Trade Study and Application of Symbiotic Software and Hardware Fault-tolerance on a Microcontroller-based Avionics System

Matthew Michael McCormack, Dr. Alvar Saenz-Otero and Prof. David W. Miller

June 2011
Trade Study and Application of Symbiotic Software and Hardware Fault-tolerance on a Microcontroller-based Avionics System

Matthew Michael McCormack, Dr. Alvar Saenz-Otero and Prof. David W. Miller

June 2011

This work is based on the unaltered text of the thesis by Matthew Michael McCormack submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of Master of Science in Aeronautical and Astronautical Engineering at the Massachusetts Institute of Technology.
Trade Study and Application of Symbiotic Software and Hardware Fault-tolerance on a Microcontroller-based Avionics System

by

Matthew Michael McCormack, 2Lt, USAF

Submitted to the Department of Aeronautical and Astronautical Engineering on May 18, 2011, in partial fulfillment of the requirements for the degree of Master of Science in Aeronautical and Astronautical Engineering

Abstract

Spacecraft avionics initially commanded the development of the microprocessor industry, enabling microprocessors to be designed specifically for the reliability demands posed by the radiation environment of space. However, microprocessors have shifted their focus from ensuring reliable operations to maximizing performance, forcing the price of spacecraft avionics to balloon. Costing over three orders of magnitude more than current state of the art general purpose processors, while performing operations an order of magnitude slower. These differences stem from the reliability requirements of operating in space, typically achieved through hardware-based modifications.

While these solutions generate the necessary reliability, they limit the engineering options for the system and force the use of outdated technology. A solution researched but never widely implemented, is the use of error detection and correction software algorithms. An ideal design lies in the combination of hardware and software methods for providing reliability.

A new avionics architecture was designed to implement a system using hardware and software to achieve reliability with COTS products. The architecture was applied to the CASTOR satellite as its primary avionics system, for verification testing of the architecture’s functionality. This architecture further aims to expand spacecraft usage of microcontrollers as the primary spacecraft avionics computers.

Thesis Supervisor: Alvar Saenz-Otero
Title: Research Scientist

Thesis Supervisor: David W. Miller
Title: Professor,
Director, Space Systems Laboratory
Acknowledgments

This work was performed primarily under contract FA70001020040 with the US Air Force/Space and Missile System Center (SMC) as part of the Space Engineering Research program. The author gratefully thanks the sponsors for their generous support that enabled this research. The author also thanks SMC/XR for sponsoring summer research at LA AFB and the TAMU cyclotron in support of research for this thesis.
# Contents

1 Introduction 21  
  1.1 Overview ........................................... 21  
  1.2 Focus ............................................ 24  
  1.3 Motivation ....................................... 25  
  1.4 Thesis Overview .................................. 33  

2 Space Radiation Environment Literature Review 35  
  2.1 Space Environment ................................ 35  
    2.1.1 Heavy Ions ..................................... 36  
    2.1.2 Protons ........................................ 38  
    2.1.3 Electrons ...................................... 39  
  2.2 Radiation Effects on Processors .................... 40  
    2.2.1 Total Dosage Effects ......................... 41  
    2.2.2 Single Event Effects ......................... 42  
  2.3 Conclusion ....................................... 47  

3 Reliable Avionics Design Literature Review 49  
  3.1 Traditional, Hardware-based Radiation Hardening Techniques ........ 50  
    3.1.1 Overcoming Total Ionizing Dosage Effects ........ 50  
    3.1.2 Overcoming Single Event Effects ............... 54  
  3.2 Software-based Approaches for Overcoming Single Event Effects .... 61  
    3.2.1 Single Processor ............................... 61  
    3.2.2 Multiple Processor ............................. 65
3.3 Error Mitigation Testing ........................................ 67
3.4 Conclusion ..................................................... 67

4 Microcontrollers as Primary Avionics 71
4.1 Spacecraft Avionics Problem Statement ................. 71
4.2 Comparison of COTS and RadHard Processors ....... 73
4.3 Rational for COTS Microcontrollers in Space .......... 76
  4.3.1 Microcontroller Background Information ............. 78
  4.3.2 Justification for Microcontrollers in Satellites ... 83
4.4 Conclusion ..................................................... 86

5 Low Cost, High Reliability Avionics Architecture Design 87
5.1 Rational for Implementing Radiation Tolerance through Synthesis of Hardware and Software .................. 88
  5.1.1 Comparison with Hardware-only System .......... 89
  5.1.2 Benefits Created by Using both Hardware and Software .... 93
5.2 COTS-based Multiprocessor Architectural Requirements and Goals .......... 94
  5.2.1 Fully Capable CD&H System ......................... 94
  5.2.2 Adaptability to Different Missions ................. 98
  5.2.3 Radiation .............................................. 100
5.3 System Design Trades/Considerations .................. 105
  5.3.1 Capabilities Trades .................................. 105
  5.3.2 Hardware Design Trades ............................ 108
  5.3.3 Hardware and Software Fault-tolerance Trades ... 109
5.4 Hardware Architecture Design ............................. 111
5.5 Design Considerations for Implementing Software-based Fault-tolerance 114
  5.5.1 Satellite Software Operations ....................... 114
  5.5.2 Metrics for Applying the Proper Level of Fault-tolerance ... 116
5.6 Conclusion ..................................................... 119
6 Implementation of a Microcontroller Based Testbed

6.1 CASTOR Avionics System ............................................. 122
   6.1.1 System Operation and Design .............................. 122
   6.1.2 Hardware Design ........................................... 124
   6.1.3 Software Design .......................................... 131
   6.1.4 Key Aspects for Achieving Radiation Tolerance ......... 137
6.2 Software Fault-tolerance Algorithm Implementation ........... 144
   6.2.1 Attitude Control Operations .............................. 144
   6.2.2 Payload Operations ....................................... 146
6.3 Conclusion ....................................................... 149

7 CASTOR Reliability Testing ........................................... 151

7.1 Testing Methodology .............................................. 151
7.2 Reliability Testing Results ..................................... 152
   7.2.1 Data Corruption Errors .................................. 153
   7.2.2 Functional Unit Errors .................................... 162
   7.2.3 Hardware Reliability ...................................... 163
7.3 Other Test Results ................................................. 164
   7.3.1 Lock-step and Restart Comparison ....................... 164
   7.3.2 Reprogramming ........................................... 167
7.4 Conclusion ....................................................... 168

8 Conclusion and Future Work .......................................... 169

8.1 Multiprocessor Architecture ..................................... 170
8.2 Microcontrollers in Spacecraft Avionics ....................... 171
8.3 Future Work ..................................................... 173

A CASTOR Schematics and PCB Drawings ............................. 177

A.1 Avionics Schematics ............................................. 177
A.2 Avionics PCB Diagrams ......................................... 186
A.3 Power Schematics ............................................... 188
A.4 Power PCB .............................................. 190

B SEL Detection/Mitigation Circuit 191
List of Figures

1-1 Errors Level to be Mitigated in Spacecraft Avionics . . . . . . . . . . 24

2-1 Heavy Ion Flux versus Atomic Mass [51] . . . . . . . . . . . . . . . . 37
2-2 Solar Flares and the Solar Cycle [63] . . . . . . . . . . . . . . . . . . 37
2-3 Heavy Ion Flux versus Energy [51] . . . . . . . . . . . . . . . . . . . . 38
2-4 Van Allen Radiation Belts [61] . . . . . . . . . . . . . . . . . . . . . . 39
2-5 Proton Flux [51] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
2-6 MOSFET Structure [55] . . . . . . . . . . . . . . . . . . . . . . . . . 41
2-7 Change in N-MOSFET Gate Voltage from 500 krad Radiation [50] . . 41
2-8 Increase in Current Draw by CMOS Device from Total Ionizing Dose [4] 42
2-9 Particle Energy and LET [51] . . . . . . . . . . . . . . . . . . . . . . 43
2-10 Comparison of Single Bit Upset with Multiple Bit Upset [8] . . . . . . 45
2-11 Timing Requirements for SET to Occur [45] . . . . . . . . . . . . . . 46
2-12 Processor Die Area [26] . . . . . . . . . . . . . . . . . . . . . . . . . 48

3-1 Shrinking Processor Size in Relation to Charged Particles . . . . . . . 50
3-2 TID Rates Compared with Orbital Altitude [28] . . . . . . . . . . . . . 51
3-3 SOI CMOS Cross Section [61] . . . . . . . . . . . . . . . . . . . . . . 52
3-4 Impacts of Aluminum Shielding on TID [48] . . . . . . . . . . . . . . 53
3-5 Example of Localized Device Shielding [4] . . . . . . . . . . . . . . . 54
3-6 Comparison of RHBD Cell with Standard CMOS Cell [4] . . . . . . . 55
3-7 Comparison of CMOS NAND gate with RadHard CMOS gate . . . . . 56
3-8 Temporal Latch Functional Block Diagram [4] . . . . . . . . . . . . . 56
3-9 DICE Latch Diagram [4] . . . . . . . . . . . . . . . . . . . . . . . . . 57
<table>
<thead>
<tr>
<th>Page</th>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-10</td>
<td>TMR Functional Block Diagram [4]</td>
<td>57</td>
</tr>
<tr>
<td>3-11</td>
<td>Guard Ring and Guard Drain Diagram [46]</td>
<td>58</td>
</tr>
<tr>
<td>3-12</td>
<td>Redundant Execution [45]</td>
<td>62</td>
</tr>
<tr>
<td>3-13</td>
<td>Reduced Precision Redundancy Truth Table [52]</td>
<td>63</td>
</tr>
<tr>
<td>3-14</td>
<td>Instruction Stream Checking Process [45]</td>
<td>64</td>
</tr>
<tr>
<td>3-15</td>
<td>Distributed Fault Tolerance</td>
<td>65</td>
</tr>
<tr>
<td>3-16</td>
<td>Relation between SEE Induced Errors and Process Size [60]</td>
<td>68</td>
</tr>
<tr>
<td>4-1</td>
<td>Processor Performance Compared with Time of Delivery to Market</td>
<td>72</td>
</tr>
<tr>
<td>4-2</td>
<td>COTS and RadHard Processor Comparison</td>
<td>75</td>
</tr>
<tr>
<td>4-3</td>
<td>Further COTS and RadHard Processor Comparison</td>
<td>77</td>
</tr>
<tr>
<td>4-4</td>
<td>Components Required to Support a Processor [16]</td>
<td>80</td>
</tr>
<tr>
<td>4-5</td>
<td>Physical Size Comparison of Processors and MCUs [1]</td>
<td>81</td>
</tr>
<tr>
<td>5-1</td>
<td>Lockstep and Distributed Operations Comparison</td>
<td>88</td>
</tr>
<tr>
<td>5-2</td>
<td>TID based upon Shielding Thickness, for a 1km Circular, Equatorial Orbit</td>
<td>101</td>
</tr>
<tr>
<td>5-3</td>
<td>Operational Schedule for Reprogramming a Faulting Processor</td>
<td>102</td>
</tr>
<tr>
<td>5-4</td>
<td>Hardware Output Redundancy Circuit</td>
<td>103</td>
</tr>
<tr>
<td>5-5</td>
<td>Hardware Checking for Reliable Outputs</td>
<td>110</td>
</tr>
<tr>
<td>5-6</td>
<td>Architecture Foundation</td>
<td>111</td>
</tr>
<tr>
<td>5-7</td>
<td>Shared Peripheral Interconnections</td>
<td>112</td>
</tr>
<tr>
<td>5-8</td>
<td>Shared I/O Interface Schemes</td>
<td>113</td>
</tr>
<tr>
<td>5-9</td>
<td>Independent Peripherals</td>
<td>113</td>
</tr>
<tr>
<td>6-1</td>
<td>Testbed Design</td>
<td>121</td>
</tr>
<tr>
<td>6-2</td>
<td>CASTOR Avionics System and its Interfaces</td>
<td>129</td>
</tr>
<tr>
<td>6-3</td>
<td>CASTOR Avionics Board</td>
<td>130</td>
</tr>
<tr>
<td>6-4</td>
<td>CASTOR Software Timing Diagram</td>
<td>131</td>
</tr>
<tr>
<td>6-5</td>
<td>CAN Message Format [34]</td>
<td>132</td>
</tr>
<tr>
<td>6-6</td>
<td>DCFT Operating Window</td>
<td>147</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Measured TID Rates on Specific Satellites [28] ........................................ 51
4.1 Computational Throughput Requirements for Spacecraft [62] ...................... 74
4.2 Comparison of COTS and RadHard Products [62] ................................. 75
4.3 COTS Processors used in Comparison ...................................................... 76
4.4 Processors Evaluation ................................................................................. 86
5.1 Characteristics of Hardware-only Fault Tolerant System ......................... 89
5.2 Standard Satellite Tasks ............................................................................. 95
5.3 Bare Bones Satellite Requirements ............................................................. 95
5.4 COTS Avionics Architecture Requirements ............................................... 104
5.5 COTS Avionics Architecture Goals ............................................................. 105
5.6 COTS Avionics Architecture Trades ............................................................ 111
5.7 Standard Satellite Software Operations ....................................................... 115
5.8 Reliability Methods and their Design Impacts ............................................ 116
5.9 Reliability Levels and Sample Operations ................................................. 118
5.10 Implementation Matrix ............................................................................. 119
6.1 CASTOR Devices and Interfaces ................................................................. 124
6.2 dsPIC33 Available Interfaces ..................................................................... 125
6.3 MCU Kalman Filter Performance Test Results .......................................... 126
6.4 CASTOR Avionics Interfaces ...................................................................... 128
6.5 ThreadX Performance .................................................................................. 132
6.6 Primary CAN Message Types ...................................................................... 133
6.7 Attitude Control Tasks’ Requirements ........................................ 145
6.8 Attitude Control Tasks’ Reliability Levels ..................................... 145
6.9 DCFT Tasks’ Requirements ...................................................... 147
6.10 DCFT Tasks’ Requirements ...................................................... 148

7.1 System Characteristics for Data Value Error Detection and Correction 153
7.2 Messages Required by Modifications of Paxos .............................. 157
7.3 Message Passing Timing Characteristics for Different Data Types .... 159
7.4 Timing Characteristics of the Extended Kalman Filter Calculations . 160
7.5 Operational Bit Rate while Periodically Restarting Processor .......... 166
7.6 Time Required to Reprogram Processor ..................................... 167
7.7 Critical Design Lessons ........................................................... 168

B.1 SEL Prototype Circuit Components .......................................... 193
List of Acronyms

COTS - Commercial Off-The-Shelf
EDAC - Error Detection And Correction
RadHard - Radiation Hardened
SECDNM - Single Error Correction, Double Error Detection
TMR - Triple Modular Redundancy
RHBP - Radiation Hardened By Process
RHBD - Radiation Hardened by Design
TID - Total Ionizing Dose
LET - Linear Energy Transfer
GCR - Galactic Cosmic Ray
CME - Coronal Mass Ejection
LEO - Low Earth Orbit
MEO - Medium Earth Orbit
GEO - Geosynchronous Earth Orbit
SAA - South Atlantic Anomaly
ISS - International Space Station
CONOPS - Concept of Operations
SEE - Single Event Effects
SEU - Single Event Upset
MBU - Multiple Bit Upset
SET - Single Event Transient
SEL - Single Event Latchup
SEB - Single Event Burnout
SEFI - Single Event Functional Interrupt
CMOS - Complementary Metal-Oxide-Semiconductor
SIMOX - Separation-by-Implantation-of-Oxygen
DICE - Dual Interlocked Cell
DC - Direct Current
SIFT - Software Implemented Fault Tolerance
TTMR - Time-TMR
RPR - Reduced Precision Redundancy
LSB - Least Significant Bit
ISC - Instruction Stream Checking
ECC - Error Correcting Code
SoC - System on a Chip
CPU - Central Processing Unit
RAM - Random Access Memory
IC - Integrated Circuit
MCU - Microcontroller
RISC - Reduced Instruction Set Computer
CISC - Complex Instruction Set Computer
RTOS - Real Time Operating System
OS - Operating System
ADCS - Attitude Determination and Control System
C&DH - Command and Data Handling
IR - Infrared
DSP - Digital Signal Processor
IMU - Inertial Measurement Unit
UNP - University Nanosatellite Program
MIT - Massachusetts Institute of Technology
SSL - Space Systems Laboratory
SPL - Space Propulsion Laboratory
CASTOR - Cathode/Anode Satellite Thruster for Orbital Repositioning
SPHERES - Synchronized Position Hold Engage and Reorient Experimental Satellites
DCFT - Diverging Cusped Field Thruster
GPS - Global Positioning System
CFD - Computational Flow Diagram
ALU - Arithmetic Logic Unit
MIPS - Million Instructions Per Second
I/O - Input/Output
ICSP - In-Circuit Serial Programming
GPIO - General Purpose Input/Output
CAN - Controller Area Network
SPI - Serial Peripheral Interface
USART - Universal Synchronous/Asynchronous Receive and Transmit
SD - Secure Digital
I²C - Inter-Integrated Circuit
PWM - Pulse Width Modulated
ADC - Analog to Digital Converter
DAC - Digital to Analog Converter
DMA - Direct Memory Access
JTAG - Joint Test Action Group
GCC - GNU Compiler Collection
PCB - Printed Circuit Board
PDU - Power Distribution Unit
FPGA - Field Programmable Gate Array
JPEG - Joint Photographic Experts Group
RS - Recommended Standard
NAND - Not AND
IDE - Integrated Development Environment
AMO - At-Most-Once
CSV - Comma-Separated Values
ASCII - American Standard Code for Information Interchange
DUE - Detected Unrecoverable Error
SDC - Silent Data Corruption
SBC - Single Board Computer
RPC - Remote Procedure Call
QFN - Quad Flat No leads package
SOI - Silicon On Insulator
SOS - Silicon on Sapphire
PnP - Plug and Play
W - Watt
mW - milliwatt (1E-3 W)
nW - nanowatt (1E-9 W)
Hz - Hertz
MHz - Megahertz (1E6 Hz)
GHz - Gigahertz (1E9 Hz)
KB - Kilobyte 2^{10} Bytes
GB - Gigabyte 2^{20} Bytes
sec - seconds
msec - millisecond (1E-3 second)
µsec - microsecond (1E-6 second)
Al - Aluminum
Si - Silicon
GaAs - Gallium Arsenide
SiO_2 - Silicon Dioxide
Xe - Xenon
eV - electronvolt
rad - absorbed radiation dosage
cm - centimeter (1E-2 meter)
mm - millimeter (1E-3 meter)
µm - micrometer (1E-6 meter)
nm - nanometer (1E-9 meter)
km - kilometer (1E3 meters)
kg - kilogram
amu - atomic mass unit
mil - \frac{1}{10000} inch
bps - \frac{1}{second} bits
Q_{crit} - Critical Charge
Chapter 1

Introduction

Spacecraft avionics have traditionally been relegated to specially designed hardware in order to operate reliably in the space environment. Continuing to rely solely on hardware for reliable operations, however, will force avionics to hit a maximum performance level which cannot be exceeded due to limitations in process size reduction and transistor node spacing. This thesis analyzes the current state of spacecraft avionics and tests a new architectural design based upon redundant COTS components and relying strongly on software-based error detection and correction algorithms for handling radiation-induced faults.

1.1 Overview

Small satellites are being designed to perform more complex missions, including precision pointing, fractionated operations, and computer vision-based navigation. With these requirements comes a direct need for advanced processors that can reliably operate in the space environment. The traditional approach for ensuring high level reliability has been through hardware-based Radiation Hardening (RadHard) techniques. These techniques primarily rely on larger process sizes, redundant transistors within logical elements, and complete circuit redundancy. While high reliability can be obtained, the performance levels fall over an order of magnitude below that of current state of the art components. Complex mission concepts can be successfully
demonstrated in a laboratory environment but are unrealizable in space because the hardware is not robust to radiation-induced errors.

In an effort to bring higher performance to space applications, designers employ complete hardware redundancy of non-RadHard processors which vote on outputs prior to execution. Voting increases the reliability of the system while the redundant hardware allows the system to utilize the higher performance capabilities of non-RadHard processors. This is the primary method employed by the spacecraft industry for using commercial, off-the-shelf (COTS) processors in the space environment. As process sizes continue to shrink, it has brought about a change in the types of faults that are experienced on-orbit. As a result, new methods are required for detecting and correcting radiation induced faults.

Radiation effects have traditionally been mitigated through hardware. A large portion of the fault tolerance achieved in hardware can be replicated in software; however, the usage of software for fault detection and correction by the spacecraft industry has been minimal. As the error paradigms change, based upon the smaller process sizes, software becomes a key similarity between generations of hardware, making it an ideal area for applying error detection and correction. An optimal design could utilize complimentary hardware and software methods for fault detection and correction.

Contemporary spacecraft are computationally limited by the performance capabilities of RadHard processors. Top-of-the-line COTS processors provide greater computational capabilities but lack the reliability required for operations in the radiation environment of near-Earth space. In order to bridge this gap, different software based fault-tolerance techniques have been devised; however, only a few have been implemented. This thesis implements and compares the plethora of reliability options available to designers of spacecraft computer systems.

This thesis juxtaposes many of the RadHard design and software fault-tolerant algorithms to form a highly reliable, COTS based spacecraft avionics architecture. The goal of this architecture is to increase performance, while decreasing power consumption and cost. As the COTS parts are anticipated to fail over time, the architecture
aims to ensure graceful degradation whereby the system maintains functionality (possibly at a degraded state) after parts of the avionics system fail.

This architecture is designed by first analyzing the environmental factors which must be overcome in space applications. Next, a survey is taken of the currently available software-based fault-tolerant methods used to ensure reliable operation within this environment. From this knowledge baseline, constraints on the architecture and testing goals are outlined. Finally, a testbed is formulated to allow for analysis of the designed architecture. This testbed allows for a trade study to be conducted whereby the advantages and disadvantages of different software fault-tolerant methods can be evaluated on a common platform for accurate comparisons. Finally, conclusions are drawn with consideration of the utility of this architecture and any further design optimizations that could be pursued.

This is the foundation for the avionics system of MIT’s entry into the University Nanosatellite Program’s (UNP) 6th intercollegiate competition, named Cathode/Anode Satellite Thruster for Orbital Repositioning (CASTOR). The avionics architecture employs redundant processing units linked over a common bus and runs different fault-tolerance algorithms to obtain each operation’s needed reliability. To supplement this, critical input or output (I/O) operations require the coordination of multiple processing units prior to externally visible actions. The design allows for reliability to be traded with computational performance on a per operation granularity. By relying heavily on software to achieve fault tolerance, it adds robustness to the design that is not based upon silicon configuration, simplifying the exchange of hardware components. Thus, the CASTOR avionics architecture enables modern hardware to reliably operate in the space environment, bringing greater processing power to small satellite designs while also broadening the design space by allowing processing power to be traded for reliability level.

Furthermore, this thesis looks to address the following issues: the implementation of microcontrollers as primary processors for small satellites, the changing paradigms imposed by modern microelectronics, and what future systems will require to ensure reliable operation.
1.2 Focus

This thesis focuses on the field of reliable computing for space processors by providing a standardized comparison of the major fault tolerant methods for overcoming single event effects (SEE) as this is the radiation hazard which is most difficult to avoid [12]. In particular, this thesis will focus on SEE errors which cause lasting and transient data corruption such as single event upsets (SEU), multiple bit upsets (MBU), and single event transients (SET). Refer to Figure 1-1 for the scope of errors this thesis investigates. Most prior studies focus on the implementation of a single fault-tolerance methodology and compare it with the non-fault tolerant implementation of the system. Moreover, each study conducts its testing on different processor cores, which operate differently and make comparison of different methodologies challenging and error-prone.

![Figure 1-1: Errors Level to be Mitigated in Spacecraft Avionics](image)

As a complement to this focus on comparing software fault-tolerance algorithms, this thesis focuses on avionics architectural design by developing and testing a new, COTS-based architecture. This design discusses the advantages of utilizing redundant microcontrollers as the primary processors in small spacecraft. Microcontrollers have rapidly grown in capabilities and now have the ability to implement the primary functions required by small spacecraft. Microcontrollers are also well suited to perform the processing needs of spacecraft avionics systems. They operate at extremely
low power levels, are very customizable, have hardware defined interfaces that are easily utilized, and are fabricated on larger process sizes which is better understood by the radiation effects community.

As feature sizes continue to shrink and multiple node upsets become more common, the old standards of combating radiation-induced errors becomes less relevant [11]. A thorough understanding of radiation-induced effects on older generation microelectronics exists. Thus, mitigation schemes have been developed to combat errors in these devices; however, the schemes primarily employ physically changing the device. The radiation-induced effects on newer technology is not fully understood which has created a situation where previously used assumptions no longer apply because the minimum transistor spacing has become so minute. Thus the old error mitigation schemes must be altered to account for the new challenges or new methods must be developed. Towards this end, schemes using both hardware and software are being investigated for their widespread applicability and adaptability. In particular, this approach to achieving reliability is viewed as the optimal design for incorporating modern processors into spacecraft.

In summary, the main focus items of this thesis are the comparison of software fault-tolerance algorithms for their effective strengths and weaknesses. The design and implementation of a new spacecraft avionics architecture, aimed at increasing performance and adaptability while decreasing cost and power consumption, is developed in order to test these algorithms.

1.3 Motivation

The radiation environment of space creates two negative effects on processors: accumulation of total ionizing dose (TID) and SEE. TID is a measure of the charge gradually accumulated within a transistor, which will cause a shift in the transistor’s required gate threshold voltage. This effect can only be mitigated through hardware based mechanisms that remove the excess charge; however, the TID accumulation is gradual enough that it does not impact most small satellite missions (based upon
mission orbit and duration). Additionally, as COTS processor architectures utilize smaller process sizes they innately increase their TID tolerance capabilities because there is less area for charges to collect. As such, the mitigation of this effect will receive minimal consideration in the development of the COTS-based avionics architecture designed this thesis. Instead, the focus will be on the instantaneous changes in transistor state caused by the impingement of a single charged particle with the transistor. Due to decreasing process sizes, modern processors suffer from an increased occurrence of SEE induced errors [4], which is what this thesis will target. The development of error mitigation schemes for a new, COTS-based avionics architecture.

Traditionally, SEE have been viewed as only impacting a single transistor. However, as process sizes are reduced below the 90 nm process size the probability for multiple node upsets greatly increases. While the charged particle radius (<300 pm) is more than six orders of magnitude smaller than the transistor area (∼1 μm), the distance a charged particle will travel through the silicon substrate (1-8 μm) is within the same magnitude as the transistor area. Thus when a charged particle’s incidence angle with the silicon substrate is not zero, there is a possibility for that particle to impinge multiple transistor nodes. As the transistor size shrinks, a single charged particle becomes capable of upsetting multiple transistor nodes since its path length through the silicon substrate remains constant while a lower incidence angle produces a greater number of interactions. Therefore particles with a shallower angle of incidence are capable of causing multiple bit upsets as opposed to a single transistor upset in larger process sizes. Overcoming this dilemma is one of the primary challenges facing modern processors in space applications.

SEE and TID can be mitigated through the utilization of special fabrication techniques when creating the processor die, such is the case with RadHard components. Device-level hardening techniques are implemented in order to limit charge buildup within the silicon substrate of the processor as well as mitigating the effects of any single transistor incorrectly changing its state. Techniques include using a larger transistor size (larger process sizes), increasing the operating voltage, decreasing the thickness of the gate oxide layer, and adding extra transistors to the logic circuits.
Increasing transistor size reduces chances of errors; however, it increases the power consumption and lowers the performance capabilities. Embedding an oxide layer within the substrate reduces the amount of charge that can be stored within a transistor substrate and mitigates the bias shift that leads to device failure. All circuits are designed to withstand a single transistor inaccurately changing states. Therefore, in order for the circuit to change logical output levels, the state of two separate transistors must be changed, providing strong protection against single transistor upsets.

Implementing these design parameters does not come without a cost. First, there are only a few foundries that are capable of producing dies at RadHard specifications which greatly increases the unit cost of the electronics. Second, many of these techniques are in direct contention with performance. Lower operating voltages and smaller process sizes were the key enablers of the rapid processor performance increases of the past two decades. Hence these two modifications limit the maximum operational clock frequency that can be utilized. Third, these techniques increase the power consumption for standard operation of the device. As a result RadHard components have a higher price and lower performance than their COTS equivalents.

The current state of the art RadHard processor is the BAE RAD750 which is based upon the PowerPC 750 architecture. In performing these modifications, the maximum clock rate that is achievable on the RAD750 is 200 MHz, while the standard PowerPC 750 core can operate at up to 1 GHz. This reduction in clock speed contributes to an order of magnitude reduction in performance. Additionally, the RAD750 costs three orders of magnitude more than the PowerPC 750 [53].

In an effort to bridge these two extremes, high reliability of the RAD750 and high performance (and low cost) of the standard PowerPC 750, Boeing developed a method for using fully redundant hardware. Boeing’s design utilizes three PowerPC 750s running in tight synchronization with their output buses being verified against each other to check for faults. If a fault is detected, all three processors save the majority state to nonvolatile memory and restart in order that a known state can be achieved in all processors upon restart. Furthermore, these resets occur periodically to ensure non-visible errors do not accumulate [10].
Boeing’s design greatly increases the computational throughput which can be reliably utilized in the space environment; however, due to the periodic resets every second, the processor does not fully achieve the maximum performance of the standard PowerPC 750 because the processor state must be continually saved and reloaded from nonvolatile memory. This further limits this architecture from being used as most satellite’s primary satellite control operations as those operations require continuous operation. However, using COTS processors pushes against the standard spacecraft avionics design template of only using RadHard components in order to utilize the increased performance available in modern processors.

