FEDERAL UNIVERSITY OF CEARÁ

TECHNOLOGY CENTER

DEPARTAMENT OF TELEINFORMATICS ENGINEERING

POSTGRADUATE PROGRAM IN TELEINFORMATICS ENGINEERING

PHD THESIS IN TELEINFORMATICS ENGINEERING

DAVID CIARLINI CHAGAS FREITAS

TWO-DIMENSIONAL ERROR CORRECTION CODE PROPOSALS TARGETING SPACE APPLICATION MEMORY REQUIREMENTS

FORTALEZA

2021

DAVID CIARLINI CHAGAS FREITAS

TWO-DIMENSIONAL ERROR CORRECTION CODE PROPOSALS TARGETING SPACE APPLICATION MEMORY REQUIREMENTS

Thesis submitted to the Postgraduate Program in Teleinformatics Engineering at Federal University of Ceará as part of the requirements for the PhD degree in Teleinformatics Engineering.
Area: Signals and Systems.
Advisor: Prof. Dr. João Cesar Moura Mota

Co-advisor: Prof. Dr. Jarbas Aryel Nunes da Silveira

FORTALEZA

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F936t Freitas, David Ciarlini Chagas.
Two-Dimensional Error Correction Code Proposals Targeting Space Application Memory Requirements / David Ciarlini Chagas Freitas. – 2021.
135 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2021.
Orientação: Prof. Dr. Joao Cesar Moura Mota.
Coorientação: Prof. Dr. Jarbas Aryel Nunes da Silveira.

1. Error Correction Code. 2. Fault Tolerance. 3. Radiation Effect. 4. Memory Reliability. I. Título.

CDD 621.38

DAVID CIARLINI CHAGAS FREITAS

TWO-DIMENSIONAL ERROR CORRECTION CODE PROPOSALS TARGETING SPACE APPLICATION MEMORY REQUIREMENTS

Thesis submitted to the Postgraduate Program in Teleinformatics Engineering at Federal University of Ceará as part of the requirements for the PhD degree in Teleinformatics Engineering. Area: Signals and Systems.

Approved on 20/12/2021

THESIS DEFENSE COMMITTEE

Prof. Dr. João Cesar Moura Mota (Advisor)
Federal University of Ceará (UFC)

Prof. Dr. Jarbas Aryel Nunes da Silveira (Co-advisor)
Federal University of Ceará (UFC)

Prof. Dra. Lirida Alves de Barros Naviner
Télécom Paris

Prof. Dr. Cesar Augusto Missio Marcon
Pontifical Catholic University of Rio Grande do Sul (PUC-RS)

Prof. Dr. Fabian Luis Vargas
Pontifical Catholic University of Rio Grande do Sul (PUC-RS)

Prof. Dr. Walter da Cruz Freitas Júnior
Federal University of Ceará (UFC)

Prof. Dr. Giovanni Cordeiro Barroso
Federal University of Ceará (UFC)

This thesis is dedicated to my family.

ACKNOWLEDGEMENTS

First of all, I would like to thank my family, especially my parents, my grandparents, my brother and my wife for always encouraging me and giving me all support I need to continue on this journey.

I would also like to thank Professors João César and Jarbas for the supervision, giving me very good insights into my work, as well as to Professor Marcon for the essential support throughout my research and Professor Lirida for her welcome to Télécom Paris and all support given in the research.

My gratitude is also extended to all my friends from the LESC laboratory and the UFC for all professional and personal moments experienced there and to IFCE for giving me time away from my professional activities.

Also, I would like to thank all the members of the Jury for accepting being examiners of this work.

Finally, I would like to thank all family and friends who, directly or indirectly, contributed to the realization of this work.

“The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom"

(Isaac Asimov)

ABSTRACT

The integrated circuit shrinkage increases the probability and the number of errors in memories due to the increase in the sensitivity to radiation. Critical memory systems employ Error Correction Codes (ECC) to mitigate these failures. Nowadays, one-dimensional ECCs fail to achieve the effectiveness needed to address the increasing number of bit flips caused by a single radiation event. Consequently, n-dimensional ECCs have been proposed to provide higher error detection and correction power. These complex ECCs built for use in critical applications increase error correction and detection capacity but implying higher redundancy, area usage, energy consumption, and critical path delay. We focus on two-dimensional ECCs, also called product codes, designed to protect memories used in space applications. It is not yet clear how the structure of a two-dimensional code and its decoding algorithm influence the correction rate and its associated cost. Therefore, this thesis aims to develop three new approaches and new decoding techniques, always focusing on the maximum correction capability of this class of ECCs with the lowest possible cost of hardware implementation. The first proposal is the Product Code for Space Application (PCoSA), an ECC product based on Hamming and parity in both rows and columns for use in memory with space-application reliability requirements. The potentialities of PCoSA were evaluated by injecting (i) thirty-six predefined error patterns and (ii) all possible combinations of up to seven bitflips. This thesis also introduces the Optimized Product Code for Space Application (OPCoSA), an ECC that optimizes its original version PCoSA, reducing 16-redundancy bits and keeping high error correction capacity. This optimized ECC was evaluated through tests with 36 specific error patterns, burst errors, and exhaustive analysis. Additionally, synthesis results in hardware, reliability, and redundancy to four other ECCs dedicated to the space application were evaluated. The last proposal is Line Product Code (LPC), that uses a Single Error Correction Algorithm (AlgSE) followed by a Double Error Correction Algorithm (AlgDE). Both algorithms explore the LPC characteristics to attain greater decoding efficiency. AlgSE is implemented with an iterative technique associated with a correction heuristic, while AlgDE is an innovative proposal that allows increasing the effectiveness of correction through the inference of errors. AlgDE allows to increase the efficiency of the LPC decoder significantly when used together with AlgSE. All performances are supported by numerical experiments

Keywords: Error Correction Code. Fault Tolerance. Radiation Effect. Memory Reliability.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND ACRONYMS

ECC	Error Correction Codes
PCoSA	Product Code for Space Application
OPCoSA	Optimized Product Code for Space Application
LPC	Line Product Code
AlgSE	Single Error Correction Algorithm
AlgDE	Double Error Correction Algorithm
SEE	Single Event Effect
OBC	On-Board Computer
SRAM	Static Random-Access Memory
SEU	Single Event Upset
MCU	Multiple Cell Upset
MBU	Multiple Bit Upset
SEFI	Single Event Functional Interrupt
SET	Single Event Transient
SED	Single Event Disturb
SBU	Single Bit Upset
SOI	Silicon on Insulator
Rad-Hard	Radiation Hardening
COTS	Commercial-Off-The-Shelf
TMR	Triple Modular Redundancy
SLR	Systematic Literature Review
XOR	Exclusive OR
SECDED	Single Error Correction and Double Error Detection
2D	Two-dimensional
IC	Integrated Circuit
RQ	Research Questions
SF	Snowballing Forward
IEEE	Institute of Electrical and Electronics Engineers
ACM	Association for Computing Machinery
ASP	American Scientific Publishers

EDAC	Error Detection and Correction
1D	One-dimensional
IDE	Integrated Development Environment
CMOS	Complementary Metal Oxide Semiconductor
S2E	Straightforward 2D-ECC
FUEC	Flexible Unequal Error Control
PC	Product Code
EPC	Extended Product Code
MC	Mixed Code
EG-LDPC	Euclidian Geometry Low-Density Parity Check
MED	Multi-bit Error Detection
MDMC	Modified Decimal Matrix Code
MMC	Modified Matrix Code
PBD	Parity per Byte and Duplication
MRSC	Matrix Region Section Code
HVD	Horizontal-Vertical-Diagonal
3D	Three-dimensional
HVDD	Horizontal-Vertical-Double-Bit-Diagonal
eMRSC	extended Matrix Region Section Code
HVPDH	Horizontal-Vertical Parity and Diagonal Hamming
PHICC	Parity Hamming Interleaved Correction Code
MTTF	Mean Time To Failure
CCC	Correction Coverage per Cost
DCC	Detection Coverage per Cost
DMC	Decimal Matrix Coding
ERT	Encoder Reuse Technique
SE	Simple Error
DE	Double Error
CLC	Column Line Code
RM	Reed Muller
DCO	Data Correction Only
DRC	Data and Redundancy bits Correction

DCOC	Data Correction Only with Cross-Check
DRCC	Data and Redundancy bits Correction with Cross-Check

NOTATION

b	Burst length
c	Codeword
C	Code length and subspace of GF(q)
Cor	Maximum number of errors the code can correct
cw	Encoded message
$C^{\perp}$	Orthogonal complement of C
$C_{i,j}$	Representation of the bit positioned in row i and column j of the check bit region
CCC_i	Correction coverage per cost
CR_i	Correction rate
CTC_i	Correction total cost
d	Minimum Hamming Distance
D	Data region
DCC_i	Detection coverage per cost
Det	Maximum number of errors the code can detect
d_H	Hamming Distance
DR_i	Detection rate
d_{min}	Minimum Hamming Distance
d_{pc}	Minimum distance of a product code
d_{mpc}	Minimum distance of a modified product code
$D_{i,j}$	Representation of the bit positioned in row i and column j of the data region
dr	Data rate
$\mathbf{e}$	Error vector
e	The total number of errors in a pattern
ec	Error combinations injected by burst error
EC	Maximum number of errors the code can correct
ED	Maximum number of errors the code can detect

G	Generator matrix
g_i	Linearly independent vectors of the Generator matrix
G'	Modified generator matrix
$GF(q)$	Galois Field of characteristic q
H	Parity check matrix
H^T	H Matrix transpose
i	Number of errors
ib	Inner bit flips
I_k	Identity Matrix of dimension k
I_{n-k}	Identity Matrix of dimension $n-k$
k	Number of data bits
ls	Levels of severity
m	Information sent by digital source
M	Number of addresses in memory
$\hat{m}$	Estimated bit information after decoding process
M_i	Metric to assess correction and detection capability with inclusion of redundancy
$MTTF$	Mean time to failure
n	Number of codeword bits
ob	Outer bit flips
P	Matrix that generates the parity symbols
P^T	Transpose of P Matrix
$P_{in}(t)$	Probability of occurring exactly i errors in a given memory word with n bits at time t
$P_n(t)$	Probability of having errors in a memory due to the rate λ over time
P_C	Column parity vectors of D matrix
P_R	Row parity vectors of D matrix
P_{RC}	Parity bit of the P_R and P_C intersection

r	Number of redundancy bits
$\mathbf{r}$	Word received by decoder
$r(t)$	Memory reliability in time t considering ε of a given ECC
$R(t)$	Reliability at time t of all memory
rr	Redundancy rate
ro	Redundancy overhead
$\mathbf{s}$	Syndrome vector
S_i	Storage elements in a convolutional code
t	Time
TCC_i	Total coverage per cost
u	Convolutional code input information
V_i	Outputs of a convolutional code
w	Message to be encoded
$\bar{x}_i$	Vector to determine the Hamming distance
$x_{i,j}$	Vector components to determine the Hamming distance
α	number of columns in the data region of a product code
β	number of columns in the check bit region of a product code
γ	total number of columns of a product code
δ	number of rows in the data region of a product code
ε	number of rows in the check bit region of a product code
θ	total number of rows of a product code
λ	Error rate
σ	Maximum number of errors
$\varepsilon(i)$	Error coverage rate for each of the i errors

CONTENTS

1 INTRODUCTION

According to Marchese; Patrone 2020, new classes of small satellites are catching the attention of the scientific community and have attracted research interest in recent years due to the wide field of applications (Wang *et al.*, 2018). Historically, space missions and satellites were developed by space agencies or large companies at a high cost. However, this context is changing, for example, due to the emergence of a standardized type of nanosatellite, the CubeSat (Marchese *et al.*, 2018; Gonzalez *et al.*, 2019). They were conceived as an educational tool in which students could develop, through hands-on experience, a complete mission, including the design, construction, launch and operation of a satellite. This attracted the attention of private companies and government agencies for the low cost of components and the availability of low cost launches (Babich *et al.*, 2020; Mughal *et al.*, 2020).

Among its applications are, for example, Earth observation, environmental monitoring, educational missions, communication, remote sensing and even military surveillance (Alam; Islam, 2018; Babich *et al.*, 2020). This technology has an even more promising future, because due to its reduced cost and size, several companies are already planning to launch a large number of these devices to form an Internet of Things (IoT) network in space, which is the case of an Illinois laboratory (Wang *et al.*, 2018) and the company Eutelsat (Marchese; Patrone, 2020), in addition to integrating satellite networks and terrestrial infrastructure as part of an overall communication network in the next generation of 5G (Marchese; Patrone, 2020).

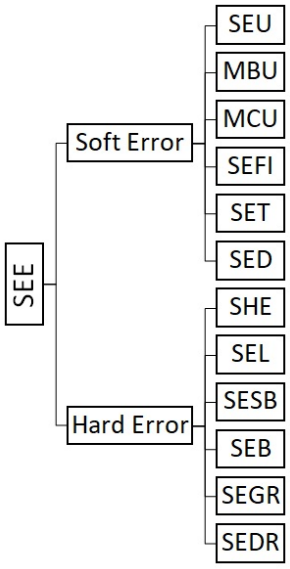
These satellites have several components, including the On-Board Computer (OBC) (Botma *et al.*, 2013; Sánchez-Macián *et al.*, 2017) which is responsible for controlling and processing the entire satellite (PuWeihua, 2017). The OBC is the most important component of the system and acts as the brain of the satellite, as it maintains synchronization between the various peripherals, controls data flow, performs energy management and is also responsible for communicating with other devices (Nagarajan *et al.*, 2014). It is exactly in the OBC that one of the components most susceptible to failure is, the Static Random-Access Memory (SRAM) (Botma *et al.*, 2013; Benevenuti *et al.*, 2019). According to NICOLAIDIS 2011,this occurs because the supply voltage and the capacitance of the nodes of these memories were reduced due to the miniaturization of the components, which results in a lower critical load and the consequent increase of the vulnerability of these memories to failures due to radiation (NICOLAIDIS, 2011).

Failure of these components due to radiation during space applications can cause serious damage. They can spread throughout the system producing even more serious failures.

This concern is even more critical in critical systems where consequences can be disastrous (Silva *et al.*, 2018), such as functional failure, loss of control (Liu *et al.*, 2016), loss of stored data and even loss of human life on certain missions as a result of radiation in the space environment (Li *et al.*, 2019).

According to NICOLAIDIS 2011, radiation-induced soft error is one of the most challenging issues affecting the reliability of today's electronic devices. There have been a lot of efforts in recent decades to measure, model and mitigate the effects of radiation. These errors are an increasing threat to integrated circuits manufactured using advanced technology. These types of errors are divided into soft errors and hard errors. Soft error is an event where the data is corrupted but the device is not permanently damaged. If so, it would be a hard error. Soft errors can have different consequences in different applications, such as a system malfunction or even a complete shutdown of the equipment. According to U.S.Department of Transportation 2016, the classification of these failures, called SEE, can be seen in Figure 1.

Figure 1 – Classification of SEE into two subsets: Soft Errors and Hard Errors



The focus of this thesis is on soft errors and therefore only they will be detailed below. All classification of Hard Errors and additional information can be found in U.S.Department of Transportation 2016 and NICOLAIDIS 2011. Soft errors are divided into:

- Single Event Upset (SEU): the particle changes only one memory cell;
- Multiple Cell Upset (MCU): the particle changes two or more memory cells;
- Multiple Bit Upset (MBU): it is an MCU that occurs in the same word;
- Single Event Functional Interrupt (SEFI): the event causes a loss of functionality, as it

affects control registers, clock and reset signals, for example;

- Single Event Transient (SET): the event causes a voltage failure in the circuit and becomes a wrong bit when captured in the memory element; and
- Single Event Disturb (SED): it is an unstable state of an SRAM memory. This instability can invert the value of the cell and this characterization becomes an SEU.

According to NICOLAIDIS 2011, SEU is used ambiguously as a synonym for Soft Error or Single Bit Upset (SBU). These faults are induced by the interaction of an ionizing particle with electronic components. Ionizing particles can be primary (such as heavy ions in the space environment or alpha particles produced by radioactive isotopes contained in the die or packaging), or secondary created by the nuclear interaction of a particle, such as a neutron or proton with silicon, oxygen or any other die atom. SEE become possible when the collected fraction of the charge released by the ionizing particle is greater than the electrical charge stored in a sensitive node. These effects are considered an important challenge regarding the reliability of electronic systems in space and have motivated many research and development efforts in industry and academia to seek ways of mitigation (NICOLAIDIS, 2011).

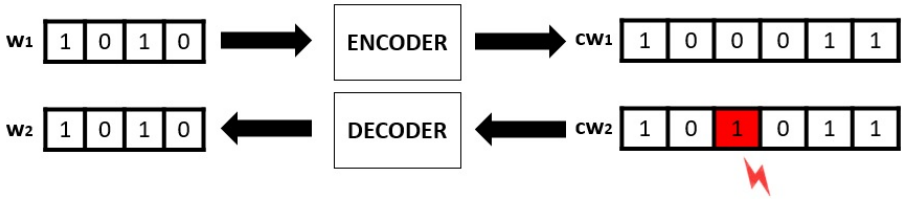
According to WANG 2017 and VARGAS; NICOLAIDIS 1994, when a high-energy particle passes through the PN junction, some of the particle's energy is absorbed by the silicon atom in its path. Furthermore, the particle creates electron-hole pairs in the silicon. These electron-hole pair undergo drift and diffusion movements under the electric field of the PN junction, thus inducing a few tenths of a nanosecond of impulse current after the charge is collected. The transient current can change the node potential and, to a certain extent, can turn the conducting tube off and the blanking tube on, causing the logic state of the device to rollover; such phenomenon is called SEU.

Mitigation of these failures is really important and the first solution to avoid SEU was to use shielding, which reduces the flow of particles to low levels, but does not completely eliminate them (KASTENSMIDT *et al.*, 2006). This solution was used to eliminate errors caused by radiation in the past. However, with the evolution in the manufacturing process of electronic devices and its consequent decrease, devices are becoming increasingly sensitive to particles. Therefore, extra techniques must be used to mitigate these errors. For example, Silicon on Insulator (SOI) structures consist of a top single-crystal silicon layer, either separated from the bulk substrate by an insulating layer or directly supported by an insulating substrate. SOI has been widely developed in microelectronics because of its advantages compared to silicon bulk

substrates. Among other benefits, SOI wafers allow radiation-hardening for sensitive applications (KONONCHUK; NGUYEN, 2014). Another widely used technique is the use of Radiation Hardening (Rad-Hard) components that are completely or partially hardened against radiation (KASTENSMIDT *et al.*, 2006). However, the purchase of these components, depending on the country, is heavily controlled by government agencies that impose complex political and commercial barriers to access to this technology (Villa *et al.*, 2017b). Thus, designers in this area are seeking to use Commercial-Off-The-Shelf (COTS) components, which are conventional components and have the ease of purchase, the reduction of costs and volume in the project and the availability of the most up-to-date technology , as Rad-Hard components typically use older technologies (Villa *et al.*, 2017a; Shim *et al.*, 2019).

However, the use of COTS components in spatial applications is only possible with the use of additional mitigation techniques, such as Hardware Redundancy, Reconfiguration and ECC (KASTENSMIDT *et al.*, 2006; Pouponnot, 2005). The first technique is characterized by extra components or paths that allow the project to continue operating even if a failure occurs. The best known example is the Triple Modular Redundancy (TMR) technique which uses three identical implementations of the same logic function and the outputs are connected to a voter that decides by majority on the correct result (KASTENSMIDT *et al.*, 2006; Pouponnot, 2005; Kumar *et al.*, 2016). The Reconfiguration technique known as Scrubbing is effective and aims to mitigate failures through a periodic process of reading, correcting errors and writing each memory address. It does not interrupt system operation and is able to prevent faults from accumulating in the system (SALEH *et al.*, 1990; Herrera-Alzu; Lopez-Vallejo, 2013; Kumar *et al.*, 2016). Finally, and the objective of this thesis study, ECC are a technique for protecting digital information against data errors. The basic concept, indicated in Figure 2, is to have an encoder to add check bits to the word w_1 and even if the code word cw_1 suffers some bit inversion due to radiation (becoming the code word cw_2), the decoding algorithm is able to restore the initial correct value of the information, making $w_2 = w_1$ (KASTENSMIDT *et al.*, 2006; Pouponnot, 2005; Kumar *et al.*, 2016).

Figure 2 – Description of how an ECC works.



Traditionally, ECCs have been widely used as a very efficient method of protecting information from errors. ECC design is continually evolving, adapting its coverage to new design needs and error conditions (Saiz-Adalid *et al.*, 2019). However, as technology advances, the number of errors becomes an increasingly important issue because more cells are included in the radius affected by a particle (Liu *et al.*, 2017; Liu *et al.*, 2018). Traditional ECC have been used, but as technology increases, more powerful error correction features are needed (Liu *et al.*, 2017; Li *et al.*, 2018; Liu *et al.*, 2018).

Therefore, traditional one-dimensional codes that have k data bits, r redundancy bits and n bits in total would be unable in some cases to mitigate the MBU on their own. Other conventional methods of memory protection, such as the Scrubbing technique in combination with simpler ECC and/or TMR are also not viable, as extending these techniques to cover MBU on a large scale will incur in excessive area increase, latency and power consumption of SRAM memories (Erozan; Cavus, 2015).

A promising solution to mitigate large-width MBUs is to build two-dimensional (2-D) ECC structures, which can provide scalable multi-bit error protection against large soft error clusters. In this configuration, the data bits have dimension $k_1 \times k_2$ and $n_1 \times n_2 - k_1 \times k_2$ bits of redundancy are added, contributing to a cross correction of rows and columns. Compared to conventional schemes with similar error coverage (TMR and Scrubbing technique), 2-D ECC architectures offer significantly lower latency and power consumption (Erozan; Cavus, 2015). The product code concept, introduced initially by Elias 1954, is quite simple as well as powerful, where shorter block codes are used instead of long block code. Basically, they are matrix codes in which the rows are coded by one code, while the columns are coded by another code. This arrangement increases your error-correcting capability, as errors are corrected by both row and column (Atta-ur-Rahman *et al.*, 2012).

Thus, it is still unclear how the structure of a two-dimensional code and its decoding algorithm influence the correction rate and its associated cost. Therefore, this thesis aims to develop new structures and new decoding techniques, always focusing on the maximum correction capacity of this class of ECCs with the lowest possible cost of hardware implementation.

With this information, the main contributions of this thesis are listed below:

– Conducting a Systematic Literature Review to identify which codes are the most used in current 2D-ECCs in critical applications, which are the main decoding methods, which are the most used word sizes, which are the methods of analysis and comparison of these

codes and which are the trends and perspectives of this class of codes;
- Development of new two-dimensional code approaches, taking into account codes with high correction capacity and lower hardware implementation cost; and
- Proposition of new two-dimensional code decoding techniques, always seeking for the highest correction rate with relatively low implementation cost.

This thesis consists of seven chapters, including the introduction. Below, a brief summary of the main points of each chapter is presented.

- **Chapter 2**:Error Correction Codes. The main concepts of one-dimensional codes and two-dimensional codes are explored;
- **Chapter 3**: Methodological Aspects. The methodological aspects used in Systematic Literature Review (SLR) and simulation tests are presented, in addition to the three main methods of analysis;
- **Chapter 4**: State of the Art in 2D ECC. The phases of a systematic literature review and a classification of two-dimensional ECCs are presented. Size, word redundancy, applications, analysis methods, and trends in 2D ECCs are analyzed;
- **Chapter 5**: Proposed ECCs. The three ECCs proposed in this thesis are presented. The structure of each code, the coding equations and the entire decoding process are detailed;
- **Chapter 6**: Results. In this chapter, simulation results for each of the contributions are presented, taking into account correction capability, reliability, hardware cost analysis and redundancy; and
- **Chapter 7**: Conclusions and Perspectives. After presenting and analyzing the results obtained, this chapter highlights the main considerations of the contributions presented and which research would still be interesting to be explored in this area.

1.1 Contributions

The contributions of this thesis are:

- Systematic literature review to investigate the most important features of 2D-ECCs used for memory failure mitigation. This SLR revealed the most used ECCs, data size and redundancy overhead, encoder and decoder implementation technology, fault injection methods, and evaluation metrics. Besides, some trends in ECCs were extracted, such as reusing the encoder inside the decoder and targeting the 3D-ECC to increase the error correction efficacy.

- Development of PCoSA, a two-dimensional ECC based on Hamming and parity of both rows and columns for use in memory during spatial applications with reliability requirements;
- Development of OPCoSA, an ECC that optimizes its original PCoSA version, reducing 16 bits of redundancy and maintaining high correction capacity;
- Development and analysis of the LPC technique, which is a proposal based on a AlgSE followed by a AlgDE. Both algorithms exploit features to improve decoding efficiency. AlgSE is implemented with an iterative technique associated with a heuristic correction, while AlgDE is an innovative proposal that allows to increase the correction efficiency;

In terms of scientific production already published:

- **Journals**
  - D. C. C. Freitas, D. Mota, C. Marcon, J. A. N. Silveira and J. Mota, "**LPC: An Error Correction Code for Mitigating Faults in 3D Memories**," in IEEE Transactions on Computers, vol. 70, no. 11, pp. 2001-2012, Nov. 2021, doi: 10.1109/TC.2020.3034400.
  - D. Freitas, D. Mota, R. Goerl, C. Marcon, F. Vargas, J. Silveira and J. Mota, "**PCoSA: a product error correction code for use in memory devices targeting space applications**", Integration – The VLSI Journal, vol. 74, no. 1, pp. 71-80, Sep. 2020, doi: 10.1016/j.vlsi.2020.04.006.
  - D. C. C. Freitas, C. Marcon, J. A. N. Silveira, L. A. B. Naviner and J. C. M. Mota, "**New Decoding Techniques for Modified Product Code used in Critical Application**," in Microelectronics Reliability, vol. 128, no. 1, Jan. 2022. doi: 10.1016/j.microrel.2021.114444
- **Conferences/Symposiums**
  - D. C. C. Freitas et al., "**Error Coverage, Reliability and Cost Analysis of Fault Tolerance Techniques for 32-bit Memories used on Space Missions**," 2020 21st International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 2020, pp. 250-254, doi: 10.1109/ISQED48828.2020.9137019.
  - D. Freitas, D. Mota, G. Martins, G. Castro, J. Silveira e J. Mota, "**Implementation of Error Correction Code for Fault Mitigation in OBC for CubeSat Nano Satellites**," 2019 Workshop on Circuits and System Design (WCAS 2019), pp. 1-4, São Paulo, SP, Brasil, 2019.

In terms of scientific production submitted and under review:

- **Journals**
  - D. Freitas, J. Silveira, C. Marcon, L. Naviner and J. Mota, "**OPCoSA: An Optimized Product Code for Space Applications**", in Integration – The VLSI Journal, 2021.
  - D. Freitas, C. Marcon, J. Silveira, L. Naviner and J. Mota, "**A Systematic Literature Review of Two-Dimensional Error Correction Code**", in Proceedings of the IEEE, 2021.
  - D. Freitas, J. Silveira, C. Marcon, L. Naviner and J. Mota, "**Checkbit Regions Inclusion Analysis in two-Dimensional Error Correction Codes**," in Microelectronics Reliability, 2021.
  - D. Freitas, C. Marcon, J. Silveira, L. Naviner and J. Mota, "**nMatrix: A new decoding algorithm for the Matrix ECC**," in Microelectronics Reliability, 2021.

2 ERROR CORRECTION CODES

2.1 Introduction

This chapter deals with the presentation of ECC that will be used as base codes or structures for the codes developed in this thesis. The basic concepts, linear block code, Hamming code and n-dimensional codes are presented.

First, the general basic concepts regarding error-correcting codes are presented, such as the representation of a typical transmission/storage system in its simplified form, in addition to the presentation of the two classes of codes: (i) block code and (ii) convolutional code.

Next, the linear block code is detailed and is defined the Hamming distance and minimum distance. The generator matrix G and the parity check matrix H which are used, respectively, in the encoding and decoding process are detailed. The version of these two matrices are arranged in a systematic way, when the data bits and the redundancy bits are physically separated in the codeword. Finally, the way to detect and correct errors in a given message is presented through the syndromes.

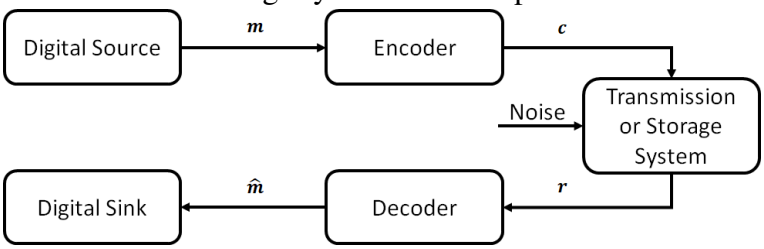
The most used linear block code, i.e. the Hamming code, is detailed and its relation of data bits, codeword length and number of redundancy bits are presented. The correction and detection capability of a generic Hamming code is determined as a function of the minimum distance. The values obtained indicate that it is a code that performs simple error correction or double error detection. The generator matrix, the parity check matrix and a syndrome table of a $Ham(7,4)$ code are also presented. Finally, Hamming code extended is used to increase the minimum distance of the Hamming code, contributing to have a code capable of correcting an error and detecting two errors simultaneously.

In the last part of the chapter, n-dimensional codes are detailed, focusing on two-dimensional ECCs. These structures were proposed in 1954 with the intention of increasing correctability by using simpler codes.

2.2 Basic concepts

The transmission and storage of digital information has many things in common. Both transfer data from an information source to a destination (LIN; COSTELLO, 1983; MOON, 2005). A typical transmission or storage system is represented in its simplified form in Figure 3.

Figure 3 – Typical transmission/storage system in its simplified form.



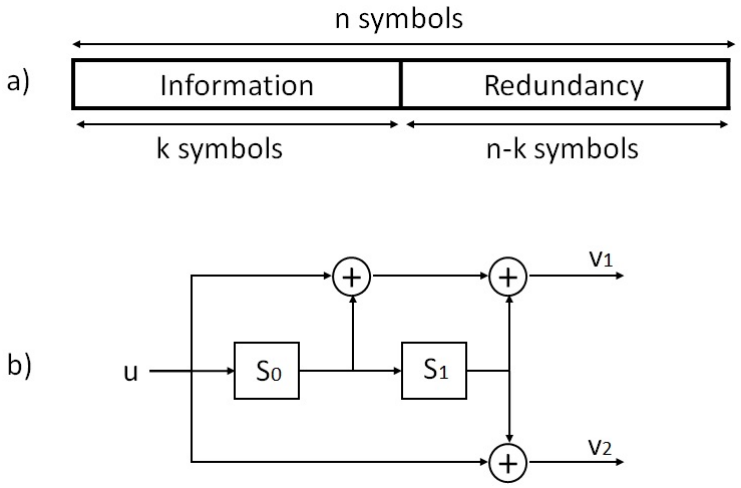
For this thesis, a storage system is used, as the focus of the application is SRAM memories. Thus, the digital source sends the information sequence **m** and the Encoder block transforms this information into a codeword named **c**. In this case, **c** is also a binary sequence that is written to memory that may have its information changed due to environmental disturbance. The process of reading the received information **r** from memory goes through the Decoder that transforms the sequence **r** into the estimated sequence $\hat{\mathbf{m}}$. Ideally, $\hat{\mathbf{m}}$ should be the replica of the information string **m**, although the noise can cause some decoding errors (LIN; COSTELLO, 1983). As stated, the focus of this thesis focuses on design and implementation proposals for the Encoder and Decoder blocks of two-dimensional ECCs.

All ECCs are based on the same basic principle: redundancy. It is added to the information to correct any errors that might occur in the transmission or storage process. In a basic form, redundant symbols are appended to information symbols to get a codeword (ZARAGOZA, 2006).

According to how redundancy is added to words, ECC can be divided into two classes: block and convolutional. Block codes process information block by block, treating each bit of information independently of the others. In other words, block encoding is a memoryless operation, in the sense that codeword are independent of each other. In contrast, the output of a convolutional encoder depends not only on current input information, but also on previous inputs or outputs, block by block or bit by bit (ZARAGOZA, 2006; LIN; COSTELLO, 1983). Figure 4 presents both classes.

A block code is represented by Figure 4 a) and is organized in such a way that for each set of k symbols are added to the word $n-k$ symbols called redundancy bits, forming a word with n symbols.In a parallel process, encoding is done all at once, different from the convolutional code, shown in Figure 4 b), which has a serial process. In this case, the convolutional code consists of a set of binary sequences and the output depends not only on the input symbols, but also on the previous inputs and/or outputs. In Figure 4 b), u is the input information, S_i are storage elements, V_i are the outputs, and $+$ are logical Exclusive OR (XOR) operations. For this

Figure 4 – Two classes of code: a) block and b) convolutional.



specific case, for example, if $u = 0$ and $[S_0S_1] = [10]$, the outputs become $[V_1V_2] = [10]$ and the memory elements are updated to $[S_0S_1] = [01]$.

On the other hand, a block code encoder divides the information sequence into k bit message blocks. A message block is represented by the binary k-tuple $\mathbf{m} = (m_1, m_2, ..., m_k)$ called message. There are a total of 2^k possible different messages. The encoder transforms each $\mathbf{m}$ message into an n-tuple $\mathbf{c} = (c_1, c_2, ..., c_k)$ of symbols called codeword. So for every 2^k messages, there are 2^k codeword. This set of 2^k codeword of length n is called block code (n,k) (LIN; COSTELLO, 1983).

In this thesis, all code proposals are based on linear block codes. A code of length n and with 2^k codeword is called a linear block code (n,k) if and only if the modulo 2 sum of any codeword is also a codeword (LIN; COSTELLO, 1983; MOON, 2005). The following section presents some important definitions and the encoding and decoding processes of a linear block code.

2.3 Linear Block Code

Linear block codes are defined and described in terms of the generator and parity check matrices and are represented by three parameters: block length n, data size k and minimum distance d_{min}. Thus, a code C is represented as (n,k,d_{min}) (MOON, 2005), where d_{min} is the minimum Hamming distance and is described by the following two equations.

Consider two vectors $\bar{x}_1 = (x_{1,0}, x_{1,1}, ..., x_{1,n-1})$ and $\bar{x}_2 = (x_{2,0}, x_{2,1}, ..., x_{2,n-1})$. So the Hamming distance between $\bar{x}_1$ and $\bar{x}_2$, denoted by $d_H(\bar{x}_1, \bar{x}_2)$, is defined as the number of

elements where the vectors differ,

$$d_H(\bar{x}_1,\bar{x}_2) = \mid \{i : x_{1,i} \neq x_{2,i}, 0 \leq i \leq n-1\} \mid = \sum_{i=0}^{n-1} x_{1,i} \oplus x_{2,i}, \tag{2.1}$$

where $\mid A \mid$ denotes the number of elements in a set A and $\oplus$ denotes sum modulo 2 (XOR) (ZARAGOZA, 2006).

Given a code C its minimum Hamming distance d_{min} is defined as the minimum Hamming distance between all possible distinct pairs of codeword in C (ZARAGOZA, 2006),

$$d_{min} = \min_{\bar{v}_1,\bar{v}_2,\in C} \{d_H(\bar{v}_1,\bar{v}_2) \mid \bar{v}_1 \neq \bar{v}_2\}. \tag{2.2}$$

In this thesis, the representations (n,k,d_{min}) or (n,k) are used equally to denote the parameters of a block code of length n, which encodes messages of length k and has a minimum Hamming distance d_{min} with code length $|C| = 2^k$.

2.3.1 Generator Matrix

According to MOON 2005, a linear block code C is a vector space of dimension k, in which there are k linearly independent vectors designated as $\boldsymbol{g_0}, \boldsymbol{g_1}, ..., \boldsymbol{g_{k-1}}$ so that each codeword $\boldsymbol{c}$ in C can be represented as a linear combination of these vectors,

$$\boldsymbol{c} = m_0\boldsymbol{g_0} + m_1\boldsymbol{g_1} + ... + m_{k-1}\boldsymbol{g_{k-1}}. \tag{2.3}$$

Defining $\boldsymbol{g_i}$ as row-vectors, G matrix of dimension k x n can be defined,

$$G = \begin{bmatrix} \boldsymbol{g_0} \\ \boldsymbol{g_1} \\ \vdots \\ \boldsymbol{g_{k-1}} \end{bmatrix} \tag{2.4}$$

Making

$$\boldsymbol{m} = \begin{bmatrix} m_0 \, m_1 \, \cdots \, m_{k-1} \end{bmatrix} \tag{2.5}$$

then the equation 2.3 can be written as

$$\boldsymbol{c} = \boldsymbol{m}G, \tag{2.6}$$

and each codeword $\boldsymbol{c} \in C$ has a representation for each vector $\boldsymbol{m}$. Since rows of G generate the linear code (n,k), G is called the generator matrix for C and the equation 2.6 represents the encoding operation for the code C.

