Hindawi
Journal of Electrical and Computer Engineering
Volume 2019, Article ID 3905094, 15 pages
https://doi.org/10.1155/2019/3905094

Research Article

Error Detection and Correction On-Board Nanosatellites Using Hamming Codes

Caleb Hillier and Vipin Balyan

Department of Electrical, Electronic and Computer Engineering, Cape Peninsula University of Technology, Cape Town, South Africa

Correspondence should be addressed to Vipin Balyan; vipin.balyan@rediffmail.com

Received 29 October 2018; Revised 30 December 2018; Accepted 14 January 2019; Published 10 February 2019

Academic Editor: Yagang Zhang

The field of nanosatellites is constantly evolving and growing at a very fast speed. This creates a growing demand for more advanced and reliable EDAC systems that are capable of protecting all memory aspects of satellites. The Hamming code was identified as a suitable EDAC scheme for the prevention of single event effects on-board a nanosatellite in LEO. In this paper, three variations of Hamming codes are tested both in Matlab and VHDL. The most effective version was Hamming $[16, 11, 4]_2$. This code guarantees single-error correction and double-error detection. All developed Hamming codes are suited for FPGA implementation, for which they are tested thoroughly using simulation software and optimized.

1. Introduction

Satellites have many applications, with earth observation (EO) being one of the primary applications. EO satellites use smart image sensors to monitor and capture data on the earth's surface, and in some cases, infrared is used to look beneath. This information can be used to monitor urban development, vegetation growth, natural disasters, etc. With satellite imaging constantly evolving and improving, the captured information contains higher detail and improved resolution (smaller than 1 meter) making the data more useful. This is achieved, using a wide range of spectral band cameras [1]. With constant improvement, the captured data are becoming increasingly sensitive and valuable. This results in a growing need for security, privacy, and reliability on-board satellites [1, 2]. It has been proven that satellites whilst in orbit are susceptible to unauthorized intrusions, meaning a satellite can be hacked. Cryptographic protection can be implemented to provide protection against these intrusions. Usually satellite links are modelled with an AWGN channel [3, 4]. Encryption is the most popular and conventional method adopted for cryptographic protection against unauthorized intrusions, on earth and in space. Encryption on-board EO satellites have become the norm when data are transmitted between the satellite and the ground station [5]. However, the encryption algorithms typically implemented are proprietary or outdated algorithms like DES. This raises concerns as encryption algorithms protecting potentially highly sensitive information should meet the current and latest encryption standards [6].

The Hamming error correction and detection code were first introduced by Richard Hamming in 1950. The work initially conducted on Hamming codes allowed large-scale computing machines to perform a large number of operations without a single error in the end result [7]. Hamming code is a linear block error detection and correction scheme developed by Richard Hamming in 1950 [7]. This scheme adds additional parity bits (r) to the sent information (k). The codeword bits can be calculated according to $n = 2r - 1$. This means the amount of information data bits can be calculated by $k = 2r - r - 1$. Using a parity-check matrix and a calculated syndrome, the scheme can self-detect and self-correct any single-event effect (SEE) error that occurs during transmission. Once the location of the error is identified and located, the code inverts the bit, returning it to its original form. Hamming codes form the foundation of other more complex error correction schemes. The original scheme allows single-error correction single-error detection (SECSED), but with an addition of one parity bit, an extended Hamming version allows single-error correction and

double-error detection (SECDED) [8]. Table 1 shows the original Hamming code's classification and parameters:

The Hamming code initially started as a solution for the IBM machines that crashed when an error occurred. When the code was developed, it was considered a breakthrough but no one in 1950 could have predicted it being used to ensure data reliability on-board nanosatellites. Due to the capabilities and versatility of Hamming codes, new papers are constantly published.

1.1. Background. EDAC codes are classified mainly according to the approach the code takes when performing error detection and correction. Hamming codes fall under the following classifications: Error detection and correction codes => Forward error correction (FEC) => Block => Binary codes. This is shown graphically in Figure 1.

A Hamming block is basically a codeword containing both the information signal (k-bit long) and parity bits (r-bit long). The Hamming scheme can function using a systematic or nonsystematic codeword (Table 2).

1.2. EDAC Scheme Selection. The EDAC scheme selection is almost solely dependent on its application. The work in this paper focuses on EDAC on-board a nanosatellite. This implies that a solution is needed to detect and correct errors induced by radiation on-board a nanosatellite during its low earth orbit (LEO), specifically the ZA-Cube missions.

The ZA-Cube missions are a series of nanosatellite missions being developed by French South African Institute of Technology (FSATI) in collaboration with Cape Peninsula University of Technology (CPUT). The name of this initial nanosatellite is the ZA-cube 1 [9]. The successor to ZA-cube 1 will be ZA-Cube 2, which will orbit at around 550 Km [10].

Ideally, the effects of single-error upset (SEU) and multiple-error upset (MEU) should have been monitored in ZA-cube 1; unfortunately, this was not done and therefore, this information is unavailable. However, this data were gathered by the Alsat-1, which was the first Algerian microsatellite launched into LEO. This work was published [11]. This data is a good indication of what ZA-cube 2 will be exposed to, as the Alsat-1 is also sun synchronised orbit (SSO) and LEO (ZA-cube 2 orbit being roughly 100 Km closer to earth). This means the radiation dosages for ZA-cube 2 should be slightly lower than that experienced by the Alsat-1. However, the data from the Alsat-1 can be considered as a worst case scenario for ZA-cube 2. This data is shown Table 3 [11].

Using the data provided in Table 3, it becomes apparent that SEU is by far more likely error to occur as a result of radiation. SEU, on average, cause ±97 bit flips to occur during a single day (Figure 2). Obviously, the majority (about 80%) of these errors occur while the nanosatellite is traversing through the SAA which could range roughly between 5 and 10 minutes.

Using the information shown using Table 3 and Figure 2, Hamming code will be used as the EDAC scheme implemented to deal with SEU. Hamming code allows both error detection and correction of single-bit upsets and can be extended to detect double bit flips. Once a double bit flip is detected, the information can be erased and a request to resend the information can be raised. Should the nanosatellite be traveling through the South Atlantic Anomaly (SAA), it is possible to implement triple modular redundancy (TMR) that can deal with double-byte errors and severe errors.

Hamming code was selected mainly as it meets the EDAC requirements, implementation complexity is minimum and cost of additional hardware is kept minimal. For example, commercial off-the-shelf (COTS) devices can be used.

Table 1: Hamming code classification and parameters.

Hamming code		Hamming (7, 4)
Developed by	Richard Hamming	
Type	Linear block code	
Code rate	$1-(r/(2r-1))$	
Alphabet (Σ)	$q=2$	
Block length	$n=2r-1$ where $r\geq 2$	$n=7$
Message length	$k=2r-r-1$	$k=4$
Distance	$d_{\min}\leq n-k+1$	$d_{\min}=3$
Notation	$[2r-1, 2r-r-1, d_{\min}]_2$	$[7,4,3]_2$

2. Overview of Hamming Code

In the sections to follow, a detailed overview of the Hamming code will be given. By touching on the parameters and foundation principles, the reader should have a solid understanding of the Hamming code.

Hamming code is an EDAC scheme that ensures no information transmitted/stored is corrupted or affected by single-bit errors. Hamming codes tend to follow the process illustrated in Figure 3. The input is errorless information of k-bits long which is sent to the encoder. The encoder then applies Hamming theorems, calculates the parity bits, and attaches them to the received information data, to form a codeword of n-bits. The processed information which contains additional parity bits is now ready for storage.

The type of storage is dependent on the application, which in this paper is RAM used by the on-board computer (OBC) of a nanosatellite. It is during storage that errors are most likely to occur. Errors usually occur when radiated particles penetrate the memory cells contained within the RAM. These errors are defined as bit flips in the memory. Hamming codes are capable of SECSED but can be extended to SECDED with an additional parity bit.

The decoder is responsible for checking and correcting any errors contained within the requested data. This is done by applying the Hamming theorem, which uses a parity-check matrix to calculate the syndrome. The decoder checks, locates, and corrects the errors contained in the codeword before extracting the new error-free information data.