The future of spacecraft systems requires that their computational capabilities be improved beyond what RadHard components are currently capable of delivering. Seeing that the spacecraft avionics constitute less than five percent of the monetary expenditures into the entire microelectronics market, one area of the microelectronics market underutilized by spacecraft avionics engineers is COTS products [29]. These products offer a large expansion of capabilities; however, they are currently not utilized as they are not inherently reliable in the space environment. To utilize the capabilities of COTS, alternate methods of achieving reliable operations must be investigated and tested to quantify their operational traits. In particular, further analysis on their impacts on system performance and reliability must be performed. The challenges of making modern processors reliable in the space environment cannot be done exclusively in hardware. As a result new approaches must be tried and tested, such as using both hardware and software to provide increased reliability.

An alternative approach to achieving reliable operations is through the use of software algorithms running in the background of all operations. A traditional application of software algorithms for reliability is memory scrubbing where the Hamming checksum is periodically computed for entries in memory, thereby inhibiting upsets in the memory from accumulating. Furthermore, it allows the protection level to be traded with the frequency of scrubbing. Beyond simply protecting memory, algorithms have been designed to ensure that the program execution follows the proper order and that the result is not influenced by an upset.
One method, Software Implemented Fault Tolerance (SIFT), utilizes two separate algorithms to monitor both the program flow and the result calculation. For protecting the program flow, all the program instructions are grouped based on their locality in memory (i.e. between program counter jumps). The groupings are calculated at compile time, and are verified by ensuring that whenever a jmp instruction is executed, that the address being jumped to is allowable from the current address. SIFT ensures that the results produced by operations are correct by redundantly executing the commands and comparing their results. This is accomplished by having the compiler replicate all commands (with the results being stored in separate memory locations) and periodically add in points where the results of a string of redundant commands are compared with each other. A minimum of three separate executions are utilized in order to ensure that once an error is detected it can be corrected [24]. Other methods for checking calculated results exist, but are based upon the same concept of redundantly executing the commands and comparing the results.

SIFT was tested on the ARGOS satellite for 136 days in a low Earth orbit. The ARGOS satellite allowed the same program to be executed simultaneously on a RadHard processor and on a COTS processor. The results were compared where the RadHard processor was assumed to always be correct. During on-orbit testing, 203 errors were detected by SIFT and 198 errors were corrected. SIFT reliably mitigated 97.5% of errors that occurred [47].

Another area of significant research deals with creating reliable code for Field Programmable Gate Arrays (FPGAs). Programming an FPGA entails outlining the logical path a signal takes through the device’s logic gates. Reliability can be created by having common inputs follow spatially separated routes that simultaneously traverse the same logical function gates. The output from these gates can be compared to each other to check for errors created by radiation induced upsets. The number of common routes and the logical operations being redundantly executed can be varied for differing levels of reliability. These methods are growing in use because developers can utilize COTS hardware and implement fault tolerance in areas of the operation where it is needed. However, one challenge to achieving reliability is that
FPGA routing algorithms are optimized to reduce latency and to minimize gate usage. These algorithms often generate routes with redundant paths that are not isolated, potentially allowing a single charged particle to upset multiple paths, overcoming the redundancy.

The use of software for creating the needed reliability is a growing area in spacecraft design. The greatest impediment however, is that even the most robust software will fail if it is based on a single hardware unit that could fail due to other environmental influences. Redundant hardware coupled with fault-tolerant software algorithms could be a potential optimization.

The proposed avionics architecture attempts to utilize the best fault-tolerant characteristics in existing systems: redundant hardware (as seen in Boeing’s design and the FPGA designs) for performance which are interconnected to allow cooperation and collaboration between processing units, and a strong utilization of software for fault tolerance where separate software algorithms (as seen in SIFT) can be used. The architecture allows for trades to be made between the hardware and software for reliability and performance.

The proposed avionics architecture relies upon multiple interconnected processors to form a distributed processing architecture. These separate processors allow for spatial separation, thus ensuring that it is impossible for a single charged particle to cause upsets within two separate processors. This spatial redundancy enables advanced processors to be incorporated into spacecraft designs and grants them access to the high performance computing necessary for modern complex missions. Additionally, having multiple processors allows for a processor to fail without jeopardizing the mission as other processors can fulfill (potentially in a degraded form) the critical mission requirements.

The architecture utilizes a minimum of three processors for error detection and correction; however, additional processors could be incorporated for further reliability or performance needs. For example, in order to prolong the system’s operational lifetime, extra processors can be incorporated into the design but left in a powered off state, making the radiation effects of the space environment negligible, until needed.
to replace a failing processor. Additionally, extra processors can be incorporated for high throughput computations or multiple/complex payloads and re-tasked in the event that a processor responsible for mission-critical operations fails.

The processors are connected in a ring network, allowing any processor to communicate with the two closest processors. This setup generates maximum reliability as any combination of processors can be used to check each other for faults while minimizing the required interfacing hardware. Having a minimum of three processors allows for the highly critical operations to be computed on three separate processors whereby any fault encountered on one processor can be corrected by the other processors. Having I/O accessed over the bus allows for tasks to be distributed across different processors, thereby producing the capacity for load sharing. Additionally it enables graceful degradation as a processor failure does not automatically imply that the I/O is no longer accessible. The consequences of utilizing a distributed topology not only increases reliability but it also aids its performance capabilities by allowing tasks to be exchanged between processors.

Because the proposed architecture relies heavily on distributed processing units, the optimal design needs a new kernel structure to fully utilize the available processors. A multikernel design is ideally suited because it relies on message passing between processors where all state data is replicated across the network of processors [7]. This kernel is then responsible for maintaining timing requirements on each processor as well as monitoring the state of other processors. If a processor on the network is detected to be malfunctioning, the kernel will re-allocate the tasks that were scheduled to occur on the failed processor, while simultaneously attempting to bring the faulty processor back to full functionality. While the system performance and capabilities may be momentarily reduced, the ability to bring a failed processor back on-line far surpasses these transient system degradations. Additionally, each processor must be ready to execute remote procedure calls for redundantly executing another processor’s operations which for high reliability require execution on separate processors.

Each operation specifies the level of reliability that it requires, and thus the software algorithm(s) being used to validate its outcomes. Processes can choose to operate
with no protection and simply execute the function once on a single processor, granting maximum performance to that operation. Alternatively, processes can run with maximum reliability by executing the process redundantly on multiple processors and then ensuring that a consensus is reached amongst the multiple processors before a final outcome is committed. Spatial and temporal redundancy can be utilized for ensuring that any radiation induced faults are detected and corrected by the system, at a cost of degraded operational throughput. A vast array of options exists between these two extremes. One is redundantly executing operations on the same processor until a fault is detected at which point the operation is sent to other processors for error correction. A second is to check the operation’s program flow to ensure that no illegal jumps are executed and restarting the operation if such an event is detected. This ability to define the fault tolerance level in software greatly alters the design of the avionics system.

A traditional system would require a separate processor for operations requiring high reliability or the entire system would be forced to operate within the constraints of the process with the highest reliability demands. In general these reliability demands impact the system performance and design costs because highly reliable processors are performance limited and have high component costs. By granting each operation its own reliability level, a system design is granted a larger array of options. Among these options is the capability to trade system performance for operational reliability, thereby allowing designs to be optimized for the given mission parameters rather than constrained to the limitations posed by RadHard components or the reliability needs of critical operations.

One key area where this architecture could be initially tested is using microcontrollers for the spacecraft avionics. Microcontrollers are ideal because they operate at a high point in the performance per power used spectrum, are well suited for rapid design and prototyping, are manufactured with a wide array of standard interfaces, and are extremely low cost. However, the ultimate adoption of microcontrollers into space systems design requires the advancement and testing of alternate fault-tolerance techniques to ensure that they contribute the needed reliability level.
1.4 Thesis Overview

This thesis discusses the design of a COTS-based avionics system, from the challenges and errors induced by operating in the space environment, to an architecture for implementing COTS processors, to testing a prototype COTS avionics system. Chapter 2 gives a background on the space environment and the types of errors it generates in spacecraft avionics systems. Followed by a literature review of the different techniques and processes that have been utilized for mitigating these errors in Chapter 3. Next, Chapter 4 discusses spacecraft avionics and makes a case for utilizing COTS microcontrollers as the primary spacecraft computers. Using this background knowledge about the environment and spacecraft avionics, Chapter 5 develops an architecture which symbiotically incorporates COTS hardware and fault-tolerant software algorithms to combat the errors generated by the space environment. This architecture aims to exceed common design metrics for RadHard processors (i.e. higher performance, lower power consumption) while providing the same level of fault tolerance.

To verify and test the validity/applicability of this new architecture, a testbed was implemented around the CASTOR satellite. The details of this implementation are discussed in Chapter 6. Followed by a characterization of the reliability and performance of the avionics system in Chapter 7. Different EDAC algorithms ability to prevent data corruption while executing various satellite functions was tested by using software to injecting errors into the processor’s state. The thesis ends with final design suggestions on how future spacecraft avionics systems can be designed to utilize the full potential of current and future COTS processors.
Chapter 2

Space Radiation Environment

Literature Review

The engineering options available for spacecraft avionics are limited to components capable of operating reliably in the space environment. One way of broadening the available options are through the use of COTS products employing alternative fault-tolerant methods. An understanding of the near-Earth space environment and its effect on the operation of microelectronic devices is required to select the optimal fault-tolerant method. This knowledge allows for the algorithms to be optimized for mitigating specific risks.

The radiation environment encountered by an Earth orbiting spacecraft is dynamic and poses many challenges to reliable computing. The environment creates two major challenges: ensuring lifetime operations by limiting the electronics’s degradation caused by the accumulation of charged particles, categorically referred to as TID, and accurately performing computations, by overcoming the instantaneous transistor state changes caused by charged particles, categorized as SEE.

2.1 Space Environment

The radiation environment of near-Earth space consists of heavy ions, protons, and electrons coming from galactic cosmic rays (GCR) and solar emissions. GCRs are
composed of \( \sim 87\% \) protons, \( \sim 12\% \) Helium (He) nuclei (\( \alpha \) particles), and the remaining \( (<1\%) \) are heavier element ions, electrons (in the form of \( \beta \) particles), x-ray photons, and gamma ray photons [61]. Solar emissions are primarily composed of protons, with a trace amounts of elemental ions. Many protons and electrons become trapped by the Earth’s magnetic field lines, increasing the concentration of charged particles. The particle flux is highly dependent upon the activity on the Sun which follows the solar cycle. The solar cycle’s primary period is approximately eleven years, with two major divisions. Solar maximum lasts for approximately seven years; while solar minimum lasts for approximately four years and results in a reduction in the particle flux by approximately two orders of magnitude.

Flux measurements are grouped by energy levels, measured in electron volts (eV) which is the kinetic energy gained by a single electron being accelerated by a 1 V electric field, with 1 eV = 1.602E-19 J (equivalently 6.24E+18 eV = 1 J). A particle’s energy level is directly related to the amount of charge it will deposit. Passive particle energy measurement devices (such as thermoluminescent dosimeters, particle track detectors, nuclear emulsions, and bubble detectors) record integrated radiation data by accumulating effects from radiation events. While active devices (such as particle dosimeters, particle telescopes, and charging monitors) collect temporal and spatial data from the instantaneous radiation effects [29]. From these measurements, three types of charged particles have been identified as the primary creators of errors in spacecraft avionics: heavy ions, trapped protons, and trapped electrons.

2.1.1 Heavy Ions

Heavy ions are the primary contributors to radiation effects occurring in spacecraft electronics. The category of heavy ions encompasses charged particles from any naturally occurring element in the periodic table, with the majority having atomic masses less than that of Oxygen at 16 atomic mass units (amu). The elements with the highest concentrations are shown in Figure 2-1.

Heavy ions originate from GCRs and the Sun. In general, when the Sun emits particles it is referred to as a solar flare; however, some emissions contain a higher
concentration of charged particles at significantly increased energy levels and are referred to as coronal mass ejections (CME). During a CME the Sun emits an average of $\sim 1.6 \times 10^{12}$ kg of charged particles at an average velocity of $\sim 489 \frac{\text{km}}{\text{s}}$ [61]. The occurrence of a CME depends upon the solar cycle. In solar minimum there are relatively few solar flares or CMEs (on the order of one every day); while solar flares and CMEs are significantly more common during solar maximum (occurring every two to four hours), as shown in Figure 2-2.
Heavy ions are the highest energy particles, generally having $>100\text{MeV/nucleon}$, making them the largest contributor to radiation-induced errors in current microelectronics [61]. A sample spectrum of heavy ion flux versus energy is shown in Figure 2-3. The flux of these charged particles increases proportionally with orbital altitude. Additionally, there is a higher concentration in the regions surrounding the Earth’s magnetic poles due to the interaction between the particles’s charge and the Earth’s magnetic field lines.

![Figure 2-3: Heavy Ion Flux versus Energy [51]](image)

2.1.2 Protons

Trapped protons originate from GCRs and solar flairs/CMEs. The protons become trapped in the Earth’s magnetic field lines, creating an oscillating band of charge surrounding the Earth. Thus the flux of trapped protons follows the Earth’s magnetic field lines, with a higher flux closer to the Earth’s magnetic equator and negligible amounts at higher orbital inclinations ($>60^\circ$). The trapped proton flux forms two regions which are collectively called the Van Allen radiation belts. The first region (the inner belt) exists from 1,000 to 10,000 km, which spans low Earth orbit (LEO)
and the lower portion of medium Earth orbit (MEO). The second region is from 13,000 to 19,000 km, is completely encompassed by MEO. The concentration of charged particles is fairly constant along rings of constant altitude and inclination (refer to Figure 2-4(a)) with one exception. In a region above the Atlantic Ocean off the coast of South America, the geomagnetic sphere is closer to the Earth (starting at \( \sim 500 \) km), causing the proton flux to increase by over four orders of magnitude; this region is known as the South Atlantic Anomaly (SAA).

![Van Allen Radiation Belts](image)

(a) Radiation Intensity  
(b) Belt Locations

Figure 2-4: Van Allen Radiation Belts [61]

Protons are primarily trapped in the inner Van Allen radiation belt (seen in Figure 2-4(b)). These protons typically have energy levels \( >10\text{MeV} \) [61], with the highest energy protons being trapped closest to the Earth’s surface. This occurs because protons can only maintain orbits in the stronger magnetic field present at lower altitudes. A complete characterization on the trapped proton flux based on energy level and altitude is shown in Figure 2-5.

### 2.1.3 Electrons

Trapped electrons are found in the outer Van Allen radiation belt (refer to Figures 2-4(b)) and contribute to eventual device failure through accumulated TID. However, they have no appreciable impact on SEE-induced errors in today’s spacecraft electronics because their energy levels are an order of magnitude below the other types of
charged particles. The vast majority of trapped electrons can be mitigated by minimal metal shielding (∼1mm of Aluminum). Thus the radiation effects of trapped electrons on microelectronics will not be discussed further in this thesis because current spacecraft design standards effectively mitigate their effects.

2.2 Radiation Effects on Processors

An interaction between charged particles (protons and heavy ions) and electronic components results in two different effects: permanent operating threshold changes and transient state errors. The permanent changes from the TID alter the voltage thresholds a transistor operates at, eventually leading to permanent device failure. The temporary impact of SEE cause a momentary change of the state of a transistor’s gate (i.e. turning an “on” transistor “off”) resulting in the wrong logic level being represented. This thesis will primarily focus on SEE.
2.2.1 Total Dosage Effects

Figure 2-6: MOSFET Structure [55]

TID refers to the slow accumulation of charge in a device’s oxide regions (SiO$_2$, the blue layer in Figure 2-6) and substrate. As this charge accumulates, a transistor’s gate voltage threshold shifts, which results in the logic level being shifted (by this DC bias voltage). Eventually, this threshold change will impact the circuit operations by inhibiting the transistor from switching (change states) because the required gate voltage has shifted outside of the operational limits of the device, this threshold shift is shown in Figure 2-7.

Figure 2-7: Change in N-MOSFET Gate Voltage from 500 krad Radiation [50]

TID also increases a device’s leakage current (wasted current that does not contribute to device operation). As the threshold voltage changes gradually the transistor
can remain controllable, but the power required to operate the transistor will dramatically increase (as seen in Figure 2-8). Since all of a device’s transistors degrade simultaneously, the potential for a system to fail from TID increases proportionately with time on orbit.

![Figure 2-8: Increase in Current Draw by CMOS Device from Total Ionizing Dose [4]](image)

Ability to withstand TID, the amount of energy that can be absorbed prior to device failure, is measured in kilorads (krads) for a specific substrate (typically silicon). A krad is equivalent to 6.24E7 MeV (10 µJ) absorbed per gram of substrate. Rad-Hard components typically have a total dosage hardness of 100 to 1,000 krads (Si). As CMOS feature sizes are reduced, the amount of TID they can absorb increases [4].

### 2.2.2 Single Event Effects

SEE are the effects induced in semiconductor devices by the interaction of a single heavy ion or proton with the device. In a dynamic process lasting <10 picoseconds, the charged particle will pass through the substrate freeing various electrons. In a process called nuclear spoliation, energy is transferred to the substrate when a charged particle collides with substrate atoms through an elastic scattering collision mechanism, whereby the charged particle liberates inner shell electrons (the K, L, or M bands) from the substrate atoms. In a process called direct ionization, the particle deposits a trail of charge along its path through the device. Freed electrons travel radially from the charged particle’s path, coming to rest after losing energy
through collisions with valence electrons in substrate. These liberated electrons will decay through diffusion, drift, and recombination mechanisms, eventually recombining or reaching the transistor node and modifying its voltage. If enough charge from the electrons can be transferred to a transistor node a SEE will occur, causing the transistor to change state.

As process sizes shrink and the operational voltages decrease, the amount of charge required to create SEE is diminished. While particles may have differing amounts of energy, the critical factor in determining if a SEE will occur is the amount of energy (charge) transferred from the particle to the substance it contacts [39]. Linear energy transfer (LET) is the measure of energy deposited per unit distance a particle travels, generally in units of millions electronvolt centimeters per milligram $\frac{MeV \cdot cm}{mg}$. Normalizing LET by assuming the particle will encounter a constant substance allows for different charged particles to be equivalently compared. The relation between particle energy and LET is shown for various particles in Figure 2-9.

![Figure 2-9: Particle Energy and LET](image)

A charged particle will deposit different amounts of charge in different microelectronic substrates (e.g. silicon, gallium arsenide, etc.) due to differences in atomic structure and density. Additionally, the speed/energy of the particle is directly re-
lated to the amount of charge it can deposit. A particle with little energy has a lower velocity, allowing it more time to interact with the microelectronics and deposit more charge; thus high speed particles do not deposit large amounts of charge because of the short duration of their interaction with the material [61]. Particles will deposit their maximum amount of charge when they come to rest in the substrate [8], this point is referred to as the Bragg peak (noted in Figure 2-9 by the red dots).

While the flux of heavy ions is significantly below that of protons (on average by nearly two orders of magnitude, reference Figure 2-3), ion’s LET are typically two or more orders of magnitude greater than those of protons (reference Figure 2-9) [51]. Thus while there is a lower probability of a device interacting with ions than protons, the ions’s LET greatly increase the probability of causing a SEE compared to protons.

For a charged particle’s interaction with the microelectronics to create a momentary effect in the transistor’s logical output. A transistor gate must accumulate sufficient charge$^1$, and thus voltage, to reverse the current state of the channel. This amount of charge is referred to as $Q_{\text{crit}}$, which is determined by the operational voltage of the transistor and the transistor’s gate size. Equation 2.1 calculates the charge needed for a standard transistor while Equation 2.2 is for circuits with active feedback against upsets [8]. From Equations 2.1 and 2.2 one notes that as process sizes shrink (capacitance decreases) and the operational voltages decreases, $Q_{\text{crit}}$ also decreases.

\begin{align*}
Q_{\text{crit}} &= C_{\text{node}}V_{\text{node}} \quad (2.1) \\
Q_{\text{crit}} &= C_{\text{node}}V_{\text{node}} + \tau_{\text{switch}}I_{\text{restore}} \quad (2.2)
\end{align*}

$C_{\text{node}}$ = Capacitance between transistor nodes  
$V_{\text{node}}$ = Operating voltage of transistor nodes  
$\tau_{\text{switch}}$ = Logic switching delay time  
$I_{\text{restore}}$ = Feedback current supplied

SEE are characterized by the transistor’s effect on the device. In a latching circuit, a SEE can cause the wrong value to be stored and thus produce an error lasting until

$^1$Simple conversions between LET and $\frac{\text{fC}}{\mu\text{m}}$ can be found in [61].
the value in memory is corrected/modified. While a SEE in a combinatorial circuit can create a transient error in a current operation. These errors can propagate through the device’s logic stream and lead to errors such as a pin outputting an incorrect value or latching incorrect data. Once these errors produce an externally visible output, it is considered a SEE induced upset. SEE can take several forms [4].

- **Single Event Upset (SEU)**

  The most common SEE-induced error is called a single event upset. An SEU occurs when a single energized particle causes one or more memory cell(s) (bits) to change state. There are two categories of SEU illustrated in Figure 2-10.

  ![Comparison of Single Bit Upset with Multiple Bit Upset](image)

  **Figure 2-10: Comparison of Single Bit Upset with Multiple Bit Upset [8]**

  - *Single Bit Upset (SBU)*

    A single bit upset occurs when a single logical value (i.e. 1 or 0) being stored in a memory element is permanently changed, shown on the left of Figure 2-10. An SEU has the largest impact when it alters a configuration register, which can cause improper device behavior/functionality. To mitigate this effect, the value in the memory element must be externally reset. This correction is most commonly done through error detection and correction (EDAC) codes.
- **Multiple Bit Upset (MBU)**

A multiple bit upset occurs when a single energized particle causes more than one memory cell (bit) to change state, shown on the right of Figure 2-10. These types of errors are particularly challenging to correct when the bits upset are within the same word as standard error correcting schemes only compensate for SBU in a data word. MBUs can be mitigated in the same way as SBU, by using an EDAC algorithm to rewrite the value. However, these algorithms are not widely used because of their overhead and the extremely low probability of MBUs on the larger process sizes typically used on spacecraft electronics.

- **Single Event Transient (SET)**

A single event transient is a temporary change in the logic level of a circuit within a device. If this logic level is latched into a memory cell, it becomes an SEU. Most SETs have a small amplitude and duration allowing them to decay before they propagate and thus appear as noise within the signal. Figure 2-11 demonstrates the timing dependencies of a SET causing an externally visible error.  

![Figure 2-11: Timing Requirements for SET to Occur](image)

- **Single Event Functional Interrupt (SEFI)**

A single event functional interrupt is an upset of an internal memory element or a circuit which causes a loss in the device’s functionality. For example, a modi-
ification of a device’s interrupt control register could trigger an unimplemented interrupt routine to execute, potentially leading to complete loss of functionality because an interrupt is continually triggered and cannot be cleared. A SEFI may be corrected by rewriting the modified memory element or by power cycling the device.

- **Single Event Latchup (SEL)**

  A single event latchup is a lockup of a functional circuit within a device, associated with a sudden and large rise in the device’s supply current. Standard operating characteristics can only be returned by power cycling the device. Depending on the amplitude and duration of the latchup, the device may experience permanent damage from the increased current flow.

- **Single Event Burnout (SEB)**

  A single event burnout is the destruction of an element (e.g. semiconductor junction) in a device from excess current flow induced by a charged particle. This typically occurs in power transistors and is similar to a SEL; however, an SEB is differentiated by uncorrectable device failure.

### 2.3 Conclusion

The space environment is composed of heavy ions, trapped protons, and trapped electrons. Of these particles only heavy ions and trapped protons contribute to SEE-induced errors. The chance of an error occurring is based upon the quantity of charge transferred and the device’s $Q_{\text{crit}}$. Within the category of SEE there are several different types, each requiring slightly different mitigation approaches. The primary types targeted by this thesis are the most common SEU and SET (see Appendix B for a brief discussion and circuit design for mitigating SEL).

SEUs and SETs are most likely to occur in a processor’s memory arrays because these consume the most die area. Additionally, memory arrays are fabricated on smaller process size transistors than the rest of the processor. As demonstrated by
the Intel Pentium M processor die seen in Figure 2-12, a large portion of the processor die’s area is composed of memory storage arrays. The predominant memory array for modern processors is the L2 cache, equivalent to the random access memory (RAM) array of a microcontroller. The L2 cache in Figure 2-12 consumes 44.5% of the die area, this large area greatly increases the chance of a high energy particle inducing an upset within this region as opposed to the 12% consumed by the arithmetic logic unit (ALU), a part of the processor’s core [26].

Thus protecting a processor’s memory arrays from the effects of the space environment pays the largest dividends when attempting to create a reliable processor for use in the space environment. The techniques used for mitigating SEUs and SETs will be discussed in the next chapter.
Chapter 3

Reliable Avionics Design

Literature Review

The flux of high energy protons and heavy ions in the space environment cause microelectronics to experience random errors. As the feature size of modern microelectronics (processors and memory especially) continues to decrease, ensuring reliability has become more demanding. Historically direct ionization by heavy ions with an LET $>1 \frac{MeV \cdot cm^2}{mg}$ was the dominant cause of errors, while protons with an LET $<0.5 \frac{MeV \cdot cm^2}{mg}$ were minor contributors. However, as transistor’s size and operating voltage are continually reduced, new challenges are arising for developing electronics which can reliably operate in the space environment.

The continual reduction of transistor’s gate size and operating voltage have enabled increases in microelectronic’s transistor density and performance capabilities. Modern transistors have a node size $<90$ nm and operate at $<1$ V, making $Q_{crit} < 1$ fC. These high transistor-density devices allow multiple transistors to fall inside the ionization path of heavy ions, increasing the probability of MBUs, Figure 3-1 provides a visualization of this effect (where the red area is a transistor node). Additionally, the lower $Q_{crit}$ allows low energy protons which in the past created negligible impacts to directly ionize transistors thereby causing upsets. With the higher flux of protons than heavy ions (by $\sim 2$ orders of magnitude, reference Figure 2-3) the frequency of upsets is expected to increase. This has created the need for new fault mitigation
schemes which operate under these new assumptions of correcting MBUs and operating with an increased upset frequency. The design and operation of existing methods will be evaluated for potential contributions to future techniques.

![Figure 3-1: Shrinking Processor Size in Relation to Charged Particles](image)

### 3.1 Traditional, Hardware-based Radiation Hardening Techniques

The first SEU was observed on Hughes’s Syncom2 satellite in 1964. From then onward, spacecraft avionics were designed to overcome the radiation effects of the space environment. The primary strategy utilized was hardware redundancy, from the transistor level to the circuit board (or unit) level. Hardware redundancy has been employed to minimize the effects of both TID and SEE.

#### 3.1.1 Overcoming Total Ionizing Dosage Effects

To ensure that avionics operate properly over the entire mission life, the electronics must be designed to handle the TID expected to accumulated over the mission (see Figure 3-2 for a general overview of orbits and their corresponding dose rates and Table 3.1 for dosage rates measured on specific satellites). The capability of an avionics system to withstand the expected TID level is determined by part selection.
(the hardware elements that compose the device, their arrangement, and the die fabrication process) and shielding.

![Figure 3-2: TID Rates Compared with Orbital Altitude [28]](image)

Table 3.1: Measured TID Rates on Specific Satellites [28]

<table>
<thead>
<tr>
<th>Satellite</th>
<th>Altitude (km)</th>
<th>Inclination (°)</th>
<th>External TID (Mrad/year)</th>
<th>TID behind 1mm of Al (krad/year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISS</td>
<td>450</td>
<td>51</td>
<td>0.150</td>
<td>1</td>
</tr>
<tr>
<td>Iridium</td>
<td>780</td>
<td>86</td>
<td>1.7</td>
<td>9</td>
</tr>
<tr>
<td>Celestri</td>
<td>1,400</td>
<td>48</td>
<td>17</td>
<td>63</td>
</tr>
<tr>
<td>Teledesic</td>
<td>1,400</td>
<td>84.6</td>
<td>18</td>
<td>51</td>
</tr>
<tr>
<td>Skybridge</td>
<td>1,457</td>
<td>55</td>
<td>28</td>
<td>73</td>
</tr>
<tr>
<td>Ecco</td>
<td>2,000</td>
<td>0</td>
<td>81</td>
<td>670</td>
</tr>
<tr>
<td>GPS-II</td>
<td>22,600</td>
<td>55</td>
<td>9,100</td>
<td>1,900</td>
</tr>
<tr>
<td>GEO</td>
<td>35,790</td>
<td>0</td>
<td>1,100</td>
<td>1,100</td>
</tr>
</tbody>
</table>

**Radiation Hardened by Process**

One of the main effects from accumulated TID is trapped charge altering the gate voltage needed to change the transistor’s state. The insertion of a layer in the substrate will increase the probability of excess charge, generated by a charged particle, harmlessly recombining. Either an oxide layer can be inserted in the substrate (used in Silicon on Insulator devices), or an epitaxial layer can be grown under regions of
high doping density (the region under the source and drain in Figure 2-6). The presence of this layer eliminates the capacitive build up that occurs between the doped regions and the substrate, greatly minimizing the ability for excess charge to build up and cause the transistor state to switch. This process has an added benefit of increased speed (∼30%) and decreased power consumption (∼15%) [14].

