if (thImg(xind , yind) == 1)

```

```

        yStarWidth = yStarWidth+1;
    else
        if (yStarWidth>0)
            try
                yCenter(xind - round((yStarWidth)/2),yind
                    )=1;
            end
            yStarWidth = 0;
        end
    end
end
end
end
end

```

*%then extract the positions*

```
[starPosX , starPosY] = find(xCenter.*yCenter == 1);
```

*%% Rotate and translate to make a 2nd image*

```
imgR = imrotate(img,2);
```

```
img2 = imgR(9:1024+8,11:1024+10);
```

*%% Find the stars again*

*%Threshold the data so I'm only looking at bright objects*

```
thImg = img2>5;
```

```

imagesc(thImg)

axis square

xCenter = zeros(ylim , xlim);
yCenter = zeros(ylim , xlim);

xStarWidth=0;
yStarWidth=0;

%Find the star centers again
for yind = 1:ylim
    for xind = 1:xlim
        %x centroid
        if (thImg(yind , xind) == 1)
            xStarWidth = xStarWidth+1;
        else
            if (xStarWidth>0)
                try
                    xCenter(yind , xind - round((xStarWidth)/2)
                        )=1;
                end
            end
            xStarWidth = 0;
        end
    end

    %y centroid

```

```

if (thImg(xind , yind) == 1)
    yStarWidth = yStarWidth+1;
else
    if (yStarWidth>0)
        try
            yCenter(xind - round((yStarWidth)/2) , yind
                )=1;
        end
        yStarWidth = 0;
    end
end
end
end

```

*%then extract the positions*

```
[starPosXn , starPosYn] = find(xCenter.*yCenter == 1);
```

*%% Calculate the displacement/rotation between frames*

```
maxNodes = 4;
```

*%randomly pick the stars to use*

```
starIndex = randsample(1:length(starPosX) , maxNodes);
```

```
deltax = 0;
```



```

deltay = 0;

for aaa = 1:50
    %find the new location of a star
    for star = 1:maxNodes
        [~,ix] = min( sqrt((starPosX(starIndex(star)) -
            starPosXn).^2+(starPosY(starIndex(star)) -
            starPosYn).^2));
        dx(star) = starPosYn(ix) - starPosY(starIndex(star))+
            deltax;
        dy(star) = starPosXn(ix) - starPosX(starIndex(star))+
            deltay;

        starX(star) = starPosX(starIndex(star));
        starXn(star) = starPosXn(ix)+deltax;
        starY(star) = starPosY(starIndex(star));
        starYn(star) = starPosYn(ix)+deltay;

        if (dx(star)>10)
            star = star -1;
        end
    end

%calculate the center of rotation
% find the slopes

```

```

for star = 1:maxNodes
    deltaSlope(star) = (starX(star)-starXn(star)) / (
        starY(star)-starYn(star));
end

%draw the lines ....
for star = 1:maxNodes
    line([starY(star), starYn(star)], [starX(star),
        starXn(star)], 'color', 'c')
end

% get the equation of the tangent lines and center points
tanSlope = deltaSlope.^-1;

tanX = mean([starY;starYn]);
tanY = mean([starX;starXn]);

%generate a system of equation and find the intersection
    of the tanget
%lines (the center of rotation)
sysEqu = [];
for star = 1:maxNodes
    sysEqu = [sysEqu; tanSlope(star), 1, tanY(star) +
        tanSlope(star)*tanX(star)];
end

```

