OPTIMIZATION OF ERROR CORRECTING CODES IN FPGA FABRIC ONBOARD CUBE SATELLITES

by

Skylar Anthony Tamke

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

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2019

DEDICATION

I dedicate this to all the people who have helped me along my path to finishing my degree. I couldn't have done it without my close friends and mentors along the way.

ACKNOWLEDGEMENTS

I would like acknowledge Montana Space Grant Consortium and the Electrical and Computer Engineering department for funding my project through completion of this degree.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

ABSTRACT

The harmful effects of radiation on electronics in space is a difficult problem for the aerospace industry. Radiation can cause faults in electronics systems like memory corruption or logic flips. One possible solution to combat these effects is to use FPGAs with radiation mitigation techniques. The following Masters of Science thesis details the design and testing of a radiation tolerant computing system at MSU. The computer is implemented on a field programmable gate array (FPGA), the reconfigurable nature of FPGAs allows for novel fault mitigation techniques on commercial devices. Some common fault mitigation techniques involve triple modular redundancy, memory scrubbing, and error correction codes which when paired with the partial reconfiguration. Our radiation tolerant computer has been in development for over a decade at MSU and is continuously being developed to expand its radiation mitigation techniques. This thesis will discuss the benefits of adding error correcting codes to the ever developing radiation tolerant computing system. Error correcting codes have been around since the late 1940's when Richard Hamming decided that the Bell computers he did his work on could automate their own error correcting capabilities. Since then a variety of error correcting codes have been developed for use in different situations. This thesis will cover several popular error correcting method for RF communication and look at using them in memory in our radiation tolerant computing system.

1

INTRODUCTION

Many have looked up at the stars and wondered what is really out there. Now humanity is beginning to push into an age of space exploration. Companies are emerging around the globe in competition to make the first great strides to the vast expanse of space. The need for ever more complex computing to run the technology of these companies is continuously rising. As a rocket design gains the ability to self stabilize/land/relaunch itself, the computing system on these vehicles need to be evermore efficient and reliable. As Astronauts strap into their spacecraft the on-board computers need to be able to transport their precious cargo safely. As a roving space probes explore the boundaries of known space, their computers needs to be able to withstand hazardous and extreme environments. Wherever space exploration will take humanity there is one constant that can't be denied, we need computers in almost everything around us.

Being able to offload tasks to computer based automation and machine learning will keep human-based errors from effecting crucial space missions. Building a computer that can process faster is only a part of the battle for space exploration. When a spacecraft leaves the protective atmosphere & magnetic field around the Earth, the electronics on-board become susceptible to radiation from far off cosmic sources.

As today's rockets get more advanced, the computers that run their systems are required to process large amounts of data quickly and reliably. The farther a space vehicle gets from its control center on Earth, operators on Earth lose the ability to manually control in real-time. This opens up the need to develop systems that can

operate on their own with no supervision and complete tasks autonomously.

When exploring the distant places of the great frontier the devices will need to gather as much data as possible. In many cases the devices will need to be able to make complex decisions to new and varied experiences in the process of mapping out space. These capabilities will need further advances in computing past our current state of the art edge technologies on the ground. Advances in machine learning and artificial intelligence are proceeding at an astonishing rate, but having systems that can support these fields in space is something that leave a bit to be desired still. Mankind needs computing to scout out space ahead of human exploration and to do this well, the systems will have to report all aspects of the new environments they encounter, as well as reacting to varied and new conditions. This means that computers will need to be able to self-diagnose when something is wrong, and be able to correct or fix any problems that arise without supervision or control of an operator.

Current ground-based computers fail in space due to cosmic radiation. When close to Earth, computers are protected by the Earth's atmosphere and magnetic field that surrounds Earth. Once an object get far enough away from the surface of planet, there are no longer these protective shells. This allows cosmic radiation to affect electronics in several different ways. Whether this is a bit flip in a digital device or a build up of charge in the features of a component, computers will eventually fail. This problem is one that NASA is greatly concerned about and is trying to motivate researches to create new solutions.

There are a few methods existing to prevent these incidents but there are various pitfalls to using them.

Engineers has employed a method called radiation hardening by process (RHBP) in past and current space electronics to try and combat the effects of radiation. As mentioned in the name, this method is implemented during the fabrication of the

device by altering the materials in a device to prevent ionic charge from building up on these devices.

NASA is focusing on a radiation hardened by design (RHBD) method that pushes the concept of building a radiation tolerant system by creating non-standard geometries to make the transistors in a device.

Another method is triple modular redundancy (TMR), this method takes the existing hardware and creates two identical copies of the system. All three system's output is fed through a voting system that will detect differences in their outputs and assumes a fault. This method is considered to be an architectural approach to fault mitigation.

MSU has been working a solution to combat the effect of radiation on electronics for near a decade. This approach revolves around using reconfigurable devices called field programmable gate array (FPGA) devices. These devices allow targeted resetting of particular sections of the design. This has allowed MSU to design a system that can reset their processing cores while maintaining existing data. The design has redundant arrays of processing cores that are dynamically swapped out when a fault is detected on one of the cores. The next step will be to take all critical information and store it to memory, so when a processing core becomes faulted the new processing core that replaces it will be able to continue where the last core left off. This memory is susceptible to faulting, which makes this method of storing data without any faulting mitigation techniques problematic for space bound electronics. Thus a method that can detect errors that pop up in the memory and correct for those errors is essential to the design.

This thesis will investigate common error correcting methods that exist in telecommunication and memory devices and their application to protecting space-based memory. Of primary interest is the speed that each of the methods takes

to complete and the amount of hardware resources each of the methods takes to implement on a FPGA. This thesis's research revolves around a softcore processing system with its own unique memory system. The processing cores are repaired when a fault is detected through several error detection processes implemented by current and previous graduate research assistants. The memory will be isolated from these error detection processes and have its own error correction system that will detect when one of the memory cells is different from the other memory cells. Once detected the error will be corrected to match the other memory cells.

This thesis will discuss the feasibility of using ECCs to improve reliability in FPGA designs. Particularly the trade off between the error correction capability of the ECC vs speed and hardware resources needed. Finding a optimized balance between these is key to creating a design with high-speed processing capabilities. This trade off is of particular interest to space computers because adding more sophisticated ECC strategies can actually result in decreased reliability due to the increased area on the FPGA that is susceptible to radiation. The following chapters specify the motivation for this project and the need error correcting codes on the system. Future demonstrations of concepts discussed in this thesis are also outlined. From this point on the system will be referred to by the designation Radiation Tolerant Computing System (RTCS).

MOTIVATION

Space technology requirements

The current cutting edge flight computers are outlined in TA 11 roadmap [19] as mainly single radiation-hardened custom processors. These systems are shown to have 35-400 million instructions per second (MIPS), 10-200 million floating point operations per second (MOPS), power requirements of 20W-30W, and power efficiency of 20 MIPS/W. Flight computers will need to move towards a goal of 1000+ MIPS, 1000+ MOPS, consume low power ($< 5W$), and achieve power efficiencies of less than 20MIPS/W. The current designs of single radiation-hardened custom processors is prompting support for moving away from these design, primarily due to high cost. This movement pushes further technology development to commercial over the shelf (COTS) for future radiation tolerant computing systems.

Harmful radiation effects

Total ionizing dose (TID) and single event effects (SEEs) are two types radiation events that cause failures on electronic devices. [3] When ionizing radiation strikes an electronic devices, it creates electrical/hole pairs in the device. The electron/hole pair causes temporary excess charge in the device. Most failures from TID happen when this excess charge gets stuck in insulation material layers within a device [23] When a electron hole pair gets stuck in the gate oxide of an transistor a channel can open, pushing the transistor into an permanently active state. [3] A TID can occur between transistor devices, while this doesn't affect the transistor directly it can lead to leakage current between devices. Leading to increase in power-usage and will lead to failure over time. [2]

Failures in devices caused from total ionizing dose (TID) are caused by low energy protons and electrons (< 30MeV/AMU). [13] A visual of this effect on a MOS device is shown the right side of Figure 2.1. [22]

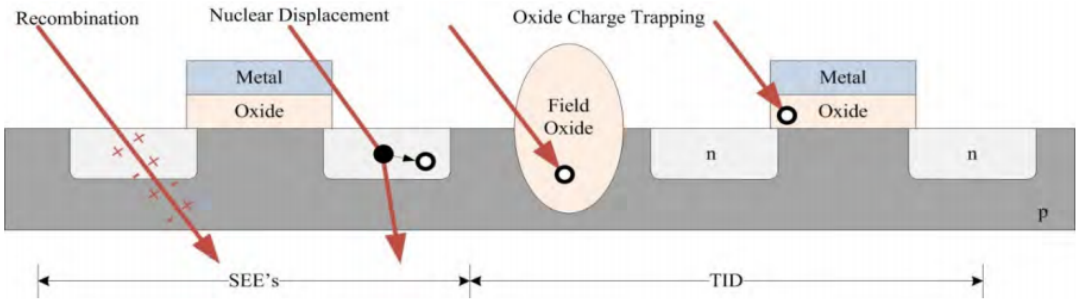

Figure 2-1. MOSFET cross-section showing radiation faults by SEEs and TID [5].

Figure 2.1: MOSFET cross-section showing radiation faults by SEEs and TID [22]

Single event effects (SEEs) occur when a high-energy particle strikes the diffusion region of a device. As the particle travels through the material it leaves behind a electron-hole pairs in its path. This incident happens so quickly that most of the electron hole pairs recombine with no effects on the material, but if this path crosses through certain parts of the circuit unwanted logic level transitions could be induced. Unlike TID, SEEs do not cause permanent damage to a device, the device may just experience a bitflip. While no damage to the device is good, erroneous data or unwanted operations can be triggered by an SEE. Possibly even a full system failure depending on what part of the device was struck. When a SEE causes induces a voltage bias on a circuit, possibly causing current flow, this is called a single event transient (SET). When a SET occurs and upsets a latched value in a flip-flop or some other memory element, this is called a single event upset (SEU). These are usually considered to be a "bit-flip" on the element that they occur. SEUs can affect both bipolar junction transistors (BJTs) and metal-oxide-semiconductor field-effect transistors (MOSFET). [7] If a SEU is detected and the device is reset then there is no

permanent damage to the device. However, if a SEU strike causes memory corruption or permeates a logic shift through multiple levels this is classified as another event called a single event functional interrupt (SEFI). SEFIs can create cascading behavior problems as well as a drop in power efficiency on the device. SEFIs are more difficult to recover from, requiring a full system power cycle or re-configuration. There is also a single event latchup (SEL) that can be caused by SEU. A SEL occurs when two transistors are stuck in states that short power to ground, causing excessive current draw and permanently damaging the device. [3] Figure 2.1 shows the different incidents that can happen for the range of SEEs.

High energy ionizing radiation is detrimental to electronics. Ionizing radiation has sufficient energy to break electrons free from their electron shell on their prospective atoms and molecules. [3] Generally the concern about radiation revolves around the concept that radioactive particles travel at relativistic speeds. Alpha and Beta particles traveling at this rate are capable of causing soft errors but can be deterred by adding shielding to a device. Gamma radiation can pass through shielding but with level of energy in the range of a few hundred keV, not having sufficient energy to affect digital electronics. Radiation damage usually comes from galactic cosmic rays (GCR), these cosmic rays are created outside of our galaxy and provide a continuous radiation environment that permeates interplanetary space. [7] [24] GCRs are made up of 85% protons, 14% alpha particles, and 1% heavier nuclei. [3] These particles contain levels of energy up to 1 GeV and are highly charged, making them densely ionizing. Table 2.1 shows the energy levels of various radioactive particles. High energy ionizing particles have large amounts of kinetic energy, which can be transferred into a device as the particle collides with the substrate. If this energy exceeds the materials band gap energy an electron could be excited from the valence band to the conduction band, creating an electron-hole pair. [1]

Particle Type	Energy Range
Trapped protons and electrons	≤ 100 MeV
Alpha particles	5 MeV
Solar protons	≤ 1 GeV
Cosmic rays	≥ 1 GeV

Table 2.1: Energies of various particles [22]

The earth is encapsulated in magnetic fields, making up the magnetosphere. These field trap low energy charged particles, diverting them from their origin path and becoming part of the Van Allen Belts. A visual of the Van Allen Belts is shown in Figure 2.2. [20] Trapped particles spiral around the magnetic field lines as they drift around the planet. Particles trapped in these field have been known to cause soft errors in electronics but can be mitigated by a certain amount of shielding. [6] The region of earth where the radiation belts are closest to earth is a spot just off the coast of Brazil called the South Atlantic Anomaly (SAA). This region is created due to the offset from the Earth's axis of rotation show in Figure 2.3. Radiation intensity can be increase by an order of magnitude while traversing the SAA, increasing the chance that electronics will be damaged by radiation. The Earth's atmosphere attenuates the majority of the remaining radiation that gets through the magnetosphere. These protective shields are the reason space radiation isn't a concern to ground-based computers.

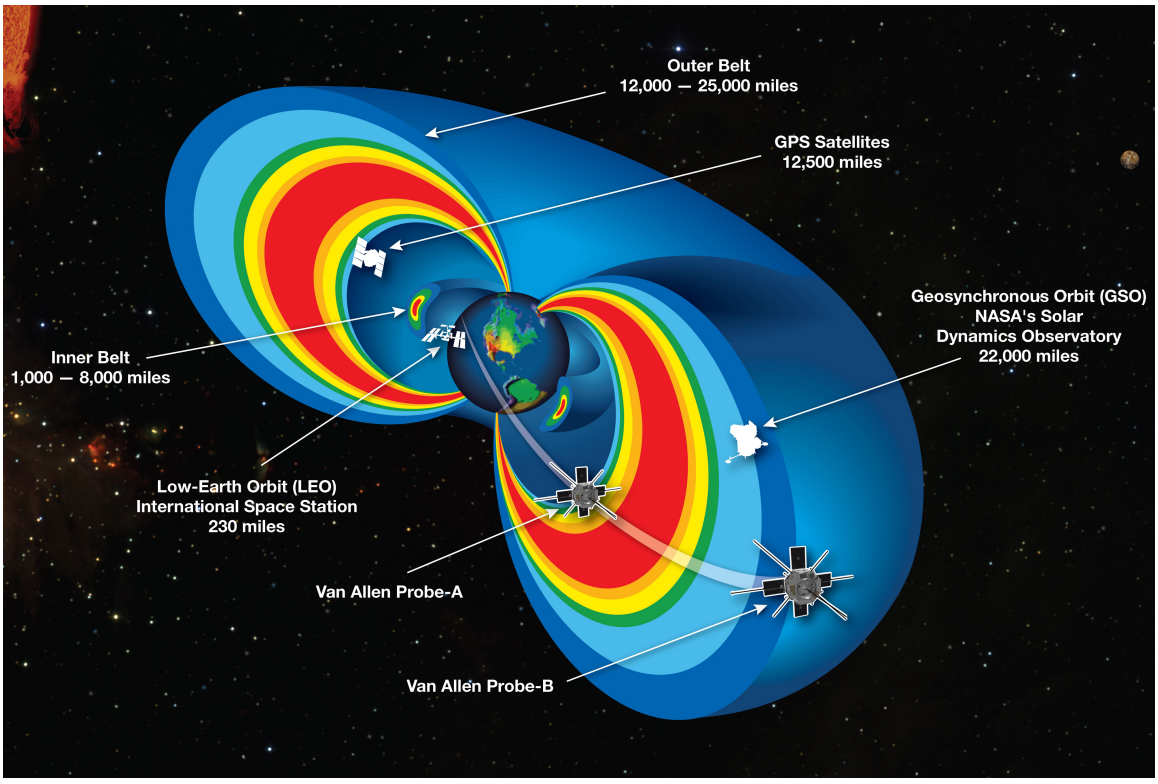

Figure 2.2: Van Allen belts

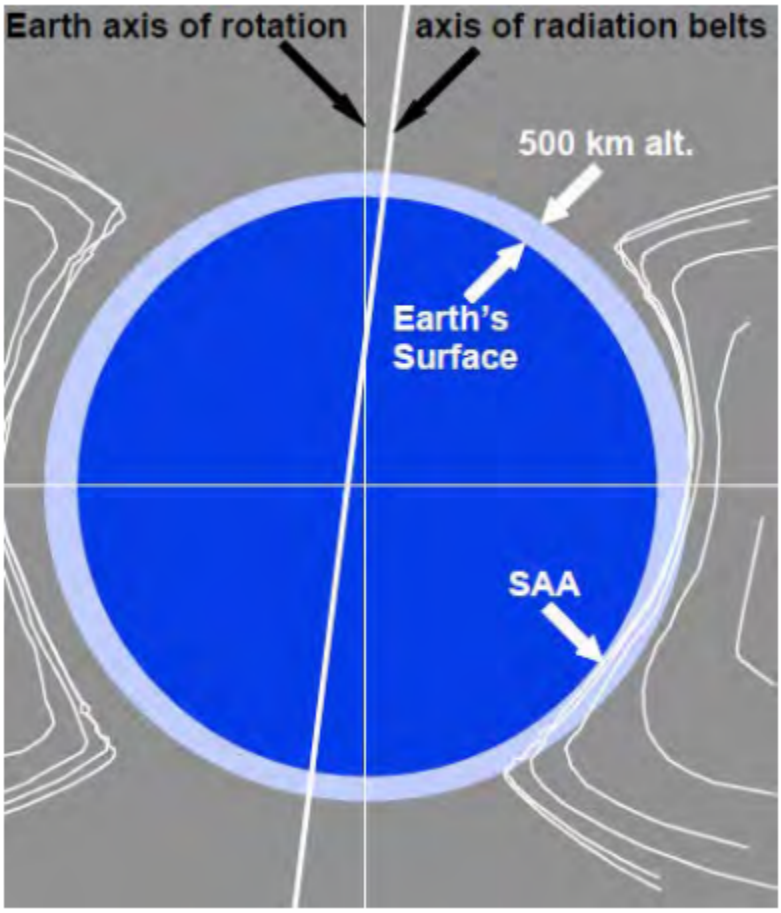

Figure 2.3: Radiation belts and location of the South Atlantic Anomaly [27]

Common Radiation Mitigation Techniques

There are various techniques employed to move towards mitigating damage from ionizing radiation. These techniques include shielding, creating radiation hardened electronics as well as enacting specific types of fault tolerant architectures.

