UNIVERSIDADE FEDERAL DO MARANHÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA AEROESPACIAL
PPGAero

EDEILSON PEREIRA PESTANA

METODOLOGIA PARA DETECÇÃO E CORREÇÃO DE ERROS CAUSADOS POR RADIAÇÃO EM COMPUTADOR DE BORDO DE UM CUBESAT

SÃO LUÍS
2022

EDEILSON PEREIRA PESTANA

METODOLOGIA PARA DETECÇÃO E CORREÇÃO DE ERROS CAUSADOS POR RADIAÇÃO EM COMPUTADOR DE BORDO DE UM CUBESAT

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia Aeroespacial da Universidade Federal Maranhão – UFMA, campus Bacanga, como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Aeroespacial.

Orientador: Prof. Dr. Allan Kardec Duailibe Barros Filho
Coorientador: Prof. Dr. Luis Claudio de Oliveira Silva

SÃO LUÍS
2022

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Pereira Pestana, Edeilson.
Metodologia para Detecção e Correção de Erros Causados por Radiação em Computador de Bordo de um Cubesat / Edeilson Pereira Pestana. - 2022.
56 p.

Coorientador(a): Luis Claudio de Oliveira Silva.
Orientador(a): Allan Kardec Duailibe Barros Filho.
Dissertação (Mestrado) - Programa de Pós-graduação em Engenharia Aeroespacial, Universidade Federal do Maranhão, Universidade Federal do Maranhão, 2022.

1. Código de Hamming. 2. CubeSat. 3. FPGA. I. de Oliveira Silva, Luis Claudio. II. Duailibe Barros Filho, Allan Kardec. III. Título.

EDEILSON PEREIRA PESTANA

METODOLOGIA PARA DETECÇÃO E CORREÇÃO DE ERROS CAUSADOS POR RADIAÇÃO EM COMPUTADOR DE BORDO DE UM CUBESAT

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia Aeroespacial da Universidade Federal Maranhão – UFMA, campus Bacanga, como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Aeroespacial.

Orientador: Prof. Dr. Allan Kardec Duailibe Barros Filho
Coorientador: Prof. Dr. Luis Claudio de Oliveira Silva

Aprovado em: _____ de_____ de _____

_____
Prof. Dr. Allan Kardec Duailibe Barros Filho (Orientador)
Universidade Federal do Maranhão - UFMA

_____
Prof. Dr. Luis Claudio de Oliveira Silva (Coorientador)
Universidade Federal do Maranhão - UFMA

_____
Prof. Dr. Ewaldo Eder Carvalho Santana (Examinador)
Universidade Estadual do Maranhão - UEMA

_____
Prof. Dr. Alex Oliveira Barradas Filho (Examinador)
Universidade Federal do Maranhão - UFMA

RESUMO

O campo dos nanossatélites está em constante evolução e é crescente a demanda por componentes eletrônicos com níveis de confiabilidade adequados a hostilidade do ambiente espacial. Isso gera uma necessidade cada vez maior por sistemas capazes de detectar e corrigir erros induzidos pelos efeitos da radiação em elementos de memória dos satélites. Neste trabalho, é proposto um método para detecção e correção de erros causados por radiação em computador de bordo de um CubSat. O método desenvolvido utiliza o algoritmo do código de Hamming em conjunto com o método do *bit de* paridade, implementado em FPGA (*Field Programmable Gate Array)*, para realizar a detecção e correção de erros causados por eventos em memória RAM (*Random Access Memorý)* de um microcontrolador. O Esquema apresentado possui capacidade para identificar e corrigir erros do tipo SEU (*Single Event Upset)* e detecção de eventos do tipo DEU (*Double Event Upset)*. O algoritmo foi simulado em *software* e ao final do trabalho foi desenvolvido um protótipo de um computador de bordo onde testes de bancada foram realizados. O método desenvolvido foi comparado ao método do código de Hamming clássico para validação dos resultados e ganhos.

Palavras-chave: CubeSat; Código de Hamming; FPGA.

ABSTRACT

The field of nanosatellites is constantly evolving and the demand for electronic components with levels of reliability adequate to the hostility of the space environment is growing. This generates an increasing need for systems capable of detecting and correcting errors induced by the effects of radiation on satellite memory elements. In this work, a method for detecting and correcting errors caused by radiation in a CubeSat onboard computer is proposed. The developed method uses the Hamming code algorithm together with the parity bit method, implemented in FPGA, to carry out the detection and correction of errors caused by events in the RAM memory of a microcontroller. The presented Scheme has the capacity to identify and correct errors of type SEU and detection of events of type DEU. The algorithm was simulated in software and at the end of the work a prototype of an on-board computer was developed where bench tests were performed. The developed method was compared to the classical Hamming code method to validate the results and gains.

Keywords: CubeSat; Hamming code; FPGA.

LISTA DE ÍNDICE

ADCS	*Altitude Determination and Control System*
CI	*Circuito Integrado*
COTS	*Commercial off-the-shelf*
DED	*Double Error Detection*
DEU	*Double Event Upset*
EDAC	*Error Detection and Correction*
EDPEC	*Error Detection and Partial Error Correction*
EMI	*Eletromagnetic Interference*
EO	*Earth Observation*
EPS	*Energy Power Supply*
FEDC	*Full Error Detection and Correction*
FPGA	*Field Programmable Gate Array*
GCR	*Galactic Cosmic Rays*
GPS	*Global Position System*
LEO	*Low Earth Orbit*
MEU	*Multiple Event Upset*
OBC	*Onboard Computer*
PBC	*Parity per Byte and Duplication*
RAD-HARD	*Radiation Hardness*
RAM	*Random Access Memory*
RF	*Radio Frequency*
SAA	*South Atlantic Anomaly*
SECDED	*Single Event Correction and Double Event Detection*
SECSED	*Single Event Correction and Single Event Detection*
SEE	*Single Event Effect*
SEU	*Single Event Upset*
SOC	*System on Chip*
TMR	*Triple Modular Redundancy*
VHDL	*Very High-Speed Integrated Circuit Hardware Description Language*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

É notório o rápido avanço das tecnologias espaciais e muito desse avanço se deve ao crescente interesse de instituições acadêmicas ao redor do mundo em ciência espacial, especialmente no desenvolvimento de nanossatélites.