```

intersectPt = inv((sysEqu(:,1:2)')*sysEqu(:,1:2))*sysEqu
    (:,1:2) '*sysEqu(:,3);

if sum(isnan(intersectPt))
    starIndex = randsample(1:length(starPosX),maxNodes);
end

for star = 1:maxNodes
    text(starY(star),starX(star)+25,num2str(star),'color'
        , 'white');
    line([tanX(star), intersectPt(1)], [tanY(star),
        intersectPt(2)])
end

hold off

A = sqrt((starX-tanY).^2 + (starY-tanX).^2);
B = sqrt((tanY-512).^2 + (tanX-512).^2);
theta = mean( 2*atan(A./B) * 180/pi );

deltax = deltax - (intersectPt(1)-512)/100;
deltay = deltay + (intersectPt(2)-512)/100;

if ( sum( (intersectPt - 512).^2) <.01

```

```
        break;
    end
end

disp(['Delta x = ', num2str(deltax)])
disp(['Delta y = ', num2str(deltay)])
disp(['Delta Theta = ', num2str(theta)])
```

APPENDIX C

MATLAB CODE FOR MARKOV RELIABILITY MODELS

*%% Simulation Parameters*

```

StrikeRate = 1.1075;           % Values in 10^n strikes/
    second
TimeLimits  = [-5,2];         % Values in 10^n hours
Tiles = 9;                    % Number of system tiles
PercentUalized = 1;          % Percent of hardware
    sensitive
ContextSwitch = 1.1;         % Context Switch Time in ms
TilePRTime = 100;           % PR Scrub time in ms
derating = .85;              % Percent that cause a SEE

```

*%% Simulation Variables*

```

strikesPerDayList = StrikeRate*PercentUalized*derating*7e-4;
timeList = logspace(TimeLimits(1),TimeLimits(2),1000)*60*60;
failureProb = [];
A = 1/Tiles * PercentUalized;
mu = 1/TilePRTime;
nu = 1/ContextSwitch;

```

*%% Run the simulation for all Strike Rates*

```

for sInd = 1:length(strikesPerDayList)
    lambda = strikesPerDayList(sInd);

```

*%% State transition probability matrix - TMR + Spares*

```

    p = zeros(29,29);

```

*%n1 (Good)*

$p(3,1) = 3*A*\lambda;$   
 $p(2,1) = 6*A*\lambda;$   
 $p(1,1) = 1-\mathbf{sum}(p(:,1));$

*%n2 (0 tile faults, 1 spare faults)*

$p(1,2) = \mu;$   
 $p(4,2) = 5*A*\lambda;$   
 $p(6,2) = 3*A*\lambda;$   
 $p(2,2) = 1-\mathbf{sum}(p(:,2));$

*%n3 (1 tile faults, 0 spare faults)*

$p(2,3) = \nu;$   
 $p(29,3) = 2*A*\lambda;$   
 $p(5,3) = 6*A*\lambda;$   
 $p(3,3) = 1-\mathbf{sum}(p(:,3));$

*%n4 (0 tile faults, 2 spare faults)*

$p(2,4) = \mu;$   
 $p(7,4) = 4*A*\lambda;$   
 $p(10,4) = 3*A*\lambda;$   
 $p(4,4) = 1-\mathbf{sum}(p(:,4));$

*%n5 (1 tile faults, 1 spare faults)*

```

p(4 ,5) = nu;
p(8 ,5) = 5*A*lambda;
p(29,5) = 2*A*lambda;
p(5 ,5) = 1-sum(p(:,5));

```

```
%n6 (1 tile fault , 1 spare fault)
```

```

p(4 ,6) = nu;
p(9 ,6) = 5*A*lambda;
p(29,6) = 2*A*lambda;
p(6 ,6) = 1-sum(p(:,6));

```

```
%n7 (0 tile fault , 3 spare fault)
```

```

p(4 ,7) = mu;
p(11,7) = 3*A*lambda;
p(15,7) = 3*A*lambda;
p(7 ,7) = 1-sum(p(:,7));

```

```
%n8 (1 tile fault , 2 spare fault)
```

```

p(7 ,8) = nu;
p(12,8) = 4*A*lambda;
p(29,8) = 2*A*lambda;
p(8 ,8) = 1-sum(p(:,8));

```

```
%n9 (1 tile fault , 2 spare fault)
```

```

p(7 ,9) = nu;

```



```

p(13,9) = 4*A*lambda;
p(29,9) = 2*A*lambda;
p(9,9) = 1-sum(p(:,9));

```

```
%n10 (1 tile fault, 2 spare fault)
```

```

p(7,10) = nu;
p(14,10) = 4*A*lambda;
p(29,10) = 2*A*lambda;
p(10,10) = 1-sum(p(:,10));

```