Note that the representation of a code given by G is not unique. For a given G, another generator G' can be obtained by performing operations on rows. So, the encoding operation defined by $\boldsymbol{c} = \boldsymbol{m}G'$ maps the message $\boldsymbol{m}$ to a codeword in C, but it is not necessarily the same codeword which would be obtained using the G generator. Thus, the codeword is different from $\boldsymbol{c}$, but it is still a codeword in C.

Thus, we can define a systematic encoder in which the message bits $m_0, m_1, ..., m_{k-1}$ can be found explicitly in the codeword. For a linear block code, the encoding operation represented by G is systematic if an identity matrix can be identified among the rows of G.

Often, a systematic encoder is written in the form

$$G = \left[I_k \; P \right] = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ 0 & 1 & 0 & \dots & 0 & p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} \\ 0 & 0 & 1 & \dots & 0 & p_{2,0} & p_{2,1} & \cdots & p_{2,n-k-1} \\ \vdots & \vdots & & \vdots & & & & \vdots & \\ 0 & 0 & 0 & \dots & 1 & p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix}, \tag{2.7}$$

where I_k is the identity matrix k x k and P is the matrix k x $(n-k)$ that generates the parity symbols. The encoding operation is

$$\boldsymbol{c} = \boldsymbol{m} \left[I_k \quad P \right] = \left[\boldsymbol{m} \quad \boldsymbol{m}P \right]. \tag{2.8}$$

The codeword is divided into two parts: the $\boldsymbol{m}$ part consists of the message symbols and the $\boldsymbol{m}P$ part consists of the parity check symbols. It is exactly the systematic encoder that will be used in all proposals for this thesis.

2.3.2 Parity Check Matrix

According to BLAHUT 2003, since C is a subspace of $GF(q)$, it has dimension k. This is equal to the number of rows in G. Since C is a subspace, it has an orthogonal complement $C^{\perp}$, which is the set of all vectors orthogonal to C. $C^{\perp}$ has dimension $n-k$. Let H be formed by lines of any set of base vectors of $C^{\perp}$, then an n-tuple $\boldsymbol{c}$ is a codeword in C if and only if it is orthogonal to every row vector of H. Then,

$$\boldsymbol{c}H^T = 0. \tag{2.9}$$

This equation allows you to check whether a word is a C codeword. The H matrix is called the parity check matrix of the code C. It is a matrix of dimension $(n-k)\ x\ n$. The equation 2.9 is a necessary condition for $\boldsymbol{c}$ to be a codeword from the matrix G. Since any relation $\boldsymbol{c}H^T = 0$ is satisfied when $\boldsymbol{c}$ is orthogonal to any row in G (MACWILLIAMS; SLOANE, 1977; ZARAGOZA, 2006), then

$$GH^T = 0. \tag{2.10}$$

As in the equation 2.7, the matrix H can be systematized, as shown in the equation 2.11.

$$H = \left[P^T\ I_{n-k}\right] = \begin{bmatrix} p_{0,0} & p_{1,0} & \cdots & p_{k-1,0} & 1 & 0 & 0 & \dots & 0 \\ p_{0,1} & p_{1,1} & \cdots & p_{k-1,1} & 0 & 1 & 0 & \dots & 0 \\ p_{0,2} & p_{1,2} & \cdots & p_{k-1,0} & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & & & & \vdots & \\ p_{0,n-k-1} & p_{1,n-k-1} & \cdots & p_{k-1,n-k-1} & 0 & 0 & 0 & \dots & 1 \end{bmatrix}, \tag{2.11}$$

where I_{n-k} is the identity matrix of dimension $n-k\ x\ n-k$ and P^T is the transpose of the matrix P presented in the equation 2.7.

2.3.3 Error Detection and Correction

According to LIN; COSTELLO 1983, let $\boldsymbol{c} = (c_0, c_1, ..., c_{n-1})$ be a codeword that has been transmitted and $\boldsymbol{r} = (r_0, r_1, ..., r_{n-1})$ the vector received by the decoder, the error vector is set to

$$\boldsymbol{e} = \boldsymbol{r} + \boldsymbol{c} = (e_0, e_1, ..., e_{n-1}) \tag{2.12}$$

being an n-tuple where $e_i = 1$ for $r_i \neq c_i$ and $e_i = 0$ for $r_i = c_i$. This n-tuple is called an error vector (or error pattern). It follows from the equation 2.12 that the received vector $\boldsymbol{r}$ is the sum of the codeword and the error vector, i.e.,

$$\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}. \tag{2.13}$$

When $\boldsymbol{r}$ is received, the decoder calculates the following $(n-k)$-tuple:

$$\boldsymbol{s} = \boldsymbol{r}H^T = (s_0, s_1, ..., s_{n-k-1}). \tag{2.14}$$

which is called syndrome $\boldsymbol{r}$. Thus, $\boldsymbol{s} = 0$ if and only if $\boldsymbol{r}$ is a codeword, and $\boldsymbol{s} \neq 0$ if and only if $\boldsymbol{r}$ it is not a code word. Therefore, when $\boldsymbol{s} \neq 0$, we know that $\boldsymbol{r}$ is not a codeword and the

presence of errors has been detected. When $\boldsymbol{s} = 0$, $\boldsymbol{r}$ is a codeword and the receiver receives $\boldsymbol{r}$ as a codeword. It is possible that certain error vectors are not detected (i.e., $\boldsymbol{r}$ contains errors, but $\boldsymbol{s} = 0$). This can occur when the error pattern $\boldsymbol{e}$ is identical to a non-zero codeword. Error patterns of this type are called undetectable error patterns.

The computed $\boldsymbol{s}$ syndrome of the received $\boldsymbol{r}$ vector actually depends only on the $\boldsymbol{e}$ error pattern and not on the transmitted word $\boldsymbol{v}$. Since $\boldsymbol{r}$ is the sum of the vectors $\boldsymbol{c}$ and $\boldsymbol{e}$ (LIN; COSTELLO, 1983), we have:

$$\boldsymbol{s} = \boldsymbol{r}H^T = (\boldsymbol{c}+\boldsymbol{e})H^T = \boldsymbol{c}H^T + \boldsymbol{e}H^T. \tag{2.15}$$

However, $\boldsymbol{c}H^T = 0$ and, consequently, the following relationship between the syndrome and the error pattern is obtained:

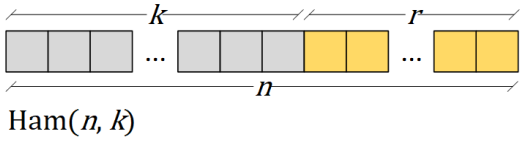
$$\boldsymbol{s} = \boldsymbol{e}H^T. \tag{2.16}$$

The bits contained in the vector $\boldsymbol{s}$ provide information about bit flips and therefore can be used for error correction. The following section deals with the most common block code, the Hamming code.

2.4 Hamming Code

Figure 5 generalizes the code proposed by R. Hamming (HAMMING, 1950), which is referenced by $Ham(n,k)$. The equations 2.17, 2.18 and 2.19 describe the relationships between n, r and k which are, respectively, the number of bits of the codeword, redundancy and data.

Figure 5 – Representation of the generic Hamming code $Ham(n,k)$.



$$n = r + k \tag{2.17}$$

$$r = \log_2(n+1) \tag{2.18}$$

$$k = 2^r - r - 1 \tag{2.19}$$

Hamming code is a code capable of detecting and correcting one-bit errors, since $d_{min} = 3$. This metric is used to measure the detection and correction rate of a Hamming code. The equations 2.20 and 2.21 calculate the maximum number of errors in any position of a codeword that a Hamming code can correct (Cor) or detect (Det), respectively (MACWILLIAMS; SLOANE, 1977).

$$Cor = (d_{min} - 1)/2 \tag{2.20}$$

$$Det = d_{min} - 1 \tag{2.21}$$

The equations 2.20 and 2.21 are exclusive, i.e., Cor or Det, but not Cor and Det simultaneously. The simultaneity relation between Cor, Det and d_{min} is given by the equation 2.22 (MOON, 2005).

$$Det = d_{min} - Cor - 1 \tag{2.22}$$

A well-known Hamming code, which is used a lot in this thesis, is $Ham(7,4)$. A possible matrix G and H, based on the equations 2.7 and 2.11, are shown, respectively, in 2.23 and 2.24. For more information about matrices G and H, readers can find in 2005.

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.23}$$

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

Table 1 – Syndromes of a $Ham(7,4)$ code

Syndrome	Error Position
0 0 0	0 0 0 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 1 1	1 0 0 0 0 0 0
1 0 0	0 0 0 0 1 0 0
1 0 1	0 1 0 0 0 0 0
1 1 0	0 0 1 0 0 0 0
1 1 1	0 0 0 1 0 0 0

The respective syndromes for these matrices G and H are presented in Table 1.

As an example, coding the message $\boldsymbol{m} = [1\ 0\ 1\ 1]$ using the equation 2.6 and the matrix G presented in the equation 2.23, the codeword $\boldsymbol{c} = [1\ 0\ 1\ 1\ 0\ 1\ 0]$ is obtained. The decoding of the codeword $\boldsymbol{c}$ is based on the equation 2.15 and in the matrix H (equation 2.24), thus obtaining the syndrome $\boldsymbol{s} = [0\ 0\ 0]$, indicating that there was no error in the message or that there is an undetectable error pattern. If an error occurs in the second position of the word received by the decoder, for example, making $\boldsymbol{r} = [1\ 1\ 1\ 1\ 0\ 1\ 0]$, the syndrome obtained becomes $\boldsymbol{s} = [1\ 0\ 1]$. By table 1, if this syndrome occurs, the second bit of the received word $\boldsymbol{r}$ must be corrected and, therefore, after correction, the corrected word becomes $\boldsymbol{r} = [1\ 0\ 1\ 1\ 0\ 1\ 0]$ and first four bits represent the word after the decoding process $\hat{\mathbf{m}} = [1\ 0\ 1\ 1]$, because both G and H are systematic.

Another way of presenting the $Ham(7,4)$ encoder and decoder is presented, respectively, in Figures 6 and 7. For hardware implementation of the encoder and decoder of this ECC, this is the way to be done.

Figure 6 – Representation of the hardware implementation of the $Ham(7,4)$ encoder.

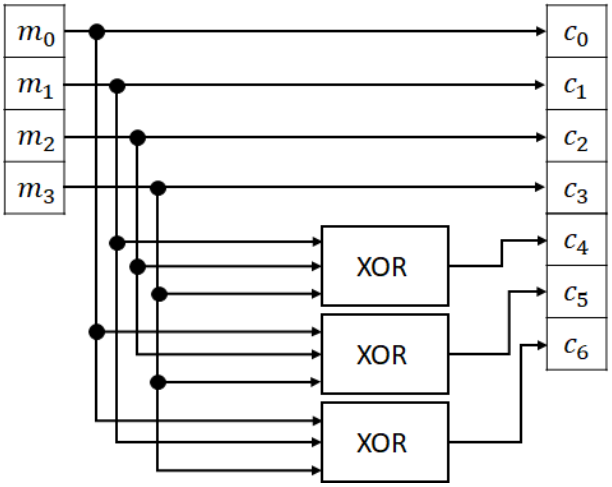
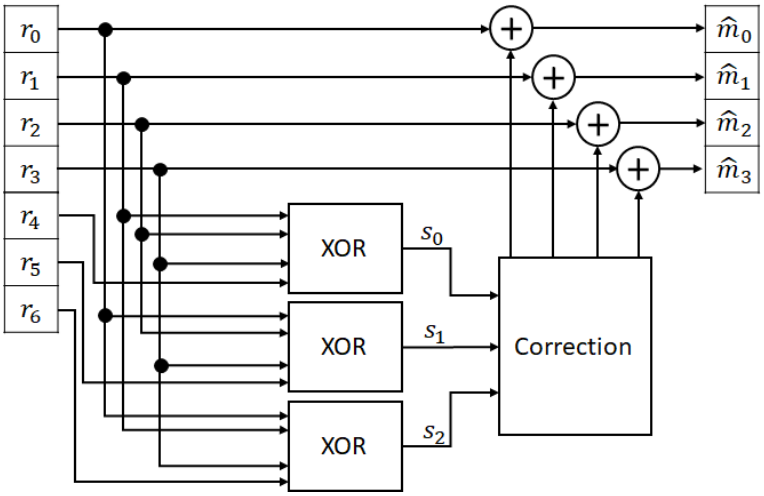


Figure 6 shows the hardware implementation of the ECC $Ham(7,4)$ encoder. The

Figure 7 – Representation of the hardware implementation of the $Ham(7,4)$ decoder.



$\boldsymbol{m}$ message is encoded systematically. The first four bits of the codeword $\boldsymbol{c}$ is the information contained in the word $\boldsymbol{m}$ and the other three digits are the checkbits calculated through three-bit logical XOR operations, following the sequence presented in the equation 2.23. Figure 7 represents the decoding process. Thus, the received word $\boldsymbol{r}$ contains the seven bits with possible errors. The decoding process performs the same idea of the logical XOR operation between the bits like in the encoding process, but with an additional bit, because in addition to the three recalculated checkbits, the respective received checkbit (each of the last three bits of $\boldsymbol{r}$) must be added in the operation. These three XOR operations have an output bit that corresponds to each of the bits of the $\boldsymbol{s}$ syndrome. After calculating the syndrome, the next step is to identify the position with error and change the bit of that position so that the decoded message $\hat{\mathbf{m}}$ is equal to the sent message $\boldsymbol{m}$.

The basic Hamming code $Ham(n,k)$ has $d_{min} = 3$, so it can correct a simple error (i.e. $Cor = 1$) or detect an error caused by a double bitflip without know whether this error has a single error source or a double error, as seen in the equations 2.20 and 2.21. Extended Hamming $Ham(n+1,k)$ adds a parity bit to the basic Hamming code, increasing d_{min} to 4. The equations 2.20 and 2.21 show that Extended Hamming can correct a single error and detect a double error simultaneously, that is, a Single Error Correction and Double Error Detection (SECDED) code (MOON, 2005). Extended Hamming, as well as conventional Hamming, will be used extensively during this thesis.

2.5 n-Dimensional Codes

As stated before, the advancement of circuit integrated technology has made MBUs larger and larger, making traditional one-dimensional codes incapable of mitigating large MBUs. A promising solution to mitigate large MBUs is to build Two-dimensional (2D) ECC structures, which can provide scalable multi-bit error protection against large clusters of soft errors (Erozan; Cavus, 2015).

Elias 1954 was the pioneer in treating an ECC in a two-dimensional format, i.e. a product code, as a simple way to build long codes based on smaller codes (Elias, 1954). Figures 8 and 9 show a product code in its normal version and in its modified version, respectively.

Figure 8 – Representation of a product code in its full version, with the data region, row and column checkbits and checkbits of checkbits.

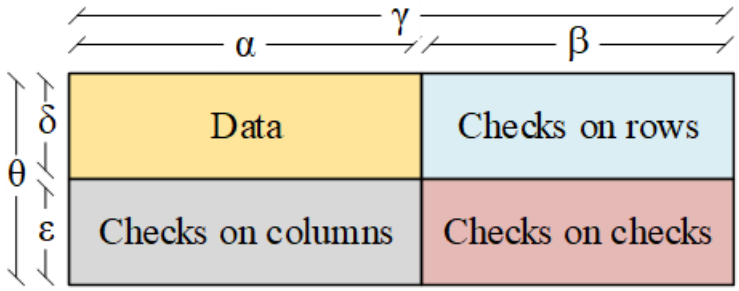
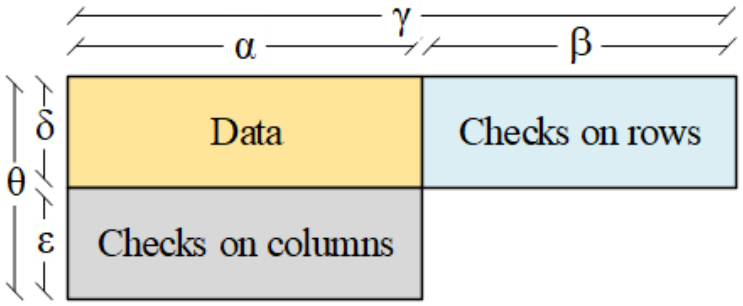


Figure 9 – Representation of a product code in its modified version, with the data and checkbits region of the rows and columns. This type of ECC does not have the checkbit region of the checkbits.



Let α and β be the number of columns, respectively, in the data and redundancy areas, and let δ and ε be the number of rows in these same areas, such that $\gamma = \alpha + \beta$ and $\theta = \delta + \varepsilon$. Then, each row of a product code is coded using the code $C_1(\gamma, \alpha, d_1)$ and each column is coded using the code $C_2(\theta, \delta, d_2)$, forming the code $C_1 \times C_2$. Therefore, each change to a bit in the data region affects the corresponding row and column of that bit.

Figure 8 illustrates the basic structure of a product code. This ECC adds a region that contains checkbits from the checkbits, increasing the Hamming distance and, consequently, the correction potential of the code (MOON, 2005). The minimum distance of a product code d_{pc} is calculated by multiplying the distances of each of the two ECCs, i.e.,

$$d_{pc} = d_1 \times d_2. \tag{2.25}$$

A product code increases the ability to detect and correct errors, but it also increases the costs associated with redundancy, implying more area and energy consumption. In this way, a modified product code reduces redundancy costs, as it does not have checkbits checking (MACWILLIAMS; SLOANE, 1977; ZARAGOZA, 2006). Figure 9 shows the structure of a modified product code and its minimum distance is given by

$$d_{mpc} = d_1 + d_2 - 1. \tag{2.26}$$

2.6 Summary

In this chapter, the main concepts for ECCs were presented. Among the types of codes presented, the block code was more detailed, as it is used for storage in memory systems, due to its encoding and decoding structure in parallel blocks.

The most used block code, due to its simplicity, is Hamming code. The matrices G and H were presented. These matrices are systematically organized as it helps to identify data bits and redundancy bits.

Another important point to highlight is that the $Ham(7,4)$ code is an ECC capable of detecting and correcting only one error. However, adding just one redundancy bit, making $Ham(8,4)$, increases the minimum distance and makes it capable of correcting one error and detecting two errors. This feature makes it a SECDED code and is called extended Hamming code, being widely used in n-dimensional codes because of this feature.

It is worth noting that the decrease in Integrated Circuit (IC) technology and the consequent increase in the number of MBUs makes certain codes in only one dimension not capable of performing an acceptable error correction. Thus, the use of 2D codes is preferable in these cases, as it increases the minimum distance and, consequently, the ECC correction rate.

The cost associated with a 2D code increases as its redundancy increases as well. Thus, modified product codes with a reduction of one of the checkbit regions can be a good design choice, as it increases the minimum distance and code correction rate compared to one-dimensional codes and has a lower cost than a conventional 2D code.

The next chapter presents the methodological aspects used in this thesis.

3 METHODOLOGICAL ASPECTS

3.1 Introduction

This chapter deals with the presentation of the methodological aspects used in this thesis. The steps of the SLR, a generic test methodology and the most used analysis methods are presented.

Five phases of the SLR are presented: Research Objective, Research Questions, Selection Criteria, Search Process, and Data Extraction. Also, all inclusion and exclusion criteria are exposed to obtain the final number of papers that are carefully evaluated, as well as which data are important to highlight in each of the papers.

A generic test methodology to obtain the correction results of this thesis is presented, describing the steps that were validated with scripts in MatLab software. The encoding, error insertion and mapping and decoding process that is done to obtain the error correction results are described.

Finally, the analysis methods used are highlighted: Error Correction Rate, Reliability, and Cost and Redundancy Analysis. These methods for measuring the potential of ECC were chosen because they are the most used in the SLR detailed in chapter 4.

3.2 Systematic Literature Review

The SLR is divided into five phases: Research Objective, Research Questions, Selection Criteria, Search Process, and Data Extraction; and each one is presented below.

3.2.1 Research Objective

This SLR investigates 2D-ECCs to mitigate multiple errors in memory systems developed between 2015 and 2020. This section displays the main aspects of the investigated works, such as the ECC organization, target application, and error analysis methods. These aspects are linked to the research questions discussed in the following.

3.2.2 Research Questions

This SLR development contemplates the main research objective together with the following Research Questions (RQ).

RQ1 – What are the most used 2D-ECCs?

Besides identifying the most used ECCs, this RQ seeks to explain the reason for using 2D-ECCs and allows classifying them and their targeting applications.

RQ2 – How have the ECC techniques been verified/tested?

The answer to this question gives the researchers an overview of how the techniques have been validated, for example, concerning memory size (this is directly linked to the redundancy bits) and whether there is any additional technique beyond the ECC.

RQ3 – What works employed to compare to other ECCs?

This RQ allows researchers to define methods used to compare ECCs and choose the best one for each given scenario.

3.2.3 Selection Criteria

Table 2 describes the selection criteria used in the search process to define the main requirements of the selected papers, avoiding the need for complete work analysis. Note that the inclusion criterion is the topic to meet the research objective - 2D-ECC papers to mitigate memory errors. The exclusion criteria specify papers published before 2015, not available in English, duplicated, and do not meet the inclusion criterion.

Table 2 – Selection Criteria

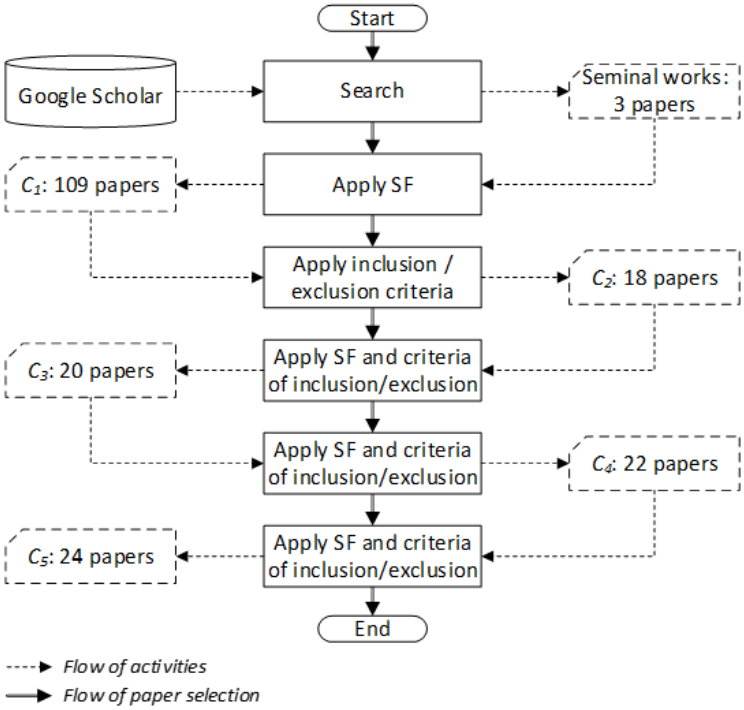
Inclusion criterium	
IC-1	Paper presenting 2D-ECC to mitigate memory errors
Exclusion criterium	
EC-1	Published before 2015
EC-2	Not available in English
EC-3	Duplicated work
EC-4	Paper does not meet the IC-1 criterium

3.2.4 Search Process

Figure 10 shows the search process that was established based on the Snowballing Forward (SF) technique. SF comprises searching for new studies that cite the studies contained in the initial set, followed by an automatic search in five bibliographic databases (MOURAO *et al.*, 2017; FELIZARDO *et al.*, 2016).

SF selects new papers based on the ones that cite the paper being examined (MOURAO

Figure 10 – Papers selection process using the SF technique along with inclusion and exclusion criteria. These steps selected 24 papers.



et al., 2017; FELIZARDO *et al.*, 2016; WOHLIN, 2016; WOHLIN, 2014). One possibility for an initial set is to identify seminal papers with various citations in a systematic review. Three papers from the same group that proposed the Matrix code were found; together, these three papers have 109 citations, respectively, 22, 24, and 63 (ARGYRIDES *et al.*, 2007; ARGYRIDES *et al.*, 2010; ARGYRIDES *et al.*, 2011); therefore, the set C_1 encompasses the 109 works that cite Matrix. The search for these papers was carried out on Google Scholar, using key terms and the number of citations.

Title, summary, year, and parts of the text of all papers of set C_1, applying inclusion and exclusion criteria to build the set C_2 were examined. The next step checks all the papers cited in C_2, followed by the same type of analysis performed in C_1 for creating the set C_3, and so on. The search process in set C_5 was completed, containing 24 papers related to the research objective since the SF process did not return new papers.

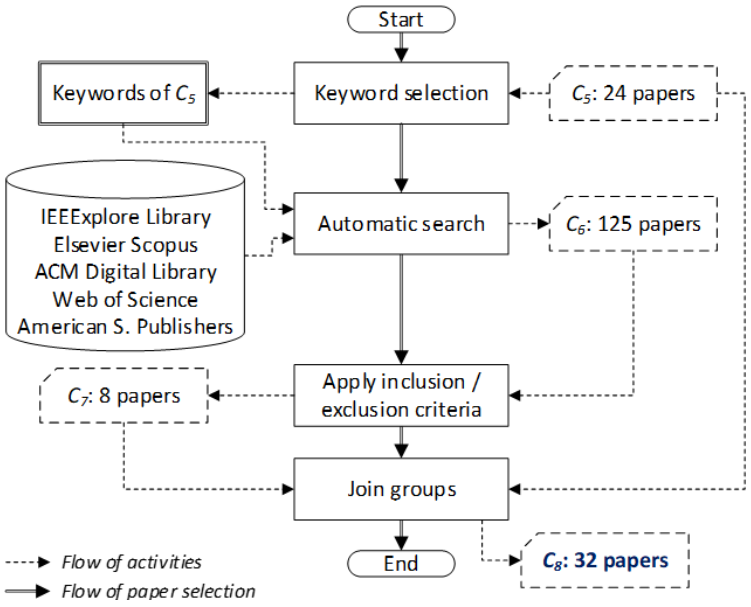
This step starts creating the string used for automatic search in Institute of Electrical and Electronics Engineers (IEEE) Xplore Digital Library, Elsevier Scopus, Association for Computing Machinery (ACM) Digital Library, American Scientific Publishers (ASP), and Web of Science databases encompassing reputed journals and conferences in science and technology. Some papers are included in more than one database, but the choice of the five bases contributes to guarantees diversity. This research was carried out between October 2020 and March 2021,

restricting to the exclusion criteria of Table 2.

The search string created based on the keywords most used in the 24 papers of C_5 is: (“error correction” OR “error detection” OR “ECC” OR “EDAC”) AND (“MCU” OR “MBU” OR “upset” OR “multiple-bit upset” OR “multiple cell upset “) AND (“matrix” OR “product code”) AND (“memory” OR “space” OR “critical”). It is important to highlight that only “EDAC” was included in the most used keywords in C_5, as some papers used this acronym to identify Error Detection and Correction (EDAC).

Figure 11 shows the automatic search on the databases, which resulted in the set C_6 containing 125 papers. The new set C_7 was created, with eight additional papers, deleting the repeated papers and applying the exclusion criteria. Finally, sets C_5 and C_7 were joined to form the final set C_8 comprising 32 papers. Besides, a manual search to avoid losing any paper in the different databases was performed; this manual search did not include any additional work.

Figure 11 – Papers selection process using the automatic search in science and technology databases



This SLR concludes embracing 32 papers, which is similar to other systematic reviews, as shown the works (ALEXANDRE *et al.*, 2020; AL-SAREM *et al.*, 2019; BAJAJ; SANGWAN, 2019; BRITO *et al.*, 2020) that analyze 49, 25, 20, and 56 papers, respectively. The SLR representativeness is worth noting, as the papers selected cover several ECCs, data and redundancy sizes, and error injection and evaluation methods.

3.2.5 Data Extraction

After the selection process, a data extraction method was applied to each work in C_8, aiming to answer the research questions employed to guide the proposed organization. Table 3 displays that the data extraction method includes: Metadata, ECC structure and organization, target application and manufacture technology, methods and metrics employed in ECC evaluation, and future works in ECC.

Table 3 – Data Extraction

Extracted Data	Description
1 - Metadata	
1.1 Metadata	Title, authors, publication year and number of citations
2 - ECC structure and organization	
2.1 Data Size	Identifies the data size and code organization
2.2 Redundancy	Evaluates redundancy overhead
2.3 ECC type	Allows classifying the 2D-ECC used in the work
3 - Target application and technology	
3.1 Target application	Identifies the features of the target application
3.2 Target technology	Identifies technology used for synthesis
4 - ECC evaluation method	
4.1 Fault injection	Examines the applied fault injection methods
4.2 Error coverage	Analysis of detected and corrected errors
4.3 Evaluation metric	Evaluates performance metrics
5 - ECC trends	
5.1 Trends	Examines ECC tendencies and future works

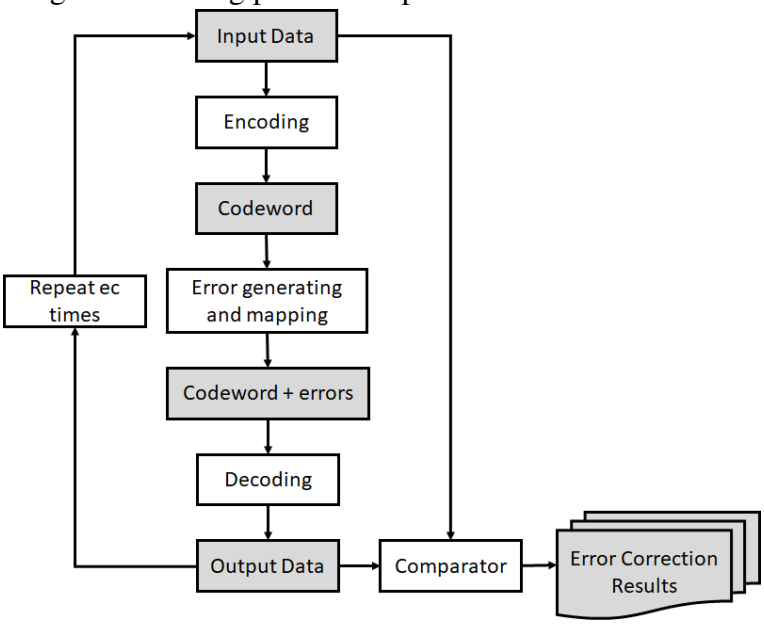
Chapter 4 details all the results obtained with the SLR.

3.3 Test Methodology

Another important methodological aspect is the organization of simulation tests. Figure 12 describes a generic methodology applied in this thesis to evaluate the correction capacity of the proposed ECCs. Each ECC has its own particularity, but generally follows the flow below.

In Figure 12, the message to be recorded in memory is sent to the Encoding block to perform the encoding through check bits and form the codeword. With the word already encoded, errors are generated and inserted. These errors can be of several types: burst, adjacent, and exhaustive, for example. Then, the codedword with the addition of errors is sent to the decoder,

Figure 12 – Encoding and decoding process for presentation of correction results.



which analyzes the check bits sent, recalculates the check bits, calculates the syndromes and corrects the errors. After correcting the errors, the output data must be the same as the message initially recorded. This equality check is done in the Comparator block to get the error correction results. This process is done *ec* times, where *ec* depends on the code length and the type of error to be inserted as shown in subsection 3.4.1. All proposed ECCs were validated with MatLab scripts that insert error pattern types in all regions of the memory array.

3.4 Analysis Method

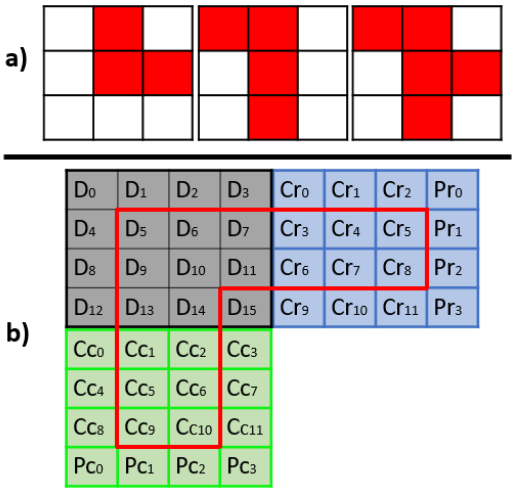
The analysis methods to measure the potential of the ECCs used in the proposals of this thesis were: error correction capacity, reliability and cost and redundancy analysis. These methods were chosen because they are the most used for analysis in current 2D-ECCs, as verified in SLR. Below, each method is presented and detailed.

3.4.1 Error correction rate and Error patterns

A measure of code quality is the ability to correct errors. Two of the 2D-ECC proposed in this thesis are designed through a set of 36 adjacent error patterns and are also tested with exhaustive errors. The other is only tested with exhaustive errors. Evaluation using burst errors is also done. Some ECCs used for comparison use an adjacent error model. All these error patterns are presented below.

The adjacent error model considers a region composed of a central incidence cell and all cells around it, accounting for nine possible error cells. The central position of the codeword error incidence is chosen randomly, and then which $x-1$ bit flips around this central error is randomly selected. There is a limitation in the code incidence area because as the error pattern region is 3×3, all edges cannot be chosen as the central point. Figure 13 a) presents, e.g., three error patterns of size 3, 4, and 5 respectively; Figure 13 b) shows the possible regions of incidence of central error, for example, for a 48-bit codeword (SILVA *et al.*, 2020a; Silva *et al.*, 2018).

Figure 13 – (a) Three adjacent error patterns in a 3×3 matrix format, and (b) possible region of central error incidence in a 48-bit word



Another type of error commonly used in this area is burst error. A burst error is a multiple error that covers b contiguous bits in a word; where at least the first and last bits are wrong (GRACIA-MORAN *et al.*, 2018; GRACIA-MORáN *et al.*, 2018). Let b be the burst length, the burst error pattern injects ec error combinations, such that:

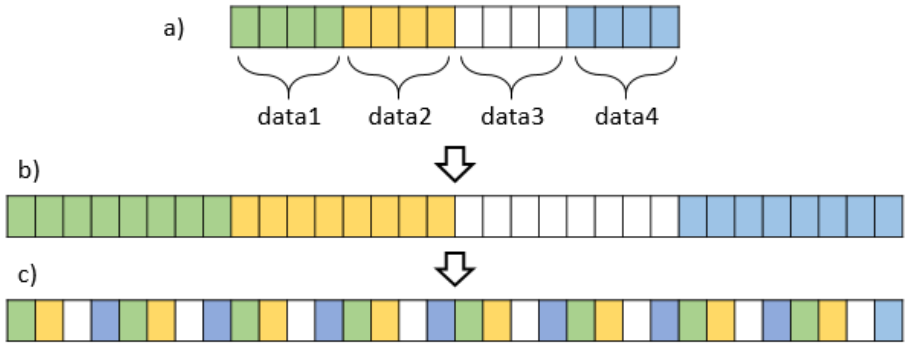
$$ec = \begin{cases} (n-b+1) \times 2^{b-2} & \forall b \geq 2 \\ n-1 & \forall b = 1 \end{cases} \tag{3.1}$$

where n is the total length of the encoded word.

Burst error analysis is done when the ECC is placed in one dimension. Figure 14 a) presents four four-bit words and Figure 14 b) presents the four words encoded in eight bits. An additional technique can be used in One-dimensional (1D)-ECCs: interleaving. According to (RADAELLI *et al.*, 2005; FREITAS *et al.*, 2020c), interleaving is an error mitigation technique

that makes MBUs become SBUs in different codewords, contributing to higher code correction capability. Figure 14 c) show the 32-bit word scrambled with the interleaving technique.

Figure 14 – Four-word eight-bit interleaving technique. In (a), four four-bit words are encoded in four eight-bit words, as shown in (b). In the 32-bit word in (b), the interleaving technique is applied to form the word in (c). This technique improves correction rates for adjacent errors.



In the exhaustive method of error injection, all possible combinations up to e errors are evaluated into a codeword of n bits. The works of (AFRIN; SADI, 2017; FREITAS *et al.*, 2020a) use this type of error. The exhaustive procedure makes the number of combinations equal to $\binom{n}{e}$. For example, in an experiment that evaluates all possible scenarios of 5 errors in a 32-bits ECC, the number of combinations is equal to $\binom{32}{5} = 201,376$. In some cases, to limit the time spent in the simulation, 20,000 samples were used for each test, which has a 99% confidence level.