2.1. General Algorithm. As mentioned previously, Hamming code uses parity bits to perform error detection and correction. The placement of these parity bits is dependent on

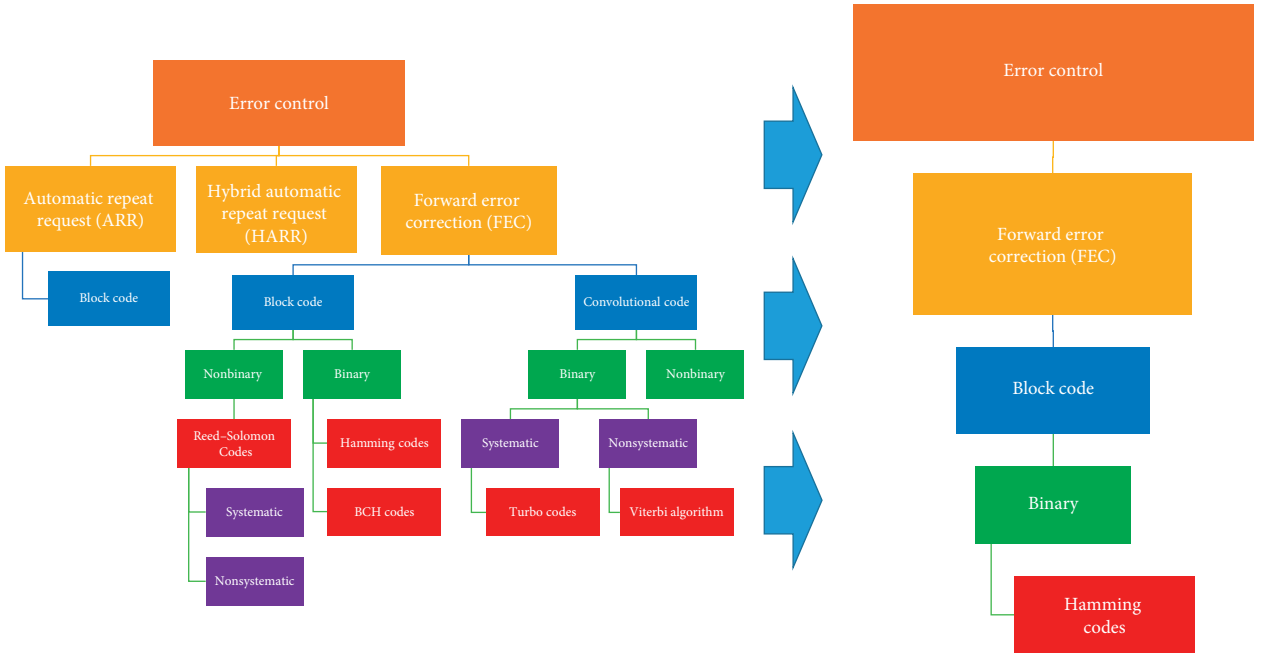

Figure 1: Classification of Hamming code.

Table 2: Systematic and nonsystematic codewords.

Systematic	$=[k_0\ k_1\ k_2\ k_3\ k_4\ k_5\ k_6\ k_7\ r_0\ r_1\ r_2]$
Nonsystematic	$=[r_0\ r_1\ k_0\ r_2\ k_1\ k_2\ k_3\ k_4\ k_5\ k_6\ k_7]$

Table 3: Memory errors that observed on Alsat-1 during a 7-year period.

Event upset observation for Alsat-1	
System monitored	OBC 386 RAMDISK memory
EDAC code	R-S (256,252)
Memory size	32 Mbytes
Hybrid	SYS84000RKXLI-70 (4M × 8-bit)
Device	Samsung SEC KM684000BLT-5L: 512K × 8-BIT SRAM
Observation period	2510 days November 29, 2002–October 12, 2009
Bits monitored	268435456
No. of single-bit errors	244150 (98.6%)
No. of multiple-bit errors	217 (0.08%)
No. of double-byte errors	2996 (1.21%)
No. of severe errors	230 (0.09%)

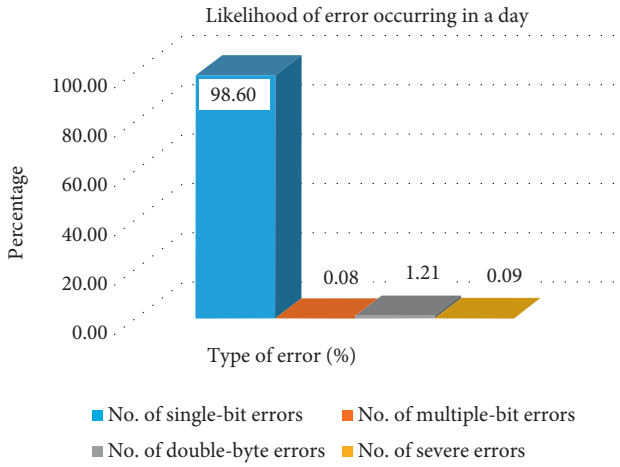

Figure 2: Likelihood of error occurring in a day.

whether or not the code is systematic or nonsystematic. In Table 4, a typical codeword layout is shown. Please note that this table only includes bit positions 1 to 16 but can continue indefinitely. From Table 4, the following observations can be made:

(i) Bit position (codeword) is dependent on the amount of data bits protected

(ii) Parity bit positions are placed according to, 2 to the power of parity bit:

(a) $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$ and $2^4 = 16$.

(iii) Parity bit's relationship to bit position and data bits (also shown using Figure 4)

(b) Parity bit 1 (P1) represents bit position: 1 (P1), 3, 5, 7, etc. (all the odd numbers)

(c) Parity bit 2 (P2) represents bit position: 2 (P2), 3, 6, 7, 10, 11, etc. (sets of 2)

(d) Parity bit 4 (P4) represents bit position: 4 (P4), 5, 6, 7, 12, 13, etc. (sets of 4)

(e) Parity bit 8 (P8) represents bit position: 8 (P8), 9, 10, 11, 12, 13, etc. (sets of 8)

(f) Parity bit 16 (P16) represents bit position: 16 (P16), 17, 18, 19, etc. (sets of 16)

(iv) The position in between the parity bits are filled with data bits.

(v) The layout makes each column have a unique parity bit combination, for each bit position.

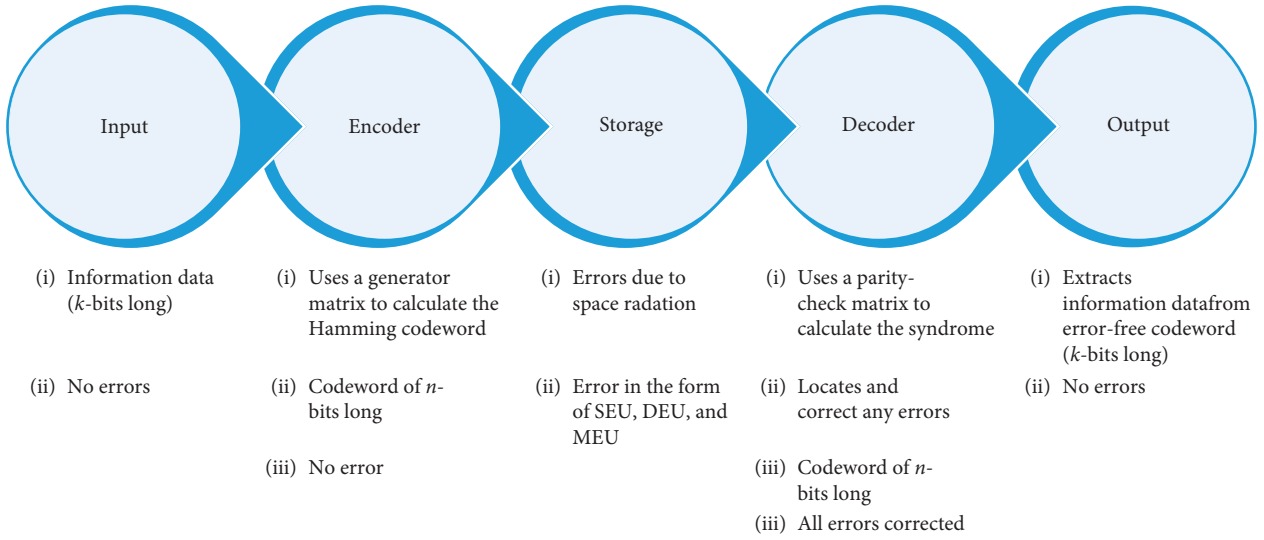

Figure 3: General layout of Hamming code.

Table 4: Bit layout of Hamming code (nonsystematic).

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	**13**	14	15	16
Encoded data bits		P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7	D8	**D9**	D10	D11	P16
Encoded data coverage (nonsystematic) syndrome	P1	✗		✗		✗		✗		✗		✗		✗		✗	
	P2		✗	✗			✗	✗			✗	✗			✗	✗	
	P4				✗	✗	✗	✗					✗	✗	✗	✗	
	P8								✗	✗	✗	✗	✗	✗	✗	✗	
	P16																✗

The syndrome allows errors to be located and corrected. For example, if parity bits P1, P4, and P8 show an error, the location of the error can be found by 1(P1) + 4(P4) + 8(P8) = 13. Therefore, the error affected data bit 9 (D9) in position 13, shown in bold. The above explanation shows the general algorithm used when implementing Hamming code.