```
%n11 (0 tile fault, 4 spare fault)
```

```

p(7,11) = mu;
p(16,11) = 2*A*lambda;
p(21,11) = 3*A*lambda;
p(11,11) = 1-sum(p(:,11));

```

```
%n12 (1 tile fault, 3 spare fault)
```

```

p(11,12) = nu;
p(17,12) = 3*A*lambda;
p(29,12) = 2*A*lambda;
p(12,12) = 1-sum(p(:,12));

```

```
%n13 (1 tile fault, 3 spare fault)
```

```

p(11,13) = nu;
p(18,13) = 3*A*lambda;

```

```
p(29,13) = 2*A*lambda;
p(13,13) = 1-sum(p(:,13));
```

```
%n14 (1 tile fault, 3 spare fault)
```

```
p(11,14) = nu;
p(19,14) = 3*A*lambda;
p(29,14) = 2*A*lambda;
p(14,14) = 1-sum(p(:,14));
```

```
%n15 (1 tile fault, 3 spare fault)
```

```
p(11,15) = nu;
p(20,15) = 3*A*lambda;
p(29,15) = 2*A*lambda;
p(15,15) = 1-sum(p(:,15));
```

```
%n16 (0 tile fault, 5 spare fault)
```

```
p(11,16) = mu;
p(22,16) = 1*A*lambda;
p(28,16) = 3*A*lambda;
p(16,16) = 1-sum(p(:,16));
```

```
%n17 (1 tile fault, 4 spare fault)
```

```
p(16,17) = nu;
p(23,17) = 2*A*lambda;
p(29,17) = 2*A*lambda;
```

$$p(17,17) = 1 - \mathbf{sum}(p(:,17));$$

*%n18 (1 tile fault, 4 spare fault)*

$$p(16,18) = \text{nu};$$

$$p(24,18) = 2 * A * \text{lambda};$$

$$p(29,18) = 2 * A * \text{lambda};$$

$$p(18,18) = 1 - \mathbf{sum}(p(:,18));$$

*%n19 (1 tile fault, 4 spare fault)*

$$p(16,19) = \text{nu};$$

$$p(25,19) = 2 * A * \text{lambda};$$

$$p(29,19) = 2 * A * \text{lambda};$$

$$p(19,19) = 1 - \mathbf{sum}(p(:,19));$$

*%n20 (1 tile fault, 4 spare fault)*

$$p(16,20) = \text{nu};$$

$$p(26,20) = 2 * A * \text{lambda};$$

$$p(29,20) = 2 * A * \text{lambda};$$

$$p(20,20) = 1 - \mathbf{sum}(p(:,20));$$

*%n21 (1 tile fault, 4 spare fault)*

$$p(16,21) = \text{nu};$$

$$p(27,21) = 2 * A * \text{lambda};$$

$$p(29,21) = 2 * A * \text{lambda};$$

$$p(21,21) = 1 - \mathbf{sum}(p(:,21));$$

*%n22 (0 tile fault, 6 spare fault)*

$p(16,22) = \mu;$   
 $p(29,22) = 3*A*\lambda;$   
 $p(22,22) = 1-\mathbf{sum}(p(:,22));$

*%n23 (1 tile fault, 5 spare fault)*

$p(22,23) = \nu;$   
 $p(29,23) = 3*A*\lambda;$   
 $p(23,23) = 1-\mathbf{sum}(p(:,23));$

*%n24 (1 tile fault, 5 spare fault)*

$p(22,24) = \nu;$   
 $p(29,24) = 3*A*\lambda;$   
 $p(24,24) = 1-\mathbf{sum}(p(:,24));$

*%n25 (1 tile fault, 5 spare fault)*

$p(22,25) = \nu;$   
 $p(29,25) = 3*A*\lambda;$   
 $p(25,25) = 1-\mathbf{sum}(p(:,25));$

*%n26 (1 tile fault, 5 spare fault)*

$p(22,26) = \nu;$   
 $p(29,26) = 3*A*\lambda;$   
 $p(26,26) = 1-\mathbf{sum}(p(:,26));$