Shielding

Shielding is an effective solution to protect electronics from alpha and beta particles, but gamma particles, neutrons, and heavy ion strikes are not mitigated by shielding. Figure 2.4 shows that the proton dose decreases by less than half when the

shielding is more than doubled. In most cases, increasing the shielding on a design is impractical for terms of mass and cost, especially since an increase of shielding is relatively ineffective at reducing the GCR spectrum. Shielding can also lead to cascading particles of secondary radiation when struck by GCR.

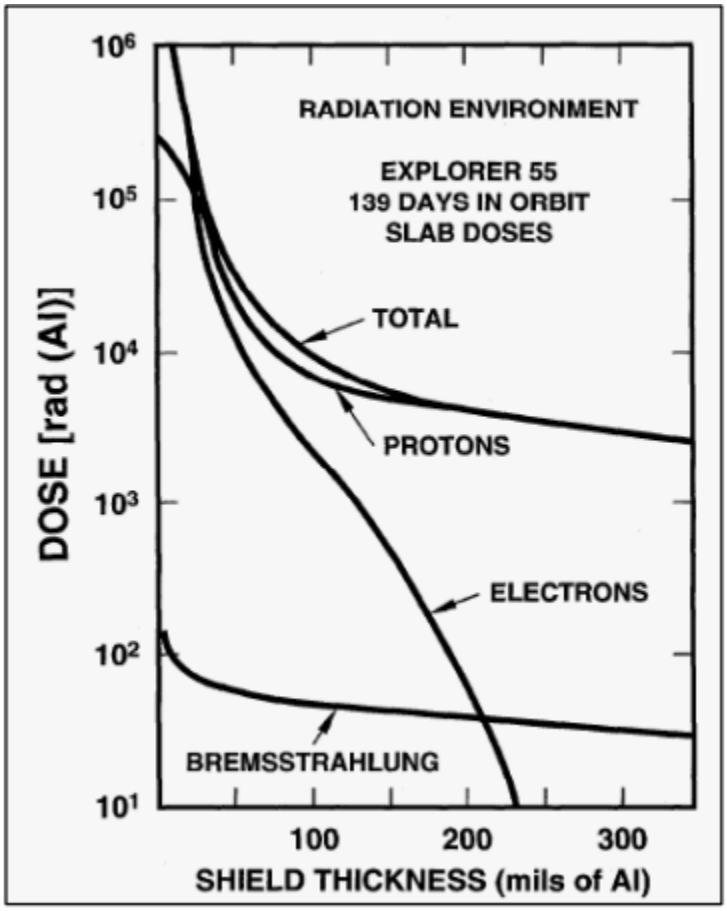

Figure 2.4: Total dose as a function of aluminum shielding thickness [25]

Material Hardening

An alternate solution to mitigating the effects of radiation is to modify the materials of the device. Usually this technique is called radiation hardening by process (RHBP), hardening devices with RHBP involves changing the steps in fabrication of the silicon device. Silicon on insulator (SOI) technology is one example of RHBP, this involves adding an insulator layer below the silicon junction (Ex. Figure 2.5).

The insulating layer can be comprised of either silicon dioxide or sapphire (also called Silicon on Sapphire (SOS)). Originally developed to reduce parasitic capacitance in a device, SOI and SOS also decrease charge trapping by greatly reducing the thickness of the semiconductor substrate. [7] Device isolation is also a benefit of this process.

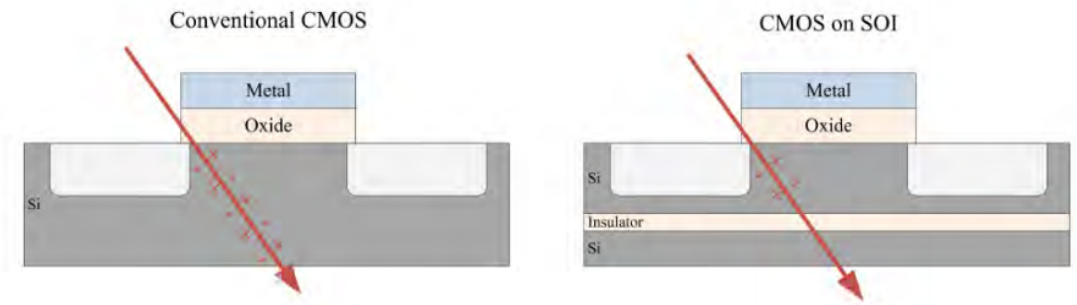

Figure 2.5: Conventional CMOS transistor compared to a silicon-on-insulator device [22]

Another technique to preventing charge from getting trapped in the insulating layers is the radiation hardening by design (RHBD). This technique involves using isolated transistors and guard rings to provide conduction paths (Figure 2.6). Charge induced by radiation strikes flows through the conduction paths instead of remaining trapped in the insulating material. [15]

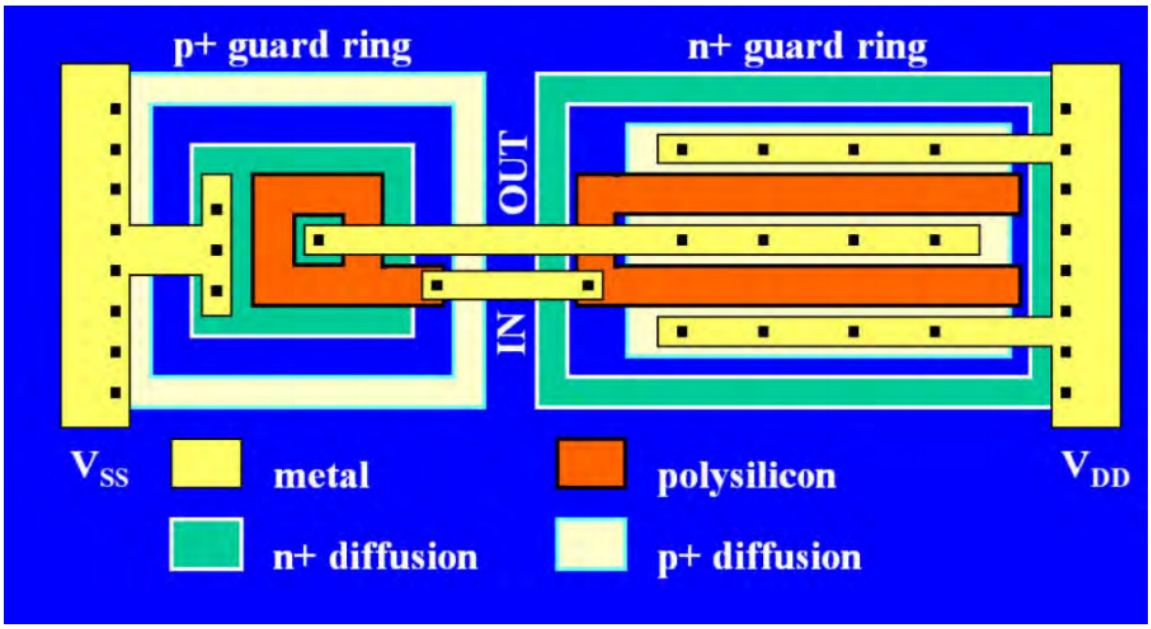

Figure 2.6: Radiation hardening by Design inverter using isolated transistors with guard rings [14]

Architectural Hardening

While shielding, RHBP, and RHBD provide extended tolerance for TID effects on electronics, SEEs can still cause faults in the system. An approach to combat SEEs is to design an architecture to handle any SEEs that strike the device. The conventional approach is to use circuit redundancy and memory checking to identify when a SEE happens and correct and resulting faults. The following paragraphs will explain these two methods in further detail.

Triple Modular Redundancy (TMR) is a common SEE mitigation technique. TMR design implement three of the same active circuit and have their outputs fed though a voting process to detect any outliers. This method assumes that an outlier signifies a fault has occured in the outlier circuit. The two other circuits provide a valid output while the faulted circuit is repaired and TMR operation can continue.

Memory scrubbing is a process where the configured pattern of a system is compared to a known good copy. This good copy is refered as the "golden copy" [3]

and would ideally be stored in a radiation tolerant memory unit such as PROM or EEPROM. If a system doesn't have sufficient fault detection capabilities it may rewrite the entire configuration, this is known as blind memory scrubbing. Whereas if the system can detect where the fault is and can correct it with the "golden copy" this would be called readback memory scrubbing. Readback scrubbing can prevent the system from going down for a full reset by detecting a fault and determining whether or not it needs to be corrected. With Readback scrubbing's capability of identifying where a faultis located, this can be used to log potential strikes and record analytical information about any radiation strikes that do happen. Blind scrubbing would just wipe out any of this existing data with a full rewrite, not even needing to know if a fault actually existed in the first place.

Triple modular redundancy and memory scrubbing are generally used together to mitigate radiation effects. [5] Using TMR prevents single upsets from disrupting normal operation, where as memory scrubbing corrects bit flips in the configuration memory. Keeping the number of upsets as low as possible and helping keep the system from failing over a large amount of time.

Existing Radiation Hardened Processors

Most existing radiation hardened processors are used by the aerospace industry, nuclear industry, and the military. The Mongoose-V built by Synova Inc, for example, is a radiation hardened flight computer used by New Horizons. [3] The Mongoose-V costs $20,000-$40,000 and can run at 15 MHz. [17] Another rad hardened process with a little more reputation is the RAD6000 by BAE Systems. This device costs between $200,000 and $300,000 and can run at a maximum of 25MHz. This rad hardened processor was used on the Spirit and Opportunity Mars rovers developed by NASA-JPL (Jet Propulsion Laboratories). This processor was succeeded by the

RAD750 by BAE Systems. Capable of running at 200MHz and costing $200,000, this rad hardened processor is two of the processors that run on the Curiosity Mars rover.

Limitations

The cost of these rad hardened systems greatly exceeds commercial processors with similar processing speeds. While large entities like private defense contractors, government funded space agencies, military can afford these products smaller organizations like University research labs or startups cannot. Rad hardened processors have had a history of lagging behind 10 years in performance (Figure 2.7).

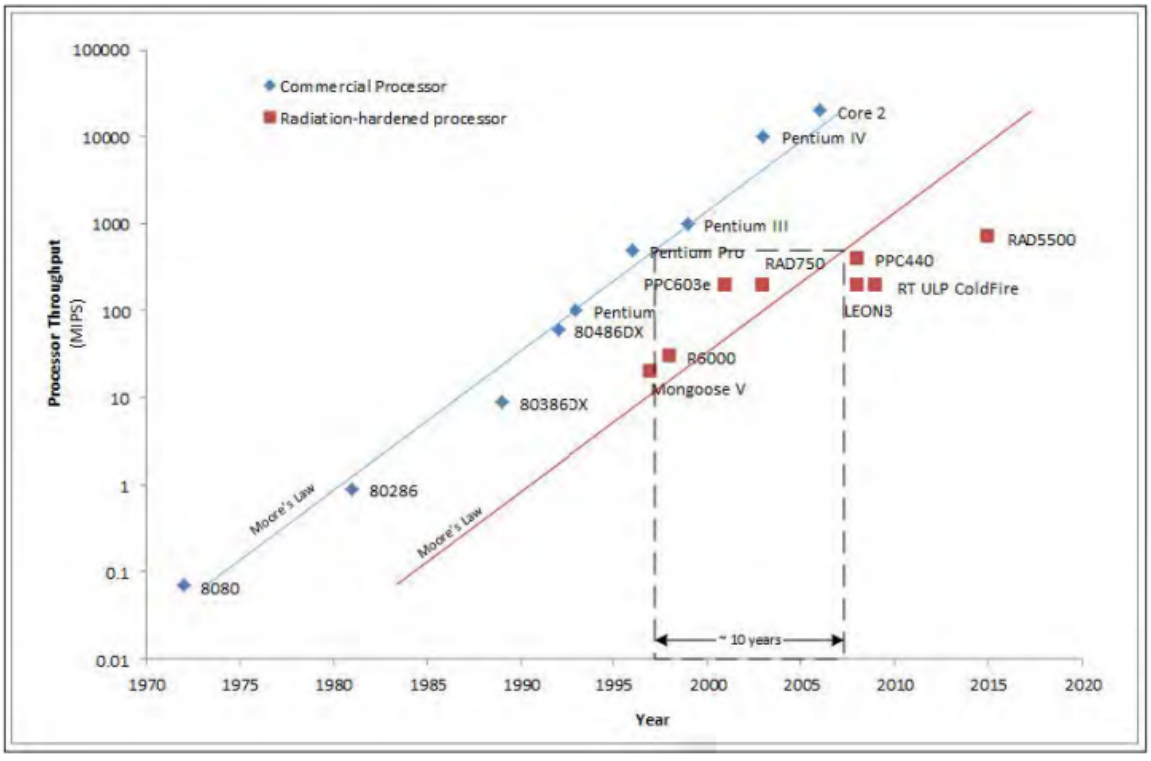

Figure 2.7: Performance comparison of radiation hard processors and commercial processors in the last 50 years [12]

Using FPGAs in the Harsh Environment of Space

A field programmable gate array (FPGA) device is a device that can be configured/reconfigured to meet a target design. FPGAs are programmed by the customer and not the retailer/manufacturer, making these devices more complex since the knowledge needed to accomplish the same task as a commercial processor is greater. FPGAs use a grid of logic blocks connected by configurable interconnects, allowing custom routing to create a wide scale of designs from simple logic gates to full featured processors. FPGAs have the capability to implement any function that an application-specific integrated circuit (ASIC) could perform with reduced prototyping development time. FPGAs have a feature called floor planning that allows for specific allocation of resources to chosen designs, which can help meet timing conditions.

Functionality of FPGAs

The two most well known FPGA manufacturing companies, Altera and Xilinx, have built into their products the option to include certain features to help reduce radiation effects on a design. Designs utilizing their soft error mitigation (SEM) controller and partial reconfiguration (PR) are two methods that MSU uses to combat the effects of radiation.

A SEM controller has the capabilities to act as a real time readback memory scrubber, being able to detect and correct soft errors within the active configuration memory on a FPGA. The controllers have several in-built modes that allow for error injection, error detection, error correction, and error classification. The SEM controller aims to eliminate as many soft errors from SEEs as possible to keep system reliability high. When implementing the SEM controller the design will have "essential" or "non-essential" bits, "non-essential" bits can be bit changes based on

active memory or stored data in registers in the logic field. Without this functionality the SEM would detect this stored data as erroneous data when compared to the "golden copy" and cause a system wide mitigateion response that would freeze system operations. Error correcting codes (ECC) and the subsection of error correcting codes called cyclic redundancy checks (CRCs) are used to detect errors in memory. The SEM controller can inject faults on demand to ensure normal error correction functionality is working as intended.