O padrão CubeSat, elaborado inicialmente em 1999 pelos pesquisadores Jordi Puig-Suari e Bob Twiggs da Universidade Politécnica do Estado da Califórnia (Cal Poly) e da Universidade de Stanford, respectivamente, impulsionaram a pesquisa e o desenvolvimento de nano e pico satélites por parte das universidades do mundo todo. Inicialmente a intenção dos pesquisadores era exatamente a de ajudar as instituições acadêmicas de todo o mundo a exercer atividades práticas de exploração científica do espaço, porém as especificações do CubeSat acabam por se tornar um padrão internacional devido à sua grande aceitação no meio científico e acadêmico (SHUFAN *et al*., 2015).

Um CubeSat é um tipo de nanossatélite utilizado em pesquisa espacial. Um CubeSat é um satélite que possui dimensões de $10x10x10\ cm$ e massa de até $1{,}33\ kg$. Um CubeSat com essas especificações é dito ser de 1U (ou uma unidade), porém é possível desenvolver CubeSats contendo mais de uma unidade (2U, 3U ou mais), somente adicionando unidades.

Os CubeSats são tipicamente construídos utilizando componentes eletrônicos comerciais ou COTS (do inglês *Commercial off-the-shelf)*.

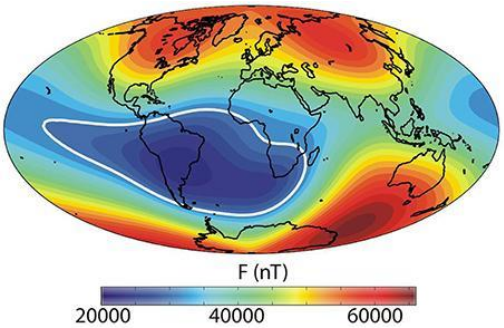
Dentre várias aplicações possíveis para um nanossatélite, a observação da terra, ou EO (do inglês *Earth Observation*) é uma das principais. Os satélites de EO usam sensores de imageamento inteligente para monitorar dados da superfície da terra. Essas informações podem ser usadas para monitorar o desenvolvimento urbano, crescimento de vegetação, desastres naturais. Com a melhoria constante, os dados capturados estão se tornando cada vez mais confidenciais e valiosos, isso resulta em uma necessidade crescente de segurança, privacidade e confiabilidade dos satélites de bordo (BANU *et al,* 2006).

As condições ambientais em que um componente eletrônico funciona podem influenciar seu comportamento e o comportamentos de todo o sistema e equipamentos. O ambiente espacial é inundado com radiação composta de partículas de alta energia do Sol, cinturões de prótons, elétrons e íons coletados do Campo magnético da Terra e raios cósmicos galácticos (GCR do inglês *Galactic Cosmic Rays*) chegando de além do Sistema Solar. A radiação sempre foi um grande problema para os satélites e pode ter efeitos na operação do satélite que variam de sem

importância à falha catastrófica de um subsistema de satélite vital como o computador de bordo (OBC).

A maior parte da corrupção de dados ocorre durante a passagem do nanossatélite por uma região denominada de Anomalia do Atlântico Sul (SAA - *South Atlantic Anomaly).* Anomalia Magnética do Atlântico Sul é uma região onde o campo magnético da terra tem uma intensidade mais baixa, como se sabe o campo magnético da terra funciona como uma barreira de proteção contra radiações oriundas de ventos solares. Como existe essa anomalia, essa região é exposta a níveis de radiação cósmica mais intensas. Uma ilustração da intensidade do campo magnético da terra está apresentada na Figura 1.

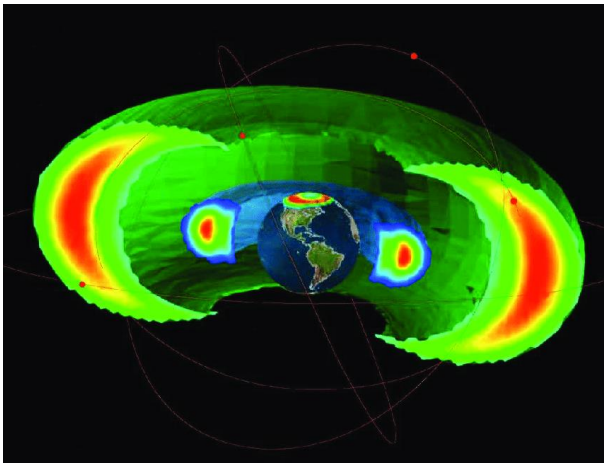
Figura 1 – Região da anomalia do atlântico Sul e campo magnético.



Fonte – Marilia Hagen e Anibal Azevedo, 2020

Além disso, existem duas zonas com grande aglomeração de partículas carregadas, principalmente elétrons e prótons, presas pelo campo magnético. Essas zonas são chamadas de cinturões de radiação de Van Allen, como pode ser visto na ilustração da Figura 2.

Figura 2 – Vista em corte transversal dos cinturões (interno e externo).



Fonte – Wayne A. Scales *et al*, 2009.

A Anomalia do Atlântico Sul (SAA – *South Atlantic Anomaly*) é uma área onde o cinturão de radiação interno de Van Allen tem sua aproximação máxima da superfície da Terra, caindo a uma altitude de 200 quilômetros. Isso leva a um aumento do fluxo de partículas energéticas nesta região e expõe os satélites em órbita a níveis de radiação mais altos do que o normal.

Os chamados *Rad-hard componentes* são componentes eletrônicos que possuem proteção contra radiação, estes oferecem uma eficiente solução para aplicações espaciais, porém o custo desse tipo de componente é elevado, dessa forma a radiação continua a ser uma das principais causadoras de anomalias em satélites já que os componentes COTS geralmente não possuem tolerância à radiação. Dessa forma é importante entender o ambiente espacial para que se possa desenvolver soluções eficientes, para os problemas causados pela radiação, com um custo razoável (BENTOUTOU *et al*, 2015).