Techniques such as these where the standard wafer fabrication process is modified in order to minimize the TID effects are referred to as Radiation Hardened By Process (RHBP). Some RHBP techniques for implanting an oxide layer are Silicon on Sapphire (SOS) or Silicon on Insulator (SOI). In SOS, the silicon layer is grown on a sapphire (Al₂O₃) crystal. This process was initially developed by RCA Labs for military/aerospace applications in order to reliably operate in radiation environments in the 1960s and is still used for satellite avionics today [14]. SOI was developed from SOS to reduce the development cost. The buried oxide layer (reference Figure 3-3) in SOI can be created using several techniques, the most common of which are described below. A semiconductor material can be grown on a monocrystalline insulator, identical to how SOS is fabricated. Alternatively, annealing (heating) an implanted Oxygen creates a buried oxide layer, in a process called Separation-by-IMplantation-of-OXygen (SIMOX). A final method grow an oxide layer on two separate semiconductor substrates. Then bury the oxide layer by bonding the two substrates together (oxide layer to oxide layer), the desired circuit is developed by etching away and doping one of the substrate layers.

![Figure 3-3: SOI CMOS Cross Section](image)

TID hardening is further accomplished by making thinner gate oxides (the middle blue section in Figure 2-6, typically made of SiO₂) and by reducing contaminants such
as Hydrogen (H$_2$) or Sodium (Na). This naturally happens as feature sizes are reduced (by shrinking the size of the gate, L in Figure 2-6) for increases in performance and decreases in power consumption. For example, 250 nm feature size transistors can tolerate $\sim$100 krad before permanent failure while 90 nm feature size transistors can withstand $>$400 krad [4].

**Shielding**

Shielding is used to mitigate TID effects by blocking charged particles from reaching the electronics. In general, Aluminum or Tantalum is placed around the sensitive electronics. The thickness is on the order of 10s to 100s of mils (1 mil is one thousandth of an inch). As shown in Figure 3-4, the largest impact of shielding occurs in the first 200 mils where the absorbed dose is reduced by three orders of magnitude. Additional shielding is less effective, as shown by the vertical green lines in Figure 3-4 which designate the thickness corresponding to each order of magnitude reduction in TID.

![Figure 3-4: Impacts of Aluminum Shielding on TID [48]](image)

Due to the extra mass cost of shielding an entire avionics box, for moderate reductions in TID sensitive components can have shielding placed directly around the sensitive region, as shown in Figure 3-5.
3.1.2 Overcoming Single Event Effects

SEE-induced errors can be catastrophic to mission performance and functionality, potentially causing a loss of instrumentation controllability or a loss of spacecraft functionality. An overview of the traditional methods for achieving SEE tolerance are discussed below. Where ever possible, the underlying logic of these methods will be applied in the design of the COTS-based, fault-tolerant avionics architecture created designed in this thesis.

Radiation Hardened by Design

In order to reduce the cost and development time associated with RHBP, the technique of radiation hardened by design (RHBD) was developed. It uses standard silicon fabrication processes in conjunction with specialized layout techniques to apply reliability at the circuit/transistor level. The use of standard silicon fabrication processes, enable RHBD to reduce the production cost while increasing the performance capabilities [46]. Some common RHBD techniques are listed below:

- Increased Device-feature Size

  Feature size is known to be linearly proportional to $Q_{crit}$, thus the most basic RHBD technique is to increase the feature size. The effects of increased feature size on a two input NAND gate is shown in Figure 3-6. The primary difference between the three functionally equivalent transistors in Figure 3-6 is the area
utilized, the RHBD cell (left) requires 44% less area than the standard cell from the previous generation (right), and the RHBD cell requires 25% more area than a standard transistor cell from the same feature size (center). This increase in feature size results in the circuit drawing more power and operating at a lower peak frequency [4].

Figure 3-6: Comparison of RHBD Cell with Standard CMOS Cell [4]

• **Redundancy**

Another major RHBD technique is to add redundant transistors or circuits to each logic element. These modifications require a minimum of two transistors in the logical element independently change states to create a change in the output. Hence if a charged particle hits one transistor and changes its state, the circuit’s output will not be effected. This redundancy comes at the cost of both increased power and area, as shown in Figure 3-7, where the number of transistors increases by 50% for the RHBD circuit but the reliability also increases as two inputs must simultaneously change for a modification of the output. The primary types of circuit level redundancy are discussed below:
Temporal Latch

A temporal latch creates fault-tolerance by finding consensus amongst voting on time delayed inputs, as shown in Figure 3-8. Delaying inputs allows for a transient upset to alter the input signal but not generate an invalid output because the voter will reject anomalous inputs. However, this circuit becomes ineffective if the upset duration is longer than the time delay between replicated inputs. Thus a trade exists between reliability and performance, a longer delay increases reliability (by detecting longer duration upsets) while simultaneously decreasing performance (all operations incur a time delay).

Dual Interlocked Cell (DICE)

A dual interlocked cell is a replicated transistor circuit, storing a copy of the data’s binary inverse. Each transistor has a redundant partner to provide dual node feedback control which ensures that a single upset will not change the circuit’s output but will be corrected by the redundant
partner [12]. The transistor redundancy produces reliability; however, it increase the required circuit area and power by at least 100%. As transistor sizes shrink, the physical separation of redundant transistors must ensure that a single particle cannot upset both nodes and overcome the DICE’s protection [4].

Figure 3-9: DICE Latch Diagram [4]

- **Triple Modular Redundancy (TMR)**

  Triple modular redundancy is the classic method for implementing circuit-level redundancy in space systems. As shown in Figure 3-10, three identical circuits send outputs to a voter for determining consensus. The circuits must operate in lockstep for the voter to atomically compare their outputs. If one input line (i.e. circuit) suffers an upset, it is unlikely for the others to be effected because of the physical separation ensuring the majority of inputs to the voter are correct. TMR has the common disadvantage of increased power consumption and transistor area.

Figure 3-10: TMR Functional Block Diagram [4]
• **Guard Rings and Guard Drains**

To ensure that charged particles only influence a single transistor, an oppositely doped region encloses individual transistors, reference Figure 3-11. These regions are referred to as guard rings (center) or guard drains (right). In addition to providing charge separation between transistors, guard rings and guard drains regulate the transistor’s voltage potential, which reduces the amount of charge a transistor can collect by $\sim 20\%$, making it harder to accidentally change the transistor’s state [46].

![Guard Ring and Guard Drain Diagram](image)

**Figure 3-11: Guard Ring and Guard Drain Diagram [46]**

**Device Level Hardening**

The final design approach is to apply redundancy at the device level. TMR can be implemented at the device level where each critical device is replicated. Boeing has created such a design using COTS PowerPC 750s for high-performance spacecraft payload operation [10]. The three processors operate in lockstep, sending outputs to a voting circuit for comparison. Discrepancies between devices indicate an upset; but there is limited information about the cause/location of the error. In order for the system to be brought back to full operational state after detecting an error, all processors must save their current data and be power cycled to bring all the processors to a known state. Additionally, periodic power cycling ensures that undetected errors do not accumulate and produce uncorrectable errors.
Another design is based upon TMR, called Time-TMR (TTMR), it changes the timing characteristics for TMR to improve performance. A TTMR system has three processors and a voting circuit; however, standard operations only send two processors’s outputs to the voting circuit. The third processor is only used for mitigating errors. This allows for two processors to continue operating while the processor with an error is reset. The processors operate in a round-robin fashion to increase the system performance [25].

SECDED Hamming Codes

In memory devices, additional bits can be designated for detecting and correcting errors. These encoding schemes are broadly referred to as error correcting codes (ECC) and can be implemented in either hardware or software. In hardware, encoding bits are physical memory locations used exclusively for EDAC; whereas software implementations use other data locations for storing the encoding bits. The most basic version of ECC is parity, where an extra bit is added to each row (or column) designating an even or odd number of logical ‘1’s. A parity bit can detect a single bit error in any row (or column); however, it cannot correct errors because there is an ambiguity about which bit is incorrect. Adding additional encoding bits allows for detection and correction of multiple bit errors. The number of errors that can be detected along with the number of errors that can be corrected is based upon the Hamming distance between code words, defined as the number of different bits between each code word. A Hamming distance of three is required for each bit error the code can correct, and a distance of two for each bit error the code can detect [9].

In satellites, the most commonly implemented algorithm is a single error correction and double error detection (SECDED) code, having a Hamming distance of four bits. This is often referred to as a (8, 4) Hamming code as 8 bits are required to encode 4 bits of data [45]. Thus an overhead of 100% extra data storage is required. Each code word in SECDED encoding is created by concatenating the results from several parity checking equations (Equations 3.1 - 3.4) with the original data words ($D_i$). Equations 3.1 - 3.4 are the standard SECDED encoding equations for 4-bit data.
\[ C_1 = D_1 \oplus D_2 \oplus D_4 \quad (3.1) \]
\[ C_2 = D_1 \oplus D_3 \oplus D_4 \quad (3.2) \]
\[ C_3 = D_2 \oplus D_3 \oplus D_4 \quad (3.3) \]
\[ C_4 = D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus C_1 \oplus C_2 \oplus C_3 \quad (3.4) \]

The data word \((D_1D_2D_3D_4) 0101\) would create the following parity check bits \((C_1C_2C_3C_4) 1100\), for a code word \((C_1C_2C_3C_4D_1D_2D_3D_4) 11000101\), which would be stored in memory. In order to verify that the stored data is correct, the code word is multiplied by a parity check matrix. The parity check matrix \((P)\) has the following format \([C_1C_2D_1C_3D_2D_3D_4C_4]\), where the \(C_i\) entries form the identity matrix and the \(D_i\) are the corresponding values (from applying Equations 3.1 - 3.4). Thus the parity check matrix for this example is:

\[
P = \begin{bmatrix}
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{bmatrix}
\]

The result of multiplying the parity matrix \((P)\) by the transpose of the stored data or code word \((E)\) is called the syndrome \((S)\), as shown in Equation 3.5. A syndrome equal to the null set occurs when there are no errors in the data. If the result is non-zero, the first entry identifies if there is a parity error (and thus an odd number of errors exist), while the remaining three bits identify the location of any errors (with the binary value equaling the location of the bit error, starting from the leftmost value \(C_1)\) [45].

\[ S = PE^T \quad (3.5) \]
3.2 Software-based Approaches for Overcoming Single Event Effects

The previous examples of SEE mitigation are the one most commonly found in current spacecraft designs. They are hardware based, relying on specialized/redundant hardware for delivering error robustness. While hardware redundancy is critical for mission assurance, there is a possibility that hardware will fail, making complete reliance on hardware a weakness. Thus there have been investigations into software-based methods of creating a SEE tolerance.

3.2.1 Single Processor

The majority of SEE mitigation methods are designed to be implemented on a single processor. The most common methods in the literature are discussed below.

Redundant Execution

Redundancy can be accomplished on the same hardware by placing temporal spacing between executions. Two primary methods have been studied: redundant execution [47] and redundant threads [45]. In redundant execution, the entire operation is completed, the results are stored. The operation is then executed again (after a time delay from completing the first iteration) and the results are compared with those from the previous iteration. Completing three executions of the same operation and comparing their results allows forward progress to be made if an error is detected.

In redundant threads, multiple instantiations of the same thread are executed concurrently and the intermediate results of operations within the threads are periodically compared, as illustrated in Figure 3-12. Thus instead of checking only the final results for errors, a more fine-grained checking is implemented, allowing for errors to be detected earlier, potentially providing more detailed information about the source of the fault.
**Redundant Virtual Machines**

Redundant virtual machines allow for different implementations of redundant execution where the physical processor components can be replicated via virtualization [45]. Virtualization is the process of simulating multiple computer systems on a single hardware unit, this is generally done through time division of the processor’s computing time. Through virtualization, the number of redundant physical processors can be reduced because extra redundant systems can be simulated [43]. Additionally, virtualization simplifies migration between physical processors as it creates a common base above the hardware-level.

**Reduced Precision Redundancy**

Reduced precision redundancy (RPR) is a version of redundant execution where instead of executing identical copies of the operation, redundant executions are computed on an upper and lower estimate, allowing for fewer bits to be compared [52]. For example, if the exact value is a 16-bit, fixed point value (e.g. 0x3579 = 13,689), while the upper and lower estimates assume that the 8 least significant bits (LSBs) are all zeros (e.g. upper estimate = 0x3600 = 13,824, lower estimate = 0x3500 = 13,568). These upper and lower estimates provide bounds for acceptable amounts of error in the final result and allow for minor SEE-induced errors to be ignored because of their negligible impact. RPR’s truth table for determining which value to use after a comparison is shown in Figure 3-13.

![Figure 3-12: Redundant Execution](image-url)
AN Codes are arithmetic error detection algorithms that redundantly implement an execution, but scale the redundant copies inputs by a predetermined factor (N). Thus the redundant output should be a factor of N different from the initial result. For example, a subtract operation could have the inputs 5 and 2, which returns 3. Applying a scaling factor (or N) of 2, the inputs would be 10 and 4 resulting in 6, which can be scaled by the inverse of the scaling factor (N = \frac{1}{2}) and compared with the original result to determine its correctness. AN codes are limited to checking addition, subtraction, multiplication, division, and roundoff operations [5]. A unique strength of AN algorithms is their ability to determine, using only a single processor, if mathematical units are functioning correctly.
Instruction Stream Checking

Instruction stream checking (ISC) does not implement redundancy, instead it creates a computational flow diagram (CFD) at compile time and monitors the program flow to ensure it only follows the predefined flow paths [47]. The process of taking program flow and converting it into a CFD for error checking is shown in Figure 3-14. On the left is the executable code as programmed, the operations are then rearranged into groups based upon their execution ordering in the center, which form the CFD on the right. Any jump in program flow not predefined (e.g. a jump from instruction 2 to instruction 6, which goes from Node 1 to Node 3) is assumed to be the effect of a SEE. This ensures that any unwanted branches do not occur and that the program follows the correct order of execution.

<table>
<thead>
<tr>
<th>Instruction Sequence</th>
<th>Rearranging into Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. sto a, b</td>
<td>1. sto a, b</td>
</tr>
<tr>
<td>2. add b, b, c</td>
<td>2. add b, b, c</td>
</tr>
<tr>
<td>3. sub b, b, d</td>
<td>3. sub b, b, d</td>
</tr>
<tr>
<td>4. br fifth</td>
<td>4. br fifth</td>
</tr>
<tr>
<td>5. cmp a, b</td>
<td>5. cmp a, b</td>
</tr>
<tr>
<td>6. beq first</td>
<td>6. beq first</td>
</tr>
<tr>
<td>7. mul d, c, d</td>
<td>7. mul d, c, d</td>
</tr>
<tr>
<td>8. cmp d, a</td>
<td>8. cmp d, a</td>
</tr>
<tr>
<td>9. bne second</td>
<td>9. bne second</td>
</tr>
<tr>
<td>10. xor f, e</td>
<td>10. xor f, e</td>
</tr>
<tr>
<td>11. cmp f, b</td>
<td>11. cmp f, b</td>
</tr>
<tr>
<td>12. br third</td>
<td>12. br third</td>
</tr>
<tr>
<td>13. and b, b, a</td>
<td></td>
</tr>
<tr>
<td>14. br third</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-14: Instruction Stream Checking Process [45]

Memory Scrubbing

Memory scrubbing is an active approach to ensuring that encoded data remains correct. To minimize the chance of errors accumulating and overcoming the encoding’s correcting ability, the data is periodically read. Upon reading any errors present in
the memory are removed by the ECC. This periodic execution allows for weaker codes to create the same reliability as more complex codes.

### 3.2.2 Multiple Processor

Another group of algorithms for SEE tolerance require multiple processors. A detailed discussion of these algorithms is given below:

**Distributed Fault-tolerance**

Distributed systems must ensure that operations are consistent across many hardware units to stop errors from being propagated [13]. The standard process for achieving distributed consensus is through an algorithm called Paxos [13], the data flow of Paxos is shown in Figure 3-15. The device that originates an operation is called the primary. The primary uses other devices in the system as backup checkers, which will execute the same operation. The results from the backup checkers are then sent to the primary in order to achieve consensus. Consensus selects the average of the majority of the results which are within a predefined range to be the correct output. This final output is shared among all units, allowing any device to act as the primary in the future.

![Figure 3-15: Distributed Fault Tolerance](image)
The flow of operations seen in Figure 3-15 can be modified based upon the system’s operations and the reliability needs. For example, if the master processor cannot be changed there is no need to pass the results to all the processors in the system. Within this distributed system scheme of operations, a couple different operational modes are possible depending upon how peripherals are connected. The primary configurations investigated in this thesis are discussed below:

- **Devices with a Single Connection to the Processor Network**

  Many devices communicating over a serial interface can only support a single master. The addition of arbitration logic/circuits can eliminate this type of setup from being a single point of failure, but they add significant complexity to the system. The fault-tolerance of devices connected to a single processor can be increased by transmitting the data generated by the device to other processors in the system. These processors will interpret the data and process it as if they were the master of the device, and retransmit their results to the originating processor for error checking. Upon reaching a consensus, the master will store the results and use them in any future computations or outputs to the device.

- **Devices Concurrently Connected to Multiple Processors on the Network**

  Some devices are designed to be connected to multiple processors. Generally this occurs when the device is setup to communicate over a bus. This setup allows for many fault-tolerance strategies. A single processor can act as the master of the device and all communication can be completed through that predefined processor, and in the case of that processor failing another processor on the bus assumes the role. A fault-tolerance procedure similar to that described for a device with a single point of connection can be utilized.

  Alternatively, all the processors can concurrently read the data from the device off the bus and begin processing the result. Each processor transmits its results to the other processors for error checking. Once a consensus is reached, one
processor communicates with the devices. This reduces latency by eliminating the need to send the initial data from the master to all the backups.

### 3.3 Error Mitigation Testing

Three methods exist for verifying the fault-tolerance of an avionics system: hardware-based fault-injection, software fault-injection, and simulation-based fault-injection. In hardware-based fault-injection the device is bombarded with high energy radiation while test functions are evaluated for proper execution [64]. This is the optimal testing method because it closely mimics the space environment, making it the most trusted testing methodology. Some limitations of hardware-based testing are the cost ($\sim \$750/\text{hour}$, with an average test duration of 20 hours [4]) and the permanent damage done to the device.

Software fault-injection tests a function for proper execution while a concurrently running operation is manipulating the computer’s state (e.g. control registers, data values, etc.) [64]. This allows tests to target specific hardware locations (e.g. error flag register), for a finer determination of both the strengths and weaknesses of the EDAC schemes.

Simulation-based fault-injection is opposite of hardware-based testing, using a computer model (e.g. processor emulator or VHDL-based processor models) of the device for testing the responses to different injected errors. This type of testing allows for targeting specific areas/functionalities of a device to determine their response to errors; however, the fidelity of the model greatly determines the accuracy of the results (as well as the time required for testing) [64].

### 3.4 Conclusion

Many techniques have been implemented for mitigating the challenges posed to reliably operate in the space environment. Mitigating TID errors typically requires hardware modifications; however, modern feature sizes could allow COTS hardware
to be resilient to many TID concerns [4]. Yet this comes at the expense of increased susceptibility to SEE-induced errors, as shown in Figure 3-16. SEE-induced errors can be combated at every design level (i.e. transistor, circuit, device), with most techniques employing a form of redundancy to detect and correct the error.

![Figure 3-16: Relation between SEE Induced Errors and Process Size [60]](image)

Redundancy allows for the detection and correction of the majority of SEE-induced errors, it will thus be a primary aspect of the COTS-based avionics architecture designed in this thesis. As hardware mitigation is cost prohibitive with low performance and software mitigation is not robust enough to guarantee MA, a synergistic combination of both will be implemented to generate the required reliability. This hybrid approach with combine software redundancy across a set of redundant hardware. Some areas of this hybrid approach that this thesis will investigate are described below:

- **Performance Driven**

  A performance driven system primarily runs a single processor EDAC algorithms. However, once an error is detected, that processor brings in additional processors to determine the correct output. This ensures that optimum performance is granted to running processes, as there is no requirement to wait on other processors or use low performance hardware. However, once an error is
detected the full hardware redundancy is brought on-line to mitigate the error. This approach is similar to SpaceMicro’s TTMR [25].

- **Reliability Driven**

  For optimal reliability, the full suite of software and hardware reliability mechanisms must be employed, often at the expense of performance. Optimal reliability utilizes SECDED encoding of data values, with high frequency memory scrubbing. Operations are executed redundantly on each processor prior to exchanging the results between processors. This allows individual processors to catch errors before broadcasting them to the network of processors. The multiple processors allow for the system to detect faulty processors enabling errors to be detected earliest at the lowest level possible. Finally, outputs are determined from the multiple processors’s consensus. This ensures that a single processor acting improperly does not inadvertently send an incorrect command.

- **Lifetime Driven**

  In order to maximize the lifetime of a system, it should have extra processors which are not used for regular operations. These cold backups should be integrated into the processor network to allow quick entry into the processor network, enabling processors that fail from accumulated radiation dosage to be rapidly swapped out by new, radiation free processors.

  This thesis will now build upon the various radiation tolerance techniques from the literature to create an optimal COTS-based avionics architecture. It will emphasize utilizing a hybrid of software and hardware redundancy for the unique benefits it generates by allowing the hardware and software to compliment each other.
Chapter 4

Microcontrollers as Primary Avionics

Spacecraft are continually being pushed to perform more complex operations autonomously [42], raising their computing requirements. When designing avionics systems for a spacecraft one wants to “optimize the availability, capability, flexibility, and reliability of the system while minimizing cost and risk” [62]. Traditional design methods and components used in spacecraft avionics systems struggle to fully optimize these design factors, because of limitations in available products.

In an effort to broaden the options available for use as spacecraft avionics, this thesis explores the usage of COTS components for increasing the functional capabilities and part selection options while maintaining reliability. In particular, this thesis focuses on the usage of microcontrollers as the primarily flight processors. Microcontrollers were chosen because of their low price and high power efficiency, making them ideal candidates for spacecraft avionics systems, especially those implementing hardware redundancy.

4.1 Spacecraft Avionics Problem Statement

Traditionally, spacecraft avionics have ensured reliable reliability through the use of specialized hardware. These RadHard components lag the performance capabilities
of their COTS equivalents (reference Figure 4-1). Recently, they hit a performance plateau, topping out with performance capabilities similar to that of a 1996 desktop computer (a late generation Pentium I). This lack of performance limits satellite operations requiring computationally intensive three-dimensional matrix processing such as computer vision and RF data cross correlation.

![Graph showing processor performance compared with time of delivery to market.](image)

Figure 4-1: Processor Performance Compared with Time of Delivery to Market

When integrated electronics were first introduced in the 1960s, their demand was largely driven by the US space program, with the Apollo program utilizing over 30% of the world’s supply supply of microelectronics [15]. This large demand allowed spacecraft designers to dictate key capabilities, such as high reliability. However, the demand for high reliability components has been replaced by a demand for high performance devices as the consumer base has shifted to personal electronics. This market shift is seen by comparing the number of processors used in cell phones, over 10 million COTS processors per year, with those used in satellites, less than 100 RAD750 processors over the last ten years [53]. This change in demand has caused the price of spacecraft electronics to balloon while the capabilities have stagnated. The simplest method for bringing higher performance and lower costs processors into the spacecraft avionics design repertoire is through the usage of COTS processors.

---

1Data compiled from [53], [54], and [20]
While this lack of high performance in RadHard processors limits future missions which push the bounds of satellite mission capabilities, the majority of spacecraft missions do not require these high performance capabilities. The FireSat example used throughout [62], requires a processor be able to execute 0.35 million instructions per second (MIPS), a trivial amount for modern processors. FireSat’s payload, an infrared (IR) sensor, creates a larger computational requirement of 37.96 MIPS, which despite being two orders of magnitude greater than the satellite’s operating requirements is well below the capabilities of most modern processors. For this reason, many RadHard processors operate at reduced clock speeds to minimize the power consumption. Table 4.1 lists the standard satellite operations and their estimated computational throughput requirements, the summation of these requirements yields a total requirement of ~1.4 MIPS.

In order to utilize COTS processors’s capabilities and thereby broadening the available spacecraft avionics options, the reliability of a COTS-based systems must be increased. Software-based EDAC algorithms, demonstrated to be 97% effective [47], could be coupled with redundant hardware, trusted by industry [10], to produce the needed reliability.

4.2 Comparison of COTS and RadHard Processors

While COTS processors can be designed for low price, high performance, or small size–RadHard processors are solely designed to handle the errors generated by the space environment. As noted in Table 4.2, COTS processors are viewed as too risky for space applications because of their lower tolerance of radiation effects.

However, today’s small, unmanned satellites are being designed to perform more complex missions such as: precision pointing, fractionated operations, and computer vision based navigation. The continuation of advancements in these missions require high-performance processors that can operate reliably in the space environment.
Table 4.1: Computational Throughput Requirements for Spacecraft [62]

<table>
<thead>
<tr>
<th>Function</th>
<th>Throughput (KIPS)</th>
<th>Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Communications</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command Processing</td>
<td>7.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Telemetry Processing</td>
<td>3.0</td>
<td>10.0</td>
</tr>
<tr>
<td><strong>Attitude Sensor Processing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rate Gyro</td>
<td>9.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Sun Sensor</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Earth Sensor</td>
<td>12.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Magnetometer</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Star Tracker</td>
<td>2.0</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>Attitude Determination &amp; Control</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kinematic Integration</td>
<td>15.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Error Determination</td>
<td>12.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Precession Control</td>
<td>30.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Magnetic Control</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Thruster Control</td>
<td>1.2</td>
<td>2.0</td>
</tr>
<tr>
<td>Reaction Wheel Control</td>
<td>5.0</td>
<td>2.0</td>
</tr>
<tr>
<td>CMG Control</td>
<td>15.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Ephemeris Propagation</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Complex Ephemeris</td>
<td>4.0</td>
<td>0.5</td>
</tr>
<tr>
<td>Orbit Propagation</td>
<td>20.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Autonomy</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple Autonomy</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Complex Autonomy</td>
<td>20.0</td>
<td>10.0</td>
</tr>
<tr>
<td><strong>Fault Detection</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitors</td>
<td>15.0</td>
<td>5.0</td>
</tr>
<tr>
<td>Fault Correction</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td><strong>Other Functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Management</td>
<td>5.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Thermal Control</td>
<td>3.0</td>
<td>0.1</td>
</tr>
<tr>
<td>Kalman Filter</td>
<td>80.0</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1,378.52</td>
<td>1.0</td>
</tr>
</tbody>
</table>

COTS processors currently deliver this level of performance; however, they lack the guaranteed reliability necessary for spacecraft operations. An overall comparison of capability/performance differences between various embedded COTS processors (listed in Table 4.3) and RadHard processors is shown in Figure 4-2 (note that the RadHard processor costs are scaled by 1000 to allow for the COTS data to be distinguishable on the same chart). The COTS processors increase in design complexity.
and capabilities as one progresses from left to right in Figure 4-2 or down Table 4.3, ranging from simple 16-bit microcontrollers to highly capable 32-bit laptop processors.

![Figure 4-2: COTS and RadHard Processor Comparison](image)

From Figure 4-2, one quickly notes RadHard processors perform slower (on average having \( \frac{1}{5} \) th the MIPS), consume more power (on average 30% more power), require more board area (on average more than double), and cost more (on average an a three

---

2 Data compiled from [53], [54], [36], [18], [34], [27], [33], [17], [19], [30], and [2]
orders of magnitude difference) than current embedded processors. If a COTS-based system can be designed which operates reliably in the space environment, there are several system-level benefits. In addition to improvements in cost, size, and performance, designers gain product flexibility from the large influx of processor options into the design space. Finally, the adoption of COTS devices shortens the learning curve for entry into the aerospace avionics field.

Within the dataset of embedded processors displayed in Figure 4-2, there are three microcontrollers (the first three blocks from the left in the COTS graphs of Figure 4-2 and the top three entries in Table 4.3). These devices are optimized to operate at low power levels, consume minimal board area, and sell at low costs. The trade-off is reduced computational capabilities, approximately an order of magnitude below current state-of-the-art embedded processors. While a microcontroller’s performance is on average 20% of a RadHard processor’s performance rating, multiple microcontrollers could be incorporated into a system design to make up for the computational disparity without exceeding the equivalent system costs associated with a RadHard system with equivalent performance.

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Manufacturer</th>
<th>Maximum Clock Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP430 [36]</td>
<td>Texas Instruments</td>
<td>25 MHz</td>
</tr>
<tr>
<td>AVR32 UC [18]</td>
<td>Atmel</td>
<td>200 MHz</td>
</tr>
<tr>
<td>dsPIC33 [34]</td>
<td>Microchip</td>
<td>40 MHz</td>
</tr>
<tr>
<td>Blackfin [27]</td>
<td>Analog Devices</td>
<td>600 MHz</td>
</tr>
<tr>
<td>i.MX51 [33]</td>
<td>Freescale</td>
<td>800 MHz</td>
</tr>
<tr>
<td>PowerPC755 [17]</td>
<td>IBM</td>
<td>600 MHz</td>
</tr>
<tr>
<td>Atom N270 [19]</td>
<td>Intel</td>
<td>1.6 GHz</td>
</tr>
</tbody>
</table>

4.3 Rational for COTS Microcontrollers in Space

In a space system, minimizing the required power and mass is essential. Consequently, low-mass and low-power devices are desired. Spacecraft avionics components are typically low-mass, but reducing a device’s footprint size can impact the avionics
system mass as the size of the footprint directly relates to the size of the avionics box. Furthermore, as processor performance increases, the required power increases exponentially (as shown in Figures 4-2(a) and 4-2(b)). With these competing needs for high-performance and low-mass/low-power the ideal avionics system utilizes compact devices with high performance to power consumption ratios. Microcontrollers fill this niche, having a small footprint and boasting some of the highest performance capabilities per unit of power consumption (in addition to performance per unit cost, both shown in Figure 4-3).