Partial reconfiguration of FPGAs allows a specified region to be reconfigured without interrupting or resetting the rest of the device. A FPGA's input/output connections, interconnects, and logic elements are referred as the fabric of an FPGA. The fabric of modern FPGAs can be divided into static logic and non-static logic. Regions of static logic can not be reconfigured while the non-static regions can be reconfigured through PR. When a PR is performed the controlling PR block will take a partial bitstream file that is generated for the region that the PR is happening on. PR will overwrite all of the non-static region that is selected by the PR controlling block without interrupting the rest of the fabric. Once the reconfigurable region is reprogrammed normal normal operation continues. Partial reconfiguration is an important aspect of combating the effects of radiation since it can reset a faulted region without needing a full reset. Saving time and possibly critical data on the design.

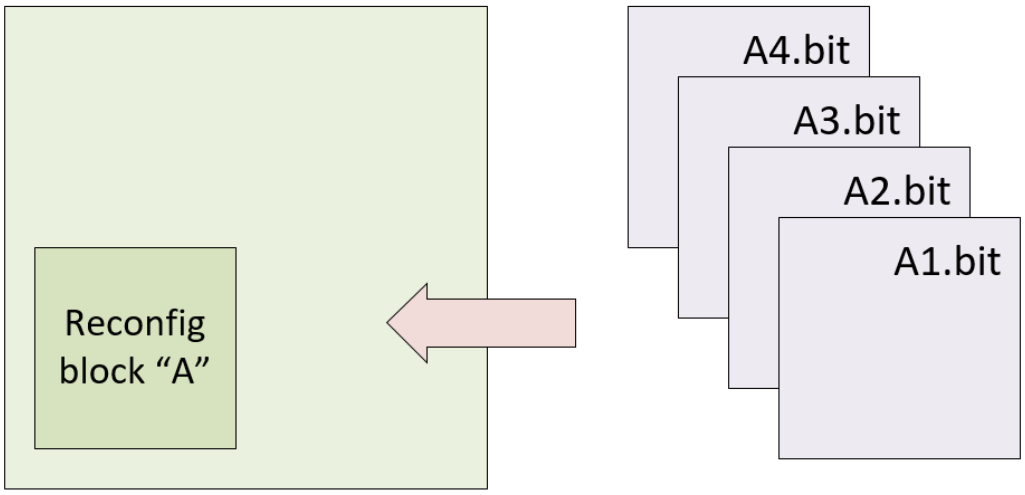

Figure 2.8: A visualization of partial reconfiguration with partial bitstreams [29]

Radiation Effects on FPGAs

Modern circuitry on electronics exhibit small feature sizes, creating less of a concern for TID damage. The oxide thickness is so small it becomes statistically improbably that a charge will get trapped. [3] Modern FPGAs are achieve TID tolerance levels of greater than 300krad when implementing a feature size of 65nm and as much as 600krad when implementing a 22nm node. [4] However, the small diffusion region in current devices increase SEE susceptibility because radiation strikes can carry enough energy change the state of an device. [21] This means that configuration memory or design memory can be altered by these strikes, causing corresponding logic elements to malfunction. These single events can cause bit flips in memory, leading to data corruption. Additionally the interconnects between logic blocks can also be effected by SEEs, either adding or removing a connection. Figure 2.9 shows a visualization of this incident.

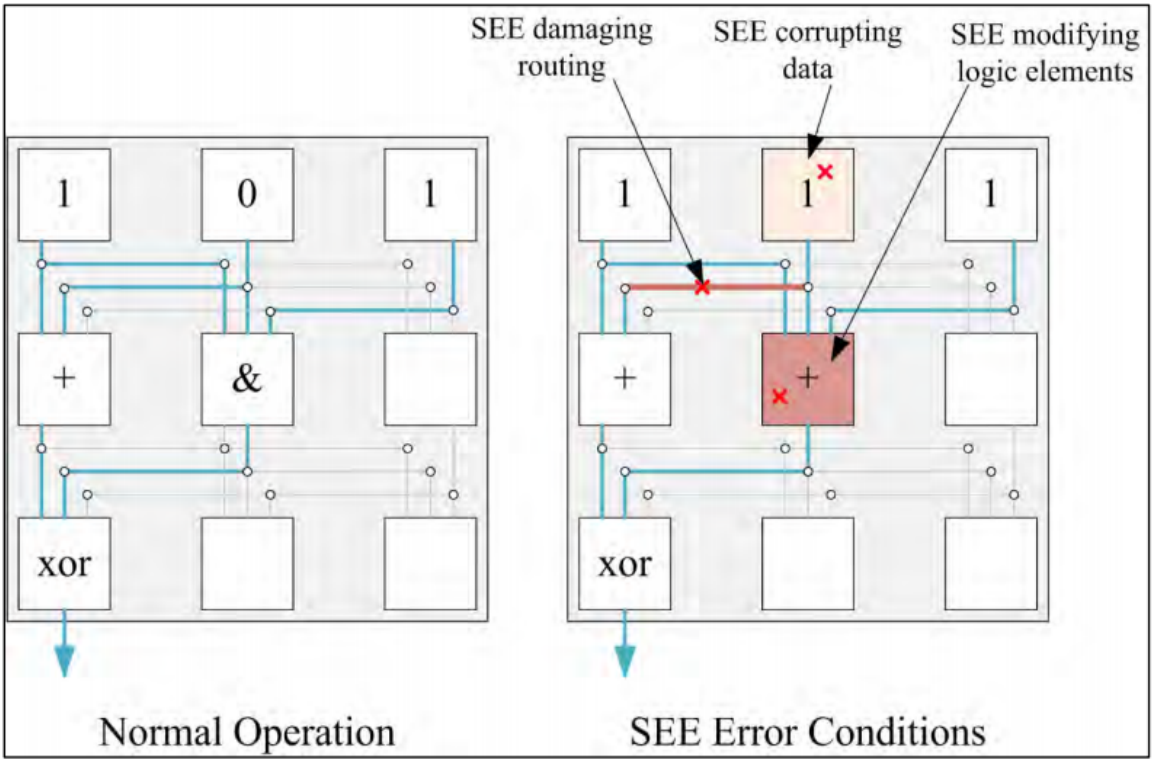

Figure 2.9: Normal FPGA operation compared to operation under SEE errors [22]

A Necessity for Fault Tolerant Computing

Faults caused by radiation in space combined with the needs of the space community provide strong a strong argument that fault tolerant computing is imperative to the advancement of space exploration. Fault Tolerant computing solutions are being researched at MSU to provide a solution to the problems faced during long-term space missions.

20

MONTANA STATES CONTRIBUTION

For the last decade MSU has been researching and developing an architecture to monitor and combat the effects of SEEs in space computing. The design started out on evaluation kits and has advanced to be flown on the ISS [3] and in free-fall low earth orbit in a self-sufficient 101030 cm or 3unit (3U) form. [16]. The next subsections will detail MSU's history of developing a radiation tolerant computing stack

Montana State University's approach

The primary platform for MSU's RTC research are on FPGAs developed by Xilinx. Xilinx FPGAs have a reputation of being used in military and space applications. Xilinx provides FPGAs that are low cost and readily available for consumer use, considered to be Commercial Over The Shelf (COTS) products readily available to universities through educational discounts and research programs. COTS FPGAs from Xilinx's 7th generation have a feature size of 28nm, which gives them inherent TID immunity as an added perk. The reconfigurable nature of a FPGA is key to MSU's RTC design, paired with low power usage and high computing power COTS FPGAs are ideal for prototyping MSU's novel architecture on. The RTC combats SEE faults by Triple Modular Redundancy (TMR) voting, real time scrubbing of the fabric using Xilinx's Soft Error Mitigation (SEM) controller to instill extra reliability into it's design. The design methodology is based on creating partitions of the fabric, each of which a soft-core Xilinx MicroBlaze processor resides within. The MicroBlaze is a 32bit Reduced Instruction Set Computer (RISC) available through Xilinx's software suite. Partial Reconfiguration allows the FPGA to reset and configure the fabric back to a working state after a SEU fault. The experimental FPGA contains 4 tiles

(current) and in the past contained up to 16 tiles. All of the tile identical to each other in how they access the fabric and each of their memory cells. At any one point in time 3 of the cells are running in a TMR setup with the remaining in reserve for when a fault happens. The TMR voter is constantly checking the outputs of the active tiles and when a output is misaligned the controlling process assumes that the active tile(s) output signifies a fault somewhere in the tile's fabric. The memory cells are also controlled by a separate TMR system that uses combinational logic to determine when the memory units have differing values and actively corrects any irregular data detected by the TMR system, allowing storage of critical data accumulated by the FPGA. When a tile is considered faulted a background process will run to repair and reset the tile through PR back to a working state, once the tile is repaired it is listed as a possible replacement tile for the next faulted tile.

TRL	Description
1	Basic Principles observed and reported
2	Technology concept and/or application formulated
3	Analytical and experimental critical function and/or proof-of-concept
4	Component and/or breadboard validation in laboratory environment
5	Component and/or breadboard validation in relative environment
6	System/subsystem model or prototype demonstration in relative environment (ground or space)
7	System prototype demonstration in a space environment
8	Actual system completed and "flight qualified" through test and demonstration (ground or space)
9	Actual system "flight proven" through successful mission operations

Table 3.1: The levels of NASA's TRL [18]

Technology Maturation

This section will detail the history of the development of the RTC at MSU. NASA gauges new technology on a measurement system called Technology Readiness Level (TRL). This system judges when a system can be flown safely on a mission or is ready for an industrial level of replication. Table 3.1 describes NASA's TRL scale and where the technology has to be in its development process before qualification.

The RTC design started in 2007 at TRL-1 and in 2010 it had advanced to TRL-2 and TRL-3. When TRL-3 was reached the design connected prototype boards to Xilinx evaluation boards. Figure 3.1 shows proof of concept to reach TRL-3. [3]

Figure 3.1: Breadboard prototype to eval boards –needs to be updated to a better resolution image for final

Testing at Texas A&M Radiation Effects Facility pushed the RTC to TRL-4. To qualify for these tests a 4"x4"x4" form factor was designed using custom Printed Circuit Boards (PCB). At this point the RTC system could be called Radiation Tolerant Computing Stack (RTCS). Two tests on the RTCS were performed in 2010 and 2011 with this new form. The form factor was mounted in a test fixture in the path of a cyclotron that bombarded the system with Krypton ion at 25MeV/AMU [3] This round of testing with the cyclotron confirmed integration of different system components and validation of testing in a laboratory environment, thus achieving TRL-4.

The next phase of the design was to test it on high altitude balloons to move towards achieving TRL-5. To fly in these conditions certain aspects of the payload had to be further developed. A power board was developed to regulate power from battery packs into the system as well as a data logging board to capture data for review on return to surface. Six of the eight balloons were through a student driven

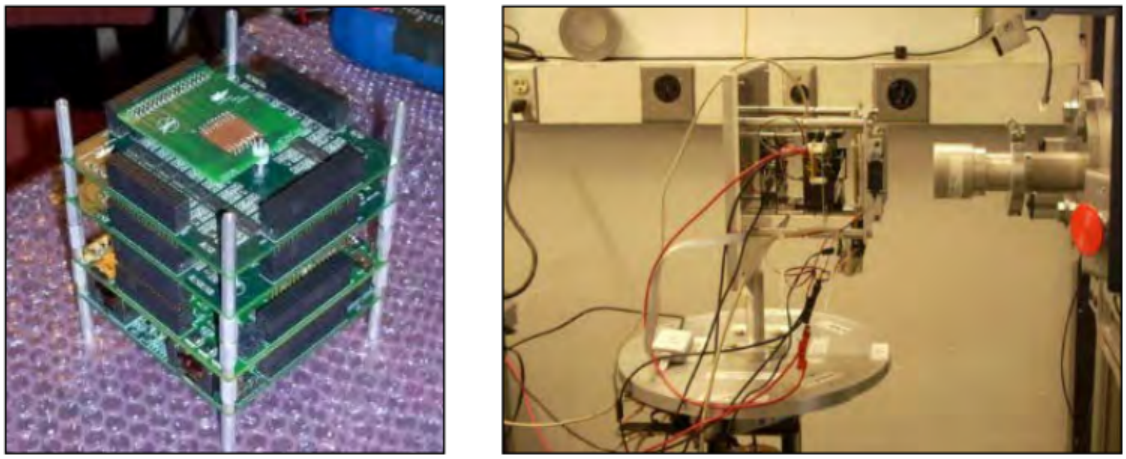

Figure 3.2: TRL-4 form factor(left), Cyclotron mounting (right)

program called Balloon Outreach, Research, Exploration and Landscape Imaging System (BOREALIS). A program ran by Montana Space Grant Consortium (MSGC). These balloon flights usually reached around 90.000ft before descent. The two other flights were performed by NASA's Columbia Scientific Balloon Facility. These flights achieved altitude of 120.000ft and took place in New Mexico. [3] These tests proved that the system could operate in harsh environments helping it achieve TRL-5.

Figure 3.3: BOREALIS balloon(left) and the CSBF balloon (right)

To achieve TRL-6 MSU chose to fly the system on a sounding rocket in 2014. The rocket vehicle was SL-9 operated by UP Aerospace LLC which achieved an altitude of 408,000ft(Figure 3.4). The form factor of the RTCS has to be redesigned for this test, which achieved a 1U form factor (10cm x 10cm x 10cm) [16]. This new design readied the RTCS for future sounding rocket missions as well as future ISS and cube-satellite missions. Attaining TRL-6 meant that the system and subsystems were validated in an end-to-end environment. TRL-7 was attempted but several hardware malfunctions prevented this achievement.

In March, 2015 the RTCS was flown on another sounding rocket down in Wallops, Virginia (Figure 3.5). This rocket required further testing before launch in the form of vibration testing and pressure testing after integration. A more solid deck platform was used to interface the RTCS in this rocket.

Figure 3.4: Up Aerospace sounding rocket demonstration [3]

Figure 3.5: Wallops sounding rocket(left), view from rocket in orbit (right)

On Dec. 14th 2017, the design had the opportunity to visit the International Space Station (ISS) with an designation of RTcMISS (pronounced "Artemis"). This successful test elevated the design to TRL-7 by successfully operating a prototype design in space.

Prior Work

Prior work at MSU included numerous Master's and Doctorate students continuously developing and improving the RTCS. Jennifer Hane developed a fault tolerant FPGA architecture to interface to a radiation sensor. [9] Justin Hogan worked on modeling reliability of various architectures. [11] Raymond Weber designed the Power board and ControlOS (the operating system controlling the Spartan 6 on the stack). [22] Sam Harkness designed a custom PCB to fit the 1U design and a data collection board. [23] Connor Julien designed the architecture that flew on Radsatg and interfaced the RTCS with the avionics support system for Radsatg. [3]

Current Testing Phase

The next stage of this design is to use error correcting codes to preserve critical data on a low earth orbit inside of a cube satellite of MSU's design. This is to be tested Radsatu which is set to be transported to the ISS in late 2019, with a release from the ISS shortly after. A successful test here will prove that the system is capable of operating in a harsh environment without losing data. Before this design can be complete an examination of several popular error correcting method will be examined in the following chapter.

Where this thesis ends is a plan to add another layer of fault mitigation to the previous design. This can be seen in Fig 3.6. The three faults types displayed in the flow diagram show the steps taken to mitigate each of those unique faults.