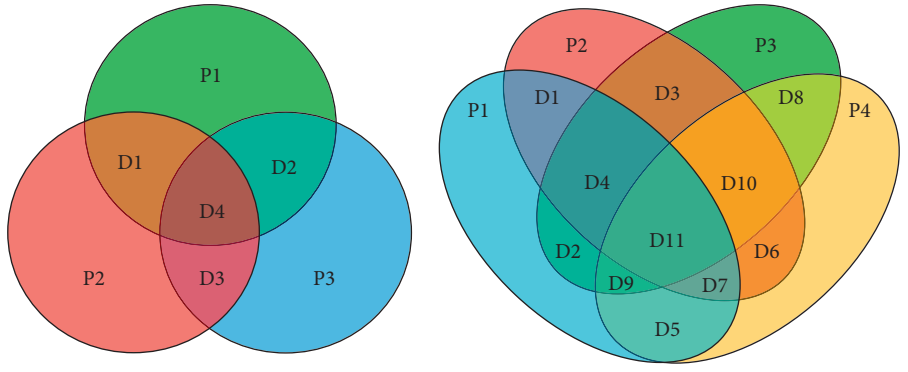

Figure 4: Parity vs data: Hamming (7, 3) (a) and Hamming (15, 11) (b).

(vi) This unique parity bit combination is known as the syndrome value.

*2.2. Construction of Generator Matrix (**G**).* Generator matrix (**G**) is used when encoding the information data to form the codeword. **G** forms one of the foundations on which the Hamming code is based. Due to the relationship between the generator matrix and parity-check matrix the Hamming code is capable of SECSED.

$\mathbf{G}_{(k\times n)}$ is defined as the combination of an identity matrix (**I**) of size $k \times k$ and a submatrix (**P**) of size $k \times r$ [12]:

$$\mathbf{G}_{k\times n} = \left[\mathbf{I}_{k\times k} \,\middle|\, \mathbf{P}_{k\times r}\right]. \quad (1)$$

*2.3. Construction of Parity-Check Matrix (**H**).* Parity-check matrix (**H**) is used when decoding and correcting the codeword, in order to extract an error-free message. **H** forms one of the foundations, on which the Hamming code is

based. Due to the relationship between the parity-check matrix and generator matrix, the Hamming code is capable of SECSEC.

$\mathbf{H}_{(r\times n)}$is defined as the combination of a negative transposed submatrix ($\mathbf{P}$) of size $k \times r$ and an identity matrix ($\mathbf{I}$) of size $r \times r$ [12]:

$$\mathbf{H}_{r\times n} = \left[-\mathbf{P}^{\mathrm{T}}_{k\times r} \mid \mathbf{I}_{r\times r}\right]. \tag{2}$$

2.4. The Relationship between G and H. The relationship between $\mathbf{G}$ and $\mathbf{H}$ is shown as

$$\mathbf{G} * \mathbf{H}^{\mathrm{T}} = 0. \tag{3}$$

As a final note on $\mathbf{G}$ and $\mathbf{H}$, it is possible to manipulate this matrix from systematic to an equivalent nonsystematic matrix by using elementary matrix operations, which are the following:

(i) Interchange two rows (or columns)

(ii) Multiply each element in a row (or column) by a nonzero number

(iii) Multiply a row (or column) by a nonzero number and add the result to another row (or column)

2.5. Hamming Encoder. The Hamming encoder is responsible for generating the codeword (n-bits long) from the message denoted by ($\mathbf{M}$) and generator matrix ($\mathbf{G}$). Once generated the codeword contains both the message and parity bits. The codeword is calculated using the formula

$$\text{codeword}_{n-\text{bits}} = \text{mod}_2\left(\mathbf{M}_{k-\text{bits}} * \mathbf{G}_{k\times n}\right). \tag{4}$$

The mathematical expression in (4) is essentially made up of AND and XOR gates. The gate/graphical expression is illustrated in (Figure 5).

2.6. Hamming Decoder. The Hamming decoder is responsible for generating the syndrome (r-bits long) from the codeword (n-bits long) denoted by $\mathbf{C}$ and parity-check matrix ($\mathbf{H}$). Once generated, the syndrome contains the error pattern that allows the error to be located and corrected. The syndrome is calculated using the formula

$$\text{syndrome}_r = \text{mod}_2\left(\mathbf{C}_{n-\text{bits}} * \mathbf{H}^{\mathrm{T}}_{r\times n}\right). \tag{5}$$

The mathematical expression in (5) is essentially made up of AND and XOR gates. The gate/graphical expression is illustrated in Figure 6.

2.7. Extended Hamming Code. The extended Hamming code makes use of an additional parity bit. This extra bit is the sum of all the codeword bits (Figure 7), which increases the Hamming code capabilities to SECDED. The extended Hamming code can be done in both systematic and nonsystematic forms. Table 5 shows this in a graphical manner.

P16, in this case, is the added parity bit that allows double error detection. By monitoring this bit and checking whether the bit is odd or even allows the double bit error to be detected, which is shown in Table 5 by bold.

3. Hamming $[16, 11, 4]_2$

Hamming $[16, 11, 4]_2$ is an extended version of Hamming code. With an additional parity bit, the code is capable of double error detection (DED). This code was implemented in a nonsystematic manner, which simplifies the detection of double errors. The construction of the generated codeword is shown in Table 6. Using general formulas, some performance aspect of the Hamming $[16, 11, 4]_2$ can be calculated (Table 7). This clearly proves that Hamming $[16, 11, 4]_2$ has a better code rate and bit overhead than Hamming $[8, 4, 4]_2$, with capabilities of SECSED.

In order to implement the Hamming $[16, 11, 4]_2$, a generator matrix (G) and parity-check matrix (H) are needed for the encoding, decoding, and the calculation of the syndrome (S). The following calculations were done in order to implement the code:

Submatrix ($\mathbf{P}$) of 11×4 dimensions is shown as

$$\mathbf{P}_{11\times 4} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}. \tag{6}$$

Identity matrix ($\mathbf{I}$) of size 11×11 (7) is used when calculating $\mathbf{G}$:

$$\mathbf{I}_{k\times k} = \mathbf{I}_{11\times 11} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7}$$

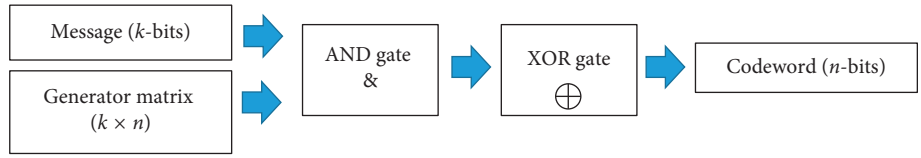

Figure 5: Gate/graphical expression of hamming encoder formula.

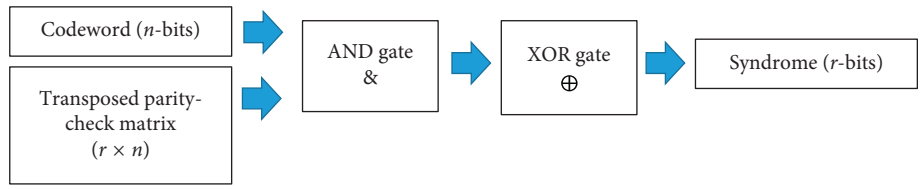

Figure 6: Gate/graphical expression of hamming decoder formula.

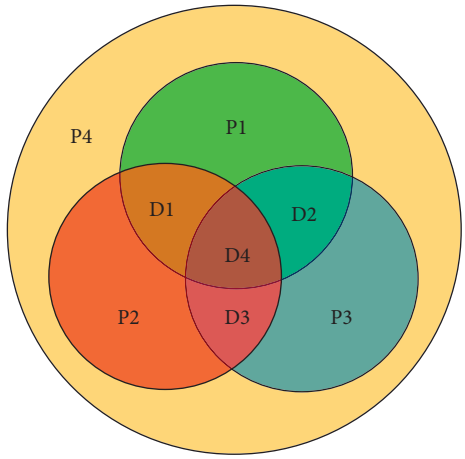

Figure 7: Parity relationship to data, extended Hamming (8, 4).