RAO *et al.* (2014) proposed assessing ECCs using the 36 error patterns of most incidence in memories, attained with simulation results with a commercial tool for evaluating strikes of neutron particles (for more information, see RAO *et al.* (2014)). Figure 15 shows these patterns that were employed to assess ECCs in other works like in Silva *et al.* (2018) and AHILAN; DEEPA (2016). These patterns were used to design two ECCs of this thesis, as will be seen in chapter 5.

These error patterns were inserted in codeword performing some adjustments. For example, the error pattern 2 has two adjacent bitflips on the same row; thus, it is inserted into the memory as follows: (i) the leftmost bit of this pattern is set as reference; (ii) the possible insertion positions are established (as shown in Figure 16 (a)); (iii) this pattern is placed in all valid positions of the matrix; (iv) each pattern is placed w times in the 8x8 matrix, with $w = 64$ only for the patterns with 1 bitflip, the other error patterns make $w < 64$.

Figure 16 exemplifies areas where error patterns 2, 3, and 18 can be placed; these areas take into account the size and shape of the pattern. We must delimit the insertion region

Figure 15 – Thirty-six error patterns used in the experiments, encompassing one simple error, ten double errors, twenty triple errors, and five quadruple errors (adapted from (RAO *et al.*, 2014)).

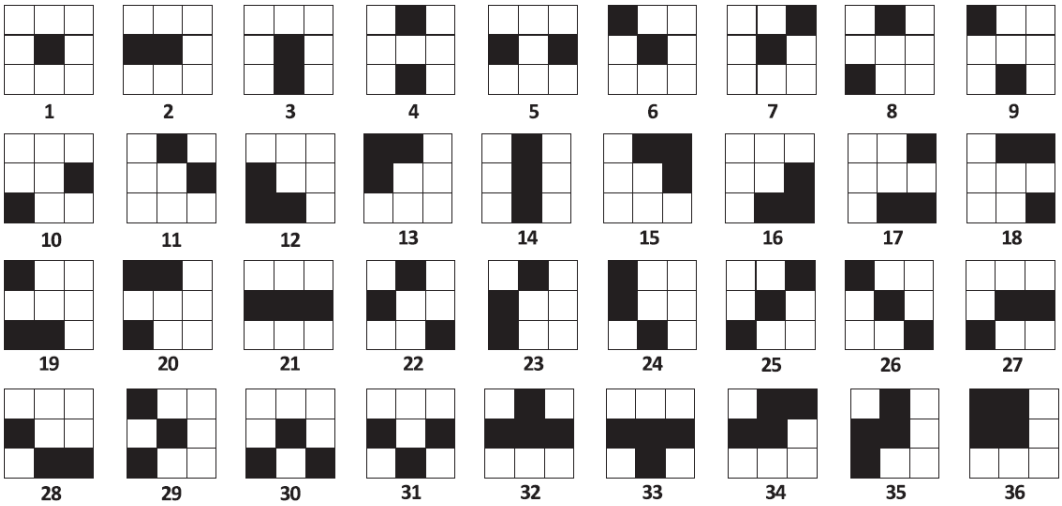
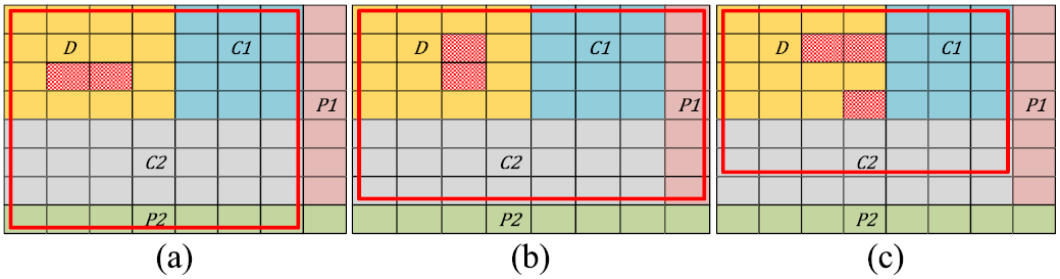


Figure 16 – (a), (b) and (c) show error patterns 2, 3 and 18, respectively. The red rectangles show regions where the pattern can be inserted into memory.



boundaries of each error pattern to ensure that the codeword obtains the exact pattern format. The region boundaries, which limits the number of patterns placed into the codeword, are highlighted by the red rectangles in Figure 16; for instance, error patterns 2 and 3 have 56 insertion possibilities, while error pattern 18 has only 42 insertion possibilities.

The reference of all error patterns is the upper left bit of each pattern. Figure 16(a) illustrates that the pattern 2 cannot be placed in the last column, as there are not two memory spaces available. Similarly, Figure 16(b) displays that pattern 3 cannot be placed on the last line. Finally, Figure 16(c) shows that the pattern 18 is limited to row six and column seven.

3.4.2 Reliability

The reliability analysis of this thesis is based on the works of SILVA *et al.* (2020) and ARGYRIDES *et al.* (2007). The following statements were assumed (also assumed by ARGYRIDES *et al.* (2007)): (i) transients faults occur with a Poisson distribution, and (ii) bit faults are statistically independent.

Let i be the number of errors and λ the error rate of a single bit per day (typical value of λ used in the experiments of this thesis is 10^{-5} upsets/bit/day (ARGYRIDES *et al.*, 2007; SILVA *et al.*, 2020a)), and assuming that (i) transients errors occur with a Poisson distribution, and (ii) bit flips occurrences are statistically independent, then (3.2) estimates $P_{in}(t)$, which is the probability of occurring exactly i errors in a given memory word with n bits at time t. Equation (3.3) estimates $P_n(t)$ which is the probability of having errors in a memory due to the rate λ over time.

$$P_{in}(t) = \binom{n}{i}(1-e^{-\lambda t})^i e^{-\lambda t(n-i)} \tag{3.2}$$

$$P_n(t) = 1 - e^{n\lambda t} \tag{3.3}$$

Let σ be the maximum number of errors for which the ECC was evaluated, and $\varepsilon(i)$ is the error coverage rate for each of the i errors, (3.4) estimates the memory reliability in time t considering ε of a given ECC.

$$r(t) = 1 - P_n(t) + \sum_{i=1}^{\sigma} P_{in}(t) \times \varepsilon(i) \tag{3.4}$$

Since M is the number of addresses in memory and each memory address consists of a single codeword, then (3.5) calculates $R(t)$, which is the reliability at time t of all memory. Note that $M = 1$ means the evaluation of a memory encompassing a single codeword.

$$R(t) = r(t)^M \tag{3.5}$$

3.4.3 Redundancy and cost analysis

The function of redundancy in an error-correcting code is to allow the detection and correction of errors in a message transmitted in a communication channel, through the decoding process (ZARAGOZA, 2006). The redundancy of a ECC is the number of parity bits in a codeword. More precisely

$$r = n - log_2 M \tag{3.6}$$

where r is the number of redundancy bits, n is the number of codeword bits, and M is the number of codewords (MOON, 2005). For a binary linear code, $M = 2^K$, so $r = n - k$.

If a large amount of redundancy bits favors the higher detection and correction rate of a code, on the other hand, the hardware implementation cost of a ECC increases significantly. Thus, as well as the cost analysis that will be presented below, the redundancy of a code must always be taken into account in a design in this area.

In this thesis, the variables dr, rr and ro will be used to represent, respectively, the data rate, redundancy rate and redundancy overhead, as shown in equations 3.7 to 3.9.

$$dr = k/n \tag{3.7}$$

$$rr = r/n \tag{3.8}$$

$$ro = r/k \tag{3.9}$$

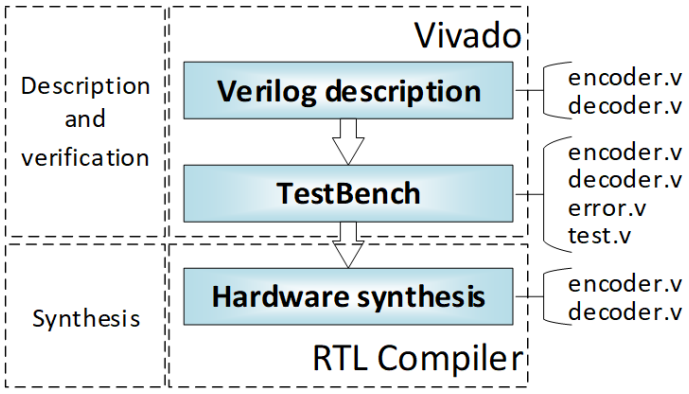
dr, referenced in some works as code rate, is the percentage of k data bits in the n codeword bits. rr and ro are metrics that regard the redundancy impact on the ECC. rr and ro are the percentage of redundancy bits r in relation to the n and k, respectively. Usually, low-cost ECCs have high values of dr and low values of rr and ro.

In addition to redundancy, the cost of a ECC is also measured by its hardware implementation cost. According to the systematic literature review, discussed in chapter 4, the most analyzed synthesis variables in the current works are area, power and delay in both the encoding and decoding processes. Usually the authors place greater emphasis and interest on the decoder values, because that's where most of the calculations take place.

In this thesis, the sequence for obtaining the synthesis results is presented in Figure 17. Initially, encoder (encoder.v) and decoder (decoder.v) are described using Verilog in Register Transfer Level (RTL). To verify the encoder and decoder behavior, it was implemented a TesbBench that includes a test file (test.v) and an error file (error.v). Next, the waveforms of the circuits were validated using Xilinx's Integrated Development Environment (IDE) software known as Vivado Design Suite. Finally, Verilog codes to obtain the values of delay, area consumption, power dissipation for encoder and decoder were synthesized. The syntheses were

performed using the RTL Compiler software with the 65nm CORE65GPSVT standard cell library.

Figure 17 – Encoder and decoder description, verification and synthesis flow.



3.5 Summary

In this chapter, the main methods used in this work were presented: the systematic literature review that was designed to find out what is most current in the area and propose the codes of this thesis, the test methodology that was used to obtain the correction results , and the most used analysis methods to compare ECCs.

In the SLR preparing process, 24 papers were obtained using the Snowballing Forward technique and the inclusion and exclusion criteria, from a set of works with many citations in this area. A search string was created based on the keywords of these works and, at the end, 32 papers were obtained. These works were the most current in this area. A table with the data to be extracted from these papers was created in order to carry out an analysis that will be carried out in the next chapter.

A generic methodology for obtaining correction results was also presented. The ECCs found in the SLR and those proposed in this thesis follow the same flow. The number of repetitions of the process depends on two variables: codeword size and type of error to be inserted.

The analysis methods that were presented were the most found in the SLR papers: correction capability and types of errors, reliability, and cost and redundancy analysis. Among the most common types of errors are adjacent, burst and exhaustive. Some works also use a table with 36 error patterns with the highest incidence rates in memories that were obtained with a commercial tool to assess strikes of neutron particles.

Another way to compare different ECCs is using reliability equations. It takes into account not only correction rates but also codeword size. This should be highlighted, as code with a higher correction capability than another can be less reliable. For example, imagine two codes *A* and *B*, where *A* has a higher correction capacity than *B*. These two ECCs have the same amount of data bits and different redundancy (code *A* has more redundancy than code *B*). The *A* code may be less reliable in this case if the difference in correction rate is not so high and the redundancy is significantly different between both.

Redundancy analysis is performed using three variables: data rate, redundancy rate and redundancy overhead. All these variables take into account the values of k, r and/or n. It is important to point out that a code with more redundancy has a higher implementation cost, in addition to a higher financial cost, larger memories need to be used. Regarding the implementation cost, the works of this thesis are based on the costs generated by the RTL Compiler software with a 65nm standard cell library.

The next chapter deals with the details of SLR. A 2D-ECC classification, redundancy metrics, most used data sizes, most used analysis methods, and different metrics for code evaluation are presented.

4 STATE OF THE ART IN 2D ECC

4.1 Introduction

This chapter presents a SLR, a study that standardizes the entire review process, excluding biases and exposing reliable conclusions (ALEXANDRE *et al.*, 2020). Besides, Bajaj and Sangwan (BAJAJ; SANGWAN, 2019) argue that SLR is a reliable, auditable, and rigorous method for knowing the status in a given research domain. This systematic review is based on (ALEXANDRE *et al.*, 2020; AL-SAREM *et al.*, 2019; BAJAJ; SANGWAN, 2019; BRITO *et al.*, 2020; LINDEN; HADAR, 2019), whose methodology provides consistent means of answering research questions in an objective and impartial manner.

The recent growth in the number of papers in the field of ECCs indicates the need to synthesize evidence found in an in-depth analysis of the state-of-the-art. Identifying the main concepts and issues addressed allows for consolidating and standardizing the proposed methods and making fair comparisons among the ECC proposals.

There is no single method for implementing an effective and efficient SLR. The researched works show that an SLR usually covers planning, conducting, and summarizing phases subdivided into other sub-phases. The planning encompasses the research objective and SLR employed protocol. The conducting identifies and selects studies based on planning and, subsequently, performs data extraction and synthesis. Finally, the summary phase reports and evaluates the synthesized data. The next subsection presents the five sub-phases that comprise the planning and conducting of the SLR carried out in this work.

4.2 Primary Studies

Table 4 presents a summary of the data collected during the evaluation of the 32 primary studies; this summary allows a comparative analysis of the following elements: (i) Work – identification of the work and authors; (ii) Year – allowing to identify the number of works and ECC tendencies along the years; (iii) Classification – containing the terminology adopted to classify the 2D-ECC types; (iv) Target application – aiming to correlate the proposed ECC type with a given target application; (v) Data size and redundancy overhead – the data and redundancy sizes of the codeword enable to verify ECC tendencies and define ECC overhead metrics; (vi) Fault injection – aiming to understand how the works validate or evaluate their ECC proposals;

(vii) Complementary Metal Oxide Semiconductor (CMOS) Technology – enabling to compare the manufacture technologies employed on the encoder/decoder synthesis.

Table 4 – Summary of data collected from 32 primary studies

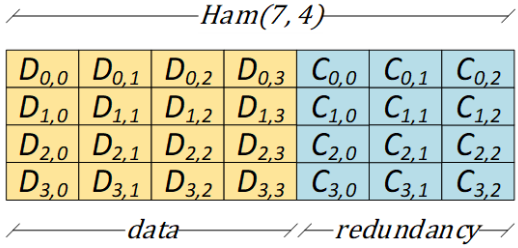
Work	Year	Classif.	Applic.	Data size	Data rate	Redund.	Fault injection	Technology
Ahilan et al.	2015	EPC	Generic	32	50.0	100.0	Random	180 nm
Anitha et al.	2015	EPC	Space	32	47.1	112.3	-	-
Erozan et al.	2015	PC	Generic	32	40.5	146.9	Adjacent	-
Liu et al.	2015	PC	Space	16	50.0	100.0	-	90 nm
Rahman et al.	2015	MC	Generic	64	70.3	42.2	Exhaustive	-
Castro et al.	2016	PC	Space	16	40.0	150.0	Adjacent	45 nm
Mandal et al.	2016	MC	Generic	49	65.3	53.1	Random	-
Manoj et al.	2016	EPC	Critical	32	47.1	112.5	-	180 nm
	2016	EPC	Critical	64	47.1	112.5	-	180 nm
Sundary et al.	2016	EPC	Generic	20	43.5	130.0	-	-
Yedere et al.	2016	EPC	Generic	32	48.5	106.3	-	90 nm
Afrin et al.	2017	S2D	Generic	32	50.0	100.0	Exhaustive	-
Kamatchi et al.	2017	EPC	Space	32	53.3	87.5	-	-
Liu et al.	2017	EPC	Space	32	42.1	137.5	Adjacent	65 nm
Raha et al.	2017	MC	Generic	32	53.3	87.5	-	-
Silva et al.	2017	MC	Critical	16	50.0	100.0	Adjacent	65 nm
Tambatkar et al.	2017	MC	Space	32	47.8	112.5	-	45 nm
Athira et al.	2018	PC	Generic	32	53.3	87.5	Random	90 nm
Goerl et al.	2018	MC	Critical	32	44.4	125.0	Random	-
Li et al.	2018	S2D	Generic	16	57.1	75.0	-	65 nm
	2018	S2D	Generic	16	40.0	150.0	-	65 nm
	2018	S2D	Generic	32	69.6	43.8	-	65 nm
	2018	S2D	Generic	32	53.3	75.0	-	65 nm
	2018	S2D	Generic	64	80.0	25.0	-	65 nm
	2018	S2D	Generic	64	69.6	43.8	-	65 nm
Moran et al.	2018	S2D	Generic	32	80.0	25.0	Adjacent	45 nm
	2018	S2D	Generic	32	66.7	50.0	Adjacent	45 nm
Silva et al.	2018	PC	Critical	16	41.0	143.8	Adjacent	65 nm
	2018	PC	Critical	16	40.0	150.0	Adjacent	65 nm
	2018	PC	Critical	16	29.6	237.5	Adjacent	65 nm
Magalhaes et al.	2019	MC	Critical	16	40.0	150.0	Adjacent	65 nm
Priya et al.	2019	EPC	Generic	32	50.0	100.0	-	-
Zhang et al.	2019	EPC	Space	32	50.0	100.0	Adjacent	180 nm
Freitas et al.	2020	PC	Space	16	33.3	200.0	Adjacent	65 nm
Freitas et al.	2020	PC	Space	16	25.0	300.0	Exhaustive	65 nm
Kumar et al.	2020	MC	Generic	32	69.6	43.8	-	-
Neelima et al.	2020	MC	Generic	64	67.4	48.4	-	28 nm
	2020	MC	Generic	64	62.1	60.9	-	28 nm
Rohde et al.	2020	MC	Space	64	44.4	125.0	-	-
Sai et al.	2020	MC	Generic	32	57.1	75.0	-	45 nm
Silva et al.	2020	PC	Critical	32	49.2	103.1	Adjacent	65 nm
Silva et al.	2020	MC	Critical	32	57.1	75.0	Adjacent	65 nm
	2020	MC	Critical	32	50.0	100.0	Adjacent	65 nm

Note: In Classification: EPC = Extended Product Code; PC = Product Code; MC = Mixed Code; and S2D = StraightForward 2D-ECC.
Note: "-" means that the work does not contain the information about the subject.

4.3 2D-ECC Classification

A 2D-ECC is characterized by having data and/or redundancy bits in two dimensions, normally named row and column. This definition allows including any 1D-ECC physically organized in rows and columns in the 2D-ECC class. We defined as Straightforward 2D-ECC (S2E) the codes organized in this 2D physical structure, but that remain to correct errors with 1D-algorithms. Figure 18 exemplifies an S2E containing 16 data bits organized in a matrix format; each row is encoded independently with Ham(7,4), a short representation of the Hamming code implemented with three redundancy bits to protect four data bits (HAMMING, 1950). Examples of S2Es are found in the works (AFRIN; SADI, 2017), (GRACIA-MORAN *et al.*, 2018) and (LI *et al.*, 2018).

Figure 18 – An S2E example containing 16-bit data (matrix D); each row of the matrix is encoded independently with Ham(7,4).



Afrin et al. (2017) propose a 4×16 matrix ECC with 32 data bits and 32 redundancy bits. Each 8 data bits of each row is independently encoded. The first redundancy bit stores the first data bit inverted. The other seven redundancy bits are XOR operations between a pair of bits - 1st and 2nd, 2nd and 3rd, 3rd and 4th, and so on (AFRIN; SADI, 2017). Gracia-Moran et al. introduce the Flexible Unequal Error Control (FUEC) methodology, developed to satisfy a certain number of syndromes to correct adjacent errors. The authors present two 2D-ECCs with 8 and 16 bits of redundancy but with the same error coverage, designed to correct 1-bit errors and 2 and 3 adjacent bits in the same row or column (GRACIA-MORAN *et al.*, 2018). Li et al. propose two 2D-ECCs with 32 data bits for correcting up to 3 burst bitflips. The ECCs add 24 and 14 redundancy bits for 4×8 and 2×16 data formats, respectively (LI *et al.*, 2018). Additionally, codes are organized using interleaving.

Although this thesis focuses on 2D-ECCs whose coding is two-dimensional, this ECC class was included for the sake of completeness and understanding. The 2D-ECC group that we are interested in is complementary to S2E; they present some level of encoding intersection

between the dimensions as a common characteristic, i.e., at least one bit of data or redundancy change implies encoding both dimensions. This complementary group was organized into three classes: Product Code (PC), Extended Product Code (EPC), and Mixed Code (MC). Figure 19 shows the number of papers for each ECC class per year, and Table 5 correlates 2D-ECC classes to works, highlighting the encoding methods that the 2D-ECC employs.

Figure 19 – The graph indicates the number of works by classification (S2E, PC, MC, and EPC) divided by year of publication.

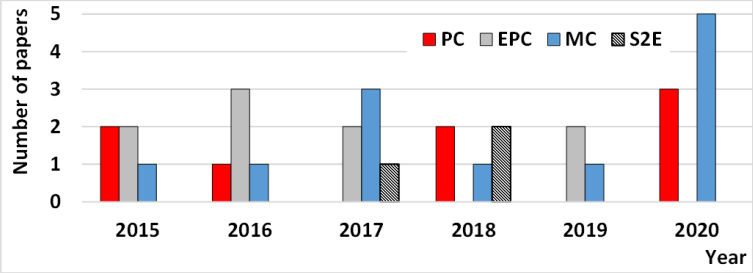


Table 5 – Class of 2D-ECC used in each work

Class	Work	Encoding Method
EPC	Pryia et al. (2019), Ahilan et al. (2015)	XOR operations and Parity
	Manoj et al. (2016), Anitha et al. (2015), Yedere et al. (2016)	Decimal Sum and Parity
	Zhang et al. (2019)	Parity
	Kamatchi et al. (2017), Liu et al. (2017), Sundary et al. (2016)	Hamming and Parity
MC	Mandal et al. (2016), Silva at al. (2017), Neelima et al. (2020)	Parity
	Rahman et al. (2015), Kuman et al. (2020)	Parity
	Goerl et al. (2018)	Parity and Duplication
	Tambatkar et al. (2017), Raha et al. (2017), Magalhaes et al. (2019)	Hamming and Parity
	Sai et al. (2020)	Hamming
	Silva et al. (2020)	Logic Operations and Parity
	Rhode et al. (2020)	XOR operations, Hamming and Parity
PC	Castro et al. (2016), Silva et al. (2018), Silva et al. (2020)b	Extended Hamming and Parity
	Erozan et al. (2015)	LDPC and Parity
	Liu et al.(2015)	MED and Parity
	Athira et al. (2018)	Hamming and Parity
	Freitas et al. (2020), Freitas et al. (2020)b	Extended Hamming in rows and columns
S2E	Afrin et al. (2017)	XOR and NOT operations
	Moran et al. (2018)	XOR operations and FUEC
	Li et al. (2018)b	3-burst error ECC in rows

4.3.1 Product Code (PC)

Elias, in 1954, described for the first time an ECC treated as a product of two codes, or simply PC, as a simple way to build long codes based on small ones.

Let α and β be the number of columns composing the data and redundancy areas, and let δ and ε be the number of rows composing the data and redundancy areas, respectively, such

that $\gamma = \alpha + \beta$ and $\theta = \delta + \varepsilon$. Then, each row of a PC is encoded using the $C_1(\gamma, \alpha, d_1)$ code, and each column is encoded using the $C_2(\theta, \delta, d_2)$ code, forming the $C_1 \times C_2$ code; therefore, each bit flip in the data region affects both the row and column of the corresponding bit. Figure 8 (as shown in chapter 2) illustrates the basic PC structure. Also, PC adds a region containing check bits of check bits, increasing the minimum Hamming distance and, consequently, the code correction potential (MOON, 2005). The PC minimum distance d_{PC} is computed by multiplying the distances of each 1D-ECC that make up the product code, i.e., $d_{PC} = d_1 \times d_2$.

PC increases the theoretical correction and detection capacity but also increases the redundancy costs, implying more area and energy consumption. Some authors proposed the modified PC to reduce the associated redundancy costs, which do not have the check bits of the check bits (MACWILLIAMS; SLOANE, 1977; ZARAGOZA, 2006). Figure 9 (as shown in chapter 2) illustrates the structure of a modified PC, and 4.1 displays its minimum distance calculation d_{mpc}.

$$d_{mpc} = d_1 + d_2 - 1 \tag{4.1}$$

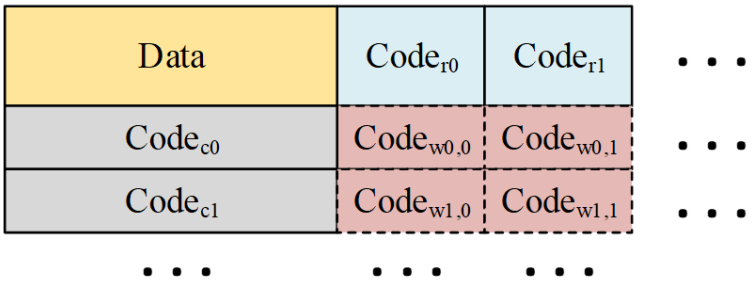
Based on this explanation, the works of (Erozan; Cavus, 2015), (CASTRO *et al.*, 2016), (Silva *et al.*, 2018), (SILVA *et al.*, 2020b), (LIU *et al.*, 2015), (ATHIRA; YAMUNA, 2018), (FREITAS *et al.*, 2020b) and (FREITAS *et al.*, 2020a) are ECCs in PC format. In (CASTRO *et al.*, 2016), (Silva *et al.*, 2018), and (SILVA *et al.*, 2020b), authors use extended Hamming in rows and parity in columns to encode each word; these works present iterative decoding using the row and column check bits to correct the data area, and the check bits of check bits to correct both the row and column check bits areas. Each correction in a given area can enable a new error correction iteratively, increasing the error correction capacity of the code. FREITAS *et al.* (2020) implemented the PCoSA, a PC applying extended Hamming to rows, columns, and check bits of check bits areas. The same authors propose LPC (FREITAS *et al.*, 2020b), a lightweight version of PCoSA codeword removing the redundancy area of check bits of check bits. LPC is a modified PC that reaches near error correction rates of PCoSA by improving the decoding algorithm. The works (Erozan; Cavus, 2015), (LIU *et al.*, 2015), (ATHIRA; YAMUNA, 2018) propose modified PCs that employ a single row of parity bits to encode columns and a more complex code to encode rows. Erozan; Cavus (2015) propose to codify each row employing the Euclidian Geometry Low-Density Parity Check (EG-LDPC); ATHIRA; YAMUNA (2018) employ five Hamming bits to encode each one of the four rows of

8 data bits; finally, LIU *et al.* (2015) propose using Multi-bit Error Detection (MED), a code capable of detecting multiple errors by applying 4 redundancy bits to each row of 8 data bits.

4.3.2 Extended Product Code (EPC)

EPC is a special case of PC that uses more than one code per row and/or column; therefore, C_1, C_2, or both are heterogeneous codes. Besides, this class can also have check bits of check bits, regardless of whether they are homogeneous or heterogeneous codes, as exemplified in Figure 20.

Figure 20 – Structure of an EPC.



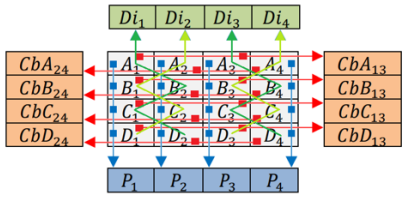
The works (KAMATCHI; THILAGAVATHI, 2017), (PRIYA; VIJAY, 2019), (MANOJ; BABU, 2016), (ANITHA; JEEVIDHA, 2015), (LIU *et al.*, 2017), (AHILAN; DEEPA, 2015), (ZHANG *et al.*, 2019), (SUNDARY; LOGISVARY, 2016), and (YEDERE; PAMULA, 2016) are classified as EPCs. KAMATCHI; THILAGAVATHI (2017) propose the Modified Decimal Matrix Code (MDMC) for encoding 32 data bits in a 2x16 matrix, with each row divided into four 4-bit regions. Each row is encoded by adding the two odd regions with the two even regions, totaling 10 bits per row since each sum requires 5 bits. Besides, each column in the data region is encoded with a parity bit. The authors in (PRIYA; VIJAY, 2019) and (AHILAN; DEEPA, 2015) divide the 32 bits of data into two rows of 16 bits. Each row encodes two sets of four bits executing a bitwise XOR operation; thus, eight redundancy bits are added per row. Columns are encoded using parity. MANOJ; BABU (2016) divided the 64 data bits into two rows of 32 bits each. A sum is applied to each set of two four-bit words for each row, resulting in five bits. This same structure is done four times per row, adding 20 redundancy bits; also, columns are encoded with parity bits. ANITHA; JEEVIDHA (2015) describe a similar technique presented in (MANOJ; BABU, 2016), but for a 32-bit codeword implemented in two rows of 16 bits, totaling ten redundancy bits per row. YEDERE; PAMULA (2016) perform another similar organization; the 32 data bits are divided into two rows. The first eight bits are added to the last eight bits for

each row, resulting in nine redundancy bits per row. LIU *et al.* (2017) split 32 data bits into eight rows. Each row encodes the four bits with parity and Hamming. However, there is no ECC to encode the four columns of the data region. The column coding is done only in the Hamming check bit region, and parity is used for each pair of bits. ZHANG *et al.* (2019) split 32-bit data into four rows. Their proposal uses parity every two bits in the row (adds four per row) and column (adds 2 per column), totaling 32 redundancy bits. Finally, SUNDARY; LOGISVARY (2016) organize 20-bit codeword in a 4×5 format; the ECC encodes each one of the four rows using Hamming and each one of the five columns using parity every two bits.

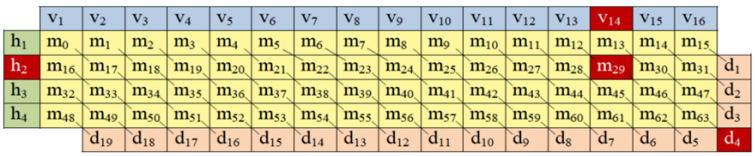
4.3.3 Mixed Code (MC)

MC is a class of 2D-ECCs containing at least one bit of data or redundancy whose change implies encoding both dimensions but cannot be classified as PC or EPC. The works of (MANDAL *et al.*, 2016), (GOERL *et al.*, 2018), (SILVA *et al.*, 2017), (SAI *et al.*, 2020), (NEELIMA; SUBHAS, 2020), (TAMBATKAR *et al.*, 2017), (SILVA *et al.*, 2020a), (RAHA *et al.*, 2017), (ROHDE; MARTINS, 2020), (MAGALHAES *et al.*, 2019), (RAHMAN *et al.*, 2015), and (KUMAR *et al.*, 2020) are examples of MC; Figure 21 displays the 2D-ECCs proposed on works (SILVA *et al.*, 2017; NEELIMA; SUBHAS, 2020).

Figure 21 – Examples of Mixed Codes: (a) MRSC (SILVA *et al.*, 2017), (b) HVD (NEELIMA; SUBHAS, 2020).



(a)



(b)

MANDAL *et al.* (2016) propose the Modified Matrix Code (MMC) for FPGA-based systems. MMC corrects multiple errors in a 7×7 data matrix, employing 13 parity bits encoded both diagonally and vertically. Additionally, MMC has two redundancy bits associated with each column; one redundancy stores the XOR of the even bits, and the other one stores the XOR of

the odd bits.

GOERL *et al.* (2018) present the Parity per Byte and Duplication (PBD) technique that uses parity for each byte and duplicates the content. For example, a 32-bit word (4 bytes) requires four parity bits, and the resulting 36 bits are duplicated, totaling 32 data bits and 40 bits redundancy.

Figure 21 (a) illustrates the Matrix Region Section Code (MRSC) developed by SILVA *et al.* (2017), a structure of 16 data bits and 16 redundancy bits, totaling a 4×8 matrix. MRSC implements the redundancy in (i) a parity bit for each one of the four diagonals in the data area; (ii) a parity bit for each one of the four rows of the data area; and (iii) Two check bits for each row resulting from the XOR operations between bits 1 and 3 and between bits 2 and 4.

SAI *et al.* (2020) propose a 2D-ECC for detecting and correcting multiple errors for a 4×8 data matrix. The ECC applies $Ham(7,4)$ to each one of the eight 4-bit diagonals of the data region, totaling an increase of 24 redundancy bits.

NEELIMA; SUBHAS (2020) propose an ECC based on Horizontal-Vertical-Diagonal (HVD) in 4×16 and 2×32 data formats, including 39 and 67 bits of redundancy, respectively. The authors call this Three-dimensional (3D) encoding technique, shown in Figure 21 (b), because it encodes bits horizontally, vertically, and diagonally. For both formats, each of the rows, columns, and diagonals has a parity bit. RAHMAN *et al.* (2015) propose the Horizontal-Vertical-Double-Bit-Diagonal (HVDD) technique that can correct up to three errors, and it is similar to the technique proposed in (NEELIMA; SUBHAS, 2020); however, in HVDD, the diagonals have two parity bits. TAMBATKAR *et al.* (2017) use HVD with Hamming applied to the redundancy bits to increase the error correction rate. The codeword has 4×8 data with 4-row parity bits, 8-column parity bits, and 11-diagonal parity bits, totaling 23 bits organized in two 8-bit words and one 7-bit word. These three words are encoded with three Hamming, each with 4 check bits, totaling a further 12 redundancy bits; the final codeword has 35 redundancy bits.

SILVA *et al.* (2020) developed the extended Matrix Region Section Code (eMRSC), a version that extends the 16 data bits of MRSC (SILVA *et al.*, 2017) to 32 bits. The authors propose a new region scheme to reduce redundancy bits while maintaining a high error correction rate; they show the experimental results with two codeword structures with 24 and 32 redundancy bits.

RAHA *et al.* (2017) present the Horizontal-Vertical Parity and Diagonal Hamming (HVPDH) method to protect a 4×8 data matrix. HVPDH adds 28 redundancy bits organized in

4-row and 8-column parity bits and 4 Hamming check bits for each of the 4 diagonal 8-bit words.

ROHDE; MARTINS (2020) propose a 2D-ECC with 64 bits of data organized with interleaving in five 11-bit words and one 9-bit word. The code has 4 check bits for each of the 6 rows and a further 56 parity bits for rows, columns, data, and check of the calculated check bits, totaling 80 check bits.

MAGALHAES *et al.* (2019) propose the Parity Hamming Interleaved Correction Code (PHICC), designed to correct multiple errors in a 4x4 data matrix. PHICC employs extended Hamming (3 check bits and a parity bit) for each row, and a parity bit for each column of the codeword, treating data and check bits in an interleaved way.

KUMAR *et al.* (2020) proposed a technique that uses only 14 parity bits to correct adjacent errors in a 32-bit data matrix, resulting in an efficacy equal to the Matrix code, reducing redundancy bits, consumed area, dissipated power, and delay.

4.3.4 Final Remark - Encoding Method

Six of the eight PC works apply parity as the C_2 method to code columns. Also, five use extended Hamming as C_1, C_2, or both encoding methods. An important fact is that 100% of the PC works utilize either extended Hamming or parity. 100% of the eight EPC papers apply parity, and three of them also apply Decimal Sum, a coding technique that uses the binary sum of n-bit words. Finally, 13 of the 14 MC works implement parity, five employ Hamming, and 100% apply one of these two techniques. Thus, the encoding methods most used in 2D-ECC implementations are Hamming and parity; their implementation simplicity, which produces low area consumption, power dissipation, and latency, is the main reason for their usage.

4.4 Data Size and Redundancy Metrics

This section explores and compares the data size of the 2D-ECC codewords together with metrics for redundancy overhead assessment.

4.4.1 Data Size

The SLR analysis revealed that most 2D-ECCs are designed to operate with standard memory-processor buses encompassing 16, 32, or 64 bits. Together with the regular matrix format of 2D-ECCs, these data sizes force the implementation of 1D-ECC with 4 or 8 bits in rows

and columns, producing 4×4, 4×8, 8×4, or 8×8 data matrices. Some 2D-ECCs implement 32 and 64 bits using 16 or 32 bits in rows, making 2×16, 4×16, and 2×32 data matrices. Exceptions are found in two special data organizations (SUNDARY; LOGISVARY, 2016) and (MANDAL *et al.*, 2016) containing 20 and 49 data bits. SUNDARY; LOGISVARY (2016) employed 20-bit data since their work targets a 20-bit multiplier; the authors also explored their proposed correction model for cases in the range of 10 to 128 bits. MANDAL *et al.* (2016) use a 7×7 data matrix, resulting in 49 bits, but they explain that the same ECC organization can be applied to other matrix sizes.