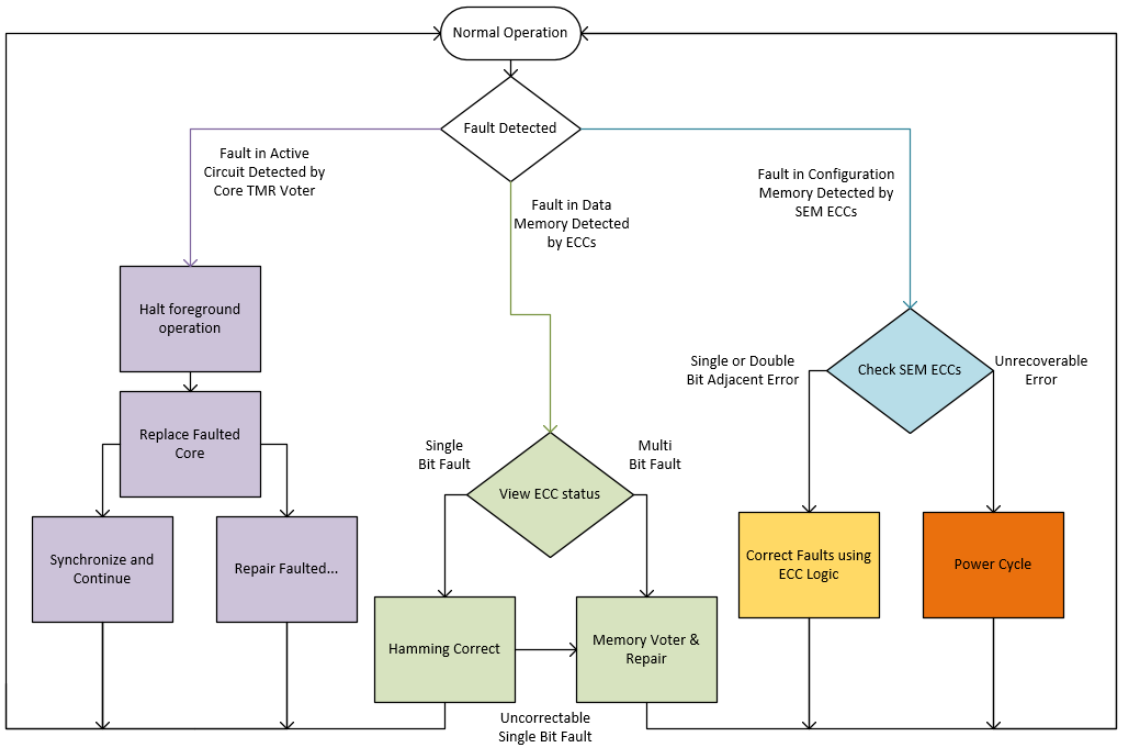

Figure 3.6: Fault Mitigation Flow diagram

First of the fault mitigation techniques implemented on teh RTC is the triple modular redundancy (TMR) method. As shown in Fig 3.7 the TMR method is designed to detect a fault in the active circuit. To do this a voter looks at the output of three identical circuits. If there is an outlier this is detected by the voter system and triggering the first step of this system. Halting foreground operation, which moves into Replace Faulted core with a space processor waiting for this specific incident. Once the core is replaced the system resumes operation after synchronizing all the processors. While the system is resuming after the faulty core is replace, a background repair method is ran to correct the faulted active circuit. This is accomplished by re-configuring the specific active circuit through Partial Reconfiguration.

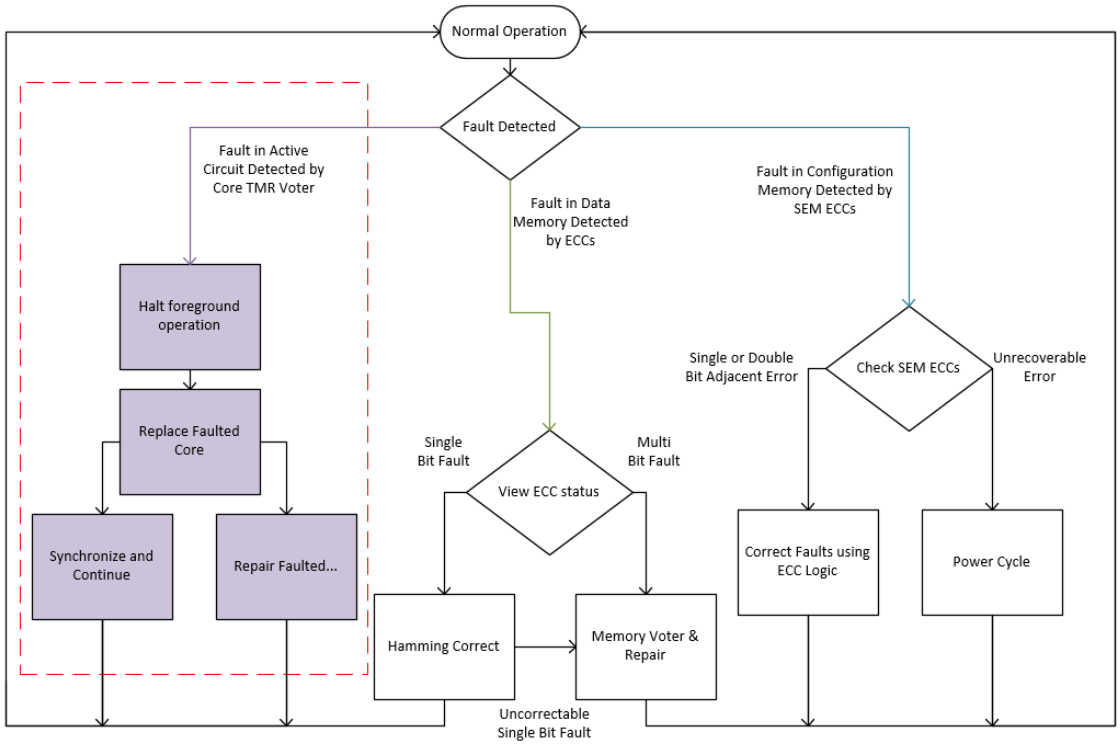

Figure 3.7: Fault Mitigation Using TMR

The second fault mititgation technique implemented on the RTC is by using Xilinx's soft error mitigation (SEM) controller. This controller creates ECCs for the configuration of the RTC. While its running it actively waits for an fault to occur in the configuration memory. Once this is detected the SEM can correct certain cases of these errors, single or double bit adjacent errors. If there is an error that can't be corrected by the SEM a power cycle will reset the configuration. A power cycle is the worst case scenario since this will effectively wipe out any critical data within the system.

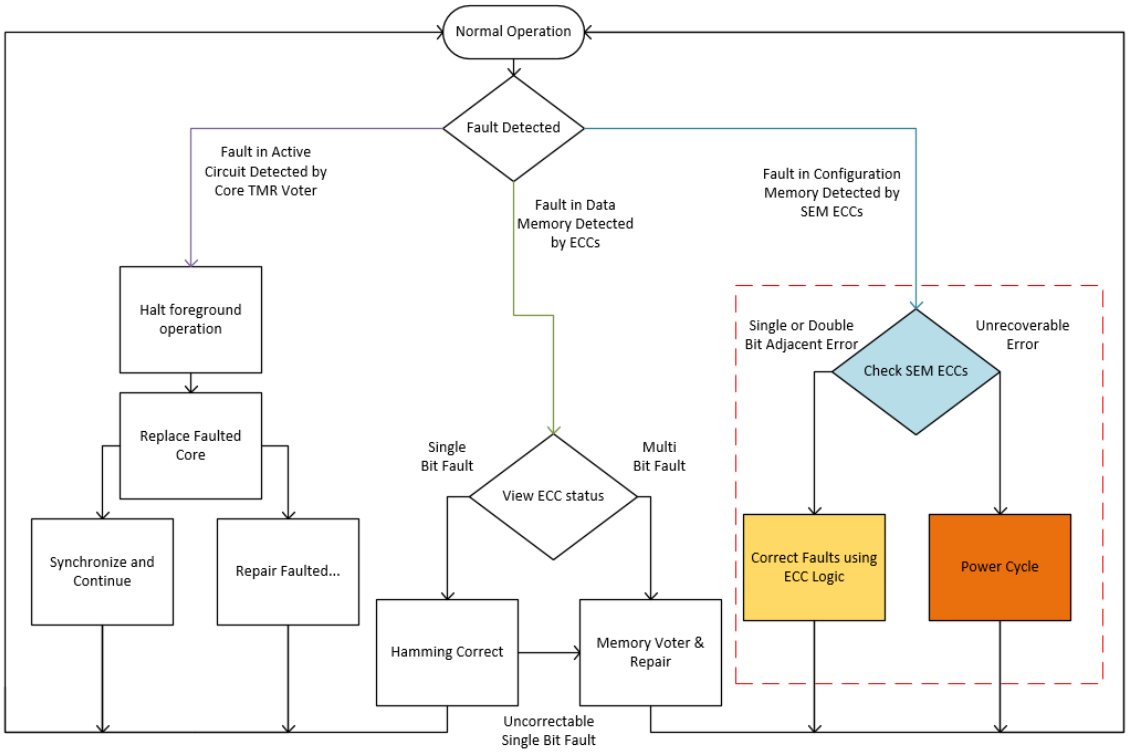

Figure 3.8: Fault mitigation using the SEM controller

My Contribution

What this thesis expanded on is the "Fault in Data Memory Detected by ECCs" fault mitigation method shown in Fig 3.9. This method looks at the memory space allocated to storing critical information from the processors that TMR would wipe out without it. This memory is still susceptible to SEEs, which is where the ECCs come in. When there is a single bit fault detected the Hamming codes will correct that flip, if this bit flip is uncorrectable due to the bit flip happening in the ECC parity bits instead of data the Memory Voter & Repair block corrects this by using a method similar to TMR. This block looks at all of the separate memory units and finds the outlier. If there is an uncorrectable bit fault or a multi bit fault, this block is able to correct this and allow the system to resume normal operation.

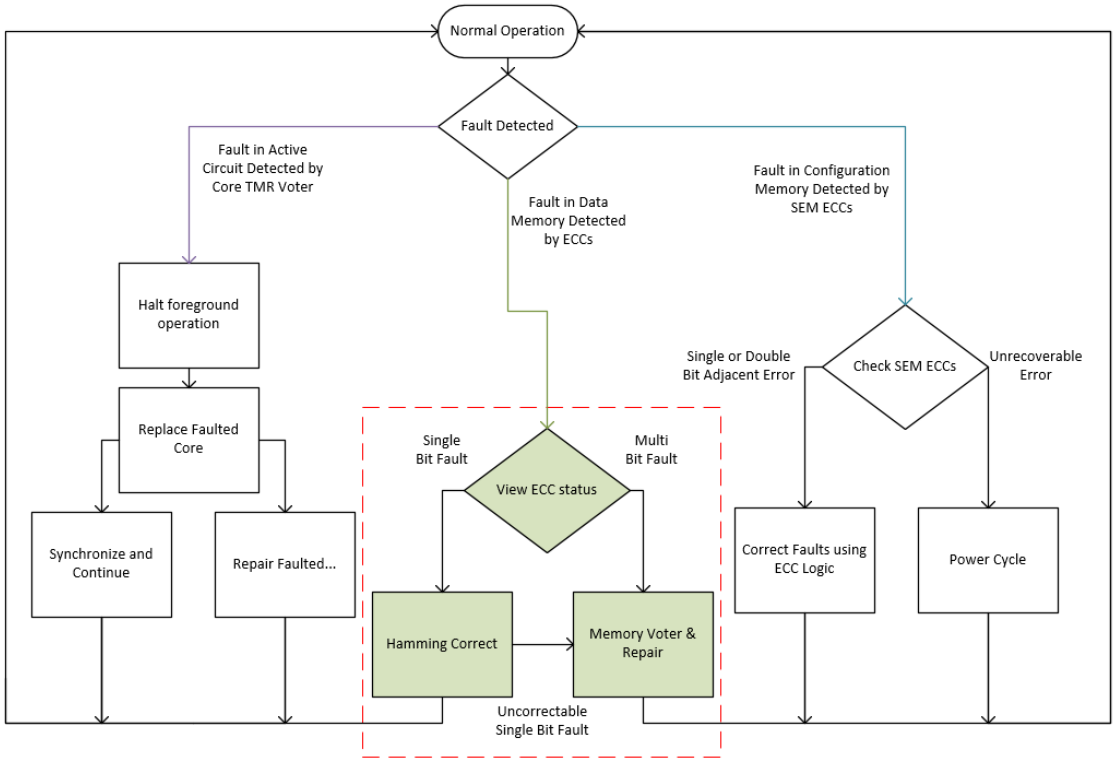

Figure 3.9: Fault mitigation using Error correcting codes in memory

THEORY

Error correcting codes (ECC) have been around since the late 1940's, first documented use was in the *Model V* computers Richard Hamming was using at Bell to relay information. [26] The codes Mr. Hamming created to check the data running through have been named Hamming codes. Being one of the first implementations of ECC and in the early age of computers this is one of the simpler existing methods. As the field of error correcting progressed, numerous other methods were created. The four methods this thesis will cover are Hamming codes, BCH codes, Turbo Codes, and Low-density parity codes. Each of these method have slightly different implementation and different capabilities.

Each of these error correcting methods has two main processes, encoding and decoding. The encoding process takes the existing data and creates a block of data called the parity-check for use in the decoding process. This process is usually the faster, less complex of the two.

The decoding process, which is usually more complex, can be broken down into two sections:

1. analyze the data with the parity bits and detect if there are any errors.

2. if there are error, correct said errors, then report position if needed.

When it comes to computational requirements the encoding process takes less resources/time to complete its task than the respective decoding process. This is usually the case because the encoding process usually takes a known set of data and know method or generator set to create parity bits. During the decoding process of the same method the known values are the encoded data and the method to check existing data against the parity bits generated during the encoding process. Some

method can be setup to report the error position after correction, while others can be setup to report how many errors were corrected or failure to correct the code. This can all be determined by the case that these codes are used.

The first of the method talked about in this thesis are Hamming Codes. This implementation is a non-systematic method, this means that the parity bits are mixed into the data string. A systematic code is separated into data and parity bits separately as shown in Figure 4.1. Systematic codes can have the parity bits on either side of the data bits, they just need to be separated from the data bits.

The rest of the methods discussed in this thesis after Hamming, in order, are BCH codes, Turbo codes, and LDPC codes. these codes are design to be systematic in these specific cases. With ECC a system can detect and correct for a certain amount of errors. The number of errors a system is capable of correcting depends on the capabilities of the method used.

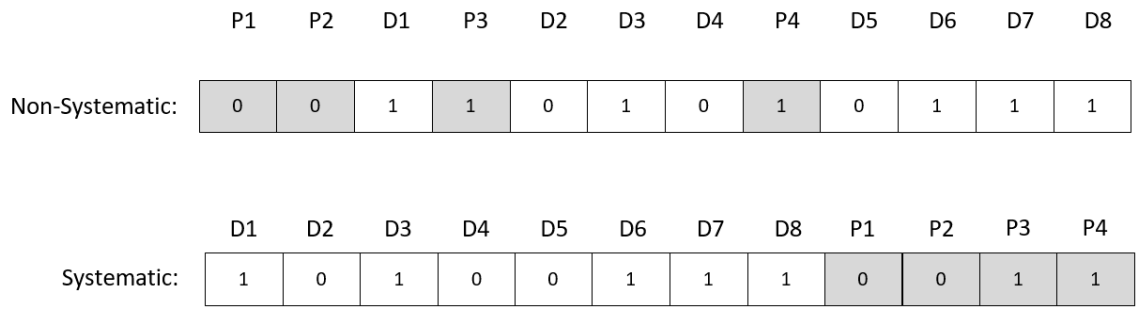

Figure 4.1: The top block is a non-systematic representation of encoded data Hamming(12,8), Bottom block is a systematic representation of the same data and parity bits Hamming(12,8)

Hamming Codes

Hamming codes were invented by a Bell Lab's employee Richard Hamming in the late 1940s. These codes were used to detect errors on the Bell Model V, which is an early computer design by Bell Labs to operate independently of an operator. When

these computers found an error, during work hours, it would flash lights and sound and during the weekends they would just continue through errors. Mr. Hamming decided that restarting his computer every time he found an error was something that could be avoided if the Hamming code itself could identify the position of the error and correct it. So in the 1950's he published what today is know as Hamming Code, a method still being used in devices today.

For this thesis the form of hamming codes mainly talked about are the Hamming(12,8) codes. This means that there are eight data bits and four parity bits, the number of parity bits allows for precise identification in memory is by using Error Correcting Codes (ECC). An example of the simplicity of Hamming code parity bit generation is shown in Fig 4.2 and an example snippet of sudo code is shown below.

```
%first hamming code Parity bit
hammingCode(1) = xor(xor(xor(xor(hammingCode(3), hammingCode(5)),
↪  hammingCode(7)), hammingCode(9)), hammingCode(11));
```

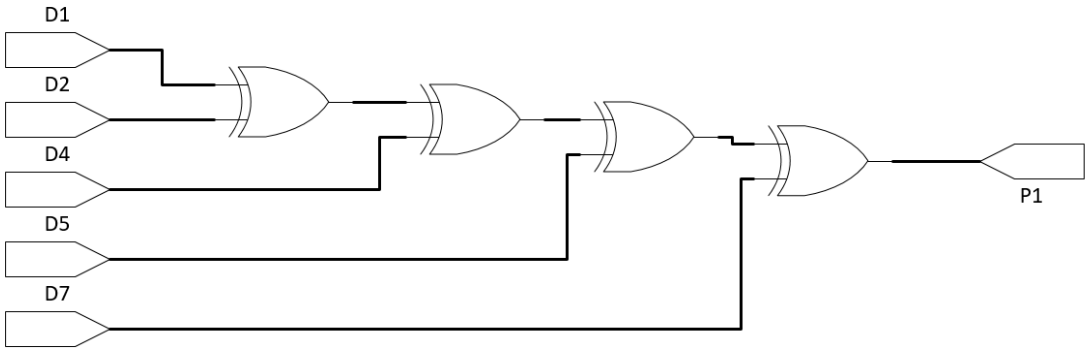

Figure 4.2: Encoding process of the 1st parity bit from a byte of data

Hamming Encoding

The main process of ending in Hamming is to xor data bits to find the required Parity bit. First before encoding a few key features need to be known. First, how much data is being encoded at a minimum. In this case 8 bits, next how many parity bits are needed. For Hamming codes, with single error correcting (SEC) capabilities, this can be found by looking at Eq. 4.1. The minimum amount of parity bits is based on the length of inputted data n, for example is n = 8 then m has to be, at a minimum, 4.

$$n \geq 2^m - 1 \tag{4.1}$$

A visual representation of all the parity bits being generated is shown in Fig. 4.3. The method discussed in this thesis is the Hamming(12,8) method, where there are 12 bits total and 8 of those are data bits as well as the figure that was referenced in the previous sentence.

