AN EXPLORATION OF ERROR-CORRECTING CODES FOR USE IN NOISE-PRONE SATELLITE ENVIRONMENTS

THESIS

Min W. Kang, 2d Lt, USAF

AFIT-ENG-MS-18-M-036

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-18-M-036

AN EXPLORATION OF ERROR-CORRECTING CODES FOR USE IN NOISE-PRONE SATELLITE ENVIRONMENTS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Min W. Kang, 2d Lt, USAF, B.S.C.S

March 2018

AFIT-ENG-MS-18-M-036

AN EXPLORATION OF ERROR-CORRECTING CODES FOR USE IN NOISE-PRONE SATELLITE ENVIRONMENTS

THESIS

Min W. Kang, 2d Lt, USAF, B.S.C.S

Committee Membership:

Dr. Kenneth M. Hopkinson, Ph.D.

(Chairman)

Major J. Addison Betances, Ph.D.

(Committee Member)

Major Daniel J. Casey, Ph.D.

(Committee Member)

Abstract

Satellites are crucial for the modern world to function properly as they provide Global Navigation Satellite System (GNSS) and global communication. However, the data that is stored on these satellites can be corrupted by the radiation found in space, and its bits can be improperly flipped. In the past, Forward Error Correction (FEC) algorithms were selected based on their strength and implemented to correct these bit flips back to their original values. This thesis seeks to determine if the strength of the FEC algorithms Reed Solomon (RS) code and Reed Solomon Product Code (RSPC) directly translates to their effectiveness. These algorithms were coded and tested in Matrix Laboratory (MATLAB) and on a Field Programmable Gate Array (FPGA) under controlled parameters, including the data set sizes, number of bit flips introduced, and the distribution of the bit flips within the data set. From the experiment's results, these other factors significantly influenced the effectiveness of the algorithms as well. Knowing what factors influence the algorithm's effectiveness enable better decision making as to which FEC algorithm to use for a given set of circumstances. The RS codes should be used if the size of the data set is small enough for a single-instance RS code and the range of expected bit flips is narrow and lower than the code's correctable limit. If the data set is large or the range of expected bit flips varies widely and surpasses the RS code's correctable limit, the RSPC should be used for a higher overall success rate in exchange for a lower number of bit flips with a 100% correction rate.

To My Family.

Acknowledgements

I would like to thank my faculty advisor Dr. Hopkinson and committee members Major Betances and Major Casey for their patience, guidance, and support during my research endeavors. I would also like to thank my friends and colleagues for their assistance and support throughout this thesis effort.

Min W. Kang, 2d Lt, USAF

Table of Contents

Page

List of Figures

List of Tables

Table Page

List of Acronyms

SEE Single Event Effect 27

SEFI Single Event Functional Interrupt 8

SEL Single Event Latch-Up 8

SET Single Event Transient 7

SEU Single Event Upset 64

SHA Secure Hash Algorithm 36

SoC System on Chip 39

VHDL Very High Speed Integrated Circuit Hardware Description Language 39

XOR Exclusive Or 13

AN EXPLORATION OF ERROR-CORRECTING CODES FOR USE IN NOISE-PRONE SATELLITE ENVIRONMENTS

I. Introduction

The modern world has become dependent on satellites to function properly as they enable services such as navigation, communication, and surveillance. Without satellites, the world would abruptly slow down as Global Navigation Satellite System (GNSS) would be lost and undersea lines for data transmission would quickly overflow their capacities. Despite their importance, satellites are hard to manage as they operate in harsh environmental conditions that deteriorate the satellite over time, and once they are launched, they are practically infeasible to repair. Structural well-being of the satellite is not the only concern for the repairs as extraterrestrial radiation has the potential to change the satellite's internal data as well.

A subset of the radiation problem was recognized as early as the 1960s, where radiation could cause bit flips during transmission of data, when the deep space Voyager missions were being planned. To combat the anticipated bit errors, a Reed Solomon (RS) code composed with a convolutional code was added to the Voyager spacecrafts for their launch in 1977 [3]. This selection of the RS codes as the Forward Error Correction (FEC) on the Voyager missions set the standard of using RS codes for future space missions. Over the years, as more spacecraft and satellites have been launched, it was noted that not only does radiation cause bit flips during transmission of data, it also causes bit flips while the data is at rest. This was first noticed in 1999 when the Advanced Research and Global Observation Satellite (ARGOS) was launched for 9 separate research missions. ARGOS was set with a Low Earth

Orbit (LEO) orbit at 834 km altitude, and over the course of roughly 3 months, an average of 5.5 radiation-induced errors per megabyte per day in memory was recorded, with 98.56% of the errors flipping one bit and 1.44% flipping multiple bits [4]. To fix such bit flip errors, data would need to be retransmitted to the satellite from the ground station on earth.

After learning of bits flips occurring in data at rest, the vast majority of research efforts on radiation mitigation focused on strengthening the satellite's hardware components in an attempt to block out radiation altogether. As it became clear that blocking 100% of radiation was nearly impossible and extremely expensive, some of the focus turned to improving the FEC codes onboard the system to correct the bit flips caused by radiation in data at rest. The strongest FEC algorithms that were being used commercially were sought and implemented on space systems. These included algorithms such as RS codes, Turbo codes, and low-density parity-check codes.

1.1 Objectives

The goal of this research is to determine if the various FEC's strength (what errors they are able to fix) directly translates to that FEC's effectiveness in an environment like space. Each FEC has its own unique strengths and weaknesses, such as RS codes being able to correct more consecutive bit errors than others at the cost of increased complexity. Yet, these strengths and weaknesses may not be the only factors that determine the algorithm's effectiveness. Other factors such as the data sizes on which FEC algorithms are applied to may also affect the algorithm's performance. The algorithms are also to be tested on two different platforms: Matrix Laboratory (MATLAB) and Field Programmable Gate Array (FPGA). The algorithms are tested on a software platform and a hardware platform to discover if there is any change in the algorithms' effectiveness if they are employed on different platforms.

1.2 Contributions

Prior research efforts involving FEC algorithms have tried to accelerate the algorithm's processing speed by using different components or running processes in parallel [5, 6, 7], or they tried to minimize the amount of hardware necessary to run the algorithms [8, 9]. Few efforts have focused on researching how good each algorithm is in space and whether any factors can be changed to improve the algorithm's success rate. Utilizing the most effective algorithm is the ultimate goal because the point of FEC algorithms is to get the system to autonomously correct the errors it detects without human intervention. In terms of FEC on satellites, the better the algorithm performs, the longer it will take until an uncorrectable instance is encountered and a retransmission of the data is needed. The length of time an algorithm can correct the errors on its own becomes increasingly important for satellites further away from earth as their orbital period typically becomes longer, making the communication windows to the satellite less frequent. This research compares the effectiveness of two FEC algorithms, RS and Reed Solomon Product Code (RSPC), with various test settings to determine if one outperforms the other and to see which factors influence each algorithm's effectiveness. Identifying the influential factors will enable better decision making as to which FEC to use under given circumstances and the traversing orbit of the satellite.

1.3 Thesis Organization

This chapter briefly introduced the subject area of FEC algorithms and their uses in space. Chapter 2 provides background information of the space environment and a summary of how RS and RSPC work. It also provides a literature review of the relevant research that have been conducted in the past to advance the mitigation of radiation effects. Chapter 3 describes the experimental methodology that was used to

test the effectiveness of RS and RSPC. Chapter 4 discusses the results and analysis of the experiments. Chapter 5 provides conclusions and recommendations for future work.

II. Literature Review

2.1 Chapter Overview

The objective of this chapter is to discuss important background knowledge that helps define the problem and the related research for radiation mitigation. This chapter begins with the radiation found in space and the effects it can cause in electronics. It then presents a few hardware and software mitigation techniques that are currently in use. Next, a review of how Reed Solomon (RS) codes are constructed and used is discussed. Finally, a brief overview of utilizing RS codes on an Field Programmable Gate Array (FPGA) concludes the chapter.

2.2 Space Radiation

Space is a volatile environment in which satellites subsist. Alongside extremes of temperature, solar flares, coronal mass ejections, and galactic cosmic rays discharge radiation throughout the solar system. There are two types of radiation in outer space: highly-charged particles and electromagnetic radiation. Particles consist mainly of elements with atomic numbers 26 or less, heavy ions, and free-floating electrons and protons. Electrons are the main source of electronic noise and signal spikes, and protons are the main source of Single Event Effect (SEE) [10]. Electromagnetic radiation are energized photons such as gamma rays and X-rays. The Earth's protective magnetic field repels a large portion of the extraterrestrial radiation, but some radiation become trapped within the Earth's magnetosphere. The trapped radiation form two ribbon-shaped regions around Earth, weakening at the magnetic axis, and are known as the Van Allen Belts as seen in Figure 1. While both the inner belt and the outer belt have trapped electrons and protons, the inner belt has a higher ratio of trapped protons and the outer belt has a higher ratio of trapped electrons. These belts roughly

act as the boundaries between the classified orbits. The inner belt extends roughly 1,609 - 12,875 kilometers above Earth and is between Low Earth Orbit (LEO) (180 - 2,000 kilometers) and Medium Earth Orbit (MEO) (2,000 - 35,786 kilometers). The outer belt extends roughly 19,312 - 40,234 kilometers above Earth and is between MEO and Geosynchronous Orbit (GEO) (35,786 kilometers and above). Aside from the extraterrestrial radiation, the trapped protons within the Van Allen belts cause a large portion of SEEs experienced by spacecrafts in the near-Earth region of space [1, 11]. Depending on the sun's cyclical activity, discharged radiation, along with the trapped radiation, is responsible for approximately 28% to 74% of all satellite malfunctions in the near-Earth region of space [12].

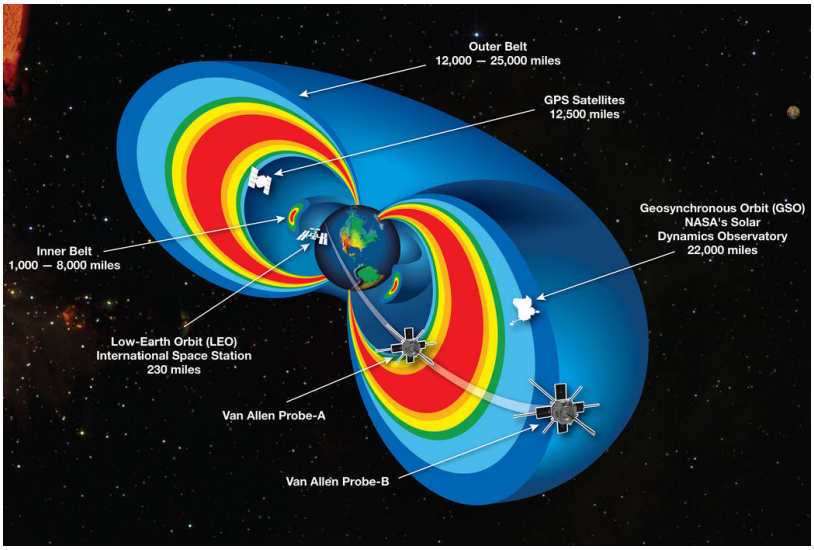


Figure 1. Van Allen Belts within Earth's Magnetosphere [1]

Radiation can interact with the satellite in two different ways according to what type of material is involved. In cases where heavy radiation such as ions or protons hits an atom and displaces it, it can change the structure of the matter the atom is a part of, slightly changing the matter's internal properties. It can also create cascading effects where the collision between the heavy radiation and the atom frees up some neutrons that can go on to collide with other atoms. With enough displacements, it

can make the material frail and shorten its intended lifespan. In cases where radiation imparts energy onto the matter, it can produce ionizing damage and free electrons that can create electron-hole pairs in transistors. When power runs through the circuit with the electron-hole pairs, an electrical field is created that moves the electrons one way and the holes the opposite way. This movement creates an improper current in the circuit that can cause a SEE to occur [13].

There are two types of SEEs. Destructive effects are known as hard errors as they cause permanent damage. Hard errors are not discussed any further as they cannot be corrected. Non-destructive effects are known as soft errors, for their effects are temporary and can usually be fixed. The majority of soft errors occur as a Single Event Upset (SEU) and a minority occur as a Single Event Transient (SET) [14].

SEUs typically cause transient errors within the system and are defined by National Aeronautics and Space Administration (NASA) as radiation-induced errors in micro-electronic circuits caused when charged particles (usually from the radiation belts or from cosmic rays) lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs [15]. This is primarily due to the high penetrating capability of protons with their positive charge as opposed to electrons with their negative charge that elements attract to fulfill their optimum valence shell. When protons collide with a cell in the system, they can cause soft errors according to their energy level, flux, and cell susceptibility [11]. Like protons, photons also have high penetrating capabilities because they have no associated charge. When photons collide with a cell in the system, the energy of the neutral photon is transferred over to the cell which can cause a soft error to occur. Soft errors from energy discharges can cause bits in the memory to flip. Most bit flips affect a single bit, but the energy discharged can affect multiple bits simultaneously, with the probabilities decreasing as it affects more bits [4].

Some SEUs can also appear as a subset called Single Event Functional Interrupt (SEFI) or as a SET in high density devices. Similar to a Single Event Latch-Up (SEL), SEFIs can change the logic configuration of a device or alter the current during a read or write process, producing an infinite loop or a partial erasure. Unlike SELs however, SEFIs are soft errors. Most regular SEFIs can be corrected by resetting the device's power. In the case of irregular SEFIs, the error-ridden process needs to be restarted again along with resetting the device's power to restore the right current to its pre-error state [16, 17]. Related to SEFIs, SETs affect the voltage or create false current pulses that disperse throughout the circuit. While most types of SEUs can be detected and corrected by data analysis or a hang-up in operation, the origin of SETs are difficult to detect as errors propagate. Without knowing and checking all of the intermediate solutions, the only way to correct SETs is to rerun the entire program from a previously known state [14].

2.3 Radiation Mitigation

2.3.1 Hardware Techniques.

With computing hardware becoming smaller and denser, the probability of space radiation causing SEUs is increasing. New radiation effects that affect multiple cells have also been observed due to recent efficiency efforts such as charge sharing between storage cells [18]. Radiation can be mitigated through hardware techniques by avoiding faults altogether or by making the hardware more tolerant.

2.3.1.1 Radiation Hardening.

To avoid faults through hardware, the materials and the fabrication process used to create the hardware must be strictly controlled. In particular, materials with the lowest amount of impurities must be used that also do not easily interact with

radiation, which rules out materials such as boron [19]. Hardware can also tolerate more radiation effects through intelligent designs. This approach attempts to design individual transistors in intrinsic ways through access transistors and transistor stacks to control the state of the memory cell in order to mitigate the effects of radiation [18]. Similar to logical software designs, the state of the memory cell will not change unless a specific order of voltages and currents pass through it. Thus, a trade-off is usually made in circuit speed for more radiation resistance.

The demand for radiation hardened hardware is low as they are usually only used in military and space applications. The low demand has produced only a small group of suppliers who develop the specialized parts, making the parts very expensive. Commercially available off-the-shelf parts can be purchased as a substitute, but the radiation hardness proclaimed by its manufacturers versus the actual performance of the parts can vary as radiation hardness is very dependent on its fabrication process. As a result, the same parts can have varying degrees of radiation responses if they were manufactured in different batches [20, 21]. Commercial components have also been observed to have varying responses to SEUs even if they were manufactured in the same batch [22].