From (1), the generator matrix is calculated:

$$\mathbf{G}_{11\times15} = \left[\mathbf{I}_{11\times11} \,\middle|\, \mathbf{P}_{11\times4}\right] = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0&1&1&0\\ 0&0&1&0&0&0&0&0&0&0&0&0&0&1&1\\ 0&0&0&1&0&0&0&0&0&0&0&1&1&0&1\\ 0&0&0&0&1&0&0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&1&0&1\\ 0&0&0&0&0&0&1&0&0&0&0&0&1&1&1\\ 0&0&0&0&0&0&0&1&0&0&0&1&1&1&0\\ 0&0&0&0&0&0&0&0&1&0&0&1&1&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&1&0&1&1\\ 0&0&0&0&0&0&0&0&0&0&1&1&0&0&1 \end{bmatrix}. \quad (8)$$

Following Hamming rules, **G** should be of size $k \times n$. Therefore, an additional column is needed to satisfy this condition for Hamming $[16, 11, 4]_2$. An 8^{th} parity column is added to **G**. This is done by adding an odd or even parity bit to each row:

$$\mathbf{G}_{k\times n} = \mathbf{G}_{11\times16} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1\\ 0&1&0&0&0&0&0&0&0&0&0&1&1&0&1\\ 0&0&1&0&0&0&0&0&0&0&0&0&1&1&1\\ 0&0&0&1&0&0&0&0&0&0&1&1&0&1&0\\ 0&0&0&0&1&0&0&0&0&0&1&0&1&0&1\\ 0&0&0&0&0&1&0&0&0&0&0&1&0&1&1\\ 0&0&0&0&0&0&1&0&0&0&1&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&1&1&1&0\\ 0&0&0&0&0&0&0&0&1&0&0&1&1&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&1&0&1&1\\ 0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1 \end{bmatrix}. \quad (9)$$

By considering the nonsystematic Hamming bit layout in Table 4, **G** calculated in (9) can be rearranged to form a nonsystematic matrix **G**:

$$\mathbf{G}_{11\times16} = \begin{bmatrix} 1&1&1&0&0&0&0&1&0&0&0&0&0&0&0&0\\ 1&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0\\ 1&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1\\ 0&1&0&0&0&0&1&1&0&0&0&0&0&0&0&0\\ 1&1&0&1&0&0&0&0&1&0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0&1&0&1&0&0&0&0&0&1\\ 0&1&0&1&0&0&0&1&0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\ 1&1&0&1&0&0&0&1&0&0&0&0&1&0&0&1\\ 0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1\\ 1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \end{bmatrix}. \quad (10)$$

Identity matrix (**I**) of size 4×4 is used to calculate **H**. From (2), the parity-check matrix can be calculated. With the additional parity bit, the negative submatrix is not used, as the result is the same. **H** is calculated in (11).

Table 5: Bit layout of extended Hamming code (16, 11).

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Encoded data bits		P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7	D8	D9	D10	D11	P16
Encoded data coverage (nonsystematic) syndrome	P1	✗		✗		✗		✗		✗		✗		✗		✗	
	P2		✗	✗			✗	✗			✗	✗			✗	✗	
	P4				✗	✗	✗	✗					✗	✗	✗	✗	
	P8								✗	✗	✗	✗	✗	✗	✗	✗	
	P16																✗

Table 6: Construction of the Hamming [16, 11, 4] codeword.

Codeword ($n = 16$)																
Information data ($k = 11$)											Parity ($r = 5$)					
D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	P1	P2	P3	P4	P5	

Table 7: Calculated performance aspects of Hamming [16, 11, 4].

Scheme	E_d	E_c	Code rate (%)	Bit overhead (%)
Hamming $[16, 11, 4]_2$	2	1	68.75	45.45

$$\mathbf{H}_{4\times15} = \left[\mathbf{P}^{T}_{11\times4} \,\middle|\, \mathbf{I}_{4\times4}\right]_{4\times15} = \begin{bmatrix} 1&0&0&1&1&0&1&0&1&1&1&1&0&0&0 \\ 1&1&0&1&0&1&1&1&1&0&0&0&1&0&0 \\ 0&1&1&0&1&0&1&1&1&1&0&0&0&1&0 \\ 0&0&1&1&0&1&0&1&1&1&1&0&0&0&1 \end{bmatrix}. \quad (11)$$

Hamming rules state **H** should be of size $r \times n$. Therefore, an additional column and row are needed to satisfy this condition for Hamming $[16, 11, 4]_2$. Due to the fourth parity bit only being used for detection, the entire row can be filled with ones. With a fourth row added, an 8^{th} parity column is added to **H**. This is done by adding an odd/even parity bit to each row.

$$\mathbf{H}_{r\times n} = \mathbf{H}_{5\times16} = \begin{bmatrix} 1&0&0&1&1&0&1&0&1&1&1&1&0&0&0&0 \\ 1&1&0&1&0&1&1&1&1&0&0&0&1&0&0&0 \\ 0&1&1&0&1&0&1&1&1&1&0&0&0&1&0&0 \\ 0&0&1&1&0&1&0&1&1&1&1&0&0&0&1&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{bmatrix}. \quad (12)$$

By considering the nonsystematic Hamming bit layout in Table 4, **H** calculated above can be rearranged to form a nonsystematic matrix **H**:

$$\mathbf{H}_{5\times16} = \begin{bmatrix} 1&0&0&0&0&1&1&0&0&1&0&1&1&1&0&1 \\ 0&1&1&0&0&1&0&0&1&1&1&1&0&0&0&1 \\ 0&0&1&0&1&0&1&1&0&1&1&1&1&0&0&0 \\ 0&0&0&1&1&1&0&0&1&0&1&1&1&1&0&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{bmatrix}. \quad (13)$$

Once **G** and **H** have been derived, it is possible to encode and decode the information data in a manner that is capable of SECDED. The data is encoded using (4). This was implemented in Matlab using mod-2 additions which are essentially exclusive-OR functions. The same equation was used to encode the information data in VHDL, however, exclusive-OR gates where used to perform the operation.

Hamming $[16, 11, 4]_2$ makes use of syndrome decoding to locate and correct any errors that occurred in the codeword during transmission or storage. The syndrome is calculated according to (5). A for-loop or case statement can then be used to locate the bit which is flipped within the codeword. Once located, the bit is simply inverted, returning it back to its original and correct state. With the codeword now error-free, the information bits are separated and extracted, returning an error-free 11-bit message to the receiver.

4. Implementation

Hamming code was implemented in both Matlab and VHDL. The approach taken to achieve the desired results is explained with the help of detailed flow charts.

4.1. Matlab. For each variation of the Hamming codes, a proof of concept model was designed in Matlab. The approach is outlined in Figure 8.

4.2. VHDL. Once proof of concept was established using Matlab, Quartus was used to implement the working VHDL model. The Hamming code was programmed using VHDL as it allows the behaviour of the required system to be modelled and simulated. This is a major advantage when optimization is required. A working model of Hamming $[8, 4, 4]_2$ and Hamming $[16, 11, 4]_2$ was programmed in VHLD using the approached detailed in Figure 9.

5. VHDL Optimization of Hamming $[16, 11, 4]_2$

In this section, the optimization of Hamming $[16, 11, 4]_2$ is done. The aim of optimization is to reduce resource usages, reduce time delays, improve efficiency, etc.