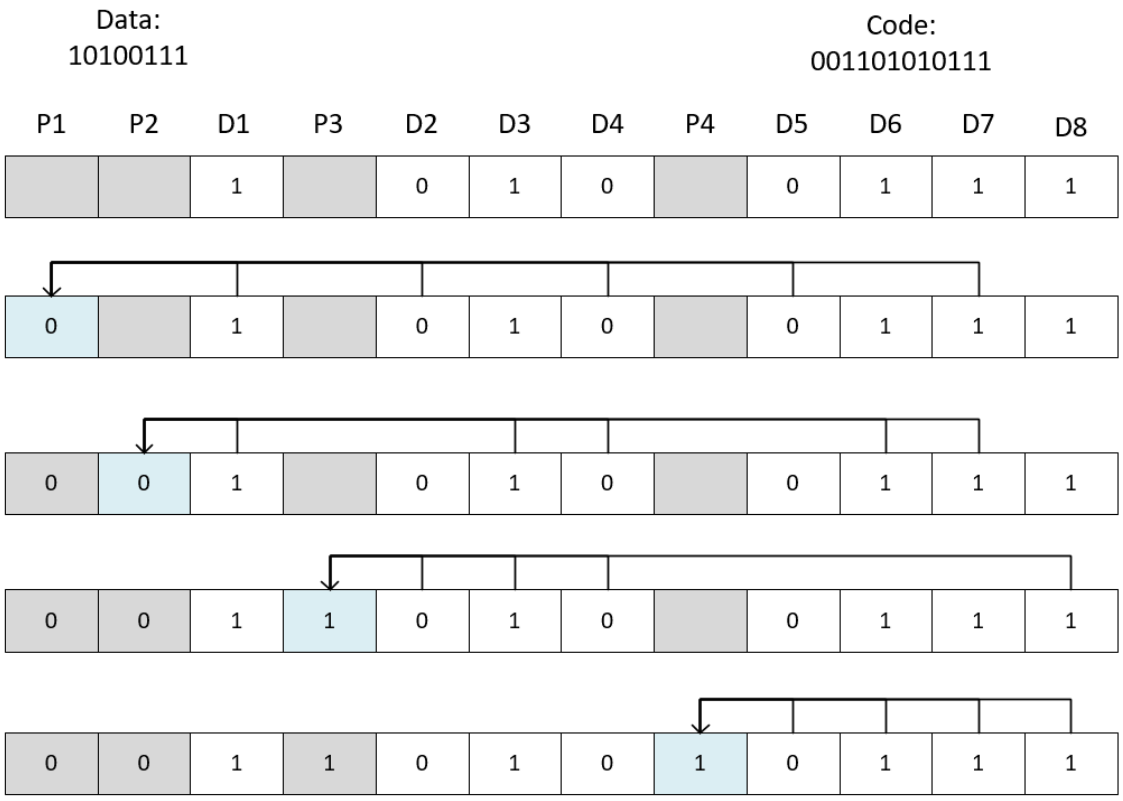

Figure 4.3: A non-systematic approach to generating Hamming parity bits for a (12,8) method

The easiest way to explain how parity bits are generated in Hamming codes is to show a visual of the non-systematic method. Equations 4.2, 4.3, 4.4, and 4.5 show the math operations to computing each of the parity bits for the given data. The $\oplus$ symbol is the XOR logical operation, which functionally simular to taking the modulo 2 of a summed value.

$$P1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \tag{4.2}$$

$$P2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \tag{4.3}$$

$$P3 = D2 \oplus D3 \oplus D4 \oplus D8 \tag{4.4}$$

$$P4 = D5 \oplus D6 \oplus D7 \oplus D8 \tag{4.5}$$

With the parity bits calculated the encoding part of Hamming is complete, the next step for this encoded data is to be decoded before use.

Hamming Decoding

The decoding process for Hamming codes uses the same process as the encoding to generate check bits to compare against the existing parity bits, this comparison implies a syndome vector is created. A syndrome vector is a set of values that represent the output of comparing either:

1. a set of parity bits vs parity check bits
2. a set of equations using the parity bits

In the case of this Hamming code method parity bits are being compared to the parity bits generated in the encoding method. As seen in Eq. 4.6, the syndrome bits are computed exactly the same way as the the parity bits are generated.

$$S1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \tag{4.6}$$

Knowing how to compute the syndrome bits can be used to the advantage of hardware implementation. Instead of storing directly the value can just be compared against the parity bit in a if statement to find any possible errors. The matlab function show in 1 shows an example of how to use the syndrome bits to correct any possible

errors. This listed code can be directly converted to hardware, an example of this in VHSIC Hardware Description Language (VHDL) can be seen in **??**.

Listing 1: Hamming Decode example

```
%detection
if(hammingCode(1) ~= xor(xor(xor(xor(hammingCode(3), hammingCode(5)),
↪  hammingCode(7)), hammingCode(9)), hammingCode(11)))
    parityErrors(1) = 1;
end

%correction
for x = 1:columns
  errorBit = errorBit + parityErrors(x);
end

%corrects the 1 bit flip in the hamming code
if(errorBit > 0)
    hammingCode(errorBit) = ~hammingCode(errorBit);
end
```

A visual of this code snippet can be seen in Fig. 4.4. This process just needs a few xor logic gates and a and gate to figure out if there is an error in the data bits exemplifying that this process can be done with small hardware footprint. An similar circuit would be built for each of the four parity bits in this hamming code (Hamming(12,8)).

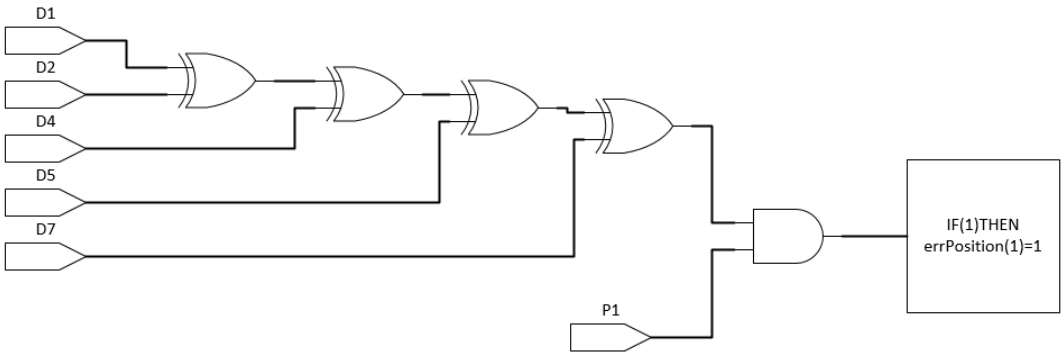

Figure 4.4: Example of checking parity bit

BCH codes

Bosw-Chaudhuri-Hocquenghem (BCH) codes are a widely used set of encoding methods. BCH codes are considered one of the most important cyclic codes in current use. Codes ($\mathbf{C}$) are considered cyclic if they can be shifted l times and display another code $\mathbf{C}$ within the code. Equations 4.7 and 4.8 show the primary aspects of a cyclic code. [8]

$$c = [c_0 c_1 c_2 ... c_{n-1}] \tag{4.7}$$

$$c = [c_{n-l} c_{n-l+l} ... c_{n-1} c_0 ... c_{n-l-1}] \tag{4.8}$$

When managing cyclic codes there are two important matrices that are needed to encode and decode BCH, Turbo, and LDPC codes. These are the generator and parity check matrices. For the parity check matrix one of the key features for defining a parity check matrix for a code can be defined by Eq. 4.9. This allows for identification

of errors if the resultant of this equation is equal to anything but zero.

$$c * H^T = 0 \tag{4.9}$$

The parity check matrix used in this thesis was generated through MATLAB 2018 using the bchgenpoly function, this function itself is outside of the scope of this thesis. This function, summarized, creates a generator matrix, that when transformed into a parity check matrix, fulfills Eq. 4.9. The generator matrix (**G**) and parity check matrix (**H**) are Toeplitz, which means that they follow the the equation 4.10. The $\varepsilon_{i,j}$ is the matrix element at the ith row and jth column. Example generator matrix 4.5 is shown next to an example parity check matrix 4.6. These two matrices are considered non-systematic matrices. In the explanation of BCH encoding a systematic generator matrix is used to encode systematic

$$\varepsilon_{i,j} = \varepsilon_{i-1,j-1} \tag{4.10}$$

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure 4.5: Example generator matrix [8]

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Figure 4.6: Example parity check matrix, paired with 4.5 [8]

BCH Encoding

BCH encoding methods can be done in a systematic style or a non-systematic style. This thesis will use a systematic style to encode data using a BCH method. The specific BCH encoding parameters used in this thesis are 11 data bits (eight actual data, three padded) and 15 total bits (BCH(15,11)). This method is capable of correcting 1 error in the data bits. Preferably, a eight bit method would have been chosen to align with the theory in the Hamming code section but for a clean method with nicely defined generator and parity check matrices. The closest setups would be four data bits, seven data bits, or 11 data bits. Using a method with less than the required amount of bit would double the required amount of hardware or completion time to accomplish encoding eight bits, plus some required storage registers/recombining circuitry. Meaning that if the eight data bits are split, there will be extra registers needed to maintain data between the two different encoding circuits. A more complex method will use more hardware but simplify the process of staging input data before encoding.

This means that the basic encoding process will have the original data embedded in the encoded output. Thus allowing for visual confirmation that the encoding process didn't change any of the existing data. By defining the type of encoding method beforehand the generator matrix can be created. The generator matrix is defined by the Eq. 4.11, either through a table lookup or by plugging in respective n,k values will end up with $1 + x + X^4$. [10].

$$g(x) = 1 + g_1 x + g_2 x^2 + ... + g_{n-k-1} x^{n-k-1} + x^{n-k} \tag{4.11}$$

The method described in this thesis uses a linear shift register with feedback connections corresponding to the coefficients of the generator polynomial as show in

Eq. 4.11. [8] The generator polynomial for this specific method is $1 + x + x^4$ [10], this information determines the generator and parity check matrices for encoding and decoding data in this BCH method, shown in 4.12 and 4.13 respectively.

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{4.12}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{4.13}$$

There are two different stages to the encoding circuit as seen in Fig. 4.7. The variable k represents a clock cycle from the beginning of the data bit being fed into the encoding circuit, c(x) is the code being inputted based on which bit the current clock cycle x was at.

- for clock cycles 1 to k, the information bits are transmitted in unchanged form (switch S2 in position 2) and the parity bits are calculated in the Linear Feedback Shift Register (LFSR) (switch S1 is on).
- for clock cycles k+1 to n, the parity bits in the LFSR are transmitted (switch S2 in position 1) and the feedback in the LFSR is switch off (S1 - off).

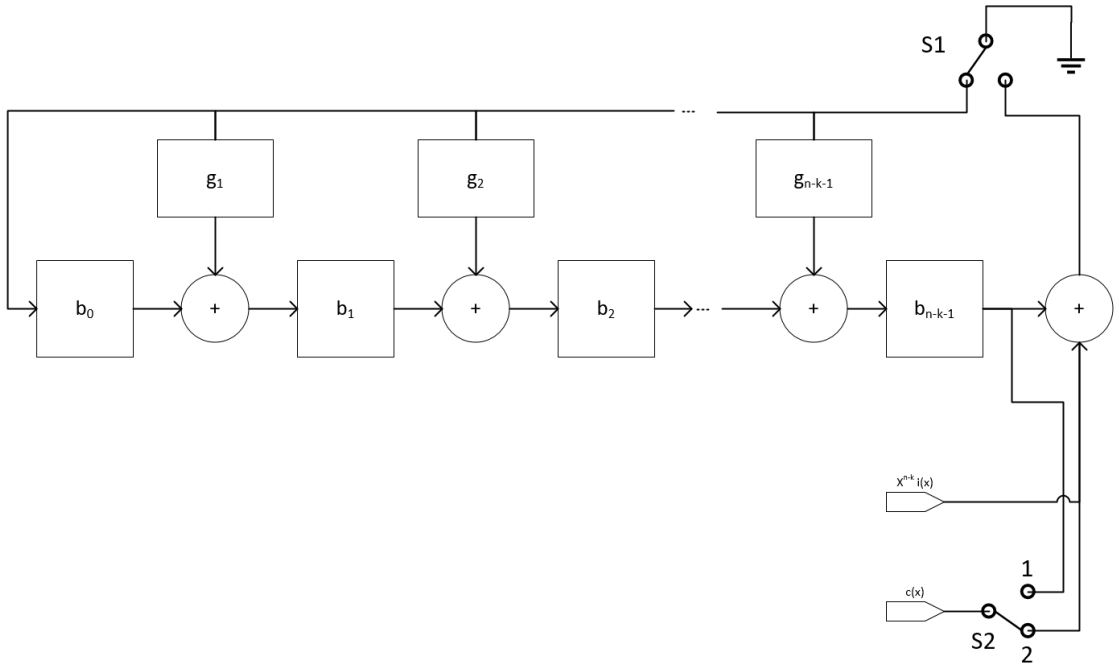

Figure 4.7: Encoding circuit for a (n,k) BCH code [8]

BCH Decoding

There are four main steps to decoding a BCH encoded word. Knowing the number of data bits, code length, encoding type (example: BCH(15,11)) are all things that need to be known before running through the four steps

1. calculate the Syndrome
2. Solving the key equation
3. Finding the error positions

Let there be known these four steps are designed for all BCH codes, whether single error correction (SEC), double error correction (DEC), or triple/more error correction (TMEC). Step 2 can be omitted from SEC and DEC since the syndrome will show the error position in the incoming code. For all other error correction methods using BCH all steps need to be accomplished. Since this thesis is looking at

single error correction in the rest of the listed codes, that is the method design this section will follow.

$$c(x) = c_0 + c_1x + c_2x^2 + ... + c_{n-1}x^{n-1} \tag{4.14}$$

$$r(x) = r_0 + r_1x + r_2x^2 + ... + r_{n-1}x^{n-1} \tag{4.15}$$

$$e(x) = e_0 + e_1x + e_2x^2 + ... + e_{n-1}x^{n-1} \tag{4.16}$$

Starting off with the required equations for calculating the syndrome. Eq. 4.17 shows three sets of equations: c(x) correlates to the incoming (encoded) message, r(x) is the received polynomial, and e(x) is the error polynomial such that their form is of Eq. 4.17. [8] Since this method has been designed for transmission over RF bands the $e(x)$ is the error noise that can infect the original data while being transmitted. In the case of this thesis $e(X)$ will take the place of possible SEEs from radiation.

$$r(x) = c(x) + e(x) \tag{4.17}$$

The first step of the decoding process is to store the resulting $r(x)$ in a buffer and to calculate the syndrome with using Eq. 4.18 with the following constraints

$$S_j = \begin{cases} \sum_{i=0}^{n-1} r_i\alpha^{i-j} & for(1 \leq j \leq 2t) \end{cases}$$

Also since $r(x) = c(x) + e(x)$ Eq. 4.19 is also true.

$$S_j = \sum_{i=0}^{n-1} r_i\alpha^{i-j} \tag{4.18}$$

$$S_j = \sum_{i=0}^{n-1}(c_i + e_i)\alpha^{i-j} = \sum_{i=0}^{n-1} c_i\alpha^{i-j} + \sum_{i=0}^{n-1} e_i\alpha^{i-j} \tag{4.19}$$

By the definition of BCH codes [8]

$$\sum_{i=0}^{n-1} c_i\alpha^{i-j} = 0 for (1 \leq j \leq 2t) \tag{4.20}$$

thus arriving at,

$$S_j = \sum_{i=0}^{n-1} e_i\alpha^{i-j} \tag{4.21}$$

Which signifies that the error polynomial is the only thing that will matter when calculating the syndrome. If the error polynomial is zero, then there are no errors.

$$S_i = r(\alpha^i) = r_0 + r_1\alpha^i + r_2(\alpha^i)^2 + ... + r_{n-1}(\alpha^i)^{n-1} \tag{4.22}$$

The syndromes can be generated by expressing Eq. 4.18 as Eq. 4.23

$$S_j = (...((r_{n-1} * \alpha^j + r_{n-2}) * \alpha^j + ...) * \alpha^j + r_0 \tag{4.23}$$

When using this equation (Eq. 4.23 for this method ($m = 4$ and generator polynomial $p = 1 + x + X^4$) the circuit of this equation (Fig 4.8) shows a high level visual of the calculation of the syndromes for this BCH method. S_0 through S_3 represent the 4 syndromes needed to detect the position of an error. Just like how Hamming codes detect the position of an error. All of the Syndromes are initialized to zero and the recieved (r(x)) is clocked into the circuit 15 times to shift all values of recieved code into the syndrome circuit.