As these space-grade components are engineered by the manufacturer to meet specific radiation standards set by the customer, they become an encompassing part in the cost of the space mission. Alongside the cost increase, the lead time for the mission also increases as it takes a considerable amount of time to design, build, and test each batch of the radiation hardened hardware to see if it passes the set criteria. Aside from systems specifically designed for warfare or for altitudes above that of GEO, radiation hardened hardware has seen relatively low performance improvements compared to other approaches in lieu of their cost as their designs typically trail their commercially available counterparts by at least 2 generations [4, 13, 21].

2.3.1.2 Radiation Shielding.

Radiation hardening is not only limited to hardening inner components such as transistors and processors. Radiation hardening can also be applied to the satellite in its entirety through an outer shield casing. Radiation shielding is accomplished through covering the satellite or particular regions with a few millimeters of radiation absorbent material such as aluminum, tin, or copper [23]. The amount of radiation that can be absorbed by the shielding varies depending on the type of radiation, shielding material that was used, and the thickness of the material.

There are two popular ways to employ radiation shielding. The first method is to employ a shield composed entirely of one material. This method is often used if the spacecraft's anticipated trajectory lies purely in LEO or within the inner Van Allen belt. Orbits close to Earth are highly protected from incoming radiation of space, but they are much more susceptible to the radiation trapped within the inner belt. As a result, these satellites must be designed to operate in the heavily proton radiated environment. Proton shielding is dominated by slowing down protons through inelastic scattering with atomic electrons, so a single thick layer of a low atomic numbered element with a high ionization energy such as aluminum has been shown to be the most effective.

The second method of radiation shielding is to employ a Graded-Z shield. This type of shield is a composite of many different atomic numbered elements with the aim to protect against a wide variety of radiation. Typical designs contain a gradient of element layers, with the higher atomic numbered elements on the outside and the lower atomic numbered elements on the inside. The layered design has multiple purposes. The outer layers are to act as a barrier and scatter the protons and electrons they encounter. The outer layers are also to absorb energized photons such as gamma rays and produce X-ray fluorescence in their stead. Each successive layer absorbs

the previous layer's X-ray fluorescence and produces their own with a lower energy level. At the end, the X-ray fluorescence's energy level is below the threshold to cause SEUs [23]. Most recently, Graded-Z shields with configurations of low, high, low atomic numbered element layers have been shown to be the most effective [21]. Graded-Z shields are often used at orbits above 9,000 kilometers as these orbits are much more susceptible to the extraterrestrial radiation discharges, meaning that the satellites must be designed to mollify the radiation effects of protons, electrons, and photons. With the same shield thickness as single material shields, graded-Z shields have been reported to be over 60% more effective at mollifying radiation effects of electrons and photons. They are also nearly as effective as single material shields at mollifying radiation effects of protons if the low atomic numbered element layer is thick enough for the radiation density as well as if they are located adjacent to the sensitive microelectronics onboard [21, 23].

A drawback to radiation shielding is the supplementary weight that is spread throughout the satellite as well as the extra volume it will cover. Such differences might seem minute, but they could pose additional problems for the mission designers who need to calculate and map everything the satellite will undergo. Radiation shielding could also cause unforeseen consequences as the radiation could create an unstable isotope from the shielding material. The unstable isotope could lose control and emit additional radiation into the satellite instead, undermining the purpose of the radiation shield in the first place [11].

2.3.2 Software Techniques.

As opposed to modifying hardware that seeks to prevent SEUs from occurring, software techniques seek to apply an intelligent control system in place to detect

and/or correct the errors caused by SEUs. Error-control coding can be added to a digital-electronic system to achieve a variety of goals [24]:

- To increase reliability of noisy data communication channels
- To control errors in order to reproduce the data accurately
- To increase signal-to-noise energy ratio
- To reduce the noise effects within a system
- To meet the demands of efficiency, reliability, and high performance of digital transmission and storage system.

Unlike communication systems on earth that can frequently communicate, a Global Navigation Satellite System (GNSS) satellite at MEO is more akin to a one-way communication channel as their orbital periods can range from 2 to 24 hours. Error control codes that best suit one-way channels are Forward Error Correction (FEC) codes that can automatically detect and correct errors at the receiving end of the channel. As satellites have limited amount of hardware and power available to use, large and complex codes cannot be used to combat very long codes. Alternatively, a hybrid error-control strategy that combines FEC, shorter codes, and a retransmission system has also been shown to be effective in one-way systems [24].

2.3.2.1 Types of Error Control.

There are many different mechanisms of error control that can be employed. One of the crudest forms of error control is to blindly add redundancy to the data by transmitting multiple duplicates and hoping that at least one copy is received correctly. This repetition of the message increases the bulkiness of the data without any guarantees that the transmitted data will be more reliable.

One of the most commonly used methods associated with computers is the use of parity checks. Data is usually partitioned into smaller partitions and an additional bit called the parity bit is appended to each partition according to whether there are an even or odd number of 1 bits in the partition. After the data is transmitted, the number of 1 bits and the parity bit can be checked to see if any errors occurred in the partitions. The detected errors within the partitions can be further narrowed down and corrected through cross-check parity bits. Appending only 1 parity bit allows for the detection of single errors; if two bit flips occurred in a partition, the system would misinterpret the partition to be correct. The error detection limit can be increased by altering the number of parity bits appended to the data. Likewise, the number of errors that can be corrected can be increased by applying more sophisticated algorithms to the parity bits [25].

Another way to control errors is through binary codes. Whereas parity checks are based on the binary nature of bits, binary codes are based on polynomial operations in a Galois Field (GF) with the prime characteristic $p = 2$. The algebraic nature of polynomials in GFs is that of a ring, where the code is cyclical based on multiples of a Generator Polynomial (GP) in the GF. Arithmetic operations in a GF are also different as they must be modded out by the characteristic $p = 2$ and the irreducible polynomial of the field. For example, addition mod 2 in a GF corresponds with Exclusive Or (XOR) logic mod irreducible polynomial.

Error detection is usually done through Cyclic Redundancy Check (CRC). This is done through binary division in GF(2) on both the transmitting side and the receiving side. The transmitting side adds the CRC remainder, a series of redundant bits that is calculated by dividing the data with padded zeros at the end with the CRC divisor, to the end of the data. The receiving side divides the data with the CRC divisor. If the remainder is all zeros, no errors have occurred. If the remainder is not all zeros,

an error has occurred within the block. Error correction is usually done through linear codes. Linear codes break the message into even length blocks and turn them into message symbols. These blocks are attempted to be mapped 1-to-1 to longer codewords, where the codewords are kept distinct from one another by a minimum number of bits. These mapped codewords are transmitted to the receiver, and the receiver checks to see if the received codewords are valid. If they are not valid, they are passed onto an algebraic decoder that calculates the received codeword's syndrome. If the calculated syndrome does not equal any syndrome vector in the syndrome table for the specified codeword sizes, the error cannot be corrected. If the syndrome is found, that syndrome's error pattern is found in the syndrome-decoding table and is subtracted from the received codeword to correct the error and retrieve the valid codeword [24, 26].

2.3.2.2 Reed Solomon Codes.

One of the oldest and most widely used error control algorithms is the RS codes. Established in 1960 by Irving Reed and Gustave Solomon, an RS code is a linear cyclic systematic non-binary block code that incorporates both parity bits and the idea of polynomial operations in GFs found in binary codes to form its fundamental basis of error detection and correction. In its early years, the mathematical concepts of RS codes were understood but was limited by the efficiency of its decoding algorithms. For nearly a decade after its invention, RS codes used finite field expansion of a function in a power series or Peterson's algorithm for binary Bose-Chaudhuri-Hocquenghem codes for its decoding operations which had a computational complexity of $O(n^3)$. This complexity limited the number of errors RS codes could computationally solve in a reasonable amount of time to 5 errors or less. Over the years, Peterson's algorithm was taken apart and streamlined by Gorenstein and Zierler,

Chien, and Forney to a complexity of $O(n^2)$. New decoding algorithms were also introduced by Berlekamp, Massey, and Sugiyama with complexities of $O(n^2)$. The improved decoding efficiencies made RS codes very popular and can be found in a wide variety of applications, from Quick Response (QR) codes to deep space exploration missions. In modern times, RS codes are often implemented with 5 separate algorithms: encoder generator, syndrome calculator, Berlekamp-Massey algorithm or Euclidean algorithm, Chien algorithm, and Forney algorithm [24]. The layout of an RS data set can be seen in Figure 2, where n is the size of the total data set, k is the number of data codewords, and t is half the size of appended parity codewords.

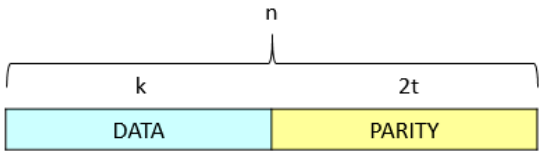


Figure 2. Layout of (n, k) Reed Solomon Code's Data Set [2]

The encoder generator is responsible for calculating the parity bits that must be appended onto the data set. First, the data is broken into groups of bits called codewords, and the k codewords are substituted with their respective GF symbols. The GF symbols in $m = (m_0, m_1, ..., m_{k-1})$ can be represented in a polynomial form to maintain its proper order as follows: $m(x) = m_0 + m_1 x + ... + m_{k-1}x^{k-1}$.

The symbols are shifted over by the number of parity codewords to be appended onto the message to create place-filling 0 codewords. The message polynomial is then divided by the GP, $g(x)$, for the specified number of parity codewords, as illustrated: $\frac{m(x)*x^{n-k}}{g(x)} = q(x) + \frac{r(x)}{g(x)}$. The remainder $r(x)$ at the end of the division operation is the actual parity codewords that need to be appended to the end of the data message. The GP is based on the consecutive symbols of the GF such that all nonzero symbols can be represented as the powers of some primitive symbols α. The GP can be calculated with $g(x) = \prod_{j=0}^{2t-1}(x - \alpha^{h+j})$, where t is half of the parity codewords, h

is an integer constant (usually 1), and the α elements can be substituted with their respective numeric GF symbols [24].

Once the message $r(x)$ is transmitted and received, the accuracy of the message is checked by the receiver. As the original transmitted message is evenly divisible by the GP, the message is also evenly divisible by the individual elements of the GP. Each of the remainders after dividing the message by a prime factor in $\frac{r(x)}{x+\alpha^i} = Q_i(x) + \frac{S_i}{x+\alpha^i}$, for $0 \leq i \leq (2t-1)$, is called a syndrome [27].

There are many ways to calculate the syndrome polynomial of the received message. One way is Horner's method, where each received codeword of the message is an intermediate step. For all except the last step, the previous step's solution is added with the current message codeword, and the quantity is multiplied by the GF symbol respective of the current syndrome position. The initial solution is always 0, and the last step is the same as the other steps except the quantity is not multiplied by the GF symbol. The final solution is the coefficient of the i^{th} syndrome, as illustrated: $S_i = (...(r_{n-1}\alpha^i + r_{n-2}\alpha^i + ... + r_1)\alpha^i + r_0)$.

If all of the syndrome polynomial's coefficients are 0, no error is detected within the received message; otherwise, there are errors within the received message. If there are detected errors, the errors are attempted to be corrected by the last 3 algorithms for the number of errors within the message are unknown.

The Euclidean algorithm produces two solutions: the error magnitude polynomial and the error locator polynomial. The main property between the error magnitude polynomial $\Omega(x)$ and the error locator polynomial $\Lambda(x)$ is their relationship as seen in $\Omega(x) = [S(x)*\Lambda(x)] \bmod x^{2t}$. This relationship can be rewritten as $\Omega(x)+\Theta(x)*x^{2t} = S(x) * \Lambda(x)$, where $\Theta(x)$ is some polynomial, to mirror the form of the Euclidean equation $a * s + b * t = c$, where a and b are the highest common factors of c.

By initializing $x^{2t} = 1$ and $S(x)$ to the calculated syndrome polynomial, the Euclidean algorithm can be used to find the error magnitude and the error locator polynomials [26, 27]:

$t = \frac{\text{parity codewords}}{2}$;
$a(x) = x^{2t}$;
$b(x) = S(x)$;
while $a(x) < t$ **do**
    $a(x) = b(x)$;
    $b(x) =$ remainder of $\frac{a(x)}{b(x)}$;
end
$\Omega(x) =$ remainder;
$\Lambda(x) =$ multiplying factors;

The error magnitude polynomial is responsible for calculating what symbol(s) must be added to the received message to correct the errors. The error locator polynomial is responsible for calculating the positions within the received message to which the rectifying symbols must be added. While the Berlekamp-Massey algorithm is slightly faster computationally than the Euclidean algorithm as its intermediate polynomials are shorter, it only produces the error locator polynomial. Therefore, the Euclidean algorithm is often used for its convenience.

After the error locator polynomial is calculated, the Chien search algorithm can be used to find its roots. This algorithm takes advantage of the fact that the GF is a finite field and tests every element of the field to see if it is a root of the polynomial. The algorithm substitutes α^n for all x in the polynomial according to the field element n that is being tested. If the polynomial evaluates to 0, the element field is a root of the polynomial and specifies that the codeword in the element position is erroneous.

The polynomials of a Chien search algorithm for GF(4) are:

$$\Lambda(x) = \Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3,$$
$$\Lambda(x) = \Lambda_0 + \Lambda_1(\alpha^{14}) + \Lambda_2(\alpha^{14})^2 + \Lambda_3(\alpha^{14})^3,$$
$$\Lambda(x) = \Lambda_0 + \Lambda_1(\alpha^{13}) + \Lambda_2(\alpha^{13})^2 + \Lambda_3(\alpha^{13})^3,$$
$$.$$
$$.$$
$$.$$
$$\Lambda(x) = \Lambda_0 + \Lambda_1(\alpha^0) + \Lambda_2(\alpha^0)^2 + \Lambda_3(\alpha^0)^3.$$

Once the roots of the error locator polynomial are found, the error values which must be applied to the received message can be calculated. The brute force method is to generate and solve a set of linear equations equal to the number of parity codewords that was added to the message. A computationally faster method is through Forney's algorithm which treats the set of equations as a Vandermonde matrix. To use Forney's algorithm, the derivative of the error locator polynomial must first be found. The differential operation, $\Lambda'(x) = \Lambda_1 + \Lambda_3 x^2 + \Lambda_5 x^4 + ...$, is also executed differently in a GF in that as the addition operator corresponds to XOR logic in GFs, all even powered terms of the original error locator polynomial are reduced to zero. The derivative of the error locator polynomial and the error magnitude polynomial are then used to construct the Forney equation $e_i = \alpha^i \frac{\Omega(\alpha^{-i})}{\Lambda'(\alpha^{-i})}$, which calculates the error values e_i. Finally, the error values are added to the received message at the i^{th} positions to procure the accurate message [26, 27].

Both the encoding and decoding aspects of RS codes revolve around the chosen GF and Primitive Polynomial (PP). The GF makes all polynomial operations cyclical in nature, and the PP guarantees that all integers within the GF can appear as valid

solutions in response to the polynomial operations. The PP also governs the order of each integer in the cyclical nature of the GF. The strength of the RS codes are governed by the number of parity codewords appended onto the data set. The codes can correct erroneous codewords up to half of the number of parity codewords, and the codes can fill in erasures up to the number of parity codewords. The number of bit flips present in the erroneous codeword does not matter since all bits and codewords act as parity elements to one another.

2.3.2.3 Reed Solomon Product Codes.

Reed Solomon Product Code (RSPC) is the same as RS except that it is done in two dimensions. The data set for RSPC is usually broken up into equal length rows and stacked on top of each other so they resemble a data table. Each row and each column is treated as its own separate RS code, and parity codewords are usually appended to all of the rows first. Then, parity codewords are appended to all of the columns, including those made up entirely of the rows' parity codewords. The layout of an RSPC data set can be seen in Figure 3, where

- n is the total number of codewords in one row,
- k is the number of row data codewords,
- t is half the size of appended row parity codewords,
- m is the total number of rows,
- c is how many rows of data there are,
- v is half the size of appended column parity codewords.