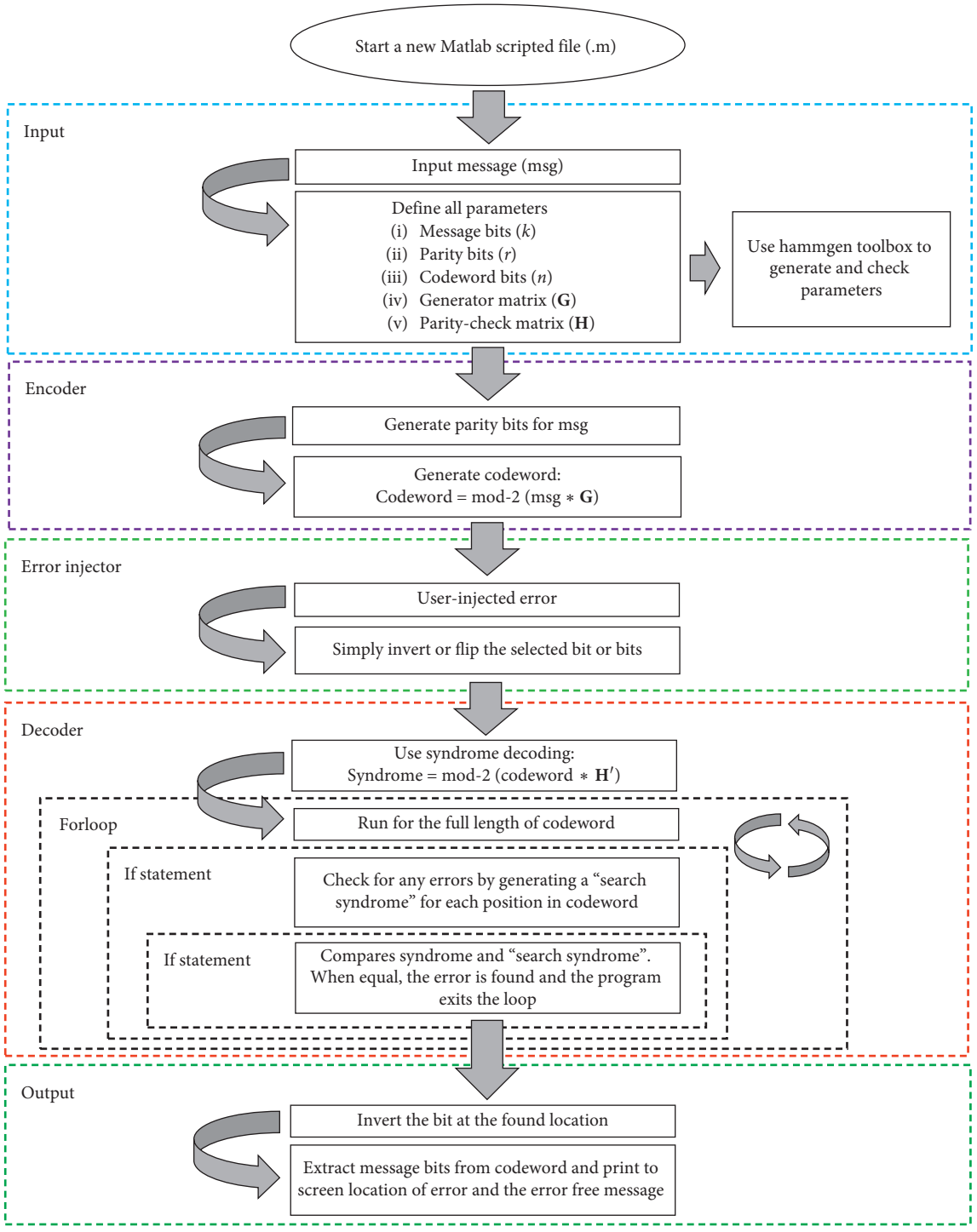

Figure 8: Overview of Matlab code (flow chart).

5.1. Code Optimization. There are many ways of optimizing VHDL code. Some of the main topics when it comes to optimization are [13]

(i) Efficient adder implementation
(ii) State machines
(iii) Signal selection
(iv) Storage structure
(v) Placement and Routing

In this paper, the Quartus Prime package is used to code, analyse, compile, and optimize the Hamming code.

5.2. Register Transfer Level (RTL) Viewer of Hamming [16, 11, 4]$_2$. Using the RTL viewer provided under tools in Quartus Prime, an overview of the I/O and VHDL code layout can be seen in Figure 10. The overview displays the input and outputs, as well as the components defined in hamming_11_16_main.vhd.

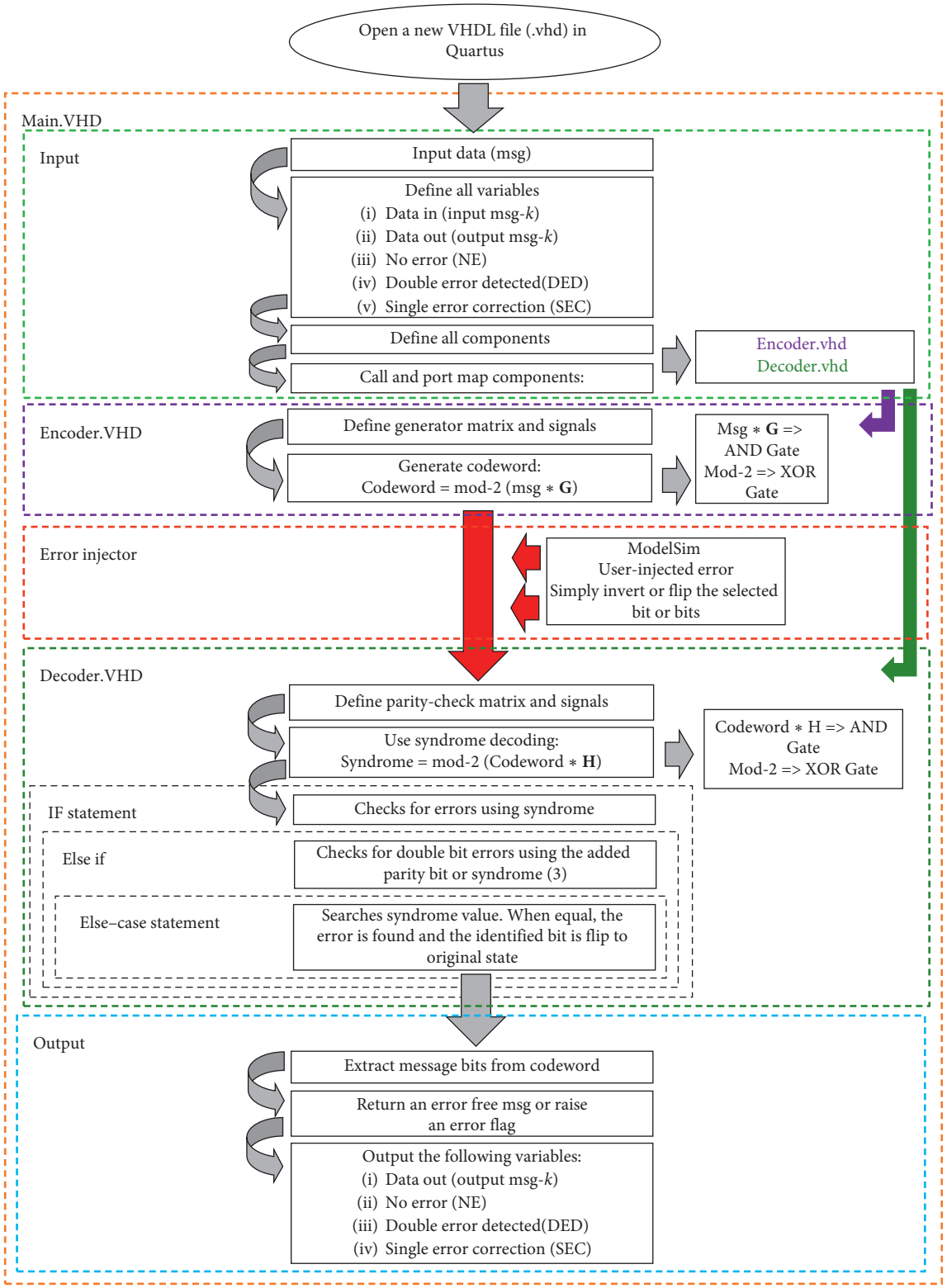

Figure 9: Overview of VHDL code (flow chart).

Using the RTL viewer tool, it is possible to step into both the encoder and decoder. The RTL viewer optimizes the netlist in order to maximize readability. This allows a unique insight into each VHDL code. The RTL view of the encoder and decoder is shown in Figures 11 and 12.

5.3. Optimization of Hamming [16, 11, 4]$_2$. The following steps are taken to optimize the code:

(i) Remove unnecessary and redundant code

(ii) Reduce constants and variables where possible

Figure 10: I/O overview of VHDL code for Hamming $[16, 11, 4]_2$.

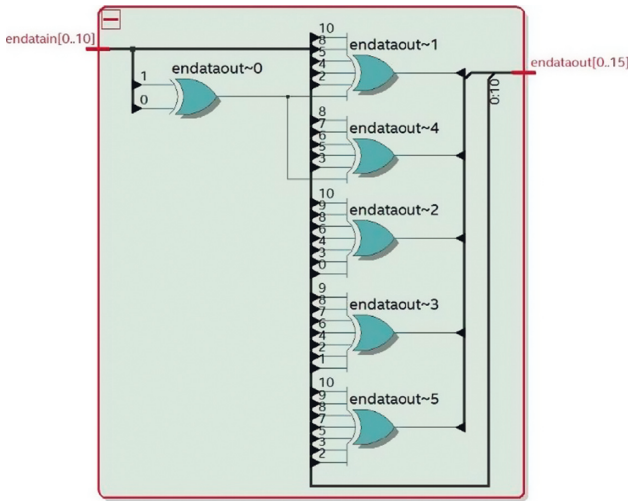

Figure 11: RTL overview of the encoder for Hamming $[16, 11, 4]_2$.

(iii) Minimize the use of if statements and loops

(iv) Convert code to structural level or gate level

The Hamming $[16, 11, 4]_2$ was reduced to structure or gate level, which resulted that all redundant codes, IF statements, loops, constants, and variables were either removed or reduced. By reducing the code to gate level, the following changes occurred:

(i) Encoder contains no constants or variables

(ii) Encoder went from performing 320 logic (AND and XOR) bit operations to only 30 XOR operations

(iii) Decoder contains no constants

(iv) Decoder contains a reduced amount of variables

(v) Decoder went from 2 IF statements to none and from 1 case statement to none

The results of reducing Hamming $[16, 11, 4]_2$ to gate level is shown for the decoder using Figure 13.