De acordo com (GOEL *et al, 2018*) A miniaturização de CI (circuitos integrados) os torna ainda mais susceptíveis a interferências externas, como interferências eletromagnéticas e radiação ionizante.

A incidência de radiação ionizante pode causar um distúrbio classificado como SEE (*Single Event Effects*). Este é um fenômeno causado pela ação de uma simples partícula que atravessa o circuito integrado (BENFICA, 2016).

Em órbita baixa ou LEO (do inglês *Low Earth Orbits*), é conhecido que um dos efeitos adversos da radiação espacial em artefatos espaciais é um erro transiente conhecido como *Single Event Upset* (SEU). Mudanças de estados de *bits* (*Bit-flips*) causados por SEU são um problema conhecido em chips de memória. A aplicação de técnicas utilizando códigos de detecção e correção de erros (EDAC – do inglês *Error Detection and Correction*) têm sido uma eficiente solução para esse problema (BENTOUTOU, 2006, 2012; BENTOUTOU *et al*., 2010; HODGART e TIGGELER, 2000).

Um dos algoritmos mais eficientes para detecção e correção de erros é o código de Hamming que foi introduzido por Richard Hamming em 1950. O código de Hamming é vastamente utilizado em sistemas com detecção e correção de erros e é capaz de realizar a detecção e correção de erros simples (*SECSED* – do inglês *Single Error Correction, Single Error Detection*), isto é, quando há apenas um bit incorreto na informação, porém é possível adaptar o código para ser capaz de realizar a detecção de um erro duplo, porém sem alterar sua capacidade de correção simples (SECDED – do inglês *Sinle Error Correction Double Error Detection*) (SAIZ-ADALID, 2014).

1.1 ESTADO DA ARTE

Em seu trabalho, Dug e Krstic (2018), apresentam e implementam dois métodos para detecção e correção de erros em FPGAs (*Field Programable Gate Array*). Os métodos avaliados foram: detecção de erro e correção parcial de erro – EDPEC (do inglês *Error Detection and Partial Error Correction*) - e detecção completa de erro e correção – FEDC (do inglês *Full Error Detection and Correction*). Ambos os métodos analisados pelos autores utilizam a lógica de detecção de erro (EDC) implementado em FPGA. Os autores ainda apresentam uma proposta de aprimoramento do método FEDC.

Os autores Hillier e Balyan (2019) avaliaram três variações do código de Hamming para aplicação em detecção e correção de erros em sistemas de nanossatélites. Estas variações foram testadas em MATLAB e VHDL (do inglês *Very High Speed Integrated Circuit Hardware Description Language*). A Versão mais eficiente do código foi o Hamming[16, 11, 4]. Esse esquema garante correção de erro simples e detecção de erro duplo (SECDED). Todos os códigos desenvolvidos no trabalho foram otimizados de forma a consumir a menor quantidade de recursos possível, além de serem passiveis de implementação em FPGA, para o qual foram testados exaustivamente com uso de *software* de simulação. A versão final do código foi capaz de proteger 11 *bits* contra SEU além de realizar DED (*Double Error Detection*).

Goerl *et al* (2018) apresentaram em seu trabalho uma abordagem para detecção e correção de erros (EDAC), causados por SEU devido à incidência de radiação ionizante ou interferência eletromagnética (EMI) em dispositivos de memória, denominado de paridade por *byte* e duplicação PBD (do inglês *Parity per Byte and Duplication*), para proteção de dados armazenados em memória. A técnica foi descrita em VHDL em conjunto com o processador LEON3 e um FPGA comercial. Os códigos foram simulados no *Software* ModelSim e resultados obtidos pelos autores demonstram que a abordagem proposta é muito eficiente para detectar e corrigir múltiplos *bit-flips* em matrizes de memória.

Os autores Ciani e Catelani (2014) apresentam uma arquitetura com tolerância a falhas para evitar erros do tipo SEU em dispositivos que usam componentes COTS e são expostos à ambientes extremos. Ciane e Catelani focam o seu trabalho em técnicas de tolerância a falhas em dispositivos aviônicos baseados em FPGA na presença de distúrbios de radiação induzidos por incidência de partículas ionizantes. Após uma explanação sobre os efeitos induzidos em dispositivos eletrônicos pelos fenômenos da radiação os autores avaliam e propõem uma técnica para detecção e correção para erros do tipo SEU. Para atender aos requisitos de um sistema

aviônico complexo, um Painel de Controle Integrado para cabine de aeronave militar é levado em consideração neste sistema. Vários testes específicos também foram realizados a fim de simular condições de perturbação e comprovar a validade da técnica proposta do tipo SECDED.

Burlyaev e Leuken (2014) apresentam, em seu artigo, uma abordagem de simulação estatística para análise de tolerância a falhas de computadores de bordo de satélites (OBCs) baseados em componentes comerciais (COTS). A técnica proposta pelos autores é baseada na modelagem do OBC e na modelagem de falhas com base nos princípios dos *Single Events Upsets* (SEUs). Foi realizado um estudo de caso aplicado à um OBC com um SoC (*System on Chip*) Microsemi SmartFusion que executa o controle de atitude do satélite. Os resultados obtidos da simulação estatística permitiram uma redução de 50% no cabeçalho de *hardware* da técnica de depuração de memória implementada, sem perda de tolerância a falhas. O método revelou desvantagens críticas de tolerância a falhas do projeto inicial do sistema que poderiam ter levado a falha da missão do satélite.

O trabalho realizado por Fauzi *et al* (2017) apresenta uma metodologia para detecção e correção de erros em *bits* utilizando o algoritmo do código de Hamming. A escolha pelo método de Hamming deu-se pelo fato deste ser muito eficiente para detectar e corrigir erros, além de ser de fácil implementação para diversos propósitos em transmissão de dados. Os autores demonstraram que o código de Hamming é capaz de corrigir apenas erro do tipo SEU (erro em um único *bit*) se houver mais de um *bit* alterado na informação o código de Hamming clássico não pode identificar a posição dos *bits* incorretos.