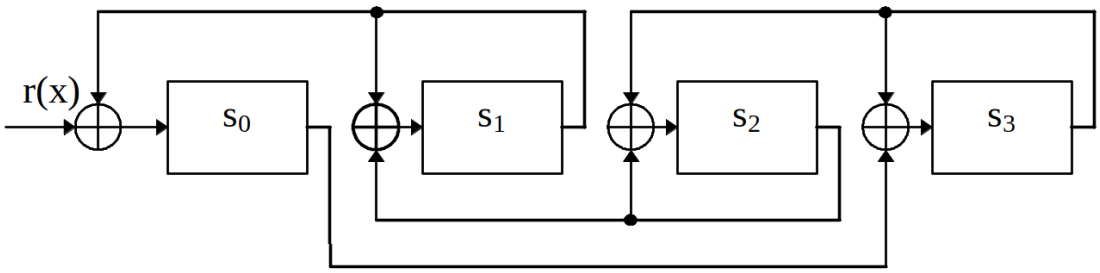

Figure 4.8: Syndrome computation circuit for m=4 [8]

Since the BCH method used in this thesis is a SEC method the second step (Solving key equation) will not have much discussed. As a high level overview of what would happen for a TMEC code the coefficients for the error locations (σ) would be calculated by using a Peterson-Gorenstein-Zieler algorithm, Euclid's algorithm, or Berlekamp-Massey algorithm [8] [10] [26].

Instead for a SEC method the error positions polynomial can be represented by Eq. 4.24. This simplifies the process of finding the error locations substantially.

$$\sigma(x) = 1 + S_1 x \tag{4.24}$$

Once the error location polynomial ($\sigma(x)$) is found the third step of the BCH algorithm can be used to find and correct erroneous data. This is accomplished by finding the reciprocal roots of $\sigma(x)$ by substituting: $1, \alpha, \alpha^2, ..., \alpha^{n-1}$ into $\sigma(x)$. A common method in BCH schemes to do this is the Chien search algorithm (Eq. 4.25.

$$\sigma_0 + \sigma_1 \alpha^j + \sigma_2 \alpha^{2j} + ... + \sigma_t \alpha^{tj} \tag{4.25}$$

The range of Eq. 4.25 is $t = 0, 1, 2, ..., 15$ and $j = 0, 1, ..., k-1$. The Chien search algorithm is evaluated on every clock cycle, if $\sigma(\alpha^j) = 0$ then that receive bit r_{n-1-j} is corrupted. Therefore if for the clock cycle j the sum is equivalent to zero

the receive bit r_{n-1-j} should be flipped.

A visual representation of the Chien search algorithm is shown in Fig 4.9. This circuit is initialized by setting the registers $c_0, c_1, .., c_t$ to the coefficients of the error location polynomial σ. The sum $\sum_{i=0}^{n-1} c_j$ is calculated, if the result of this is zero. The corresponding error bit is found and corrected after a being delayed through a buffer and corrected with an XOR gate. On each proceeding clock cycle the c_j registers are multiplied by their corresponding α_j before another summation is completed. This whole operation is repeated $t = 15$ times for this specific method.

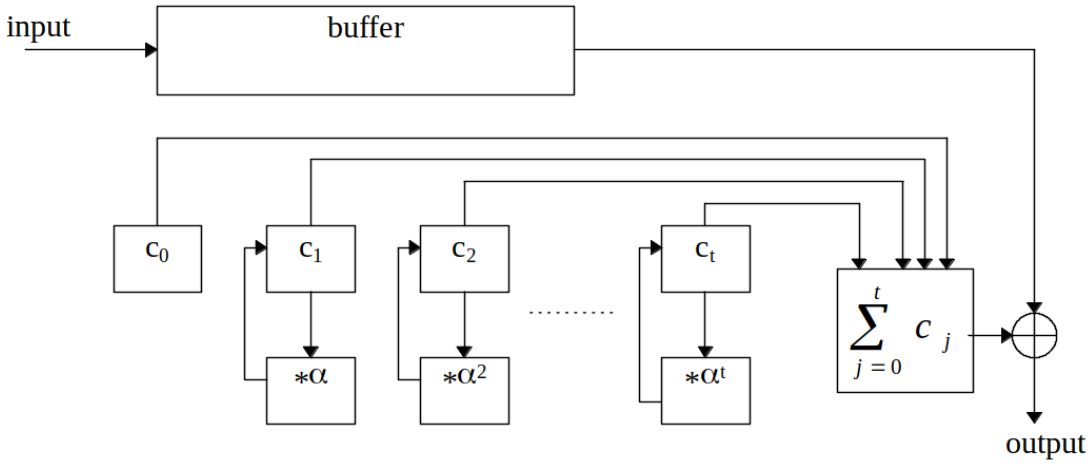

Figure 4.9: Chien's search circuit

The main portion of the encoding process for C_2 and C_3 only requires shift registers and xor gate to accomplish its encoding.

For a BCH method that corrects more than one error, the corresponding circuits for step 2 and step 3 will be much more complex. SEC is the simplest method for using BCH codes and this is the smallest the area of hardware usage can be for this method. The results section will detail resource usage for this method in more detail.

Turbo codes

Developed in 1993 [10], turbo codes performance benchmarks passed most previous FEC block methods. Known for closely approaching the channel capacity. Turbo codes are a popular method for safely transmitting data between RF systems.

Turbo encoding

Turbo encoding is a method that takes up quite a few more registers for the output. By looking at Fig 4.10 the design shows that there are three outputs for one set of data. One of the outputs will have the original form, making this method systematic. The C_2 and C_3 codes are sent through a feed-forward/backwards register system to encode the data. The third set of encoded data is ran through an interleave process, which reorders the code in the index positions shown in Fig 4.10.

Listing 2: Example Turbo encoding using Matlab

```
for i = 1:8 \\
    tmp = xor(msg_in(i),xor(reg(1),reg(2)));
    c(i) = xor(tmp,reg(2));
    reg = [tmp reg(1)];
end
```

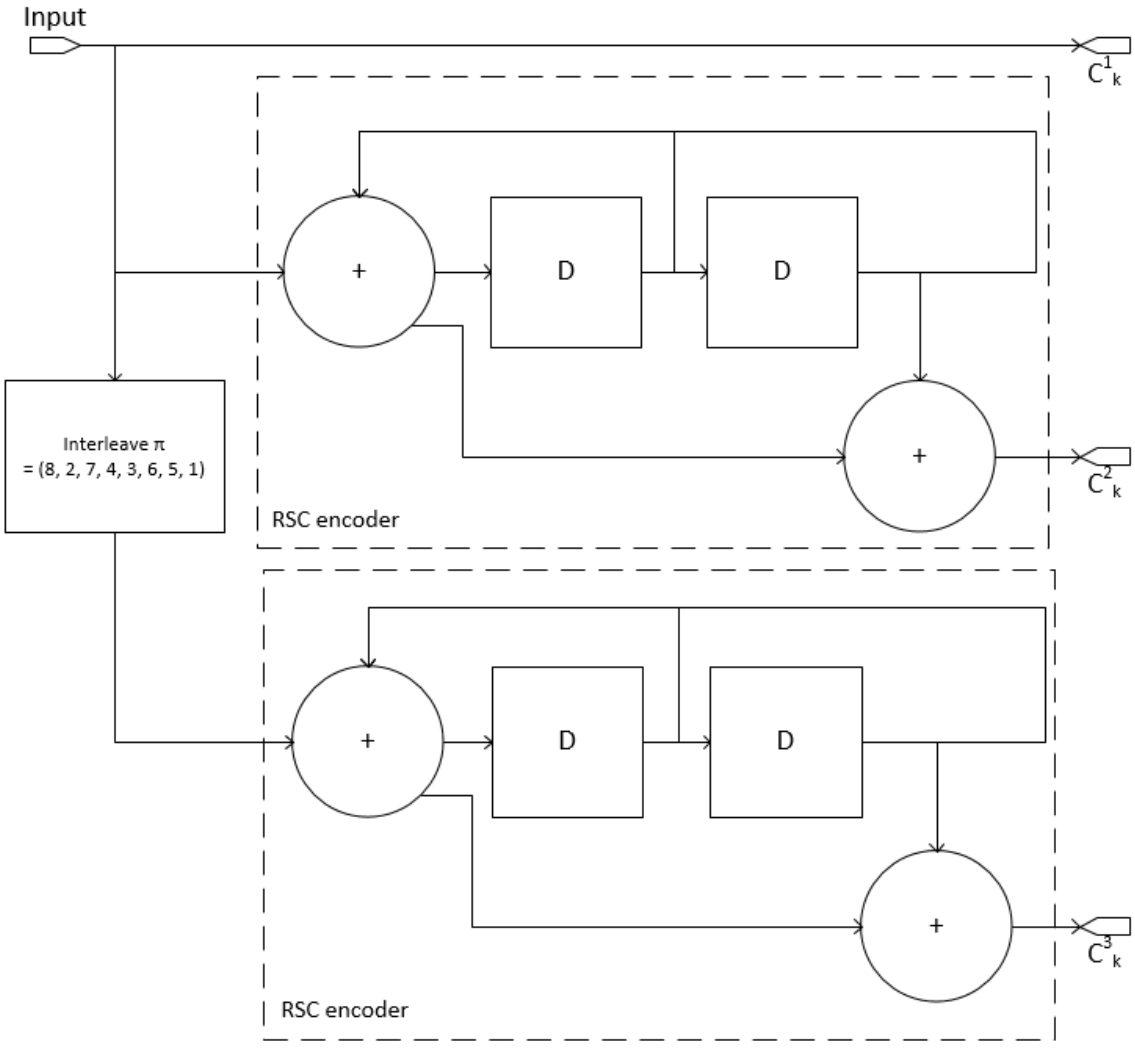

Figure 4.10: Encoding structure for Turbo codes [10]

The code in Listing 2 shows the Turbo encoding method in matlab. The input code is ran through a recursive systematic convolutional (RSC) encoder as shown in Fig 4.10. The method that a simple RSC encoder uses to create the parity bits for a Turbo code is shown in Listing 2. Once the code is processed by the Encoding stage of this method there will be triple the original amount of bits for transmission. For example if a input of the form [1 0 1 0 1 1 0 0] the Turbo encoding process shown would result in the matrix shown in Eq. 4.26. As can be seen in the first

row of the output (Eq. 4.26) is the original input data, the next two rows show the post-encoding process parity bits.

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{4.26}$$

Turbo Decoding

There are three key steps to decoding a Turbo code

1. Decoder 1: Compute $L_1(C_k^{(1)})$ and de-interleav
2. Decoder 2: compute $L_2(C_k^{(1)})$ and interleav
3. If decoding converges or iteration count is reached, stop and make a hard decision to output.

These steps are shown in the visual representing Fig 4.11.

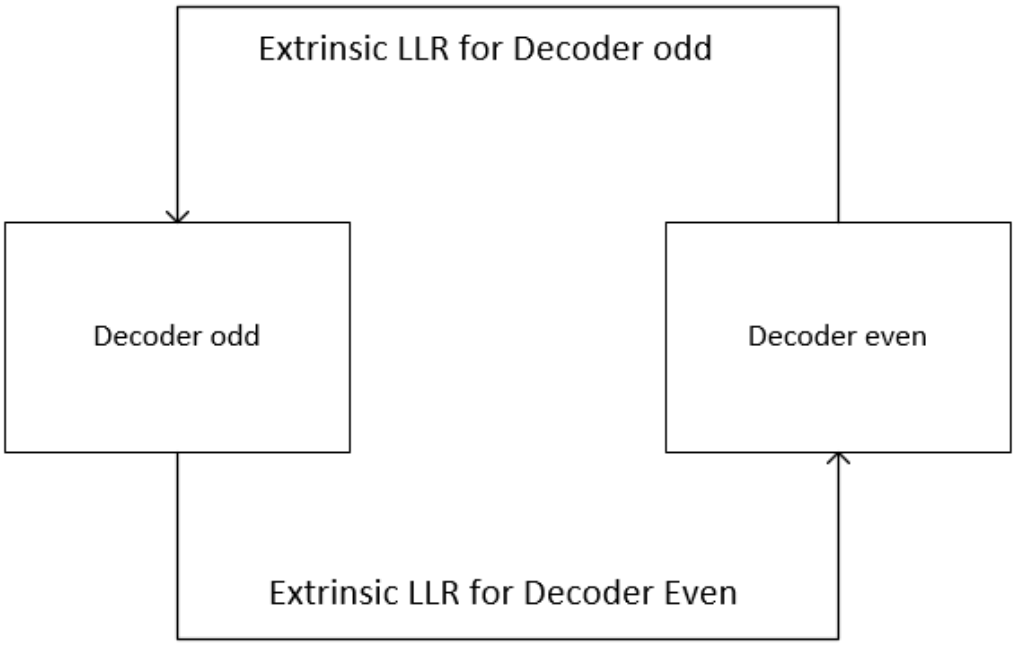

Figure 4.11: A high level visual of turbo decoding [10]

The first of these steps can be accomplished by looking at the inputted code and taking the encoded data and splitting into its original data (the encoding is systematic) and separating the two different parity bit sets to split between the decoder 1 and decoder 2 process. Part of the decoder 1 process is to take the data and reshape it for processing through a inbuilt function designed by Dr. Y. Jiang. An example of this reshaping process is shown in the matrices show in Eq 4.27 (original) and Eq 4.28.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.27}$$

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{4.28}$$

The in build function is called a logmap function. This function takes an input and runs the input through a set of $log(e^{abs(var1-var2)})$ functions that converge the approximation values toward, ideally, the correct value the more times that this function is ran. This condition only works if there is an alternation between Decoder 1 and Decoder 2. This process depends on a convergence happening. If the process shown in Fig 4.11 never converges the output shouldn't be trusted.

While this process is used widely in transmission reliability, there are two key functions used that are drawbacks to using this method on an FPGA. The log function and exp function used in the logmap function. This will be discussed in more detail in the results section of this thesis.

Low-Density Parity codes

The fourth of the encoding methods examined in this thesis, low-density parity codes (LDPC) were discovered in the early 1960's and originally called Gallager codes. This encoding method didn't come into use until the mid 1990's. As with the other encoding methods this is a high level observation of the required theory needed to complete this version of a LDPC coding scheme. After discussing Turbo codes earlier in this chapter the read should be aware there are a few advantages to using LDPC codes compared to Turbo codes. [10]

- Low error floor
- Superior burst error correcting capabilities

While these are good there are a few drawbacks to using LDPC codes compared to Turbo codes.

- High encoder complexity
- Interconnect in decoder is large and irregular

The main thing that this thesis is concerned about is hardware use vs time, this is assuming that the decoding process can correct an error(s) every time there is one detected. For Turbo codes the error rate was extremely high, for this method the low error floor attribute that a major factor in determining the results of using this method.