Table 6 shows that more than half of the works (i.e., (Erozan; Cavus, 2015), (AFRIN; SADI, 2017), (GOERL *et al.*, 2018), (SILVA *et al.*, 2020b), (GRACIA-MORAN *et al.*, 2018), (KAMATCHI; THILAGAVATHI, 2017), (SAI *et al.*, 2020), (LI *et al.*, 2018), (PRIYA; VIJAY, 2019), (TAMBATKAR *et al.*, 2017), (SILVA *et al.*, 2020a), (ATHIRA; YAMUNA, 2018), (RAHA *et al.*, 2017), (MANOJ; BABU, 2016), (ANITHA; JEEVIDHA, 2015), (LIU *et al.*, 2017), (AHILAN; DEEPA, 2015), (ZHANG *et al.*, 2019), (YEDERE; PAMULA, 2016), and (KUMAR *et al.*, 2020)) assess 2D-ECCs targeting 32-bit data memories. Eight authors carry out experiments with 16-bit memories ((CASTRO *et al.*, 2016), (SILVA *et al.*, 2017), (Silva *et al.*, 2018), (LI *et al.*, 2018), (LIU *et al.*, 2015), (FREITAS *et al.*, 2020b), (FREITAS *et al.*, 2020a), and (MAGALHAES *et al.*, 2019)) and four authors work with 64-bit memories ((NEELIMA; SUBHAS, 2020), (LI *et al.*, 2018), (MANOJ; BABU, 2016), (ROHDE; MARTINS, 2020), and (RAHMAN *et al.*, 2015)).

Table 6 – Relationship between number of 2D-ECCs and Data Size

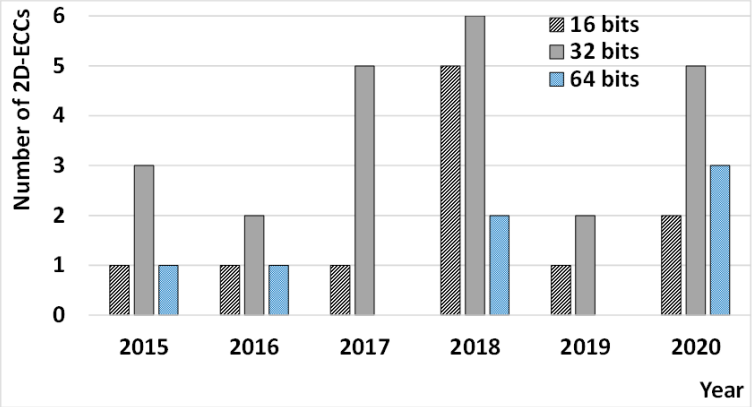
	16 bits	20 bits	32 bits	49 bits	64 bits
Number of Papers	11	1	23	1	7
Percentage	25.6%	2.3%	53.5%	2.3%	16.3%

Note that Table 6 has more than 32 ECC, as some papers discuss more than one ECC size. (LI *et al.*, 2018) (2018) propose a 32-bit ECC in a 4×8 data region and evaluate the proposed scheme for 16 and 64 bits, showing the number of redundancy bits added, the ability to correct errors, and the encoder and decoder latencies. (MANOJ; BABU, 2016) (2016) developed a proposal considering 64 bits, but also presented the implementation in 32 bits, showing the number of redundancy bits and error correction ability.

Figure 22 shows the number of 2D-ECC per year regarding the memory size. Most studies are for proposals of 32-bit data, which had the largest number of publications in the years

2017, 2018 and 2020. Besides, in all years, the number of 2D-ECCs with 32-bit data always is equal to or greater than the number of 2D-ECCs of other data sizes. Finally, the last year has shown a growth of 64-bit proposals; however, the number of works and sampling time is not sufficiently representative.

Figure 22 – Number of papers published per year, from 2015 to 2020, in relation to the data size



4.4.2 Redundancy Metrics

The number of redundancy bits is one of the determining factors in detecting and correcting errors. Since 2D-ECCs are normally used to mitigate critical system failures, these codes typically have numerous redundancy bits. Additionally, the proportion of the redundancy bits in relation to the data or codeword bits directly influences the memory storage area and the implementation costs of the encoding and decoding circuits.

This section was explored using only dr and ro (as shown in subsection 3.4.3) since rr is complementary to dr. For example, $Ham(7,4)$, a short representation of the Hamming code with $k = 4$ and $r = 3$, has $dr = 4/7 = 57.1\%$, $rr = 3/7 = 42.9\%$ and $ro = 3/4 = 75\%$. Figure 23 to Figure 26 show the number of 2D-ECC works according to dr and ro, and the publication year. These figures present 43 proposals for 2D-ECC; this number is higher than the 32 selected works because some works have more than one proposal. This is the case of (Silva *et al.*, 2018), which proposes two 16-bit codes, including 23 and 38 redundancy bits. The same is true for other works such as (GRACIA-MORAN *et al.*, 2018), (NEELIMA; SUBHAS, 2020), (LI *et al.*, 2018), (SILVA *et al.*, 2020a), and (MANOJ; BABU, 2016), which change the number of redundancy or data bits.

Figure 23 shows the total number of 2D-ECCs according to the dr range. The graphic was divided into six ranges; the leftmost range represents codes with a lot of redundancy

Figure 23 – Number of 2D-ECC proposals per dr range.

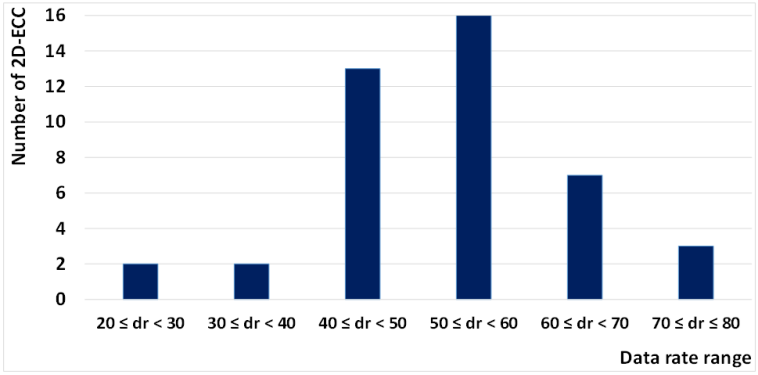


Figure 24 – Number of 2D-ECC per year and dr range.

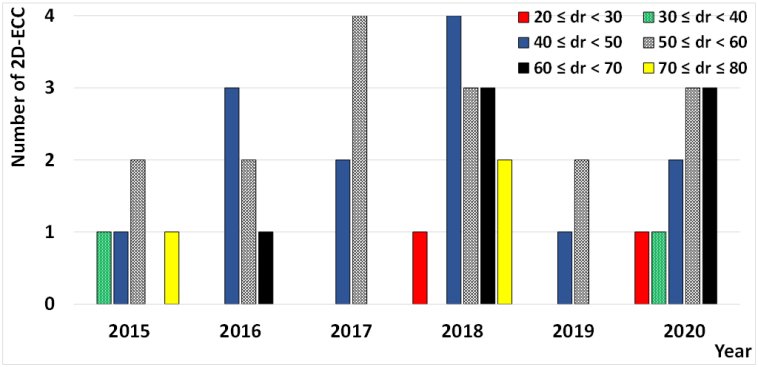


Figure 25 – Number of 2D-ECC proposals per ro range.

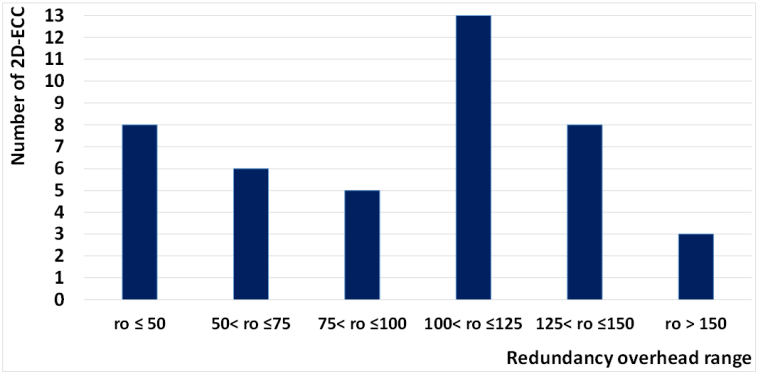
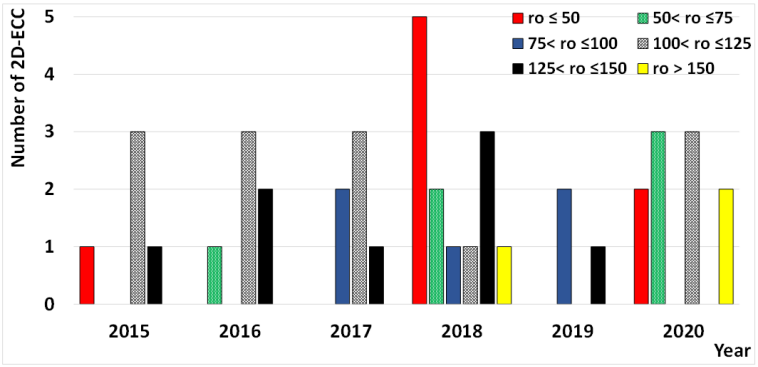


Figure 26 – Number of 2D-ECC proposals per year and ro range.



in relation to the size of the final word, and the rightmost range represents codes that have a low redundancy rate in relation to the total encoded data. On the one hand, 29 ECCs (67%)

have dr between 40% and 60%, meaning that most current 2D-ECCs use an average of 50% of their area for redundancy; i.e., they present ECC with a sound tradeoff between error correction efficacy and implementation and usage costs. Besides, seven ((CASTRO *et al.*, 2016), (Silva *et al.*, 2018), (TAMBATKAR *et al.*, 2017), (RAHA *et al.*, 2017), (MANOJ; BABU, 2016), (LIU *et al.*, 2017), and (AHILAN; DEEPA, 2015)) of the nine papers with more than three citations are in this dr range. On the other hand, only four proposals are within the limits of low and high dr ranges. Three works ((GRACIA-MORAN *et al.*, 2018), (LI *et al.*, 2018), and (RAHMAN *et al.*, 2015)) describe very low-cost codes ($70 \leq dr \leq 80$), justifying that many applications require low-energy decoder implementations and low-memory usage with acceptable error correction capacity. Furthermore, only two other works ((Silva *et al.*, 2018) and (FREITAS *et al.*, 2020a)) present very high-cost codes ($20 \leq dr < 30$), justifying that critical or space applications require ECCs with more correction capacity and thus higher redundancy rates, as the process technology decreases, rising the MBU rates.

Figure 24 aims to correlate the data illustrated in Figure 23 with the years in which the works took place. Figure 24 shows the number of proposals increases in recent years, signaling a trend in 2D-ECC researches.

Figure 25 shows the number of 2D-ECC according to six *ro* ranges. The leftmost and rightmost ranges represent codes that added little and lot redundancy in relation to data, respectively. Almost 50% of the 2D-ECCs have an *ro* between 75% and 125%, and among the nine proposals with more than three citations, four are in this range ((CASTRO *et al.*, 2016), (KAMATCHI; THILAGAVATHI, 2017), (TAMBATKAR *et al.*, 2017), and (RAHA *et al.*, 2017)).

Finally, Figure 26 correlates the data shown in Figure 25 with the years in which the works were published. It is worth mentioning that works with very high *ro* rates are showing a recent growth, as can be seen in 2018 (Silva *et al.*, 2018), and 2020 (FREITAS *et al.*, 2020b) and (FREITAS *et al.*, 2020a). Besides, 2D-ECCs with *ro* in the range (100, 150] have had publications in all evaluated years, representing practically 50% of all ECCs.

4.5 Most used Analysis Methods

This section presents methods and metrics used in the studies to assess the proposed 2D-ECCs, more specifically, (i) methods of fault injection to explore the capabilities of correction and detection error according to error patterns; (ii) meantime to failure used to estimate the ECC reliability over its use; (iii) evaluation costs for implementing and operating the decoders and

encoders in given manufacturing technology; and (iv) metrics for reaching a multi objective function.

4.5.1 Fault Injection Method

Error detection and correction are the primary objectives of an ECC; the accurate evaluation of these objectives is critical in the ECC design. It is essential to define error patterns to carry out fair comparisons among ECCs. This SLR found five types of error injection patterns: Adjacent, Exhaustive, Random, Burst, and 36 predefined error patterns. Table 4 presents the papers and the error injection method used. These error patterns were detailed in the section 3.4.1.

Adjacent errors are used in the works (Erozan; Cavus, 2015), (CASTRO *et al.*, 2016), (SILVA *et al.*, 2017), (Silva *et al.*, 2018), (SILVA *et al.*, 2020b), (GRACIA-MORAN *et al.*, 2018), (SILVA *et al.*, 2020a), (LIU *et al.*, 2017), (FREITAS *et al.*, 2020b), (ZHANG *et al.*, 2019), and (MAGALHAES *et al.*, 2019) for imitating the structure of MCUs that occur around a certain neighborhood, as indicated by the works of (OGDEN; MASCAGNI, 2017; WIRTHLIN *et al.*, 2014; RAO *et al.*, 2014; QUINN *et al.*, 2007; LERAY *et al.*, 2004; SATOH *et al.*, 2000). In (CASTRO *et al.*, 2016), (Silva *et al.*, 2018), and (SILVA *et al.*, 2020a), the authors verified one million words generated pseudo-randomly for each scenario; they placed patterns ranging from one to eight errors in adjacent cells. The authors in (SILVA *et al.*, 2017) performed a million pseudo-random positions for each scenario, varying from one to seven errors; the authors of (Erozan; Cavus, 2015) performed a similar analysis, extending to 12-error patterns.

The authors, in (SILVA *et al.*, 2020b), evaluate the ECC efficacy employing a set of 10,000 patterns for each scenario ranging from one to eight errors, considering adjacent errors with a maximum distance of one bit. The authors in (GRACIA-MORAN *et al.*, 2018) evaluated the ECC proposal with experiments that considered injection of errors of the type (i) simple, (ii) horizontal adjacent from two to eight bits, (iii) vertical adjacent from two to five bits, and (iv) squares in 2×2, 3×3, and 4×4 formats. In (LIU *et al.*, 2017), all adjacent patterns from one to four errors were added in the codeword. The authors of (FREITAS *et al.*, 2020b) describe that the incidence of radiation or heating errors occurs within a certain neighborhood. They proposed a failure model that considers a central and 24 adjacent cells, forming a 5×5 area, as a possible error region. Finally, in (ZHANG *et al.*, 2019), the authors describe experimental results showing that a radiation event generates a maximum of up to four errors.

The works (AFRIN; SADI, 2017), (FREITAS *et al.*, 2020a), and (RAHMAN *et al.*, 2015) employ exhaustive analysis of errors in the experimental results but limit the maximum number of errors to avoid a lot of computational time. AFRIN; SADI (2017) injected all error possibilities a 32-bit data, but only in the data region, i.e., the redundancy region is not evaluated. FREITAS *et al.* (2020) simulated all combinations from one to seven bitflips. Finally, RAHMAN *et al.* (2015) do not detail the injection method, but they explain their solution detected all error combinations and correct up to three error.

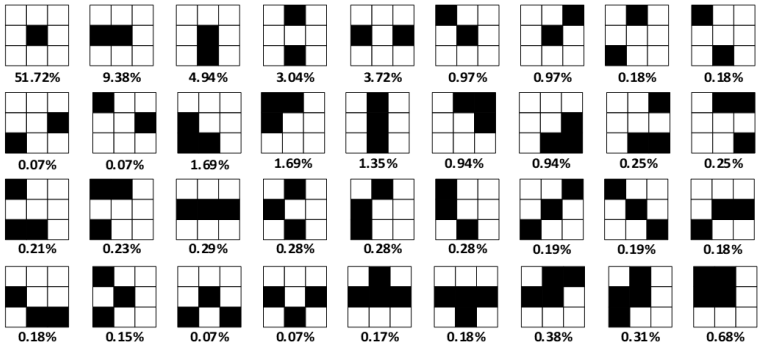
The fast evaluation of non-polarized scenarios with different criticality levels leads to works like (MANDAL *et al.*, 2016; GOERL *et al.*, 2018; ATHIRA; YAMUNA, 2018; AHILAN; DEEPA, 2015) adopting randomness as a way of injecting errors in their 2D-ECC experiments. MANDAL *et al.* (2016) compared the ECC efficacy in a 32×32 memory randomly injecting patterns with up to 40 errors. GOERL *et al.* (2018) randomly select from 1 to 10 bitflips per round error scenario to be verified in a 32-bit register. The experiments proposed by ATHIRA; YAMUNA (2018) injected one million test vectors with one to eight errors distributed in random positions. Finally, AHILAN; DEEPA (2015) mention that their experiments were verified employing hundreds of random errors.

The work (GRACIA-MORAN *et al.*, 2018; SAI *et al.*, 2020; NEELIMA; SUBHAS, 2020; LI *et al.*, 2018; PRIYA; VIJAY, 2019; TAMBATKAR *et al.*, 2017; RAHA *et al.*, 2017) describe examples of burst errors. The work of GRACIA-MORAN *et al.* (2018) focuses on the design of a 2D-ECC to mitigate adjacent errors, but they evaluate the ECC efficacy for tolerating burst errors. SAI *et al.* (2020) explored a 2D-ECC capable of correcting until 8 errors in burst format. NEELIMA; SUBHAS (2020) explore two works that use the burst error method to analyze the ECC efficacy in 32-bit memory; one work capable of correcting up to four errors and the other one capable of correcting up to 11 errors, both in burst format. LI *et al.* (2018) propose an ECC capable of correcting up to three burst errors in each data matrix row. PRIYA; VIJAY (2019) evaluate an ECC to mitigate burst errors in a group of information bits affected by radiation. TAMBATKAR *et al.* (2017) describe the existence of methods to detect and correct single, multiple, or burst errors, implying various redundancy overloads and energy consumption. Finally, RAHA *et al.* (2017) describe an ECC based on Multidirectional Parity Code for mitigating errors in the data region and Hamming for increasing the ability to correct burst errors in a noisy environment.

RAO *et al.* (2014) performed a simulation of neutron incidence using a commercial

assessment tool. Having as inputs details about the radiation environment and the target memory layout and technology, the tool uses a nuclear database for calculating the distribution of the current pulses generated for each memory cell. These pulses of current are then injected into a SPICE netlist to extract the SEU and MBU rates. The evaluation was performed for an SRAM memory with a 45nm CMOS technology. Figure 27 shows the 36 adjacent error patterns with the respective probability of occurrence. ROHDE; MARTINS (2020), FREITAS *et al.* (2020), and OGDEN; MASCAGNI (2017) are works that used this standard for 2D-ECC efficacy evaluation.

Figure 27 – Single and multiple error patterns in a 45nm SRAM with the respective probability of occurring each error pattern (adapted from (RAO *et al.*, 2014)).



The works (KAMATCHI; THILAGAVATHI, 2017; SAI *et al.*, 2020; NEELIMA; SUBHAS, 2020; LI *et al.*, 2018; PRIYA; VIJAY, 2019; TAMBATKAR *et al.*, 2017; LIU *et al.*, 2015; RAHA *et al.*, 2017; MANOJ; BABU, 2016; ANITHA; JEEVIDHA, 2015; ROHDE; MARTINS, 2020; SUNDARY; LOGISVARY, 2016; YEDERE; PAMULA, 2016) do not present the error injection method for ECC assessment. Instead, some of these works only report the maximum number of errors that the code can correct, and other works present a mathematical formulation to prove the ECC efficacy.

4.5.2 Reliability

The works (SILVA *et al.*, 2020a; ANITHA; JEEVIDHA, 2015; FREITAS *et al.*, 2020b; ZHANG *et al.*, 2019; FREITAS *et al.*, 2020a; MAGALHAES *et al.*, 2019) present the ECC efficacy degradation criterion across time, which is build based on equations (3.2) to (3.5) proposed by ARGYRIDES *et al.* (2007). The objective of these works is to compare ECCs according to the failure probability in a given time interval, assuming that errors can be cumulative over time and the number of errors is proportional to the codeword size. Thus, although the increase in redundancy bits tends to raise the error correction rate, this redundancy

increase also raises the probability of errors occurring over time.

Scrubbing is a memory error cleaning technique consisting of reading each memory address, correcting the error bits based on ECC, and writing the corrected data at the same address (HE *et al.*, 2020; ZHANG *et al.*, 2020; WANG *et al.*, 2018; STODDARD *et al.*, 2017). Since the incidence of memory errors can occur with both spatial and temporal distances, the evaluation of Mean Time To Failure (MTTF) metric and scrubbing technique can be explored, for example, in two situations of systems containing memory protected by ECC: (i) considering that the system cannot perform scrubbing in memory, as it operates in a critical situation, such as a space mission that implies reduced battery consumption, or (ii) considering that the system can perform scrubbing in memory in any time interval. In the first case, MTTF serves as a limit for the ECC efficacy degradation criterion since errors remain cumulatively in memory and no error recovery can be made throughout the system operation. In the second case, MTTF can define a tradeoff between the desired ECC efficacy and the scrubbing period. Thus, the choice of ECCs with different efficacy and costs of implementation and operation can be compensated by a shorter scrubbing period.

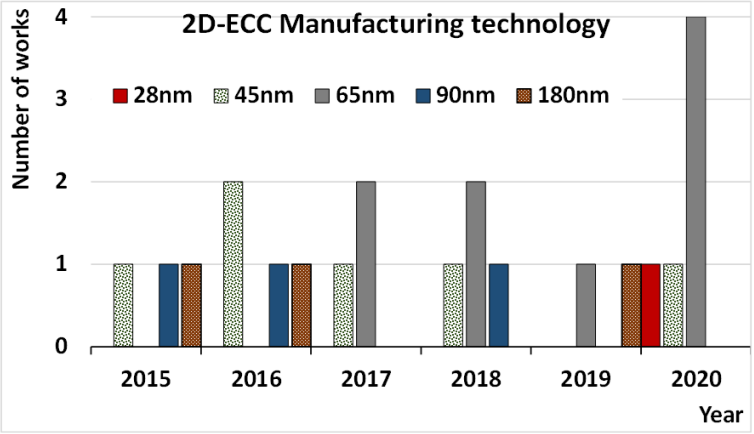
SILVA *et al.* (2020) computed reliability for M= 1, 8, and 16, and $\lambda = 10^{-5}$ for 8000 operating days. FREITAS *et al.* (2020) treat memory reliability for 15000 days using $M = 1$ and three values of λ (10^{-4}, 10^{-5}, and 10^{-6}); the same authors explore in (FREITAS *et al.*, 2020a) a thousand address memory (M=1000). ZHANG *et al.* (2019) also calculate the reliability of a thousand address memory using $\lambda = 10^{-5}$. MAGALHAES *et al.* (2019) evaluate using $M = 16$ and $\lambda = 10^{-4}$, 10^{-5} and 10^{-6}. Additionally, ANITHA; JEEVIDHA (2015) do not discuss the reliability formulation but use it to explore the ECC efficacy as a function of time. Finally, ARGYRIDES *et al.* (2011) expanded their work presented in (ARGYRIDES *et al.*, 2007) for evaluating memory sizes ranging from $1Mb$ to $128Mb$ with $\lambda = 10^{-5}$.

4.5.3 Process Technology

The logical or physical implementations are other features usually described in ECC works; Figure 28 shows the 2D-ECC manufacturing technologies most evaluated.

Twenty of all the evaluated papers (65%) describe the manufacturing technology used in the encoder and decoder 2D-ECC synthesis. Of this total, MANOJ; BABU (2016) and AHILAN; DEEPA (2015) implement two technologies, resulting in the 22 works of Figure 28. The SLR analysis shows a tendency of exploring technologies with 65 nm or less and the

Figure 28 – Number of 2D-ECC works organized by the manufacturing technology and year.



disappearance of older technology evaluations. Most works, 75%, are implemented at 45 or 65 nm, having a peak on 65 nm technology in 2020. The last exploration of 90 nm CMOS technology occurs in 2018, and the last work evaluating 180 nm technology was published in 2016. Still, in 2019 (ZHANG *et al.*, 2019), the authors performed some experiments referring to previous work employing 180 nm. Finally, only in 2020, one work explored a technology below 45 nm.

The SLR analysis showed that all syntheses have CMOS as the base technology, with 28 nm as the technological limit, far from the recent sub-5 nm CMOS technologies. Although most of these studies report that the latest technologies are more susceptible to errors, none of them present experiments on the efficacy of ECCs for correcting or detecting errors in the face of the technology variation. Besides, spatiotemporal error patterns concerning technology variation or investigations about error patterns in different memory operation environments were not found. Filling these gaps allows defining error injection patterns to explore and validate ECC proposals.

4.5.4 Multiobjective Metrics for ECC Assessment

The target application requirements, including correcting and detecting error effectiveness and implementation and operation efficiency in the memories and encoding and decoding circuits determines the ECC choice. Thus, studies as (CASTRO *et al.*, 2016; SILVA *et al.*, 2017; Silva *et al.*, 2018; SILVA *et al.*, 2020b; GRACIA-MORAN *et al.*, 2018; SILVA *et al.*, 2020a), which were based on the seminal papers (ARGYRIDES *et al.*, 2007) and (ARGYRIDES *et al.*, 2011), propose multiobjective metrics to assess correction capabilities.

ARGYRIDES *et al.* (2007) propose Correction Coverage per Cost (CCC) and Detec-

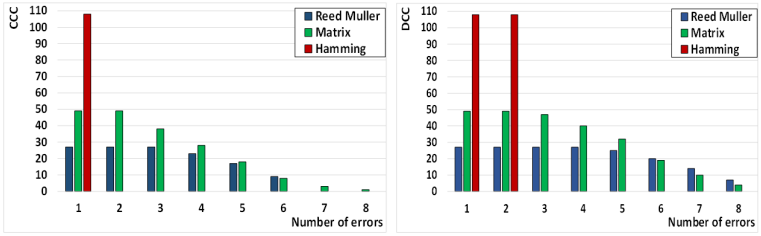
tion Coverage per Cost (DCC) metrics, defined in (4.2) and (4.3), respectively, to evaluate the ECC efficacies on detecting and correcting errors regarding the power, delay and area costs.

$$CCC_i = \frac{CR_i}{Power \times Delay \times Area} \tag{4.2}$$

$$DCC_i = \frac{DR_i}{Power \times Delay \times Area} \tag{4.3}$$

The authors normalized Area, Power and Delay according to a physical implementation without ECC protection, and *CR* (correction rate) and *DR* (Detection rate) are presented in percentage values. Note that the correction and detection rates depend on the number of errors; thus, expressing CR_i, DR_i, CCC_i, and DCC_i according to the number of errors i was chosen, generating discrete graphics, as exemplified in Figure 29. Additionally, in (ARGYRIDES *et al.*, 2011), the authors employed only the CCC_i metric proposed in (ARGYRIDES *et al.*, 2007). The works (ARGYRIDES *et al.*, 2007; ARGYRIDES *et al.*, 2011) do not describe if the implementation and operation costs are extracted from the syntheses of decoder, encoder, or both.

Figure 29 – Correction Coverage per Cost (CCC) and Detection Coverage per Cost (DCC) according to the number of errors (based on (ARGYRIDES *et al.*, 2007)).



CASTRO *et al.* (2016) improve CCC_i, and DCC_i metrics proposing the Total Coverage per Cost (TCC_i), as shown in (4.4) . TCC_i covers both detection and correction performances regarding the number of errors i and employing in the denominator only the decoder implementation and operation costs. The authors use costs associated only with the decoder.

$$TCC_i = \frac{CR_i \times DR_i}{Power \times Delay \times Area} \tag{4.4}$$

SILVA *et al.* (2017) use the TCC_i metric of (CASTRO *et al.*, 2016) regarding the implementation and operation costs of both encoder and decoder. Area and Power were achieved

by adding the individual values of the encoder and decoder, but Delay considered the highest value between the encoder and decoder since the authors explain this value defines a frequency rate, which is not cumulative. All values were normalized by dividing by the smallest value. Additionally, the same authors of (SILVA *et al.*, 2017) use the TCC_i metric in (Silva *et al.*, 2018) but considering only the decoder to compose the synthesis costs.

SILVA *et al.* (2020) proposed CTC_i, which considers both encoder and decoder synthesis costs, as seen in (4.5) and (4.6). In (SILVA *et al.*, 2020b), the same authors use CDC_i metric, which is the same CTC_i metric adapted from (SILVA *et al.*, 2020a). It is worth mentioning that CR_i for (SILVA *et al.*, 2020b; SILVA *et al.*, 2020a) were obtained for adjacent errors, and the results were normalized (divided by the highest value) after calculating CTC_i or CDC_i.

$$CTC_i = \frac{CR_i}{Cost(Encoder) + Cost(Decoder)} \tag{4.5}$$

$$Cost(Encoder/Decoder) = Area \times Power \times Delay \tag{4.6}$$

(GRACIA-MORAN *et al.*, 2018) (2018) point out the importance of adding redundancy cost for computing a multiobjective metric. Therefore, based on (GRACIA-MORáN *et al.*, 2018), they proposed (4.7) to calculate the M_i metric, which includes Redundancy cost as the dr metric, i.e., data rate.

$$M_i = \frac{CR_i \times DR_i}{Area \times Power \times Delay \times Redundancy} \tag{4.7}$$

Multiobjective metric assists in the ECC selection. Still, the analysis of all the multiobjective metrics used so far to compare 2D-ECCs demonstrates three gaps in this research area: (i) non-inclusion of memory characteristics, (ii) lack of parameterization, and (iii) lack of standardization.

Regarding the target memory characteristics, the analysis was restricted to the relationship between data bits versus redundancy. No analyzed work considered in its multiobjective metrics the physical costs of implementing and operating the memories, such as the energy consumption for writing and reading the codeword; i.e., the implementation and operation analyzes were restricted to the encoding and decoding circuits.

The multiobjective metric must have parameterizable objectives to represent target application requirements better. For example, power consumption may be of greater importance than the area occupied for battery-powered applications; thus, attributing weights for each parameter allows reaching a most promising ECC for a given target application.

Each work chooses a multiobjective metric concerning relevant criteria or objectives for comparing the ECCs. However, there is no standardization of multiobjective metrics that could take into account the target application, and the lack of standardization difficult the comparison of ECCs from different works and often leads to biased analysis.

4.6 Summary

A SLR resulting in 32 work selection was conducted; a thorough analysis of these works allowed to consolidate five features of 2D-ECC studies used for mitigating memory errors: (i) 2D-ECC classification; (ii) data size and redundancy metrics; (iii) target application; (iv) analysis methods; and (v) trend on matrix ECCs.

2D-ECCs according to their coding model were classified. Depending on the codeword structure, 2D-ECCs are classified as PC, EPC, or MC, if a single bitflip changes two or more encoding regions; otherwise, they are classified as S2E.

When evaluating the codeword data size, more than half of the works being assessed employ 32-bit data ECCs; besides, they use about 50% of the codeword for redundancy bits, even though there are cases where the redundancy rate is larger, as in critical or space applications. Besides, there is a close relationship between the 2D-ECC classification and the target application; e.g., ECCs classified as PC and EPC are more likely to be used in space applications.

The validation and analysis of the 2D-ECC effectiveness are usually performed by injecting adjacent, exhaustive, or burst errors. 75% of the works use 45nm or 65nm CMOS manufacturing technology to compare encoder and decoder synthesis costs, such as area consumed and power dissipated; additionally, some works propose the use of multi-objective metrics considering error detection and/or correction efficacy together with synthesis costs; however, so far, there is no standardization of these metrics.

The SLR showed some trends in 2D-ECCs, such as the use of Decimal Matrix Coding (DMC) together with the Encoder Reuse Technique (ERT) technique. Also, several authors use three coding axes in the same 2D plane, and some authors are working on ECCs with three or more dimensions.

The ECC Efficacy and efficiency comparison require a standardized verification methodology, allowing to extract advantages/disadvantages and tradeoffs of each proposal regardless of specific bias defined in each work. However, SLR showed a diversity of experiments and metrics that hinder fair comparative analysis among ECCs. Therefore, the results presented here can help understand different approaches to find a standardization that is of great value for future 2D-ECC proposal analyses.

5 PROPOSED ECCS

5.1 Introduction

This chapter deals with the presentation of the three ECCs proposed in this thesis: PCoSA, OPCoSA and LPC; and the reasons for their choices based on the SLR presented in the previous chapter.

The reasons for choosing ECCs and their structures are listed. The structure of PC is chosen because the Hamming distance increases and the code correction rate is higher. The choice for basic Hamming and parity codes, forming the Extended Hamming, are determined because they are widely used in this area due to their simplicity of implementation and low cost. Finally, the amount of 16 data bits is based on the amount of papers with 16 data bits in SLR and based on important references from ECCs in this area.

Next, the structure of the code PCoSA and its equations are presented. It is a ECC with 16 bits data bits and 48 bits redundancy, forming a codeword with 64 bits. It is code designed to correct 100% of the 36 error patterns presented in RAO *et al.* (2014). Thus, several tables are created with the syndromes found after inserting each of the 36 error patterns in all positions of the codeword region. These tables are used in the decoding process to perform error correction.

The second proposed ECC is the OPCoSA, which differs from the PCoSA by the reduction of 16 bits, forming a ECC with 16 data bits and 32 redundancy bits. The encoding and decoding process follows the same idea as PCoSA with the exception of deleting one of the checkbit regions.

Finally, the last code proposed is LPC which keeps the same encoding structure and amount of bits as OPCoSA but completely modifies the decoding process to achieve higher correction rates. LPC has an iterative decoding process for Simple Errors (SEs) correction and an innovative algorithm proposal for Double Errors (DEs) correction.

5.2 Reasons for Choosing the Proposed ECCs

The ECCs proposed in this thesis were chosen based on the results obtained during the preparation of the SLR presented in chapter 4.

Three ECCs are proposed in the PC format, as this configuration has well-defined

correction, detection and Hamming distance equations, even knowing that additional techniques used in the decoding algorithm can exceed the resulting values equations or even reduce them if a reduction in the cost of implementation is designed. In the other types, for example the MC, the correction capacity and the Hamming distance depend on the organizational structure of each ECC and are quite diverse, as shown in the SLR presented of the previous chapter.

Another reason for choosing PC is that the Hamming distance of the code increases, favoring a higher error correction and detection. On the other hand, these types of codes are known to increase redundancy and implementation cost.

After choosing the structure of ECC, the next step is to define which codes to use in the rows and columns of these 2D-ECCs. According to SLR, the most used codes to compose these two-dimensional structures are Hamming and Parity and several times they are used together to form the extended Hamming. The reason is the simplicity of implementation, the low consumption of area, energy and latency. Thus, all proposals in this thesis use extended Hamming both in rows and columns.

The size of 16 databits for each ECC was defined based on two factors: i) size of the ECCs of the SLR and important references in the elaboration of this thesis. The first factor indicates that almost 80% of SLR works use 16 or 32 bits of databits. For the second, the works of CASTRO *et al.* (2016), SILVA *et al.* (2017), Silva *et al.* (2018) and one of the pioneering papers in the area and basis for the beginning of SLR from the previous chapter, ARGYRIDES *et al.* (2007), use 16 bits of data.

5.3 Product Code for Space Applications - PCoSA

The structure of the code PCoSA was initially based on the code Column Line Code (CLC) by Silva *et al.* (2018), which uses Extended Hamming to encode the rows and parity to encode the columns. PCoSA, in turn, uses extended Hamming for both rows and columns.

Therefore, PCoSA(64,16), which is based on Extended Ham(8,4), is the smallest possible product code format that implements PCoSA. Figure 30 illustrates PCoSA(64,16) format, wherein a 16-bit word (represented by bits D_0-D_{15}) is encoded into 64 bits distributed as follows: (i) 16 data bits, (ii) 12 row-check bits C1, (ii) 7 row-parity bits P1, (iii) 21 column-check bits C2 and (iv) 8 column-parity bits P2. This code format makes PCoSA have a minimum distance $d = 16$ since Extended Hamming has $d = 4$, increasing the detection and correction capability of PCoSA compared to CLC.

Figure 30 – PCoSA structure with 16 data bits. The code has five regions: data (D), check bits of the D rows (C1), check bits of the columns D and C1 (C2), parity of the rows D and C1, and C2 (P1), and parity of all columns (P2).