O trabalha desenvolvido por Jung e Choi (2018) apresenta um sistema de processador *on-board* adotando Redundância Modular Tripla TMR (*Triple Modular Redundancy*) com o conceito de janelas de mitigação e depurador externo, além de sugerir um modelo matemático que prevê a taxa de falha do sistema do processador de bordo usando apenas as informações dos recursos de configuração do sistema. O método apresentado é capaz de estimar a confiabilidade do sistema do processador como uma função da taxa de SEUs, o número de janelas de mitigação e a espessura do escudo do processador de bordo. Além disso, uma diretriz do projeto do sistema do processador é fornecida para obter uma boa capacidade de mitigação de SEUs e confiabilidade do sistema.

Os autores Bentoutou e Bensikaddour (2015) apresentaram uma análise estatística da atividade de SEUs (*Single Event Upsets*) para o dispositivo de memória SRAM do microcontrolador do computador de bordo do microssatélite Alsat-1 lançado em órbita LEO (aproximadamente 686 km de altitude) no ano de 2002. Os dados demonstram que durante

eventos solares extremos há um significativo aumento na taxa de SEUs que está correlacionado com o aumento da incidência de prótons de alta energia ($E > 100\ MeV$). Um monitor de nêutrons na estação terrestre foi utilizado para ilustrar a correlação de longo prazo entre raios cósmicos e os distúrbios no Alsat-1. A análise dos dados levou à conclusão que 98,6% desses eventos causam mudança de nível lógico em um único *bit*, 1,22% causa mudança lógica em 2 *bits*, e o restante resulta em vários *bits* com estado lógico alterado, classificados como transtornos graves.

Os autores Tolentino *et al* (2018) propuseram uma técnica alternativa de detecção e correção de erros que pode ser utilizada como código de correção e detecção de erros aprimorado (EEDC). Os resultados dos experimentos realizados no trabalho demostraram que, usando o EEDC proposto, foram alcançadas uma taxa de transmissão de dados mais rápida e um incremento de cabeçalho minimizada quando comparado à códigos CRC e Hamming, cujas desvantagens são o quadro de dados fixo do gerador polinomial e o uso adicional de *bits* de paridade, respectivamente.

No trabalho apresentado por Reviriego *et al* (2012) é apresentado um esquema para realização de correção de erros duplos (DEC – do inglês *double error correction*) utilizando o código de Bose-Chaudhuri-Hocquenghem (BCH). O método proposto garante a detecção de erros simples e duplos além de reduzir a potência consumida pois utiliza apenas um subconjunto dos *bits* da síndrome para detecção de erros. O método apresentado pode ser útil para aplicações em memórias onde a decodificação é interrompida quando nenhum erro é detectado em uma palavra binária para reduzir o consumo de energia pelo sistema.

1.2 OBJETIVOS

Este trabalho possui como principal objetivo propor uma nova metodologia para detecção e correção de erros causados por radiação em computador de bordo de um CuseSat.

Para atingir o objetivo principal, os autores terão que alcançar alguns objetivos específicos listados abaixo.

- Apresentar a fundamentação matemática do método do código de Hamming em conjunto com o método de paridade para realizar detecção e correção de erros,
- Elaborar o algoritmo do código de Hamming e método de paridade em VHDL e implementa-lo em FPGA;

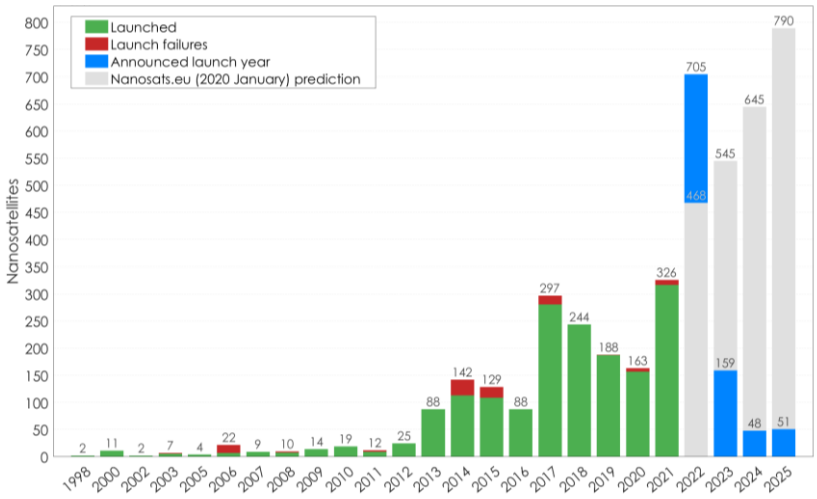
- Implementar a arquitetura de hardware baseada em FPGA e realizar testes que possam validar o método proposto e possível uso em um OBC para cubesat.

1.3 JUSTIFICATIVA

O crescente interesse por parte de instituições de pesquisa, universidades e empresas ao redor do mundo pela exploração espacial, impulsionado pela criação de padrões de nanossatélites, vêm possibilitando o desenvolvimento de novas aplicações para os satélites de pequeno porte a um custo cada vez mais reduzido.

De acordo com Kulu (2020), até o ano de 2021 já foram lançados 1802 nanossatélites ao espaço e há uma previsão de que serão lançados mais 2680 nos próximos quatro anos. Na Figura 3 é possível visualizar algumas informações sobre nanossatélites lançados desde 1998, bem como uma previsão para lançamentos até o ano de 2025.

Figura 3 – Lançamentos de nanossatélites.



Fonte – Kulu, 2020.

Na Figura 3 é possível identificar um padrão de crescimento no número de lançamentos de nanossatélites no decorrer dos anos. Isso demostra que o interesse por esse tipo de tecnologia tem se traduzido em aplicação prática.