LDPC encoding

A high level examination of LDPC encoding shows that the process is a fast and quick process. The main part of the encoding process relies on a generator matrix to create parity bits for unique data input. The generator matrix is based on the parity check matrix that will be needed in the decoding stage of the LDPC codes discussed in this thesis. The equations to make a parity check matrix for a LDPC 4/7. This method has four data bits and three parity bits making each chunk of encoded data being seven bits long. The parity bits are generated from equations 4.29, 4.30, and 4.31 by setting the resulting *E1,E2,E3* equal to zero and solving for the corresponding *C5,C6,C7*. This is the hand method for creating parity bits for LDPC codes. For using these equations with any input a generator matrix has to be created with these parameters. First the parity check matrix is made directly from the equations listed below, based on the bits of the resulting code on the right side of each of the parity

check equations, shown in Eq. 4.32.

$$E1 = C1 \oplus C2 \oplus C3 \oplus C5 \tag{4.29}$$

$$E2 = C1 \oplus C2 \oplus C4 \oplus C6 \tag{4.30}$$

$$E3 = C1 \oplus C3 \oplus C4 \oplus C7 \tag{4.31}$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{4.32}$$

Once the parity check matrix is found with the corresponding parity check equations, a generator matrix can be created from cutting out the identity matrix and rotating the matrix and restoring the identity matrix to the form of Eq. 4.33. The generator matrix is design so that the parity check matrix can pull any error out of the encoded data.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{4.33}$$

Once the generator matrix (4.33)is found the inputted data can be ran through Eq. 4.34 to find the corresponding encoded output. Notice that the generator matrix has an identity matrix making up part of of the matrix as a whole, this shows that the generator matrix is designed to be systematic. Thus the original inputted data

can be seen in the first four bits of the encoded data.

$$encodedData = (inputdata * GeneratorMatrix)\%2 \tag{4.34}$$

Figure 4.12 shows an visual representation of the encoding process for a LDPC method.

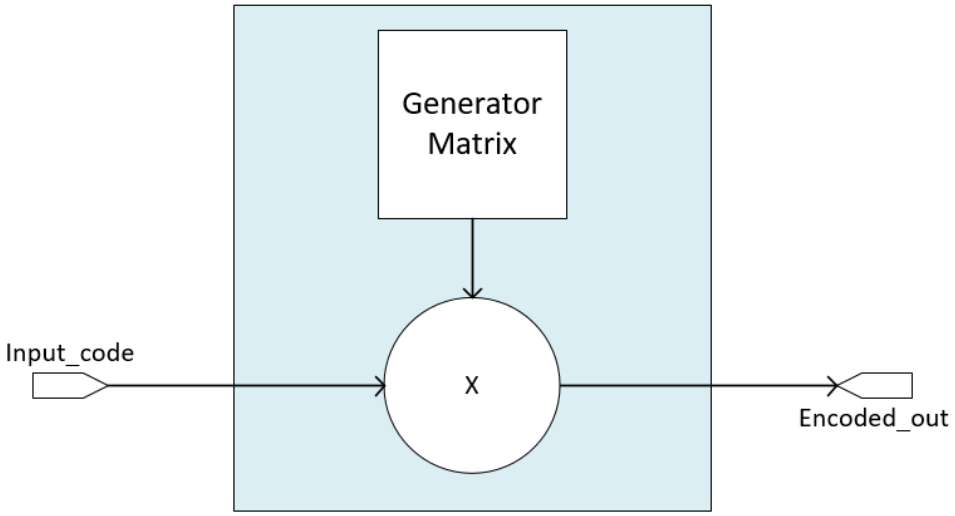

Figure 4.12: LDPC encoding method using a generator matrix

LDPC decoding

LDPC codes have been found to have multiple different method for decoding any LDPC code as long as the parity check matrix and code parameters like code-length and data length. These method can be parallel or sequentially designed, for this thesis a "bit-flip" (BF) method will be used to decode LDPC codes for the reason that this method allows for quick implementation into the hardware with a low amount of resources usage. There are four main steps to the BF method:

1. check parity matrix equations against encoded data

2. identify position of erroneous data

3. correct erroneous data

4. repeat until negative values are non-existent

The first step is as simple as using the parity check matrix (Eq. 4.32) to find which equations (Eq. 4.29, 4.30, and 4.31) reported a non-zero value (Eq. 4.35). These reported values can be multiplied against the parity check matrix to find an resulting positional vector showing the location of any erroneous data (Eq. 4.36).

$$checkError = encodedData * parityCheckMatrix^T \tag{4.35}$$

$$errorPositions = checkError * parityCheckMatrix \tag{4.36}$$

Once the location of the erroronous data is found the rest of the operation is just to flip those bits and repeat the prior steps (1-4) until completion of a predetermined number of iterations.

A hi-level representation of the LDPC decoder used in the process discussed in this section is shown in Fig. 4.13.

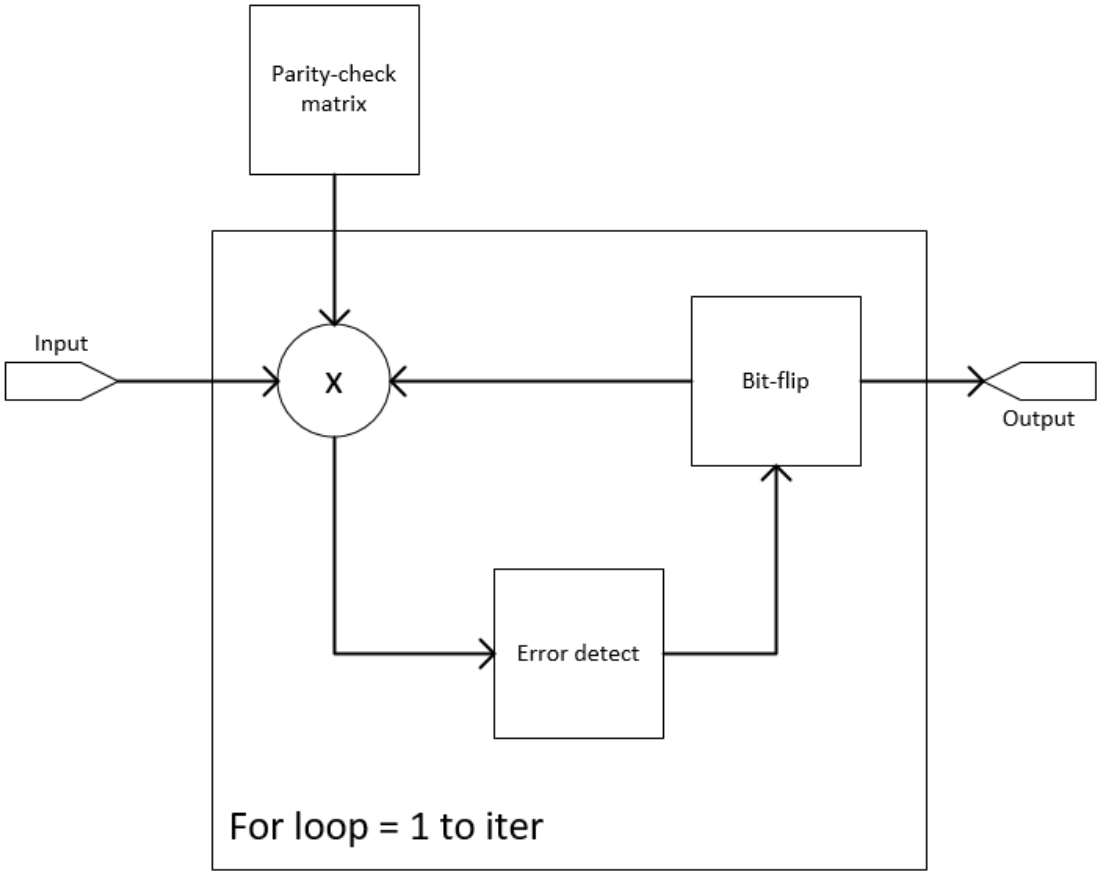

Figure 4.13: Block diagram of a LDPC decoder

RESULTS AND EXPERIMENTS

This section is for the results of the synthesis and implementation of each ECC method, detailing the required resources to implemented the discussed methods. The main features of each method that is under inspection are

1. Error correction Capability
2. Speed of execution
3. Area used

In each of the sections below there are a resource utilization and timing results. All of these values are compiled into a table at the end of this section in Table 5.7. A figure of merit, based on the listed features above, will also gauge the viability of each of the methods as well. This will determine which method will be used in Radsatu.

There are a few key terms that will pop in the tables below. LUTx: Look Up Table (x=inputs) and FDRE : single D-flipflop with an data(D), clock enable(CE), and synchronus reset(R), and multiplexer (MUXn) (n=inputs)

Hamming code

Hamming ECC requires only XOR logic gates to implement both the encoding and decoding processes in hardware. A look up table is used in place of logic gates to aid in timing of the system.

Hamming encoding

The encoding process takes 2ns on a 100MHz to complete its task as shown in Fig 5.1.

Resource Type	required
LUT4	2
LUT5	2

Table 5.1: Resource utilization of Hamming encoding

Table 5.1 show the required logic elements needed to run the Hamming encoding method in FPGA fabric. As seen in later methods this is a relatively small usage of resources.

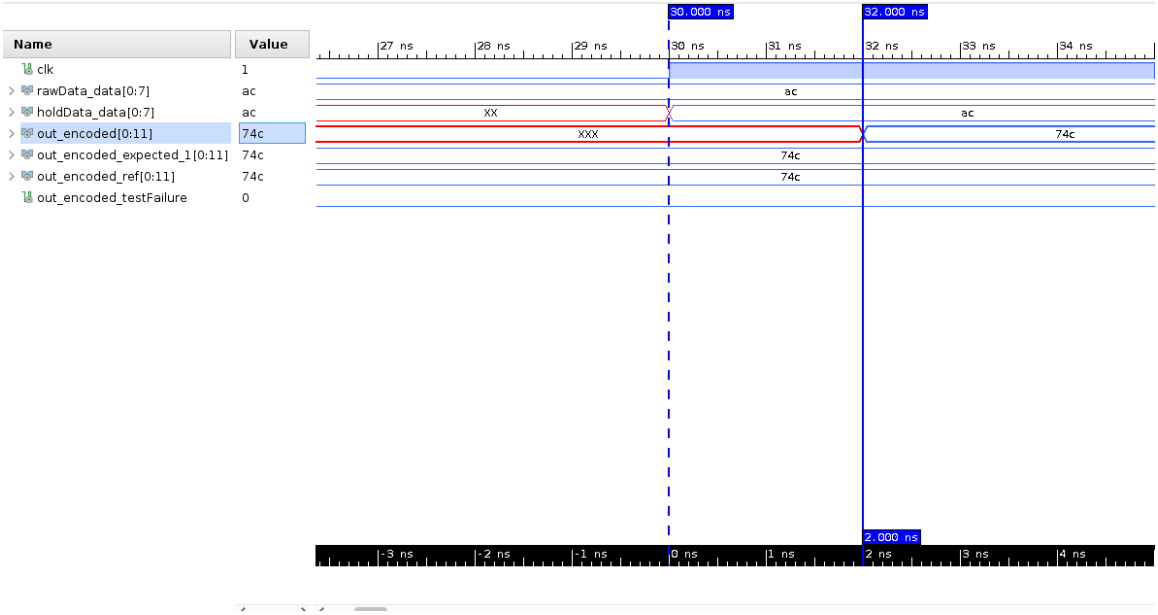

Figure 5.1: Timing for Hamming encoding

Hamming decoding

The process to run through the hamming decoding process is 2ns with a 100MHz clock, as seen in Figure 5.2.

Table 5.2 shows the resulting required resource utilization for the Hamming decoding process used in this design.

Resource Type	required
LUT5	2
LUT6	2

Table 5.2: Resource utilization of Hamming decoding

Figure 5.2: Timing for the decoding process with hamming codes

BCH Code

Implementing BCH codes in hardware was quite the task. This coding scheme seemed to be suited for large data lengths. The implementation used in this thesis was based on a single bit at a time shift method. Thus needing a clock cycle per bit going through the shift registers, plus a few more for bit management.

BCH encoding

BCH encoding is completed decent amount of hardware resources (relative to Hamming) as shown in Table 5.3.

The encoding process took 18ns as shown in Fig 5.3. This is due to the method shifting bits into a shift register as previously mentioned.

Resource Type	required
LUT1	12
LUT2	4
LUT3	1
LUT6	1
FDRE	4

Table 5.3: Resource utilization of BCH encoding

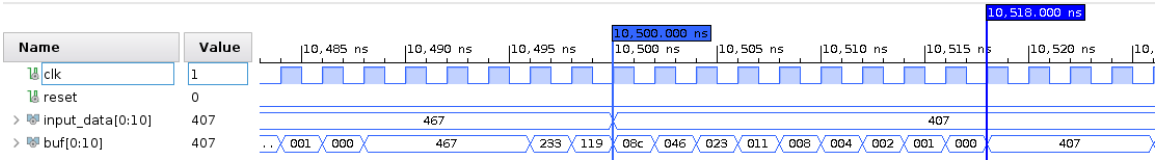

Figure 5.3: Encoding timing for BCH codes(18ns)

BCH decoding

The decoding stage of the BCH algorithm implemented in this thesis require significantly more resources to build compared to the encoding process, the resources required are shown in 5.4.

The BCH decoder took 66ns to complete decoding, as shown in Fig 5.4. The clock speed for the encoding and decoding process of this BCH method was 100MHz.

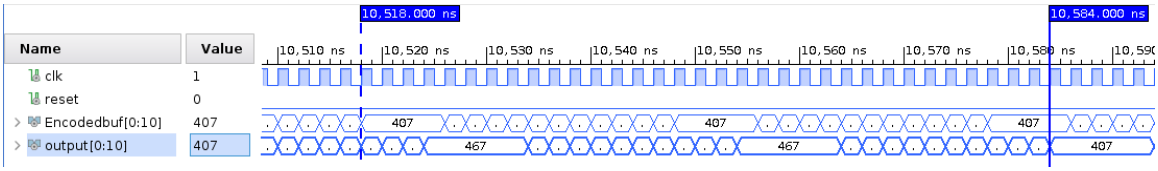

Figure 5.4: Decode timing for BCH method (66ns)

Resource Type	required
LUT1	12
LUT2	8
LUT3	23
LUT4	3
LUT6	6
FDRE	46

Table 5.4: Resource utilization of BCH decoding

Turbo Code

The results for Turbo ECC have been left out of this section for 2 key reasons.

1. Required math functions
2. Area usage

These two topics caused extensive issues with converting to vhdl with Matlab's hdl-coder tool. The first problem (math functions) is that the method being used required a several log function calls and several e^x calls. Each of these functions calls require an lookup table with enough resolution to maintain the data resolution being passed through. Which leads to the second issue, where in Matlab each of these functions required a 2^13 resolutions, which in terms of a lookup table requires a lookup table capable of storing 2^13 values. In Xilinx's current generation of FPGAs, which are only capable of 2^6 lookup tables at max, would require numerous linked look up tables to accomplish this task (128 lookup tables). This usage of look up tables eliminated this method from being used in the radiation tolerant computing stack.

LDPC encoding

LDPC codes were extremely simple to implement in hardware, the process of setting up the parity-check bits was very similar to setting up Hamming codes, just a little simpler. As shown in Table 5.5 LDPC encoding is a simple process, the data set is fed in and ran through Eq. 4.34. The timing of the LDPC encoding process is shown in Fig 5.5, having an instantaneous output in simulation. When implemented on a real system a 2ns delay could be assumed due to the average setup time on a logic gate

Resource Type	required
LUT3	3

Table 5.5: Resource utilization of LDPC encoding

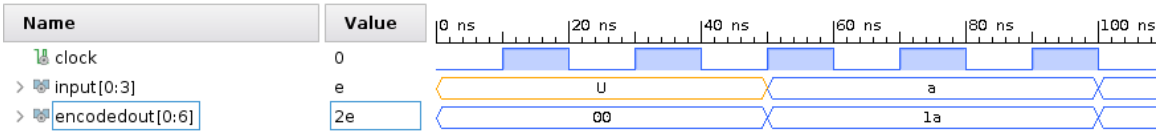

Figure 5.5: Encoding timing for the LDPC coding block

LDPC decoding

The decoding process of LDPC is a little more complex than Hamming decoding, The timing for this process is extremely short, which is suitable for real time use. While this process is short, the implementation used ends up using a large amount of look up tables (LUT) as shown in Table 5.6.

Resource Type	required
LUT2:	7
LUT3:	5
LUT4:	5
LUT5:	36
LUT6:	529
MUXF7:	14
MUXF8:	5

Table 5.6: Resource utilization of LDPC decoding

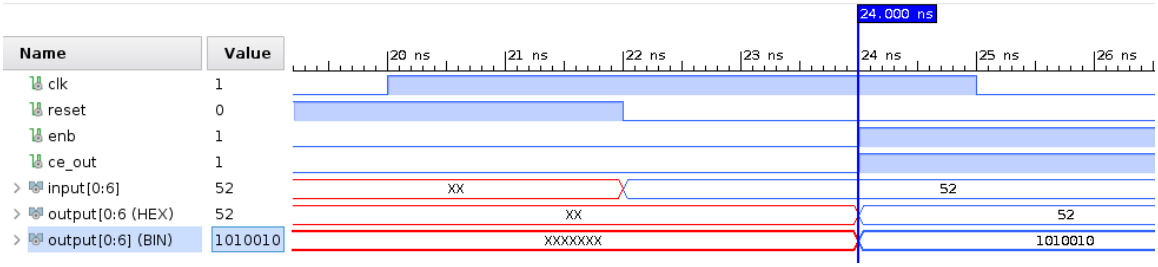

Figure 5.6: LDPC decoding Timing

Summation of resource usage

Table 5.7 shows the different resources for each of the different methods. As a reminder of references to each of the acronym shown in the table: Look up table(LUT), D Flip-Flop with Clock Enabled and Synchronus Reset (FDRE), adn multiplexer (MUX). This information can be abstracted to show how many slices on the FPGA are used. A slice is a section of the FPGA that has a set amount of resources contained within it. Each slice, on a 7-Series device, contains 4 LUTs, 8 Flip-flops, 1 arithmetic carry, 128 bits of RAM, and 64 bits of shift registers. [28] With this relation, we can approximate how many slices are used.