6. Testing and Simulations

Hamming codes have many communication and memory applications. They are extremely popular for their effectiveness when it comes to correction of single-bit flips and the detection of double-bit flips.

Hamming $[16, 11, 4]_2$ allows SECDED, while providing a better code rate and bit overhead than standard Hamming codes. Hamming $[16, 11, 4]_2$ generates a codeword of double-byte size, which is convenient as most memory blocks work on a byte standard. The code was implemented in VHDL on both a behaviour/dataflow and gate level (optimized) during implementation. Hamming $[16, 11, 4]_2$ code was optimized from a resource reduction and timing approach, after which it was tested thoroughly using software techniques.

6.1. Top-Level Requirements. The top-level design in Figure 14 shows a proposed EDAC system. The i386 microcontroller communicates using 11 bits of data over the D-Bus. The data bits together with the 5 bits parity allow a double-byte codeword to be stored in the RAM. This layout ensures that SEUs are detected and corrected during memory read and write cycles.

6.2. Software Tests and Reports. Using software, the developed Hamming code is tested and analysed. This is done on three levels, namely, functionality (ModelSim), resource usage (compilation report), and timing analysis (TimeQuest). An overview of the tested VHDL code is shown in Figure 15. Figure 15 shows the inputs and outputs (I/O), I/O registers, clocking circuit, and the components that make up the Hamming $[16, 11, 4]_2$ code.

6.3. Functionality. Hamming $[16, 11, 4]_2$ is capable of single-error correction and double-error detection. With the help of ModelSim, this is clearly shown in Figure 16. I/O registers are triggered using the rising edge of clk and can be cleared using clr. This will allow synchronisation and enables the OBC to do a full EDAC reset if necessary. Datain (input), dataout (output), and data_to_memory (stored codeword) display the data in the system, while NE (no error), DED (double error detection), and SEC (signal error correction) serve as indication flags. In Figure 16, the functionality of Hamming $[16, 11, 4]_2$ is proven. In Figure 16, the following should be noted:

(i) All registers are cleared using the clear signal "clr" (this is shown using ( ))

(ii) Single-bit errors are introduced in memory using a bit flip in data_to_memory (bits 0 to 15) and flagged by SEC (this is shown using ( ))

(iii) Double-bit errors are introduced in memory using bit flips in data_to_memory (bits 0 to 15) and flagged by DED (this is shown using ( ))

(iv) Note: thanks to Hamming $[16, 11, 4]$, dataout is unaffected by single-bit errors and only gets cleared upon the detection of double-bit errors.

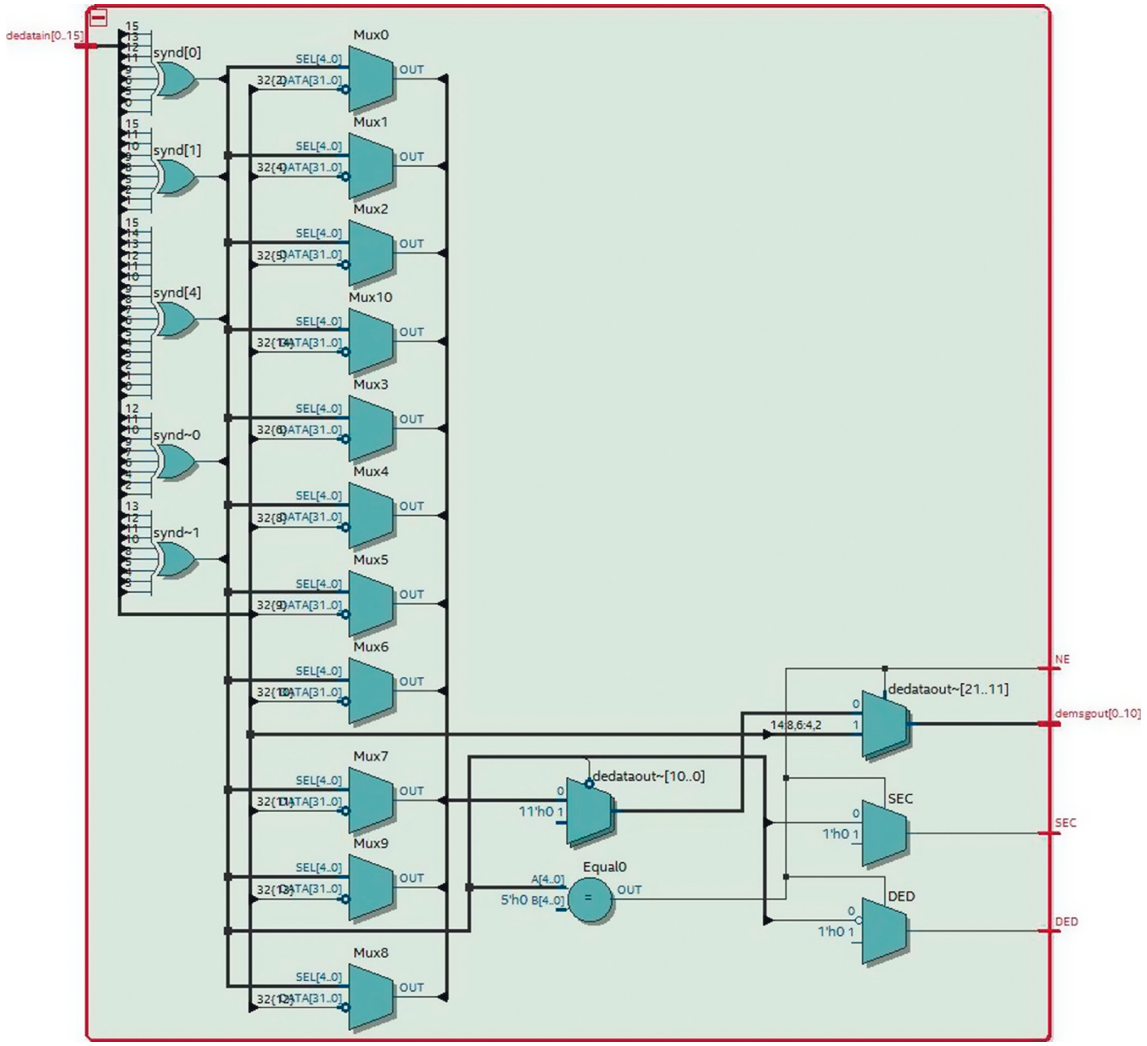

Figure 12: RTL overview of the decoder for Hamming $[16, 11, 4]_2$.

In this paper, EDAC schemes are extensively discussed. ZA-cube 2 nanosatellite was used as a case study when conducting research on space radiation, error correction codes, and Hamming code. All findings, results, and recommendations are concluded in the sections to follow.

6.4. Findings. Space radiation has caused numerous mission failures. Through research, it became apparent that failures caused by SEU and MEU are extremely common and SEEs are more frequent while the nanosatellite is in the SAA. It was found that there are a number of EDAC schemes and techniques currently used, most commonly Hamming, RS codes, and TMR. The EDAC schemes are usually implemented using an FPGA. From the literature survey, it is clear that there is a need for research in the area of EDACs. By conducting an in-depth literature review, it was established that Hamming code was capable of performing the functionality desired.

In order to understand space radiation, a study was conducted using the orbital parameters of nanosatellite ZA-Cube 2. This radiation study was conducted using OMERE and TRIM software which allows the simulation of earth radiation belts (ERBs), galactic cosmic radiation (GCR), solar particle events (SPE), and shielding. In the case of ERBs, protons of a maximum integral flux of $1.65 \times e^{3^{\circ}} cm^{-2} \cdot s^{-1}$ flux at energy ±100 KeV were trapped, which decays to a minimum integral flux of $1.55\,cm^{-2}\,s^{-1}$ flux at energy ±300 MeV. It was found that the common nanosatellite casing of 2 mm Al is only effective as a shield against protons below 20 MeV and heavy ions below 30 MeV. In order to ensure that SEE does not occur, additional mitigation techniques are needed to protect sensitive/vulnerable devices. These techniques could be triple modular redundancy (TMR), software EDAC schemes, and others.