O barateamento dos componentes eletrônicos é um importante aspecto a ser notado, pois dessa forma é possível diminuir os custos envolvidos nos projetos e missões utilizando nanossatélites, porém a utilização dos componentes COTS em artefatos espaciais traz consigo

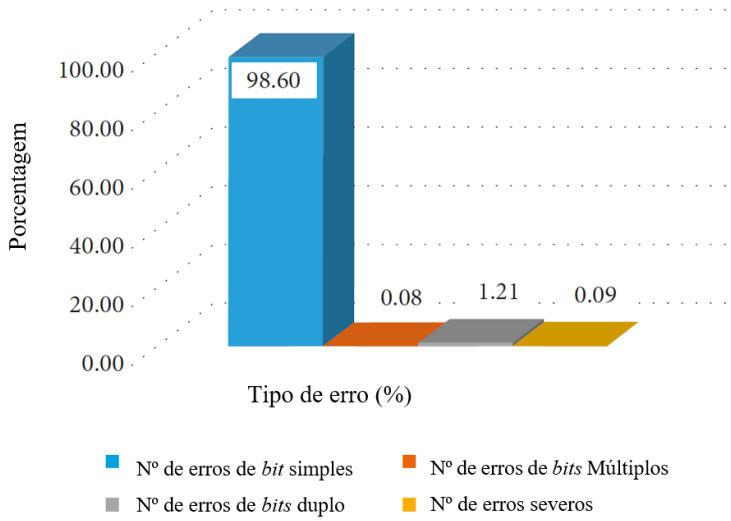
algumas desvantagens e problemas que precisam ser superados, como o fato destes não possuírem tolerância à radiação espacial.

De acordo com Ciani e Catelani (2019) componentes sem tolerância à radiação possuem baixa confiabilidade quando expostos ao ambiente espacial devido à indução de erros do tipo *bit-flips* ocasionados pela radiação incidente nestes. Erros deste tipo alteram o conteúdo das informações armazenadas, principalmente em memórias RAM (*Randon Access Memmory*) o que pode causar falhas catastróficas e inviabilizar a continuação da missão do nanossatélite.

Ciani e Catelani (2019) apresentam em seu trabalho dados sobre erros causados por incidência de radiação coletados pelo nanossatélite Alsat-1 durante sua missão em 2009. Os dados demonstram que, em órbita baixa, 97% dos erros identificados é do tipo SEU, isto é, alteração em apenas um *bit* da informação de memória e que 80% dos erros ocorreram durante a passagem do nanossatélite por uma zona chamada de Anomalia do Atlântico Sul (SAA – do inglês *South Atlantic Anomaly*), que é uma região onde o cinturão interno de Van Allen mais interno se aproxima da superfície da terra e onde há uma alta densidade de partículas carregadas expondo o satélite à níveis mais elevados de radiação.

Na Figura 4, é possível observar um gráfico comparando os três tipos de erros mapeados na missão do Alsat-1.

Figura 4 – Comportamento de erros ocorridos na missão.



Fonte – Adaptado de (Ciani e Catelani, 2019)

Umas das formas de contornar o problema causado pelos efeitos da radiação é a utilização de mecanismos de detecção e correção de erros. Desta forma é possível aumentar a

confiabilidade de todo o sistema e ao mesmo tempo evitar o alto custo envolvido na utilização de componentes com tolerância à radiação (*Rad-Hard*), que podem tornar o projeto proibitivo.

O código de Hamming pode ser implementado para realizar a detecção e correção de erros de *bits* simples, isto é, quando apenas um *bit* é corrompido numa palavra binária de tamanho fixo, entretanto é possível aumentar a robustez do código de Hamming através da utilização deste em conjunto com outras técnicas de detecção e correção de erros para que seja possível realizar detecção de erros duplos, assim os erros podem ser identificados e a informação corrompida pode ser descartada.

O código de Hamming atende aos requisitos de EDAC, possui baixa complexidade de implementação e o custo de *hardware* adicional é mínimo. Um FPGA comercial pode ser utilizada para realizar a implementação do esquema EDAC.

1.4 ORGANIZAÇÃO DO TRABALHO

No capítulo 2 é realizada uma apresentação dos conceitos fundamentais sobre nanossatélites, o padrão CubeSat, seus subsistemas e as condições adversas à que estes são expostos em ambiente espacial.

No capítulo 3 é realizada uma apresentação do código de detecção e correção de erros de Hamming e como o mesmo é desenvolvido de forma matemática. Ainda no capítulo 3 é apresentado o método para detecção e correção de erros desenvolvido para este trabalho, bem como as características do mesmo.

No capítulo 4 são apresentados os materiais escolhidos para a construção do protótipo do OBC para os testes de bancada realizados ao final do trabalho.

No capítulo 5 é apresentado a implementação e análise de alguns testes, bem como a avaliação dos resultados alcançados e demonstração dos ganhos em comparação com o código de Hamming clássico.

As considerações finais são apresentadas no capítulo 6 e após estas segue as referências utilizadas na pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 NANOSSATÉLITES

Nesta sessão serão apresentados alguns conceitos relacionados à nanossatélites com um enfoque no padrão CubeSat, os subsistemas que o compõe e as condições aos quais estes podem ser expostos em ambiente espacial.

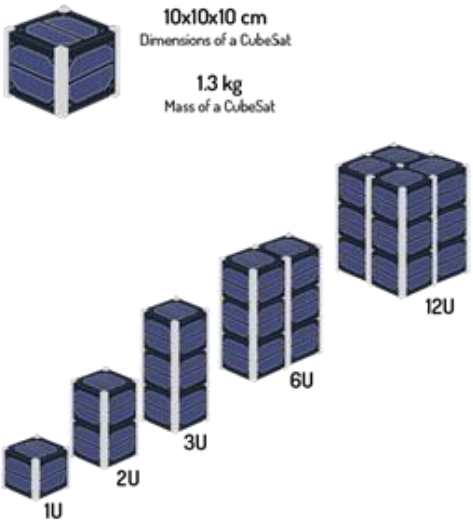
2.1.1 O padrão CubeSat

O padrão CubeSat surgiu do esforço colaborativo dos pesquisadores Jordi Puig-Suari e Bob Twiggs da Universidade Politécnica do Estado da Califórnia (Cal Poly) e da Universidade de Stanford, respectivamente, em 1999 e acabou por impulsionar a pesquisa e o desenvolvimento de nano e pico satélites por parte das universidades do mundo todo. Inicialmente, a intenção dos pesquisadores era a de fornecer um meio acessível para que a comunidade cientifica pudesse desenvolver e exercer atividades práticas de exploração científica do espaço, porém as especificações do CubeSat acabam por se tornar um padrão internacional devido à sua grande aceitação no meio cientifico e acadêmico. Hoje, pequenas universidades e até mesmo escolas podem desenvolver seus próprios CubeSats (SHUFAN et al., 2015).