![Figure 4-3: Further COTS and RadHard Processor Comparison](image)

Microcontrollers add a wealth of design options, from device modularity to a diversity of standard parts. Microcontrollers enhance the capability to rapidly prototype satellite avionics systems because of their well defined modularity and plethora of built-in interfaces. The modularity allows for the same microcontrollers to be utilized in diverse applications, and the abundance of standard communication interfaces increases the adaptability of the avionics system to different mission requirements. The wide array of standard microcontrollers enables mission requirements to be matched to hardware components. Finally, these design traits are ideal for distributed systems where each subsystem or component can have the processing power it requires on-board. In contrast to relying on a high-performance centralized avionics system. Thus while individual microcontrollers may have lower performance capabilities than

---

3Data compiled from [53], [54], [36], [18], [34], [27], [33], [17], [19], [30], and [2]
RadHard processors, a distributed system of microcontrollers can provide significant performance increases while decreasing the power consumption and cost of the system.

### 4.3.1 Microcontroller Background Information

A microcontroller (MCU) is “a microcomputer that contains most of its peripherals and required memory inside a single integrated circuit along with the CPU” [6]. MCUs follow the SoC design strategy by placing all the needed computing functionality (i.e. central processing unit (CPU), both program and temporary memory, and interface hardware) on a single integrated circuit (IC). The SoC design is beneficial because it reduces the avionics system’s design time, component count, and complexity.

**Performance**

The performance difference between MUCs and embedded processors is shown in Figure 4-2(a). MCUs have lower performance capabilities because they are fabricated using older process sizes (i.e. larger transistors) and forgo many of the high-end performance circuits found on state of the art processors such as cache memory, branch predictors, and out of order execution. Additionally, MCUs operate at significantly reduced clock frequencies, generally <100 MHz whereas state of the art processors operate at >1 GHz.

Another performance limiter on MCUs stems from their reduced and simplified design. Many MCUs operate on a smaller data bus than standard processors, MCUs frequently use an 8-bit or 16-bit data bus, in stark contrast to state of the art processors which utilize a 64-bit data bus. This affects the performance of an MCU when data values are larger than the bus size. For example, a 16-bit MCU must use a minimum of two operations to perform the addition of two 32-bit numbers (i.e. float or int data types) whereas a 32-bit processor could accomplish this in one operation. MCUs also lose performance from not having specialized hardware circuits for floating point math operations found in modern processors. MCUs must instead perform the operations through a much slower series of software operations.
MCUs simplify processor operations with their basic functional architecture. An MCU operates under a reduced instruction set computer (RISC) architecture which restricts the processor instructions to a short list of simple commands, generally executable in a single processor cycle. Complex operations are performed by combining multiple commands. This contrasts modern processors which run on a complex instruction set computer (CISC) architecture, where a large array of instructions are available that can perform complex operations (e.g. Fourier transform) with a single instruction, but the number of processing cycles required is dependent on the instruction’s complexity [32]. Initially, it appears that a RISC architecture guarantees increased performance; however, this is not necessarily true as certain CISC instructions requiring multiple clock cycles equivalent RISC implementation require multiple instructions (i.e. the x86, a CISC, instruction LOOPZ count, location; decrements the count register and performs a relative jump if the zero flag is not set, while the RISC equivalent would require the following instruction sequence DEC count; BRZ location;) [23]. This RISC architecture makes debugging MCUs simpler as there are fewer processor instruction which must be understood. Further, it simplifies estimating the execution time for a given process.

While MCUs do not have the computational throughput of current RadHard processors, the difference is constantly decreasing as the market pushes MCU improvement. Further, multiple MCUs could provide the same capabilities as a single RadHard processor at lower system costs (i.e. size, cost, and power consumption).

**Power Usage**

As shown in Figure 4-2(b) MCUs consume minimal power, on average ~80 milliwatts (mW). Specialized operating modes allow for additional reductions in power consumption by scaling back the clock frequency and not powering unused peripherals, dropping power consumption to 10s of nanowatts (nW). This contrasts standard processors which require >1 Watt (W). Despite low performance capabilities, MCUs outperform all competitors in their computational throughput per unit of power consumed (as seen in Figure 4-3(a), by the MSP430 in particular).
This low power consumption is particularly impressive when also considering the SoC design of an MCU. An MCU’s power consumption not only accounts for the CPU, as in a standard processors, but also the power consumed by the critical support hardware, such as memory and interface hardware which are on separate ICs and not included in a standard processor’s power consumption. This reduction in power consumption and board area required to support the processor becomes more noticeable as the processors become more complex, as shown in Figure 4-4 of a single board computer’s (SBC) required external components. Each of these components consumes a nontrivial amount of power, generally in the mW to W range, making a final avionics system such as the SBC version of the RAD750 consume $\sim$10 W (an increase of 5 W over the processor alone [53]). This drastic reduction in power consumption makes MCUs an advantageous growth path for satellite avionics.

Figure 4-4: Components Required to Support a Processor [16]
Size/Area

MCU families present the avionics system designer with various levels of computational capability that can be matched to differing memory sizes and I/O availability. Within this matrix of design choices, the IC’s footprint is directly related to the number of I/O provided. The smallest MCUs can be found in 6 pin (4 GPIO) SOT-223 packages (consuming 5.94 mm$^2$, 9.21 mils$^2$ and shown on the left in Figure 4-5), and the largest ones in 100 pin (86 GPIO) TQFP packages (consuming 256 mm$^2$, 396.8 mils$^2$ and shown in the center of Figure 4-5). Even the largest MCU consumes less than half the area of a standard processor, as shown in Figure 4-5. The SoC design’s need for minimal extra circuitry creates an effect similar to that noted for power consumption. MCUs only require bypass capacitors while a standard processor requires a large array of extra ICs, as shown in Figure 4-4 for a SBC.

![Figure 4-5: Physical Size Comparison of Processors and MCUs](image)

This board area savings from MCUs may not appear to create a major satellite design advantage as the area difference between the largest MCU and the smallest processor is a reduction of only 0.4 in$^2$ (2.56 cm$^2$). However, when the external support ICs are also considered the benefit becomes noticeable. Particularly because a smaller board area reduces the size requirements on the avionics box which reduces the system mass. Assuming an aluminum box with a minimum thickness of 62.5 mils ($\frac{1}{16}$ in or 1.5875 mm), for every square inch of board area eliminated equates to a mass savings of $\sim$0.02 pounds (every square millimeter equates to a mass reduction of $\sim$0.016 grams). The mass savings can be calculated using Equation 4.1.
\[ m_{\text{eliminated}} = (2L_{\text{reduced}}W_{\text{reduced}} + 2Hk) t\rho \]  

(4.1)

\begin{align*}
L_{\text{reduced}} &= \text{reduction in board length} \\
W_{\text{reduced}} &= \text{reduction in board width} \\
H &= \text{height of box} \\
t &= \text{box thickness}
\end{align*}

\[ \rho = 0.101 \text{ lb/in}^3 (2.8 \text{ mg/mm}^3) \]  

Note: \( H \) is generally 0.8 in (20.32 mm), \( t \) is generally 0.1 in (2.54 mm), and \( k \) is the larger of \( L_{\text{reduced}} \) and \( W_{\text{reduced}} \)

**Unit Cost**

The low cost of MCUs is a product of their large market demand (>4 billion/year [22]) and design simplicity. Their small die size allows for thousands of MCUs to be fabricated on a single silicon wafer (approximately 2,530 to 11,500 on a standard 30 cm wafer) which reduces the production costs and increases the robustness to defects in the die [32]. These factors allow MCUs to be sold at low costs (from $0.30/MCU for the simplest MCUs to $12.60/MCU for the most complex), which are at most 33% of the cheapest embedded processor’s cost\(^4\). The three orders of magnitude difference between the costs of RadHard and COTS processors poses a major paradigm change for spacecraft costs, by allowing spacecraft component costs to not be driven by the cost of RadHard avionics.

**Versatility**

As hinted at previously, MCUs bring a depth of variety and versatility to the avionics design space. The same package size and pin layout can be used for applications as diverse as analog to digital conversion (ADC) of input signals to pulse width modulated (PWM) output signals. This is possible because an MCU’s pins are multiplexed to perform various software selectable functions. This versatility allow for the same pin to simultaneously function as an input for one device and an output from another device. For example, the analog inputs are commonly multiplexed with GPIO

\( ^4\)Prices were quoted from Digikey
functionality, thus the same pin can be used for analog input and a digital output – if the analog device’s output can tolerate being driven. These types of optimizations for achieving maximum capability out of minimal hardware resources have been researched by Pumpkin Incorporated [38]. Furthermore, MCU families provide variations in the device’s capabilities (e.g., a large RAM array in one and specialized motor controller logic in another). This allows different models of the same device to fulfill diverse system requirements from the large I/O capabilities needed for interfacing with actuators to the extra RAM needed for performing math operations.

A final area of MCU versatility is their ease of programming. Within an MCU family minimal modifications are necessary to port drivers from one MCU model to a different model, generally only requiring modifications to the I/O setup. This aids in the development and allows for devices to be easily changed as the software can be transferred between devices.

4.3.2 Justification for Microcontrollers in Satellites

MCUs increase the components available for use in spacecraft avionics. The wide array of sizes, performance characteristics, and predefined interfaces allows for a processor to be selected that match the design requirements as opposed to forcing a RadHard processor to meet the requirements. Their small size and low power consumption make them ideal for hardware replication and allows the creation of distributed systems where a separate MCU is responsible for the processing needs of each subsystem/critical device. These design factors make MCUs prime candidates for use in spacecraft avionics systems.

Towards this end, many small satellites and CubeSats (a small satellite category with a volume constraint of 10 cm x 10 cm x 10 cm and a mass constraint of 1 kg) have employed MCUs as their primary avionics systems. A few examples of MCUs employed in satellite designs include Taylor University’s TEST satellite which incorporated an MCU driven interface board between each instrument and the primary avionics computer [59]. TU Delft’s Delfi-C3 CubeSat utilizes a distributed MCU based architecture for all its avionics [3]. Montana State’s MEROPA CubeSat used
an MCU as its primary avionics [41]. Finally, Pumpkin Incorporated markets a line of CubeSat avionics systems with MCUs as the primary avionics computers. MCUs are a growing trend in university’s small satellite programs because of their reduced budgets and focus on simplicity.

**Meets Satellite Needs**

The capabilities of MCUs have steadily increased, and as shown in Table 4.1, the basic operating requirements of a satellite are minimal (∼1.4 MIPS) when compared to modern processor’s capabilities (shown in Figure 4-2). This throughput requirement is below that of the least capable MCU (having 5 MIPS [36]), thus MCUs fulfill the processing requirements for spacecraft with a 100% while using less than half of the processor’s capabilities, two of the main assumptions used throughout [62] for designing an avionics systems.

One of the larger processing requirements for satellites is generated by the payload; a single payload can require >40 MIPS [62]. This higher performance requirement is achievable by many MCUs (such as Microchip’s dsPIC33 [34] and Atmel’s AVR32 [18]), and when requirements exceed the MCUs capabilities, multiple MCUs can be incorporated to achieve the needed performance (generally with a lower system cost than a standard microprocessor).

The performance capabilities and available interfaces make MCUs fully capable of handling standard satellite operations. Further, their high degree of on-chip functionality creates many system-level advantages to an avionics system built around MCUs, from reductions in size/mass to reductions in power consumption.

**Opportunities Created**

MCUs pose many design benefits to the field of satellite avionics design, from high MIPS/W to a wide diversity of available models. Furthermore, they are ideal for the creation of distributed and/or redundant systems. MCU’s small size (with an average footprint 10% the size of an average RadHard processor’s footprint) allow for designs to incorporate a large array (>6 for every RadHard processor) of MCUs without ex-
ceeding the board size of a RadHard system. This array could cycle through different MCUs over the course of the orbit/mission, with any unneeded MCUs in a powered off state. Placing the MCUs in this powered off state minimizes the radiation damage each acquires because the electric fields normally present within the microelectronics are absent and cannot capture charge. This operating scheme would extend the system’s operational lifetime by increasing the amount of TID it can accumulate, as well as enabling the system to gracefully degrade as components fail.

Alternatively, MCUs could support a highly distributed architecture where each subsystem/component has a dedicated MCU for its processing needs. These MCUs are connected in a network for sharing data and coordinating operations. This distribution of processing around the satellite allows for great spacial separation which aids in enabling graceful degradation. Furthermore, this distribution of MCUs to the processes and components which need them allows for each MCU to be selected based upon its specific functionality requirements. This distributed implementation is a design space that the spacecraft avionics community is attempting to develop for use with larger multicore processors such as those produced by Tilera [58]. However, implementing this distributed system with multiple hardware MCUs instead of a single, multicore processor posses reliability and interface advantages as it allows for device failure and increases the system’s I/O capabilities. Creating a situation where using extra components will simplify the system by facilitating error handling. Additionally, this type of architecture supports the creation of a universal bus needed for establishing a plug and play framework for spacecraft components [31].

Thus MCUs stand to create dynamic changes to current spacecraft avionics designs, by bringing many potential increases to system robustness and adaptability. They provide most of the required functionality on a single IC, which greatly simplifies many challenges associated with creating a multiprocessor system. Additionally it reduces the avionics system’s power consumption, size, and cost, thereby opening the design space up to many new and alternative designs.
4.4 Conclusion

While a single MCU will never have the same computing power as a dedicated microprocessor, MCUs have advanced to the point of providing capabilities on par with many RadHard processors. Further, the SoC structure of MCUs fills many spacecraft design goals, in particular they present a maximum solution for MIPS/W. MCUs pose advantages in unit size, cost, and versatility over other processor options, as shown in Table 4.4. Thus with the proper supporting architecture, to increase their radiation tolerance, MCUs could create tremendous gains for spacecraft avionics systems, as many university programs have demonstrated.

Table 4.4: Processors Evaluation

<table>
<thead>
<tr>
<th>Metric</th>
<th>MCU</th>
<th>Microprocessor</th>
<th>RadHard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability, TID (krad)</td>
<td>40 - 100</td>
<td>200 - 250</td>
<td>400 - 1,000</td>
</tr>
<tr>
<td>Reliability, SEE (errors bit−day)</td>
<td>10⁻⁶ - 10⁻⁷</td>
<td>10⁻³ - 10⁻⁴</td>
<td>10⁻⁸ - 10⁻¹⁰</td>
</tr>
<tr>
<td>Process Size (nm)</td>
<td>130 - 300</td>
<td>32 - 90</td>
<td>90 - 150</td>
</tr>
<tr>
<td>Performance (MIPS)</td>
<td>10 - 100</td>
<td>500 - 100,000</td>
<td>10 - 400</td>
</tr>
<tr>
<td>Cost ($)</td>
<td>1 - 10</td>
<td>25 - 500</td>
<td>10,000 - 200,000</td>
</tr>
<tr>
<td>Power Consumption (W)</td>
<td>0.0002 - 0.3</td>
<td>0.25 - 135</td>
<td>1 - 7.5</td>
</tr>
<tr>
<td>Size (in²/cm²)</td>
<td>0.056/0.36</td>
<td>0.22/1.44</td>
<td>0.97/6.25</td>
</tr>
<tr>
<td>Versatility</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

Data compiled from [53], [54], [36], [18], [34], [27], [33], [17], [19], [30], and [2]
Chapter 5

Low Cost, High Reliability
Avionics Architecture Design

The avionics system of a modern spacecraft is typically designed to consume: 1-3% of the spacecraft mass, <0.5% of the volume, and 6-11% of the available power [62]. Minimizing these resource utilizations is constrained by the available components, in particular RadHard devices. The limited number of RadHard components has brought about a singularity in satellite avionics system architectures. This thesis proposes a new, COTS-based avionics architecture which will broaden the satellite avionics design space while simultaneously reducing the system costs (e.g. price, power consumption). A primary focus of this architecture is to meet or exceed the computational performance of existing RadHard architectures while maintaining a sufficient level of radiation tolerance to meet the spacecraft’s operational requirements; thereby furthering the field of spacecraft avionics systems.

The avionics architecture will utilize multiple processors to create a distributed system, capable of balancing the workload. Further, the redundant hardware will increase an operation’s reliability by having the operation execute on multiple hardware units which will compare the results with each other prior to committing one. However, as not all operations require this level of reliability the system performance can be increased by allowing these less critical operations to be independently executed on separate processors and the results instantly committed. By viewing reliability at
the function level, it adds flexibility to the system which can be used to increase the performance. As Figure 5-1 demonstrates, the distributed system can commit three tasks in the time required by the system operating in lockstep to perform two tasks because of the distributed system’s added flexibility. Some operations, such as task C in Figure 5-1, can execute independently because they are at a lower reliability level. Furthermore, the distributed system removes the need for lock stepped timing, eliminating a potential source of errors.

As there are several classes of satellite and significant differences in the operational requirements for each, the design of the avionics architecture in this thesis focused on small satellites (<200 kg) operating in lower LEO (having an altitude <1,000 km), with a short mission lifetime (less than two years). These constraints ensure that the satellite will be outside of the high flux regions of the inner Van Allen radiation belt (the proton belt, see Chapter 2 for further details), which coupled with the limited mission duration minimizes the amount of TID the system must be capable of withstanding. Thereby making the major reliability requirement the mitigation of SEE-induced errors. Finally, the overarching design goals for the architecture are to minimize the resources consumed (in particular mass, power, and cost) while keeping the system modular enough to accommodate a wide variety of missions.

5.1 Rational for Implementing Radiation Tolerance through Synthesis of Hardware and Software

Traditionally, fault-tolerance was implemented exclusively through physical modifications to the hardware (discussed in Chapter 3); however, these methods are ex-
expensive, decrease performance, and are beginning to approach design limitations. A completely software-based fault-tolerance is not acceptable, as a hardware failure could not be serviced and would prematurely disable the system. A combination of hardware and software for fault-tolerance would allow each fault-tolerance arena to compliment the other. For example, processor intensive fault-tolerance algorithms could be coupled with high performance non-RadHard processors, thereby enabling the hardware’s speed to compensate for the software algorithm’s increased processing requirements and the software to allow for lower priced processors to be used. This dynamic union creates a versatile design space that can scale performance and reliability levels to meet the system’s needs.

5.1.1 Comparison with Hardware-only System

The baseline hardware-only fault-tolerance method for COTS components is to operate redundant hardware in lockstep and compare their results. This method has proven itself reliable in past space missions [10]. Alternatively, the RadHard baseline is the RAD750, the top of the line RadHard processor [53]. The COTS-based architecture based upon a distributed system proposed in this thesis aims to meet or exceed both of these existing baselines in performance while providing sufficient reliability to successfully accomplish the spacecraft’s mission. Table 5.1 enumerates key characteristics of these baseline systems.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Maximum MIPS</th>
<th>Minimum Size (mm²)</th>
<th>Power (W)</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock Stepped</td>
<td>690</td>
<td>1875</td>
<td>18.75</td>
<td>$175</td>
</tr>
<tr>
<td>RadHard</td>
<td>400</td>
<td>625</td>
<td>5.0</td>
<td>$200k</td>
</tr>
<tr>
<td>3 MCU System</td>
<td>108</td>
<td>312</td>
<td>0.25</td>
<td>$21.02</td>
</tr>
<tr>
<td>10 MCU System</td>
<td>357</td>
<td>1039</td>
<td>0.82</td>
<td>$70.07</td>
</tr>
<tr>
<td>3 Embedded Processors</td>
<td>2677</td>
<td>748</td>
<td>2.35</td>
<td>$135</td>
</tr>
</tbody>
</table>

The systems in Table 5.1 illustrate several key design issues. Firstly, COTS components increase performance and decrease cost. Secondly, smaller embedded/MCU-
based systems reduce power consumption. To illustrate the broadened design space from incorporating embedded COTS processors and MCUs multiple theoretical configurations are listed, each demonstrating a unique contribution (3 MCUs - smallest, cheapest; 10 MCUs - low system impacts from high number of processors; 3 embedded processors - highest performance).

**Monetary Cost**

Implementing a non-RadHard system requires multiple processors and their supporting hardware to allow for continued operations after device failure. The price differential between COTS and RadHard components is significant, for the price of a single RadHard processor, over 100 of the most expensive COTS processors could be purchased. While the price differences between COTS processors is minimal, having a maximum difference of $\sim$1k (1.25% of a RadHard processor’s price).

One cost not accounted for in these comparisons is the cost of testing to validate the reliability of the system. As noted by Avery, testing costs can become significant [4]. However, a large portion of this cost is viewed as a onetime expense to verify the architecture performs as expected. Once this testing occurs the cost is reduced.

**Performance**

The performance of multiple processors operating in lockstep is less than or equal to that of a single processor. In the operating mode used by Boeing’s lockstepped COTS system, the hardware periodically restarts to maintain a consistent state and thereby eliminate the chance of hidden errors accumulating [10]. This further reduces the performance capabilities of the system. In a distributed system with no redundant operations, the theoretical maximum processing power is equal to the summation of each individual processor’s capability. As redundancy is added, to increase the system’s reliability, performance drops proportionately to the level of redundancy. In tests with high-performance, dual-core processors performing redundant operations have proven to cause as little as a 5% drop in performance when compared to a single execution on a single processor system [45].
In a COTS-based distributed architecture, the system’s performance is directly related to the processor’s performance and the number of processors. Thus an MCU-based system will have low performance capabilities, while a system of embedded processors (such as Analog Device’s Blackfin or Freescale’s i.MX) will have high performance potential. The system costs (price, power consumption, and size) of both setups remain competitive with a lock stepped hardware system. However, a unique trade is created where satellites with simple payloads (such as CubeSats) could rely on a redundant MCU-based system and satellites with complex payloads, requiring increased computational capability, could incorporate redundant embedded processors. Further, a hybrid approach could be utilized where multiple embedded processors could be utilized for computationally intensive tasks such as three dimensional matrix processing and MCUs for general satellite operations and I/O.

**Operational Lifetime**

The space environment is a challenge to long term operations where maintenance is not available. The duration a processor can properly function in a radiation environment is component specific, with RadHard components having demonstrated greater than 15 years of continuous operations. To ensure these long lifetimes, RadHard components are fabricating to withstand a minimum of 100 krad (Si) of radiation dosage (with processors often being designed to withstand over 1 Mrad (Si) [53]). However, as mentioned in Chapter 3, COTS processors are becoming more tolerant to TID, leaving only SEE-induced errors which must be corrected.

With redundant processors operating in lockstep, the largest challenge to maintaining reliability is ensuring proper operation of the compare unit, which poses a single point of failure for the system in addition to being the largest performance bottleneck. A failed comparison unit renders the redundant system useless, and the comparison rate regulates the system’s maximum performance, as all processors must stall while waiting for the comparison. A distributed system eliminates this component and places the comparison operation within the processors. While at first this appears to simply shift the failure point to the processor performing the compar-
ison, by utilizing multiple processors for the comparison and requiring a majority consensus, the single point of failure has been shifted to the entire redundant system, making it more robust. There is still a vulnerability as the compare operation is not atomic and thus could suffer from an upset, but the probability is greatly reduced when compared to a dedicated voting circuit. Assuming the compare operation does not suffer from an upset, a majority of the processors must have identical errors in the result for the comparison to produce an incorrect output, which has a low probability of occurring. Allowing faulty processors to be identified as those that disagree with the final output. The distributed architecture also eliminates the performance bottleneck, by allowing processors to perform tasks independently while waiting on the comparison results from other processors, utilizing time that was previously wasted. Timing constraints do arise from the inter-processor network’s data rate; however, this only limits the granularity between comparison points in a given function.

Using a multiple processor design contributes to graceful degradation by allowing the system to continue performing the mission in a degraded, less-efficient manner after a processor fails. An MCU-based design poses tremendous potential, as the cost of incorporating additional processors is nearly negligible, requiring over fifteen MCUs to consume the same area utilized by Boeing’s processors operating in lockstep [10] [17]. These extra MCUs can be used as cold backups, only brought online when an MCU fails to extend the system’s operational life beyond that of a single MCU.

Compatibility

The predominant satellite avionics design requires interfacing with a single RadHard processor. Thus replicating device’s inputs to multiple processors is a current design challenge; however, many devices are migrating towards a plug-and-play (PnP) or universal satellite bus interface and have begun shifting component interfaces towards a bus-based communication standard that is ideal for connecting to an array of processors. Thus while there are currently issues in distributing components’s I/O between multiple processors, future satellite components will not encounter this challenge.
5.1.2 Benefits Created by Using both Hardware and Software

When the space industry was driving the design of microelectronics, the performance capabilities of RadHard and COTS processors was minimal. Since the major limitation facing spacecraft designers then was the processor's computational capability, it necessitated putting the reliability into the hardware for increased performance. However, technology has progressed and now most spacecraft missions are not computationally bottlenecked by the performance capabilities of modern processors, as shown in Chapter 4. In fact, processors have much greater performance capabilities than spacecraft currently require. This extra processor power could be used for implementing the reliability in software, thereby shifting some of the reliability demands away from the hardware, allowing simpler hardware measures to be used lowering design time and unit cost. This synthesis of using both hardware and software for reliability creates the potential for developing truly optimized spacecraft avionics.

As noted previously, this symbiotic approach of using hardware and software to support each other greatly broadens the design options. Both hardware and software bring unique advantages to error mitigation, and their combination allows for system optimization where one to covers the vulnerabilities of the other. Through the complimentary nature of using both hardware and software, it enables small, low cost systems – such as CubeSats – to guarantee a modicum of reliability. Thereby allowing CubeSats to support high-risk missions. Additionally, it enables more onboard processing of complex data, more payloads to be incorporated onto the same physical bus, and/or increase the complexity of operations performed. This migration away from employing only hardware for reliability could revolutionize the satellite design approach by eliminating one of the largest impediments to spacecraft design: the high cost, low performance avionics.
5.2 COTS-based Multiprocessor Architectural Requirements and Goals

The avionics system is the glue that interconnects the different systems in a satellite and enables the spacecraft to perform complex tasks. A robust avionics system must be capable of handling the foreseen system needs and carries a margin for unforeseen needs, such as the need to sample sensors at a higher frequency to mitigate aliasing. The design of an avionics architecture for using multiple COTS processors to produce capabilities similar to or better than current RadHard architectures is outlined below.

5.2.1 Fully Capable CD&H System

At a minimum, the avionics system must perform the spacecraft’s command and data handling (CD&H) operations. A way of quantifying this duty capabilities is to sum the requirements individual tasks must perform, a listing of necessary tasks is shown in Table 5.2. These tasks are central to the avionics system successfully fulfilling the C&DH functionality whereby the avionics system facilitates interactions between various systems, process and collect all data, and enforce all timing requirements. The requirements for supporting these tasks span both physical hardware (e.g. interface to components) and software (e.g. perform control calculations at 5 Hz).

Requirements

The tasks listed in Table 5.2 illustrate the three foundational capabilities an avionics system must meet: computational throughput (e.g. performing state estimation at desired frequency), data storage (e.g. maintaining all sensor data between communication downlinks), and ability to interface (e.g. enabling connections to payloads). In order to quantify the requirements on these capabilities, a theoretical, barebone system design was created.

The barebone satellite is assumed to be three-axis stabilized with a control frequency of 1 Hz. All components are connected through the same serial data bus
Table 5.2: Standard Satellite Tasks

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Operations</th>
<th>Housekeeping Data Collection</th>
<th>Interfacing</th>
<th>Periodic Control</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Command Processing</td>
<td>Sensor Data Collection</td>
<td>Component Connections</td>
<td>Sensor Data Processing</td>
<td>Operation</td>
</tr>
<tr>
<td></td>
<td>System Monitoring</td>
<td>Actuator Control</td>
<td>Exchange Information/Data</td>
<td>State Estimation</td>
<td>Data Collection</td>
</tr>
</tbody>
</table>

and have a data readout rate of $<100 \frac{\text{bits}}{\text{second}}$ (bps). It is assumed to have a single payload which communicates over the serial bus that the other components use, and it requires minimal data processing and storage. From these specifications, each capability’s minimum requirements have been determined and listed in Table 5.3 along with the total avionics system requirements.

Table 5.3: Bare Bones Satellite Requirements

<table>
<thead>
<tr>
<th>Category</th>
<th>Computational Throughput (MIPS)</th>
<th>Data Storage (bps)</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations</td>
<td>1</td>
<td>80</td>
<td>N/A</td>
</tr>
<tr>
<td>Housekeeping Data Collection</td>
<td>0.1</td>
<td>2,048</td>
<td>Serial</td>
</tr>
<tr>
<td>Interfacing</td>
<td>N/A</td>
<td>N/A</td>
<td>Serial</td>
</tr>
<tr>
<td>Periodic Control</td>
<td>1</td>
<td>32</td>
<td>N/A</td>
</tr>
<tr>
<td>Payload</td>
<td>0.5</td>
<td>512</td>
<td>Serial</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2.6</strong></td>
<td><strong>2,672</strong></td>
<td><strong>Serial</strong></td>
</tr>
</tbody>
</table>

An aspect left out of the data in Table 5.3 is the differentiation between time critical tasks and time independent tasks. Certain operations, such as the periodic control task, will produce incorrect outputs if not held to rigid timing constraints;
while others, such as the payload task(s), can occur at any point, within the tasks’s operating window, without damaging the satellite’s performance. These two types of tasks are used by the COTS-based architecture to increase performance by weaving time independent tasks amongst time critical tasks (similar to what is shown in Figure 5-1). This performance increase is compounded by allowing separate hardware units to each execute different time independent tasks simultaneously.