D_0	D_1	D_2	D_3	$C1_0$	$C1_1$	$C1_2$	$P1_0$
D_4	D_5	D_6	D_7	$C1_3$	$C1_4$	$C1_5$	$P1_1$
D_8	D_9	D_{10}	D_{11}	$C1_6$	$C1_7$	$C1_8$	$P1_2$
D_{12}	D_{13}	D_{14}	D_{15}	$C1_9$	$C1_{10}$	$C1_{11}$	$P1_3$
$C2_0$	$C2_1$	$C2_2$	$C2_3$	$C2_4$	$C2_5$	$C2_6$	$P1_4$
$C2_7$	$C2_8$	$C2_9$	$C2_{10}$	$C2_{11}$	$C2_{12}$	$C2_{13}$	$P1_5$
$C2_{14}$	$C2_{15}$	$C2_{16}$	$C2_{17}$	$C2_{18}$	$C2_{19}$	$C2_{20}$	$P1_6$
$P2_0$	$P2_1$	$P2_2$	$P2_3$	$P2_4$	$P2_5$	$P2_6$	$P2_7$

The encoding process calculates the check bits $C1_q$, $P1_q$, $C2_q$ and $P2_q$ through the equations 5.1 to 5.11, where q is the bit- index and $\oplus$ is the xor operation.

$$C1_q = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+3}, \forall q \in 0,3,6,9 \tag{5.1}$$

$$C1_{q+1} = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3}, \forall q \in 0,3,6,9 \tag{5.2}$$

$$C1_{q+2} = D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3}, \forall q \in 0,3,6,9 \tag{5.3}$$

$$C2_q = \begin{cases} D_q \oplus D_{q+4} \oplus D_{q+12}, \forall q \in 0,1,2,3 \\ D_{q-4} \oplus D_{q-1} \oplus D_{q+5}, \forall q \in 4,5,6 \end{cases} \tag{5.4}$$

$$C2_{q+7} = \begin{cases} D_q \oplus D_{q+8} \oplus D_{q+12}, \forall q \in 0,1,2,3 \\ D_{q-4} \oplus D_{q+2} \oplus D_{q+5}, \forall q \in 4,5,6 \end{cases} \tag{5.5}$$

$$C2_{q+14} = \begin{cases} D_{q+4} \oplus D_{q+8} \oplus D_{q+12}, \forall q \in 0,1,2,3 \\ D_{q-1} \oplus D_{q+2} \oplus D_{q+5}, \forall q \in 4,5,6 \end{cases} \tag{5.6}$$

$$P1_q = D_{4q} \oplus D_{4q+1} \oplus D_{4q+2} \oplus D_{4q+3} \oplus C1_{3q} \oplus C1_{3q+1} \oplus C1_{3q+2}, \forall q \in 0,1,2,3 \tag{5.7}$$

$$P1_q = C2_{7q-28} \oplus C2_{7q-27} \oplus C2_{7q-26} \oplus C2_{7q-25} \oplus C2_{7q-24} \oplus C2_{7q-23} \oplus C2_{7q-22}, \forall q \in 4,5,6 \tag{5.8}$$

$$P2_q = D_q \oplus D_{q+4} \oplus D_{q+8} \oplus D_{q+12} \oplus C2_q \oplus C2_{q+7} \oplus C2_{q+14}, \forall q \in 0,1,2,3 \tag{5.9}$$

$$P2_q = C1_{q-4} \oplus C1_{q-1} \oplus C1_{q+2} \oplus C1_{q+5} \oplus C2_q \oplus C2_{q+7} \oplus C2_{q+14}, \forall q \in 4,5,6 \tag{5.10}$$

$$P2_7 = P1_0 \oplus P1_1 \oplus P1_2 \oplus P1_3 \oplus P1_4 \oplus P1_5 \oplus P1_6 \tag{5.11}$$

With the codeword, the decoding process is given by the equations 5.12 to 5.27. Equations 5.12 to 5.14 and 5.15 to 5.17 compute the recalculated check bits $rC1_q$ and $rC2_q$, respectively. Additionally, Equations 5.18 to 5.22 compute the recalculated parity bits $rP1_q$ and $rP2_q$.

$$rC1_q = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+3}, \forall q \in 0,3,6,9 \tag{5.12}$$

$$rC1_{q+1} = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3}, \forall q \in 0,3,6,9 \tag{5.13}$$

$$rC1_{q+2} = D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3}, \forall q \in 0,3,6,9 \tag{5.14}$$

$$rC2_q = \begin{cases} D_q \oplus D_{q+4} \oplus D_{q+12}, \forall q \in 0,1,2,3 \\ D_{q-4} \oplus D_{q-1} \oplus D_{q+5}, \forall q \in 4,5,6 \end{cases} \tag{5.15}$$

$$rC2_{q+7} = \begin{cases} D_q \oplus D_{q+8} \oplus D_{q+12}, \forall q \in 0,1,2,3 \\ D_{q-4} \oplus D_{q+2} \oplus D_{q+5}, \forall q \in 4,5,6 \end{cases} \tag{5.16}$$

$$rC2_{q+14} = \begin{cases} D_{q+4} \oplus D_{q+8} \oplus D_{q+12}, \forall q \in 0,1,2,3 \\ D_{q-1} \oplus D_{q+2} \oplus D_{q+5}, \forall q \in 4,5,6 \end{cases} \tag{5.17}$$

$$rP1_q = D_{4q} \oplus D_{4q+1} \oplus D_{4q+2} \oplus D_{4q+3} \oplus C1_{3q} \oplus C1_{3q+1} \oplus C1_{3q+2}, \forall q \in 0,1,2,3 \tag{5.18}$$

$$rP1_q = C2_{7q-28} \oplus C2_{7q-27} \oplus C2_{7q-26} \oplus C2_{7q-25} \oplus C2_{7q-24} \oplus C2_{7q-23} \oplus C2_{7q-22}, \forall q \in 4,5,6 \tag{5.19}$$

$$rP2_q = D_q \oplus D_{q+4} \oplus D_{q+8} \oplus D_{q+12} \oplus C2_q \oplus C2_{q+7} \oplus C2_{q+14}, \forall q \in 0,1,2,3 \tag{5.20}$$

$$rP2_q = C1_{q-4} \oplus C1_{q-1} \oplus C1_{q+2} \oplus C1_{q+5} \oplus C2_q \oplus C2_{q+7} \oplus C2_{q+14}, \forall q \in 4,5,6 \tag{5.21}$$

$$rP2_7 = P1_0 \oplus P1_1 \oplus P1_2 \oplus P1_3 \oplus P1_4 \oplus P1_5 \oplus P1_6 \tag{5.22}$$

Applying Equations 5.23 to 5.26, the decoding algorithm computes $sind = [sC1, sP1, sC2, sP2]$ - a vector composed of four syndromes; i.e., $sC1$ and $sC2$, which are the check bit syndromes of $C1$ and $C2$, respectively, and $sP1$ and $sP2$, which are the row and column parity syndromes, respectively.

$$sC1 = \sum_{q=0}^{3} (C1_{3q} \oplus rC1_{3q}) + (C1_{3q+1} \oplus rC1_{3q+1}) + (C1_{3q+2} \oplus rC1_{3q+2}) \tag{5.23}$$

$$sC2 = \sum_{q=0}^{3} (C2_q \oplus rC2_q) + (C2_{q+7} \oplus rC2_{q+7}) + (C2_{q+14} \oplus rC2_{q+14}) \tag{5.24}$$

$$sP1 = \sum_{q=0}^{3} P1_q \oplus rP1_q \tag{5.25}$$

$$sP2 = \sum_{q=0}^{3} P2_q \oplus rP2_q \tag{5.26}$$

PCoSA decoding algorithm explores the $sind = [sC1, sP1, sC2, sP2]$ vector in the binary format $sind_b = [sc1, sp1, sc2, sp2]$. Equation 5.27 presents how to compute a binary element of $sind_b$ from its counterpart in $sind$. For example, if $sind = [0,2,2,3]$, then $sind_b = [0,1,1,1]$.

$$sx = \begin{cases} 0, & if sX = 0 \\ 1, & otherwise \end{cases} \tag{5.27}$$

As already indicated in chapter 4, PCoSA and OPCoSA were designed to correct the 36 error patterns presented in RAO *et al.* (2014). All 36 patterns were inserted in all possibilities and regions of Figure 30. For example, an error pattern containing a simple error is placed in the five regions (D, C1, P1, C2 and P2); an error pattern with an adjacent double error on the same row, is placed in 7 possibilities: D, D ∪ C1, C1, C1 ∪ P1, C2, C2 ∪ P1 and P2. The operator ∪ represents an area composed of more than one region; for instance, D ∪ C1 indicates that a double error has occurred, and one bit of this error is in region D and the other one is in region C1, as shown in Figure 31 (a).

Figure 31 – Error patterns (a) 2 and (b) 32 of Figure 27 placed in regions D ∪ C1, and 3D ∪ C1, respectively. The bold bits with red background represent the error pattern.

D_0	D_1	D_2	$\mathbf{D_3}$	$\mathbf{C1_0}$	$C1_1$	$C1_2$	$P1_0$
D_4	D_5	D_6	$\mathbf{D_7}$	$\mathbf{C1_3}$	$C1_4$	$C1_5$	$P1_1$
D_8	D_9	D_{10}	D_{11}	$C1_6$	$C1_7$	$C1_8$	$P1_2$
D_{12}	D_{13}	D_{14}	D_{15}	$C1_9$	$C1_{10}$	$C1_{11}$	$P1_3$
$C2_0$	$C2_1$	$C2_2$	$C2_3$	$C2_4$	$C2_5$	$C2_6$	$P1_4$
$C2_7$	$C2_8$	$C2_9$	$C2_{10}$	$C2_{11}$	$C2_{12}$	$C2_{13}$	$P1_5$
$C2_{14}$	$C2_{15}$	$C2_{16}$	$C2_{17}$	$C2_{18}$	$C2_{19}$	$C2_{20}$	$P1_6$
$P2_0$	$P2_1$	$P2_2$	$P2_3$	$P2_4$	$P2_5$	$P2_6$	$P2_7$

(a)

D_0	D_1	D_2	D_3	$C1_0$	$C1_1$	$C1_2$	$P1_0$
D_4	D_5	D_6	D_7	$C1_3$	$C1_4$	$C1_5$	$P1_1$
D_8	D_9	D_{10}	$\mathbf{D_{11}}$	$C1_6$	$C1_7$	$C1_8$	$P1_2$
D_{12}	D_{13}	$\mathbf{D_{14}}$	$\mathbf{D_{15}}$	$\mathbf{C_{19}}$	$C1_{10}$	$C1_{11}$	$P1_3$
$C2_0$	$C2_1$	$C2_2$	$C2_3$	$C2_4$	$C2_5$	$C2_6$	$P1_4$
$C2_7$	$C2_8$	$C2_9$	$C2_{10}$	$C2_{11}$	$C2_{12}$	$C2_{13}$	$P1_5$
$C2_{14}$	$C2_{15}$	$C2_{16}$	$C2_{17}$	$C2_{18}$	$C2_{19}$	$C2_{20}$	$P1_6$
$P2_0$	$P2_1$	$P2_2$	$P2_3$	$P2_4$	$P2_5$	$P2_6$	$P2_7$

(b)

Simulating all the possibilities of placing the 36 error patterns, considering the cases of miscorrection (which occur in patterns that have triple errors in the same row) allows us to create a table that associates the error pattern, its positioning and the values of the sind vector. Figure 31 (b) exemplifies the error pattern 32 mapped in 3D ∪ C1, indicating three errors in region D and one error in region C1.

After injecting all error patterns, Table 7 shows all 16 possibilities of syndrome $sind_b$; only bold patterns marked with '*' need to go through the correction algorithm as errors outside region D can be recalculated from D information.

Table 7 – Mapping of error patterns using $sind_b$=[sc1, sp1, sc2, sp2].

	$sind_b$	Error type	Number of error patterns
	0 0 0 0	No error	-
	0 0 0 1	Outside region D	4
	0 0 1 0	Outside region D	4
	0 0 1 1	Outside region D	20
	0 1 0 0	Outside region D	4
	0 1 0 1	Outside region D	8
	0 1 1 0	Outside region D	8
*	**0 1 1 1**	**Patterns 21,33**	**Total 84 - only 4 in region D**
	1 0 0 0	∅	-
	1 0 0 1	∅	-
*	**1 0 1 0**	**Pattern 36**	**Total 84 - only 4 in region D**
*	**1 0 1 1**	**Patterns 2, 5, 34**	**Total 17 - only 9 in region D**
	1 1 0 0	∅	-
*	**1 1 0 1**	**Pattern 14**	**Total 4 - only 2 in region D**
*	**1 1 1 0**	**Patterns 3, 4, 35**	**Total 18 - only 9 in region D**
*	**1 1 1 1**	**Several patterns**	**Total 213 - several in region D**

Note: ∅ - means an unreachable syndrome.

The column Number of error patterns shows the number of patterns displayed in Figure 27 for a given $sind_b$. For example, $sind_b$=[0, 0, 0, 1] encompasses four error patterns occurred outside region D; these patterns are 1, 2, 5 and 21, and all patterns falling in the region P2. Tables 19 to 23 described in Appendix A detail the error patterns, the correction method applied and region that generates a given $sind_b$ combinations, which are bolded and marked with '*' in Table 7.

This syndrome occurs with error patterns in the format $m \times c$, where m and c are the numbers of rows and columns with errors, respectively. For example, Figure 31 (a) and (b) display a 1×2 and 2×3 error format, respectively. In cases of miscorrection, or in cases where the error is in regions such as $D \cup C2$, the error patterns may have different dimensions from those calculated. The PCoSA algorithm uses Equations 5.28-5.30 to calculate the error size T.

$$T = m \times c \tag{5.28}$$

$$m = max(sC1, sP1) \tag{5.29}$$

$$c = max(sC2, sP2) \tag{5.30}$$

Table 24 in appendix A describes all possibilities of T, together with the corresponding pattern, region and correction method.

5.4 Optimized Product Code for Space Application - OPCoSA

Figure 32 shows the OPCoSA organization consisting of 16 data bits (D_0 to D_{15}), 12-row check bits ($C1_0$ to $C1_{11}$), 4-row parity bits ($P1_0$ to $P1_3$), 12-column check bits ($C2_0$ to $C2_{11}$), and 4-column parity bits ($P2_0$ to $P2_3$); it is the same organization as PCoSA (FREITAS *et al.*, 2020a), but without the check bits of check bits region, reducing 16-redundancy bits (33% of reduction). This modification removes OPCoSA from the product code class, being considered a modified product code.

Figure 32 – OPCoSA structure with 16 data bits.

D_0	D_1	D_2	D_3	$C1_0$	$C1_1$	$C1_2$	$P1_0$
D_4	D_5	D_6	D_7	$C1_3$	$C1_4$	$C1_5$	$P1_1$
D_8	D_9	D_{10}	D_{11}	$C1_6$	$C1_7$	$C1_8$	$P1_2$
D_{12}	D_{13}	D_{14}	D_{15}	$C1_9$	$C1_{10}$	$C1_{11}$	$P1_3$
$C2_0$	$C2_1$	$C2_2$	$C2_3$				
$C2_4$	$C2_5$	$C2_6$	$C2_7$				
$C2_8$	$C2_9$	$C2_{10}$	$C2_{11}$				
$P2_0$	$P2_1$	$P2_2$	$P2_3$				

This reduction of a PCoSA checkbit region to form the OPCoSA and the consequent alteration of the decoding algorithm is the focus of this subsection. This change in format makes the minimum distance of a modified product code, obtained by Equation 2.26, less than a conventional product code (MACWILLIAMS; SLOANE, 1977; ZARAGOZA, 2006).

Equation 2.26 shows that OPCoSA has a minimum distance of 7 because it uses a minimum distance in both codes (rows and columns) equal to 4. The organization of the OPCoSA matrix causes rows and columns to cross in just a single bit, and changing this bit implies the variation of three other bits in the line and three other bits in the column, thus modifying 7 bits; i.e., the minimum distance of 7.

OPCoSA coding employs equations 5.1 to 5.3 to compute the row check bits, equation 5.7 to calculate the row parity bits (the same as PCoSA), equations 5.31 to 5.33 to calculate the column check bits, and equation 5.34 to compute the column parity. This difference in relation to PCoSA for the equations $C2_q$ and $P2_q$ is due to the structure of the OPCoSA with one region less than the PCoSA. The same difference occurs for the decoding process.

$$C2_q = D_q \oplus D_{q+4} \oplus D_{q+12}, \forall q \in 0,1,2,3 \tag{5.31}$$

$$C2_{q+4} = D_q \oplus D_{q+8} \oplus D_{q+12}, \forall q \in 0,1,2,3 \tag{5.32}$$

$$C2_{q+8} = D_{q+4} \oplus D_{q+8} \oplus D_{q+12}, \forall q \in 0,1,2,3 \tag{5.33}$$

$$P2_q = D_q \oplus D_{q+4} \oplus D_{q+8} \oplus D_{q+12} \oplus C2_q \oplus C2_{q+4} \oplus C2_{q+8}, \forall q \in 0,1,2,3 \tag{5.34}$$

In OPCoSA decoding, equations 5.12 to 5.14 and 5.18 calculate the recalculated check bits rC1 and parity bits rP1, respectively; also, equations 5.35 to 5.37 and 5.38 recalculated check bits rC2 and parity bits rP2, respectively.

$$rC2_q = D_q \oplus D_{q+4} \oplus D_{q+12}, \forall q \in 0,1,2,3 \tag{5.35}$$

$$rC2_{q+4} = D_q \oplus D_{q+8} \oplus D_{q+12}, \forall q \in 0,1,2,3 \tag{5.36}$$

$$rC2_{q+8} = D_{q+4} \oplus D_{q+8} \oplus D_{q+12}, \forall q \in 0,1,2,3 \tag{5.37}$$

$$rP2_q = D_q \oplus D_{q+4} \oplus D_{q+8} \oplus D_{q+12} \oplus C2_q \oplus C2_{q+4} \oplus C2_{q+8}, \forall q \in 0,1,2,3 \tag{5.38}$$

As in PCoSA, the equations 5.23 to 5.27 are used to obtain the vectors *sind* and $sind_b$. The next step in the decoding process of OPCoSA, as well as PCoSA, is to obtain the syndrome table by placing the 36 patterns of RAO *et al.* (2014) in the OPCoSA codeword. Table 8 describes the positioning of all 36 error patterns in all regions, including all sixteen combinations of S_b. The correction algorithm works only if at least one error occurs in region D; i.e., only if S_b=[0111], [1010], [1011],[1101], [1110] or [1111]. Besides, Table 8 shows the type, number, and placement of the error patterns.

Table 8 – Mapping of error patterns using S_b = [sc1 sp1 sc2 sp2].

	S_b	Error type	Number and placement
	0 0 0 0	No error	-
	0 0 0 1		4 outside region D
	0 0 1 0		8 outside region D
	0 0 1 1		60 outside region D
	0 1 0 0		4 outside region D
	0 1 0 1	Unreachable syndrome	-
	0 1 1 0	Unreachable syndrome	8
*	**0 1 1 1**	**Patterns 21,33**	**2 inside region D**
	1 0 0 0		2 inside region D
	1 0 0 1	Unreachable syndrome	-
*	**1 0 1 0**	**Patterns 13, 20, 36**	**5 inside region D**
*	**1 0 1 1**	**Patterns 2, 5, 13,**	**Total 17 - only 9 in region D**
		15, 18, 20, 27, 31, 34	
	1 1 0 0		60 outside region D
*	**1 1 0 1**	**Pattern 14**	**1 inside region D**
*	**1 1 1 0**	**Patterns 3, 4, 12**	**Total 18 - only 9 in region D**
		13, 19, 20, 23, 24, 29, 35	**18 inside region D**
*	**1 1 1 1**	**Several patterns**	**94 in several regions**

Note: Lines marked with * are detailed in the next tables.

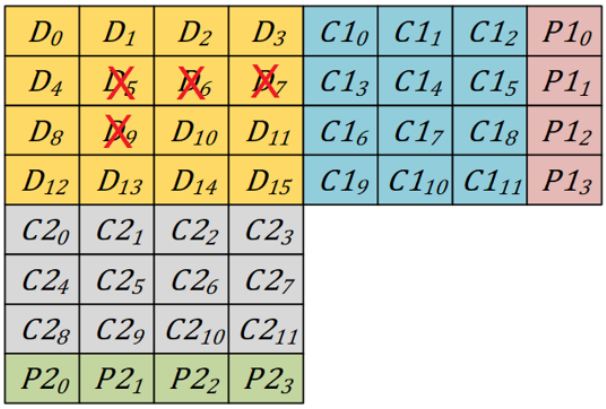
After mapping each S_b, the decoding algorithm searches for the syndrome-based error pattern, as shown in tables 25 to 29 in appendix B. This procedure is done for the first five S_b patterns in bold; for S_b=[1111] (Table 30), there is one more step.

S_b=[1111] occurs with error patterns in the format m×c, where m and c are the numbers of errors in the rows and columns, respectively. In cases of miscorrection or in cases where the error is in regions such as D ∪ C2, the error pattern can have diverse dimensions. The OPCoSA decoding algorithm uses equations 5.28 to 5.30 to calculate T. Table 30 presents all the T possibilities, the corresponding error pattern S, and the correction method.

The default conditions enable to correct several patterns in addition to those presented by the set of 36 error patterns. For example, the default condition for error patterns with S_b=[1111] is “Check m and c (equations 5.29 and 5.30). If m ≥ c, apply Hamming to all rows.

Otherwise, apply Hamming to all columns". The pattern that has four diagonal errors (bits D0, D5, D10, and D15), for instance, is corrected because S_b=[1111], S=[4444], and the default condition would do the correction of all bits. Figure 33 exemplifies another 4-bit error pattern that OPCoSA can correct, which is not included in the 36 error patterns. The syndromes of this error pattern are S_b=[1111] and S_b=[2232] (T=2×3). Thus, the decoding algorithm corrects "Inverting the two bits indicated by the two rows and the double error column and then apply Hamming to all columns". The correction process is done in two parts: (i) first, D5 and D9 are corrected; then, (ii) the complete correction is performed applying Hamming to all columns to correct D6 and D7 bits.

Figure 33 – Error pattern example that is not part of the 36 patterns analyzed but is fixed by the OPCoSA decoding algorithm.



5.5 Line Product Code - LPC

The third code proposed in this thesis is LPC - this code was proposed to increase the protection to 16 data bits keeping the same structure as OPCoSA (Figure 32). LPC is composed of a mixed decoding approach, including an iterative SE correction algorithm, followed by a DE correction algorithm. The main originality of the iterative algorithm is to carry out a preliminary analysis of the number of SEs in the rows and columns to apply correction heuristics. The DE algorithm is an original proposal for increasing the correction efficacy through error inference.

The coding equations are the same as for OPCoSA. LPC employs equations 5.1 to 5.3 to compute the row check bits, equation 5.7 to calculate the row parity bits (the same as PCoSA and OPCoSA), equations 5.31 to 5.33 to calculate the column check bits, and equation 5.34 to compute the column parity (the same as OPCoSA).

In LPC decoding, equations 5.12 to 5.14 and 5.18 calculate the recalculated check

bits rC1 and parity bits rP1, respectively (the same as PCoSA and OPCoSA); also, equations 5.35 to 5.37 and 5.38 recalculated check bits rC2 and parity bits rP2, respectively (the same as OPCoSA).

LPC uses the equations 5.39 to 5.42 to calculate the checkbits and parity syndromes of rows and columns, respectively, sCr_q, sPr_q, $sCcr_q$, and sPc_q.

$$sCr_q = C1_q \oplus rC1_q, \forall q \in 0,1,...,11 \tag{5.39}$$

$$sPr_q = P1_q \oplus rP1_q, \forall q \in 0,1,2,3 \tag{5.40}$$

$$sCc_q = C2_q \oplus rC2_q, \forall q \in 0,1,...,11 \tag{5.41}$$

$$sPc_q = P2_q \oplus rP2_q, \forall q \in 0,1,2,3 \tag{5.42}$$

Equations 5.43 and 5.44 describe the computations of sCR_q and sCC_q, which are achieved by applying a logical OR in the check-bit syndromes of the rows and columns, respectively.

$$sCR_q = sCr_q + sCr_{q+1} + sCr_{q+2}, \forall q \in 0,3,6,9 \tag{5.43}$$

$$sCC_q = sCc_q + sCc_{q+4} + sCc_{q+8}, \forall q \in 0,1,2,3 \tag{5.44}$$

Table 9 shows that sCR_q and sCC_q, together with sPr_q and sPc_q, are used to analyze whether the decoded data contains errors and the type of error that has been detected. For each row and column q, a SE is represented by SEr_q and SEc_q, respectively; similarly, a DE is denoted by DEr_q and DEc_q.

Table 9 – Meaning of the combinations of the syndrome bits

sC	sP	Error detection
0	0	None – or a possible quadruple error
0	1	Parity bit – or a possible triple error
1	0	Even error – a possible DE
1	1	Odd error – a possible SE

Note: Legend: (sC, sP) are the tuples (sCR_q, sPr_q) or (sCC_q, sPc_q) $\forall$ q $\in$ 0,1,2,3.

Once an SE is detected, the position of this error within the row or column is obtained by combining the weights of the check-bit syndromes. Equations 5.45 and 5.46 describe the error addresses in a row and column q, respectively.

$$EAr_q = 4sCr_q + 2sCr_{q+1} + sCr_{q+2}, \forall q \in 0,3,6,9 \tag{5.45}$$

$$EAc_q = 4sCc_q + 2sCc_{q+4} + sCc_{q+8}, \forall q \in 0,1,2,3 \tag{5.46}$$

5.5.1 Single Error Correction Algorithm - AlgSE

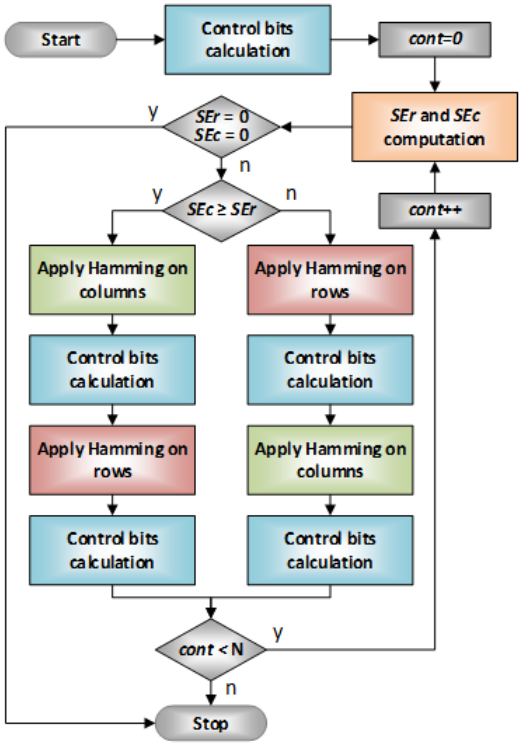
AlgSE applies SE corrections of type Ham(8,4) for rows and columns; this correction starts recalculating the check and parity bits, and with these values and the input codeword, calculate the syndromes. The Control bits calculation box of Figure 34 implements the set of calculus described above.

SEs can be corrected iteratively by applying Hamming first on rows and then on columns, or vice-versa. Several alternatives combining the number of successive errors on column or row corrections were assessed. The experiments demonstrated that AlgSE is more effective when starting the error correction for the set (i.e., rows or columns) with the highest number of SEs followed by one correction with the other set (i.e., rows followed by column or vice-versa). Therefore, AlgSE performs an heuristic that decides the correction order using the SE_r and SE_c variables, which are calculated by Equations 5.47 and 5.48, respectively.

$$SE_r = \sum_{q=0}^{3} SEr_q \tag{5.47}$$

$$SE_c = \sum_{q=0}^{3} SEc_q \tag{5.48}$$

Figure 34 – High-level description of AlgSE.



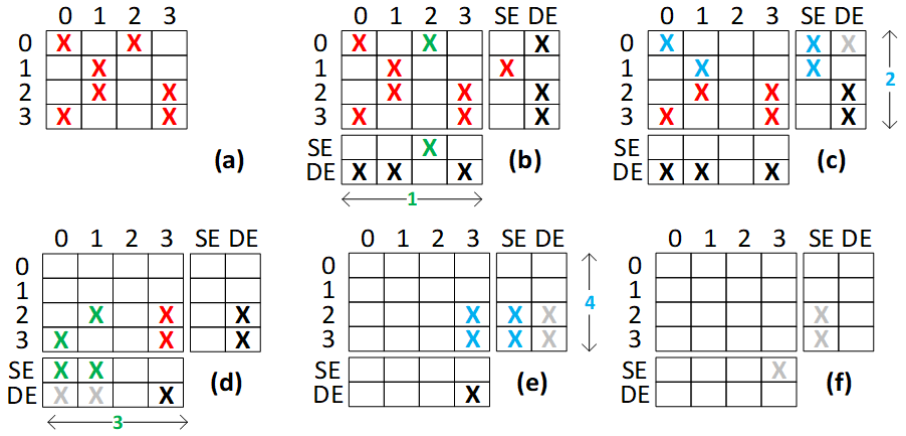
If $SE_c = SE_r = 0$, the algorithm considers that no SE was detected and ends the codeword decoding. Otherwise, AlgSE executes a loop sequence controlled by the cont counter. Each algorithm loop corrects first columns and then rows if $SE_c \geq SE_r$, or the opposite when $SE_c < SE_r$. Corrections to columns and rows occur in the Apply Hamming on columns/rows boxes.

The experiments explore up to 4 iterations through a sequence of column/row error corrections; because the LPC organization containing 4 rows and 4 columns does not allow a SE pattern needing more than 4 correction passages.

Figure 35(a) shows a pattern with 7 SEs in the data area. Figure 35(b) shows these same errors with the corresponding control variables indicating whether the algorithm detects an SE or DE in each row and column. The double arrows show whether the correction refers to the row or column, and the arrow number describes the sequence of correction steps.

Figure 35(b) shows that the execution of Apply Hamming on columns box on steps 1 allows correcting $D_{0,2}$. Next, Figure 35(c) displays that the execution of the Control bits calculation box produces two SEs ($D_{0,0}$ and $D_{1,1}$) on rows, which are corrected executing Apply Hamming on rows box. Subsequently, AlgSE starts a new loop recomputing SE_r and SE_c variables. Once again, $SE_c \geq SE_r$, thus, Figure 35(d) illustrates that AlgSE applies Hamming on columns to fix bits $D_{3,0}$ and $D_{2,1}$ in step 3. Finally, Figure 35(d) shows that the control

Figure 35 – Example of SE correction in the data area obtained through consecutive AlgSE loops. The check and parity bits do not contain errors and were omitted on purpose to avoid overloading the figure.



bits are recalculated, resulting in two SEs in rows ($D_{2,3}$ and $D_{3,3}$) that are corrected executing Apply Hamming on rows. AlgSE starts the last loop recomputing SE_r and SE_c variables. At this moment, Figure 35(f) shows that this error pattern was entirely correct; thus, $SE_r = SE_c = 0$ forcing to stop the AlgSE execution.

The $cont < N$ test defines the number of loops to be executed by the algorithm. The name of the algorithm is associated with N; i.e., $AlgSE0$, ..., $AlgSE3$ correspond to $N = 0, 1, 2, 3$, respectively. Any AlgSE greater or equal than $AlgSE1$ could correct the error pattern scenario illustrated in Figure 35(a).

Note that the AlgSE loop technique can be done automatically if the algorithm evaluates the number of loops necessary to correct each pattern dynamically, making a test to verify if an error correction has performed after each loop; if there were no error corrections, then the algorithm finishes. However, this type of exploration is not the focus of this thesis.

5.5.2 Double Error Correction Algorithm - AlgDE

The DE correction makes a cross-analysis of DEs detected in rows and columns, increasing the ability to correct data in a product code. This technique is based on the LPC matrix format, which allows for checking all data bits using Ham(8,4) arranged in columns and rows. While Ham(8,4) is a SECDED code, the rows and columns crossing allows inferring DEs by the majority analysis of the error events.

The correction technique analyzes all DE combinations with the corresponding error values obtained by the syndrome computation. Each DE combination produces a one-bit codeword address that the syndromes would point to as the cause of a SE. Although this address

does not point to the real source of the error, it is used to associate groups of DEs.

The extended Hamming code does not employ the parity bit in the error-bit address calculation; thus, although Ham(8,4) consists of 8 bits, only 7 bits are used, resulting in 21 combinations of DEs $\binom{7}{2}$. The syndromes produce 7 error addresses computed by Equations 5.45 and 5.46, with DEs being homogeneously distributed in 21 combinations, so each address corresponds to 3 DEs. For example, address 1 is produced whenever there are DEs described in the following three tuples: (D2,D3), (D0,C1) and (D1,C0). Figure 36 partially shows the 21 DE combinations, with the corresponding syndromes and the reference address for each combination.

Figure 36 – DE combinations for Ham(8,4). The parity bit is not represented, as it is not used to calculate the reference address, which is limited to 21 combinations $\binom{7}{2}$. Symbol 'E' represents an error in a bit within the codeword.

Comb.	Codeword							Recomp.			Syndromes				Double error	
	D0	D1	D2	D3	C0	C1	C2	C0	C1	C2	sC0	sC1	sC2	EA		
1			E	E						E	0	0	1	1	D2	D3
2	E					E			E	E	0	0	1	1	D0	C1
3		E			E			E		E	0	0	1	1	D1	C0
4		E		E					E		0	1	0	2	D1	D3
5	E						E		E	E	0	1	0	2	D0	C2
6			E		E			E	E		0	1	0	2	D2	C0
...																
19	E				E				E	E	1	1	1	7	D0	C0
20		E				E		E		E	1	1	1	7	D1	C1
21			E				E	E	E		1	1	1	7	D2	C2
	3	5	6	7	4	2	1									

Error address (**EA**)

Table 10 displays the 21 combinations, shown in Figure 36, grouped in sets of three tuples. Table 10 also shows the address used to reference the bitflip in the case of SE correction.

Table 10 – The 21 combinations of DEs grouped according to the address produced by the check bit syndromes.

Address	Single Error	Double Error		
1	C2	D2,D3	D0,C1	D1,C0
2	C1	D1,D3	D0,C2	D2,C0
3	D0	D1,D2	D3,C0	C1,C2
4	C0	D0,D3	D1,C2	D2,C1
5	D1	D0,D2	D3,C1	C0,C2
6	D2	D0,D1	D3,C2	C0,C1
7	D3	D0,C0	D1,C1	D2,C2

Figure 37 illustrates that AlgDE encompasses two steps. The first step contains two nested loops (between lines 5 and 37) used to fill the data matrix, which points out the bits of the

data area where the DEs may have occurred. The second step contains two other nested loops (between lines 38 and 43) that invert the data bits identified as DE in the data matrix.

Figure 37 – Pseudo-code of the DE correction algorithm - AlgDE.