Decoding of errors is done in an alternating fashion between the dimensions, where all of the rows are decoded, then all of the columns are decoded. This loop continues

until all of the errors are fixed, no change was made in any of the rows or columns, or the loop is iterated up to the maximum correctable codeword times.

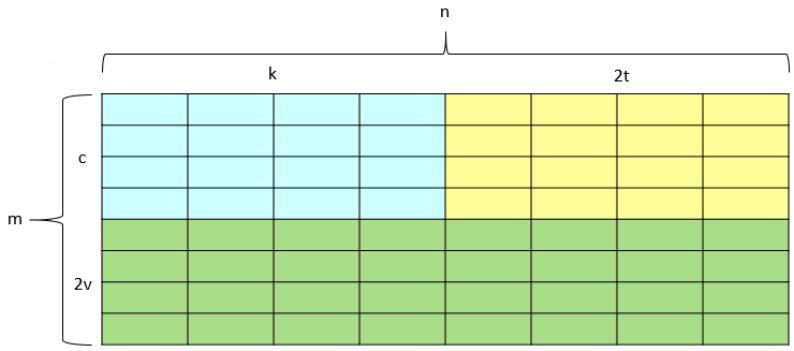


Figure 3. Layout of (n, k) x (m, c) Reed Solomon Product Code's Data Set [2]

2.4 FPGA

In response to the ever-increasing cost and development time needed for an Application Specific Integrated Circuit (ASIC), programmable logic devices were created for the manufacturers to build digital circuits more quickly. Such efforts further evolved to create an integrated circuit called an FPGA in the 1980s [28]. With this invention, customers themselves could configure the circuit to perform specific functions, similar to an ASIC. This was possible due to the programmable logic blocks contained in the FPGA. The logic blocks could be configured and connected like logic gates according to Very High Speed Integrated Circuit Hardware Description Language (VHDL).

In the beginning, the performance of FPGAs significantly lagged behind that of their General Purpose Computer (GPC) counterparts with limited logic blocks and slow clock rate. Despite the performance gap at the start however, the performance of FPGAs have overtaken GPCs in certain aspects due to their high degree of hardware arrangement that can be left to the application designer. This flexibility allowed FPGA manufacturers to focus on hardware updates and incorporate components

such as multipliers to improve its efficiency. On one hand, Moore's Law in the microprocessor industry decreased the size of complementary metal-oxide-semiconductor technology by 40%, doubled the transistors per unit area, and doubled the clock frequency every 2 years. Such advances allowed GPC performance to double every 18 months. On the other hand however, FPGAs directly benefited from the doubling of transistors and clock frequency as well as the new hardware components, quadrupling FPGA performance's upper limit every 2 years [29]. In modern times, GPCs outperformed FPGAs when performing complex tasks. Alternatively, FPGAs outperformed GPCs when processing fixed algorithms [30].

2.5 FPGA on Satellites

Satellites are expensive, hard to design, and time intensive to build and launch. They are even more arduous because after launch, they cannot be repaired or amended. As satellites have limited weight capacity and available power to use, its limited resources must be allocated properly. One of the most vexing problems in satellite design is to maximize the efficiency of computing power, available bandwidth to the ground station, and electrical power. The satellite's intended main functionality usually takes up the majority of the computing power, and adding error correcting algorithms to the processors' list of tasks will nullify its efficient use of available bandwidth to the ground station. By incorporating FPGAs onto the satellite, the computing intensive tasks of error control can be done in a parallel, pipelined fashion to increase throughput by efficiently using every clock cycle without significantly increasing the power consumed [31].

Another advantage of using FPGAs on satellites is that they are re-programmable logic devices. Unlike ASICs, their logic designs can be changed while in orbit, meaning that the same hardware can be used for multiple functions [17]. They are also good

at rectifying SEFIs as the FPGA can be powered off when not in use and powered on only when they are needed. Even in the case of SETs, programs on FPGAs can be rerun independently of the satellite's main functions.

2.6 Reed Solomon Codes on FPGA

With satellites moving deeper into space, error correction coding has become increasingly important to mitigate the effects of SEUs. Among the various FECs invented, convolutional, RS, and turbo codes have been widely researched and used on Earth. The Consultative Committee for Space Data Systems (CCSDS) have initially proposed using RS codes in the data link for satellite communication, and many partnering space agencies have adopted using RS codes as their FEC as early as 1999, when the Chinese used RS(255, 223) codes in their Shi-Jian 5 satellite [32].

The progression of using RS codes in space has advanced over the years, but as satellites are beginning to hold increasingly larger amounts of data, the size and structure of the RS codes have grown as well. To assuage the load on the satellites' main hardware components tasked with the satellite's core functionality, research has been done to put RS codes on FPGAs.

Many of the RS implementations for FPGAs have been limited in scope. There have been two general approaches researched within the past decade. One method was to increase the speed of the algorithm by avoiding brute force calculation methods and by using the least amount of logic blocks in the FPGAs [7]. Brute force calculations were avoided by using other algorithms such as Berlekamp-Massey algorithm and Forney's algorithm. The use of logic blocks was reduced by using shift registers or register transfer logic in designs such as the encoder and Berlekamp-Massey algorithms [8, 9, 33]. The other method was to increase the throughput of data by parallelization. There are many designs of RS that utilize parallelization, with the

vast majority focusing on parallelizing the decoding functionality of the algorithms such as Chien and Forney [5, 6].

2.7 Chapter Summary

Satellites operate in a harsh environment with varying temperatures and free-floating radiation. The radiation satellites encounter can cause SEUs to occur, corrupting the data stored on board. There are many mitigation techniques currently in use, where FEC is one of the most popular forms to combat soft errors. As one of the best performing FECs, RS codes have become a prominent choice to employ on FPGAs in the effort to further protect satellites from harmful radiation.

III. Methodology

3.1 Chapter Overview

This chapter discusses the methodology used to test Reed Solomon (RS) algorithms for use on satellites in outer space. An introduction to the experiment's goals and initial hypothesis is given, along with the assumptions and limitations used to conduct the experiment. Variables used throughout this research is classified and explained, and an outline of the experiments is given.

3.2 Experiment Objective

The objective of the experiment is to measure the effectiveness of two separate RS codes in order to determine the relationship between the algorithms' strengths and the Single Event Upset (SEU)-ridden environment in which they must operate. The two algorithms that will be tested are the original RS codes and the Reed Solomon Product Code (RSPC). Depending on the size of the algorithms and the parity codewords that are added, the strength of the relative codes is easy to calculate, but the effectiveness of the codes can vary depending on the situation for which they are used. The RS algorithms will first be simulated in Matrix Laboratory (MATLAB). The qualitative performance metrics that will be measured are whether the algorithms can positively or negatively detect the existence of errors, whether they can correct the errors accurately, and whether they can detect if all of the errors were correctly fixed. The quantitative performance metrics that will be measured are the maximum number of errors the algorithm could fix and how many errors were introduced into the data set. The results of the algorithms will be further verified through additional simulations in ModelSim and through physical implementation in

an Field Programmable Gate Array (FPGA) to verify that the software simulations are realistic.

3.3 Experiment Hypothesis

In this experiment, the hypothesis is that the RSPC algorithms will perform better than the original RS codes for the majority of the test cases. The RSPC might perform better because it has two dimensions of parity codewords instead of one, meaning that there are two layers of error correction available. While the amount of parity codewords in each layer may be less than the original RS codes, it might be able to switch back and forth between the horizontal and vertical dimensions and correct some of the errors in each run. However, the original RS codes will be able to correct the number of errors that near their maximum correctable limit better than the RSPC because they have more parity codewords with which to run their analysis and their error detection and correction are less sensitive to the distribution of errors than the RSPCs.

3.4 Experiment Scenario

The scenario being simulated is defined as a Global Navigation Satellite System (GNSS) satellite moving through space at Medium Earth Orbit (MEO). The satellite is responsible for storing and providing valuable data to its numerous clients, whether they are other satellites or other patrons on Earth. The satellite has FPGAs that have RS or RSPC implementations, data tables, and hashes of the original data that have been transmitted from the ground stations on Earth. Periodically, the FPGAs experience byzantine faults in the data sets, ranging from a couple random bit flips to entire swaths of bit flips. At regular intervals, the on-board RS code checks the data tables to preserve their integrity. If no errors are found, no further action is

instigated. If errors are found, it tries to detect and correct the errors. If there are too many errors to fix or the hash of the corrected data does not match its respective stored hash, it will send a signal to the ground station to request a fresh set of data.

3.5 Assumptions

The following assumptions were made for this experiment:

- The original data set and their respective hashes are transmitted correctly to the satellite.
- RS operational code and data hashes on the FPGA are stored in hardened hardware; they are not corruptible.
- SEUs on the data set only cause bit flips to occur; no hard errors occur that are not reversible.
- No SEU occurs on the data while the algorithms are running.
- There is a reliable source of electricity to power on the FPGA.

3.6 Limitations

There are many limitations imposed on the experiment to restrain its scope. First, the data set and their hashes are to be transmitted accurately without any errors to the satellite. Accurate transmission of the data set is not imperative for the experiment as the transmission errors might be able to be corrected with the RS codes, but accurate transmission of the hashes is essential to see if the corrected data set is accurate. Any errors within the hashes will make it difficult to determine if the RS codes accurately corrected the data set without the aid of another form of cryptographic integrity function.

The hashes and the operational code of the RS algorithms are to be stored in radiation hardened hardware to make them theoretically incorruptible. If any errors occur in the hashes, the ability to check the accuracy of the corrected data set will be lost. If any errors occur in the operational code of the RS algorithms, the algorithms will not work as intended and would most likely cause more errors trying to fix the errors it detected.

The Single Event Effect (SEE)s that occur are to only appear as SEUs that cause bit flips to occur in the data set. A certain threshold of soft errors can be managed and corrected by Forward Error Correction (FEC) such as the RS codes. The experiment does not consider SEE hard errors as they occur less frequently than soft errors, nor does it consider hard errors from repeated radiation damage over time [13, 14]. The addition of hard errors would expand the scope of the experiment and change its objective instead of maintaining a specific focus on the relationship between the RS algorithms and the SEUs.

The experiment's tests are conducted in a controlled setting where variables such as data sizes, number of errors, and the distribution of errors are all regulated. Unlike the tests however, real-world instances can vary widely depending on the activities occurring in space. The experiment's results only provide a general overview of what might happen if the satellite experienced similar conditions as in the experiment.

3.7 Response Variables

The response variables are summarized in Table 1. All four variables are categorical that record if the RS codes were able to accurately detect and correct the errors in the data set. Through these response variables, it will be possible to determine where the algorithms failed if they could not correct the errors accurately. Other factors such as timing and resources used on the FPGA are not recorded as they do

not relate to the objective of the experiment. These other factors can also change, subject to the computer specifications and the FPGA that the tests are run on.

Table 1. Observed Response Variables Used to Compute Performance of Algorithms

Variable Name	Normal Operating Level	Measure Precision	Relationship to Objective
posErrorDet	0 - 3	Positive, Negative, False Positive, False Negative	Correct detection of error existence
accurCorrection	0 - 1	True or False	All errors were corrected accurately
correctionDet	0 - 1	True or False	Fixed errors were detected correctly

3.8 Control Variables

The control variables are summarized in Table 2. Two qualitative variables are rsType and errorDist. The rsType specifies whether RS or RSPC is being tested, and the errorDist specifies how the errors that are introduced into the data set are dispersed. The quantitative variable errorRate specifies how many errors are introduced in each run. The errorRate percentage is the specified percentage of the maximum number of correctable codewords of RSPC sizes. The number of errors calculated for the RSPC sizes will be used to test the respective RS sizes as well to illustrate the satellites maintaining a specific orbit that encounters the same number of SEUs regardless of the FEC used. Instead of transferring the number of errors over from RSPC sizes, the same percentages could be used to calculate the number of errors for the RS sizes to compare the performance of each algorithm to each other. However, this would make the algorithms experience different numbers of SEUs and would make it more difficult to assess which algorithm performed better at varying altitude orbits in space.

Table 2. Control Variables. x represents the varying number of bit flips slash error distribution mode can introduce depending on the location and direction of the linear slash within the data set.

Variable Name	Normal Operating Level	Proposed Settings	Predicted Effects
rsType	Factor	RS, RSPC	Different RS types
errorDist	Factor	Uniform, Gaussian, Slash	How the errors are distributed throughout the data set
numBitFlips	0 - 100%	25%, 50%, 75%, 90%, x	Number of errors introduced depending on RSPC sizes

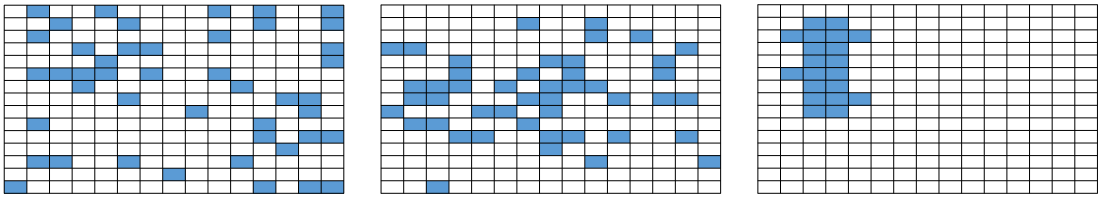
3.9 Constant Factors

There are many factors that are held constant throughout the experiment. One constant factor is that the Galois Field (GF) being used is the lowest size necessary to accommodate the data size being tested. This minimizes the amount of resources needed as fewer calculations need to be done, especially on the FPGA where resources are limited. The chosen GF(m) size also uses the lowest number of bits per codeword that they can handle (m bits per codeword) to ensure that error correction is possible by maintaining the properties of a GF. Larger number of bits per codeword can be utilized by adding extraneous 0's at the front of the codeword message, but if a bit flip occurs at one of the extraneous 0's, the cyclical nature of the GF would be broken as such errors would produce a string of bits that does not correlate to any codeword.

The percentage of parity codewords that are added to the data sets is kept at roughly 25% of the maximum set size ($2^m - 1$) allowed by the GF(m) for the RSPC sizes. This percentage differs slightly based on the GF(m) as the maximum set size may not be divisible by 4, and the number of parity codewords added to the data set must be even. The 25% was chosen as the specified parity codeword size because in working with small GF sizes such as 3, the minimum number of parity codewords that could be added was 2. A bigger percentage for the parity codewords was also not

chosen because even with 25%, the total ratio of data bits to parity bits was roughly 55:45. A bigger percentage would have skewed the ratio in the parity bits favor and made it more difficult to compare the performance of the different algorithms.

3.10 Error Distribution Modes



(a) Uniform distribution of 50 bit flips **(b) Gaussian distribution of 50 bit flips** **(c) Slash error distribution**

Figure 4. Distribution of Errors for Reed Solomon Product Codes with Galois Field(4). White box represents valid codeword, and blue box represents erroneous codeword.

3.10.1 Uniform Distribution.

There are three different ways errors are distributed throughout the data set. The first mode uniformly distributes the errors, as seen in Figure 4a. This is accomplished through the MATLAB function `randi` that returns a specified number of uniformly distributed pseudo-random integers given a minimum and maximum integer range. The lower bound of the range is always set to 1, and the upper bound of the range is set to the total number of bits present in the data set. The returned integers from the `randi` function represent the positions of bits within the data set that are flipped due to SEUs. This mode is analogous to elements of radiation randomly affecting the data set, where each bit in the data set has an equal chance to be flipped.