**Goals**

Fulfilling the requirements listed in Table 5.3 does not improve upon existing spacecraft avionics. As the avionics system is the backbone upon which spacecraft capabilities are developed, the utilization of new capabilities is vital to spacecraft advancement. Certain goals have been added into the architectural design of the avionics system to ensure that the capabilities present in today’s modern devices are made available. These goals include minimizing price, minimizing power consumption, and maximizing computational throughput.

- **Size and Cost**

  One of the largest advantages a COTS-based design has over a RadHard system is its size and cost. The physical size of COTS components are continually being reduced while the size of RadHard components have stagnated. As mentioned in Chapter 4, reducing the avionics size leads to decreases in satellite mass. Additionally, the reduced physical size lowers the area sensitive to high energy particles, reducing the number of potential upsets. The most expensive COTS system cost is an order of magnitude below the least expensive RadHard component’s cost. This reduction in price is an enabler for future spacecraft missions, such as fractionated systems, where the avionics system must be replicated across all satellites. In addition to primary components (i.e. processors) the supporting equipment also adds board area usage and monetary costs to the system design. Thus keeping the complete system size and cost to a minimum inspires growth and reduces the avionics system’s impact on the spacecraft.
• **Power Consumption**

In terms of power consumption, a COTS-based systems presents a tremendous advantage over RadHard systems; however, not all COTS components generate this advantage. The spectrum of available COTS processors ranges from MCUs which consume <5 mW [36] to high performance, multicore desktop computer processors which consume >135 W [21]. Total power consumption is a critical aspect of a satellite design as many system characteristics stem from the power generation requirement; therefore, minimizing the power required by the avionics system (which is always powered on) poses tremendous benefits to the entire satellite. A hidden aspect of the avionics system’s power consumption is from the supplemental components (e.g. external memory arrays, interface ICs, etc.), thus minimizing the required supporting ICs is highly beneficial as it not only minimizes the avionics system’s power consumption but also the size, cost, and system complexity.

• **Performance**

The adoption of COTS devices could revolutionize spacecraft avionics through the increasing computational throughput. Many COTS devices can perform several orders of magnitude more MIPS than the top of the line RadHard processor (see Figure 4-2(a)). This increase in computational capabilities poses the greatest advancements to missions spacecraft can perform because it enables complex operations to be performed with increased autonomy.

Since the COTS-based avionics architecture being designed in this thesis requires multiple processors/cores to create a distributed system, it does not have to implement lockstep operations (as Boeing’s COTS-based system does [10]) allowing it to operate at higher performance levels. A distributed system allows processors to operate independently setting the maximum with a maximum theoretical performance equal to the summation of the individual processor’s performance capabilities. Furthermore, it alleviates a major redundant design challenge of maintaining clock synchronization, which can lead to false error
detections when synchronization is lost. Increasing the system’s processing throughput promotes the expansion of spacecraft mission capabilities.

By pushing the avionics system’s architectural design with these goals in size, cost, power consumption, and performance the architectural design will improve upon existing spacecraft avionics. Additionally, they aid in demonstrating how COTS components enable future satellite missions.

5.2.2 Adaptability to Different Missions

A different MIT Space Systems Laboratory (SSL) project, Synchronized Position Hold Engage and Reorient Experimental Satellites (SPHERES), demonstrated that the same hardware can support separate missions if a modicum of adaptability is included in the initial design. SPHERES is an autonomous control and formation flight testbed on-board the International Space Station (ISS) [49], and has tested capabilities outside of its original design goals because of its hardware expansion port (e.g. vision based navigation [56]).

Requirements

The design requirement of being able to accommodate future needs was inspired by this lesson from SPHERES. It places an emphasis on remaining robust/adaptable to different satellite missions and their specific needs. An ideal avionics architecture would require minimal modification when migrating between different satellite missions (creating a plug and play capability), thereby allowing for maximum re-usability between satellites.

The design must be capable of growing/adapting to unknown future mission and component needs. Potential growth vectors include the number of processors and the interchangeability of the hardware components. A satellite could require N processors for computational or reliability requirements, ensuring that the architecture supports a variable number of processors enables it to satisfy these mission requirements.
Another architectural adaptability requirement is supporting migration between different hardware components, without a complete system redesign. For example, one system might aim at simplicity and utilize multiple MCUs interconnected via a serial peripheral interface (SPI) bus; while a more advanced, higher performance system might have multiple embedded processors interconnected with TTEthernet [40]. The overarching architecture should not be limited to specific hardware components. In summary, the architecture’s adaptability requirements are to support future design options/modifications and hardware versatility.

Goals

An ideal avionics architecture is able to support multiple missions without significant modifications, while simultaneously lending itself to support unique, mission specific operations. This often leads to suboptimal designs, but the benefits from increased flexibility outweigh these losses. The architecture designed in this thesis aims to demonstrate this claim and therefore has the goal of being highly adaptable both in terms of overall versatility and functional modifications.

- **Architectural Adaptability**

Architectural adaptability is the versatility of the components to support different missions needs. An example of this would be the ability/ease of increasing the non-volatile memory size. Can it be increased by simply changing ICs or does it require a new board layout/software structuring? Thus a design goal is to structure the architecture to prefer standard hardware and software interfaces which allow for growth. This applies beyond processors to include critical supporting components such as ADC/DACs, power regulators, and interface ICs (e.g. serial transceivers or serial protocol converters). Since each mission will require different interfaces, a diversity of hardware interfaces is desired. One option is to utilize components with extra, unused interfaces. Alternatively, some component families offer different models for each supported interface. Architectural adaptability ensures that the architecture is not simply a one-
time solution, by remaining viable as technology matures and ensuring that the components can support different functions and interfaces.

- **Functional Adaptability**

Functional adaptability refers to the hardware’s ability to support a broad array of mission types. Design characteristics which enable this adaptability are increasing processing power and unique uses of the multiple processors. Higher computational capabilities allows for different software implementations of the same operation. For example, increasing a system’s computational capabilities allows it to implement higher control frequencies or operate additional payloads. These capabilities provide a foundation for system growth.

Multiple processors enable unique processing constructs where the system is capable of performing more than any individual component. These constructs range from distributed processing of commands to parallel calculations on large data sets. Allowing for the advantages of these constructs to be fully explored is critical for creating progress in future avionics architectures as not all applications/utilizations of the architecture. Thus a goal of avionics architecture is to create a foundation for further development of the capabilities generated by the architecture, particularly in the field of distributed and parallel processing.

### 5.2.3 Radiation

The COTS-based avionics system must be capable of guaranteeing sufficiently reliable operations in the space environment over the mission lifetime despite the hardware’s inability to guarantee this trait. The architecture does not need to completely stop radiation effects from occurring, but ensure that any radiation induced errors do not prohibit the spacecraft from accomplishing its mission. Ideally, this reliability is created by EDAC; however, errors with minimal impacts can be allowed. As the system ages, individual components may fail from accumulated radiation dosage, but the system will gracefully degrade by maintaining enough functionality to perform its mission. The author has termed these requirements radiation tolerance.
Requirements

For a mission altitude <1,000 km and duration <2 years, the maximum TID the electronics behind \( \frac{1}{16} \) in (1.5875 mm) of Al (the anticipated minimum thickness of a satellite’s avionics box) are anticipated to experience <6 krad (Si) as show in Figure 5-2. Modern electronics with a feature size \( \leq 250 \) nm have a TID hardness >40 krad (Si) [4], an order of magnitude greater than the expected dosage level. Thus all designs meeting this maximum feature size requirement will be assumed to have sufficient TID tolerance to operate within these orbital constraints.

Figure 5-2: TID based upon Shielding Thickness, for a 1km Circular, Equatorial Orbit

To tolerate SEE-induced errors the architecture must perform redundant checking, which will not completely mitigate these errors but will greatly reduce the probability of them propagating. Redundant checking can take the form of comparing the data stored in multiple memory locations, which can be augmented with error correcting codes and periodic scrubbing. However, this will not mitigate all SEE-induced errors. Some could corrupt control/configuration registers, resulting in the device still operating but not producing correct results. Such errors are mitigated by checking the operation’s results with ones generated on a redundant copy of the functional hardware, such as multiple processing cores or processors. Functions executed across
redundant hardware can be checked by running the results through a consensus algorithm such as Paxos [13] to determine the correct output. This requires the redundant processors be capable of communicating with each other.

Beyond redundant checking of the results, checking for liveness mitigates other SEE-induced errors. Upon detecting a failed processor, the live processors divide up the critical tasks and one takes on the job of reviving the failed processor. During this time, the system may stop performing non-critical tasks as it is operating in a degraded state. If the failed processor is brought back to an operational state, the operational tasks are redistributed and full system functionality is returned. This process is shown in Figure 5-3. In summary, the avionics system must be enclosed within an Al enclosure at least $\frac{1}{16}$ in (1.5875 mm) thick for TID mitigation; and it must include multiple, interconnected processing units (cores or processors) to support software-based EDAC.

![Figure 5-3: Operational Schedule for Reprogramming a Faulting Processor](image)

Figure 5-3: Operational Schedule for Reprogramming a Faulting Processor
Goals

There is a large spectrum of potential errors that radiation effects can induce in the avionics system (see Chapter 2 for in depth details). The primary type of radiation errors targeted by the avionics architecture designed in this thesis are those leading to incorrect outputs (from data corruption errors). These errors generally result from upsets in memory locations or configuration bits. Other radiation induced errors include control flow errors, latchup errors, and functional interrupts.

This architecture relies primarily on software algorithms for radiation tolerance (such as those created by Oh [47]). This architecture aims to support implementing and testing a wide array of these algorithms, allowing the architecture to compare different methods for characterizing their strengths and weaknesses. The ability to test different algorithms broadens the versatility the architecture must provide.

Robustness against hardware failure is needed to enable software-based protection, as reliable commands to devices are worthless if they cannot be sent to the device. An ideal method of implementing this hardware robustness is by exclusively using devices which communicate over a bus and connecting the processors to the bus, ensuring the ability to communicate with the device as long as a processor is operational. However, this greatly narrows the device selection field as some devices cannot communicate over a bus, thus alternative solutions must be utilized. A possible solution is shown for devices requiring only GPIO in Figure 5-4.

![Figure 5-4: Hardware Output Redundancy Circuit](image)

Figure 5-4: Hardware Output Redundancy Circuit

103
Additionally, the architecture should enable analysis on the costs and benefits of using hardware versus software for fault-mitigation schemes, as the development of this trade space will drive improvements of the COTS-based architecture designed in this thesis. This further impacts the future of spacecraft avionics systems as the author anticipates that fault-tolerance will be through a combination of hardware and software mechanisms.

Summary

The COTS-based avionics architecture must meet the following criteria: it must be able to perform all the computational requirements of the spacecraft’s operations, it must be able to be used on different satellite missions, and it must have redundant hardware which uses primarily software for creating radiation tolerance. The requirements which the avionics architecture must provide are shown in Table 5.4.

<table>
<thead>
<tr>
<th>Category</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability</td>
<td>Exceed items in Table 5.3 and meet or exceed Rad-Hard processor</td>
</tr>
<tr>
<td>Adaptability</td>
<td>Able to be transferred to other satellites, not fixed to a single hardware component</td>
</tr>
<tr>
<td>Radiation</td>
<td>Has redundant hardware, utilizes software for EDAC of data corruption errors on critical outputs</td>
</tr>
</tbody>
</table>

To ensure that the architecture pushes the bounds of current avionics capabilities, specific design goals were added to the architecture’s design. In particular, these goals focus upon reducing the avionics system’s costs (size, cost, power consumption, and performance), developing its ability to be re-used on separate satellite missions, and providing different EDAC options for radiation-induced errors based upon the operation’s criticality and performance needs. These architectural design goals are listed in Table 5.5.
<table>
<thead>
<tr>
<th>Category</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size and Cost</td>
<td>Minimize</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>Minimize</td>
</tr>
<tr>
<td>Performance</td>
<td>Maximize, in particular harness potential from distributed/parallel processing</td>
</tr>
<tr>
<td>Architectural Adaptability</td>
<td>Be able to utilize on satellites with different missions</td>
</tr>
<tr>
<td>Implementation Adaptability</td>
<td>Be able to easily interchange components in the design</td>
</tr>
<tr>
<td>EDAC Algorithm  Adaptability</td>
<td>Be able to utilize different algorithms, not be locked into a single method</td>
</tr>
<tr>
<td>Radiation Errors</td>
<td>Be able to mitigate data corruption errors and handle events such as SEL, SEFI, etc</td>
</tr>
</tbody>
</table>

5.3 System Design Trades/Considerations

Many of the goals discussed in the previous section are in contention with each other, for example minimizing the power consumption and maximizing the performance capability cannot both be accomplished. To determine an optimal design, the design goals must be traded, the analyses are shown in further detail below.

5.3.1 Capabilities Trades

High performance is in contention with nearly every other design goal as processor performance is directly proportional to its size, cost, power consumption, complexity, and radiation tolerance. However, increasing system performance is a critical enabler of future satellite missions. Thus these design factors are analyzed and ranked for their priority of being achieved.

Size and Cost

COTS processors’s performance is directly related to their size. However, the physical size of an avionics system is rarely a constraining item in satellite designs, with a possible exception being CubeSats due to their stringent volume constraint. Thus size has a low priority, and is the final factor considered in system optimization.
The cost differential between RadHard and COTS is tremendous; however, the cost differential between different COTS processors is minimal (<$1k, \sim 1\% of a RadHard processor’s cost). When compared to RadHard systems the COTS-based architecture makes tremendous cost gains regardless of the performance. For this reason, cost is given a low priority when optimizing the system.

**Power Consumption**

The most critical trade is between power consumption and processor performance. There is a full spectrum of available options which follow a general trend of increased performance capabilities resulting in increased power consumption. The spectrum is bounded by MCUs on one end, requiring low-power (<5 mW) for low-performance (<5 MIPS) [36], and multicore server processors on the other end, requiring high-power (>135 W) for high-performance (>10,000 MIPS) [21]. RadHard systems such as the RAD750 (5 W, 400 MIPS) [53] are within this spectrum.

Neither high-performance nor low-power consumption can be completely maximized while still meeting the other system design goals. The two must be traded with a goal of keeping the system’s power consumption below that of a RadHard system and the processing throughput above that of a RadHard system. A solution was not found; however, solutions favoring each side were found.

For minimal power consumption, MCUs (such as [36], [18], or [34]) should used. These devices offer the highest MIPS/W (seen in Figure 4-3) allowing systems to be designed that consume minuscule amounts of power (<1 W). However, they struggle to provide computational performance greater than state of the art RadHard systems. MCUs outperform the RadHard processors listed in [62], but do not outperform the RAD750. An MCU-based system’s performance capabilities can be increased by utilizing a large number of MCUs in a highly distributed architecture. In such a system, an MCU-based system could provide more computing throughput for less power than the RAD750.

A maximum performance system would utilize embedded processors (such as Texas Instrument’s OMAP [35] or Freescale’s i.MX51 [33]). These processors deliver
impressive computing capabilities (>1,000 MIPS) for <5 W of power consumption which appears to meet the goal of outperforming RadHard systems. However, the requirement for redundant hardware often makes these systems exceed the power consumption of a RadHard system. Additionally, an FPGA or similar device would be needed to support the required I/O capabilities, further increasing the power demands. As COTS components continue to improve, this system design will be able to outperform a RadHard system in all categories, making it an optimal design. A processor which meets this description is Analog Device’s Blackfin [27], which can provide >1,000 MIPS while consuming <0.5 W. Thus making it possible to incorporate multiple processors and an FPGA while remaining under the power consumption levels of the RAD750.

Complexity

A design aspect which is often overlooked due to lack of quantifiability is the system’s complexity. A design can be feasible, yet so complex it impedes wide spread adoption. In general, as processors increase in computational capability, their simplicity and I/O capabilities decrease. Thus high performance system create two complexity challenges, the first is debugging problems with the advanced processor and the second is having sufficient I/O. The first hurdle is overcome by copious datasheet reading and testing; while the second generally requires additional ICs (typically an FPGA). Lower performance designs, such as an MCU-based system providing <200 MIPS, avoid both of these challenges. To increase the probability of the architecture being adopted, complexity was given a medium priority.

Radiation Tolerance

A final trade which must be considered is the hardware’s interaction with the radiation environment as it relates to performance. As discussed earlier in this thesis, decreasing process sizes (i.e. increase performance) have increased TID tolerance but an increased SEE-induced error rate. For the orbital constraints specified, TID tolerance is not an issue (as discussed earlier). Thus the design goals are to minimize
SEE-induced errors (increase process size) while increasing computational capabilities (decrease process size). Additionally, the challenges of SEE are foreseen to dramatically increase for devices using process sizes <90 nm. Thus an optimal design will utilize a technology a couple generations older than 90 nm process size for the highest computational throughput while increasing the basic hardware SEE tolerance. In general, the highest performance desktop and server processors are fabricated on the smallest process sizes (with state of the art processors currently use a 32 nm process size), embedded processors are fabricated on process sizes a couple generations older (currently 90 - 150 nm), and MCUs on even larger process sizes (currently 180 - 250 nm). The desire to keep the probability of a SEE-induced errors relatively low resulted in this trade being given a medium priority.

5.3.2 Hardware Design Trades

For the optimal avionics architecture, several hardware design factors must be analyzed. The primary design factors are the number of processors in the system and the methodology used to interconnect peripheral components to the processor network. The advantages and disadvantages of different design options are considered below.

Number of Processors

The architecture requires hardware redundancy. However, the extent or method of the redundancy is not specified. Complexity is minimized by having only two processors, because reliability is increased and interfacing with peripheral is clearly defined. However, only having two processors limits the fault-tolerant capabilities, as well as the adaptability and performance of the system.

A minimum of two redundant operations are required to detect errors, and three for determining the correction to apply after detecting an error. Systems such as SpaceMicro’s TTMR [25] rely on this fact, and primarily use two redundant processors but have a third which is activated after an error is detected for determining the correct result. Additional processors, beyond those required for reliability oper-
ations, can be powered down if not in use, thereby minimizing any radiation dosage that would have accumulated. Additionally, extra processors increase the adaptability of the system, allowing it to perform a wider array of operations. Finally, a highly redundant system can be used to explore distributed system techniques for load balancing or performance increases. As long as a system implements a minimum of three processors the requisite reliability is available, making this a low priority trade.

Peripheral Interfacing

Ideally, a fully redundant system would have every processor interfaced with every peripheral. However, this creates demands upon the components which the architecture cannot support. Some missions call for devices which can only be connected to a single processor, limiting the robustness of the device’s operability. In some cases esoteric methods can be invoked to create redundant connections; however, these increase the complexity of the architecture. In some situations it may be advantageous to risk of losing a component’s functionality in order to minimize the risks from increasing complexity. As interfacing with devices is critical to proper satellite operations, this was ranked as a medium priority trade in the architectural design evaluation.

5.3.3 Hardware and Software Fault-tolerance Trades

A final tradespace investigated in the development of the avionics architecture is the aspect of software and hardware fault-tolerance. This thesis desires to depart from the industry standard or providing radiation tolerance by hardware-only mechanisms (i.e. RadHard components), yet not to an extent where complete reliance is on software mechanisms (i.e. SIFT [47]). The strengths of both hardware and software mechanisms are sought, to create synergistic gains from applying both.

Some EDAC algorithms perform checking in software, while others utilize external hardware components; this thesis aims to apply both approaches for achieving higher reliability. For example, an output which must be robust to internal processor upsets (i.e. upsets in memory values, control flow errors, etc.) could rely on ex-
ternal hardware to verify consistent outputs from multiple devices using components such as RadHard AND gates (simple schematic implementation shown in Figure 5-5). While less critical operations can rely on software-only checking. This flexibility in the reliability level an operation needs is a key aspect of this trade.

Operations done in hardware are typically faster than the equivalent in software; however, software has the advantage of modularity, making it easier to apply to a plethora of devices. As noted earlier, hardware fault-tolerance has a lengthy development time and is costly to fabricate; and software fault-tolerance is inherently weaker as errors can occur within the fault-tolerance mechanism. Thus it is desired to have mechanisms such as ECC in memory arrays implemented in hardware for its performance benefits on all operations. Augmenting the hardware ECC with a software-based periodic scrubbing algorithm further strengthens the system’s fault-tolerance. Identifying and developing additional symbiotic relationships, such as the one discussed above, between hardware and software is a goal of this trade, making it rank as a medium priority.

A complete listing of the trades encountered in structuring the avionic architecture are shown in Table 5.6. These trades were either towards achieving higher performance or increased radiation tolerance, with varying levels of impact upon the avionics system. High impact trades were given high priority levels to ensure design choices focused around these trades.
Table 5.6: COTS Avionics Architecture Trades

<table>
<thead>
<tr>
<th>Trade</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase Performance</td>
<td>Decrease Size and Cost</td>
</tr>
<tr>
<td>Number of Processors</td>
<td>Radiation Tolerance</td>
</tr>
<tr>
<td>Increase Performance</td>
<td>Decrease Complexity</td>
</tr>
<tr>
<td>Number of Processors</td>
<td>Performance</td>
</tr>
<tr>
<td>Hardware-based Fault-tolerance</td>
<td>Software-based Fault-tolerance</td>
</tr>
<tr>
<td>Increase Performance</td>
<td>Increase Radiation Tolerance</td>
</tr>
<tr>
<td>Increase Performance</td>
<td>Decrease Power Consumption</td>
</tr>
</tbody>
</table>

Note:* Low priority assumes a minimum of three processors.

5.4 Hardware Architecture Design

Using the design requirements (reference Table 5.4), goals (reference Table 5.5), and trades (reference Table 5.6), a COTS-based avionics architecture was constructed. As a foundation, the architecture utilizes a minimum of three interconnected processors. If the processors are high performance embedded processors/digital signal processors (DSPs) an I/O enhancing device such as an FPGA would probably be necessary.

Figure 5-6: Architecture Foundation

Note:* An FPGA introduces a single point of failure, all FPGA operations must implement internal TMR or similar checking as a minimum precaution.
The baseline architectural model with external hardware checking (Figure 5-6(a)) and internal software checking (Figure 5-6(b)) is shown in Figure 5-6 (note: the hardware and software checking can be implemented simultaneously). From this basic construct, device interfaces with the processor network can be setup in a plethora of ways depending upon the reliability design and the device’s operating characteristics. A few design options are discussed below.

Many spacecraft devices communicate using a bus protocol. This type of interface is well suited for a multiprocessor system as the bus can be connected directly to each processor. Devices with a bus interface are ideally connected to the architecture as shared peripherals, shown in Figure 5-7. This connection scheme allows for multiple operating modes, ranging from guaranteeing high-reliability outputs to guaranteeing quick interactions with the device. High-reliability is generated by having a dedicated processor act as the master, where it is primarily responsible for interacting with the device, and all the other processors snoop on the bus for data sent by the device. Whenever data from the device is detected all processors perform the necessary calculations on the data and relay their results to the processor acting as the master. The master will calculate the consensus value and output it to the device, shown in Figure 5-8(a) (where the * denotes the processor acting as the master). Minimal time delay interactivity is created by having the processor serving as the master alternate in a round-robin fashion. To ensure optimum system performance, the first processor capable of handling the device’s request will notify the other processors that it is servicing the device and independently process the device’s request, as show in Figure 5-8(b). Many variants and combinations exist between these two endpoints.
Some devices only support connecting to a single processor, such as simple serial protocols. If a device like this were to be directly connected to multiple processors there is a potential that separate processors will drive the data lines oppositely at the same instant. However, this arbitration problem can be solved with external circuitry. However for simplicity, these devices can simply be connected to a single processor, as shown in Figure 5-9. While this does create a single point of failure, ideally this strategy will only be used for devices which are a part of a set, such as torque coils. Thus the full functionality is spread across the system, minimizing the impact of a processor failure. Other devices with this communication protocol should use arbitration circuits to ensure their ability to communicate with the avionics system. This interface method trades simplicity with risk. Further, it increases the reliance on the interprocessor bus as data must be passed for proper device operation.
To construct an avionics system within this architectural framework, use the system’s requirements to determine the level of processor needed (i.e. MCU or embedded processor) and the number. Using an estimate of the volume of data that must be communicated between the processors, select an interprocessor communication medium along with a network topology. Finally, interface the required peripherals to the avionics system using the methods described above. The interface method is selected based upon the peripheral’s supported communication protocols and its criticality to proper spacecraft operations, where higher criticality devices should be interfaced with more processors. A block diagram of the system can be constructed from these design decisions. Next, components selection should be finalized and initial prototyping can begin to verify that the system will meet the mission requirements and be robust to radiation induced errors.

5.5 Design Considerations for Implementing Software-based Fault-tolerance

There is a trade space between performance and reliability when implementing the fault-tolerance algorithms. This is a unique trade space in the design of satellite avionics systems because a traditional avionics architectures forces the design to set the reliability level to meet the needs of the most critical operation. The COTS-based avionics architecture turns this paradigm around and provides a spectrum of reliability levels which functions can choose from to meet their reliability needs. Thus the initial system design must consider the broad spectrum of software operations’s performance and reliability requirements to ensure that both are adequately met.

5.5.1 Satellite Software Operations

There are several basic operations that a satellite avionics system must perform (reference Table 5.2), each of which has a generic range of performance and reliability constraints. These have been specified for small, LEO satellites in Table 5.7. The
values in Table 5.7 are averages which will be used for understanding the implementation of software-based fault-tolerance on the COTS-based, multiprocessor architecture. Table 5.7 should only be used as a guide, the final implementation should scrutinize all operations independently to determine their exact constraints.

<table>
<thead>
<tr>
<th>Software Operations</th>
<th>Performance</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Processing</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>System Monitoring</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Sensor Data Collection</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Actuator Control</td>
<td>Medium</td>
<td>Low - Medium</td>
</tr>
<tr>
<td>Exchange Information/Data</td>
<td>Medium - High</td>
<td>Low - High</td>
</tr>
<tr>
<td>Sensor Data Processing</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>State Estimation</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Control Commands</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Payload Operation</td>
<td>Low - High</td>
<td>Low</td>
</tr>
<tr>
<td>Payload Data Collection</td>
<td>Low - High</td>
<td>Low</td>
</tr>
</tbody>
</table>

In general, for the operations listed in Table 5.7 the performance and reliability requirements are opposite of each other (e.g. command processing requires low performance and high reliability). This is seen when analyzing a satellite’s response to a command it receives from the groundstation. If a 500 millisecond delay is inserted between the satellite receiving the command and its execution there is no detriment to the satellite’s functionality. However, ensuring the proper operation executes is critical because accidentally performing the wrong command could impairs the satellite’s capability to function properly. Not all operations can tolerate the addition of delays. The state estimation has a high performance requirement as it is a computationally intensive process that must be repeatedly performed at a high frequency and a 500 millisecond delay could cause problems in stabilizing the satellite’s response to attitude commands. While an incorrect state estimation has a minor impact on the satellite’s functionality.

Similar observations can be made on the remaining satellite operations in Table 5.7. A simple test for evaluating the performance requirement is to characterize the impact of “dramatically” increasing the time required to perform the operation on
the satellite’s functionality. If the operation fails to execute properly or it negatively impacts the satellite’s functionality then the operation’s performance requirement is high (e.g. state estimation), whereas if it only creates a delay in the system’s response then the performance requirement is low (e.g. command processing).

An operation’s reliability can be evaluated by examining the impact of the operation creating an incorrect output. As SEE can cause a myriad of errors, one must examine the results from the incorrect output with the worst possible impact on the system. If the output generates a condition which endangers future satellite operations, the operation’s reliability requirement is high (e.g. command processing). Whereas if the output only contaminates payload data or results in an error with minimal repercussions (e.g. an incorrect state estimate) the operation’s reliability requirement is low.

### 5.5.2 Metrics for Applying the Proper Level of Fault-tolerance

After examining the software operations to determine their constraints and requirements, a designer can align each operation’s needs with the available solutions. The strengths and weaknesses of the primary options for generating reliability are listed in Table 5.8. The exact performance impacts of each reliability method varies with the characteristics of the operation it is protecting.

<table>
<thead>
<tr>
<th>Reliability Method</th>
<th>Reliability Strengths</th>
<th>Reliability Weaknesses</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Reliability</td>
<td>None</td>
<td>Cannot detect faults</td>
<td>No impact</td>
</tr>
<tr>
<td>SECDED Encoding</td>
<td>Corrects single bit errors</td>
<td>Cannot correct multiple bit errors</td>
<td>Decoding overhead</td>
</tr>
<tr>
<td>Redundant Execution, single processor</td>
<td>Corrects multiple bit errors</td>
<td>Cannot detect functional errors</td>
<td>225%-400%</td>
</tr>
<tr>
<td>Redundant Execution, separate processors</td>
<td>Spatial and temporal separation</td>
<td>Cannot guarantee timing</td>
<td>Message passing overhead</td>
</tr>
</tbody>
</table>
Reliability Methods

Table 5.8 highlights the characteristics of the primary reliability methods used for mitigating data corruption errors, and serves as a starting point for determining the correct reliability method to apply. Each row in the table is a distinct reliability method (discussed in Chapter 3), they can be implemented independently or in conjunction with others for increased robustness to errors (e.g. a function could apply redundant execution or both SECDED encoding and redundant execution). Combining reliability methods increases the overall reliability of the function by allowing one method to cover the vulnerabilities of another method. In addition to using multiple methods concurrently, one reliability method can be utilized until an error is detected at which point a more robust method is invoked to mitigate the detected error. Combinations of the four methods in Table 5.8 produce the eight reliability levels shown in Table 5.9 with example operations.