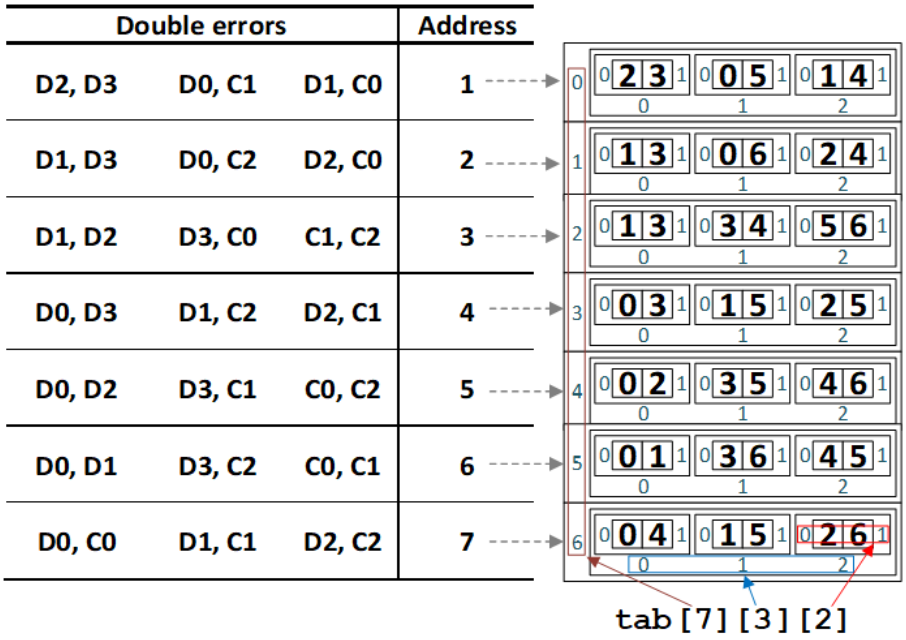
```
1   IN NATURAL tab[7][3][2], EAr[4], EAc[4]
2   IN BOOLEAN DEr[4], DEc[4]
3   INOUT BOOLEAN decWord[4][4]

4   NATURAL data[4][4] ← 0s
5   for rc ← 0 to 1 {
6      for k ← 0 to 3 {
7         BOOLEAN vDE ← rc = 0 ? DEr[k] : DEc[k]
8         if vDE = 1 {
9            NATURAL add ← rc = 0 ? EAr[k] : EAc[k]
10           BOOLEAN expt ← 0
11           for j ← 0 to 2 {
12              NATURAL b1 ← tab[add-1][j][0]
13              NATURAL b2 ← tab[add-1][j][1]
14              BOOLEAN e1 ← 0, e2 ← 0
15              if b1 = 3
16                       e1 ← rc = 0 ? DEc[b1] : DEr[b1]
17              if b2 = 3
18                       e2 ← rc = 0 ? DEc[b2] : DEr[b2]
19              if (b1=4 OR e1=1) AND (b2=4 OR e2=1) {
20                       if b1 = 3 {
21                              rc=0 ? data[k][b1]++ : data[b1][k]++
22                              expt ← 1
23                       }
24                       if b2 = 3 {
25                              rc=0 ? data[k][b2]++ : data[b2][k]++
26                              expt ← 1
27                       }
28              }
29           }
30           if expt = 0 {
31                NATURAL b ← add = 3 ? 0 : add - 4
32              if b = 0 AND b = 3
33                       rc=0 ? data[k][b]++ : data[b][k]++
34           }
35        }
36     }
37  }
38  for j ← 0 to 3 {
39     for k ← 0 to 3 {
40        if data[j][k] = 2
41           decWord[j][k] ← !decWord[j][k]
42     }
43  }
```

AlgDE has as inputs the EAr and EAc integer vectors (Equations 5.45 and 5.46), the Boolean vectors DEr and DEc, and the tab matrix, which is constructed from the logic described in Table 10, as illustrated in Figure 38. Additionally, the algorithm can read/write from/in the decWord matrix that contains the data in the LPC format.

AlgDE starts by zeroing the data matrix, which has the same size as the LPC data area; the redundancy bits are not part of this matrix, as only the data region is double-checked.

The outermost loop of the first step controls the correction of the DEs detected in the rows ($rc = 0$) and, later, in the columns ($rc = 1$), using the variable rc in all decisions corresponding to the row or column. The innermost loop runs through the four rows or four columns of the data matrix. The variable vDE informs that rows or columns are analyzed only if there is a DE. In case of an error, the add variable receives the address that identifies the DE in the row or column. The add variable is decremented from 1 to point to the first index of the tab matrix (see the relationship of the Address column to the tab indexes in Figure 38).

Figure 38 – Composition of the tab matrix from Table 10.



The variables b1 and b2 receive the first and second elements of the error tuples that are associated with the second index of the tab matrix; this second level has dimension 3, implying that the loop between lines 11 and 29 is executed three times. The DE pointed in a row is checked with the corresponding columns that must also point to a DE; if so, e1 and/or e2 receive 1. However, this check is not performed when b1 or b2 points to a redundancy bit (i.e., b1 $\geq$ 4 or b2 $\geq$ 4) since LPC does not double-check redundancies.

For any valid error tuple, the lines 21 and 25 increment the positions equivalent to each element of the tuple in the data matrix. The goal is to get the data matrix to indicate the correct DE combination by crossing rows and columns. If there is a DE in the data, the data matrix will have at least one bit incremented twice due to the passage through the rows and columns, resulting in a cell with value 2. The combinations (C1,C2), (C0,C2), and (C0,C1), which are described in lines 3, 5, and 6 of Table 10, respectively, have only check bits. Therefore, these combinations do not change the data matrix; they are placed in the tab matrix only for the logical completeness of the matrix.

AlgDE uses the variable *expt* to catch an exception that occurs when the variable vDE informs that there is a DE in a row/column, but there is no indication of this DE in the corresponding column/row. This exception happens when DE is a combination of the parity bit and a data or check bit. AlgDE considers only the combinations of parity bits and data bits since the code correction is done only in the data area. If the execution of the loop between lines 11 and 29 does not produce at least one DE entry, lines 22 and 26 will not be executed, keeping *expt*

at 0; thus, the algorithm deduces that the DE was related to the parity, causing the data associated with the parity bit to be increased in the data matrix (line 33). Line 31 associates the address of each data with the address obtained in the EAr or EAc vector. In this case, the address of the DE is the same as the address of an SE since the parity bit does not change EAr or EAc; i.e., add equal 3, 5, 6, and 7 means data bits D0, D1, D2 and D3, respectively.

Instead of using an exception mechanism, the tab matrix could have dimensions [7][4][2], making each address have four possible DE combinations instead of just 3. However, the analysis performed within the AlgDE execution showed that the number of errors fixed was higher when using an exception mechanism. By increasing the number of valid combinations, the number of false DEs also increased, reducing the effectiveness of the algorithm.

AlgDE finishes by performing the second step, which makes a nested double loop changing all the positions of the codeword that had a double increment in the data matrix.

Figure 39 to Figure 41 exemplify three scenarios containing several types of DEs with the corresponding syndromes (sC0, sC1, sC2), DE control signals (DEr and DEc), as well as addresses regarding the computed error (EAr and EAc). The lower right rectangle of each figure depicts the structure of the LPC code with the data matrix computation; additionally, the areas referring to the check bits are presented, although they are not present within AlgDE, to illustrate the computed error tuples. For easy viewing, cells with a value of 0 are displayed without content.

Figure 39 – Scenario containing four bitflips that generate four DEs annotated in the row control variables (DEr) and columns (DEc). A rectangle with double edges represents the data matrix.

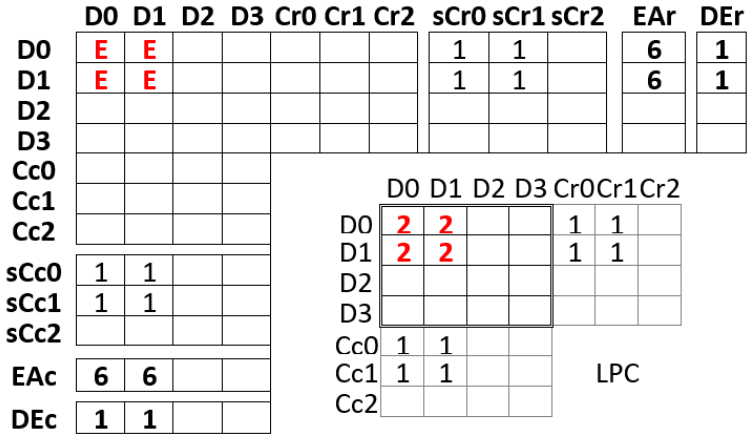


Figure 39 contains only DEs with address 6, indicating that the error tuples are (D0,D1), (D3,C2), and (C0,C1). The execution of AlgDE verifies that the pair (D3,C2) cannot be a source of DE since the column and the row associated with D3 have $vDE = 0$. Therefore, only

Figure 40 – Scenario containing six bitflips that generate six DEs noted in the row (DEr) and column (DEc) control variables.

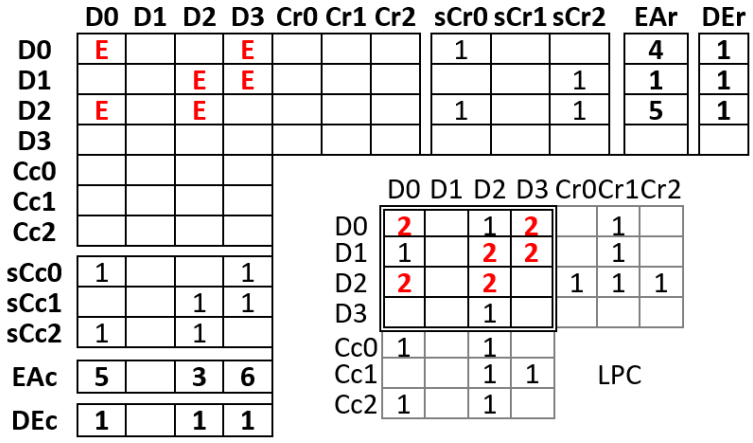
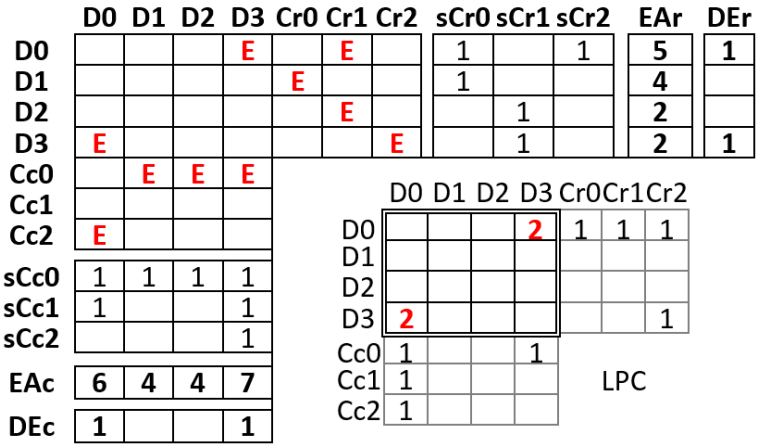


Figure 41 – Scenario containing ten bitflips that generate two DEs annotated in the row control variables (DEr) and columns (DEc), in addition to 4 unique errors not reported to AlgDE.



the tuples (D0,D1) and (C0,C1) could be the source of DE. Since the redundancy bits are not double-checked, only the pair (D0,D1) is incremented in the data matrix. Consequently, at the end of the first step, the data matrix will have only the bits that had an error noted with the value 2, allowing the second step of AlgDE to invert these four bits, correcting the entire data area.

Figure 40 illustrates three columns and three rows with DEs; i.e., only the row associated with D3 and the column associated with D1 are correct. These lines are used to reduce the possibility of valid error tuples. Thus, address 4 for the first line indicates that only the tuples (D0,D3) and (D2,C1) are valid, while the tuple (D1,C2) is not valid. A similar situation occurs with address one on the line, which invalidates the tuple (D1,C0), and with addresses 5, 3 and 6 in the columns, where the tuples (D3,C1), (D3,C0) and (D3,C2), respectively, are not valid. After only increasing the cells referring to the valid tuples, the data matrix has a set of cells with a value of 2, another with a value of 1, and the rest with a value of 0 (cells without content). Again, the final step of AlgDE can correctly invert all the DEs contained in the data, reaching

100% efficiency in correction and errors.

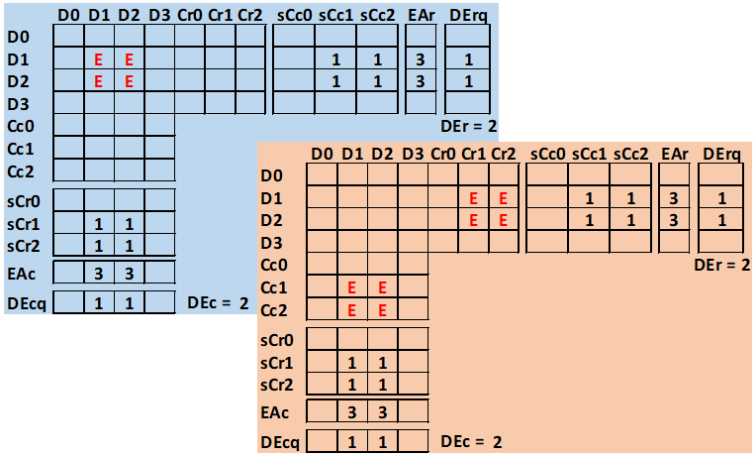
The error scenario of Figure 41 contains DEs in data and redundancy bits, as well as SEs in some redundancy bits. SEs are not reported to AlgDE, so in these cases, the algorithm assumes that rows or columns have no error. This situation cause $EAr = 5$ to refer only to the tuples (D3,C1) and (C0,C2), and $EAr = 2$ to refer only to the tuple (D0,C2). Similarly, $EAc = 6$ points only to the tuples (D3,C2) and (C0,C1), and $EAc = 7$ points only to the tuple (D0,C0). When executing AlgDE, the data matrix will contain only the DEs occurring in the data bits annotated with the value 2. The final step of AlgDE inverts the two corresponding bits in the decWord matrix, making all the data have the correct values; i.e., the algorithm achieves 100% effectiveness in correcting errors.

AlgDE does not have SE information as input; thus, it must be used in conjunction with AlgSE, and the experimental results show that the effectiveness of AlgDE is superior when performed after AlgSE.

The DEs correction technique does not correct redundancy bits, as they can be recalculated from the data. Consequently, whenever the technique can correct 100% of the data, it will achieve 100% effectiveness.

The technique does not guarantee the correction of all DEs for any scenario. Additionally, this technique is subject to anomalous situations; e.g., DEs generated in the check bits can be wrongly calculated as DEs in the data bits. However, these anomalies only exist when the number of errors increases a lot, more precisely with eight or more errors. Figure 42 exemplifies an anomaly case; two error scenarios generate the same syndrome values, preventing AlgDE to infer which are the right error positions; thus, the AlgDE execution corrects the data area, independent of the error scenario.

Figure 42 – Two error scenarios that generate the same syndrome values.



5.6 Summary

This chapter presented the reasons for choosing the proposed ECCs. Then, each one of the codes were detailed through their structures, equations and encoding and decoding processes.

The proposals presented as product codes or modified product codes with 16 data bits were chosen based on the research carried out in the chapter that there is the SLR. The Hamming and parity codebases that were used to compose the proposed ECCs were also chosen based on the large amount of papers that use them.

The first ECC proposed was the PCoSA which is a code in the format (64,16) with 16 data bits in a 4x4 format and 48 redundancy bits divided for the rows (16 bits) and for the columns (32 bits). The coding process was presented through equations. This proposed code is designed to perform 100% correction of 36 error patterns (RAO *et al.*, 2014) in any positions. In the decoding process, all syndromes generated with the positioning of all these error patterns in the code word were analyzed. Then, tables were created that related the syndromes generated with the ways of correcting errors.

This methodology created to design the PCoSA through the use of 36 error patterns was also used in the second proposed ECC, the OPCoSA. This one has the same structure as PCoSA, but without the checkbit checking region. The intention of reducing the codeword by 16 bits is to reduce implementation costs even knowing that there may be a decrease in the correction rate, as it goes from a product code to a modified product code. The decoding process uses tables to find the correction forms, but each one is different from the one obtained with PCoSA.

The third and last ECC proposed in this thesis was the LPC. This code maintains exactly the same structure as OPCoSA, i.e. a modified product code, and maintains the same coding form. In this code, the objective was to keep the same structure as OPCoSA, increasing its correction capacity through innovative techniques. Thus, LPC used a completely different way of decoding. A correction algorithm for SE called AlgSE was proposed that can be used iteratively in addition to a specific correction technique for DE.

The next chapter presents the results obtained for each of the three proposed ECCs.

6 RESULTS

6.1 Introduction

This chapter deals with the presentation of the results of the three ECCs proposed in this thesis: PCoSA, OPCoSA and LPC, through correction test, reliability, hardware synthesis cost and redundancy results.

The first results presented are from PCoSA. Initially, the detection and correction tests up to seven bit flips in an exhaustive way and the reliability compared to other codes used in critical applications are presented. The hardware synthesis process for both encoder and decoder is discussed for all ECCs to know the values of area, power and delay.

For OPCoSA, the codeword format that was used for testing with burst errors is initially presented. Afterwards, the simulation results through exhaustive tests are presented and compared with the PCoSA and four more ECCs. The reliability between OPCoSA and PCoSA is discussed for three failure probabilities. Finally, the results of the hardware synthesis and redundancy costs of the ECCs analyzed are presented and discussed.

The results of the last ECC of this thesis focus more on the correction test simulations, as several test combinations were performed. The LPC correction result is analyzed for the four types of AlgSE correction techniques and for the four heuristics created to control the iterations of the SEs correction algorithm. The correction rate of the four iteration levels of AlgSE with and without the use of AlgDE is also evaluated. The latest simulations are done to evaluate LPC correction rates using COTS and Rad-Hard components; an evaluation is also carried out with exhaustive tests to compare with other ECCs, including PCoSA. The decoder costs of LPC and other ECCs are also discussed.

6.2 Product Code for Space Applications - PCoSA

Figure 43 (a) and (b) shows the detection and correction rates, respectively, for the experimental result that compares PCoSA to Matrix (ARGYRIDES *et al.*, 2007), CLC (CASTRO *et al.*, 2016), Reed Muller (RM)(2,5) (CASTRO *et al.*, 2016; Silva *et al.*, 2018) and PBD (GOERL *et al.*, 2018). Although PCoSA was designed to correct and detect the 36 error patterns illustrated in Figure 27, we performed an exhaustive set of simulations containing all combinations from one to seven bitflips in an 8×8 memory to explore PCoSA's ability to correct

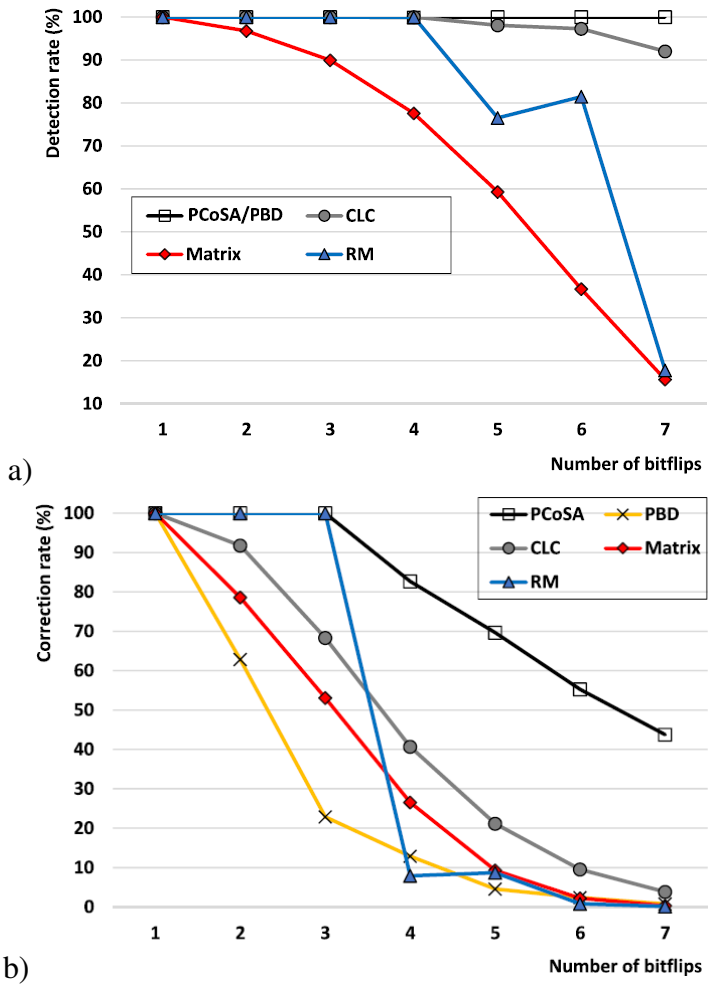


Figure 43 – (a) Detection and (b) correction rates of PCoSA, PBD, CLC, Matrix and RM codes. The simulation is done using all combinations from 1 to 7 bitflips.

more errors. Note that using all error patterns up to 7 bits reduces the error correction capacity of PCoSA but increases the fairness of the results.

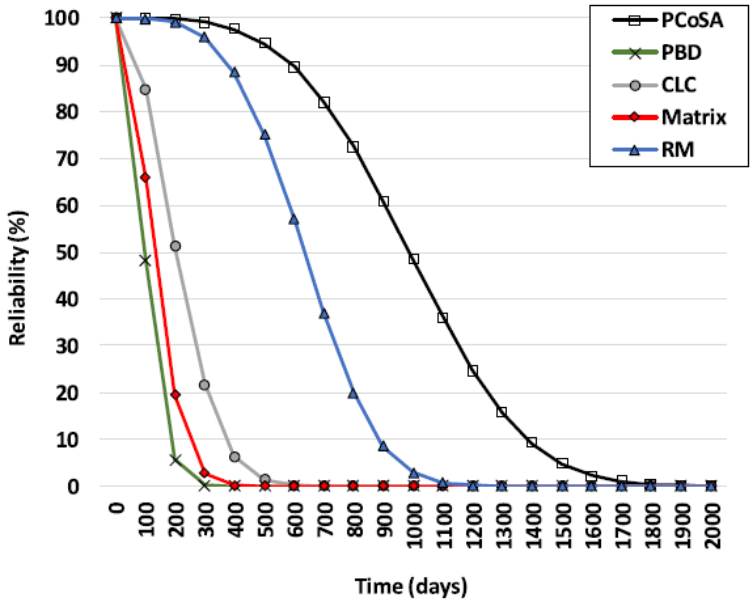
Figure 43 (a) shows that all error cases using PCoSA were detected. This high detection rate happens because PCoSA has a matrix structure with two syndromes (the Hamming-check and parity bits) for each row and column. The PCoSA and PBD codes reach the highest detection rates up to 7 bitflips. PBD does not achieve 100% detection for only in the 4 bitflip cases (in this case, the code reaches 99.9%). This analysis is done in the work of GOERL *et al.* (2018), which explains that some cases of 4 and 8 bitflips are not detected. Matrix has the worst performance, detecting only 16% of cases with 7 bitflips.

Figure 43 (b) demonstrates that the PCoSA and RM codes have 100% correction rates for up to three bitflips. For cases with more bitflips, PCoSA reaches higher correction rates. For example, it reaches 82.7% of correction rate in the experiments with four bitflips; whereas, the PBD code has the lowest correction rates up to three bitflips. Additionally, from four to seven

bitflips, PBD, Matrix, and RM have the lowest correction rates, reaching 0.76%, 0.34%, and 0.16% for 7 bitflips, respectively.

Figure 44 shows the reliability $R(t)$ of the five ECCs regarding correction capacity and a memory with $M = 1000$. The results demonstrate that PCoSA is the most reliable ECC throughout the period, having a much smoother reliability drop curve over time. The reliability of PCoSA is 99.99%, 94.63%, and 48.41% at times 100, 500, and 1000, respectively. On the opposite side is PBD, having an abrupt reliability drop, with $R(t) = 48.13\%$ at time 100 and tending to zero from time 300.

Figure 44 – Reliability of PCoSA, PBD, CLC, Matrix and RM.



Encoding and decoding modules of the five ECCs evaluated in this work were synthesized to analyze their implementation costs. Figure 45 illustrates the encoding and decoding schemes considering various types of memories (i.e., manufacturing technologies, sizes, formats, and protocols) with specific reading and writing drivers to clarify the synthesized modules. It is important to note that while the ECC encoder and decoder modules are only dependent on the ECC algorithms, the driver modules are memory configuration dependent.

Table 11 displays the synthesis results for the encoder and decoder of PCoSA(64,16) and all other evaluated ECCs, considering the encoding and decoding of a 16-bit data; these results encompass area consumption, power dissipation, and delay, which were achieved with the Cadence RTL Compiler software synthesis for 65nm CMOS technology under normal operating conditions.

Independent of the evaluated ECC since most calculations are performed on the

Figure 45 – Flow of encoding and decoding of the five ECCs evaluated, highlighting the modules (in green) that were synthesized.

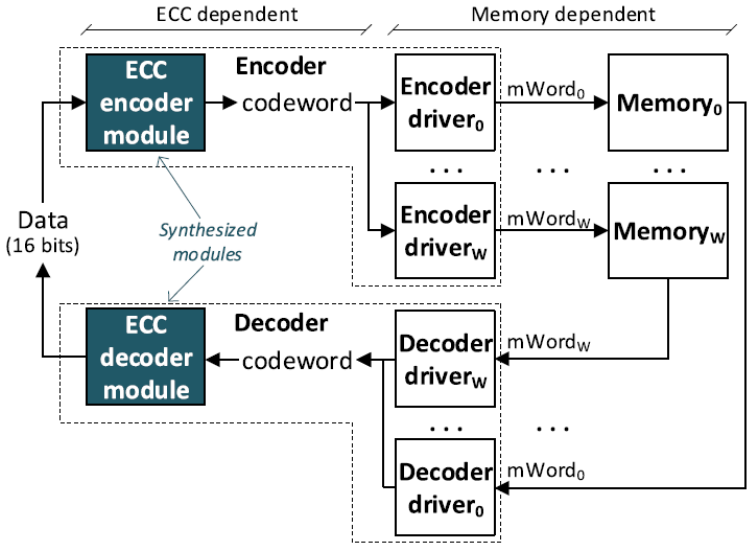


Table 11 – Area consumption, power dissipation, and delay for the encoder and decoder synthesis analysis of all evaluated ECCs.

	Area (um^2)		Power (mW)		Delay (ns)	
	Encoder	Decoder	Encoder	Decoder	Encoder	Decoder
PCoSA	528	10051	0.043	0.848	0.43	1.91
Matrix	298	1090	0.010	0.050	0.15	1.01
CLC	435	1351	0.024	0.076	0.35	1.33
PBD	154	415	0.010	0.031	0.13	0.60
RM	504	4312	0.037	0.737	0.74	2.12

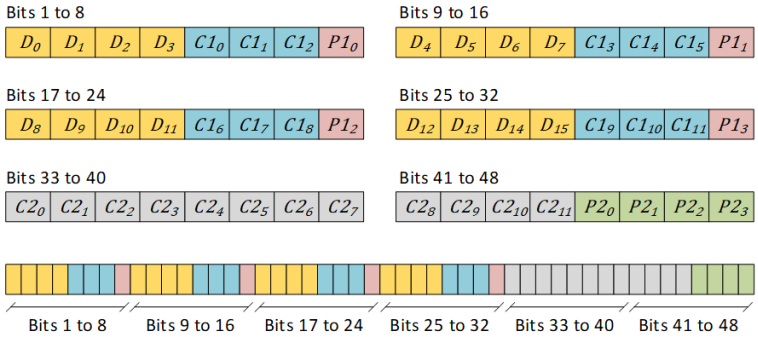
decoder side, it has higher values of area consumption, power dissipation, and delay when compared to the encoder. For example, the area consumption and power dissipation of the PCoSA decoder are about twenty times greater than the encoder, while the encoder delay is four and a half times less than the decoder.

Also, comparing Figure 44 to Table 11 enables us to observe a trade off between reliability and synthesis costs. On the one hand, Figure 44 clearly demonstrates that PCoSA has a great advantage in terms of reliability over other ECCs; on the other hand, Table11 shows that this reliability implies high costs in area consumption and power dissipation, especially when compared to PBD that it is a very low-cost ECC. Finally, the comparison of PCoSA with RM (which is the second most reliable code) shows that PCoSA consumes slightly more than twice the area and dissipates only 15% more power.

6.3 Optimized Product Code for Space Application - OPCoSA

The error correction capacity evaluation, considering burst and exhaustive tests, is done with the vector-type structure presented in Figure 46. OPCoSA can correct 100% of cases until burst error with $l = 4$; however, the ECC has 0% correction for $l > 4$.

Figure 46 – OPCoSA representation in a 48-position vector format.

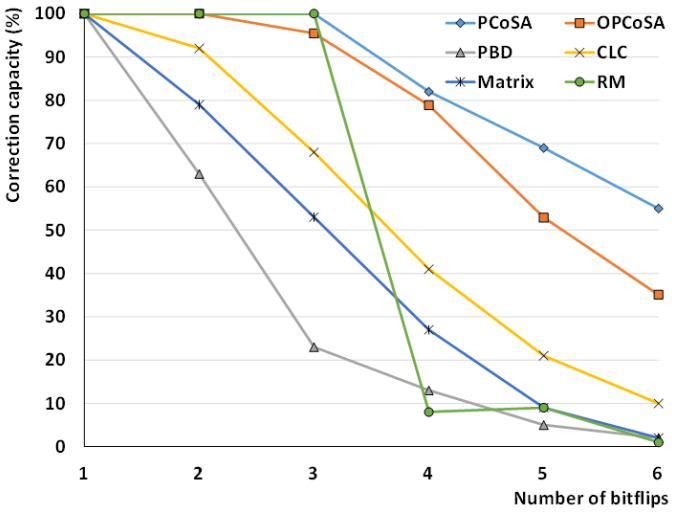


The exhaustive tests were performed with sets of one to six bitflips. Figure 47 shows that only PCoSA and RM have 100% correction up to three bitflips. OPCoSA achieves 100% error correction with two bitflips but reduces it to 95.4% with 3 bitflips. CLC performs better than Matrix and PBD in all range of errors, but these three ECCs show low correction capacity compared to the others from 1 to 3 errors. From four bitflips to six, RM presents a high error correction reduction, which is near to PBD that reaches only 13%, 5%, and 2%. In this same last error range, PCoSA and OPCoSA have rates much higher error correction capacity than the other ECCs; OPCoSA reaches 79%, 53%, and 35% of error correction for 4, 5, and 6 bitflips, respectively.

It is worth noting that the OPCoSA decoding algorithm was designed to correct 100% of the 36 error patterns. δ metric was defined to understand the percentage of these patterns within the set of all possible error scenarios. Additionally, δ_f displays the effectiveness of the decoding algorithm to correct other scenarios with the same f errors.

Let $p_f = (pS_f, pE_f)$ be a tuple containing the start and end numbers of the f error patterns exposed in Figure 27, then $p_1 = (1,1)$, $p_2 = (2,11)$, $p_3 = (12,31)$ and $p_4 = (32,36)$. Let σ_p the be the number of all positions that pattern p can assume within the OPCoSA codeword, q_f be the sum of all σ_p with the same number of f errors, such that $q_f = \sum_{p=pS_f}^{pE_f} \sigma_p$. Let m_f be the total number of combinations with e errors within the OPCoSA codeword; i.e., an exhaustive analysis, such that $m_f = \binom{48}{f}$ (note that 48 is the number of bits of the OPCoSA codeword); thus, $\sigma_f = q_f/m_f \times 100\%$ is the error pattern representativeness within

Figure 47 – Correction capacity of PCoSA, OPCoSA, PBD, CLC, Matrix, and RM. The simulation is done using all combinations from 1 to 6 bitflips.



all possibilities with e errors. For instance, $q_2 = \sum_{p=2}^{11} \sigma_p = 280$, and $m_2 = \binom{48}{2} = 1128$, thus, $\sigma_2 = 280/1128 \times 100\% = 24.82\%$. Table 12 describes δ_f, q_f, and m_f for the range of 1 to 4 errors in Figure 27.

Table 12 – Representativeness of the 36 error patterns concerning the total error combinations.

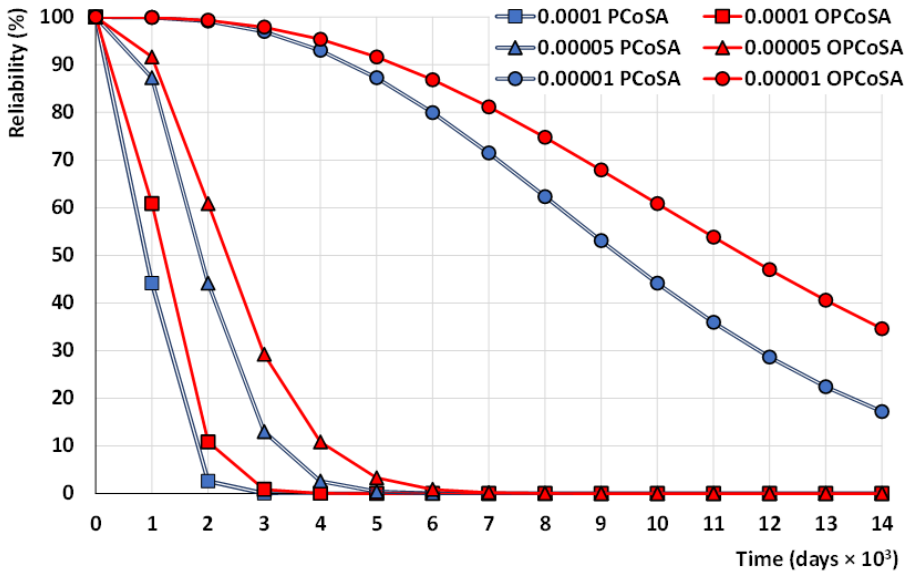
f	q_f	m_f	δ_f	Correction
1	48	48	100.00%	100.0%
2	280	1128	24.82%	100.0%
3	464	17296	2.68%	95.4%
4	122	194580	0.06%	79.0%

On the one hand, the representativeness decrease with the increase in the number of errors is evident and understandable since exhaustive verification consider combinations of errors that leave the 9-cell envelope shown in Figure 27; besides, this growth is proportional to the factorial of the number of errors f. On the other hand, even with low representativeness, the OPCoSA decoding algorithm achieves a high correction rate, reaching 79% of error correction for representativeness of only 0.06% in the case of $f = 4$. The explanation for this high correction efficacy comes from the OPCoSA matrix format, where every row and column has associated a Hamming code that can correct one error and detect two errors. This organization favors error patterns spaced in rows and columns, achieved in the exhaustive exploration, making these errors attended by different Hamming codes. For example, a 4-bit error pattern arranged on the same data row is perceived as four single errors in four data columns. The most complex error patterns that OPCoSA handles are the concentrated ones; therefore, an error pattern evaluation on a 3×3

matrix achieves such a high correction rate.

For reliability calculation, this work uses $M = 1$ (i.e., $R_{ECC}(t)$ is computed regarding a single codeword) to simplify the exhibition of the results. Figure 48 shows $R_{PCoSA}(t)$ and $R_{OPCoSA}(t)$ encompassing three values of λ (1×10^{-4}, 5×10^{-5}, and 1×10^{-5}) and a range of 14000 days. The horizontal axis is time expressed in days, while the vertical axis is the reliability of OPCoSA and PCoSA expressed in %. The reliabilities considering the other ECCs were not included in Figure 48 because FREITAS *et al.* (2020) already shown that PCoSA has a higher reliability than PBD, CLC, Matrix, and RM throughout the same range of days and considering the same values of λ.

Figure 48 – Reliabilities provided by PCoSA and OPCoSA. The reliability regards three values of λ (probability of bit faults per day). The horizontal axis is the time in days, and the vertical axis is the reliability in %



The λ parameter indicates the error incidence rate in memory. For example, $\lambda = 10^{-4}$ indicates the probability of one error in a single bit every 10,000 days; since OPCoSA codeword has 48 bits, OPCoSA has the probability of one bitflip every 208 days. As errors occurrence in $R_{ECC}(t)$ are computed cumulatively, Figure 48 illustrates that in 3000 days, the memory would have 14 bit flips, leading to reliability close to zero for both ECCs.

Figure 48 displays that OPCoSA is more reliable than PCoSA for all periods and values of λ. For instance, with $\lambda = 10^{-5}$, OPCoSA reaches a rate of 100%, 96%, and 74% for days 1, 4000, and 8000, respectively, while for the same days, PCoSA reaches rates 100%, 92%, and 61%.

ECCs were also evaluated in terms of redundancy costs using two criteria: (i) code

redundancy rate, computed by Equation 3.8, and (ii) redundancy rate added in relation to the number of data bits, computed by Equation 3.9.

Table 13 contains the results of *rr* and *ro*, which shows that the lowest redundancy rates are for the Matrix and RM codes; the highest rate is for PCoSA, while OPCoSA is 11.9% above the average for *rr* and 23% above the average for *ro*.

Table 13 – Redundancy rate results.

ECC	*rr*(%)	*ro*(%)
PCoSA(64,16)	75.0	300
OPCoSA(48,16)	66.6	200
PBD(36,16)	55.5	125
CLC(40,16)	60.0	150
Matrix(32,16)	50.0	100
RM(32,16)	50.0	100

Figure 49 displays the costs of the hardware synthesis of the evaluated ECCs, considering area consumption, power dissipation, and delay of encoders and decoders.

The ECC decoder costs are much higher than the encoder ones since most calculations occur in the decoding process. The synthesis results for both encoder and decoder show that PBD, followed by Matrix, is the lowest cost ECC. On the one hand, considering only the encoder synthesis, OPCoSA appears in third place. On the other hand, considering only the decoder, CLC is the third most efficient ECC. Finally, except for the decoding delay, OPCoSA has lower synthesis costs than PCoSA, showing the efficiency of the proposed approach.