3.10.2 Gaussian Distribution.

The second mode allocates the errors along a Gaussian distribution, as seen in Figure 4b. This is accomplished through the MATLAB function `TruncatedGaussian`

designed by Bruno Luong that returns a specified number of pseudo-random integers from a Gaussian distribution given a specified range [34]. The μ is set to half of the total bits in the data set, the range is set from $-\mu$ to (total bits - μ) to center the Gaussian distribution around 0, and σ is set to a fifth of the total bits for the scale parameter of the distribution. The returned integers are added with μ to move the range from 1 to total number of bits in the data set and represent the positions of bits in the data set that are flipped. This mode is analogous to a cluster of radiation elements affecting the data set, with the effects mostly concentrated near the impact zone and decreasing in number the further it is from the impact zone. The impact zone could have been anywhere in the data set, but for this experiment, it was limited to the center of the data set so the specified number of bit flips could be drawn from the entire range of the data set.

3.10.3 Slash Distribution.

The third mode distributes the errors in a linear slash across the data set, as seen in Figure 4c. The linear slash can start anywhere in the data set and can travel in 8 directions:

1. up,
2. right,
3. down,
4. left,
5. up and right,
6. down and right,
7. down and left,

8. up and left.

The direction is chosen with the MATLAB function `randi` that returns a random integer between 1 and 8. Depending on the RS algorithm, the width and distance of the slash differs for each direction.

For RSPC, if the direction is up or down, the width of the slash horizontally can range from 1 bit to $(\lceil RowParityCodewords/4 \rceil * BitsPerCodeword)$ bits, and the distance vertically can range from 1 bit to total row bits. If the direction is left or right, the width of the slash vertically can range from 1 bit to $(\lceil ColParityCodewords/4 \rceil)$ bits, and the distance horizontally can range from 1 bit to $(Column{*}BitsPerCodeword)$ bits. If the direction is diagonal, the width of the slash can range from 1 bit to $(\lceil (RowParityCodewords + ColumnParityCodewords)/4 \rceil * BitsPerCodeword)$ bits, and the distance can range from 1 bit to $\lceil \sqrt{Rows + Columns * BitsPerCodeword} \rceil$ bits, with each distance unit moving in bit-stepwise fashion.

For RS codes, if the direction is up or down, the width of the slash horizontally can range from 1 bit to $(\lceil TotalParityCodewords/(Rows + Columns) * BitsPerCodeword \rceil)$ bits, and the distance vertically can range from 1 bit to total row bits. If the direction is left or right, the width of the slash vertically can range from 1 bit to $(\lceil TotalParityCodewords/(Rows + Columns) \rceil)$ bits, and the distance horizontally can range from 1 bit to $(Columns * BitsPerCodeword)$ bits. If the direction is diagonal, the width of the slash can range from 1 bit to $(\lceil TotalParityCodewords/(Rows + Columns) \rceil * BitsPerCodeword)$ bits, and the distance can range from 1 bit to $\lceil \sqrt{Rows + Columns * BitsPerCodeword} \rceil$ bits, with each distance unit moving in bit-stepwise fashion.

These ranges for the width of the slash were chosen to limit the number of errors to correctable levels for the RS algorithms. The distances were chosen so that the slash could continue until the end of the data set.

As the width of the slash was limited to correctable levels, bit flips apart from the slash were also scattered throughout the data set. The area in which the additional bit flips could be present were the areas adjacent to the slash that was up to a quarter of the width distance away from the slash on either side. Of all the bits present in those areas, a quarter of them were randomly selected with the `randi` function to be flipped. The addition of bit flips created the necessary conditions to push the error limit to uncorrectable levels for comparison purposes. This mode is analogous to a wave of radiation affecting the data set, resulting in the slash, and the additional bit flips are due to cascading effects of the radiation.

3.11 Tested Sizes

The RSPC sizes are selected first since the RSPC sizes grow faster than RS sizes. The two dimensions of RSPC are kept equal throughout the research to keep the design simple and limited in scope. The sizes of the RSPC can be seen in Table 3, and the addition of the data and parity codewords equal the maximum set size $(2^m - 1)$ of the GF(m) to maximize the number of data bits in the RSPC sizes.

Table 3. Reed Solomon Product Code Sizes

GF	Row & Col Data Bits	Row & Col Parity Bits	Total Data Bits	Total Parity Bits	Max Correctable Bits
3	15	6	75	72	39
4	44	16	484	416	224
5	115	40	2645	2160	1160
6	282	96	13254	10560	5664

After the RSPC sizes were selected, RS sizes were selected next. RS sizes were calculated for each RSPC size by figuring out the best combination of the number of data codewords, parity codewords, and number of instances for each GF(m) that got the number of data bits and parity bits as close to those of the RSPC size as possible.

The sizes of the respective RS codes for each RSPC can be seen in Table 4. Unlike the RSPC, the addition of the data and parity codewords in RS code sizes do not equal the maximum set size of the GF(m) because the priority was to get the number of data bits and parity bits as close as possible to those of the RSPC for comparison purposes.

Table 4. Reed Solomon Code Sizes with Respect to Reed Solomon Product Codes

RSPC GF	RS GF	Data Bits	Parity Bits	# of Instances	Total Data Bits	Total Parity Bits	Max Correctable Bits
3	6	78	72	1	78	72	36
3	5	75	70	1	75	70	35
3	4	28	24	3	84	72	36
4	6	162	144	3	486	432	216
4	5	80	70	6	480	420	210
4	4	28	24	17	476	408	204
5	6	204	168	13	2652	2184	1092
5	5	85	70	31	2635	2170	1085
6	6	210	168	63	10500	8400	4200

The number of bit flips for each RSPC size is based on the size's maximum correctable codewords which is ($MaxCorrectableBits/BitsPerCodeword$). Bit flips are based on correctable codewords instead of bits because the RS algorithms correct errors at the codeword level, meaning that it does not matter if one bit in the codeword is flipped or if all of the bits in the codeword is flipped. Percentages for the bit flips are based on 25% intervals of the correctable codewords except 90% is substituted for 100%. Bit flips equal to 100% of correctable codewords is specifically avoided because it requires the erroneous codewords to be in certain positions to be correctable. The number of bit flips for each RSPC size can be seen in Table 5, and the number of bit flips for each RS size can be seen in Table 6. The same number of bit flips for uniform and Gaussian error distribution modes is used for the corresponding RS sizes as the RSPC sizes so their performances can be compared to each other. The number

of bit flips for the slash error distribution mode is different for each RS and RSPC data sizes because the radiation slash is based on each memory block's dimensions and amount of appended parity codewords.

Each combination of the $x = 2$ algorithms, $n = 13$ data set sizes, $d = 3$ error distribution modes, $r = 5$ rates of bit flips, and $t = 2$ platforms is tested $y = 30$ times. Each combination is run $y = 30$ times based on the central limit theorem and Gosset's t-distribution. The central limit theorem states that multiple samples of the mean of independent random variables will approach a normal distribution when the number of samples is sufficiently large [35], and Gosset showed that even for a data pool numbering in the thousands, when the number of samples is 30, the t distribution becomes a close fit to the normal distribution [36]. As the fit to the normal distribution improves, better conclusions can be drawn about the population from the collected sample.

Table 5. Reed Solomon Product Code Bit Flips to Evaluate. % bit flips are calculated based on RSPC max correctable codewords: $(MaxCorrectableBits/BitsPerCodeword)$**.** x **represents the varying number of bit flips slash error distribution mode can introduce depending on the location and direction of the linear slash within the data set.**

RSPC GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips	Slash Bit Flips
3	3	7	10	12	$x \in [1, ..., 32]$
4	14	28	42	50	$x \in [1, ..., 135]$
5	58	116	174	209	$x \in [1, ..., 659]$
6	236	472	708	850	$x \in [1, ..., 3213]$

3.12 MATLAB Simulation

The RS and RSPC algorithms are first simulated in MATLAB version R2017a. MATLAB is a proprietary programming language that was developed by MathWorks in 1984 and has come to be known as the Language of Technical Computing for its variety of uses in engineering and scientific endeavors [37]. MATLAB was chosen

Table 6. Reed Solomon Bit Flips to Evaluate. % bit flips are calculated based on RSPC max correctable codewords: $(MaxCorrectableBits/BitsPerCodeword)$**.** x **represents the varying number of bit flips slash error distribution mode can introduce depending on the location and direction of the linear slash within the data set.**

RSPC GF	RS GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips	Slash Bit Flips
3	6	3	7	10	12	$x \in [1, ..., 150]$
3	5	3	7	10	12	$x \in [1, ..., 145]$
3	4	3	7	10	12	$x \in [1, ..., 104]$
4	6	14	28	42	50	$x \in [1, ..., 689]$
4	5	14	28	42	50	$x \in [1, ..., 540]$
4	4	14	28	42	50	$x \in [1, ..., 298]$
5	6	58	116	174	209	$x \in [1, ..., 2046]$
5	5	58	116	174	209	$x \in [1, ..., 1240]$
6	6	236	472	708	850	$x \in [1, ..., 5670]$

as the software platform to conduct the simulations because it has a relatively low learning curve and implements features such as parallelization and file manipulation. It also maintains an active community that participates in creating toolboxes and other functions through File Exchange that add even more capabilities to MATLAB.

The RS algorithm is implemented to execute in sequential order as seen in Figure 5. First, the data sizes for individual instances and the composite of all instances, minimum GFs, and their corresponding Primitive Polynomial (PP) is provided to the algorithm. The individual instance's GF(m) and PP are used to construct the field elements of the GF(m) by using the MATLAB function `deconv` that returns the binary equivalent of the quotient and remainder of the long division between the field element and the PP. The remainder is converted to a decimal form through the MATLAB function `bi2de` and is stored in the field element's position in a matrix. Randomized data is created for each RS instance, with each data codeword ranging from 0 to $2^m - 1$. The data codewords, GF(m), and the field elements matrix is passed to the encoder that produces the specified number of parity codewords to be appended to the data. A 256 bit hash for the entire message is calculated with the Secure Hash

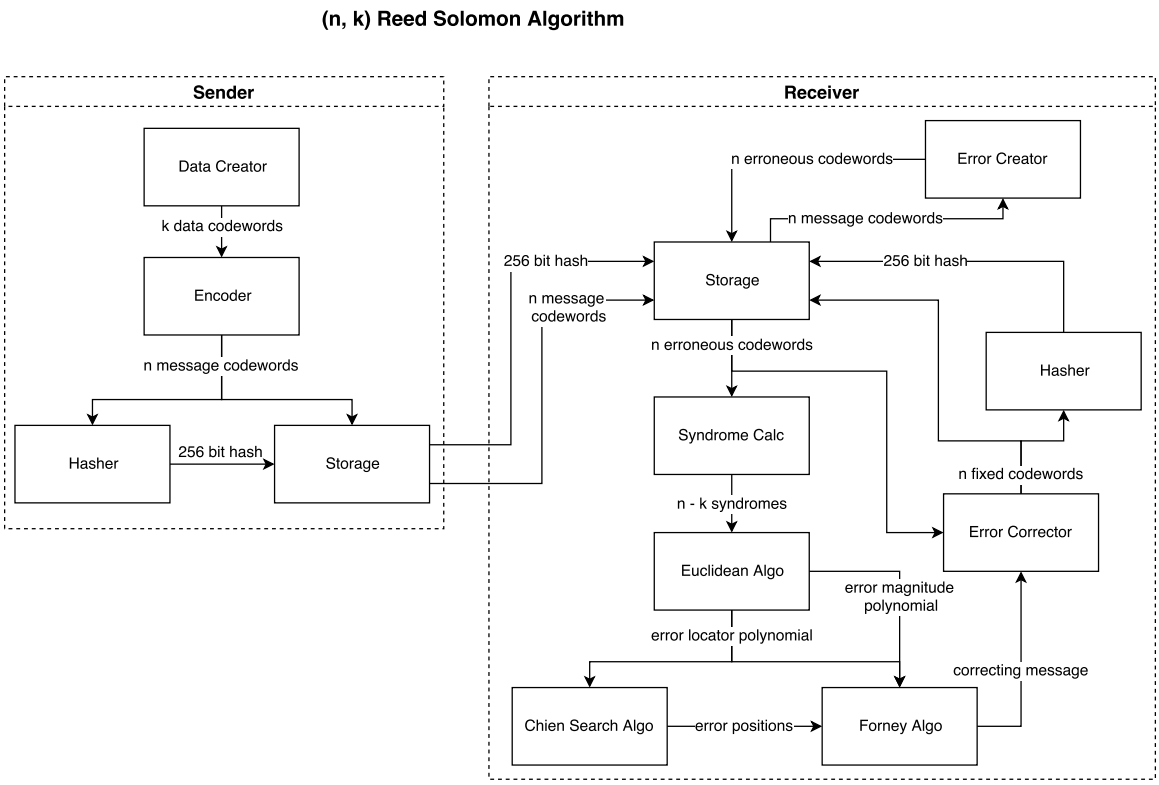


Figure 5. Data Flow Diagram of Reed Solomon Code Algorithm

Algorithm (SHA)-256 using the MATLAB function `DataHash` constructed by Jan Simon [38]. This bit hash is used to check the accuracy of the fixed codes.

Errors to be introduced are created all at once for the composite of all RS instances to get the desired effects of the chosen error distribution mode throughout the entire data set. As error creation deals with the total amount of data and parity bits, a bigger GF(m) usually needs to be used to first find out which codewords within the data set are to be corrupted. After all of the errors are created, the errors in each corrupted codeword is scaled down to fit the individual RS instances' GF(m) (m bits per codeword) by either setting the error to 1 or modding the error by 2^m. The errors are then added to each of the RS instances.

After the errors are added, the instance is passed onto the syndrome calculator. If all of the syndromes for the instance come out to be zero, no error is detected and

no further action is instigated. If the syndromes are not all zeros, the syndromes are passed onto the Euclidean algorithm. The Euclidean algorithm generates the error locator polynomial and the error magnitude polynomial for the instance. The error locator polynomial is passed onto the Chien Search algorithm which goes through all of the GF(m)'s field elements to find the codeword positions of the errors. The positions and the error magnitude polynomial are then passed onto the Forney algorithm that calculates what values need to be added to the received instance to correct the errors. These values are added to the instance, and the instance is stored until all of the RS's instances are corrected. A SHA-256 of the composite of all instances is calculated and compared to the original hash to find out whether all of the errors were corrected accurately or not.

The RSPC algorithm is implemented similarly to the RS algorithm. The biggest difference is that due to the two-dimensionality of RSPC, the error detection and correction process for each horizontal and vertical instance had to be implemented in a loop. The loop is set up to continue until no error is detected in any of the horizontal or vertical instances or if the loop is iterated as many times as there are columns (codewords in one horizontal instance) within the data set. The error detection and correction of the horizontal and vertical instances are also set to run only if the opposing dimension corrected an error in the last iteration. If the opposite dimension did not correct any errors in the last iteration, it would be pointless for the dimension to check again as not of the codewords would have changed since the last time they were checked. This loop is necessary for RSPC because if an iteration can correct even one error, the number of errors in the horizontal and vertical instances with the corrected error could have decreased to correctable levels and produce a cascading effect on the rest of the data set. Aside from the two-dimensionality and the loop, the instances are checked the same way as the RS algorithm.