There are two types of ECC, namely, error detection codes and error correction codes. For protection against radiation, nanosatellites use error correction codes like Hamming, Hadamard, Repetition, Four Dimensional Parity, Golay, BCH, and Reed Solomon codes. Using detection capabilities, correction capabilities, code rate, and bit overhead, each EDAC scheme is evaluated. The evaluation of Hamming codes is shown in Table 8. Nanosatellites in SSO LEO of around 550 km experience on

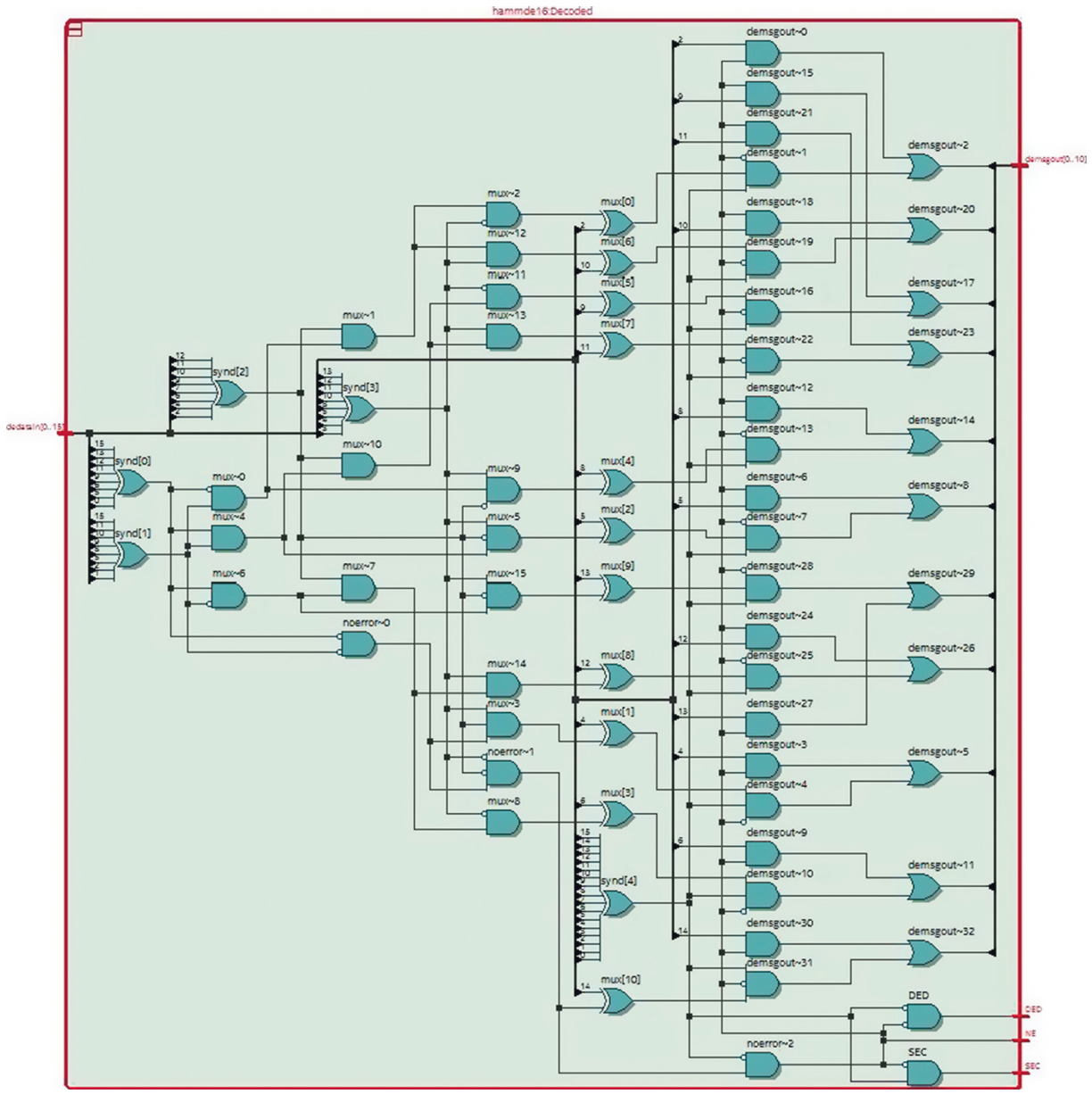

Figure 13: RTL overview of the optimized decoder for Hamming $[16, 11, 4]_2$.

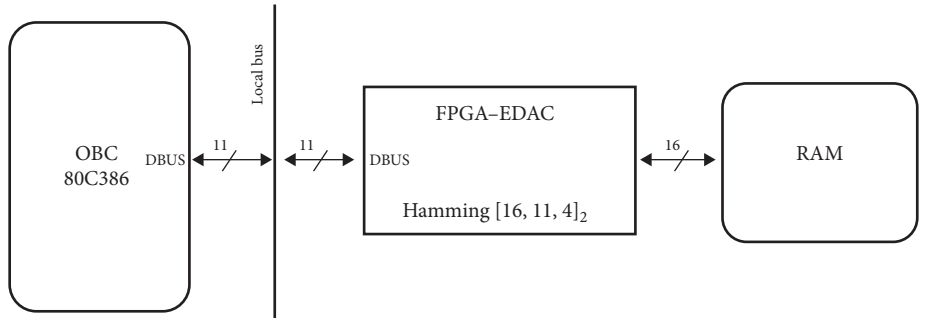

Figure 14: Functional block diagram of Hamming $[16, 11, 4]_2$.

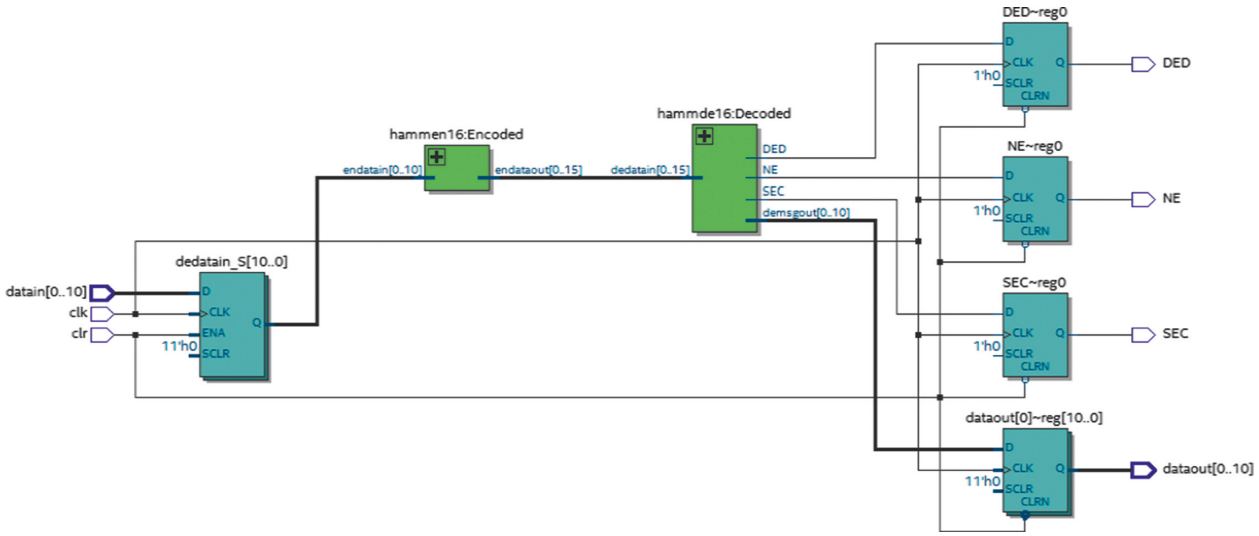

Figure 15: Full RTL overview of VHDL code for Hamming $[16, 11, 4]_2$.

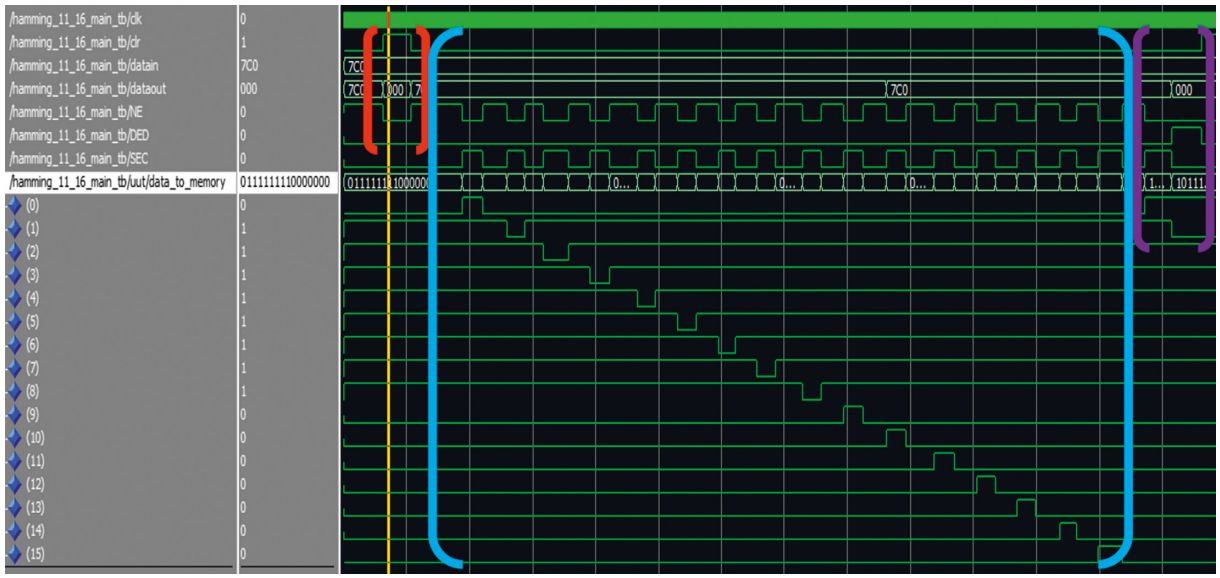

Figure 16: ModelSim simulation of Hamming [16, 11, 4] SECDED capabilities.