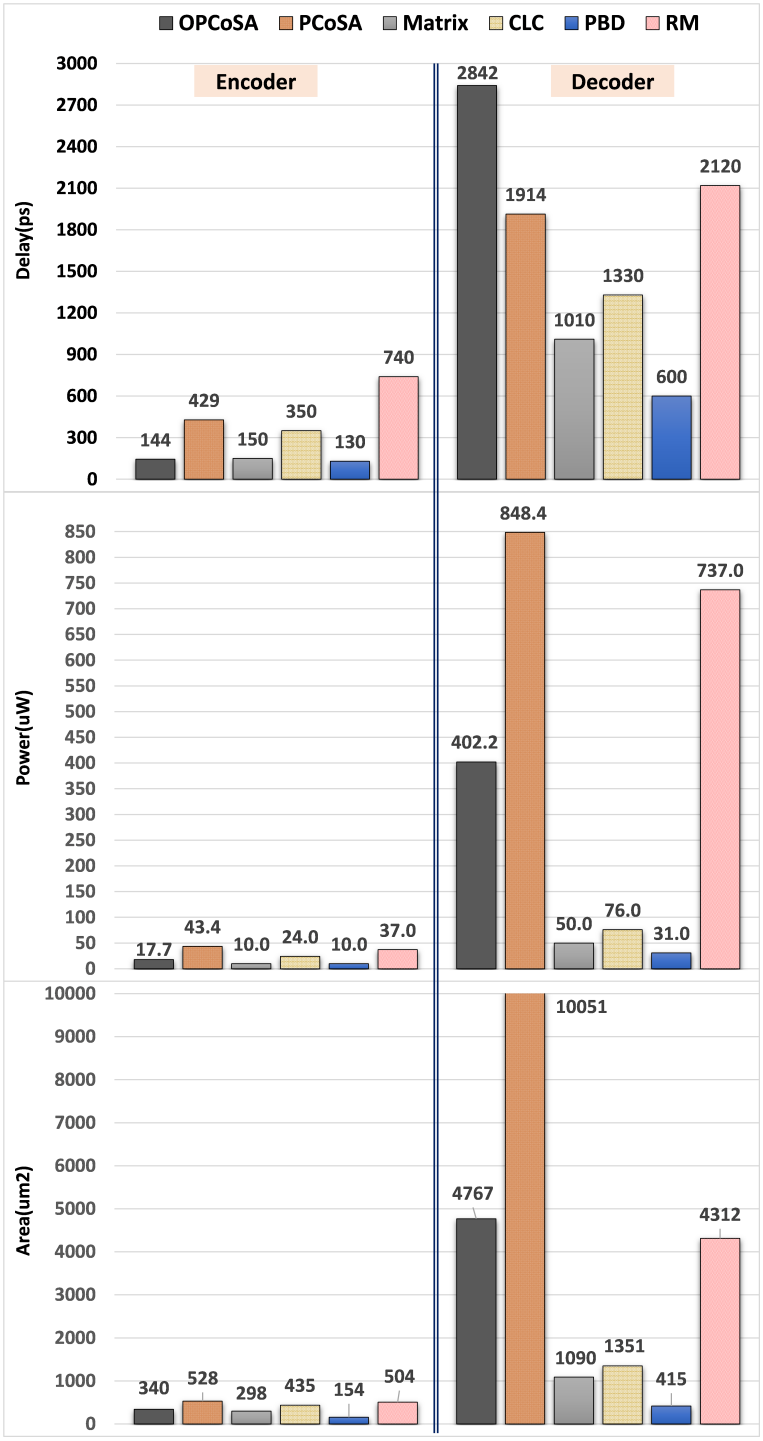
6.4 Line Product Code - LPC

6.4.1 Evaluation of the Error Correction Technique with Row-Column Cross-checking

The LPC implements four error correction techniques applied to AlgSE, which use Ham(8,4) and rows and columns correlation: Data Correction Only (DCO), Data and Redundancy bits Correction (DRC), Data Correction Only with Cross-Check (DCOC) and Data and Redundancy bits Correction with Cross-Check (DRCC). Table 14 summarizes the correction region and whether there is crosscheck verification of all AlgSE correction techniques.

Figure 50 shows the correction capacity of each technique for scenarios from 1 to 10 errors, taking as a reference the correction values obtained with DRCC, which resulted in fewer error corrections for all scenarios.

Figure 49 – Hardware cost of the encoder and decoder of the six ECCs, using Cadence's RTL Compiler synthesis tool for 65nm CMOS technology.

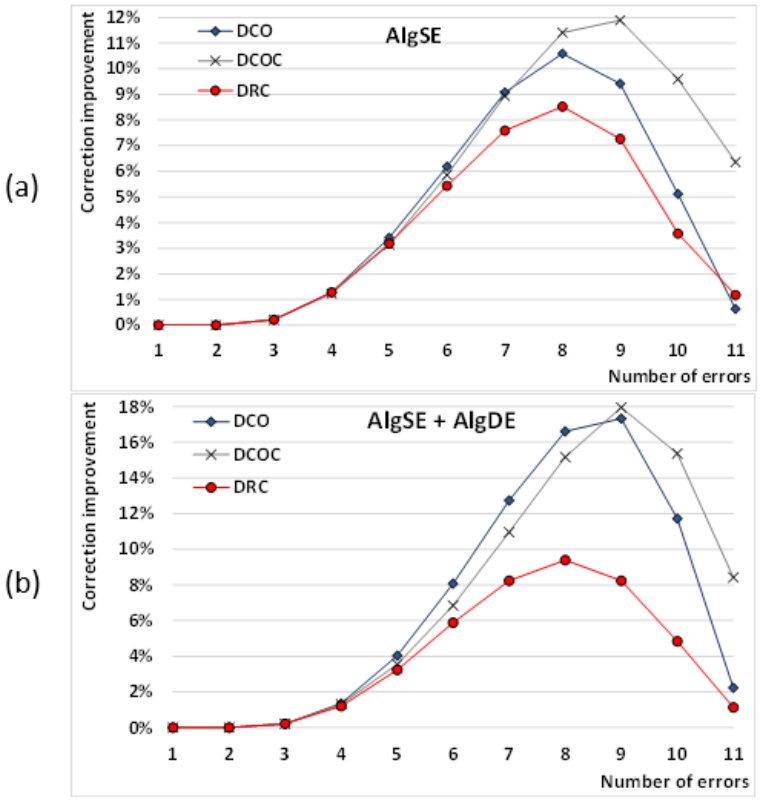


All techniques achieved a 100% error correction rate in scenarios with 1 or 2 bit flips; however, with 3 or more bit flips, the techniques exhibit different correction capabilities. Figure 50(a) shows that DCO, DRC, and DCOC increase the error correction capacity for scenarios of up to 8 errors when compared to the DRCC reference. With 9 error scenarios, only DCOC still shows a growth trend, but the relative error correction capacity tends to reduce from this point

Table 14 – Crosscheck and correction regions of the AlgSE correction techniques.

Technique	Correction		Crosscheck
	Data Bits	Data + Check bits	Verification
DCO	X		
DRC		X	
DCOC	X		X
DRCC		X	X

Figure 50 – Variation of the error correction capacity using the DCO, DCOC, DRC, and DRCC techniques, for the LPC decoder using AlgSE and AlgDE. The values are showing the percentage difference for the worst case (DRCC).



on.

DCO and DCOC provide to AlgSE almost the same efficacy up to 7 error scenarios, with a small advantage for the DCO technique. However, Figure 50(b) shows that when associating AlgDE with the decoder, DCO obtains even more significant results, which are only overcome from scenarios with 9 errors or more. DCO results in greater correction efficacy for AlgSE with up to 7 error scenarios, and this efficacy is enhanced with the application of AlgDE. However, aggregating AlgDE in the decoder produces an anomalous correction behavior; from 8 error scenarios, although DCOC exhibits better results than DCO for AlgSE, the same does not happen in the decoding result when AlgDE is added, suggesting that some corrections made during the AlgSE execution prevented some corrections performed by AlgDE.

The experimental results suggest that LPC is more effective when error corrections are performed on the data, without considering the redundancy bits, due to the matrix format allowing the verification of the data to occur both by columns and by rows. Additionally, the iterative method with the cross-checking technique, which corrects errors only when the equivalent row/column also points out an error, is effective only when the number of errors grows a lot, as in this case, the technique minimizes the possibility of wrong corrections.

6.4.2 Evaluation of the AlgSE Iterative Approach

The next figures presents the results of AlgSE error correction capacity in terms of the correction heuristics and iterative degree. All the experiments performed in this subsection use AlgSE with the DCO technique. This simulation explored several heuristics to determine the most efficacy error correction procedure. In this experiment was described the results of four heuristics: BasicLoop - each iteration first corrects rows then corrects columns (the procedure, starting with columns and then rows, produces the same results due to the symmetry of the code, and the exhaustive evaluation); InvertLoop - in one iteration corrects rows after columns, in the subsequent iteration inverts the order, and so on; PriorityLoop - each iteration corrects only rows or columns, privileging those with the highest number of SEs. This method makes twice as many iterations so that the number of row/column corrections is the same for all methods; FairPriorityLoop - each iteration corrects rows and then corrects columns or vice versa; the number of SEs defines the row/column or column/row order.

Figure 51(a) and (b) show the relative error correction capacity of each heuristic, considering only AlgSE and joint effect of applying AlgDE, respectively. The experiment shows that the FairPriorityLoop heuristic is the most effective for all evaluated error scenarios. The BasicLoop heuristic is less efficacious than FairPriorityLoop, but much higher than the other heuristics. We adopted the FairPriorityLoop heuristic for the execution of all other experiments due to the results obtained in this experiment.

Next, the iterative effectiveness of error correction for the LPC was evaluated. Table 15 presents the correction percentage for scenarios from 1 to 10 errors, with AlgSE performing from one to four loops; the experiment uses the DCO technique with the FairPriorityLoop heuristic.

The results display that, regardless of the number of loops, AlgSE obtains the same correction efficacy for up to three bit flips. For scenarios with four and five bit flips, the second

Figure 51 – Comparison of four heuristics used to control AlgSE iterations. The values presented are normalized according to the least efficacy heuristic for each error scenario.

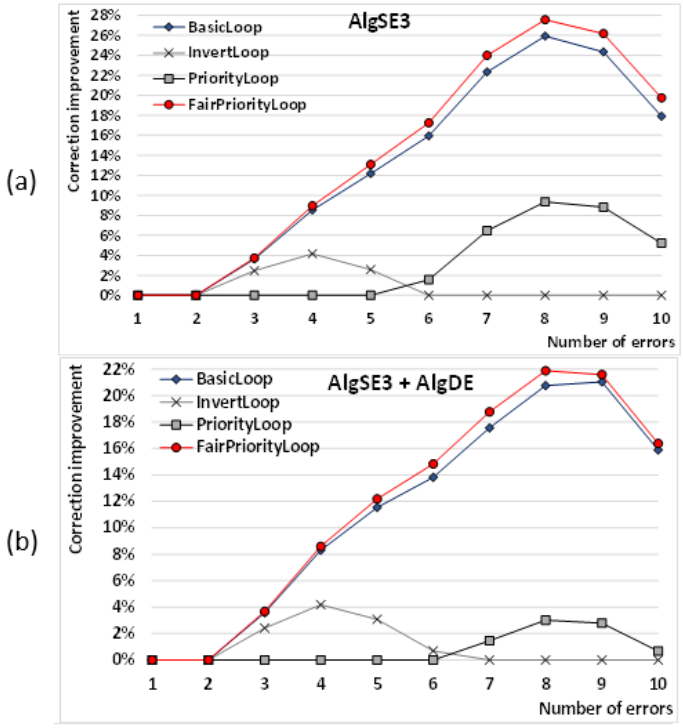


Table 15 – Error correction percentage considering the AlgSE iterative procedure and scenarios from 1 to 10 errors.

Errors	1	2	3	4	5	6	7	8	9	10
AlgSE0	100	100	98.52	92.31	79.94	62.46	43.07	25.97	13.6816	6.34434
AlgSE1	100	100	98.52	93.83	84.15	68.81	49.69	30.73	16.0830	7.15057
AlgSE2	100	100	98.52	93.83	84.15	68.91	49.93	30.97	16.1790	7.16654
AlgSE3	100	100	98.52	93.83	84.15	68.91	49.93	30.98	16.1793	7.16653

iteration level increases the correction efficacy significantly. With six and seven bit flips, a third iterative level allows to increase the number of corrected errors, and from scenarios with eight bit flips, a fourth iterative level is needed. However, the gains achieved become percentage smaller as the number of errors in the scenarios grows. Additionally all the iterative levels higher than four achieve the same error correction efficacy. This experiment concludes that AlgSE1 is almost as good as AlgSE3, with much less execution time.

6.4.3 Evaluation of the AlgDE Error Correction Efficacy

Table 16 displays pairs of lines containing error correction values; the first line describes the error correction capacity obtained with AlgSE and the second one, the increase of this capacity when inserting AlgDE. This experiment uses AlgSE with 1 to 4 iterations, DCO technique and FairPriorityLoop heuristic. AlgDE improves the correction capacity for all error

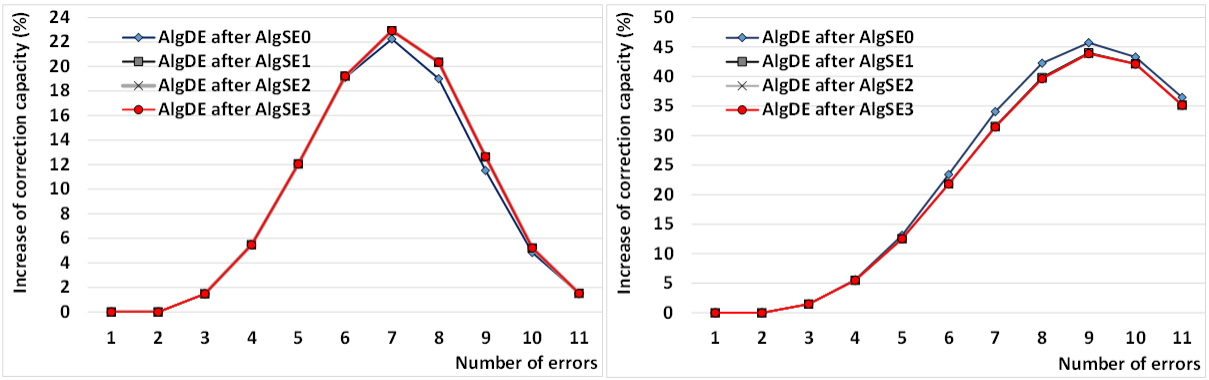
scenarios, allowing LPC to achieve 100% correction for 3 error scenarios; besides, for 4 error scenarios, LPC decoding reaches an efficacy between 97.8% and 99.3%.

Table 16 – Error correction efficacy of AlgSE alone and AlgSE together with AlgDE, considering scenarios from 1 to 11 errors.

Errors	1	2	3	4	5	6	7	8	9	10	11
AlgSE0	100	100	98.52	92.31	79.94	62.46	43.07	25.97	13.68	6.34	2.62
AlgSE0 + AlgDE	100	100	100	97.80	92.01	81.55	65.31	44.97	25.21	11.19	4.13
AlgSE1	100	100	98.52	93.83	84.15	68.81	49.69	30.73	16.08	7.15	2.78
AlgSE1 + AlgDE	100	100	100	99.30	96.22	88.02	72.61	51.07	28.71	12.36	4.29
AlgSE2	100	100	98.52	93.83	84.15	68.91	49.93	30.97	16.18	7.17	2.78
AlgSE2 + AlgDE	100	100	100	99.30	96.22	88.12	72.85	51.32	28.83	12.38	4.29
AlgSE3	100	100	98.52	93.83	84.15	68.91	49.93	30.98	16.18	7.17	2.78
AlgSE3 + AlgDE	100	100	100	99.30	96.22	88.12	72.85	51.32	28.83	12.38	4.29

Figure 52, obtained from Table 16, highlights the ability to correct errors with and without AlgDE. Figure 52(a) shows, in values relative to AlgSE, that the maximum gain of AlgDE occurs with scenarios of 7 errors; e.g., AlgSE3 manages to achieve only 49.93% error correction, and when inserting AlgDE, error correction reaches 72.85%, i.e., a 22.92% increase. Figure 52(b) reveals that when inserting AlgDE the percentage of absolute gain in error correction is increased up to scenarios with 9 errors, and then gradually, the gain is reduced. Due to the high computational cost, we did not extend this assessment to scenarios with more errors; however, it is possible to note that for scenarios with 11 errors, the LPC encoding results in values lower than 5%, not justifying explorations of more aggressive error scenarios.

Figure 52 – The increase in error correction capacity when inserting AlgDE - (a) shows the relative difference between AlgSE and AlgDE; (b) shows the difference in absolute value.



The results of this experiment emphasize that the effectiveness in correcting errors provided by AlgDE is not negligible. Additionally, it allows us to verify that the gains for AlgSE0, which has the least complexity, are higher than for the more complex algorithms (i.e.,

AlgSE1, AlgSE2, and AlgSE3), where there is practically no relative difference.

6.4.4 Data and Redundancy Implementations in Memory

The next simulations assess the effect of errors occurring in the data and redundancy regions separately. The importance of this analysis lies in the possibility of choosing COTS or Rad-Hard memories in critical applications. On the one hand, COTS components in space applications provide state-of-the-art memory technologies that reduce the design and implementation costs compared to Rad-Hard memory costs. On the other hand, although a Rad-Hard memory is not entirely insensitive to radiation, it is much more reliable than a COTS memory (AGNESINA *et al.*, 2019; AGNESINA *et al.*, 2018; ESPOSITO *et al.*, 2015; Shim *et al.*, 2019).

The LPC implementation was proposed in a heterogeneous memory system - a 16-bit memory containing data and a 32-bit memory covering the redundancy bits. The data writing and reading are carried out simultaneously in these memories by an encoder/decoder circuit responsible for synchronizing the information. Therefore, the efficiency of memory technology in data and redundancy regions was explored.

The experiments presented use AlgSE with one iteration, DCO technique, and FairPriorityLoop heuristic.

Figure 53 displays the correction capacity for all combinations with up to 16 bitflips in the data area only, considering both the potential of only applying AlgSE and the joint use of AlgSE with AlgDE. The experiment shows that for scenarios of up to 3 errors, the correction capacity is 100%, and this capacity remains above 90% with 4 and 5 errors; however, the correction efficacy declines dramatically - the error correction capacity is null from 9 errors on.

Figure 54 illustrates the correction capacity for all combinations with up to 32 bitflips, regarding only AlgSE since AlgDE is not applicable in the redundancy area. Although the errors are in the redundancy region, the corrections are applied to the data. Thus, errors evaluated in the redundancy area that generate false errors in the data area, implying wrong corrections that modify the data.

The correction capacity is maintained above 90% up to 6 bitflips, decreasing smoothly until zero with 16 bitflips. From 16 to 31 bitflips, the correction capacity is less than 10%, with less than 3%, on average. Finally, when the entire redundancy area is in error (i.e., 32 upsets), AlgSE reaches 100% correction because when inverting all redundancy bits, there is no correction made in the data area.

Figure 53 – Error correction capacity of AlgSE alone and combined with AlgDE, for errors affecting only the data region.

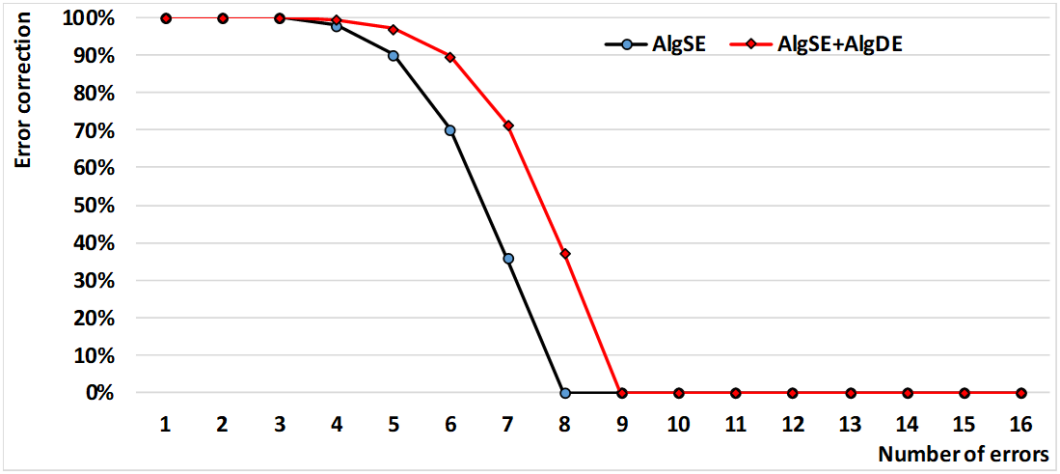
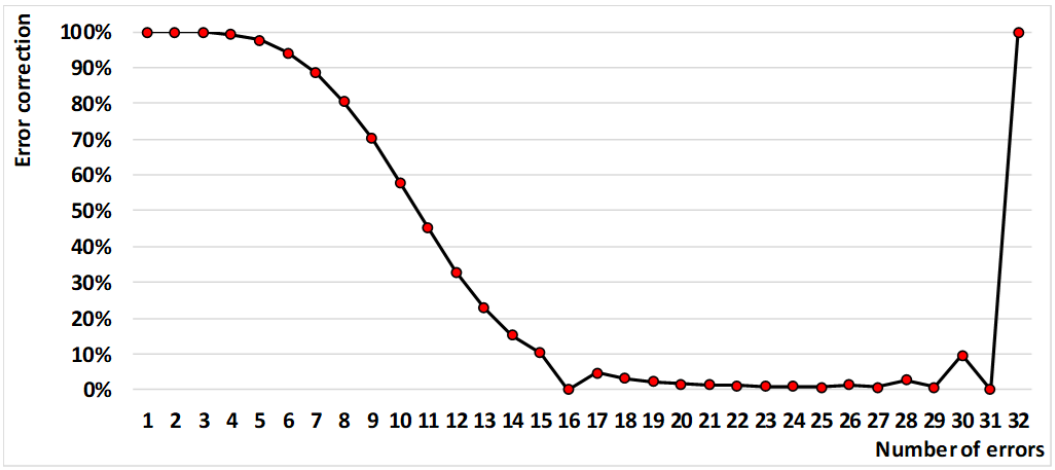


Figure 54 – Error correction capacity of AlgSE with all combinations of errors in the redundancy area.



These experiments show that the data region degrades more quickly than the redundancy region; thus, it's interesting that the physical implementation of the data area be made with a less-sensitive radiation memory, such as a Rad-Hard, and the redundancy area be implemented with a COTS memory.

6.4.5 LPC Encoder and Decoder Syntheses

Table 17 presents the synthesis results of the LPC encoder and AlgSE0, AlgSE1, and AlgDE algorithms used in LPC decoding. All AlgSEs were implemented with DRC and FairPriorityLoop. The results include area consumption, power dissipation, and delay, which were achieved with the Cadence RTL Compiler software for CMOS technology with the 65nm CORE65GPSVT standard cell library under normal operating conditions. The entire hardware implementation was performed with combinational circuits using Verilog language. Only a

subset of the encoding and decoding algorithms are presented, aiming to elucidate the order of magnitude of the synthesis costs.

Table 17 – Analysis of area consumption, power dissipation, and delay for the LPC encoding and decoding algorithms.

LPC Circuit		Area(um^2)	Power(uW)	Delay(ns)
Encoder		340	17	0.14
Decoder	AlgSE0	2724	260	1.94
	AlgSE1	5890	730	2.00
	AlgDE	2218	290	1.64

The LPC encoder has a very low implementation complexity that reflects values in order of magnitude lower than those obtained in decoding. The comparison between AlgSE0 and AlgSE1 shows that the iterative degree more than doubles the area and power values. AlgSE0 and AlgSE1 have critical paths of near delay; however, AlgSE1 requires one more clock cycle for the second algorithm iteration.

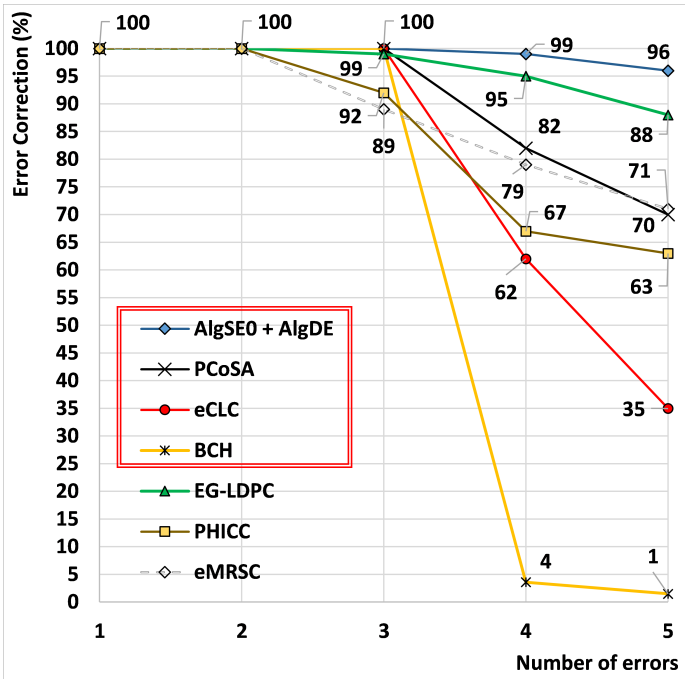
Additionally, AlgDE has an implementation cost close to AlgSE0. Concerning area consumption and power dissipation, the implementation of AlgSE0 associated with AlgDE results in a lower cost than the implementation of AlgSE1 alone. Finally, the combined evaluation of the synthesis information together with the error correction values (Table 17) shows that AlgSE0+AlgDE is more effective and efficient than AlgSE1.

6.4.6 LPC compared to other Space Application ECCs

This last subsection correlates the LPC correction algorithms with other ECC correction methods designed for space applications. Except for BCH code, we used the error correction and synthesis costs provided by the related works; as illustrated by Figure 55, this consideration limited the error correction rates for scenarios with 1 to 5 errors.

Only LPC, BCH and PCoSA were evaluated by exhaustive methods of inserting faults, taking all possible error scenarios into account. The other ECCs considered specific error patterns (e.g., scenarios with only adjacent errors), which drastically reduces the number of scenarios to be assessed. This fact privileges designing high-effective algorithms with low implementation costs, not allowing a fair comparison among ECCs. AlgSE0+AlgDE combination was chosen to represent the LPC decoding algorithms, a highly effective combination that does not penalize the synthesis cost excessively. Figure 55 reveals that although the algorithms used in PCoSA and LPC focus on correcting any error pattern, they reach the highest correction rates.

Figure 55 – Analysis of the error correction capacity of 7 ECCs; the red double-bordered rectangle surrounds ECCs that achieve 100% correction with up to three errors.



Also, the union of AlgSE with AlgDE makes LPC have a correction capacity superior to PCoSA. On the one hand, BCH can correct all 3-error patterns, whereas there are 3-error patterns where EG-LDPC cannot correct all bitflips. On the other hand, the correction capacity of EG-LDPC degrades very slowly with the increase in the number of errors, but the BCH quickly reduces the correction capacity from three errors.

Table 18 presents the synthesis costs of the five ECCs for the same 65nm CMOS technology. Additionally, the last column of Table 18 shows the redundancy rate (RR) for each code, given the number of redundancy bits compared to the total number of bits in the codeword.

Table 18 – Synthesis costs and redundancy rate of five ECCs

ECC	Area		Power		Delay		RR(%)
	(um^2)	(%)	(mW)	(%)	(ns)	(%)	
eMRSC	1,709	17	0.374	44.1	1.54	80.6	50
PHICC	1,761	17.5	0.344	40.6	0.96	50.3	60
eCLC	3,360	33.4	0.331	39	2.50	130.9	60
AlgSE0 + AlgDE	4,492	49.2	0.550	64.9	3.58	187.4	67
PCoSA	10.051	100	0.848	100	1.91	100	75

Table 18 shows that LPC surpasses PCoSA in almost all the items evaluated in this work, except for the delay caused by the execution of two algorithms in series. However, the low complexity of the other ECC algorithms implies gains in all synthesis aspects compared to AlgSE0+AlgDE.

6.5 Summary

This chapter presented the results of the three ECCs proposed in this thesis: PCoSA, OPCoSA and LPC. The analyzes were based on correction rates, redundancy, reliability and hardware synthesis costs of each code, comparing to ECCs used in memories during spatial applications.

The first results presented were from PCoSA which is an ECC product-type that uses Hamming and parity on both rows and columns. The error detection and correction capabilities enable using PCoSA in space application memories. The validation of the proposed technique was performed using two sets of simulations. The first set considers 36 patterns, containing double, triple and quadruple errors, which were captured in memory simulation focused on spatial applications. The second set of simulations was exhaustively performed using all possible combinations of one to seven bit flips within an 8×8-bit memory. The results were analyzed and discussed comparing with four other codes (Matrix, CLC, RM and PBD), equally designed for use in space application memories.

In all error cases, adjacent or not, PCoSA presented 100% of detection rate. This capability is due to (i) its matrix format and (ii) the existence of two syndromes for each row and column (Hamming check and parity bit). The other codes presented lower detection rates; the Matrix code achieves the worst performance, detecting only 16% of seven bitflips. PCoSA and RM showed 100% of correction rate up to 3 bitflips. From 4 to 7 bitflips, PCoSA has 82.7%, 69.7%, 55.3% and 43.7% of correction rate, respectively. PBD has the worst correction rates up to 3 bitflips, and from 4 bitflips the lowest rates are for PBD, Matrix and RM; For 7 bitflips these codes have 0.76%, 0.34% and 0.16% of correction rate, respectively.

OPCoSA was the second analysed ECC, a product code requiring 32-redundancy bits to protect 16-data, which is based on PCoSA that requires more 16-redundancy bits. OPCoSA offers high correction capacity and a consequent decrease in hardware costs in relation to OPCoSA. The experimental results demonstrate that the correction rate up to four bitflips remains like PCoSA and above the other four compared ECCs.

OPCoSA reaches 100% of error correction for 36 specific error patterns and obtained 100% correction for burst errors of sizes one to four. The correction capacity difference between OPCoSA and PCoSA is a maximum of 4.5% for exhaustive error scenarios of up to four bitflips. The great advantage of OPCoSA is that it offers the same functionality as PCoSA, but with 16 bits less redundancy, this directly contributes to the decreased area, power, and delay costs. As

for reliability, three tests were performed varying the number of bit faults per day; in all cases, and for the entire period, OPCoSA has the highest reliability rates.

The last ECC analyzed was the LPC - a code that allows exploring techniques in several axes, attaining high error correction efficacy with low synthesis costs. The LPC decoder implementation with two consecutive algorithms was proposed, which correct single errors (AlgSE) and double errors (AlgDE).

Three research axes with AlgSE were explored: (i) the number of iterations of the algorithm, (ii) the use of a heuristic to choose the row/column or column/row correction order, and (iii) the decision of the region and the way to correct errors. The results showed a little gain from the second iteration, with no gain, observed with more than four iterations. The FairPriorityLoop heuristic, which at each iteration always corrects the row/column pair, giving priority to the one with more SEs, showed greater effectiveness without additional implementation cost. The DCOC technique that checks for errors and corrects only the data has achieved the best effectiveness, in general, standing behind DCO, which does not check for errors, only when the number of errors is between 4 and 8. AlgDE, which is based on the inference of errors by crossing rows and columns, is an innovative algorithm that allows increasing the efficiency of the LPC decoder significantly when used in conjunction with AlgSE. Finally, the comparative results show that when implemented with AlgSE0+AlgDE, the LPC decoder is much more effective than the other ECCs evaluated here, like PCoSA, although the synthesis costs penalize it.

7 CONCLUSIONS AND PERSPECTIVES

The scaling down of the electronic components implies physical changes that make them more susceptible to failures due to radiation. These failures occur when high-energy alpha particles or neutrons collide with a transistor, changing the content of one or more cells permanently or transiently. There are several techniques to mitigate failures in space applications and ECCs are the most used. However, one-dimensional ECCs fail to achieve the effectiveness needed to address the increasing number of bit flips caused by a single radiation event. Consequently, two-dimensional ECCs have been proposed to provide higher error detection and correction power. Thus, this thesis proposes three ECCs in code-product format with high error correction rate with specific decoding algorithms.

This thesis involves a systematic literature review aiming to investigate current 2D-ECCs to mitigate multiple errors in memory systems. Based on the results of this review, this thesis uses the most frequent tests to compare different ECCs, as analysis of error correction rate, reliability, hardware overhead and redundancy. In addition, this review is also used to choose the structure, code-based and data bits length of the three proposed ECCs.

The proposed ECCs are codes with Hamming check bits and parity in both rows and columns. PCoSA is a product code with 16 bits of data and 48 bits of redundancy; whereas OPCoSA and LPC have the same amount of data bits but are modified product codes. PCoSA was designed to perform 100% correction of certain error patterns and then validated for exhaustive errors. OPCoSA was created based on PCoSA to decrease 16 redundancy bits and consequently it has a lower cost of hardware synthesis, even having a lower correction rate. Finally, LPC was designed to increase the correction rate of OPCoSA while maintaining same structure. Thus, two algorithms were created: AlgSE and AlgDE.

The summary of main conclusions of this thesis are:

– In the systematic literature review carried out, five features were highlighted from the 2D-ECCs: (i) 2D-ECC classification; (ii) data size and redundancy metrics; (iii) target application; (iv) analysis methods; and (v) trend on matrix ECCs. In (i), ECCs can be classified according to the encoding approach. (ii) Almost 80% of ECCs use 16 or 32 bits of data and on average 50% of the codeword bits are used for redundancy. (iii) Product codes and extended product codes are most commonly used in space applications. (iv) There are several types of errors analyzed, e.g. adjacent, burst and exhaustive and that 75% of the works use library cells of 45 or 65nm to perform code synthesis. (v) Some

techniques are being used in 2D-ECCs, such as ERT and there are already initial studies of ECCs in more than two dimensions;

- PCoSA was designed to correct certain 36 error patterns in any positions of its codeword and the algorithm achieved this correction capability. A simulation for exhaustive errors was also performed and it was possible to detect 100% of cases due to the matrix format and the existence of two syndromes for each row and column. The error correction rate of up to seven bit flips was the highest, as was the reliability throughout the analyzed period. On the other hand, this implies high costs in area consumption and power dissipation;
- OPCoSA was a proposal designed to reduce the costs of hardware synthesis and the amount of redundancy bits of PCoSA. This ECC was organized with 16 bits redundancy less and it was designed using the same idea as PCoSA. It achieved 100% correction for the 36 error patterns it was designed for and also it achieved high correction rates for the exhaustive testing.Up to four bit flips, the correction results have a maximum difference of 4.5% from the PCoSA results. The best advantage of OPCoSA is that it has a high correction rate, better reliability and lower cost than PCoSA;
- LPC was explored through the simple error correction algorithm in several axes: (i) number of iterations of the correction algorithm, (ii) use of heuristics to choose whether the correction starts with the row or the column and (iii) use of different regions to correct errors. In (i), the results show that the gain from the second iteration is very low and that no gain is observed with more than four iterations. (ii) The heuristic that initiates the decoding process based on the amount of SEs has the best effectiveness without additional hardware synthesis cost. (iii) The technique that checks errors only in the data region has the best effectiveness;
- LPC was designed to improve the double error correction rate and its algorithm was based on error inference through crossing rows and columns. The results show that the use of this technique with the simple error correction algorithm increases the error correction capability. Using AlgSE and AlgDE together increases the correction rate compared to PCoSA, and consequently to OPCoSA, but the synthesis cost is penalized.

In general, it was noticed that the amount of redundancy bits and the use of heuristics in the decoding algorithm favor a greater capacity for error correction, but these factors always penalize the area, power and delay values of the system. It is noteworthy that the choice of the ECC to be used depends on the applicability and that this choice must always consider the

correction capability and hardware overhead.

In the following, the main directions for future work are presented:

- Reduce the PCoSA and OPCoSA algorithm;
- Design PCoSA and OPCoSA for other error patterns;
- Review the most current adjacent error patterns;
- Develop a heuristic to correct triple errors.

The first topic refers to the problem of reducing the synthesis cost of the PCoSA and OPCoSA algorithms. It is a fact that both achieve high correction rates, but the intention of this reduction is to improve the algorithm itself, keeping the same error correction capabilities. The focus of this topic should be the restructuring of correction methods for each of the syndromes.

The second topic is the investigation of the PCoSA and OPCoSA design for error patterns different from the 36 initially presented. The correction rate of both codes for any errors based on the design for the mitigation of 36 error patterns is already known. The fact is that it is not known whether designing both ECCs for a reduced or even extended number of error patterns considerably affects the correctness of the code.