De acordo com NASA (2017), um dos motivos da grande aceitação do padrão CubeSat é devido custo reduzido do desenvolvimento técnico dos CubeSats, quando comparado com os custos envolvidos em projetos com satélites de grande porte. Isso tem proporcionado um crescimento exponencial da popularidade dos CubeSats desde o seu início.

Um CubeSat é um satélite que possui dimensões de 10x10x10 cm e massa de até 1,33 Kg. Um CubeSat com essas especificações é dito ser de 1U (ou uma unidade). Ainda é possível desenvolver CubeSats contendo mais de uma unidade (2U, 3U ou mais), somente adicionando unidades. Na Figura 5 está apresentado um resumo das especificações e configurações para um CubeSat.

Figura 5 – Configurações de um CubeSat.



Fonte – Alén Space (2019).

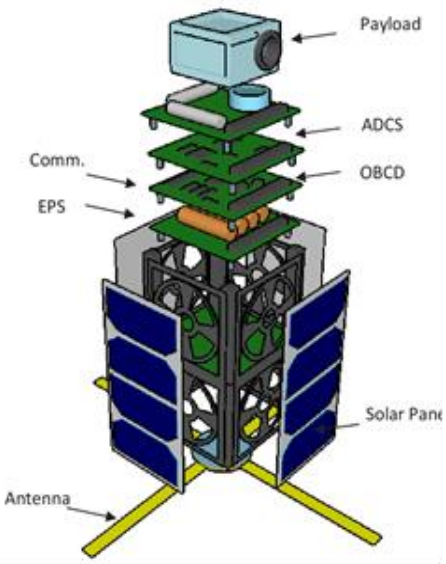
Além da padronização das dimensões físicas como largura, altura e massa, o CubeSat também especifica protocolos de comunicação entre os subsistemas do satélite, todos esses requisitos podem ser atingidos fazendo-se uso dos chamados componentes COTS (do inglês Comercial off-the-shelf) que são componentes eletrônicos de custo reduzidos e encontrados com facilidade no mercado, o que torna possível reduzir o tempo de desenvolvimento e o custo do projeto. A ideia central do padrão CubeSat é que seja possível o desenvolvimento de tecnologias espaciais com custo reduzidos, já que missões com satélites convencionais são extremamente caras e inacessíveis à muitas instituições. A possibilidade de utilização de componentes COTS torna mais acessível o desenvolvimento destas tecnologias, porém devem ser observados os requisitos de confiabilidade do sistema como um todo.

2.1.2 Arquitetura CubeSat e Subsistemas

Um Cubesat é formado por módulos, isto é, é dividido em subsistemas que executam diferentes tarefas. Cada subsistema é responsável por um conjunto de tarefas, porém, todos estes são interdependentes formando o sistema complexo do satélite. Porém alguns desses subsistemas são de grande importância para o funcionamento do satélite e outros variam com o tipo de missão que o mesmo será destinado. São exemplos de subsistemas o computador de bordo (OBC – On-board Computer), o módulo de energia (EPS – *Electrical Power Supply*), o sistema de comunicação por rádio frequência (RF), o controle de atitude (ADCS – *Attitude*

Determination and Control System), a carga útil (*Payload*), etc. Na Figura 6 é possível verificar uma representação dos subsistemas de um CubeSat.

Figura 6 – Subsistemas de um Cubesat.



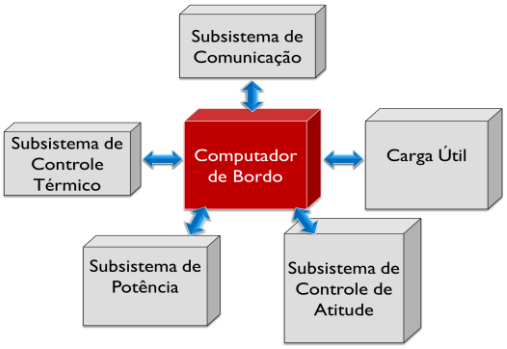
Fonte - Yanes at al, 2018.

2.1.3 Computador de bordo (OBC)

Em um Cubesat, um computador de bordo (OBC - *On-board Computer*) é um sistema computacional, que processa as informações enviadas para o satélite e as informações geradas pelos outros subsistemas deste (RAZZAGHI, 2012). Sendo considerado como o “cérebro” do satélite, pois é ele que gerencia os demais subsistemas e é diretamente responsável pelo sucesso da missão, o OBC deve ser desenvolvido para ser o mais robusto possível, mas obedecendo as restrições de projeto do padrão CubeSat.

Na Figura 7 é possível ver que na estrutura básica de um Cubesat, o computador de bordo é o subsistema central e todos os outros subsistemas dependem do bom funcionamento do OBC.

Figura 7 – Estrutura dos subsistemas de um CubeSat.



Fonte – Elaborado pelo autor.

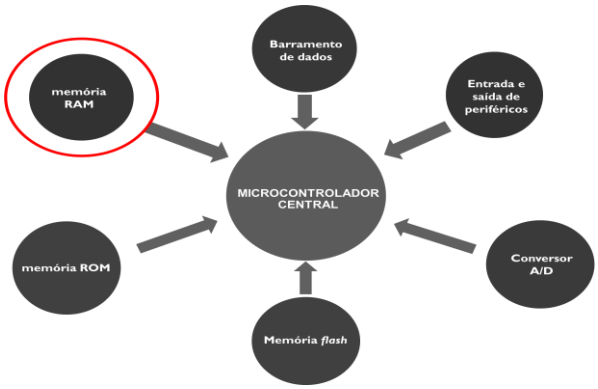
Os principais elementos de um OBC são: microcontrolador central, memória RAM, memória Flash, memória de armazenamento para os dados obtidos, barramento de dados, barramento de controle, entradas de periféricos e *clock*. As principais funções de um OBC são (WELLS G. J.; STRAS, 2003):

1. Monitoramento e armazenamento da telemetria gerada pelo satélite para transmissão futura;
2. Codificação e decodificação dos pacotes oriundos da comunicação com a estação base;
3. Processamento de telecomandos da estação base;
4. Monitoramento dos subsistemas, analisando o seu correto funcionamento e reinicializando o sistema quando necessário.