The performance characteristics specified in Table 5.8 are vague quantities because of the variability in the function which the reliability algorithms protect. For example, redundant execution on separate processors generates a 4,000% increase in processing time for simple operations such as incrementing values in a buffer; whereas the same reliability method applied to a Kalman filter calculation only creates a 162% increase in processing time. In general, more complex operations (those requiring longer processing time) will incur smaller performance degradations from the decoding or the message passing overhead.

Implementation

The eight reliability levels in Table 5.9 are loosely structured to provide increased robustness to errors and decreased performance as reliability levels increase, the matrix in Table 5.10 shows the performance and reliability provided by each reliability level. Selecting the reliability level for an operation is accomplished by matching the operation’s performance and reliability requirements (such as those in Table 5.7) to the amount of protection required. When matching a function to a reliability level,
Table 5.9: Reliability Levels and Sample Operations

<table>
<thead>
<tr>
<th>Reliability Level</th>
<th>Reliability Method</th>
<th>Sample Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No reliability</td>
<td>Pointing attitude state estimation</td>
</tr>
<tr>
<td>1</td>
<td>SECDED encoding</td>
<td>Buffer data</td>
</tr>
<tr>
<td>2</td>
<td>Redundant execution, on the same processor</td>
<td>Noncritical actuator commands</td>
</tr>
<tr>
<td>3</td>
<td>Redundant execution, on separate processors</td>
<td>Actuator outputs</td>
</tr>
<tr>
<td>4</td>
<td>SECDED encoded data with redundant execution on the same processor</td>
<td>Sensor data conversion</td>
</tr>
<tr>
<td>5</td>
<td>SECDED encoded data with redundant execution on separate processors</td>
<td>Critical sensor data processing</td>
</tr>
<tr>
<td>6</td>
<td>Redundant execution on the same processor compared between multiple processors</td>
<td>System monitoring</td>
</tr>
<tr>
<td>6'</td>
<td>Redundant execution on the same processor compared between multiple processors if different results occur on primary processor</td>
<td>Periodic control law computation</td>
</tr>
<tr>
<td>7</td>
<td>SECDED encoded data with redundant execution on the same processor with outputs compared between multiple processors</td>
<td>Command to change the satellite’s state</td>
</tr>
<tr>
<td>7'</td>
<td>SECDED encoded data with redundant execution on the same processor with outputs compared between multiple processors if different results occur on primary processor</td>
<td>Noncritical Command Processing</td>
</tr>
<tr>
<td>8</td>
<td>Hardware-voted redundant outputs</td>
<td>Critical payload device operation</td>
</tr>
</tbody>
</table>

the primary emphasis is on meeting the reliability requirement. After selecting a reliability level, the operation’s run time and the impact from the reliability mechanism’s overhead must be analyzed to ensure that the operation still executes properly. Examples of performance implications include changing the satellite’s state which has a quick run time (<5 msec) causing it to suffer a larger impact from the reliability method’s performance overhead compared to computing the Kalman filter gains with its significantly longer run time (>80 msec) which experiences minimal performance
impacts from the reliability method’s overhead. Thus the operations required reliability needs must be balanced with the reliability method’s impacts on performance.

Table 5.10: Implementation Matrix

<table>
<thead>
<tr>
<th>Performance Requirement</th>
<th>Reliability Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Low</td>
<td>1</td>
</tr>
<tr>
<td>Medium</td>
<td>2</td>
</tr>
<tr>
<td>High</td>
<td>0</td>
</tr>
</tbody>
</table>

The avionics architecture designed in this thesis provides the unique capability of adapting an operation’s reliability to meet performance requirements. This adaptability enables the system to impact only the critical functions with the negative performance overhead of increased reliability; instead of forcing every function to operate under these performance impacts. Furthermore, the reliability level of a function can be modified by a software update.

The optimal implementation of these reliability levels is currently more of an art form than a scientific process due to the variable nature of satellite functions. Each operation must be individually evaluated for optimal performance and reliability prior to final implementation. The generic guidelines for implementation have been presented to aid in the development of systems utilizing this avionics architecture.

5.6 Conclusion

From an analysis of space system’s avionics requirements and from the knowledge of the space environment’s interactions with electronics, the structure of a new, COTS-based avionics architecture has been devised. This architecture supports all the fundamental operations of a satellite, and ensures that the architecture can grow and develop in stride with the COTS technology it relies upon. Additionally, it is capable of mitigating many radiation induced errors using both software and hardware methods, which allows it to modify the software’s reliability implementation according to the operation’s needs.
The core architecture has a minimum of three, interconnected processors, which serve as hardware backups for each other. Additionally, they enable many of the software-based reliability operations through their ability to communicate with each other. Ideally, all peripherals can directly communicate with all the processors; however, as this is not always possible several alternative interfacing methods were discussed. These hardware design decisions set the architecture’s foundation upon which the software-based reliability mitigates SEE-induced errors.

Each software function has a unique mixture of reliability and performance that it requires. This architecture aims to create the flexibility of meeting each function’s needs to maximize the system’s resource utilization. The primary software techniques for increasing reliability are data encoding and redundant execution. The mixture of these elements incorporated into each function’s operations determines its level of reliability and the change in execution time it will experience. This combination of hardware and software design strategies constitutes the COTS-based avionics architecture designed in this thesis.
Chapter 6

Implementation of a Microcontroller Based Testbed

A testbed was designed in order to characterize and verify the reliability of the architecture developed in Chapter 5. The core of this testbed are three MCUs, which are interconnected via a Controller Area Network (CAN) bus. The rough design of this system for creating highly reliable outputs is shown in Figure 6-1.

![Figure 6-1: Testbed Design](image)

This testbed doubled as the avionics system for the MIT SSL’s Cathode Anode Satellite Thruster for Orbital Repositioning (CASTOR) satellite. This functionality of serving as a satellite’s avionics system aided in refining the architecture by ensuring it could command and control a satellite and its payload(s).
6.1 CASTOR Avionics System

The CASTOR satellite is an ESPA-class satellite, defined by having a volume <24.0 in (60.69 cm) x 20.0 in (50.80 cm) x 28.0 in (71.12 cm) and a mass <215 lb (97.52 kg) [44], being built by the MIT SSL as a part of the UNP’s 6th intercollegiate competition. The mission of CASTOR is to measure the on-orbit performance of a diverging cusped field thruster (DCFT), a new type of low complexity, throttle-able power consumption, electric thruster designed by the MIT Space Propulsion Laboratory (SPL). The simplicity of CASTOR’s mission along with its single payload which required minimal data processing made it an ideal satellite on which to implement the initial version of the COTS-based, multiprocessor architecture. An MCU-based system was chosen to minimize the system costs and complexity. Reliable operations will be achieved by having the redundant MCUs implement different forms of software-based fault-tolerance.

6.1.1 System Operation and Design

The CASTOR avionics system was designed according to the architectural design requirements and goals outlined earlier in this thesis (see Tables 5.4 and 5.5 for specifics). At its core are three MCUs, which communicate over a common bus. These processors provide hardware and software redundancy to mitigate SEE-induced errors. Each operation can be implemented with a different the type and level of redundancy depending upon the criticality of its outputs.

Basic Operating Strategy

The avionics’s main operating mode mimics Space Micro’s TTMR (discussed in Chapter 3) except for operating the processors in lockstep [25]. Two MCUs are operated as an unsynchronized redundant pair, both performing the same operations and compare the results with each other. The satellite’s housekeeping operations: telemetry data collection, system monitoring, and payload actuation are performed in this manner. If a discrepancy is noted, the third MCU also processes the operation, and the
committed result is determined by running a modified version of Paxos (a standard consensus algorithm for distributed systems) [13]. For actuating mission-critical devices, multiple MCUs must send independent actuation commands which are checked by fault-tolerant (RadHard) components (discussed in Chapter 5). Currently, these checking components are simply AND logic gates (see Figure 5-5), which can be purchased from Intersil. An IC with four, two-input AND gates capable of withstanding 100 krad (Si) costs $140 and the same device with 300 krad (Si) dosage tolerance can be purchased for $360 [37]. These RadHard components are viewed as acceptable additions to the COTS-based system as they are only used for mission-critical outputs and have a minimal impact on the system’s performance and cost.

During nominal operations, the third MCU is tasked with performing the high throughput operations of payload data collection and precision attitude control calculations. Periodic faults in these operations are acceptable as they will either be outliers that can be filtered out of the payload data or momentary erroneous control commands that will be mitigated in the next control cycle (at most 1 second later). Hence, minimal software reliability is employed in order to access the full performance potential of the processor.

**System Health Monitoring**

In addition to fulfilling these satellite operations, all the MCUs are continually monitoring each other to determine their health status. If one MCU detects that another MCU is failing, it power cycles the failing MCU. If the MCU does not begin operating properly, one healthy MCU takes on all the system tasks while the other healthy MCU begins reprograms the failing MCU. Each processor has the capability to program the other processors through a dedicated connection to the other processor’s in-circuit serial programming (ICSP) port. This programming operation is expedited by the fact that all three MCUs are running the same operating code, with the operation of independent peripherals assigned to specific MCUs. Each MCU has a predesignated ID, which it receives by reading an external input. Once the failing MCU is programmed, a quick check is performed to ensure that the failure has been
mitigated. Upon passing that check all MCUs return to their ideal operating state. If the MCU fails this check, it is permanently powered down. This process is similar to the operating paradigm in Figure 5-3.

**Hardware Interfaces**

The system has hardware interfaces for communicating with all of the satellite’s components. A complete list of devices and their interface can be seen in Table 6.1.

<table>
<thead>
<tr>
<th>System</th>
<th>Device</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avionics</td>
<td>SD Memory Card *</td>
<td>SPI, CAN</td>
</tr>
<tr>
<td></td>
<td>Inter-MCU Bus *</td>
<td></td>
</tr>
<tr>
<td>Structures</td>
<td>Solar Panel Release Mechanism</td>
<td>PWM Output</td>
</tr>
<tr>
<td></td>
<td>Thermal Sensors</td>
<td>Analog Input</td>
</tr>
<tr>
<td></td>
<td>Thermal Couples</td>
<td>Analog Input</td>
</tr>
<tr>
<td>Attitude Control and</td>
<td>Torque Coils (x3)</td>
<td>PWM Output</td>
</tr>
<tr>
<td>Determination</td>
<td>Reaction Wheels (x3) *</td>
<td>RS-485</td>
</tr>
<tr>
<td></td>
<td>Sun Sensors (x6) *</td>
<td>SPI</td>
</tr>
<tr>
<td></td>
<td>Inertial Measurement Unit *</td>
<td>SPI</td>
</tr>
<tr>
<td></td>
<td>Magnetometer (x2) *</td>
<td>SPI</td>
</tr>
<tr>
<td></td>
<td>Global Positioning System receiver</td>
<td>RS-422</td>
</tr>
<tr>
<td>Communications</td>
<td>Transceiver</td>
<td>USART</td>
</tr>
<tr>
<td>Power</td>
<td>Voltage Sensors (x5)</td>
<td>Analog Input</td>
</tr>
<tr>
<td></td>
<td>Current Sensors (x5)</td>
<td>Analog Input</td>
</tr>
<tr>
<td></td>
<td>DC/DC Power Converter (x2) *</td>
<td>SPI</td>
</tr>
<tr>
<td>Propulsion</td>
<td>Xenon Flow Controller (x2) *</td>
<td>SPI</td>
</tr>
<tr>
<td></td>
<td>Cathode Keeper Control</td>
<td>Digital Output</td>
</tr>
<tr>
<td></td>
<td>Cathode Heater Control</td>
<td>Digital Output</td>
</tr>
<tr>
<td></td>
<td>Anode Control</td>
<td>Digital Output</td>
</tr>
<tr>
<td></td>
<td>JPEG Camera</td>
<td>USART</td>
</tr>
</tbody>
</table>

An * indicates a bus device.
An o indicates that the interfaces uses an external IC to communicate over USART with the MCU.

### 6.1.2 Hardware Design

The hardware system is constructed around three MCUs, interconnected with a CAN bus in a flat network topology. The MCUs are Microchip dsPIC33FJ256GP710As (dsPIC33), which are fabricated using 250 nm process size transistors, operate at
3.3 V, and achieve a maximum throughput of 40 MIPS (well above the baseline requirement of 1.4 MIPS found in Table 4.1) with a clock speed of 80 MHz. A key advantage of the dsPIC33 is its hardware support for floating point multiplication and division, which is not common in MCUs.

The CAN bus’s physical communication layer is a half-duplex differential signal, with a maximum throughput rate of $1 \frac{Mbit}{sec}$ (Mbps). The MCUs transmit and receive CAN messages through a direct memory access (DMA) interface, which allows for the message processing to occur using a minimum number of CPU processing cycles. This channel is the primary inter-MCU communication medium, with a shared non-volatile memory array serving as the backup.

Each MCU has the following I/O functionality built-in: 12-bit analog to digital converters (ADC), SPI, USART, inter-integrated circuit (I$^2$C), pulse width modulation (PWM), and joint test action group (JTAG). In addition to these specific I/O modules, every pin on the dsPIC33 can be used for commanding digital signals as a general purpose input/output (GPIO). A full listing of the dsPIC33’s available interfaces is shown in Table 6.2.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-bit ADC</td>
<td>32</td>
</tr>
<tr>
<td>CAN</td>
<td>2</td>
</tr>
<tr>
<td>SPI</td>
<td>2</td>
</tr>
<tr>
<td>USART</td>
<td>2</td>
</tr>
<tr>
<td>I$^2$C</td>
<td>2</td>
</tr>
<tr>
<td>PWM</td>
<td>8</td>
</tr>
<tr>
<td>JTAG</td>
<td>1</td>
</tr>
<tr>
<td>ICSP</td>
<td>3</td>
</tr>
<tr>
<td>External Interrupt</td>
<td>24</td>
</tr>
<tr>
<td>GPIO</td>
<td>86</td>
</tr>
</tbody>
</table>

Each MCU was given a unique hardware identifier, used to indicate specific tasks it must execute. The MCU learns its identifier by reading two GPIO pins and stores this value in memory. This memory location is frequently updated by re-reading the pins to protect against memory upsets.
Processor Evaluation

The CASTOR avionics system requires sufficient performance capability to execute all the satellite’s functionality while guaranteeing reliable operations in the space environment. To verify that the processing power needed to support this requirement was available on the selected MCU, the execution time of one of the most computationally intensive operations was analyzed. The attitude determination and control system’s (ADCS) periodic control task, which takes in sensor measurements and the prior time step’s estimate of the satellite’s current attitude and applies a Kalman filter and a periodic control law to compute all the actuator commands, was selected. Properly timed execution of this task ensures that the satellite attitude can be stabilized in the desired orientation. CASTOR’s threshold rate requirement for performing the ADCS’s periodic control task is 1 Hz, with an objective requirement of 5 Hz. As this task is large and complex, it was reduced to its most computationally intensive calculation, calculating the Kalman filter gains [62], to determine if the dsPIC33’s performance capability was sufficient. A simple program was created to compute the Kalman filter gains from an existing state matrix. The test results were then compared to Matlab’s outputs with the same input (state matrix), to verify the accuracy of the calculations. The execution timing results based upon the data’s level of precision is shown in Table 6.3 (comparison with the Matlab results was removed for a more accurate timing simulation).

<table>
<thead>
<tr>
<th>Unit Type</th>
<th>Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit Fixed Point</td>
<td>71.89</td>
</tr>
<tr>
<td>32-bit Fixed Point</td>
<td>50.88</td>
</tr>
<tr>
<td>32-bit Floating Point</td>
<td>15.00</td>
</tr>
<tr>
<td>64-bit Fixed Point</td>
<td>10.66</td>
</tr>
<tr>
<td>64-bit Floating Point</td>
<td>7.68</td>
</tr>
</tbody>
</table>

From these tests, it was concluded that the dsPIC33 has the requisite performance capability to fulfill the CASTOR mission as it calculated the Kalman filter gains with a surplus of 69.8 msec of extra processing time per control cycle, using the highest precision values at the objective control rate of 5 Hz. A lower performance MCU
would not be as ideal because this surplus time is needed by the MCU for conducting the remaining operations, to include calculating the control law and commanding the control system’s actuators.

Operational flexibility is available by trading different data types and execution frequency. This flexibility is used by the CASTOR avionics system for graceful degradation. If the MCU performing the control calculations fails, those calculations can be delegated to other MCUs which by lowering the unit precision (i.e. from 64-bit floating point units to 32-bit fixed point units) are capable of meeting the timing constraints of performing all their old tasks in addition to the ADCS tasks that the failed MCU was performing. The system pointing accuracy is degraded from the reduced output precision, but the functionality remains despite what should have been a complete loss of functionality.

Data Storage

Outside of the MCUs, another major component of CASTOR’s avionics system is the external non-volatile memory array, which was implemented using a 2 gigabyte (GB) secure digital (SD) memory card. This memory card included built-in ECC, hardware implemented SECDED data protection. With 2 GB of data space, the satellite can collect data for over a week without downlinking information to the ground station prior to filling the available memory space. This reduces the risk of losing data from overwriting a memory location when the available data space is full. As CASTOR’s concept of operations (CONOPS) calls for daily communication periods where data is downlinked, the memory size was deemed sufficient.

System Interfaces

CASTOR has demanding interface requirements. Using Table 6.1 to total the number of required interfaces, and Table 6.2, to calculate the number of available interfaces per processor the required system interfaces was calculated. The compilation of this data for the entire avionics system, demonstrating the number of interfaces required and the number available, is shown in Table 6.4.
Table 6.4: CASTOR Avionics Interfaces

<table>
<thead>
<tr>
<th>Interface</th>
<th>Required (Used)</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC Input</td>
<td>42</td>
<td>96</td>
</tr>
<tr>
<td>CAN</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>SPI</td>
<td>1 (6)</td>
<td>6</td>
</tr>
<tr>
<td>USART</td>
<td>5 (7)</td>
<td>6</td>
</tr>
<tr>
<td>I2C</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>PWM</td>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>JTAG</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ICSP</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>External Interrupt</td>
<td>0</td>
<td>72</td>
</tr>
<tr>
<td>GPIO</td>
<td>66</td>
<td>258</td>
</tr>
<tr>
<td><strong>Total Pins</strong></td>
<td><strong>268</strong></td>
<td><strong>300</strong></td>
</tr>
</tbody>
</table>

The amount used is not explicitly listed if it equals the number required.

From Table 6.4, one notices that the avionics system used more SPI and USART communication ports than required. As noted in Table 6.1, the RS-485 communication creates a bus architecture that can communicate with the MCU over USART. RS-485 supports up to 32 different devices on the bus, CASTOR has only nine. As the reaction wheels are critical to proper satellite functionality and they communicate over RS-485, it was decided to connect all three MCUs to the RS-485 bus. Thereby allowing any of the MCUs to interact with the devices on the RS-485 bus (making all nine RS-485 devices shared peripherals) while also creating another interprocessor communication mechanism. This setup ensures that all the processors must fail before the reaction wheels cannot be commanded.

By connecting the RS-485 bus to each MCU it increased the number of USART ports required beyond the combined total available from the three MCUs. This problem was fixed by placing a serial communication converter IC between one of the USART devices (the JPEG Camera) and the MCU. The IC allowed for the camera to be connected to the MCU’s SPI bus, for the overhead of a 16 pin IC, an oscillator, and two passive components (the schematic can be seen in Appendix A).

All of CASTOR’s SPI devices (eight SPI devices plus the converted camera signals, for a total of nine) could be setup as independent peripherals (connected to a common bus) and thus be connected to one SPI port on a single MCU (or have the bus
arbitrated between all the MCUs). However, this design would leave a single point of failure, because if that MCU (or arbitration circuitry) were to fail all the devices connected to it would be lost. For this reason, all the available SPI ports were utilized and the devices were spread across the available MCUs to minimize the impact of an MCU failure. Among the SPI devices, the SD memory card needs to be connected as a shared peripheral so that all three MCUs are capable of writing and reading this shared data. Additionally, the SD card has a high latency and is thus not an ideal device to have on the same bus with time-sensitive sensors and actuators, such as the inertial measurement unit (IMU) which make high frequency (10 Hz) measurements which are integrated to propagate the state estimate. For this reason, each MCU has a dedicated SPI port connected to the SD card. To ensure that only one MCU can drive the bus at a given time, a bus arbitration circuit, requiring two pins per MCU and four ICs, was placed between the memory and the MCUs; the schematic is shown in Appendix A. The CASTOR avionics system interfaces are shown in Figure 6-2.

This approach of connecting a few devices to each MCU was done throughout the system to ensure gracefully degradation, as a single MCU failure would not remove
significant functionality, but simply impair the accuracy and speed of operations. Other components this technique was applied to include the PWM outputs for the torque coils (where each MCUs is connected to a different axis’s torque coil) and the analog sensor inputs (which are divided evenly between two MCUs). In a related manner, all the DCFT’s support system control signals (the heater and keeper power converters’s enable signals) can be generated from multiple MCUs. The signals are connected to the devices by an OR gate. This setup keeps the functionality of turning the power converter on/off available in the midst of an MCU failure. In the case of operating the anode converter (and in effect the thruster), which draws >100 W giving it the potential to drain the satellite’s batteries and thereby cause other systems on the satellite to fail, an AND gate is used to require two separate MCUs to send agreeing signals to operate the thruster. The complete schematics and printed circuit board (PCB) layouts for the CASTOR avionics system, shown fully assembled in Figure 6-3, are in Appendix A, along with detailed information about the CASTOR hardware design.

Figure 6-3: CASTOR Avionics Board
6.1.3 Software Design

The CASTOR avionics system relies heavily on its hardware features for providing a foundation for software-based fault-tolerance algorithms which ensure reliable operations. Additionally, the CASTOR avionics must provide a hard real-time environment for the ADCS task to function properly. Expanding beyond the ADCS task’s timing requirements, Figure 6-4 shows a high level timing diagram of CASTOR’s software priorities. The design goals of reliability and performance are often at odds with each other as many of the reliability operations increase the required processing time by adding redundant executions and comparison operations while many real-time constraints desire short execution times in order to guarantee operations occurs within their specified timing window.

![Figure 6-4: CASTOR Software Timing Diagram](image)

The software development for the dsPIC33 is performed in Microchip’s MPLAB Integrated Development Environment (IDE), which encompasses the C compiler, assembler, linker, and loader tools which are all based upon the GNU compiler collection (GCC) toolchain. The GCC tool chain is modified to target the executable code towards the 16 bit dsPIC processor’s 81 operations instead of the 245+ x86 instructions.
RTOS

To enforce the necessary timing constraints, a real-time operating system (RTOS) is used. It schedules tasks to execute in order of their priority, at the exact rate specified. Without this guarantee, integrated sensor measurements could incur timing inaccuracies that lead to incorrect results. The RTOS used by CASTOR is Express Logic’s ThreadX, which is a microkernel operating system (OS), meaning its structure is highly compartmentalized so that only the needed functionality is present in the operational code. This greatly minimizes the overhead in the OS and allows ThreadX to achieve the high performance characteristics shown in Table 6.5. Additionally, ThreadX has flight heritage from NASA’s Deep Impact space probe and Mars Reconnaissance Orbiter satellite.

<table>
<thead>
<tr>
<th>Table 6.5: ThreadX Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ThreadX Characteristics</strong></td>
</tr>
<tr>
<td>Minimal Kernel Size</td>
</tr>
<tr>
<td>Minimal RAM Required</td>
</tr>
<tr>
<td>Boot Time</td>
</tr>
<tr>
<td>Context Switch Time</td>
</tr>
</tbody>
</table>

CAN Driver

Above the ThreadX kernel are several user-level software applications. One grouping is the interface drivers for interacting with the devices connected to the MCU (reference Table 6.1 for a complete list of devices). The interface most integral to the system’s reliability is the CAN bus, whose message format is shown in Figure 6-5.

Figure 6-5: CAN Message Format [34]
CASTOR utilizes the 29-bit identifier in the arbitration field to encode the sending and receiving MCU identifiers, leaving the entire eight byte data field available for message data. The first byte (8 bits) is used to encode the message number, which is used to ensure an at-most-once (AMO) message passing semantics. Sending messages in an AMO manner ensures that the data either arrives at its destination or is never sent, ensuring that lost or accidentally retransmitted messages do not cause unwanted events to occur. The second byte (8 bits) encodes the packet type which informs the receiving MCU how to process the remaining six data bytes (48 bits). The major message types are enumerated in Table 6.6. This small data size is extremely limiting, for example it is unable to pass a double precision floating point number in a single message, thus several types of message designate large data types. Other message types are for performing Paxos (a consensus algorithm) and for monitoring other MCU’s health.

Table 6.6: Primary CAN Message Types

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>0x01</td>
</tr>
<tr>
<td>Transmit Int Array</td>
<td>0x08</td>
</tr>
<tr>
<td>Transmit Long Array</td>
<td>0x36</td>
</tr>
<tr>
<td>Transmit Float Array</td>
<td>0x51</td>
</tr>
<tr>
<td>Transmit Double Array</td>
<td>0x7A</td>
</tr>
<tr>
<td>Paxos Propose</td>
<td>0x80</td>
</tr>
<tr>
<td>Paxos Proposal Return</td>
<td>0x84</td>
</tr>
<tr>
<td>Paxos Accept</td>
<td>0x8F</td>
</tr>
<tr>
<td>Paxos Accept Return</td>
<td>0x9C</td>
</tr>
<tr>
<td>Redundant Execution</td>
<td>0xAA</td>
</tr>
<tr>
<td>MCU Restart</td>
<td>0xED</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>0xFF</td>
</tr>
</tbody>
</table>

Data Storage Driver

Critical to satellite operations is the MCU’s interface with the non-volatile memory (SD card). This memory is primarily written to by applications requiring permanent data storage. The only time when values are read from the memory is prior to the data being down-linked to the ground station. Storing data in non-volatile memory occurs
through the following process: the data is created in the MCU’s RAM buffer, written to the MCU’s non-volatile FLASH memory, and then sent to the external SD card. Sending data to the SD card occurs as soon as the SD card is available after either a periodic software interrupt triggers the transmission or when the MCU’s FLASH memory fills to $\geq 75\%$ of its data capacity (set at 3.75 KB). This data path is used to batch groups of writes to the external memory, thereby reducing contention between different MCUs trying to access the SD card. It was modeled after the technique of journaling in Linux’s ext3 file system [57].

When data is saved to the MCU’s FLASH memory, it is stored in 128 byte segments, because this is smallest amount of data that can be written to the MCU’s FLASH memory. Each write incurs approximately a four millisecond overhead between the command being initiated and the logic being set. When writing to the external SD card, the timing is nondeterministic as the MCU must wait to be granted access to the memory bus. Ideally this is a negligible time delay, but could require up to 150 millisecond if the other processors must first commit data to the SD card. Once the MCU has access, it can store the data in 512 byte segments on the SD card, each of which has approximately a four millisecond delay between the data arriving and it being stored. The SD card acknowledges successful storage by sending the MCU the number of bytes written.

Data is stored on the SD card in the American standard code for information interchange (ASCII) data format. While this is a suboptimal format for minimizing the data storage utilized and it increases the number bits in danger of being upset, this format simplifies processing the satellite data. There are two categories of information stored on CASTOR. The first category is housekeeping data from sensor measurements such as the battery voltage and the solar panel temperature. This data is stored as ASCII text in a comma-separated value (CSV) file. The second category is JPEG encoded image data. Each data category has a separate directory on the SD card, where the filenames denote the date of recording. Upon successfully downloading a file to the ground station, a command to erase that file is transmitted to free space on the SD card.
As previously mentioned, the SD card is a backup communication channel between MCUs. A primary challenge of communicating this way is triggering the receiving MCU to read the data. Each MCU performs a periodic check of its buffer on the SD card (every 60 seconds). If data is present, the MCU determines that the CAN bus is faulty and transitions to using the memory bus for inter-MCU communication. At which point the frequency of checking the buffer is increased to every 250 milliseconds.

**Other Interface Drivers**

The remaining drivers facilitate communication with devices, performing functions such as commanding actuators and reading sensors. These interfaces typically consist of a protocol initialization to setup the communication standards for the interface, some also enable separate devices to communicate over the same physical bus with different operational characteristics (e.g. CASTOR’s magnetometer utilizes SPI with an idle low clock state while the IMU requires a high idle clock state). Beyond setting up the communication protocol, the driver contains commands for interacting with the device. These commands are device specific but can be thought of as `put();` or `get();` commands where `put();` commands the actuator to perform an operation and `get();` returns a sensor reading.

**Fault-tolerance Algorithms**

Another grouping of software operations are the fault-tolerance algorithms. Different fault-tolerance algorithms are used based upon the timing and reliability requirements of the function/data. These algorithms take two primary forms (as seen in Chapter 3), repeated execution on a single processor and single execution on multiple processors.

Algorithms requiring a single processor are called by wrapping the function to be checked in the fault-tolerant algorithm, in its simplest form, this is done by placing the function as an argument to the fault-tolerant algorithm. Placing the fault-tolerant algorithms into the software architecture aids in modifying the reliability level of a function. The single processor methodologies mentioned in Chapter 3 that can be applied using this framework are redundant execution, RPR, and AN codes. Run-
ning redundant virtual machines would require implementing a virtualization layer below the kernel; while this does pose many advantages, this methodology was not implemented due to a lack of virtual machine software available for embedded 16-bit processors. ISC is encoded at compile time and the time required to incorporate it into the MPLAB IDE was not available. Finally, SECDED encoding and memory scrubbing can be performed on specific buffers/data regions; however, it was left out of the list of single processor reliability options because it is independent of any function being executed.