```
%n27 (1 tile fault , 5 spare fault)
```

```
p(22,27) = nu;
```

```
p(29,27) = 3*A*lambda;
```

```
p(27,27) = 1-sum(p(:,27));
```

```
%n28 (1 tile fault , 5 spare fault)
```

```
p(22,28) = nu;
```

```
p(29,28) = 3*A*lambda;
```

```
p(28,28) = 1-sum(p(:,28));
```

```
%n29 (Failed)
```

```
p(1,29)= 0;
```

```
p(29,29)= 1;
```

```
%% State transition probability matrix – TMR no scrubber
```

```
pTMRnoS = zeros(3,3);
```

```
%Good State
```

```
pTMRnoS(2,1) = 3*A*lambda;
```

```
pTMRnoS(1,1) = 1-pTMRnoS(2,1);
```

```
%1 Tile Fault
```

```
pTMRnoS(3,2) = 2*A*lambda;
```

```
pTMRnoS(2,2) = 1 - pTMRnoS(3,2);
```

*%Failed*

$p_{\text{TMRnoS}}(3,3) = 1;$

$p_{\text{TMRnoS}}(1,3) = 0;$

*%% State transition probability matrix – TMR+Scrubber*

$p_{\text{TMR}} = \mathbf{zeros}(3,3);$

*%Good State*

$p_{\text{TMR}}(2,1) = 3*A*\lambda;$

$p_{\text{TMR}}(1,1) = 1 - p_{\text{TMR}}(2,1);$

*%1 Tile Fault*

$p_{\text{TMR}}(3,2) = 2*A*\lambda;$

$p_{\text{TMR}}(1,2) = \mu;$

$p_{\text{TMR}}(2,2) = 1 - p_{\text{TMR}}(3,2) - p_{\text{TMR}}(1,2);$

*%Failed*

$p_{\text{TMR}}(3,3) = 1;$

$p_{\text{TMR}}(1,3) = 0;$

*%% State transition probability matrix – No TMR*

$p_{\text{NoTMR}} = \mathbf{zeros}(2,2);$

*%Good State*

```
pNoTMR(2,1) = 1*A*lambda;
```

```
pNoTMR(1,1) = 1-pNoTMR(2,1);
```

```
%Failed
```

```
pNoTMR(2,2)= 1;
```

```
pNoTMR(1,2)= 0;
```

```
%% Run the Markov Chain for all times
```

```
for ind = 1:length(timeList)
```

```
%No TMR
```

```
xNoTMR = [1,0];
```

```
xNoTMR = xNoTMR*((pNoTMR')^(1000*timeList(ind)));
```

```
%TMR
```

```
xTMRnoS = [1,0,0];
```

```
xTMRnoS = xTMRnoS*((pTMRnoS')^(1000*timeList(ind)));
```

```
%TMR+scrubber
```

```
xTMR = [1,0,0];
```

```
xTMR = xTMR*((pTMR')^(1000*timeList(ind)));
```

```
%TMR+spares
```

```
x = zeros(1,29); x(1) = 1;
```

```
x = x*((p')^(1000*timeList(ind)));
```

```

failureProb = real ([ failureProb ; x(29)*100,xTMR(3)
                    *100,xNoTMR(2)*100,xTMRnoS(3)*100] );
end
end

%% Plot probably of failure vs time
figure(1)
timeList = timeList/60/60;      %Covert times from sec to hrs
semilogx(timeList , failureProb (:,3) , timeList , failureProb (:,4) ,
          timeList , failureProb (:,2) , '—' , timeList , failureProb (:,1) ,
          '-.' , 'linewidth' , 2);
legend( 'Non-Redundant' , 'TMR' , 'TMR+Scrubbing' , 'TMR+Scrubbing+
        Spares ');
xlabel( 'Hours' );
ylabel( 'Probability of Failure[%]' );

```