3.13 FPGA Verification

The RS and RSPC algorithms are next tested on the Altera FPGA Cyclone V System on Chip (SoC) 5CSXC6. This FPGA device has an Advanced RSIC (Reduced Instruction Set Computing) Machine (ARM) Cortex dual processor core, 41,509 adaptive logic modules, 224 (18 bit x 19 bit) multipliers, and 110,000 logic elements. The Altera FPGA was chosen as the hardware testing platform because of its low cost, easy accessibility, and suitable resources available on the device. The algorithms were first implemented in Very High Speed Integrated Circuit Hardware Description Language (VHDL) using Quartus Prime version 16.1 Lite Edition and verified using ModelSim 10.5b.

The algorithms are implemented similarly to the MATLAB versions. Each sub-algorithm function in MATLAB is implemented as an entity in VHDL, and for loops are substituted with counters and conditional statements to allow for sequential processing of data. The VHDL implementations are also set to execute in sequential order. Each entity has an incoming wait signal tied to the previous entity's done signal and an outgoing done signal that is tied to the next entity's wait signal. This forces each entity to wait until its proper time when all of the necessary information is available for the entity to execute accurately. The major difference between the MATLAB implementations and the VHDL implementations is that data creation and error creation is done prior to the execution of the algorithms. The data creation and error creation are accomplished through the MATLAB implementations, and their binary representations are transferred over to the VHDL implementations. Thus, the VHDL implementations are only concerned with the receiving side of the RS data flow (Figure 5). The final accuracy checks of the corrected data sets are tied to the corresponding light-emitting diodes to provide a visual confirmation or rejection of the data set's accuracy.

3.14 Summary

The goal of this experiment is to determine how the two RS algorithms and their relative strengths relate to their effectiveness in an SEU-ridden environment. The algorithms will be tested with three different error distribution types to gauge each algorithm's strengths and weaknesses. Furthermore, the algorithms are tested on both software and hardware to verify that their effectiveness translates over without any significant differences.

IV. Results and Analysis

4.1 Chapter Overview

This chapter describes the results of the Reed Solomon Product Code (RSPC) and Reed Solomon (RS) experiments for each error distribution mode. The results are analyzed to determine the relationship between the control and response variables as well as the relationship between the Matrix Laboratory (MATLAB) simulations and the Field Programmable Gate Array (FPGA) implementations. Recommendations for which algorithm to use is presented based on the results.

4.2 Results

For all of the experiments, the tested data size, error distribution mode, and the number of introduced errors were recorded. In MATLAB simulations of RS and RSPC, all three response variables (posErrorDet, accurCorrection, correctionDet) were recorded. The posErrorDet was assigned based on the output of the syndrome calculator and a comparison between the received message and the original message. The accurCorrection was assigned based on the comparison between the output of the algorithm and the original message. The correctionDet was assigned based on the comparison between the two hashes. In FPGA tests however, only the accurCorrection was recorded as it was to verify the that effectiveness of the algorithms did not significantly change when implemented in hardware. The effectiveness of each data size was based on the average of accurately corrected data sets out of $y = 30$ runs.

4.2.1 RSPC Results.

The results for the MATLAB simulations of RSPC can be seen in Tables 7 and 8 and in graphical representations in Figures 6 and 7. In the data generated with uniform error distribution, the effectiveness of all of the tested data sizes suffer as more errors are introduced as expected. A general trend is also present where the bigger Galois Field (GF)s outperform the smaller GFs, tending to 100% effectiveness, up to the 75% error rate. At the 90% error rate however, the inverse happens where the smaller GFs outperform the bigger GFs. This inversion is due to the fact that as more errors are introduced in the bigger GFs proportionately, the chances of erroneous codewords forming an uncorrectable lock grid becomes higher as there are more combinations of lock grids available in bigger GFs than the smaller ones. This effect is mitigated up to the 75% error rate as the bigger GFs have more codewords to spread out the errors, and the codewords are composed of more bits, meaning that there is a higher chance of an Single Event Upset (SEU) flipping a bit in a previously corrupted codeword.

The same two trends also appear in the data generated with Gaussian error distribution. All of the tested data sizes decrease in effectiveness as more errors are introduced, and the bigger GFs outperform their smaller counterparts until 90% error rate. The effectiveness of the algorithm as a whole decreases faster with Gaussian error distribution than uniform error distribution. This is due to RSPC's inherent weakness. As RSPC has two dimensions of parity codewords, each dimension has a smaller amount of parity codewords to work with. The separate dimensions make correcting isolated errors easier, but they are not as effective at correcting large numbers of errors in close vicinity to one another, which is exactly how the Gaussian error distribution spreads out the errors. The data generated with the slash error distribution mode also shows that bigger GFs perform better than smaller GFs. This

is because there are more combinations of slashes that can be made with bigger GFs that are still correctable.

The Fisher's exact test is performed with the MATLAB RSPC data as seen in Table 9 to find out which control variables affect the effectiveness of the algorithm. In all four data sizes, the number of bit flips is a significant contributor to whether or not the errors can be fixed accurately, as seen with the low p-values below 0.05 for a 95% confidence interval. The error distribution mode is significant in the three bigger data sizes but not in the smallest data size. This can be either due to the similar proportions of success found in the random sampling group tested or due to the small dimensions of the data set itself. Even though the error distribution modes try to spread out the errors accordingly, the small data size limits their endeavors and makes their efforts regularly overlap, so the positioning of the errors within the data set does not greatly vary with respect to the error distribution mode. The small dimensionality of the data set is more likely to be the root cause of the error mode distribution's insignificant influence over the success rate in this case as the positioning of the errors is observably different for each mode in the bigger GFs, but it is hard to discern for GF(3).

The results for the FPGA implementations of RSPC can be seen in Tables 10 and 11 and in graphical representations in Figures 8 and 9. The same trends appear in the FPGA tests of the RSPC as in the MATLAB simulations. The effectiveness of the data sizes at various error rates and error distribution modes are also very similar. Fisher's exact test is performed with these data points to see if there are any notable differences, but the results come out to be the same: the number of bit flips is statistically significant in all of the data sizes while the error distribution mode is statistically significant in only the three bigger sizes. The data results from the MATLAB simulations and the FPGA implementations are also used to conduct the

chi-squared test. The high p-value of 1 means that the differences between the data sets are not statistically different, meaning that the MATLAB simulations accurately represent the performance of the RSPC on the FPGA.

Table 7. Performance of Reed Solomon Product Code Sizes with Uniform Error Distribution in MATLAB. Average percentage of accurately corrected data sets out of $y = 30$ runs.

RSPC GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips
3	100%	90%	30%	26.67%
4	100%	100%	36.67%	10%
5	100%	100%	96.67%	3.33%
6	100%	100%	100%	0%

Table 8. Performance of Reed Solomon Product Code Sizes with Gaussian and Slash Error Distributions in MATLAB. Average percentage of accurately corrected data sets out of $y = 30$ runs.

RSPC GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips	Slash Bit Flips
3	100%	76.67%	36.67%	16.67%	60%
4	100%	100%	40%	16.67%	86.67%
5	100%	100%	76.67%	0%	96.67%
6	100%	100%	96.67%	0%	100%

Table 9. Fisher Exact Test Results for Reed Solomon Product Codes in MATLAB. p-values of 0.05 or less signify that there is a statistically significant association between the control variable and the response varibale for a 95% confidence interval.

RSPC GF	Control Variable	Response Variable	p-value
3	errorDist	accurCorrection	0.822
3	numBitFlips	accurCorrection	2.2e-16
4	errorDist	accurCorrection	0.02778
4	numBitFlips	accurCorrection	2.2e-16
5	errorDist	accurCorrection	0.003071
5	numBitFlips	accurCorrection	2.2e-16
6	errorDist	accurCorrection	1.43e-3
6	numBitFlips	accurCorrection	2.2e-16

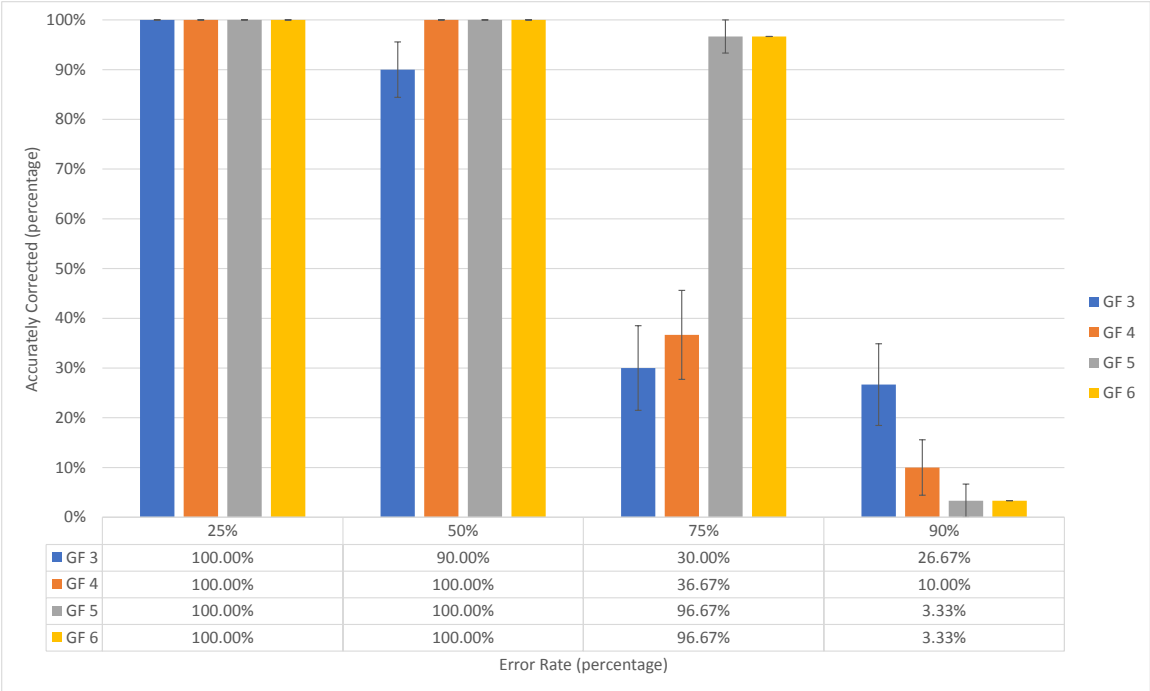


Figure 6. Performance of Reed Solomon Product Codes with Uniform Error Distribution in MATLAB. Each bar represents average percentage of accurately corrected data sets out of $y = 30$ runs. Each error bar represents the standard error of the mean for $y = 30$ runs.

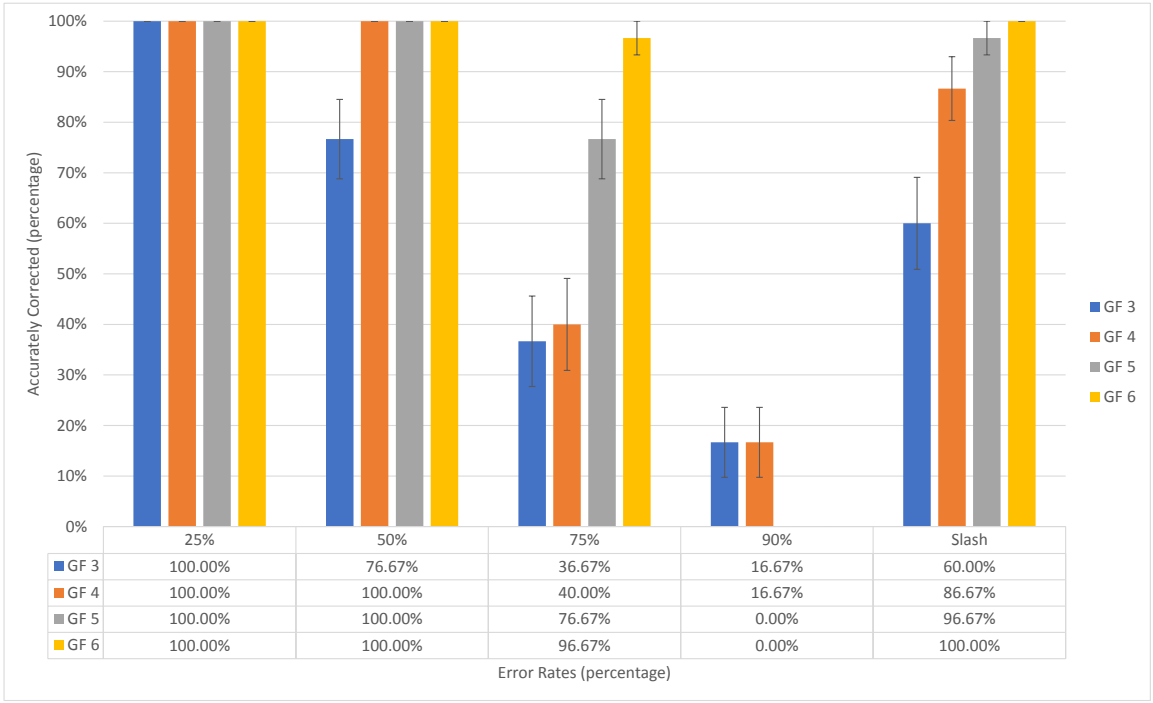


Figure 7. Performance of Reed Solomon Product Codes with Gaussian and Slash Error Distributions in MATLAB. Each bar represents average percentage of accurately corrected data sets out of $y = 30$ runs. Each error bar represents the standard error of the mean for $y = 30$ runs.

Table 10. Performance of Reed Solomon Product Code Sizes with Uniform Error Distribution on FPGA. Average percentage of accurately corrected data sets out of $y = 30$ runs.

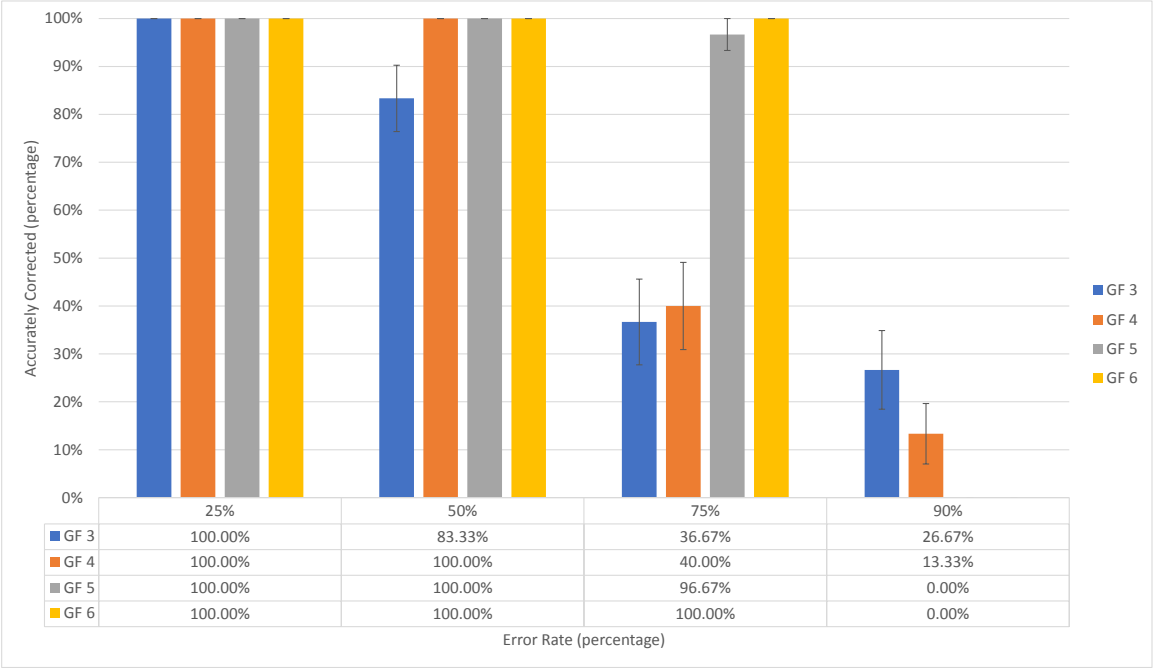
RSPC GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips
3	100%	83.33%	36.67%	26.67%
4	100%	100%	40%	13.33%
5	100%	100%	96.67%	0%
6	100%	100%	100%	0%

Table 11. Performance of Reed Solomon Product Code Sizes with Gaussian and Slash Error Distributions on FPGA. Average percentage of accurately corrected data sets out of $y = 30$ runs.