Algorithms requiring execution on multiple processors must pass data over the CAN bus. Specific CAN messages indicate to a receiving MCU if the data is an input or output. The reliance on other processors forces the system's processing time to be indeterminate. In the event of this indeterminism causing a timing error, the processor will utilize the value it computed internally without checking the results with other MCUs. To minimize the possibility of corrupt data being transferred between processors, which would create a false detected unrecoverable error (DUE). The data transmitted over the CAN bus can be error encoded.

Upon receiving a remote procedure call (RPC) to redundantly execute a function and transmit the results to the requesting MCU for error checking, the MCU must first ensure it has all the input data. Collecting the input data could require several messages as only six bytes (48 bits) can be transmitted in each packet, and SECDED encoding adds an overhead of 100% (four encoding bits for every four data bits) leaving three bytes (24 bits) of data that can be transmitted in each message. Once all the data has been received, if it was encoded it must be decoded, and if uncorrectable errors are detected that data value must be requested again. The processor can then execute the function and prepare the results for transmission to the originating MCU.

An MCU initiating an RPC must wait for the results from the other MCUs prior to committing an output. Once all the results are obtained, the processor compares the results to determine if an error occurred. Correction requires all three MCUs; while two can be used for detection. If an error is found, one of the MCUs without an error begins working to correct the faulty MCU. At the recognition of the first error,
the properly operating MCU simply notes which MCU was in error. Upon recognizing a second error within a time period of 100 seconds, the faulting MCU is restarted. If another error is detected within 100 seconds of the reset that MCU is reprogrammed. The lack of corrective actions take after the initial detection is based upon the high probability that an upset is within a data buffer which will be overwritten shortly and thus naturally stop creating errors. The scaling of corrective actions is used to minimize the time spent fixing faulty MCUs.

In addition to the redundancy algorithms there are reliability checks in the satellite’s operating code, including the periodic control algorithms and the housekeeping logic. In the housekeeping logic are functions which initiate corrective actions for the satellite when a sensor measurement begins traveling outside an expected bound. These corrective actions generally require the satellite to enter a safe-mode and stop performing its primary mission, and thus should not be haphazardly entered into. For this reason, sequential sensor measurements must be outside of the anticipated range before corrective actions are begun. Additionally, to mitigate potential errors from minor sensor discrepancies the comparisons use an RPR-based comparison scheme where only the upper half of the data is compared.

This concludes the major software design aspects for the CASTOR avionics. A plethora of other software applications were implemented; however an in-depth analysis of these is outside the scope of this thesis.

6.1.4 Key Aspects for Achieving Radiation Tolerance

Redundancy, in many forms, is the primary method for ensuring reliable operations on CASTOR. The hardware has redundant devices; the software is stored in redundant locations and executes operations redundantly. The radiation tolerance of CASTOR and this COTS-based architecture is achieved through the coupling of hardware and software redundancy techniques to minimize the system’s costs while maintaining mission functionality.
Hardware Redundancy

In the hardware setup, three identical MCUs are used. At a minimum, this redundancy allows for two processors to fail prior to losing system functionality. Beyond this basic provision of redundancy, critical satellite operations can be controlled by multiple MCUs, ensuring operability after an MCU failure. An example of this is the 5 V power converter’s active high, enable signal, which can be commanded by two MCUs; each MCU’s control signal is tied to a pull down resistor and transmitted to the device through an OR gate (reference Figure 5-4). This ensures that either MCU can enable the 5 V converter, it cannot be inadvertently disabled by an MCU failing to maintain the active high signal, and in the event that both processors fail the power converter will be disabled saving power.

For mission-critical operations, where an erroneous input could result in harm to the satellite’s health, outputs from multiple MCUs have been connected to AND gates prior to the device (reference Figure 5-5); NAND gates could be used for active low outputs. This only lets an output be expressed if a majority of the results from separate processors match. This forces an active high output to rely on multiple processors achieving consensus amongst their outputs before the device receives a command. For reliability purposes, these AND/NAND gates should be RadHard components, and by keeping the number of these RadHard checking components \( \leq 200 \) the system costs remains below that of a single RadHard processor. On CASTOR, two were employed for creating the signals which command the DCFT.

Another utilization of the redundant MCUs was to distribute devices across the system, so that each processor was responsible for different device interactions. In particular, independent peripherals which could only be connected to one MCU, a few of these devices were connected to each MCU. When a family of devices (i.e. X, Y, and Z axis torque coils on CASTOR) fits this criteria, they were dispersed across the available MCUs. This ensured that if one MCU failed, it did not remove an entire satellite functionality (e.g. ability to operate torque coils) but only a subset of the functionality (e.g. ability to create magnetic torques along a single axis). Ideally, the
system or satellite operators would then be able to detect this failure and compensate for the partial loss of functionality.

A large portion of this architecture’s software reliability requires inter-processor communication, thus hardware redundancy was built into the inter-processor communication. The primary inter-processor bus is the CAN bus, the secondary is the SD card, and the final is the RS-485 bus which primarily communicates with ACDS components (reaction wheels and sun sensors). Thus multiple communication links must fail prior to the MCUs losing their ability to communicate with each other.

A final area of hardware radiation tolerance comes from the usage of bus based communication with devices. The primary buses utilized are RS-485 and SPI. The RS-485 bus is connected to each processor, thereby enabling all the processors to communicate with any of the RS-485 devices (see Table 6.1 for a complete listing) and ensuring communication with the devices as long as an MCU is operational. This same approach could have been utilized with the SPI bus through the addition of another bus arbitrator and multiplexer circuit. However, this approach was not utilized for two reasons: performance and simplicity. Giving each processor a separate SPI bus allows for simultaneous operation of multiple devices and thus higher system performance. Reducing the required circuitry by only having the devices connect to a single MCU simplified the design and reduced the number of components that could potentially fail. For example, CASTOR has two magnetometers that communicate over SPI. These magnetometers were connected to separate MCU’s SPI buses to ensure that magnetometer measurements could be taken after an MCU failed.

The SD card also employs hardware error correction on data stored on the memory card. This hardware checking is enacted every time the data is read, thus by periodically reading the data it will be scrubbed for errors. This further level of error protection increases the reliability of the external memory and thus the system.

Hardware redundancy was built into the architecture to form a backbone for radiation tolerance because radiation effects are spatially diverse. The probability of any transistor node experiencing an upset is $\sim2\times10^{-5}$ per day [60]. Therefore, the system’s survivability is increased by having devices physically separated, by spreading
the system’s functionality across a larger area it reduces the probability of an upset simultaneously impacting two separate devices.

Software

Hardware redundancy is the foundation upon which the software creates radiation tolerance, as even the most robust software will fail if the hardware is not operational. As opposed to traditional systems, which primarily rely upon hardware mechanisms, this architecture uses software mechanisms as the primary source of reliability. This software-based reliability takes several different forms depending upon the desired level of robustness needed by the function.

• MCU liveness

Software can be utilized to verify the liveness of each processor. This is accomplished by passing messages between the processors over the CAN bus. These messages, referred to as heartbeats, signal liveness, much like a person’s heartbeat. Heartbeat requests are sent periodically (every 30 seconds) from one MCU to the other MCUs on the network. Upon receiving a heartbeat request, an MCU will reply back with a heartbeat acknowledgment, thereby demonstrating its liveness.

If a heartbeat acknowledgment is not received the following corrective actions are taken. First, the message will be resent in case the packet was lost or corrupted in transmission. Next, other live MCUs will be queried if they have recently received a heartbeat acknowledgment from the unresponsive MCU. Upon finding no live MCU that has recently communicated with the unresponsive MCU, the live MCUs will select one MCU to attempt bringing the non-responsive MCU back online. The chosen MCU will perform the corrective actions discussed previously, resulting in either the non-responsive MCU coming online or being deemed no longer operational.

The heartbeating process is continually operating on the MCUs. If the CAN bus fails, the MCUs continue sending heartbeats over the memory bus and the
heartbeat rate degrades to once every 90 seconds. In addition to determining liveness, the heartbeat also allows for processors to perform a rudimentary health checks by determining if an MCU has recently been restarted. This is ascertained by passing a heartbeat number, all the MCUs will initially have the same heartbeat number. However, if an MCU is restarted, its heartbeat number will be re-initialized. Thus when it receives a heartbeat request with a number greater than its current one it determines that it fell behind (most likely from restarting) and will reply with its current heartbeat number (to inform the other MCUs it fell behind), update its heartbeat number to the value sent, and write its log that it fell behind on the heartbeat count.

- **Reprogramming**

The processors are capable of reprogramming each other through ICSP. The ICSP network between the MCUs has a ring topology. With only three processors this creates a flat network, where all the processors are interconnected. Connecting additional processors changes this paradigm so that a processor would only be able to reprogram the two processors immediately adjacent to it on the ICSP ring, or additional GPIO pins would need to be allocated to ICSP.

ICSP is Microchip’s proprietary standard for programming MCUs, it is a two wire serial interface for directly load executable hex files on the MCU’s non-volatile program memory. A PIC processor enters ICSP programming mode after the reset pin (MCLR) is toggled and a 32-bit key, 0x4C434851, is transmitted to the ICSP data port. Once in ICSP programming mode, commands and data can be sent to the processor at a maximum rate of 5 Mbps. For reprogramming, the entire program memory must first be erased (by setting all data values to ‘1’). This clearing operation takes ~330 msec. Once the memory is cleared, it can be programmed one row (192 bytes, 64 instructions) at a time. Ideally, one row of data can be programmed in ~3.4 msec, and CASTOR’s entire 256 KB of program memory (requiring 1,366 row writes) in ~4.67 seconds. The final step is to program the MCU’s 12 configuration registers which control
key settings such as enabling the watchdog timer and selecting the oscillator. This requires \( \sim 16 \) msec. Once all the required programming is completed, the failed MCU is allowed to operate regularly.

- **Redundant Storage**

The memory arrays constitute the largest area of transistor nodes in the MCU (reference Figure 2-12) and therefore have the highest probability of experiencing an upset. In order to mitigate inevitable upsets in the program memory, redundant storage is employed. The executable code is redundantly stored on the MCU’s FLASH memory and on the external SD card. Ideally, these redundant copies are SECDED encoded (through software encoding). If an MCU notices it is malfunctioning, it can attempt to reprogram itself by decoding its redundant copy and reprogram itself. If the SECDED decoding slows reprogramming enough to impact the satellite functionality, an operational MCU can use its executing program memory (the primary copy, without SECDED encoding) for reprogramming a failed MCU. Finally, an additional redundant copy is stored on the SD card providing another layer of redundancy. Having multiple layers of redundancy allows for double bit errors to be corrected despite the encoding only fixing single bit errors. Upon detecting a double bit error, the processor can query a redundant copy for its value. After decoding the data to check for errors the error free value replaces the incorrect one.

Redundant copies of the program memory limits the size of the operational code. The dsPIC33 on CASTOR has 256 KB of non-volatile memory, thus to store a SECDED encoded copy, the operating code (including the temporary buffers for writing to the SD card) cannot exceed 85 KB. The current version of the CASTOR operating code is 89.6 KB. If the code is not SECDED encoded the memory could support a 128 KB program (an increase of 51% in operating code length), which is the method currently employed by CASTOR. The time required to reprogram another processor without SECDED decoding the data is 1.92 sec for an 85 KB program and 2.71 sec for a 128 KB program.
• Fault-tolerant Algorithms

The major source of software-based reliability comes from applying the reliability algorithms in Chapter 3. These algorithms employ varying levels of redundancy. In redundant execution on a single processor, an operation is performed multiple times and the results from each execution are compared, equal results from a majority of the executions are considered to be the correct result. To increase the robustness of the redundancy, a stronger the algorithm must be employed. For example merely executing a function multiple times will enable detection of upsets that occur within the computational chain while leaving it blind to permanent hardware errors (e.g. the ALU being permanently set to perform XOR). By replicating the inputs and data variables the algorithm is able to detect errors in these data buffer’s values. Using separate hardware units and comparing their results gains the ability to detect these failures.

A couple modifications to the standard comparison of redundant executions exist. The first is RPR, where instead of checking to ensure that the results are exactly identical, it compares a portion of the result (generally disregarding eight or sixteen of the LSBs). This is ideal for applications such as sensor measurements where different sensors or timing intervals between sampling could create minor deviations but not significant fluctuations. In this situation, RPR makes comparing the data simpler as these minor differences are disregarded. Control calculations can be formulated to operate properly with the reduced level of I/O precision by simulating this precision reduction as sensor noise.

Another modification is using AN codes. AN codes modify the inputs of replicated functions by applying a scaling factor (optimal scaling was found to be achieved by using a scaling factor of -2 [47]). The results are then compared to verify that they differ by the scaling factor. This technique can be used on a single processor to determine if a functional unit is operating properly. However, AN codes cannot be applied to all functions. Some functions will generate errors when their inputs are scaled due to exceeding data type constraints and
other issues. Alternatively, some function are purely logical operations, and the results cannot be guaranteed to generate an accurate result when scaled (e.g. comparison operations with a constant). AN codes are primarily beneficial for checking mathematical operations.

The reliability of the output is increased by comparing results produced by different physical devices. This eliminates the chance of a functional unit (e.g. multiplication unit) consistently creating the same error (e.g. always off by a factor of 2), which would not be detected by most of the single processor methods discussed. Additionally, the replication of data into memory buffers across the system increases robustness, as the separate processors make it impossible for a single upset to change all the redundant values. Thus the utilization of multiple processors greatly increases the reliability of the results.

6.2 Software Fault-tolerance Algorithm Implementation

Examples of implementing fault-tolerance algorithms on the CASTOR satellite are discussed to clarify the process used for determining the reliability level to implement. The implementation of reliability algorithms for the ADCS and payload operations functions are discussed below.

6.2.1 Attitude Control Operations

The ADCS operations perform a wide array of tasks to ensure that the satellite’s body is oriented in the desired direction. These tasks can be grouped into the following categories, acquiring and processing sensor data, calculating the state estimate and control law commands, and commanding actuators. Each of these categories has different reliability and performance requirements, which are listed in Table 6.7. Different reliability levels will be implemented on each task to achieve an optimal utilization of the system’s resources.
The performance requirements for the ADCS operations are medium to high because the operations must be executed at a frequency of 1 Hz or greater, placing the burden on completing large computations (such as the state estimation and control law processing) inside this one second window while leaving time for the other tasks to execute. The reliability requirements are low to medium because if an actuator output command is incorrect, the effect will only impact the system for that control cycle (≤1 second), causing a negligible change in the satellite’s attitude. The state estimate has a lower reliability requirement because it is an estimate, implying uncertainty in the accuracy of the data, which makes adding precautions to ensure its exactness a waste of effort.

The initial reliability ranges were selected based upon the operation’s reliability requirements. Reliability levels zero through two were selected for the state estimation task, while levels three through five were selected for the other ADCS tasks. Further down selection was based upon the performance requirements of the operations and resulted in selecting the reliability levels shown in Table 6.8.

Both the actuator commanding and sensor data acquisition and processing were implemented at reliability level 4 (SECDED encoding of redundant execution on the same processor). This methodology ensures that when the data/commands are stored in memory, bit errors in those values can be overcome, and that commands are checked
within the processor prior to being sent to the device. This checking is performed on the same processor because the overhead of utilizing multiple processors creates a large drop in performance for these short run time operations (an order of magnitude increase in processing time).

As the state estimation assumes a certain level of error, adding reliability mechanisms to mitigate additional errors was viewed as frivolous. Thus maximum performance was sought by using reliability level 0 (no reliability mechanisms). On the other hand, error mitigation is desired for the control law calculation. Thus reliability level 3 (redundant execution on separate processors) was implemented because execution on separate processors allows for the highest performance of the functions as the run time is large enough to overcome the message passing overhead.

For the implementation of the ADCS operations an emphasis was placed on balancing the performance and reliability requirements to achieve an optimal utilization of processor resources. Thus the full spectrum of reliability mechanisms were employed on the same core group of satellite functions, illustrating the robustness and adaptability of the COTS-based, multiple processor architecture.

6.2.2 Payload Operations

The payload of the CASTOR satellite is the DCFT and its associated performance measurement components. The DCFT is an electric propulsion thruster set to operate at \(\sim 100 \, \text{W} \), which is \(\sim 61\% \) of the total power generated by the satellite’s solar arrays. For the satellite to remain power positive, the thruster can only be fired for a limited portion of the orbit (shown in Figure 6-6 labeled “firing angle”). The DCFT requires that the flow rate of Xenon (Xe) be periodically controlled and that the thruster’s performance be measured. The flow rate is controlled to ensure that the thruster does not consumed more than the budgeted 100 W (as the thruster can operate over a power spectrum of 40 - 300 W). As this is the first on-orbit operation of the DCFT there is a need for operational performance data. This data will primarily be gathered by photographing the thruster in operation.
The baseline DCFT tasks are listed in Table 6.9 along with their performance and reliability requirements. The DCFT’s requirements cover a wide spectrum, from the high-reliability and low-performance of commanding the thruster to the low-reliability and low-performance of collecting the thruster’s performance data. Commanding the thruster requires extreme reliability to ensure the thruster only operates when commanded and the satellite can support it, yet its performance requirements are minimal as it is only executed twice per orbit (∼90 minutes for a 500 km circular orbit), making an extra second of processing time a negligible addition. Controlling the flow rate of Xe is similar to the attitude control task, requiring moderate reliability and performance. However, an incorrect command could create a large power draw which coupled with the longer control period of 20 seconds (0.33 Hz) makes mitigating errors important for controlling the power consumption. Additionally, the function must execute within the control period window, creating a moderate performance requirement. Finally, an average of two pictures need to be collected per orbit while the thruster is operating. There is no impact on the spacecraft mission if this data becomes corrupted, only a loss of an operational data point (in a worst case situation). Additionally, the data collection function has no timing constraints beyond occurring while the thruster is operating (∼20 minute window), making it have a low performance requirement.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Performance</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCFT operation commanding</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>DCFT flow rate control</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>DCFT performance data collection</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>
Using the reliability requirements in Table 6.9 the following groupings of reliability levels were selected, levels six through seven for commanding the DCFT to operate, levels three through five for controlling the Xe flow rate, and levels zero through two for collecting the DCFT performance data. These groupings were reduced to the reliability levels specified in Table 6.10 after analyzing the performance requirements of each operation. Improperly operating the DCFT poses serious mission implications, thus levels 7 (SECDED encoding of redundant execution on the same processor compared between separate processors) and 8 (hardware voted redundant outputs) were selected for maximum reliability. This implementation utilizes all the available hardware and software resources available from the architecture. The commands are encoded to protect against errors in the storage, the functions are executed multiple times to verify the proper outcome and each processor compares its output with the other processors to eliminate the chance of functional unit errors contaminating the results on a single processor. After achieving consensus between all the processors, multiple processors must send activation signals to the DCFT for it to operate, ensuring that a single faulty processors cannot operate the DCFT. While this limits the DCFT to only being operational if multiple processors are functional, a failed processor implies that there is a larger error that needs to be attended to first.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Reliability Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCFT operation commanding</td>
<td>7 and 8</td>
</tr>
<tr>
<td>DCFT flow rate control</td>
<td>5</td>
</tr>
<tr>
<td>DCFT performance data collection</td>
<td>0</td>
</tr>
</tbody>
</table>

Controlling the DCFT’s flow rate of Xe is the operation with the highest performance requirement, yet it is still very minimal when compared to other system’s tasks (such as those in the attitude control system). Output errors pose potential side effects to healthy satellite operations. This task shares many similarities with the attitude controls system’s control law processing task, with a lower performance requirement and greater dangers from a lack of reliability. Thus the reliability level of 5 (SECDED encoding of redundant execution on separate processors) was selected.
The SECDED encoding was added to aid in minimizing the potential of errors occurring. Finally, the DCFT performance data collection was given a reliability level of 0 (no reliability mechanisms) because the loss of data due to errors is acceptable. In this situation, the performance requirements were not considered in the reliability level selection because there was no need for increased reliability.

The remaining satellite functions’s reliability levels were implemented using the same approach, where the reliability requirements were used as the first criterion for selecting a grouping of reliability mechanisms. The selected groupings are then downselected based on the performance constraints of the function, with the final selection being based upon test results to ensure that both the performance and the reliability requirements were met without any ill effects.

### 6.3 Conclusion

The CASTOR avionics design implements the design requirements and goals of the COTS-based avionics architecture outlined in Chapter 5 as well as the CASTOR avionics system requirements. The CASTOR design illustrates how many of the key features of the COTS-based architecture can be employed to mitigate errors, from the utilization of two checking MCUs, augmented with a high performance MCU, to the continual MCU liveness checking via heartbeating. This hardware platform verified many of the claims about this new, COTS-based avionics architecture. The software-based fault tolerance impacted the system’s performance by allowing operations a flexibility in their reliability level. Finally, the CASTOR avionics system proved the viability of the MCU-based avionics architecture for use in spacecraft designs.
Chapter 7

CASTOR Reliability Testing

The CASTOR avionics system was designed to be a proof of concept implementation of the COTS-based, multiprocessor architecture design outlined in Chapter 5 which guarantees reliable operations in the space environment by mitigating SEE-induced errors. Upon completing assembly of the hardware testbed, the software based fault-tolerance mechanisms were tested in order to determine the system’s ability to detect and correct errors. This initial testing was conducted using software fault-injection to create errors in specific regions of the processor. The results of this testing were analyzed to determine the characteristics of the various error detection and correction methods implemented by the architecture.

7.1 Testing Methodology

Prior to an avionics architecture being accepted by the spacecraft industry it must undergo reliability testing via high-energy particle bombardment (hardware-based fault-injection) using a cyclotron or linear accelerator facility to test and characterize the system’s responses to radiation-induced errors. Several factors prohibited this testing from being conducted on the CASTOR avionics system. First, an emphasis of this initial testing was to determine the design’s robustness. Thus having control over the type and location of errors through software was preferred over the randomness inherent with hardware-based testing. Second, the average maximum area which can
be radiated in a testing facility is a one inch diameter circle, 0.785 in\(^2\) (5.07 cm\(^2\)). This small area makes it impossible to irradiate all three MCUs simultaneously because each has an area of 0.397 in\(^2\) (2.56 cm\(^2\)). Some facilities can irradiate larger areas, up to 3.24 in\(^2\) (20.9 cm\(^2\)); however, a reliability measure of the CASTOR design was to physically separate the processors for fault isolation, resulting in each MCU having a minimum separation of 1.25 in (3.18 cm). Future testing focused iterations of this system could be adapted to collocate all three MCUs within the required area as well as utilize smaller package size MCUs. The dsPIC33 comes in a quad flat no leads package (QFN) which consumes 31.7\% of the current area usage or 0.126 in\(^2\) (0.81 cm\(^2\)), potentially allowing up to 6 MCUs to fit within the irradiated area (0.785 in\(^2\), 5.07 cm\(^2\)). Finally, the time and funding for this testing was not available.

Software-based fault-injection was viewed as the ideal test method for this architecture because it allowed for high granularity of error discrimination in testing on the actual hardware and kept the costs minimal. As this is the first implementation of the architecture, this testing methodology was seen as ideal because it generates needed feedback on the architecture’s strengths and weaknesses to faults in specific regions of the processor.

### 7.2 Reliability Testing Results

Using software-based fault-injection, a series of reliability tests were performed on the CASTOR avionics system. These tests were designed to test both the robustness and the performance implications associated with the reliability algorithms/methods implemented. This data was fed back into the design of the reliability methods employed on the system to improve the reliability methods. The testing was primarily focused on the following areas data corruption errors, functional unit errors, and the ability for hardware collaboration to mitigate errors.
7.2.1 Data Corruption Errors

The primary focus of this thesis was to mitigate data corruption errors, the preponderance of testing was conducted to determine the ability of the system to detect and correct data corruption errors. Data corruption refers to bit errors within the values being utilized as inputs, outputs, or constants within a function. These types of errors are predominantly generated by bit flips within the processor’s memory arrays or registers.

Analysis of SECDED Encoding with Redundant Copies

As mentioned in Chapter 3, one method for mitigating errors within the memory region is through SECDED encoding with memory scrubbing. However, this reliability is not gained freely. When compared with the less robust method of storing the data in multiple locations, interesting implications between the increases in array size and in computational time become evident. This can best be seen in Figure 7-1, from which the data in Table 7.1 was produced.

Table 7.1: System Characteristics for Data Value Error Detection and Correction

<table>
<thead>
<tr>
<th>Reliability Method</th>
<th>Required Array Size Increase</th>
<th>Required Processing Time ((\mu\text{sec}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECDED Encoding</td>
<td>2x</td>
<td>1.98</td>
</tr>
<tr>
<td>Parity Encoding</td>
<td>1.125x</td>
<td>0.28</td>
</tr>
<tr>
<td>Duplicate Data, same processor</td>
<td>2x</td>
<td>0.18</td>
</tr>
<tr>
<td>Duplicate Data, two processors</td>
<td>1x</td>
<td>19.29</td>
</tr>
<tr>
<td>Triplicate Data, same processor</td>
<td>3x</td>
<td>0.21</td>
</tr>
<tr>
<td>Triplicate Data, three processors</td>
<td>1x</td>
<td>28.19</td>
</tr>
</tbody>
</table>

As can be seen from Table 7.1, there is a large performance disparity (>843% increase in required computational time) between the SECDED encoded data and the comparison of duplicated/triplicated data buffers on the same processor. However, this high performance operation reduces the reliability of the result as each redundant
Figure 7-1: Timing Characteristics for Encoding and Redundant Storage of Data
value could experience an upset which leaves a different value, rendering the comparison unable to correct the errors. Alternatively, if only one of the redundant copies of the data is modified by an upset the comparison can correct multiple bit errors which the SECDED encoding could not correct. Finally, the lowest memory usage option is to redundantly store the data values on separate processors and use the interprocessor bus for comparing values. However, this method executes the slowest of all tested methods. An optimum design occurs with parity encoding where error detection is received at low area and execution time costs.

**Paxos Analysis**

Whenever multiple processors are used, messages must be passed between the processors to distribute the data (either as inputs or outputs). This message passing creates a performance bottleneck for simple operations on large data arrays because of the overhead associated with distributing the data. As mentioned in Chapter 3 (Figure 3-15) the standard consensus algorithm for distributed systems is Paxos which requires a large number of messages to be passed [13]. However, within the constraints of the CASTOR avionics system some of the assumptions of Paxos become invalid.

The Paxos constraint with the largest impact is the necessity for any processor to be capable of assuming the role of master, which requires each processor to store the final results from all redundant operations on the system. This does not hold on CASTOR as many devices are limited to having only a single MCU being the master due to interface requirements or do not require a knowledge of the execution history to properly operate the device. Thus eliminating this assumption removes the requirement for all results to be distributed across all the processors and can further be used to justify eliminating the two-phase commit process. The modifications these changes to Paxos create on the message passing scheme can be seen in Figure 7-2, where the dashed lines are the messages which can be eliminated because only the master must have the results from across the system. If the two-phase commit is removed, the “alternate allow point” becomes the end of the consensus algorithm, leading directly to the “process executed” stage.
A further modification which can be utilized to reduce the message passing overhead is to broadcast messages across the bus, allowing one processor to simultaneously transmit data to multiple processors. Passing messages using a broadcast technique is well suited to a bus topology as only one processor can drive the bus; however, in more complex topologies this can lead to inefficiencies as all the receiving processors must queue on sending a response back. Thus the usage of a broadcast message passing scheme depends upon the system dynamics, for CASTOR it proved to be a highly efficient communication protocol.

Depending upon the characteristics of the operation, it may not be required to transmit the inputs to the redundant processors. This occurs when a device is setup as a shared peripheral and all the processors snoop on the bus to receive data from the device simultaneously. However, for the calculations in this thesis a worst case was assumed where the input and output data must be transmitted to all the processors. The justification for changing from a two-phase commit to a single-phase commit is that a two-phase commit ensures a consistent initial state across all the processors; however, by having the master communication the input data to every processor a similar assurance of having the same initial state is created. Data corruption errors could occur in the interprocessor communication buffer prior to the message being transmitted or while it is in the receive buffer, so for the highest reliability levels
this operation should remain a two-phase commit to eliminate the chance of errors occurring and generating a false DUE. Through these modifications the number of messages required for determining consensus can be reduced from 6N-6 to N messages (where N is the number of processors in the network), as shown in Table 7.2.

<table>
<thead>
<tr>
<th>Message Passing Scheme</th>
<th>Number of Messages for a 3 Processor Implementation</th>
<th>Number of Messages for an N Processor Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Paxos</td>
<td>12</td>
<td>6N-6</td>
</tr>
<tr>
<td>Paxos, results not distributed to all</td>
<td>8</td>
<td>4N-4</td>
</tr>
<tr>
<td>Paxos, using Broadcast</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Paxos, results not distributed to all and using Broadcast</td>
<td>6</td>
<td>2N</td>
</tr>
<tr>
<td>Paxos, Single-Phase Commit</td>
<td>6</td>
<td>3N-3</td>
</tr>
<tr>
<td>Paxos, Single-Phase Commit and results not distributed to all</td>
<td>4</td>
<td>2N-2</td>
</tr>
<tr>
<td>Paxos, Single-Phase Commit using Broadcast</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Paxos, Single-Phase Commit using Broadcast and results not distributed to all</td>
<td>3</td>
<td>N+1</td>
</tr>
</tbody>
</table>

**Message Passing Analysis**

Multiple processors create many unique opportunities and operating paradigms, but the necessity of replicating data between processors limits the system performance because of the timing cost incurred when passing messages between processors (\(~ 4.71 \mu s\) in each direction on CASTOR). A design strategy which could lessen the overhead of message passing is to implement a high-speed data bus with a data rate within an order of magnitude of the processor’s clock. On a system such as CASTOR this would mean data rates greater than 4 Mbps and could be obtained using communication standards, such as TTTech’s TTP (10 Mbps) or TTEthernet (1 Gbps). Alternatively, a communication protocol with less overhead than CAN (at 51.1%) could achieve similar effects by increasing the effective data rate; however,
these schemes will likely reduce the robustness of the communication network (a key advantage of CAN).