Table 8: Evaluation of Hamming code.

Schemes (n, k, D_{min})			E_d	E_c	Code rate (%)	Bit overhead (%)
Hamming						
7	4	3	1	1	57.14	75.00
15	11	3	1	1	73.33	36.36
31	26	3	1	1	83.87	19.23
63	57	3	1	1	90.48	10.53
Extended hamming						
8	4	4	2	1	50.00	100.00
16	11	4	2	1	68.75	45.45
64	58	4	2	1	90.63	10.34
1024	1017	4	2	1	99.32	0.69

average ±97 SEU bit flips per day, with an average of 98.6% of all errors being SEU. Around 80% of all errors occur within the SAA region.

Hamming codes are classified as error detection and correction codes that are forward error correction block binary codes. These codes are based on the use of parity bits which allows EDAC using a generator matrix and a parity-check matrix. Normal Hamming codes make use of a syndrome decoder which ultimately allows the error to be located. Once located, the error is corrected to its original state. Three variations of Hamming was designed and tested during the completion of this paper. A short summary of these codes is shown in Table 9.

Hamming $[16, 11, 4]_2$ was then converted to gate level and optimized from a resource approach and timing approach. The results of the optimized code are shown in Table 10. With the Hamming code design being complete, hardware tests were considered. The international JEDEC standards are recommended when conducting a proton (JESD234) and heavy ion test (JESD57A). These standards

Table 9: Summary of developed codes.

Hamming scheme	Variation	Approach	Capabilities	Improvements	Matlab model	VHDL model
Hamming $[7, 4, 3]_2$	Original	Systematic	SECSED	None	Yes	Yes
Hamming $[8, 4, 4]_2$	Extended	Nonsystematic	SECDED	Hamming distance	Yes	No
Hamming $[16, 11, 4]_2$	Extended	Nonsystematic	SECDED	Hamming distance, overhead, and code rate	Yes	Yes

Table 10: Original vs optimized Hamming compression.

Description		Hamming $[16, 11, 4]_2$		
		Original	Resource reduction	Timing reduction
Resource summary	ALM	12	12	12
	ALUT	13	13	13
	7-input	0	0	0
	6-input	7	2	7
	5-input	0	0	0
	4-input	0	0	0
	≤3-input	6	22	6
	Dedicated logic registers	24	24	24
	I/O pins	27	27	27
	Max fan-out	24	24	24
	Total fan-out	165	145	165
	Average fan-out	1.81	1.59	1.81
Timing analysis	Clock period	1 ns	2.25 ns	1.8 ns
	From node	Datain_s[5]	Datain_s[2]	Datain_s[8]
	To node	Dataout_s[10]~reg0	Dataout[1]~reg0	Dataout[8]~reg0
	Data arrival time	5.624 ns	5.987 ns	5.394 ns
	Data required time	4.827 ns	6.076 ns	5.491 ns
	Slack	−0.797 ns (violation)	0.089 ns	0.097 ns

ensure reliable and accurate test results. iTemba lab in Cape Town houses an accelerator capable of proton (up to 200 MeV) and heavy ion testing. In order to conduct SEE tests at a facility like iTemba lab, detailed and extensive planning is necessary. This planning and actual testing are done by both the ion beam operators and satellite engineers. This ensures cost and time is kept to a minimal.

7. Conclusion

In this paper, a detailed look at different EDAC schemes, together with a code comparison study was conducted. This study provides the reader with a good understanding of all common EDAC schemes, which is extremely useful should different EDAC capabilities be needed.

Hamming code was extensively studied and implemented using different approaches, languages, and software. The final version of the Hamming code was Hamming $[16, 11, 4]_2$. This code allows SECDED. Using only 12 adaptive logic modules (ALMs), the code is extremely small, meaning the selected FPGA will consume a minimal amount of power. By optimizing the resource usage, the average fan-out can be reduced from 1.81 to 1.59 and runs on a period of 1.8 ns with no violation and an arrival time of 5.987 ns. When optimized from a timing perspective, the code can be optimized to runs off a clock period of 1.8, with no violations and an arrival time of 5.394 ns.

Due to Hamming $[16, 11, 4]_2$ resource usage of the original code already being so small, the timing optimization approach is recommended it is 0.593 ns faster. This implies that a Hamming code was developed capable of protecting 11 bits of information against SEU and capable of DED. The code is capable of reading and writing to memory within 5.394 ns using only 12 ALMs and 24 registers.

The data from Alsat-1 is considered as a worst case scenario. This means Hamming $[16, 11, 4]_2$ is theoretically capable of preventing all single-bit errors (SBE) or ±98.6% of all SEUs experienced onboard ZA-cube 2. Hamming $[16, 11, 4]_2$ detection capability also allows the prevention of some multiple-bit errors (MBE). Should the nanosatellite be traveling through the South Atlantic Anomaly (SAA), it is possible to implement triple modular redundancy (or a similar EDAC scheme) together with Hamming $[16, 11, 4]_2$. This will equip the nanosatellite to deal with double-byte errors and severe errors.

The field of nanosatellites is constantly evolving and growing at an astonishing pace! This is due to the fact that it provides a platform from which the boundaries of space and technology are constantly being pushed. As technology advances memory chip cell architecture is becoming more and denser, especially with the development of nanotechnology. This creates a growing demand for a more advanced and reliable EDAC system that is capable of protecting all memory aspects of satellites. The code developed in this paper guarantees SECDED at fast speeds.

In this paper, EDAC schemes with a focus on Hamming codes are extensively discussed and studied. The ZA-cube 2 nanosatellite was used as a case study when conducting research on error correction codes, Hamming codes, and the optimizing of Hamming codes.

Data Availability

The data wherever required is referenced in the paper.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] R. Banu and T. Vladimirova, "On-board encryption in earth observation small satellites," in *Proceedings 40th Annual 2006 International Carnahan Conference on Security Technology*, pp. 203–208, Lexington, KY, USA, October 2006.

[2] J. Daemen and R. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, Spriger-Verlag publication, Berlin, Germany, 1st edition, 2002.

[3] W. Sun, J. P. S. Mba, and P. M. Sweeting, "Micro-minisatellites for affordable EO constellations: RapidEye and DMC," in *Proceedings of IAA Symposium on Small Satellites for Earth Observation*, Berlin, Germany, April 2001.

[4] W. E. Burr, "Selecting the advanced encryption standard," *IEEE Security & Privacy Magazine*, vol. 1, no. 2, pp. 43–52, 2003.

[5] S. P. Bain, "The increasing threat to satellite communications," *Online Journal of Space Communication*, vol. 6, no. 6, 2003.

[6] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 492–505, 2003.

[7] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[8] L.-J. Saiz-Adalid, P. Gil, J.-C. Baraza-Calvo, J.-C. Ruiz, D. Gil-Tomás, and J. Gracia-Morán, "Modified hamming codes to enhance short burst error detection in semiconductor memories," in *Proceedings of the Tenth European Dependable Computing Conference*, pp. 62–65, Newcastle, UK, May 2014.

[9] F. Cput, *Zacube-1*, French South African, Institute of Technology, Bellville, South Africa, 2017.

[10] D. De Villiers and R. Van Zyl, *ZACube-2 : the Successor to Africa's First Nanosatellite*, French South African, Institute of Technology, Bellville, South Africa, 2018.

[11] Y. Bentoutou, "A real time EDAC system for applications onboard earth observation small satellites," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 1, pp. 648–657, 2012.

[12] J. s. Chitode, *Analog And Digital Communication*, Technical Publication, Pune, India, 2nd edition, 2009.

[13] M. Gschwind and V. Salapura, *Optimizing VHDL Code for FPGA Targets*, University of Sussex, Brighton, UK, 2014.