RSPC GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips	Slash Bit Flips
3	100%	76.67%	36.67%	16.67%	60%
4	100%	100%	40%	16.67%	86.67%
5	100%	100%	76.67%	0%	96.67%
6	100%	100%	96.67%	0%	100%

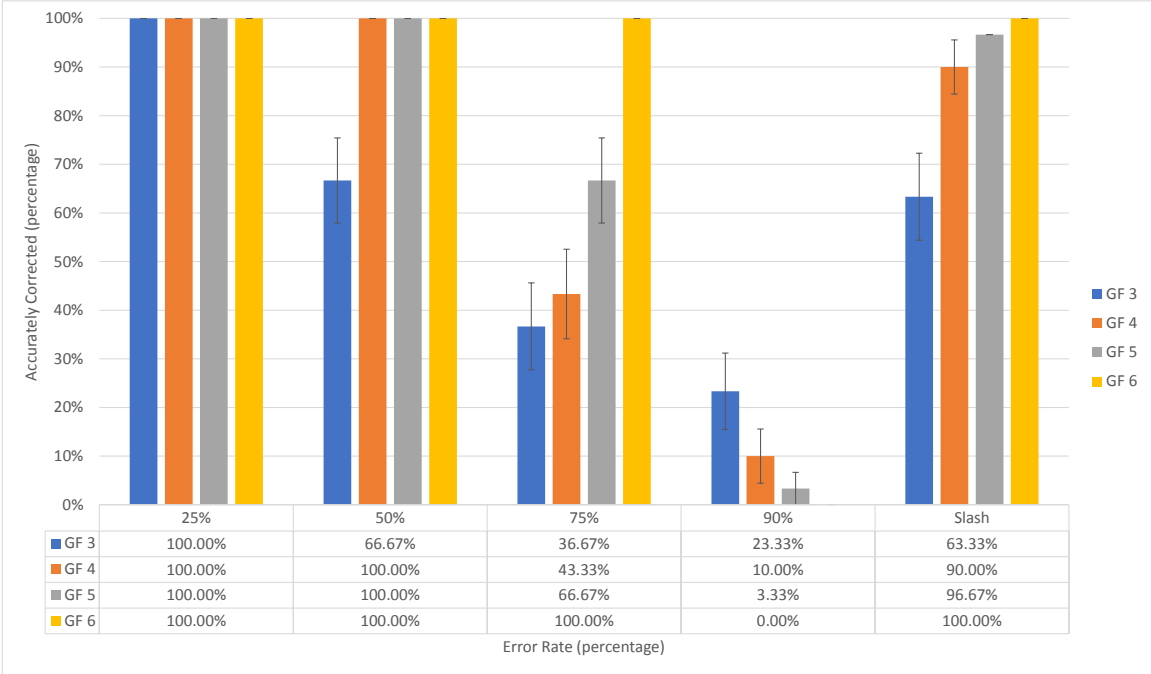
Table 12. Fisher Exact Test Results for Reed Solomon Product Codes on FPGA. p-values of 0.05 or less signify that there is a statistically significant association between the control variable and the response varibale for a 95% confidence interval.

RSPC GF	Control Variable	Response Variable	p-value
3	errorDist	accurCorrection	0.6756
3	numBitFlips	accurCorrection	8.84e-16
4	errorDist	accurCorrection	0.009141
4	numBitFlips	accurCorrection	2.2e-16
5	errorDist	accurCorrection	0.001865
5	numBitFlips	accurCorrection	2.2e-16
6	errorDist	accurCorrection	0.001723
6	numBitFlips	accurCorrection	2.2e-16



	25%	50%	75%	90%
GF 3	100.00%	83.33%	36.67%	26.67%
GF 4	100.00%	100.00%	40.00%	13.33%
GF 5	100.00%	100.00%	96.67%	0.00%
GF 6	100.00%	100.00%	100.00%	0.00%

Figure 8. Performance of Reed Solomon Product Codes with Uniform Error Distribution on FPGA. Each bar represents average percentage of accurately corrected data sets out of $y = 30$ runs. Each error bar represents the standard error of the mean for $y = 30$ runs.



	25%	50%	75%	90%	Slash
GF 3	100.00%	66.67%	36.67%	23.33%	63.33%
GF 4	100.00%	100.00%	43.33%	10.00%	90.00%
GF 5	100.00%	100.00%	66.67%	3.33%	96.67%
GF 6	100.00%	100.00%	100.00%	0.00%	100.00%

Figure 9. Performance of Reed Solomon Product Codes with Gaussian and Slash Error Distributions on FPGA. Each bar represents average percentage of accurately corrected data sets out of $y = 30$ runs. Each error bar represents the standard error of the mean for $y = 30$ runs.

4.2.2 RS Results.

The results for the MATLAB simulations of RS can be seen in Tables 13 and 14 and in graphical representations in Figures 10 and 11. In the data associated with uniform error distribution, many observations can be made. Likewise with the RSPC data above, the effectiveness of all of the data sizes decline as more errors are introduced into the data set. Within this trend, the higher GF(m) data sizes for each corresponding RSPC size seems to be more effective than the lower GFs except for the sizes equivalent to RSPC GF(3). Out of the three data sizes for RSPC GF(3), the best performing size across all four error rates is RS GF(5). This may be due to the fact that RSPC GF(3) is small enough that one instance of RS GF(5) is big enough to cover all of its field elements. As GF(5) is the smallest GF that can cover it all in one instance, it provides one of the best combinations of bits per codeword

and the number of erroneous codewords it can fix. GF(6) can also cover all of the field elements in one instance, but because it has more bits per codeword, it requires less parity codewords than GF(5) to achieve the same number of parity bits as GF(3) and thus can correct less erroneous codewords as well. The larger GF would have performed better if there were more consecutive bit flips than randomized bit flips, but the low error count coupled with the uniform error distribution mode made the majority of bit flips occur in their own individual codeword. Therefore, the number of erroneous codewords was always around the limit of what GF(6) could correct while GF(5) had one to two more codewords it could have fixed beyond what was actually corrupted. The overall effectiveness of the RS codes are lower than the RSPC because the RS codes only get to check and correct errors in one iteration. If any instance in the data set has too many corrupted codewords for the algorithm to fix, the entire run is deemed a failure. RSPC on the other hand can have as many iterations as there are codewords in one row to see if the number of erroneous codewords within a corrupted row or column have decreased to a correctable level.

In the data associated with Gaussian error distribution, similar observations are noted as above. All of the data sizes' effectiveness decreases as the number of bit flips rise, and except for the sizes corresponding to RSPC GF(3), the bigger RS GFs outperform the smaller GFs. As for the sizes corresponding to RSPC GF(3), RS GF(5) still outperforms both GF(6) and GF(4). However, as the Gaussian error distribution mode tends to concentrate bit flips in the middle of the data set, more multiple bit flips occurred within the same codewords and thus raised the effectiveness of both GF(5) and GF(6) than with the uniform error distribution. The data for the slash error distribution mode also follows the same trend where the bigger RS GF sizes outperform the smaller GFs for their corresponding RSPC sizes as bigger data sets have more combinations of slashes that can be corrected.

The Fisher's exact test is performed with the MATLAB RS data points to see what factors influenced the effectiveness of the codes. A summary of the results is given in Table 15. In all of the data sizes, both the number of bit flips and the error distribution mode are significant influencers of the codes' effectiveness as seen with the low p-values below 0.05 for a 95% confidence interval.

The results for the FPGA implementations of the RS codes can be seen in Tables 16 and 17 and in graphical representations in Figures 12 and 13. Similar trends appeared in the FPGA tests as in the MATLAB simulations. The overall effectiveness of the codes decrease as more errors are introduced, and the bigger GFs perform better than the lower GFs for their respective RSPC sizes except for those concerning RSPC GF(3). Even in the FPGA implementations, RS GF(5) outperform GF(6) and GF(4) for the reasons mentioned above. There are some minor differences in the performance of some data sizes between the MATLAB and FPGA versions, but all of them stay in range of their respective standard error ranges to each other so the differences are not significant. Fisher's exact test is performed again with these data points to see if any changes occurred. The number of bit flips is statistically significant in all of the tested data sizes. However, the error distribution mode is not significant for RSPC GF(3) RS GF(4) and RSPC GF(6) RS GF(6). Similarly to the previous case, this can be due to the random sampling group that was tested or due to the data set's sizes. After inspecting the error positions of the tested sizes, the root cause being the data set's sizes was dismissed because the positioning of the errors was observably different for each distribution mode. The sampling group was tested next by running the experiment for those sizes again and performing the Fisher's exact test. With this new group of data, the error distribution mode was a significant influencer of the sizes' effectiveness, so the original data's results were deemed to be caused by the sampling group that was tested. In situations where the proportions of success were

similar among the different error distribution modes, the success rate of the slash error distribution mode greatly affected the outcome as that mode only had a total of 30 runs while the other modes had a total of 120 runs. This root cause is further supported by the fact that the error distribution mode is deemed to be significant for the sizes in question in the MATLAB simulations as well. The data results from the MATLAB simulations and the FPGA implementations are also used to conduct the chi-squared test to see if the results are statistically different. The high p-value of 1 means that there is no statistical difference between the two data sets, so the MATLAB simulations accurately represented the performance of the RS codes on the FPGA.

Table 13. Performance of Reed Solomon Code Sizes with Uniform Error Distribution in MATLAB. Average percentage of accurately corrected data sets out of $y = 30$ runs.

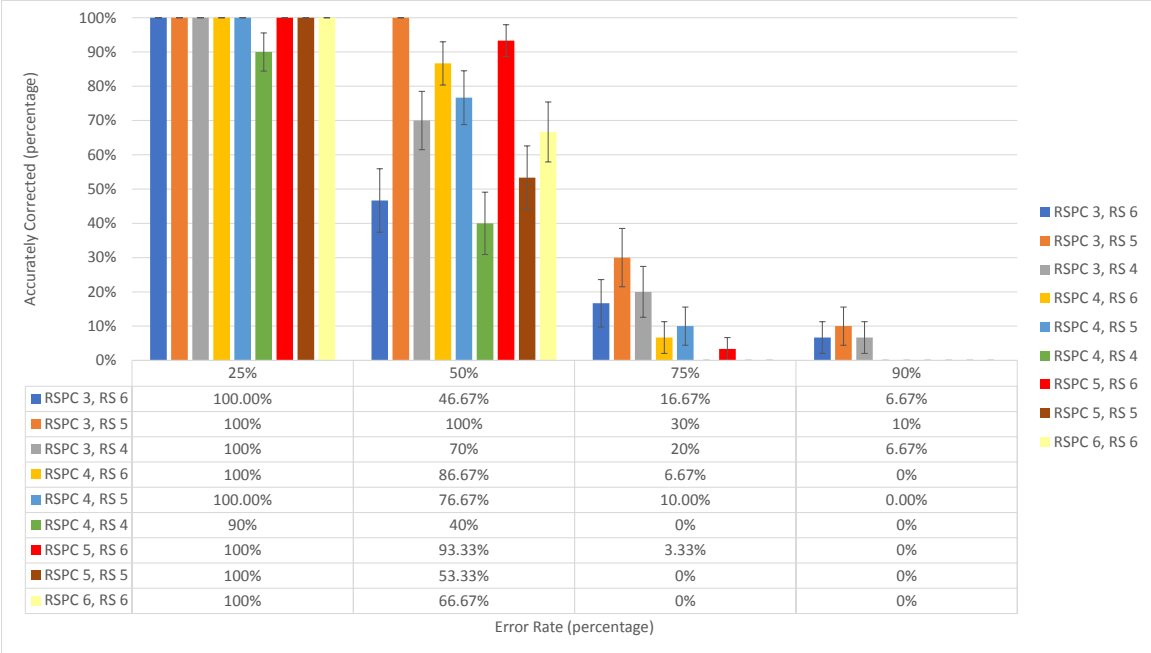
RSPC GF	**RS GF**	**25% Bit Flips**	**50% Bit Flips**	**75% Bit Flips**	**90% Bit Flips**
3	6	100%	46.67%	16.67%	6.67%
3	5	100%	100%	30%	10%
3	4	100%	70%	20%	6.67%
4	6	100%	86.67%	6.67%	0%
4	5	100%	76.67%	10%	0%
4	4	90%	40%	0%	0%
5	6	100%	93.33%	3.33%	0%
5	5	100%	53.33%	0%	0%
6	6	100%	66.67%	0%	0%

Table 14. Performance of Reed Solomon Code Sizes with Gaussian and Slash Error Distributions in MATLAB. Average percentage of accurately corrected data sets out of $y = 30$ runs.

RSPC GF	RS GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips	Slash Bit Flips
3	6	100%	73.33%	23.33%	13.33%	73.33%
3	5	100%	100%	43.33%	16.67%	90%
3	4	100%	60%	20%	10%	80%
4	6	100%	40%	3.33%	0%	83.33%
4	5	96.67%	30%	0%	0%	80%
4	4	76.67%	3.33%	0%	0%	43.33%
5	6	100%	10%	0%	0%	80%
5	5	100%	3.33%	0%	0%	46.67%
6	6	100%	0%	0%	0%	63.33%

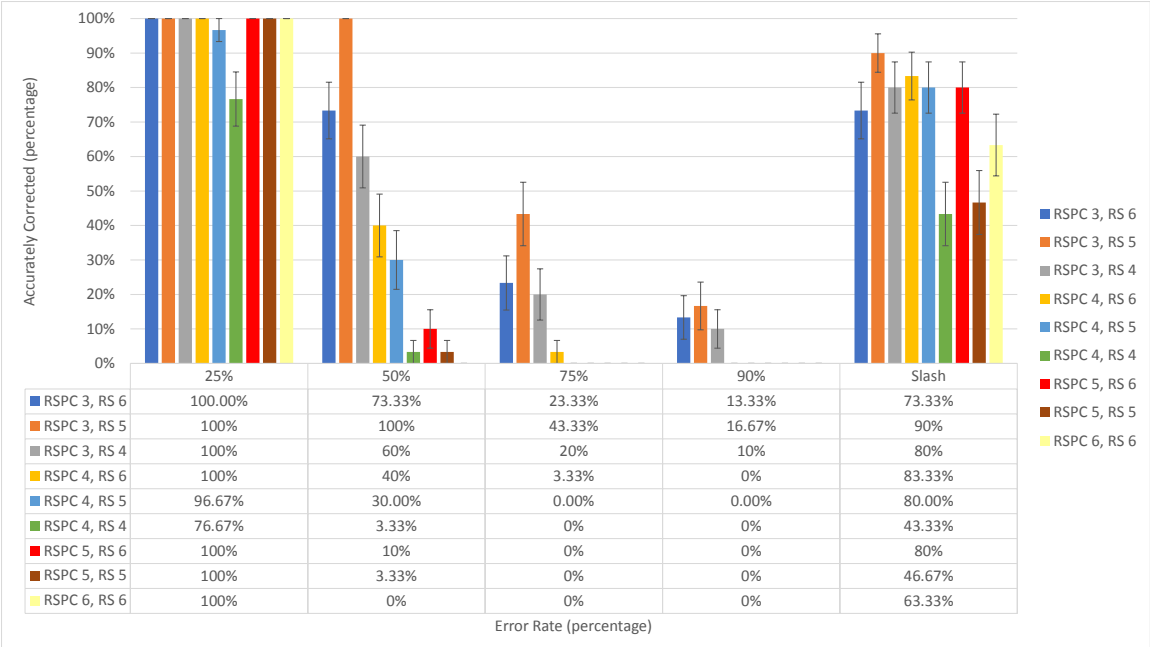
Table 15. Fisher Exact Test Results for Reed Solomon Codes in MATLAB. p-values of 0.05 or less signify that there is a statistically significant association between the control variable and the response varibale for a 95% confidence interval.