Logo, o OBC é os subsistemas diretamente responsável pelo sucesso a missão. E deve ser desenvolvido para ser o mais robusto possível.

Observando em mais detalhes o computador de bordo, é possível ver que o mesmo tem uma estrutura como a apresentada na Figura 8.

Figura 8 – Estrutura do computador de bordo



Fonte – Elaborado pelo autor.

Destacada em vermelho na Figura 8, tem-se o dispositivo onde ocorre a maior parte da corrupção de dados devido à incidência de radiação.

2.2 EFEITOS DO AMBIENTE ESPACIAL

Depois que um satélite é lançado e colocado em órbita, suas operações permanecem amplamente autônomas. Durante o voo orbital, ele estará exposto a uma variedade de efeitos mecânicos, térmicos e eletromagnéticos que podem ter um impacto direto em suas operações. Um resumo das perturbações às quais os CubeSats podem ser expostos ao orbitar a Terra é explicado nas seções a seguir.

2.2.1 Ventos e erupções solares

Han e Kim (2006) definem o vento solar como um fluxo constante de gases carregados eletricamente (também conhecido como plasma) e campo magnético que emana do sol, criado quando os gases na atmosfera do sol são aquecidos o suficiente para atingir a velocidade de escape e voar para o sistema solar. Estes ventos podem atingir velocidades superiores a 400km/s e podem ser particularmente devastadores para materiais semicondutores e células fotovoltaicas que estão diretamente expostas a ele.

As erupções solares são um fenômeno que ocorre quando a energia magnética que se acumulou na atmosfera solar é liberada repentinamente. Essa energia causa a emissão de radiações que afetam todo o espectro eletromagnético, desde ondas de rádio na extremidade do comprimento de onda longo, passando pela emissão ótica de raios-X e raios gama na extremidade do comprimento de onda curto. Isso aumenta a ionização da atmosfera superior causando interferência com redes de telecomunicações (GPS, GSM, rádio de ondas curtas), aumentando o calor dentro da camada atmosférica externa e, consequentemente, aumentando o arrasto atmosférico sobre os satélites em LEO (Holman & Benedict, 2007).

2.2.2 Ondas eletromagnéticas no espaço

Durante as tempestades solares, que podem ser imprevisíveis, a atividade das ondas eletromagnéticas é acelerada nas regiões vizinhas do sol e pode ser levada pelas chamas e pelo

vento. De longos comprimentos de onda a gama, raios X, UVs e ondas infravermelhas, os efeitos podem ser devastadores (Hardy, 2009).

2.2.3 Efeitos induzidos por radiação em semicondutor no espaço

A exposição de componentes COTS à radiação e mudanças extremas de temperatura no LEO pode ter inúmeras consequências dentro do CubeSats, tais como:

3 Degradação progressiva de componentes em termos de eficiência;

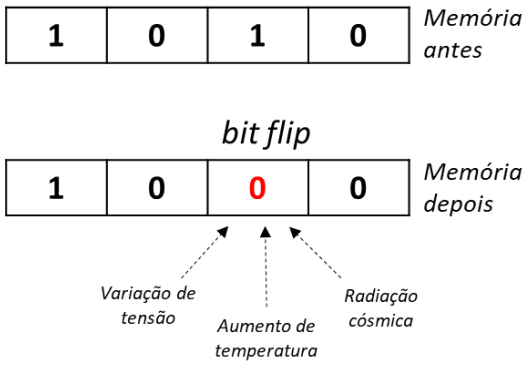
4 Erros de *bit* ou *flips* de *bit* durante a transferência de dados (telemetria e telecomandos);

5 Travas que podem levar a curto-circuitos (Hardy, 2009).

No espaço, o exterior da espaçonave é exposto à temperaturas extremas de cerca de +150 ° C no lado voltado para o sol até -150 ° C no lado voltado para a sombra. Dentro do satélite, as variações de temperatura são menos severas. Uma faixa de temperatura mínima de operação para cada componente eletrônico usado em um CubeSat é geralmente especificada entre + 85 ° C e -40 ° C, que é uma faixa de temperatura industrial (Han & Kim, 2006).

Na Figura 9 pode-se observar uma informação binária de 5 bits que após sofrer o bit-flip, tem o estado logico de um dos seus bits alterado (destacado em vermelho) e isso acarreta em uma informação corrompida.

Figura 9 – Erro em bit.



Fonte – Elaborado pelo autor.

De acordo com a literatura a inversão de *bit* pode ocorrer basicamente por três causa:

- Redução ou variação no nível de tensão aplicado ao dispositivo de memória;
- Aumento de temperatura;
- Ou devido à radiação cósmica.

As duas primeiras causas são facilmente solucionadas já que é possível garantir nível de tensão adequado ao dispositivo através da utilização de reguladores de tensão e filtros, para a segunda causa, mesmo os COTS são fabricados para operação na faixa de temperatura encontradas no ambiente espacial. Já o *bit flip* causado por radiação cósmica pode ser soluciona utilizando dispositivo com proteção a radiação, ou implementando métodos de detecção e correção de erros.

3 CÓDIGO DE DETECÇÃO E CORREÇÃO DE ERROS

3.1 CÓDIGO DE HAMMING

O código de Hamming é um esquema de detecção e correção de erros de bloco linear desenvolvido por Richard Hamming em 1950. O trabalho inicialmente conduzido em códigos de Hamming permitiu que máquinas de computação em grande escala executassem um grande número de operações sem um único erro no resultado final (Hillier e Balyan, 2019).