The third point that will help future work in this area of fault tolerance is to know what are the most current error patterns that occur in memory devices. Several works test with exhaustive patterns, others with adjacent and others with specific patterns, however, the error formats vary a lot depending on the technology being used, for example 65nm, 45nm, 22nm and so on. The intent of this study is to draw up a study of current work and list the most common error formats and their respective technology so that subsequent ECCs can be fairly designed and compared.

The fourth and last topic deals with the elaboration of a heuristic to correct triple errors for LPC code. LPC already has an algorithm for simple error correction, called AlgSE, and it already has one for double error correction, AlgDE. The intention of this topic is to create the triple error correction algorithm (AlgTE) and use it together with AlgSE and AlgDE and perform an analysis to know if there is an improvement in the error correction rate, even knowing that the synthesis cost will be penalized.

This thesis gives its contribution focused on 2D-ECCs. However, as the reader may have noticed, there are other problems to be investigated in decoding algorithms. As a future perspective of this work, we intend to deeply investigate the four aforementioned topics, since they are a natural continuation of this work.

BIBLIOGRAPHY

AFRIN, R.; SADI, M. S. An efficient approach to enhance memory reliability. In: **2017 4th International Conference on Advances in Electrical Engineering (ICAEE)**. [*S. l.*: *s. n.*], 2017. p. 170–175.

AGNESINA, A.; SIDANA, A.; YAMAGUCHI, J.; KRUTZIK, C.; CARSON, J.; YANG-SCHARLOTTA, J.; LIM, S. K. A novel 3d dram memory cube architecture for space applications. In: **2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)**. [*S. l.*: *s. n.*], 2018. p. 1–6.

AGNESINA, A.; YAMAGUCHI, J.; KRUTZIK, C.; CARSON, J.; YANG-SCHARLOTTA, J.; LIM, S. K. Bringing 3d cots dram memory cubes to space. In: **2019 IEEE Aerospace Conference**. [*S. l.*: *s. n.*], 2019. p. 1–11.

AHILAN, A.; DEEPA, P. Modified decimal matrix codes in fpga configuration memory for multiple bit upsets. In: **2015 International Conference on Computer Communication and Informatics (ICCCI)**. [*S. l.*: *s. n.*], 2015. p. 1–5.

AHILAN, A.; DEEPA, P. Radiation induced multiple bit upset prediction and correction in memories using cost efficient cmc. **Journal of Microelectronics, Electronic Components and Materials**, v. 46, n. 4, p. 257–266, 2016.

AL-SAREM, M.; BOULILA, W.; AL-HARBY, M.; QADIR, J.; ALSAEEDI, A. Deep learning-based rumor detection on microblogging platforms: A systematic review. **IEEE Access**, v. 7, p. 152788–152812, 2019.

Alam, T.; Islam, M. T. A dual-band antenna with dual-circular polarization for nanosatellite payload application. **IEEE Access**, v. 6, p. 78521–78529, 2018.

ALEXANDRE, G. R.; SOARES, J. M.; Pereira Thé, G. A. Systematic review of 3d facial expression recognition methods. **Pattern Recognition**, v. 100, p. 107108, 2020. ISSN 0031-3203. Disponível em: https://www.sciencedirect.com/science/article/pii/S0031320319304091.

ANITHA, B.; JEEVIDHA, B. Low overhead decimal matrix code with dynamic network on chip against multiple cell upsets. In: **2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)**. [*S. l.*: *s. n.*], 2015. p. 1–6.

ARGYRIDES, C.; PRADHAN, D. K.; KOCAK, T. Matrix codes for reliable and cost efficient memory chips. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 19, n. 3, p. 420–428, 2011.

ARGYRIDES, C.; ZARANDI, H. R.; PRADHAN, D. K. Matrix codes: Multiple bit upsets tolerant method for sram memories. In: **22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)**. [*S. l.*: *s. n.*], 2007. p. 340–348.

ARGYRIDES, C. A.; REVIRIEGO, P.; PRADHAN, D. K.; MAESTRO, J. A. Matrix-based codes for adjacent error correction. **IEEE Transactions on Nuclear Science**, v. 57, n. 4, p. 2106–2111, 2010.

ATHIRA, J.; YAMUNA, B. Fpga implementation of an area efficient matrix code with encoder reuse method. In: **2018 International Conference on Communication and Signal Processing (ICCSP)**. [*S. l.*: *s. n.*], 2018. p. 0254–0257.

Atta-ur-Rahman; Qureshi, I. M.; Naseem, M. T. Adaptive resource allocation for ofdm systems using fuzzy rule base system water-filling principle and product codes. In: **2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)**. [*S. l.*: *s. n.*], 2012. p. 805–810.

Babich, F.; Comisso, M.; Cuttin, A.; Marchese, M.; Patrone, F. Nanosatellite-5g integration in the millimeter wave domain: A full top-down approach. **IEEE Transactions on Mobile Computing**, v. 19, n. 2, p. 390–404, 2020.

BAJAJ, A.; SANGWAN, O. P. A systematic literature review of test case prioritization using genetic algorithms. **IEEE Access**, v. 7, p. 126355–126375, 2019.

Benevenuti, F.; Chielle, E.; Tonfat, J.; Tambara, L.; Lima Kastensmidt, F.; Zaffari, C. A.; dos Santos Martins, J. B.; Santos Cupertino Durão, O. Experimental applications on sram-based fpga for the nanosatc-br2 scientific mission. In: **2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [*S. l.*: *s. n.*], 2019. p. 140–146.

BLAHUT, R. E. Algebraic codes for data transmission. In: BLAHUT, R. E. (Ed.). [*S. l.*]: Cambbridge, 2003. p. 1–497.

Botma, P. J.; Barnard, A.; Steyn, W. H. Low cost fault tolerant techniques for nano/pico-satellite applications. In: **2013 Africon**. [*S. l.*: *s. n.*], 2013. p. 1–5.

BRITO, K. d. S.; LIMA, A. A. de; FERREIRA, S. E.; BURéGIO, V. de A.; GARCIA, V. C.; MEIRA, S. R. de L. Evolution of the web of social machines: A systematic review and research challenges. **IEEE Transactions on Computational Social Systems**, v. 7, n. 2, p. 373–388, 2020.

CASTRO, H. d. S.; SILVEIRA, J. A. N. da; COELHO, A. A. P.; SILVA, F. G. A. e; MAGALHAES, P. d. S.; LIMA, O. A. de. A correction code for multiple cells upsets in memory devices for space applications. In: **2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)**. [*S. l.*: *s. n.*], 2016. p. 1–4.

Elias, P. Error-free coding. **Transactions of the IRE Professional Group on Information Theory**, v. 4, n. 4, p. 29–37, 1954.

Erozan, A. T.; Cavus, E. An eg-ldpc based 2-dimensional error correcting code for mitigating mbus of sram memories. In: **FPGAworld'15: Proceedings of the 12th FPGAworld Conference 2015**. [*S. l.*: *s. n.*], 2015. p. 21–26.

ESPOSITO, S.; ALBANESE, C.; ALDERIGHI, M.; CASINI, F.; GIGANTI, L.; ESPOSTI, M. L.; MONTELEONE, C.; VIOLANTE, M. Cots-based high-performance computing for space applications. **IEEE Transactions on Nuclear Science**, v. 62, n. 6, p. 2687–2694, 2015.

FELIZARDO, K. R.; MENDES, E.; KALINOWSKI, M.; SOUZA, E. F.; VIJAYKUMAR, N. L. Using forward snowballing to update systematic reviews in software engineering. In: **Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement**. New York, NY, USA: Association for

Computing Machinery, 2016. (ESEM '16). ISBN 9781450344272. Disponível em: https://doi.org/10.1145/2961111.2962630.

FREITAS, D.; MOTA, D.; GOERL, R.; MARCON, C.; VARGAS, F.; SILVEIRA, J.; MOTA, J. Pcosa: A product error correction code for use in memory devices targeting space applications. **Integration**, v. 74, p. 71–80, 2020. ISSN 0167-9260. Disponível em: https://www.sciencedirect.com/science/article/pii/S0167926019305243.

FREITAS, D. C. C.; MOTA, D.; MARCON, C.; SILVEIRA, J. A. N.; MOTA, J. Lpc: An error correction code for mitigating faults in 3d memories. **IEEE Transactions on Computers**, p. 1–1, 2020.

FREITAS, D. C. C.; MOTA, D.; SIMõES, D.; LOPES, C.; GOERL, R.; MARCON, C.; SILVEIRA, J.; MOTA, J. C. M. Error coverage, reliability and cost analysis of fault tolerance techniques for 32-bit memories used on space missions. In: **2020 21st International Symposium on Quality Electronic Design (ISQED)**. [*S. l.*: *s. n.*], 2020. p. 250–254.

GOERL, R. C.; VILLA, P. R.; POEHLS, L. B.; BEZERRA, E. A.; VARGAS, F. L. An efficient edac approach for handling multiple bit upsets in memory array. **Microelectronics Reliability**, v. 88-90, p. 214–218, 2018. ISSN 0026-2714. 29th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis (ESREF 2018). Disponível em: https://www.sciencedirect.com/science/article/pii/S0026271418306103.

Gonzalez, C. E.; Rojas, C. J.; Bergel, A.; Diaz, M. A. An architecture-tracking approach to evaluate a modular and extensible flight software for cubesat nanosatellites. **IEEE Access**, v. 7, p. 126409–126429, 2019.

GRACIA-MORAN, J.; SAIZ-ADALID, L.-J.; BARAZA-CALVO, J.-C.; GIL, P. Correction of adjacent errors with low redundant matrix error correction codes. In: **2018 Eighth Latin-American Symposium on Dependable Computing (LADC)**. [*S. l.*: *s. n.*], 2018. p. 107–114.

GRACIA-MORáN, J.; SAIZ-ADALID, L. J.; GIL-TOMáS, D.; GIL-VICENTE, P. J. Improving error correction codes for multiple-cell upsets in space applications. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 26, n. 10, p. 2132–2142, 2018.

HAMMING, R. W. Error detecting and error correcting codes. **The Bell System Technical Journal**, v. 29, n. 2, p. 147–160, 1950.

HE, G.; ZHENG, S.; JING, N. A hierarchical scrubbing technique for seu mitigation on sram-based fpgas. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 28, n. 10, p. 2134–2145, 2020.

Herrera-Alzu, I.; Lopez-Vallejo, M. Design techniques for xilinx virtex fpga configuration memory scrubbers. **IEEE Transactions on Nuclear Science**, v. 60, n. 1, p. 376–385, 2013.

KAMATCHI, C. V. S.; THILAGAVATHI, B. Detection and correction of multiple upsets in memories using modified decimal matrix code. **Journal of Computational and Theoretical Nanoscience**, v. 14, p. 1543–1547, 2017. American Scientific Publishers.

KASTENSMIDT, F. L.; CARRO, L.; REIS, R. **Fault-Tolerance Techniques for SRAM-based FPGAs**. [*S. l.*]: Springer, 2006. v. 1.

KONONCHUK, O.; NGUYEN, B. Y. **Silicon-on-insulator (SOI) Technology - Manufacture and Applications**. [*S. l.*]: Elsevier - Woodhead Publishing, 2014. v. 58.

KUMAR, K. N.; REDDY, N. A.; SHANMUKH, P.; VINODHINI, M. Matrix based error detection and correction using minimal parity bits for memories. In: **2020 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)**. [*S. l.*: *s. n.*], 2020. p. 100–104.

Kumar, M.; Digdarsini, D.; Misra, N.; Ram, T. V. S. Seu mitigation of rad-tolerant xilinx fpga using external scrubbing for geostationary mission. In: **2016 IEEE Annual India Conference (INDICON)**. [*S. l.*: *s. n.*], 2016. p. 1–6.

LERAY, J.-L.; BAGGIO, J.; FERLET-CAVROIS, V.; FLAMENT, O. Atmospheric neutron effects in advanced microelectronics, standards and applications. In: **2004 International Conference on Integrated Circuit Design and Technology (IEEE Cat. No.04EX866)**. [*S. l.*: *s. n.*], 2004. p. 311–321.

Li, J.; Reviriego, P.; Xiao, L.; Argyrides, C.; Li, J. Extending 3-bit burst error-correction codes with quadruple adjacent error correction. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 26, n. 2, p. 221–229, 2018.

LI, J.-Q.; XIAO, L.-Y.; GUO, J.; CAO, X.-B. Efficient implementations of multiple bit burst error correction for memories. In: **2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)**. [*S. l.*: *s. n.*], 2018. p. 1–3.

Li, P.; Dang, W.; Qin, T.; Zhang, Z.; Lv, C. A competing risk model of reliability analysis for nand-based ssds in space application. **IEEE Access**, v. 7, p. 23430–23441, 2019.

LIN, S.; COSTELLO, D. J. Error control coding - fundamentals and applications. In: LIN, S.; COSTELLO, D. J. (Ed.). [*S. l.*]: Prentice-Hall, 1983. p. 1–624.

LINDEN, D. van der; HADAR, I. A systematic literature review of applications of the physics of notations. **IEEE Transactions on Software Engineering**, v. 45, n. 8, p. 736–759, 2019.

Liu, S.; Li, J.; Reviriego, P.; Ottavi, M.; Xiao, L. A double error correction code for 32-bit data words with efficent decoding. **IEEE Transactions on Device and Materials Reliability**, v. 18, n. 1, p. 125–127, 2018.

Liu, S.; Reviriego, P.; Xiao, L. Evaluating direct compare for double error-correction codes. **IEEE Transactions on Device and Materials Reliability**, v. 17, n. 4, p. 802–804, 2017.

LIU, S.; XIAO, L.; GUO, J.; MAO, Z. Fault secure encoder and decoder designs for matrix codes. In: **2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)**. [*S. l.*: *s. n.*], 2015. p. 181–185.

LIU, S.; XIAO, L.; LI, J.; ZHOU, Y.; MAO, Z. Low redundancy matrix-based codes for adjacent error correction with parity sharing. In: **2017 18th International Symposium on Quality Electronic Design (ISQED)**. [*S. l.*: *s. n.*], 2017. p. 76–80.

Liu, S.; Xiao, L.; Mao, Z. Extend orthogonal latin square codes for 32-bit data protection in memory applications. **Microelectronics Reliability**, v. 63, p. 278–283, 2016.

MACWILLIAMS, F. J.; SLOANE, N. J. A. The theory of error-correcting codes. In: . [*S. l.*]: North-Holland, 1977. p. 1–777.

MAGALHAES, P.; ALCANTARA, O.; SILVEIRA, J. Phicc: An error correction code for memory devices. In: **2019 32nd Symposium on Integrated Circuits and Systems Design (SBCCI)**. [*S. l.*: *s. n.*], 2019. p. 1–6.

MANDAL, S.; PAUL, R.; SAU, S.; CHAKRABARTI, A.; CHATTOPADHYAY, S. A novel method for soft error mitigation in fpga using modified matrix code. **IEEE Embedded Systems Letters**, v. 8, n. 4, p. 65–68, 2016.

MANOJ, S.; BABU, C. Improved error detection and correction for memory reliability against multiple cell upsets using dmc pmc. In: **2016 IEEE Annual India Conference (INDICON)**. [*S. l.*: *s. n.*], 2016. p. 1–6.

Marchese, M.; Patrone, F. E-cgr: Energy-aware contact graph routing over nanosatellite networks. **IEEE Transactions on Green Communications and Networking**, v. 4, n. 3, p. 890–902, 2020.

Marchese, M.; Patrone, F.; Cello, M. Dtn-based nanosatellite architecture and hot spot selection algorithm for remote areas connection. **IEEE Transactions on Vehicular Technology**, v. 67, n. 1, p. 689–702, 2018.

MOON, T. K. Error correcting coding - mathematical methods and algorithms. In: MOON, T. K. (Ed.). [*S. l.*]: Wiley-Interscience, 2005. p. 1–804.

MOURAO, E.; KALINOWSKI, M.; MURTA, L.; MENDES, E.; WOHLIN, C. Investigating the use of a hybrid search strategy for systematic reviews. In: **2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)**. [*S. l.*: *s. n.*], 2017. p. 193–198.

Mughal, M. R.; Ali, H.; Ali, A.; Praks, J.; Reyneri, L. M. Optimized design and thermal analysis of printed magnetorquer for attitude control of reconfigurable nanosatellites. **IEEE Transactions on Aerospace and Electronic Systems**, v. 56, n. 1, p. 736–747, 2020.

Nagarajan, C.; D'souza, R. G.; Karumuri, S.; Kinger, K. Design of a cubesat computer architecture using cots hardware for terrestrial thermal imaging. In: **2014 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology**. [*S. l.*: *s. n.*], 2014. p. 67–76.

NEELIMA, K.; SUBHAS, C. Efficient adjacent 3d parity error detection and correction codes for embedded memories. In: **2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)**. [*S. l.*: *s. n.*], 2020. p. 1–5.

NICOLAIDIS, M. **Soft Errors in Modern Electronic Systems**. [*S. l.*]: Springer Science, 2011. v. 41.

OGDEN, C.; MASCAGNI, M. The impact of soft error event topography on the reliability of computer memories. **IEEE Transactions on Reliability**, v. 66, n. 4, p. 966–979, 2017.

Pouponnot, A. L. R. Strategic use of see mitigation techniques for the development of the esa microprocessors: past, present, and future. In: **11th IEEE International On-Line Testing Symposium**. [*S. l.*: *s. n.*], 2005. p. 319–323.

PRIYA, M.; VIJAY, M. M. Error detection and correction for sram systems using improved redundant matrix code. In: **2019 International Conference on Recent Advances in Energy-efficient Computing and Communication (ICRAECC)**. [*S. l.*: *s. n.*], 2019. p. 1–8.

PuWeihua. A kind of cots onboard computer fault-tolerant design. In: **2017 Prognostics and System Health Management Conference (PHM-Harbin)**. [*S. l.*: *s. n.*], 2017. p. 1–5.

QUINN, H.; MORGAN, K.; GRAHAM, P.; KRONE, J.; CAFFREY, M. Static proton and heavy ion testing of the xilinx virtex-5 device. In: **2007 IEEE Radiation Effects Data Workshop**. [*S. l.*: *s. n.*], 2007. p. 177–184.

RADAELLI, D.; PUCHNER, H.; WONG, S.; DANIEL, S. Investigation of multi-bit upsets in a 150 nm technology sram device. **IEEE Transactions on Nuclear Science**, v. 52, n. 6, p. 2433–2437, 2005.

RAHA, P.; VINODHINI, M.; MURTY, N. S. Horizontal-vertical parity and diagonal hamming based soft error detection and correction for memories. In: **2017 International Conference on Computer Communication and Informatics (ICCCI)**. [*S. l.*: *s. n.*], 2017. p. 1–5.

RAHMAN, M. S.; SADI, M. S.; AHAMMED, S.; JURJENS, J. Soft error tolerance using horizontal-vertical-double-bit diagonal parity method. In: **2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)**. [*S. l.*: *s. n.*], 2015. p. 1–6.

RAO, P. M.; EBRAHIMI, M.; SEYYEDI, R.; TAHOORI, M. B. Protecting sram-based fpgas against multiple bit upsets using erasure codes. In: **2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)**. [*S. l.*: *s. n.*], 2014. p. 1–6.

ROHDE, T. M.; MARTINS, J. B. dos S. Multi-bit-upset memory using new error correction code methodology. In: **2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)**. [*S. l.*: *s. n.*], 2020. p. 1–4.

SAI, G. M.; AVINASH, K. M.; NAIDU, L. S. G.; ROHITH, M. S.; VINODHINI, M. Diagonal hamming based multi-bit error detection and correction technique for memories. In: **2020 International Conference on Communication and Signal Processing (ICCSP)**. [*S. l.*: *s. n.*], 2020. p. 0746–0750.

Saiz-Adalid, L.; Gracia-Morán, J.; Gil-Tomás, D.; Baraza-Calvo, J. .; Gil-Vicente, P. Ultrafast codes for multiple adjacent error correction and double error detection. **IEEE Access**, v. 7, p. 151131–151143, 2019.

SALEH, A.; SERRANO, J.; PATEL, J. Reliability of scrubbing recovery-techniques for memory systems. **IEEE Transactions on Reliability**, v. 39, n. 1, p. 114–122, 1990.

SATOH, S.; TOSAKA, Y.; WENDER, S. Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on dram's. **IEEE Electron Device Letters**, v. 21, n. 6, p. 310–312, 2000.

Shim, D. E.; Sidana, A. S.; Yamaguchi, J. S.; Krutzik, C.; Nakamura, D.; Lim, S. K. Flashrad: A reliable 3d rad hard flash memory cube utilizing cots for space. In: **2019 IEEE Aerospace Conference**. [*S. l.*: *s. n.*], 2019. p. 1–8.

SILVA, F.; FREITAS, W.; SILVEIRA, J.; LIMA, O.; VARGAS, F.; MARCON, C. An efficient, low-cost ecc approach for critical-application memories. In: **2017 30th Symposium on Integrated Circuits and Systems Design (SBCCI)**. [*S. l.*: *s. n.*], 2017. p. 198–203.

SILVA, F.; FREITAS, W.; SILVEIRA, J.; MARCON, C.; VARGAS, F. Extended matrix region selection code: An ecc for adjacent multiple cell upset in memory arrays. **Microelectronics Reliability**, v. 106, p. 113582, 2020. ISSN 0026-2714. Disponível em: https://www.sciencedirect.com/science/article/pii/S0026271419302835.

SILVA, F.; MUNIZ, A.; SILVEIRA, J.; MARCON, C. Clc-a: An adaptive implementation of the column line code (clc) ecc. In: **2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)**. [*S. l.*: *s. n.*], 2020. p. 1–6.

Silva, F.; Silveira, J.; Silveira, J.; Marcon, C.; Vargas, F.; Lima, O. An extensible code for correcting multiple cell upset in memory arrays. **Journal of Electronic Testing**, v. 34, p. 417–433, 2018.

STODDARD, A.; GRUWELL, A.; ZABRISKIE, P.; WIRTHLIN, M. J. A hybrid approach to fpga configuration scrubbing. **IEEE Transactions on Nuclear Science**, v. 64, n. 1, p. 497–503, 2017.

SUNDARY, M. S.; LOGISVARY, V. Multiple error detection and correction over gf(2m) using novel cross parity code. In: **2016 10th International Conference on Intelligent Systems and Control (ISCO)**. [*S. l.*: *s. n.*], 2016. p. 1–6.

Sánchez-Macián, A.; Reviriego, P.; Tabero, J.; Regadío, A.; Maestro, J. A. Sefi protection for nanosat 16-bit chip onboard computer memories. **IEEE Transactions on Device and Materials Reliability**, v. 17, n. 4, p. 698–707, 2017.

TAMBATKAR, S.; MENON, S. N.; SUDARSHAN, V.; VINODHINI, M.; MURTY, N. S. Error detection and correction in semiconductor memories using 3d parity check code with hamming code. In: **2017 International Conference on Communication and Signal Processing (ICCSP)**. [*S. l.*: *s. n.*], 2017. p. 0974–0978.

U.S.Department of Transportation. **Single Event Effects Mitigation Techniques Report**. Springfield, Virginia, 2016. 48 p.

VARGAS, F.; NICOLAIDIS, M. Seu-tolerant sram design based on current monitoring. In: **Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing**. [*S. l.*: *s. n.*], 1994. p. 106–115.

Villa, P.; Bezerra, E.; Goerl, R.; Poehls, L.; Vargas, F.; Medina, N.; Added, N.; de Aguiar, V.; Macchione, E.; Aguirre, F.; da Silveira, M. Analysis of cots fpga seu-sensitivity to combined effects of conducted-emi and tid. In: **2017 11th International Workshop on the Electromagnetic Compatibility of Integrated Circuits (EMCCompo)**. [*S. l.*: *s. n.*], 2017. p. 27–32.

Villa, P. R. C.; Goerl, R. C.; Vargas, F.; Poehls, L. B.; Medina, N. H.; Added, N.; de Aguiar, V. A. P.; Macchione, E. L. A.; Aguirre, F.; da Silveira, M. A. G.; Bezerra, E. A. Analysis of single-event upsets in a microsemi proasic3e fpga. In: **2017 18th IEEE Latin American Test Symposium (LATS)**. [*S. l.*: *s. n.*], 2017. p. 1–4.

Wang, J.; Zhang, R.; Yuan, J.; Du, X. A 3-d energy-harvesting-aware routing scheme for space nanosatellite networks. **IEEE Internet of Things Journal**, v. 5, n. 4, p. 2729–2740, 2018.

WANG, P. Chapter 9 - single event effects in avionics. In: WANG, P. (Ed.). **Civil Aircraft Electrical Power System Safety Assessment**. Butterworth-Heinemann, 2017. p. 239–258. ISBN 978-0-08-100721-1. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780081007211000091.

WANG, W.-C.; HO, C.-C.; CHANG, Y.-H.; KUO, T.-W.; LIN, P.-H. Scrubbing-aware secure deletion for 3-d nand flash. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 37, n. 11, p. 2790–2801, 2018.

WIRTHLIN, M.; LEE, D.; SWIFT, G.; QUINN, H. A method and case study on identifying physically adjacent multiple-cell upsets using 28-nm, interleaved and secded-protected arrays. **IEEE Transactions on Nuclear Science**, v. 61, n. 6, p. 3080–3087, 2014.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: **Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2014. (EASE '14). ISBN 9781450324762. Disponível em: https://doi.org/10.1145/2601248.2601268.

WOHLIN, C. Second-generation systematic literature studies using snowballing. In: **Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2016. (EASE '16). ISBN 9781450336918. Disponível em: https://doi.org/10.1145/2915970.2916006.

YEDERE, N. K.; PAMULA, V. K. Performance analysis of decimal matrix code and modified decimal matrix code. In: **2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)**. [*S. l.*: *s. n.*], 2016. p. 1–5.

ZARAGOZA, R. H. M. The art of error correcting code. In: ZARAGOZA, R. H. M. (Ed.). [*S. l.*]: John-Wiley Sons, 2006. p. 1–269. ISBN 978-0-470-01558-2.

ZHANG, F.; YAN, J.; MA, L.; LI, Y.; GAO, W. Multi-bit upset mitigation with double matrix codes in memories for space applications. In: **2019 IEEE International Conference on Unmanned Systems and Artificial Intelligence (ICUSAI)**. [*S. l.*: *s. n.*], 2019. p. 146–149.

ZHANG, R.; XIAO, L.; LI, J.; CAO, X.; LI, L. An adjustable and fast error repair scrubbing method based on xilinx essential bits technology for sram-based fpga. **IEEE Transactions on Reliability**, v. 69, n. 2, p. 430–439, 2020.

APPENDIX A – SYNDROME TABLES - PCOSA

This appendix presents six tables that list the error pattern, region of incidence, syndrome, and correction method used to correct the bit flips in the PCoSA code.

Table 19 – $sind_b$ = [0, 1, 1, 1].

Pattern	Region	sind	Correction method
21	D 2D ∪ C1	[0 1 3 3]	Triple error row is obtained with SP1, and bitflips are corrected using SP2
33	3D ∪ C2 2D ∪ C1 ∪ C2	[0 2 3 2]	The bitflip out the triple error row is corrected by selecting the center column using SC2, and the second error row using the lower SP1. After recalculating the syndromes, the error row is known by SP1 and bitflips are changed using SP2

Table 20 – $sind_b$ = [1, 0, 1, 0].

Pattern	Region	sind	Correction method
36	D D ∪ C1 D ∪ C2 D ∪ C1 ∪ C2	[2 0 2 0] [1 0 2 0]	The four bitflips are corrected by referencing the upper left bit of the pattern. This bit is found using the upper SC1 and the leftmost SC2. The position of this reference bit allows us to change the other three bitflips

Table 21 – $sind_b$ = [1, 0, 1, 1].

Pattern	Region	sind	Correction method
2	D D ∪ C1	[1 0 2 2]	The row is obtained using SC1, and bitflips are corrected by knowing the column positions through SP2
5	D D ∪ C1	[1 0 2 2]	
34	D 3D ∪ C1 3D ∪ 3C1 2D ∪ 2C2 D ∪ C1 ∪ 2C2	[2 0 3 2] [1 0 3 2]	Correction algorithm applies Hamming to the leftmost and rightmost columns, causing a double error to remain in the column. Next, the syndromes are recalculated, and the two bitflips are corrected by obtaining the column by SC2 and the rows by SP1

Table 22 – $sind_b$ = [1, 1, 0, 1].

Pattern	Region	sind	Correction method
14	D	[3 3 0 1]	It is a triple error in the column marked
	2D ∪ C2	[2 3 0 1]	SP2; bitflips are corrected by rows using SP1

Table 23 – $sind_b$ = [1, 1, 1, 0].

Pattern	Region	sind	Correction method
3	D	[2 2 1 0]	These patterns contain double errors
	D ∪ C2	[1 2 1 0]	pointed by SC2; bitflips are corrected
4	D	[2 2 1 0]	by rows using SP1
	D ∪ C2	[1 2 1 0]	
35	D	[3 2 2 0]	SC1 points to the first row of the
	2D ∪ 2C1		pattern, which is corrected with
	3D ∪ C2	[2 2 2 0]	Hamming. Next, the syndrome is
	D ∪ 2C1 ∪ C2		recalculated, and Hamming is applied
	D ∪ 3C2	[1 2 2 0]	to the second row (pointed by SCI).
			Finally, the algorithm recalculates the
			syndrome and corrects the remaining
			two bitflips knowing; rows are
			pointed by SPI

Table 24 – $sind_b$ = [1, 1, 1, 1].

T	Pattern	Region	Correction method
1×1	1	-	SC1 and SC2 point to bitflip row and column that is corrected by Hamming
1×2	12, 13	D ∪ C2	All these patterns have a size of 2 x 2 but are defined as 1 x 2 because a bitflip occurs outside region D. The correction algorithm checks the first row of the bitflip through SC1 or SP1, corrects the bitflip that is inside D and then applies Hamming to the first seven columns
	15-20		
	13, 15	D ∪ C1 ∪ C2	
	18, 20		
1×3	21	D	Only error pattern 21 has dimension 1 x 3; the other error patterns are set here because they have bitflips outside of the region D. The correction algorithm checks for columns that have errors using SC2 and applies Hamming.
		D ∪ 2C1	
	27	2D ∪ C2	
		D ∪ C1 ∪ C2	
	28	D ∪ 2C2	
	30	D ∪ 2C2	
	31	2D ∪ C2	
		D ∪ C1 ∪ C2	
2×1	-	2D	The 2 x 1-dimension error only occurs in the second verification set (which considers all combinations of up to 7 bitflips). These cases use Hamming in the first four rows
2×2	6-13, 15-	D	The error correction algorithm first checks if any rows have double errors. If so, it corrects the bitflip whose column is pointed by SP2 and fixes by Hamming each row. If there is no double-column error, Hamming fixes each of the first four rows
	20	D ∪ C1	
	6-11	D ∪ C2	
2×3	32	D	The correction algorithm checks if there are any double errors in the column. If so, it corrects the error using the top row pointed by SP1. Subsequently, the algorithm checks the columns pointed by SP2 and corrects the bitflips applying Hamming. Otherwise, it checks the columns pointed by SP2 and corrects bitflips using Hamming
		3D ∪ C1	
		D ∪ 3C1	
		D ∪ 3C2	
	33	D	
		3D ∪ C1	
		D ∪ 3C1	
		D ∪ 2C1 ∪ C2	
	27, 28	2D ∪ C1	
	30, 31	D ∪ 2C1	
3×1	14	D	This pattern refers to a triple error in the column; so, the error correction algorithm applies Hamming on the first four rows
		D ∪ 2C2	
3×2	22-24	D	Since these are simple line error patterns, the error correction algorithm applies Hamming to the first four rows
		D ∪ C1	
		2D ∪ C2	
		D ∪ 2C2	
	22, 23	D ∪ C1 ∪ C2	
	29	D	
		2D ∪ C1	
		2D ∪ C2	
		D ∪ C1 ∪ C2	
		D ∪ 2C2	
3×3	25, 26	D	They are just simple error patterns in the rows; the error correction algorithm applies Hamming on rows one to four
		2D ∪ C1	
		D ∪ 2C1	
		2D ∪ C2	
		D ∪ C1 ∪ C2	
		D ∪ 2C2	
-	-	-	Default: Applies Hamming to all rows

APPENDIX B – SYNDROME TABLES - OPCOSA

This appendix presents six tables that list the error pattern, region of incidence, syndrome, and correction method used to correct the bit flips in the OPCoSA code.

Table 25 – S_b = [0, 1, 1, 1].

Pattern	S	Correction method
21	[0 1 3 3]	Apply Hamming to all columns and then Hamming to all rows
33	[0 1 3 2]	

Table 26 – S_b = [1, 0, 1, 0].

Pattern	S	Correction method
13, 20	[1 0 1 0]	Invert the intersecting bit between sC1 and sC2
36	[2 0 2 0]	The four bitflips are corrected by referencing the upper
	[2 0 1 0]	left bit of the pattern, which is found using the upper sC1
	[1 0 2 0]	and the leftmost sC2. The position of this reference bit allows us to change the other three bitflips
		Default: Apply Hamming to all rows

Table 27 – S_b = [1, 0, 1, 1].

Pattern	S	Correction method
2, 5, 27, 31	[1 0 2 2]	Apply Hamming to all columns
2, 5	[1 0 1 1]	
13, 15, 18, 20, 34	[1 0 2 1]	Invert the bit indicated by the double error above and by the double error indicated by the column. After that, apply Hamming to all columns
27, 31	[1 0 3 3]	Apply Hamming to all columns
34	[2 0 1 1]	
34	[2 0 3 2]	Invert the bit indicated by the double error above and
34	[2 0 2 1]	by the double error indicated by the column. After
34	[1 0 3 2]	that, apply Hamming to all columns
		Default: Apply Hamming to all columns

Table 28 – S_b = [1, 1, 0, 1].

Pattern	S	Correction method
14	[3 3 0 1]	Apply Hamming to all rows
		Default: Apply Hamming to all rows

Table 29 – S_b = [1, 1, 1, 0].

Pattern	S	Correction method
3, 4, 29	[2 2 1 0]	Apply Hamming to all rows
3, 4	[1 1 1 0]	Apply Hamming to all rows
12, 13	[2 1 1 0]	Invert the bit indicated by the double line error and the
19, 20, 35		leftmost one by the column and then applies Hamming
		to all lines
23, 24, 29	[3 3 1 0]	Apply Hamming to all rows
35	[3 2 2 0]	Invert the bit indicated by the double line error and the
35	[3 2 1 0]	leftmost one by the column and then applies Hamming
35	[2 1 2 0]	to all lines
35	[1 1 2 0]	Apply Hamming to all rows
		Default: Apply Hamming to all rows

Table 30 – S_b = [1, 1, 1, 1].

T	Pattern	S	Correction method
1×1	1	[1 1 1 1]	If the position of the row error is 1 and
	14		Column 7, apply Hamming to all
	21		columns. Otherwise, apply Hamming to
			all rows
1×2	35	[1 1 2 0]	Apply Hamming to all rows
	12, 16, 17, 19,	[1 1 2 1]	
	22-24, 29, 33		
	6-11, 21	[1 1 2 2]	
1×3	32	[1 1 3 3]	Apply Hamming to all columns
	21, 25, 26, 28, 30	[1 1 3 3]	
2×1	15-18, 27,	[2 1 1 1]	Apply Hamming to all columns
	28, 30, 31		
	6-11, 14	[2 2 1 1]	
	32, 33		
2×2	12, 13, 15-20	[2 1 2 1]	Invert the bit indicated by row double
			errors and then apply Hamming to all
			columns
	27, 28 30, 31	[2 1 2 2]	Apply Hamming to all columns
	22-24, 29, 32, 33	[2 2 2 1]	Apply Hamming to all rows
	6-11, 25	[2 2 2 2]	Apply Hamming to all columns
	32	[1 2 2 1]	Apply Hamming to all rows and then to
			all columns
2×3	32, 33	[1 2 3 2]	Invert the two bits indicated by the two
	32, 33	[2 2 3 2]	rows and the double error column and
			then apply Hamming to all columns
	27, 28, 30, 31	[2 1 3 3]	Apply Hamming to all columns
	25, 26	[2 2 3 3]	
3×1	14, 22, 25, 26	[3 3 1 1]	Apply Hamming to all rows
3×2	22-24, 29	[3 3 2 1]	
	25, 26	[3 3 2 2]	
3×3	25, 26c	[3 3 3 3]	
-	-	-	Default: Check r and c (equations 5.29
			and 5.30). If m $\geq$ c, apply Hamming to
			all rows. Otherwise, apply Hamming to all columns.