RSPC GF	RS GF	Control Variable	Response Variable	p-value
3	6	errorDist	accurCorrection	0.008455
3	6	numBitFlips	accurCorrection	2.2e-16
3	5	errorDist	accurCorrection	0.005291
3	5	numBitFlips	accurCorrection	2.2e-16
3	4	errorDist	accurCorrection	0.004121
3	4	numBitFlips	accurCorrection	2.2e-16
4	6	errorDist	accurCorrection	1.16e-5
4	6	numBitFlips	accurCorrection	2.2e-16
4	5	errorDist	accurCorrection	5.95e-6
4	5	numBitFlips	accurCorrection	2.2e-16
4	4	errorDist	accurCorrection	0.01403
4	4	numBitFlips	accurCorrection	2.2e-16
5	6	errorDist	accurCorrection	1.66e-7
5	6	numBitFlips	accurCorrection	2.2e-16
5	5	errorDist	accurCorrection	0.03345
5	5	numBitFlips	accurCorrection	2.2e-16
6	6	errorDist	accurCorrection	0.000156
6	6	numBitFlips	accurCorrection	2.2e-16



	25%	50%	75%	90%
RSPC 3, RS 6	100.00%	46.67%	16.67%	6.67%
RSPC 3, RS 5	100%	100%	30%	10%
RSPC 3, RS 4	100%	70%	20%	6.67%
RSPC 4, RS 6	100%	86.67%	6.67%	0%
RSPC 4, RS 5	100.00%	76.67%	10.00%	0.00%
RSPC 4, RS 4	90%	40%	0%	0%
RSPC 5, RS 6	100%	93.33%	3.33%	0%
RSPC 5, RS 5	100%	53.33%	0%	0%
RSPC 6, RS 6	100%	66.67%	0%	0%

Figure 10. Performance of Reed Solomon Codes with Uniform Error Distribution in MATLAB. Each bar represents average percentage of accurately corrected data sets out of $y = 30$ runs. Each error bar represents the standard error of the mean for $y = 30$ runs.



	25%	50%	75%	90%	Slash
RSPC 3, RS 6	100.00%	73.33%	23.33%	13.33%	73.33%
RSPC 3, RS 5	100%	100%	43.33%	16.67%	90%
RSPC 3, RS 4	100%	60%	20%	10%	80%
RSPC 4, RS 6	100%	40%	3.33%	0%	83.33%
RSPC 4, RS 5	96.67%	30.00%	0.00%	0.00%	80.00%
RSPC 4, RS 4	76.67%	3.33%	0%	0%	43.33%
RSPC 5, RS 6	100%	10%	0%	0%	80%
RSPC 5, RS 5	100%	3.33%	0%	0%	46.67%
RSPC 6, RS 6	100%	0%	0%	0%	63.33%

Figure 11. Performance of Reed Solomon Codes with Gaussian and Slash Error Distributions in MATLAB. Each bar represents average percentage of accurately corrected data sets out of $y = 30$ runs. Each error bar represents the standard error of the mean for $y = 30$ runs.

Table 16. Performance of Reed Solomon Code Sizes with Uniform Error Distribution on FPGA. Average percentage of accurately corrected data sets out of $y = 30$ runs.

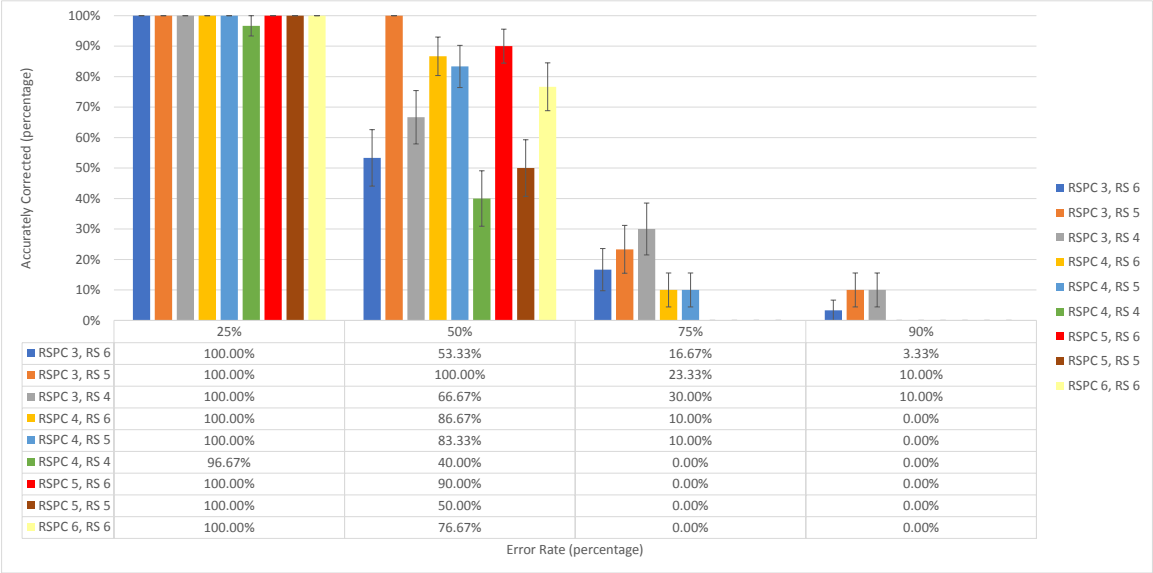
RSPC GF	RS GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips
3	6	100%	53.33%	16.67%	3.33%
3	5	100%	100%	23.33%	10%
3	4	100%	66.67%	30%	10%
4	6	100%	86.67%	10%	0%
4	5	100%	83.33%	10%	0%
4	4	96.67%	40%	0%	0%
5	6	100%	90%	0%	0%
5	5	100%	50%	0%	0%
6	6	100%	76.67%	0%	0%

Table 17. Performance of Reed Solomon Code Sizes with Gaussian and Slash Error Distributions on FPGA. Average percentage of accurately corrected data sets out of $y = 30$ runs.

RSPC GF	RS GF	25% Bit Flips	50% Bit Flips	75% Bit Flips	90% Bit Flips	Slash Bit Flips
3	6	100%	73.33%	30%	10%	83.33%
3	5	100%	100%	40%	13.33%	86.67%
3	4	100%	63.33%	16.67%	6.67%	70%
4	6	100%	43.33%	0%	0%	80%
4	5	96.67%	26.67%	0%	0%	83.33%
4	4	86.67%	0%	0%	0%	43.33%
5	6	100%	3.33%	0%	0%	86.67%
5	5	93.33%	3.33%	0%	0%	40%
6	6	100%	0%	0%	0%	56.67%

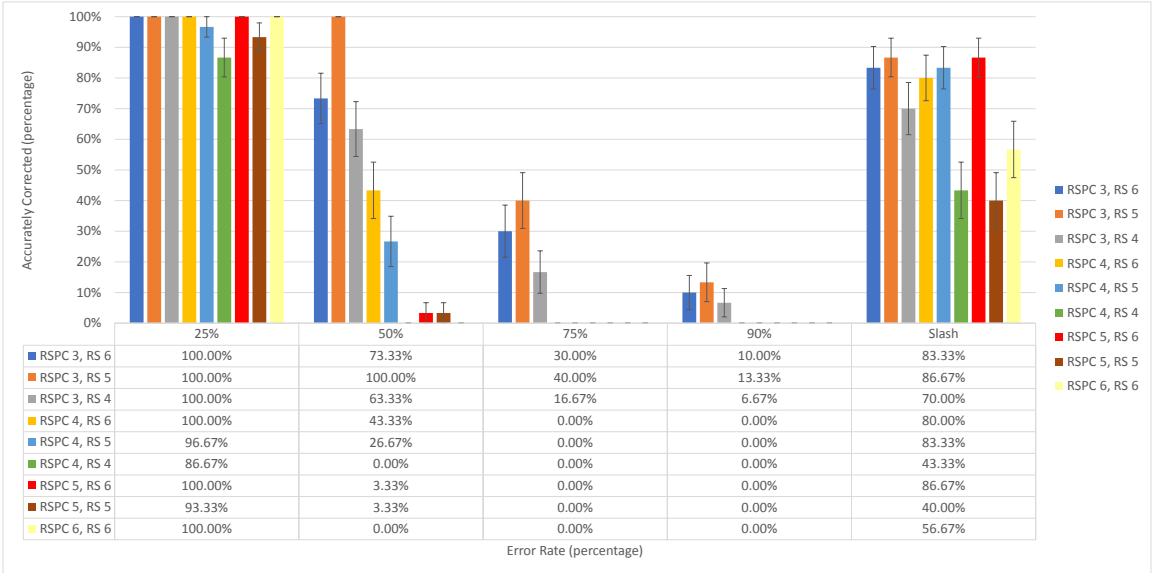
Table 18. Fisher Exact Test Results for Reed Solomon Codes on FPGA. p-values of 0.05 or less signify that there is a statistically significant association between the control variable and the response varibale for a 95% confidence interval.

RSPC GF	RS GF	Control Variable	Response Variable	p-value
3	6	errorDist	accurCorrection	0.0003153
3	6	numBitFlips	accurCorrection	2.2e-16
3	5	errorDist	accurCorrection	0.01106
3	5	numBitFlips	accurCorrection	2.2e-16
3	4	errorDist	accurCorrection	0.07135
3	4	numBitFlips	accurCorrection	2.2e-16
4	6	errorDist	accurCorrection	5.11e-5
4	6	numBitFlips	accurCorrection	2.2e-16
4	5	errorDist	accurCorrection	5.12e-7
4	5	numBitFlips	accurCorrection	2.2e-16
4	4	errorDist	accurCorrection	0.02211
4	4	numBitFlips	accurCorrection	2.2e-16
5	6	errorDist	accurCorrection	1.48e-9
5	6	numBitFlips	accurCorrection	2.2e-16
5	5	errorDist	accurCorrection	0.04733
5	5	numBitFlips	accurCorrection	2.2e-16
6	6	errorDist	accurCorrection	0.07135
6	6	numBitFlips	accurCorrection	2.2e-16



	25%	50%	75%	90%
RSPC 3, RS 6	100.00%	53.33%	16.67%	3.33%
RSPC 3, RS 5	100.00%	100.00%	23.33%	10.00%
RSPC 3, RS 4	100.00%	66.67%	30.00%	10.00%
RSPC 4, RS 6	100.00%	86.67%	10.00%	0.00%
RSPC 4, RS 5	100.00%	83.33%	10.00%	0.00%
RSPC 4, RS 4	96.67%	40.00%	0.00%	0.00%
RSPC 5, RS 6	100.00%	90.00%	0.00%	0.00%
RSPC 5, RS 5	100.00%	50.00%	0.00%	0.00%
RSPC 6, RS 6	100.00%	76.67%	0.00%	0.00%

Figure 12. Performance of Reed Solomon Codes with Uniform Error Distribution on FPGA. Each bar represents average percentage of accurately corrected data sets out of $y = 30$ runs. Each error bar represents the standard error of the mean for $y = 30$ runs.



	25%	50%	75%	90%	Slash
RSPC 3, RS 6	100.00%	73.33%	30.00%	10.00%	83.33%
RSPC 3, RS 5	100.00%	100.00%	40.00%	13.33%	86.67%
RSPC 3, RS 4	100.00%	63.33%	16.67%	6.67%	70.00%
RSPC 4, RS 6	100.00%	43.33%	0.00%	0.00%	80.00%
RSPC 4, RS 5	96.67%	26.67%	0.00%	0.00%	83.33%
RSPC 4, RS 4	86.67%	0.00%	0.00%	0.00%	43.33%
RSPC 5, RS 6	100.00%	3.33%	0.00%	0.00%	86.67%
RSPC 5, RS 5	93.33%	3.33%	0.00%	0.00%	40.00%
RSPC 6, RS 6	100.00%	0.00%	0.00%	0.00%	56.67%

Figure 13. Performance of Reed Solomon Codes with Gaussian and Slash Error Distributions on FPGA. Each bar represents average percentage of accurately corrected data sets out of $y = 30$ runs. Each error bar represents the standard error of the mean for $y = 30$ runs.

4.3 Comparisons Between RSPC and RS

4.3.1 RSPC Comparisons.

Even though the algorithms' strengths and maximum correctable codewords can be calculated, it does not mean that they will always be able to correct that amount of corrupted codewords. As seen from the collected data, the performance of the algorithms can vary depending on the environment in which they operate. RSPC is first examined, followed by an examination of the RS codes.

As the number of bit flips are calculated from the RSPC's maximum correctable codewords, it is certain that the number of bit flips never exceed RSPC's limit. In other words, even if each bit flip took place in a different codeword, all of the test combinations and runs had a chance to be correctable. Yet because the RSPC is sensitive to the positioning of erroneous codewords, it can only take as little as $RowParity + ColParity$ bit flips to make the data set uncorrectable in the worst case scenario. As the number of bit flips is always correctable, the success of the tests came down to the positions of the corrupted codewords in which bit flips occurred. While the combinations of uncorrectable lock grids increase with the data set size, the chances of the bit flips actually forming an uncorrectable lock grid inversely decreases as the data set size increases for error rates of 75% or less. Thus, RSPC is appropriate to use for sensitive data where a large number of errors is expected.

RSPC is also more suitable to use as the number of maximum correctable codewords each size can fix is based on $(Rows * (ColParity/2)) + (Cols * (RowParity/2)) - (RowParity/2 * ColParity/2)$ as opposed to RS codes that is based on $Parity/2 * NumOfInstances$. This allows the RSPC to have a higher upper limit of the codewords it can correct, which also translates to more fixable bits overall. RSPC is generally better at fixing random, non-consecutive erroneous codewords than consecutive

erroneous codewords, but the correctable rate of consecutive erroneous codewords does not lag far behind.

The speed of the RSPC might be slower than the multiple-instance versions of the RS codes as it can have many more iterations to run through than one iteration for each RS instance. However, not all rows and columns need to be processed as they can be skipped entirely if no error is detected. The maximum number of iterations also do not need to be completed if either the rows or the columns do not change any bits in one iteration.

4.3.2 RS Comparisons.

As the number of bit flips is derived from the RSPC sizes, they do not align with the RS codes' maximum correctable codewords. In many of the RS instances, even the 75% number of bit flips exceeds the data size's maximum correctable codewords limit. This is one of the reasons why the performance of RS codes drop drastically at 75% and higher. Another reason for the reduced performance is that multiple instances of small GFs are used to mirror the RSPC sizes. The use of small GFs makes the entire data set frail as the instances have a lower maximum correctable codeword limit than if one large GF(m) is used. If the bit flips ever congregate in a specific location, like with the Gaussian error distribution mode, the chances of the corrupted codewords exceeding an instance's correctable limit is high, which condemns the data set to fail.