Method	Resource usage	Speed
Hamming Encoding	LUT4: 2 LUT5: 2	2ns
Hamming Decoding	LUT5: 2 LUT6: 2	2ns
BCH Encoding	LUT1: 12 LUT2: 4 LUT3: 1 LUT6: 1 FDRE: 4	18ns
BCH Decoding	LUT1: 12 LUT2: 8 LUT3: 23 LUT4: 3 LUT6: 6 FDRE: 46	66ns
LDPC Encoding	LUT3: 3	2ns
LDPC Decoding	LUT2: 7 LUT3: 5 LUT4: 5 LUT5: 36 LUT6: 529 MUXF7: 14 MUX8: 5	2ns

Table 5.7: Total resource usage, based on Xilinx synthesis tools. Each of the resource's fan in value is indicated by the number following the resource. (Example: LUT4 = 4 input look up table). Clock rate for this table is 100MHz

Table 5.8 shows the resulting slice usage for each of the methods described in this thesis except Turbo encoding/decoding. This is due to the method of Turbo decoding requiring a much larger amount of resources compared to the other methods, without the decoding process the encoding process is useless.

Method	Slice Usage
Hamming Encoding	1
Hamming Decoding	1
BCH Encoding	6
BCH Decoding	14
LDPC Encoding	1
LDPC Decoding	146

Table 5.8: Slice usage for each method, Each slice, for a 7-Series Xilinx FPGA, contains 4 LUTs, 8 flip-flops, 1 arithmetic/carry chain, 128 bits of RAM, and 64 bits of shift registers [28]

Now that a metric for how many slices are used, an approximation of area based no the FPGA that this method is used on. For this thesis these methods are going to be used on an Artix 7 200T, which by referencing Table 5.9, shows that this particular device has 33,650 slices available on the device. Using this as a maximum number of resources we can predict the amount of area taken up with each method (by percentage). This is shown in Table 5.10 in percentage. It can be noted that these values are very small.

Using the percent area defined in Table 5.10 and the basic size of one Artix 7 200T chip ($6250mm^2$) a table can be created to gauge how likely the chance of ionizing radiation striking a section of the FPGA that is currently active. To do this a few things need to be established first. What are the chances that our device will be struck at all, how big is this strike, and how often. For the first question, it can be assumed that at some point our device will be struck somewhere on the device. As for how big is this strike, since this strike will usually be ionizing radiation. We can assume the size of an electron, which is $2.82x10 - 15m$ [7]. Since our feature size

7 Series FPGA	Total Slices
7A12T	2,000
7A15T	2,600
7A25T	3,650
7A35T	5,200
7A50T	8,150
7A75T	11,800
7A100T	15,850
7A200T	33,650

Table 5.9: Each of the listed FPGAs are of the Artix 7 Series family, known for their high performance per watt

(28nm) is much larger than this we can assume that only one feature will be struck if the strike is normal to the surface and up to 60 deg from the surface normal. [11] Third, how often, using a fault rate of $1.02E^{-15}ms$ [11] a value of .93312 strikes per day on the whole device, at a minimum. Combining all of these factors with the % area displayed in Table 5.10 a minimum strike frequency for each of the methods can be estimated. This is shown in Table 5.11, which shows that the method with the highest slice usage has the highest chance of being struck by the .93312 strikes in a day.

Method	Area (%)
Hamming Encoding	2.9e-3
Hamming Decoding	2.9e-9
BCH Encoding	1.7e-2
BCH Decoding	4.1e-2
LDPC Encoding	2.9e-3
LDPC Decoding	4.3e-1

Table 5.10: Each of the area entries shows the percentage of slices used on a Artix-7 200T FPGA, this value is based on only using this method with none of the other slices being used for any other design.

Method	Strike Chance (%)
Hamming Encoding	2.77e-3
Hamming Encoding	2.77e-3
BCH Encoding	1.66e-2
BCH Decoding	3.88e-2
LDPC Encoding	2.77e-3
LDPC Decoding	4.04e-1

Table 5.11: This table displays an approximation of the active circuit being faulted by an ionizing strike to the device (Artix 7 200T). This values is only for the methods described in Table 5.10

Choosing an ECC method

To decide on which method to implement on the next cube satellite in design at MSU a *figure of merit* (T) had to be established with all of the methods. Eq 5.1 is the *figure of merit* that will be used in this thesis to pick which method will be used on Radsatu.

$$T = \frac{(ErrorCorrectionCapability) * Speed}{Area(slices)} \tag{5.1}$$

In Eq. 5.1 the Error Correction Capability variable represents how many errors each ECC method can correct for. The categories for this variable are single error correction (SEC), double error correction (DEC), and triple or more error correction (TMEC) are gauged in a linear scale based on how many errors can be corrected. The higher the number the more errors that can be corrected by this method.

The Speed in Eq. 5.1 represents how fast the method can take a block of information and complete its process on that block. This variable is represented on a inverse scale, the smaller the number the slower the process.

Finally the Area in Eq. 5.1 is based on the number of slices that are needed to implement the designs that were discussed in this thesis.

Using these metrics, a *figure of merit* for each of the methods can be created. This is seen in Figure 5.7, the values shown are based on the values found in Table 5.12.

Method	Error correction capability	Speed (ns)	Area (slices)
Hamming Encoding	1	2ns	1
Hamming Decoding	1	2ns	1
BCH Encoding	3+	18ns	6
BCH Decoding	3+	66ns	14
LDPC Encoding	1	2ns	1
LDPC Decoding	1	2ns	146

Table 5.12: Figure of merit table, based on the values found in development of this thesis. Clock rate for this table was 100MHz

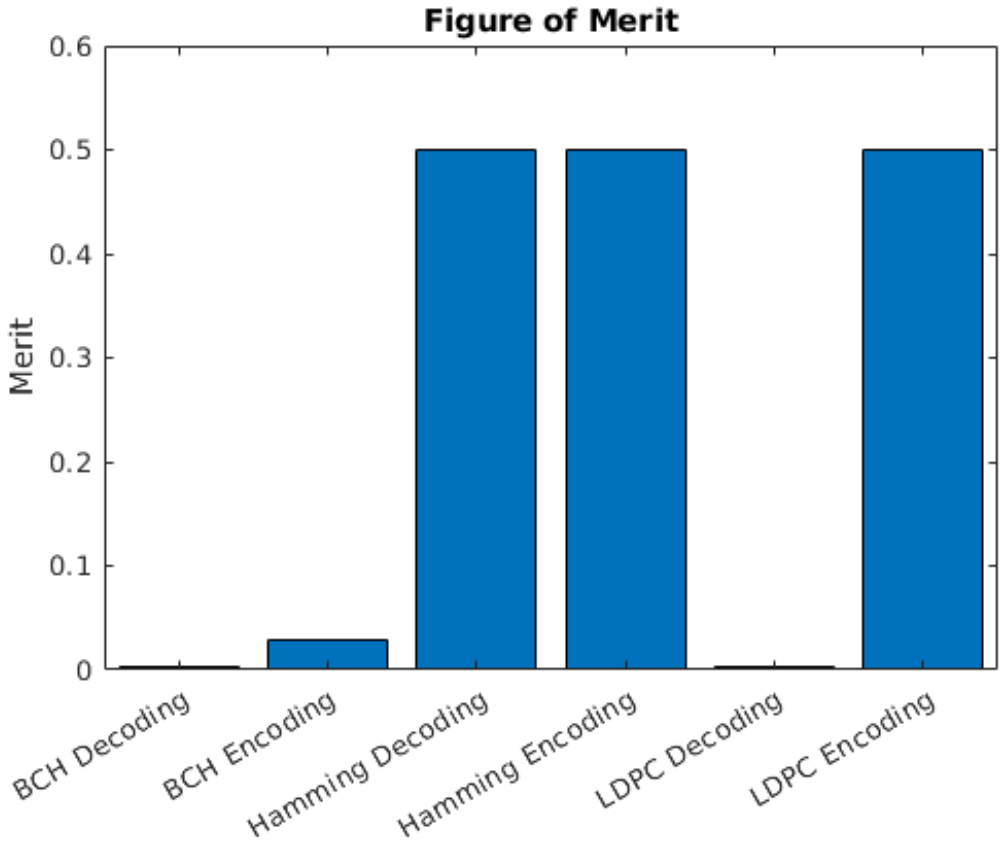

Figure 5.7: A gauge of merit for each of the viable methods, using metrics detailed in Table 5.12

As seen in Figure 5.7, the best scores (highest) are Hamming Encoding, Hamming Decoding, and LDPC Encoding. Since LDPC Decoding scored so low on Figure 5.7 the LDPC Encoding method is eliminated from choice of usage in Radsatu. This leads to having a final high scoring methods of Hamming Encoding and Decoding. This solidified the decision of using Hamming Codes for ECC on Radsatu.

Radsatu choice

The ECC method of choice for Radsatu happens to be Hamming Codes. After generating a *figure of merit*, the Hamming Code method had the highest combined figure of merit. Now that a method has been chosen, this method can be implemented on the Radsatu's memory as seen in Fig. 5.8. This design encodes data as it is stored to memory, when the data is accessed from the memory unit it goes through a decoding block to check the parity bits against the data. If there is an error, this is corrected and the data is pushed to the DataOut port.

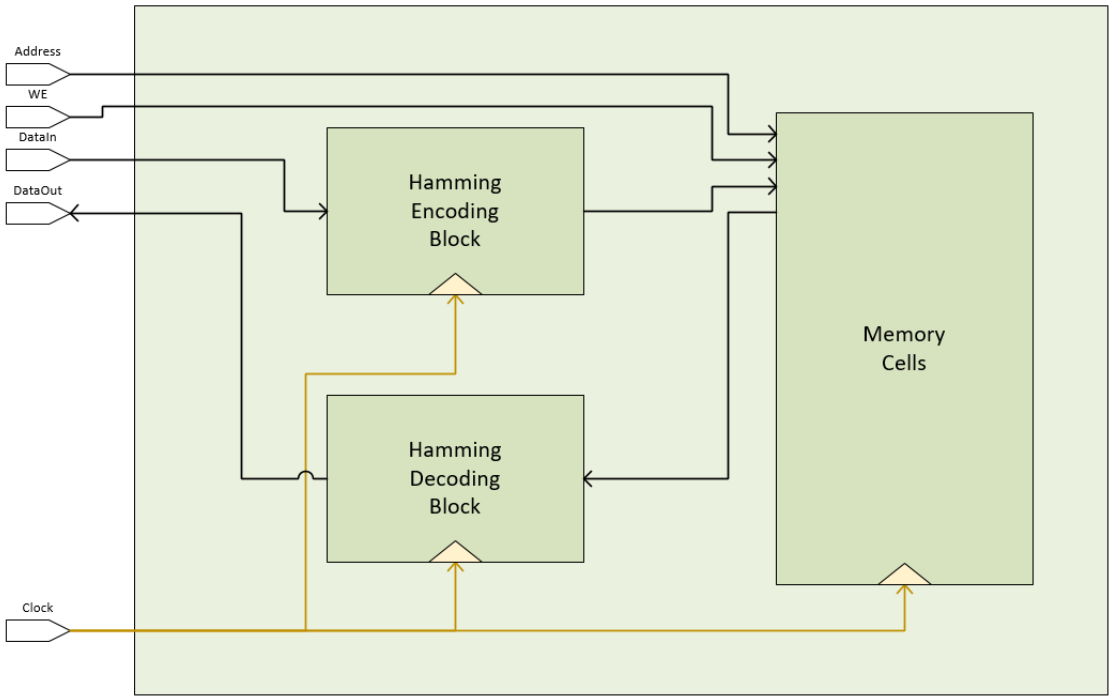

Figure 5.8: Memory Scheme on RTC

CONCLUSION

The error correcting methods described in this thesis were performed to push the development of the FPGA based radiation tolerant computing system at MSU. This venture was an effort to add a layer of fault mitigation on top of the previous design while maintaining the same performance of the system. This system has an technology maturation opportunity in the Radsatu project, integrating ECCs into the design of the system increase the likelihood of the project succeeding. The research team at MSU eagerly anticipate receiving data from Radsatu after its launch in to low Earth orbit.

REFERENCES CITED

[1] A. Bar-Lev. *Semiconductors and Electronic Devices (3rd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

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

[3] Connor Russel Julien. A radiation tolerant computer mission to the international space station. 2017. [Online; accessed March 21, 2019].

[4] J. Fabula. The total ionizing dose performance of deep submicron cmos processes. 2008.

[5] M. Fras, H. Kroha, J. V. Loeben, O. Reimann, R. Richter, and B. Weber. Use of triple modular redundancy (tmr) technology in fpgas for the reduction of faults due to radiation in the readout of the atlas monitored drift tube (mdt) chambers. In *IEEE Nuclear Science Symposuim Medical Imaging Conference*, pages 834–837, Oct 2010.

[6] C. H. Tsao, R. Silberberg, J. Adams, and J. R. Letaw. Cosmic ray effects on microelectronics. part 3. propagation of cosmic rays in the atmosphere. page 74, 08 1984.

[7] A. Holmes-Siedle and L. Adams. *Handbook of radiation effects*. Oxford science publications. Oxford University Press, 1993.

[8] E. Jamro. The design of a vhdl based synthesis tool for bch codecs. 1997.

[9] Jennifer Susan Hane. A fault-tolerant computer architecture for space vehicle applications. 2012. [Online; accessed March 21, 2019].

[10] Y. Jiang. *A Practical Guide to Error-control Coding Using Matlab*. Artech House, 2010.

[11] Justin Allan Hogan. Reliability analysis of radiation induced fault mitigation-strategies in field programmable gate arrays. 2014. [Online; accessed March 21, 2019].

[12] A. S. Keys, M. D. Watson, D. O. Frazier, J. H. Adams, M. A. Johnson, and E. A. Kolawa. High-performance, radiation-hardened electronics for space environments. 2007.

[13] B. LaMeres. Radsat - radiation tolerant small sat computer system. *Proceesings of the AIAA/USU conference*, 2015.

[14] B. J. LaMeres. Fpga-based radiation tolerant computing. 2012.

[15] B. J. LaMeres, S. Harkness, M. Handley, P. Moholt, C. Julien, T. Kaiser, D. Klumpar, K. Mashburn, L. Springer, and G. A. Crum. Radsat - radiation tolerant smallsat computer system. 2015.

[16] Mehrparvar, Arash. Cubesat design specification. February 20, 2014. [].

[17] NASA. New horizons: Spacecraft systems and components. 2016.

[18] Nasa.gov. Nasa ta 11, 2015. [Online; accessed March 20, 2019].

[19] Nasa.gov. Nasa ta 11. 2015. [Online; accessed March 20, 2019].

[20] Nasa.gov. Van allen radiation belts, NDA. [Online; accessed March 6, 2019].

[21] H. Quinn. High-performance computing for airborn applications. 2010.

[22] Raymond Joseph Weber. Reconfigurable hardware accelerators for highperformance radiation tolerant computers. 2014. [Online; accessed March 21, 2019].

[23] Samuel Andrew Harkness. Experiment platform to facilitate flight testing of faulttolerant reconfigurable computer systems. 2015. [Online; accessed March 21, 2019].

[24] W. Schimmerling. The space radiation environment: an introduction. *The Health Risks of Extraterrestrial Environments*, 2011.

[25] J. Schwank. Space and military radiation effects in silicon-on-insulator devices. 09 1996.

[26] T. Thompson. *From Error-Correcting Codes Through Sphere Packings to Simple Groups*. Number v. 21 in Carus Mathematical Monographs. Mathematical Association of America, 1983.

[27] P. Walter Schimmerling. The space radiation enviroment: An introduction. 2011.

[28] Xilinx. 7 series fpgas configuratble logic block - user guide. 2016.

[29] Xilinx. Vivado design suite user guide: Partial reconfiguration. 2018.