Even with the high cost of message passing (for even the most reduced version of Paxos), there is an execution timing where performing redundant operations on separate processors is quicker than redundancy on the same processor, as shown in Equations 7.1 and 7.2 for the most reduced version of Paxos. This is the ideal point as performance and reliability both increase under these conditions. Increasing the effective bus data rate lowers the execution time required to enter this ideal operating zone. However, certain critical operations require increased reliability but are not computationally time consuming enough to mitigate the message passing costs. For these operations, there is a design trade between executing the redundancy on a single processor for the performance benefits and increasing the reliability of the operation by executing on multiple processors.

![Figure 7-3: Message Passing Timing Characteristics](image)

In order to characterize this trade space, the time required for message passing was analyzed by measuring the time required to transmit different data types and array sizes. This data has been compiled in Figure 7-3 and was used to derive Table
7.3 for creating an estimate on the time required to transmit data between processors, roughly 4.71 $\mu$sec (referred to as $t_{bit, TX}$) on the CASTOR avionics system.

Table 7.3: Message Passing Timing Characteristics for Different Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Average Time to 1 Processor ($\mu$sec $_{bit}$)</th>
<th>Average Time to 2 Processors ($\mu$sec $_{bit}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit int</td>
<td>4.66</td>
<td>4.71</td>
</tr>
<tr>
<td>32-bit long</td>
<td>4.73</td>
<td>4.76</td>
</tr>
<tr>
<td>32-bit float</td>
<td>4.70</td>
<td>4.73</td>
</tr>
<tr>
<td>64-bit double</td>
<td>4.66</td>
<td>4.70</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>4.69</strong></td>
<td><strong>4.73</strong></td>
</tr>
</tbody>
</table>

With this timing estimate for message passing, one can estimate the processing time required for redundant execution on multiple processors (using the most reduced version of Paxos) to be more efficient than redundant execution on the same processor. This occurs when the function’s execution time is greater than or equal to the message passing time, where the message passing time is the summation of the number of messages passed over the network (n), set by the third column in Table 7.2 for the version of Paxos being implemented, with each message requiring the time found by multiplying the number of bits transmitted (m) with the data rate ($t_{bit, TX}$). The inequality in Equation 7.1 defines the minimum processing time.

$$t_{operation} \geq t_{msg, passing} \quad (7.1)$$

$$t_{msg, passing} = mn t_{bit, TX}$$

m = number of bits transmitted
n = number of messages passed over the network (from Paxos)
$t_{bit, TX}$ = data rate

Meeting the timing requirements of Equation 7.1 guarantees that the time required for message passing is less than that of a second redundant execution on the same processor (which generates the ability to detect errors) and is therefore more efficiently executed on multiple processors. However, correcting errors requires three executions. Thus Equation 7.1 can be relaxed to the operating time given by Equation 7.2 for creating a performance increase when correcting errors. For CASTOR, where n is assumed to be three, Equation 7.2 is satisfied when $t_{operation} \geq 7.065m \mu$sec.
\[ 2t_{\text{operation}} \geq t_{\text{msg\_passing}} \] 

(7.2)

Functions that fulfill these timing requirements (meeting Equation 7.2) experience performance benefits when executed redundantly on multiple processors. When a function’s \( t_{\text{operation}} \gg t_{\text{msg\_passing}} \) it can be broken down into distinct sections which can be checkpointed. These checkpointed values can be distributed and compared to detect errors prior to calculating the operation’s final output which benefits performance by detecting errors earlier. A function which exhibits these traits is the ADCS control task which uses the sensors and actuators to maintain a desired orientation of the satellite. The ADCS control task is easily broken into distinct segments, converting the sensor data to the units/coordinates needed for further calculations, calculating the predicted state through an extended Kalman filter, computing the state commands from the periodic control law, and converting the state commands into actuator outputs. These four segments were chosen to be the checkpointed values because of their ability to act as stepping stones to the next checkpoint and each independently meets the timing requirements specified in Equation 7.2. For example, the extended Kalman filter calculation requires a 144 entry input of the sensor data to produce a 144 entry Kalman gain matrix which is used for calculating the state commands in the control law. Thus both of these 144 entries are the checkpointed values. Additionally, the timing analysis for calculating the Kalman gains is shown in Table 7.4, demonstrating that for floating point data types the requirements of Equation 7.2 are met.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Average Time to Execute (msec)</th>
<th>Average Time to TX Results (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit long</td>
<td>14.65</td>
<td>65.29</td>
</tr>
<tr>
<td>32-bit float</td>
<td>40.4</td>
<td>63.14</td>
</tr>
<tr>
<td>64-bit double</td>
<td>73</td>
<td>125.59</td>
</tr>
</tbody>
</table>

Thus complex functions such as calculating the extended Kalman filter gains generate a performance increase when redundantly executed on multiple processors,
whereas a simpler operation such as addition or division of elements of an array by a constant value (reference Figure 7-4) decreases performance because it does not meet the timing specifications of Equations 7.1 or 7.2.

![Figure 7-4: Simple Operation Timing Characteristics](image)

**Data Corruption Summary**

The use of redundant execution or data encoding generates a high probability that an upset-generated error in data values will not propagate to the output. Each method offers a unique set of strengths and weaknesses. Redundant execution on the same processor gives the highest performance, but is blind to functional errors within the processor making it susceptible to silent data corruption (SDC) produced by repeated incorrect outputs. Utilizing multiple processors for redundant operations has a performance penalty from message passing; however, this penalty is indirectly proportional to the processing time required by the function. It builds upon the reliability from redundant execution on the same processor by requiring multiple processors to create the same error before allowing SDC to occur. Yet it creates a vulnerability by creating the potential for each processor to generate different outputs.
impeding consensus. Finally, both these redundant executions suffer from the chance of experiencing an upset during the comparison operation which is not atomic or protected from SEE-induced errors.

Encoding the data to aid in the detection and correction of memory cells further improves the system's reliability by stopping errors from progressing outside of the memory arrays. SECDED creates a unique robustness capability by being able to correct errors in redundant copies of data that all have the same errors or all have different errors, something redundant copies cannot perform. Parity encoding provides an optimal, minimum protection encoding scheme by having the lowest total costs in array size and processing time increases; however, it cannot correct errors once they are detected. This additional level of robustness from SECDED or parity encoding can be applied with the redundant execution on either the same processor or multiple processor; however, they carry a significant performance implication.

\section{Functional Unit Errors}

A weakness of redundant execution on the same processor is the malfunction of a processor’s functional unit (e.g. ALU performing incorrect operation or a bit flip within the status register) causing consistently incorrect results. By generating the same error at each execution incorrect but matching outputs will be produced defeating the redundant execution and causing SDC. The capability to detect and correct these types of errors is a key necessity for a fault-tolerant avionics system.

Such errors can be detected using a single processor by performing a test operation which checks the functionality; however, this test program could only inspect a subset of the operations and would also be in danger of falsely detecting a malfunction because the input or output suffered an upset. For purely arithmetic operations, AN codes can detect these functional unit errors because they use differing inputs to check the same operational properties. Neither of these error detection strategies are ideal because of their limited utility and potential for false detection.

It was deemed more effective to utilize the existing redundant hardware to detect functional unit errors by comparing the results between different processors. The
probability of experiencing the same functional error on a separate hardware units is low (~4E-10), allowing the system to confidently correct these errors by relying on the other processors to produce correct results.

### 7.2.3 Hardware Reliability

Part of CASTOR’s symbiotic relationship between hardware and software for generating reliability is that while a large portion of the reliability is gained through software, much of the basic functionality is guaranteed by the hardware. Additionally, key capabilities of each method are used to compliment or reinforce each other. These are exploited most in the commanding of mission-critical devices.

### Hardware Checking

The CASTOR avionics system required high-criticality outputs be determined by voting on the outputs from two separate processors. The design of the circuitry which supports this functionality for GPIO is discussed in Chapter 5 (reference Figure 5-5).

During operational testing, it was verified that the output ($V_{out}$ in Figure 5-5) was activated (went high) when both processors issued the command. Further, transitions such as a processor being restarted resulted in the output being deactivated (going low). Thus verifying the desired operational characteristics of only being operational when the system was fully functional and in consensus about operating the device. One challenge is upon the loss of a processor, the device can no longer be operated. This potential loss of functionality must be evaluated against the chance of a faulty command creating an unwanted the output. Alternatively, additional complexity could be added to enable additional processors to vote on commanding the device and only require a majority consensus to activate the device.

### Output Commandability

To ensure operability of certain devices that are commanded by a single processor, multiple processors’s outputs were connected to the devices through an arbitration
circuit. This allows for multiple processors to command the device and thereby eliminates a single point of failure. A detailed discussion of the arbitration circuit for GPIO can be found in Chapter 5 (reference Figure 5-4).

In testing, this circuit allowed for the device to be commanded when only a single processor was operational. If the processors sent conflicting commands the device would be commanded into the active state and no damage is done to the arbitration circuit. This is seen as the ideal operating scheme because if a processor was malfunctioning and commanded the device incorrectly, that processor would not be able to remove the device’s functionality from the system. If the device should be in the active state, the processor would be unable to change the device’s state. Alternatively, if the device was in an inactive state the processor would be able to command the device to an active state, and if this was detrimental to the system the device could be brought back to the inactive state by having a separate processor powered off the faulty processor, removing the incorrect command.

\section{7.3 Other Test Results}

In addition to testing the different reliability measures employed by CASTOR, additional testing was conducted on key features supporting the avionics system’s reliability operations. The tests included an analysis of the performance capabilities of a system requiring periodic restarts (such as Boeing’s COTS avionics architecture [10]) and the characteristics of a processor reprogramming another processor.

\subsection{7.3.1 Lock-step and Restart Comparison}

COTS avionics architecture implementing redundant processors running in lockstep and often require periodic restarts (on the order of once every second) to ensure that the processors remain in the same state. This minimizes the chance of undetected errors accumulating and creating an unrecoverable error such as each processor generating a different result. The effect of periodically resetting was modeled on the CASTOR avionics system to measure its performance impacts.
A simple test program was designed to measure this performance impact. The test iterated through a data buffer and increment all the values by a constant. Outside of this test program, an interrupt was created which saved the processor’s current state to non-volatile RAM and restarted the processor. After re-initializing, the processor would read the saved state data from a predetermined address in the non-volatile FLASH memory. This data included the memory location where the operation stopped at. With this information, the processor can continue executing the test program from where it was at prior to restarting. The duration required to execute the entire program was measured. The interrupt’s period and the data buffer’s size were varied to analyze their performance impacts, which are shown in Figure 7-5.

Table 7.5 is derived from Figure 7-5, and specifies the computational performance (in $\mu$sec/bit) for each interrupt frequency tested. The frequency of the interrupt was swept to determine the degree to which it impacted the operation. It was determined that the function’s performance (in $\frac{\mu\text{sec}}{\text{bit}}$) was linearly related to the frequency of the interrupt according to Equation 7.3, where $f$ is the interrupt’s frequency. The
frequency was never dropped to the 1 Hz rate used by Boeing due to the simplicity of the function under test completing execution before the interrupt occurred. However, the data can be extrapolated to account for the operational characteristics of such a system. The performance impact of continually saving the processor’s state and reloading the old state imparts a large performance decrease to the system’s capabilities because of the time requirements for writing and reading from the non-volatile RAM (a minimum of 2 milliseconds, seen in the constant term in Equation 7.3).

\[
\frac{\mu\text{sec}}{\text{bit}} = 0.747f + 2.37
\]  

(7.3)

Even when comparing the performance to that of the same operation being triplicated on the same processor (requiring 1.388 \( \frac{\mu\text{sec}}{\text{bit}} \), determined from the Single Processor 3x line in Figure 7-1(b)) there is a large performance difference (minimum of 3.97 \( \frac{\mu\text{sec}}{\text{bit}} \)). However, the performance losses from restarting are comparable to the losses from message passing between processors (requiring \( \sim 19.61 \frac{\mu\text{sec}}{\text{bit}} \) determined from the Two Processors 1x line in Figure 7-1(b)). Thus for simple operations, a system periodically restarting lags the performance of single processor executing the functions redundantly and has similar performance to a message passing system if the restart rate is high enough. A test of the difference between restarting and message passing should be performed using a more complex test operation, such as the Kalman filter gains calculation, to fully characterize the differences.


7.3.2 Reprogramming

The CASTOR avionics was designed such that every processor has the capability to reprogram another, faulty processor. For the dsPIC33 MCU, the maximum data rate at which programming can be accomplished is 5 MHz and with the required write delays it creates the theoretical programming performance specifications seen in Table 7.6. The timing performance of the reprogramming implementation on the CASTOR avionics system is also shown in Table 7.6.

<table>
<thead>
<tr>
<th>Program Size (KB)</th>
<th>Theoretical Time (sec)</th>
<th>Implemented Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.188</td>
<td>0.38</td>
<td>1.52</td>
</tr>
<tr>
<td>32</td>
<td>0.96</td>
<td>12.6</td>
</tr>
<tr>
<td>256</td>
<td>5.05</td>
<td>47.6</td>
</tr>
</tbody>
</table>

The implemented reprogramming code lags the theoretical maximum performance by a significant amount (∼836%) due to delays from reading the dsPIC’s FLASH program memory and from inefficiencies in transmitting the serial data having a maximum rate of ∼1 Mbps. Instead of directly toggling the bits to reprogram the failed dsPIC, a more efficient strategy would be to utilize SPI communication; however, this was not viewed as a critical concern for CASTOR because if the reprogramming required a few additional seconds to recover a failing processor no lasting damage would occur, the spacecraft’s attitude may drift for a minute but these errors will be quickly mitigated. The largest impact to the satellite would be missing a trivial number of attitude control command updates and delaying the responses to external stimulus, such the solar panels exceeding the desired temperature (in a worst case scenario). The minor effects could be rapidly corrected once the programming task was finished and all the processors were fully operational, thus posing no danger to the satellite’s mission. Additionally, the increased system capability from bringing the failed processor back on-line far outweigh these momentary operational challenges.
7.4 Conclusion

Throughout the testing of the CASTOR avionics system, many design and performance characteristics of the COTS-based multiple microcontroller architectural design were realized. Lessons were learned from how to select reliability functions to the ideal implementation of message passing between processors. Through critical analysis of the various reliability tests imposed on the CASTOR avionics, the critical design points listed in Table 7.7 were determined.

<table>
<thead>
<tr>
<th>Design Aspect</th>
<th>Lesson Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data corruption mitigation</td>
<td>The methods tested (redundant execution and SECDED encoding) mitigate most data corruption errors, each with unique strengths and weaknesses</td>
</tr>
<tr>
<td>Function unit errors</td>
<td>Relying on multiple processors to detect functional unit errors is an ideal solution</td>
</tr>
<tr>
<td>Impact of restarting lock-stepped system</td>
<td>Same order of magnitude impact as message passing has on simple operations</td>
</tr>
<tr>
<td>Message passing overhead</td>
<td>High cost for simple operations; however, functions meeting Equation 7.2 receive performance increase</td>
</tr>
<tr>
<td>Inter-processor network data rate</td>
<td>MCU’s built-in communication protocols proved to have larger overhead than desired. Higher speed data rates are ideal, but they add complexity from additional circuitry or reduce communication protocol robustness</td>
</tr>
<tr>
<td>Hardware supporting software reliability</td>
<td>Strong benefits can be gained from adding hardware support for checking key reliability operations of simple GPIO-based outputs</td>
</tr>
<tr>
<td>Reprogramming rate</td>
<td>The capability of reviving failed processors outweighs the minor costs of not performing critical operations for a few seconds because of the net system gain</td>
</tr>
</tbody>
</table>
Chapter 8

Conclusion and Future Work

This thesis explored the creation, implementation, and testing of a novel spacecraft avionics architecture which relies upon redundant COTS hardware and software algorithms to create the same reliability and performance capabilities currently offered by RadHard processors. The avionics architecture is unique in its distributed system nature as opposed to the standard approach of utilizing redundant hardware operating in lockstep with output comparing hardware. Operating as a distributed system allows for the reliability level of each software operation to be placed at different levels according to the operation’s output constraints, in particular the result’s correctness and the operation’s run time.

The avionics architecture was implemented on the CASTOR satellite using three redundant MCUs. These MCUs both divided the work load and aid each other in performing error checking results. The redundant processors are interconnected in a flat network for transferring data to support: error checking operations, task sharing, and liveness checking. In addition to having separate processors perform error checking operations, redundancy can also be created internally on a single processor through repeated execution and comparison of results prior to committing a final output. The architecture can be augmented, for high-criticality operations, by incorporating hardware checking outputs from multiple MCUs, using RadHard components. This ability to tailor the amount of reliability given to a function allows for the system to accommodate a wide array of mission requirements.
The implementation of this architecture on CASTOR allowed for extensive testing and characterization of the architecture, which provided critical feedback which was used to improve the architecture and its supporting fault-tolerance mechanisms. The relationship between the different reliability algorithms and their impacts on a function’s run time was explored. Further, CASTOR provided the avionics architecture with a real satellite and its operating requirements to use for verifying that the architecture supported the necessary capabilities.

8.1 Multiprocessor Architecture

This thesis investigated the usage of multiple processors, operating without the constraints of producing outputs in lockstep, for creating a reliable avionics platform. This setup is ideal for spacecraft applications where hardware spares have been a standard design feature because the system must be capable of tolerating a hardware failure to ensure longevity. The avionics architecture supports for a variable number of processors allowing the architecture to be used for systems requiring high performance processing to systems requiring increased longevity, along with many systems in between these two extremes.

This avionics architecture further distinguishes itself through the usage of COTS processors. COTS are utilized for their design advantages in cost, performance capabilities, power consumption, and required board area. Additionally, COTS processors bring a diversity of products and capabilities to the spacecraft avionics design space. COTS processors can be considered for use in spacecraft avionics because of their increased TID tolerance from modern processors’s reduction in process size. Which coupled with the proper reliability mechanisms enable the COTS products to mitigate SEE-induced errors.

A first-level analysis of SEE-induced error mitigation techniques were discussed in this thesis for handling data corruption errors. The techniques of redundant execution on a single and multiple processors along with data encoding were investigated. Redundant execution on a single processor proved to be a high-speed method for
quickly identifying transient errors; however, it left the possibility for errors to not be detected. While not creating an optimal solution, many of the reliability limitations of redundant execution on a single processor can be mitigated by performing the redundant operations on separate physical devices; however, this creates potential performance disadvantages. Many of these performance challenges stem from the data rate between processors, thus reducing the amount of data which must be communicated or increasing the data rate between processors makes the implementation of redundant execution on separate processors less costly. Additionally, encoding the data for detection and correction of errors greatly strengthens the robustness of the system to data corruption errors.

Finally, this multiple processor architecture aligns itself with the growth path of modern processors towards multicore processors. These advanced devices compliment the COTS-based, multiprocessor architecture designed in this thesis by providing hardware redundancy on a single IC. The same logic used with multiple processors can be implemented with multicore processors, with the advantage that cores on the same processor have significantly higher inter-processor data rates, allowing simpler operations to be performed redundantly on separate processing units quicker than they could be redundantly executed on a single processing unit (increasing both system performance and the reliability of the result). Thus this architecture is seen as an ideal growth path for future satellite avionics systems.

8.2 Microcontrollers in Spacecraft Avionics

This thesis also analyzed using microcontrollers for all spacecraft computing needs. This is a growing trend in the small satellite and CubeSat design arena; however, it is not found in larger satellite designs. Microcontrollers have a SoC design, placing all the basic support components (e.g. RAM, interface controllers, and nonvolatile program memory) on the IC, making microcontrollers self-sufficient. Additionally, microcontrollers have reduced cost, power consumption, and size, which are all design factors spacecraft avionics designers aim to minimize.
Microcontrollers have many interface controllers on the processor IC itself, which reduces the number of extra interface ICs required. By reducing the number of components required and making modifications to the system software operations as opposed to hardware the avionics system becomes simpler and more adaptable. These traits allow for microcontroller-based systems to be rapidly prototyped and have minimal system costs (size, power consumption, cost) which are not standard features for spacecraft processors.

One of microcontrollers key benefits to spacecraft avionics is their large performance to power ratio (generally \( \sim 1,000 \frac{\text{MIPS}}{\text{W}} \)). Microcontrollers deliver an order of magnitude increase in \( \frac{\text{MIPS}}{\text{W}} \) over conventional processors, which coupled with the SoC design requiring minimal supporting hardware enables microcontrollers to reduce the avionics system power requirements by \( >75\% \).

Microcontrollers are limited in their processing capabilities; however, standard satellite operations do not require high computational throughput. In general, satellite’s require 1-3 MIPS for basic operations, which is easily met by microcontrollers. Some payloads however, do require high computational throughput. Microcontrollers can generate this processing power through the use of implementation of distributed system, whereby multiple processors operate independently to increase the system performance beyond that of any single processor. The presence of multiple processors further strengthens the system by allowing for processors to fail without jeopardizing the satellite’s mission.

Through testing on the CASTOR satellite, microcontrollers proved to be capable of performing all the required operations while also providing many design improvements. With one of the largest benefit being the simplicity brought to the system, effecting the setup of hardware interfaces, the programming of the operating code, and many other design aspects. Additionally, a large number of highly versatile I/O are provided, generating flexibility in the system’s capabilities. Further, the anticipated performance limitations proved to be nonexistent, with the system having more capability than anticipated or required.
8.3 Future Work

This thesis began many investigations, primary of which was the usage of a distributed, multiprocessor system for SEE-induced error mitigation. A basic utilization of a distributed system was explored in this thesis, significant work can be done in furthering the avionics architecture and its fault detection and mitigation strategies. In particular, increasing the modularity of the architecture to dynamically allow for changes to the number of operating processors should be pursued.

Limited analysis was given to growing the number of processors the architecture can support. Increasing the number of processors generates many ideal operational capabilities, such as cycling processor usage (for radiation, performance, and power applications), dealing with unexpected processing challenges, and performing more complex computations. Some of the design issues that would need to be addressed are the optimal design for the interprocessor network, how devices are connected to the processors/network, and how the processors are commanded (i.e. is there a master and many slaves or is it a peer-to-peer network). In particular, allowing for processors to join and leave the network at any point creates operational advantages for the different operational capabilities.

Other architectural design aspects that could be further studied include increasing the robustness of serial component connections, investigating the optimal interprocessor communication structure, increasing the architecture’s standardization and flexibility to allow for the implemented system to be easily transferred between satellites, and utilization of the redundant processors for parallel computing. Currently some serial components are only connected to a single processor which creates a single point of failure for the device. Developing a robust method connecting these devices to multiple processors would greatly enhance the system’s robustness. Furthermore, it would allow for multiple processors to receive the device’s data instantaneously without requiring use of the inter-processor network.

For fault detection, it is ideal to utilize different processors operating on the same data; however, the message passing overhead associated with the current CAN bus
implementation greatly impacts the run time of some operations. Higher throughput buses exist, such as TTTech’s TTP, which can increase the data rate by an order of magnitude. Or a communication protocol with less overhead could create an effective increase in the data rate. Increasing the communication bandwidth enables further reliance on multiple processor output checking which is more robust at mitigating SEE-induced errors.

An ideal implementation of the avionics architecture would be to have a standardized layout which could be utilized by different spacecraft without requiring modifications. In particular, it would be ideal if the design fit within the standard CubeSat constraints to provide maximal flexibility in utilization. This standardized format would specify the location of various I/O and allow for component interchangeability to support mission requirements. Furthermore, this modularity would aim to turn the avionics board into a PnP component that requires no modifications for usage by different satellite missions.

Using the redundant processors to efficiently perform parallel computations enables the architecture to perform complex operations such as three-dimensional matrix processing which future spacecraft missions will rely on for performing tasks such as computer vision-based navigation. From this basic parallel processing scheme, further development could be conducted towards designing redundant sets of parallel processing units that are dynamically reconfigurable. Thereby creating both a high performance and a high reliability system, possibly with minimal size and power costs if implemented using microcontrollers. Pursuing these developments ensures that the architecture can be used for future high-performance operations which creates support for the architecture’s utility, ideally increasing the its adoption.

The error detection and mitigation strategies addressed in this thesis are primarily adaptations of existing algorithms for correcting data errors. There are other algorithms for mitigating control flow errors, clock upset errors, and hardware faults (such as SEL), and other SEE-induced errors. The implementation and integration of these algorithms into the architecture’s core competencies would greatly increase its reliability. Further, as process sizes continue to shrink, the likelihood of MBUs
greatly increases and error detection and mitigation strategies must be developed to match the challenges associated with these types of errors. Therefore, additional work on the error detection and mitigation strategies, for both this architecture and for future processors, will be beneficial to future spacecraft avionics systems’s designs.

Finally, the usage of microcontrollers in small, university spacecraft has become accepted; however, commercial satellites have not examined their usage. Investigation into the factors stopping commercial satellites from utilizing microcontrollers would yield insight into how to increase the adoption of microcontrollers for spacecraft avionics. Additionally, increased study on the optimal design for a redundant, microcontroller-based avionics system would be insightful as this thesis focused on demonstrated a proof of concept and thus not all available trades were fully explored.
Appendix A

CASTOR Schematics and PCB Drawings

The following figures are the detailed design schematics and PCB layout documents for the CASTOR avionics board, the design of which can be found in Chapters 4, 5, and 6 of this thesis. Additionally, the board which provides the 3.3 V and 5 V power supplies to the avionics board is shown.

A.1 Avionics Schematics

The schematic documents for the CASTOR avionics board are shown in Figure A-1 through Figure A-12.
Figure A-1: Microcontroller 1
Figure A-2: Microcontroller 2
Figure A-3: Microcontroller 3

Figure A-4: Data Connectors
Figure A-5: Power Connections
Figure A-6: Memory Bus Circuitry
Figure A-7: Reprogramming Bus

Figure A-8: RS-485 Bus Interface
Figure A-9: Communications Modems

Figure A-10: JPEG Camera
Figure A-11: DAC Outputs

Figure A-12: Thermal Couple Interface
A.2 Avionics PCB Diagrams

The PCB layout for the CASTOR avionics board can be seen in the following figures. The board was a 4 layer board (2 signal layers and 2 power plane layers), utilizing 753 vias, using a minimum trace width of 7 mils, and a trace spacing of 7 mils to interconnected 143 components. The PCB drawings can be seen in Figure A-13 to Figure A-15.

Figure A-13: Top Layer PCB
A.3 Power Schematics

This section discusses the CASTOR power distribution board used to create the 3.3 V and 5 V signals which are required by the avionics board. This board was called the power distribution unit (PDU). The schematics for the board can be seen in Figures A-16 to Figure A-18.

Figure A-16: PDU Power Converters

Figure A-17: PDU Current Sensors
Figure A-18: PDU Connectors
The PCB layout of the power distribution board can be seen in Figure A-19.
Appendix B

SEL Detection/Mitigation Circuit

This thesis primarily dealt with SEE resulting in data corruption or incorrect logical operations. Another SEE induced error is SEL, where the device fails into a high current draw state. If the device is not rapidly power cycled, an SEL can result in permanent device damage or failure. In order to detect and mitigate these effects, a supplemental circuit was designed and tested (see Figure B-1).

![SEL Detection/Mitigation Circuit](image)

Figure B-1: SEL Detection/Mitigation Circuit

The SEL detection and mitigation circuit is composed of two components a Hall effect current sensor and a comparator OpAmp. The current sensor measures converts the current flowing into the device into an analog voltage. This voltage is then input into the comparator ($v_P$ in Figure B-2) along with a reference voltage ($v_N$ in Figure B-2). The reference voltage is predetermined based upon the known operating conditions of the IC. The output of the comparator circuit is connected to the IC’s active low reset or MCLR pin. With the comparator connected this way, its output signal is
specified in Equation B.1. For this design \(-V_{cc}\) was set equal to GND to accommodate the LVTTL logic levels (\(<0.5 \text{ V} = \text{low signal and } >2.5 \text{ V} = \text{high signal}\)). Thus an active low reset will be triggered if the sensed input to the IC (\(v_N\)) becomes greater than the reference voltage (\(v_P\)).

![Comparator OpAmp](image)

\[
v_O = \begin{cases} 
  V_{cc} & v_P > v_N \\
  -V_{cc} & v_P < v_N 
\end{cases}
\]  

(B.1)

This circuit was prototyped using the schematic seen in Figure B-3 and the components listed in Table B.1. With these components, a measured current of 0 A created an output of 2.5V and the output voltage increased by 0.2 V. A standard current draw by the resistor is 0.15 A, to set the reference voltage the maximum current was increased by an additional 50% to set the reference voltage at 2.545 V. This prototype can be seen in Figure B-4.

![SEL Circuit Schematic](image)
In testing, this circuit was capable of detecting currents greater than the limit (0.225 A) and send the active low signal, proving its utility as an SEL detection and mitigation circuit. This circuit was not further tested or incorporated into the CASTOR design due to time constraints.
Bibliography

[1] “SMD (Sufrace Mount Device) / SMT (Surface Mount Technology)”.