Instead of multiple RS instances, one large GF(m) should be used to encode and decode the entire data set, like in RSPC GF(3) RS GF(5) and RS GF(6). Unlike the RSPC, RS codes are not sensitive to the positioning of errors unless multiple instances of RS codes are used for a single data set. Single instances of RS codes only care about how many erroneous codewords there are to determine if the data set is correctable or not, so its performance for a given number of bit flips is easy to predict. If the

number of bit flips is equal to or less than the size's maximum correctable codewords limit, the RS codes will fix the errors 100% of the time. If the number of bit flips exceeds the limit however, the performance of the RS codes drop drastically. This can be evidenced in the collected data for RSPC GF(3) RS GF(5) and RS GF(6). RS GF(5) had a limit of 7 erroneous codewords, so the algorithm was able to correct all of the 3 and 7 bit flip instances before its performance dropped. RS GF(6) had a limit of 6 erroneous codewords, so the algorithm was able to correct all of the 3 bit flip instances before its performance dropped. Thus, RS codes are appropriate to use if the range of error rates the codes will deal with is well established.

The multiple instanced version of the RS codes process faster than the single instance version due to their smaller GFs. The number of calculations for each GF(m) increases exponentially as the number of field elements which each sub-algorithm's calculations are based on is $2^m - 2$. The multiple instanced version does require an iteration for each instance, but the total number of calculations processed for the single instance version still outweigh the multiple instanced version. Furthermore, if the data size for the single instanced version requires a large GF(m), the processing of RS becomes slower than its equivalent RSPC version. This slowdown becomes more and more apparent for sizes of RSPC GF(6) and bigger, which translates to RS GF(12) and bigger.

4.3.3 Recommendations.

The processing speed of each algorithm did not substantially matter in this research as the tested sizes were small. Even for future experiments, the processing speed will most likely not be the limiting factor as MATLAB simulations can be multi-threaded and the utilized FPGA had a maximum clock frequency of 925 MHz. The selection between which RS algorithm to use essentially depends on the expected errors of

the environment. If the range of expected errors per chosen data unit size is fairly narrow, a corresponding single instance RS algorithm should be used as there is a guaranteed 100% correction rate up to its maximum correctable codewords limit. If the range of expected errors per chosen data unit size varies widely, a corresponding RSPC algorithm should be used because it has higher chances of correcting the errors than the RS algorithm if the number of errors exceeds RS's limit.

4.4 Summary

This chapter discussed the results and analysis of the experiments based on the different algorithms, error distribution modes, and utilized platforms. The results demonstrate that the effectiveness of the RSPC algorithms tend toward 100% below 75% error rate as bigger data sizes are used, and the effectiveness of the RS algorithms tend toward 100% below 50% error rate as bigger data sizes are used. The results also show that there is no statistical difference of the algorithms based on the platform they are tested on.

V. Conclusions

Satellites play a pivotal part in the functioning of the modern world. They not only enable critical services like Global Navigation Satellite System (GNSS) and provides global communication capabilities, they also monitor the earth for weather patterns, wildfires, and ocean temperatures. As valuable as they are, the harsh environmental conditions satellites must operate in degrades their hardware components and limits their operational lifetime. Extraterrestrial radiation also causes problems in the data through Single Event Upset (SEU) that flip bits from 1 to 0 or 0 to 1. Forward Error Correction (FEC) codes have been utilized in the past to mitigate the effects of SEUs, with their primary selection criteria being the algorithm's strength to correct bit flips.

5.1 Research Summary

In this research, the effectiveness of two FEC codes were tested and compared: Reed Solomon (RS) and Reed Solomon Product Code (RSPC). These algorithms were tested under very controlled parameters that dictated everything from the algorithm's data set sizes to the number of bit flips that would be introduced. The only part that was not controlled was the creation of data bits as the algorithms are to work with any given data. This controlled setting was necessary to determine exactly which factors significantly influenced the effectiveness of the algorithms.

Each combination of the $x = 2$ algorithms, $n = 13$ data set sizes, $d = 3$ error distribution modes, $r = 5$ rates of bit flips, and $t = 2$ platforms were tested $y = 30$ times. For both algorithms, performance decreased as the number of bit flips increased. The bigger Galois Field (GF)s outperformed the smaller GFs, tending towards 100% effectiveness for error rates up to 75% for RSPC and up to 50% for the RS codes. In the RS codes, the bigger GFs also outperformed the smaller GFs

that corresponded to the same RSPC sizes except for RSPC GF(3) where RS GF(5) and GF(6) only needed 1 instance each to cover all of the field elements. In these cases, the lowest GF(m) with 1 instance performed the best as they would correct more erroneous codewords than their bigger counterparts.

The algorithms also generally performed better at correcting uniformly distributed errors than with Gaussian error distribution. This was because RSPC, with its two dimensions, has less parity codewords in each dimension to correct errors with, and as most of the RS codes had to be broken into multiple instances, the concentration of errors in the middle instances exceeded the amount of erroneous codewords each instance could fix. In the cases where a single instance of RS codes was enough to cover all of the field elements, Gaussian error distribution was more easily correctable than with uniform error distribution as the positioning of the erroneous codewords did not matter. The platform on which the algorithms were tested did not significantly make a difference, so the Matrix Laboratory (MATLAB) simulations accurately portrayed the effectiveness of the algorithms as if they were implemented on a Field Programmable Gate Array (FPGA).

5.2 Research Contributions

This research demonstrated that the FEC algorithm's strength is not the only factor that influences the effectiveness of the algorithm. For both RS and RSPC, factors such as the size of the data set on which the algorithm will be implemented on, the selected GF(m) the algorithm will be based on, the number of errors expected between data checks, and the distribution of errors all play a significant role in the algorithm's performance. Based on the experiment's results, each algorithm is appropriate to use under different circumstances. RS codes should be used if the range of expected bit flips is narrow. A corresponding single instance RS code can be

implemented that will guarantee 100% correction rate up to its maximum correctable codewords limit. RSPC should be used if the range of expected bit flips varies widely. RSPC has a higher maximum correctable codewords ceiling than a corresponding RS code, and with two dimensions of error detection and correction, it has higher chances of correcting the errors. RSPC will not be able to guarantee 100% correction rate unless the number of bit flips is below $RowParity + ColParity$, but if the number of bit flips exceeds the RS's limit, RSPC has much better chances of correcting the errors.

5.3 Future Work

The results from this research can be seen as a baseline that tested the effectiveness of two classic yet popular FEC algorithms: RS and RSPC. Much more work can be done to refine these results, including:

- Test additional FEC codes: the recently established Turbo codes and low-density parity-check codes are gaining popularity in the space industry. The effectiveness of these codes can be tested and compared to the effectiveness of RS and RSPC.

- Test different data set size configurations: all of the RSPC sizes (row and column) were equivalent in size. The effectiveness might differ if the sizes for row and column were different, like in the DVD implementation of RSPC. Likewise, all of the multiple-instanced RS codes had the same instance size. If the expected error distribution mode is known beforehand, differently sized instances can be implemented in an effort to counter the expected bit flips.

- Incorporate Parity Volume Set Specification (Parchive) into RS and RSPC: when a FEC algorithm cannot correct the errors, the entire data set must

be retransmitted to the satellite from the ground station. Parchive adds an additional layer of correction by appending independent parity blocks for the data set. When there are too many errors, Parchive allows the ground station to retransmit a certain number of its parity blocks to correct the mistakes that could not be self-corrected without having to retransmit the entire data set.

Bibliography

1. Holly Zell. Radiation Belts with Satellites, 2017.

2. Martyn Riley and Iain Richardson. Reed-Solomon Codes: An Introduction to Reed-Solomon Codes: Principles, Architecture and Implementation, 1998.

3. Matt Keti. *Reed-Solomon Error-correcting Codes The Deep Hole Problem*. Phd dissertation, University of California, Irvine, 2012.

4. Philip Shirvani. *Fault-Tolerant Computing for Radiation Environments*. Phd dissertation, Stanford University, 2001.

5. Mustapha Elharoussi, Asmaa Hamyani, and Mostafa Belkasmi. VHDL Design and FPGA Implementation of a Parallel Reed-Solomon (15, K, D) Encoder/Decoder. *International Journal of Advanced Computer Science and Applications*, pages 33–37, 2013.

6. Victor Yu Krachkovsky, Yuan Xing Lee, and Hari Krishna Garg. Decoding of Parallel Reed-Solomon Codes with Applications to Product and Concatenated Codes. *IEEE International Symposium on Information Theory*, page 55, 1998.

7. Hyun-Yong Lee and In-Cheol Park. A Fast Reed-Solomon Product-Code Decoder Without Redundant Computations. *IEEE International Symposium on Circuits and Systems*, pages 381–384, 2004.

8. P. Parvathi and P. Rajendra Prasad. FPGA Based Design and Implementation of Reed-Solomon Encoder & Decoder for Error Detection and Correction. In *Conference on Power, Control, Communication and Computational Technologies for Sustainable Growth*, pages 261–266, 2015.

9. Hikmat Abdullah. Implementation of Reed-Solomon Encoder/Decoder Using Field Programmable Gate Array. *Journal of Engineering and Development*, pages 104–114, 2006.

10. Sébastien Bourdarie and Michael Xapsos. The Near-Earth Space Radiation Environment. *IEEE Transactions on Nuclear Science*, pages 1810–1832, 2008.

11. Norsuzila Ya'acob, Akmarulnizam Zainudin, Magdugal R., and Nani Fadzlina Naim. Mitigation of Space Radiation Effects on Satellites at Low Earth Orbit (LEO). In *IEEE International Conference on Control System, Computing and Engineering*, pages 56–61, 2016.

12. Space Flight Environment International Engineering Newsletter. volume 5.4, 1994.

13. Doug Sinclair and Jonathan Dyer. Radiation Effects and COTS Parts in SmallSats. In *AIAA/USU Conference on Small Satellites*, pages 1–12, 2013.

14. Dariusz Makowski. *The Impact of Radiation on Electronic Devices with the Special Consideration of Neutron and Gamma Radiation Monitoring.* Phd dissertation, Technical University of Łódź, 2006.

15. NASA STI Program. Technical report, 2012.

16. D. N. Nguyen and L. Z. Scheick. SEE and TID of Emerging Non-Volatile Memories. *IEEE Radiation Effects Data Workshop*, pages 62–66, 2002.

17. David M Hiemstra. A Review of Xilinx Field Programmable Gate Array Radiation Effects Performance. In *IEEE Canadian Conference on Electrical and Computer Engineering*, pages 1–4, 2016.

18. Chunhua Qi, Liyi Xiao, Tianqi Wang, Jie Li, and Linzhe Li. A Highly Reliable Memory Cell Design Combined with Layout-Level Approach to Tolerant Single-Event Upsets. *IEEE Transactions on Device and Materials Reliability*, pages 388–395, 2016.

19. Robert Baumann and Eric Smith. Neutron-Induced Boron Fission as a Major Source of Soft Errors in Deep Submicron SRAM Devices. In *IEEE International Reliability Physics Symposium*, pages 152–157, 2000.

20. M. R. Shaneyfelt, P. S. Winokur, T. L. Meisenheimer, F. W. Sexton, S. B. Roeske, and M. G. Knoll. Hardness Variability in Commercial Technologies. *IEEE Transactions on Nuclear Science*, pages 2536–2543, 1994.

21. Wesley Fan, Clifton Drumm, Stanley Roeske, and Gary Scrivner. Shielding Considerations for Satellite Microelectronics. *IEEE Transactions on Nuclear Science*, pages 2790–2796, 1996.

22. Craig Underwood. The Single-Event-Effect Behaviour of Commercial-Off-The-Shelf Memory Devices-A Decade in Low-Earth Orbit. *IEEE Transactions on Nuclear Science*, pages 1450–1457, 1998.

23. William Atwell, Kristina Rojdev, Sukesh Aghara, and Sirikul Sriprisan. Mitigating the Effects of the Space Radiation Environment: A Novel Approach of Using Graded-Z Materials. *American Institute of Aeronautics and Astronautics*, pages 1–12, 2013.

24. Irving S Reed and Xuemin Chen. *Error-Control Coding For Data Networks.* Kluwer Academic Publishers, Boston, 1999.

25. Scott A Vanstone and Paul C van Oorschot. *An Introduction to Error Correcting Codes with Applications.* Kluwer Academic Publishers, Boston, 1989.

26. Todd K Moon. *Error Correction Coding Mathematical Methods and Algorithms*. John Wiley & Sons Inc, New Jersey, 2005.

27. C.K.P. Clarke. Reed-Solomon Error Correction, 2002.

28. Altera. About Us: History, 2017.

29. Keith Underwood. FPGAs vs. CPUs: Trends in Peak Floating-Point Performance. In *International Symposium on Field Programmable Gate Arrays*, pages 171–180, 2004.

30. Christopher Cullinan, Christopher Wyant, and Timothy Frattesi. Computing Performance Benchmarks among CPU, GPU, and FPGA. Technical report, 2012.

31. Kosta Varnavas, William Herbert Sims, and Joseph Casas. The Use of Field Programmable Gate Arrays (FPGA) in Small Satellite Communication Systems. In *International Conference on Advances in Satellite and Space Communications*, pages 86–89, 2015.

32. Yongmei Liu, Yong Guan, Jie Zhang, Guohui Wang, and Yan Zhang. Reed-Solomon Codes for Satellite Communications. In *International Conference on Control, Automation and Systems Engineering*, pages 246–249, 2009.

33. S. Kaulgud and M. Mukherjee. FPGA Implementation of Reed-Solomon Codes. In *International Conference & Workshop on Emerging Trends in Technology*, pages 1241 – 1244, 2011.

34. Bruno Luong. Truncated Gaussian, 2010.

35. Nikos Drakos. The Central Limit Theorem, 1996.

36. Student. The Probable Error of a Mean. *Biometrika*, pages 1–24, 1908.

37. *MATLAB: The Language of Technical Computing - Getting Started with MATLAB*. The MathWorks, Natick, version 5 edition, 1997.

38. Jan Simon. DataHash, 2017.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23–03–2018	Master's Thesis	Sept 2017 — Mar 2018

4. TITLE AND SUBTITLE

An Exploration of Error-Correcting Codes for use in Noise-Prone Satellite Environments

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Kang, Min, W., 2d Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-18-M-036

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Space Vehicles Directorate
POC: Madeleine Naudeau
3550 Aberdeen Avenue, Kirtland AFB, NM 87117-5776
Email: madeleine.naudeau@us.af.mil phone: 505-293-2272

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RV

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Satellites are crucial for the modern world to function properly as they provide GNSS and global communication. However, the data that is stored on these satellites can be corrupted by the radiation found in space, and its bits can be improperly flipped. In the past, FEC algorithms were selected based on their strength and implemented to correct these bit flips back to their original values. This thesis seeks to determine if the strength of the FEC algorithms RS code and RSPC directly translates to their effectiveness. These algorithms were coded and tested in MATLAB and on a FPGA under controlled parameters, including the data set sizes, number of bit flips introduced, and the distribution of the bit flips within the data set. From the experiment's results, these other factors significantly influenced the effectiveness of the algorithms as well. Knowing what factors influence the algorithm's effectiveness enable better decision making as to which FEC algorithm to use for a given set of circumstances.

15. SUBJECT TERMS

Forward Error Correction, Reed Solomon Codes, Reed Solomon Product Codes, MATLAB, Field Programmable Gate Array

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Kenneth M. Hopkinson (ENG)
U	U	U	UU	87	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x 4579 Kenneth.Hopkinson@afit.edu