O código de Hamming para detecção e correção de erros consiste na adição de alguns bits de paridade (r) às informações enviadas (k). Os *bits* de paridade são calculados através de uma matriz geradora que é utilizada para dar forma à chamada palavra-código (n). A quantidade de bits da palavra-código pode ser calculada de acordo com a equação (1).

$$n = 2^r - 1 \tag{1}$$

Isso significa que a quantidade de *bits* de dados de informação pode ser calculada usando a equação (2).

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

Usando uma matriz de verificação de paridade, o esquema pode detectar a ocorrência de um erro e corrigir automaticamente o *bit* corrompido e dessa forma permitir a extração da informação original, livre de erros.

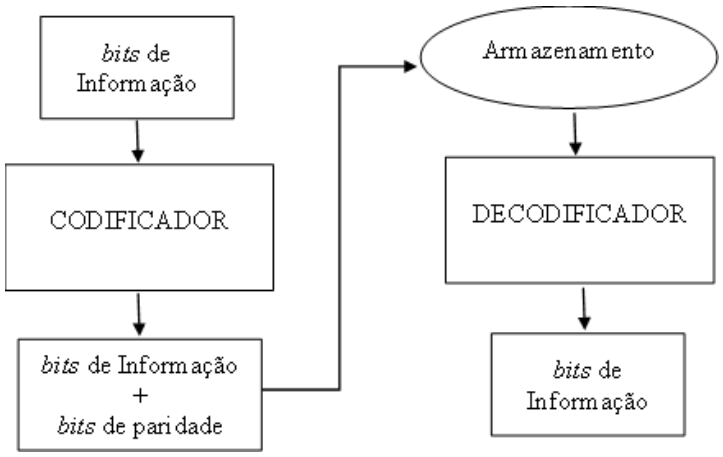
O código de Hamming é um esquema que adiciona *bits* de paridade que se sobrepõem à informação a ser protegida de forma a permitir que erros de *bit* único possam ser detectados e corrigidos.

O código de Hamming pode ser dividido em dois blocos de códigos: o codificador e o decodificador. O codificador é o bloco responsável por gerar a palavra-código, isto é, a união

entre os *bits* da informação a ser protegida com os *bits* de paridade calculados para esta. Já o decodificador é o bloco onde é realizada a checagem da informação, que é a palavra código, em busca de erros. Essa checagem é realizada através de uma verificação de paridade da palavra-código. É no bloco do decodificar onde é possível detectar e corrigir o erro ocorrido. Intermediando o codificador e o decodificador está o processo de armazenamento da informação, que é onde os erros podem ser corrompidos.

O fluxograma apresentado na Figura 10 ilustra as etapas do código de Hamming.

Figura 10 – Fluxo da informação no código de Hamming.



Fonte – Elaborado pelo autor.

A volatilidade das memórias RAM (*Randon Accesss Memory*) contribui para que os erros aconteçam com maior frequência nesta, quando comparado a outros tipos de memória. Os erros geralmente ocorrem quando as partículas irradiadas penetram nas células de memória contidas na RAM. Esses erros são definidos como mudanças de *bit* na memória. O código de Hamming clássico é capaz de realizar a detecção e correção de um único *bit* corrompido na informação, que é denominado pela sigla SECSED (do inglês - *Single Event correction and Single Event Detection)*.

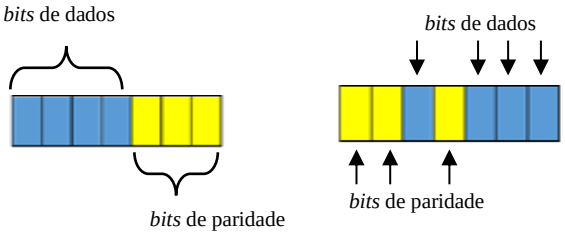
3.1.1 Detecção e correção de erros (SECSED)

Conforme mencionado no tópico anterior, o código de Hamming usa *bits* de paridade para realizar a detecção e correção de erros. A posição desses *bits* de paridade depende se o código é sistemático ou não sistemático. No formato de código sistemático, os *bits* de paridade

são adicionados ao final da palavra-código, já no formato não-sistemáticos os *bits* de paridade ocupam todas as posições que são potência de 2 na palavra-código.

Na Figura 11 é possível verificar a configuração do código sistemático e não sistemático.

Figura 11 – Esquerda: esquema sistemático; direita: esquema não-sistemático;



Fonte – Elaborado pelo autor.

Na Tabela 1, um layout típico de palavra-código é mostrado. Observe que esta tabela inclui apenas as posições de bit 1 a 7, mas pode continuar indefinidamente.

Tabela 1 – Layout do código de Hamming não-sistemático.

Posição do *bit*		1	2	3	4	5	6	7
bit codificado		P1	P2	D1	P4	D2	D3	D4
Cobertura dos dados codificados	P1	**X**		**X**		**X**		**X**
	P2		**X**	**X**			**X**	**X**
	P4				**X**	**X**	**X**	**X**

Fonte - Adaptado de (Hillier e Balyan 2019).

Da Tabela 1, as seguintes observações podem ser feitas:

I. A posição do *bit* (palavra-código) depende da quantidade de bits de dados protegidos

II. As posições dos *bits* de paridade são determinadas de acordo com, 2 à potência do *bit* de paridade:

(a) $2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8 \ e \ 2^4 = 16$

III. Relação do *bit* de paridade com a posição do *bit* e os *bits* de dados.

(b) *bit* de paridade 1 (P1) representa as posições dos *bits*: 1 (P1), 3,5, 7, etc. (todos as posições impares)

(c) *bit* de paridade 2 (P2) representa as posições dos *bits*: 2 (P2), 3, 6, 7, 10,11, etc (pares alternados).

(d) *bit* de paridade 4 (P4) representa as posições dos *bits*: 4 (P4), 5, 6, 7, 12,13, etc (sequência de quatro *bits*).

(e) *bit* de paridade 8 (P8) representa as posições dos *bits*: 8 (P8), 9, 10, 11, 12,13, etc (sequência de oito *bits*).

(f) *bit* de paridade 16 (P16) representa as posições dos *bits*: 16 (P16), 17, 18, 19, 20,21, etc. (sequência de dezesseis *bits*).

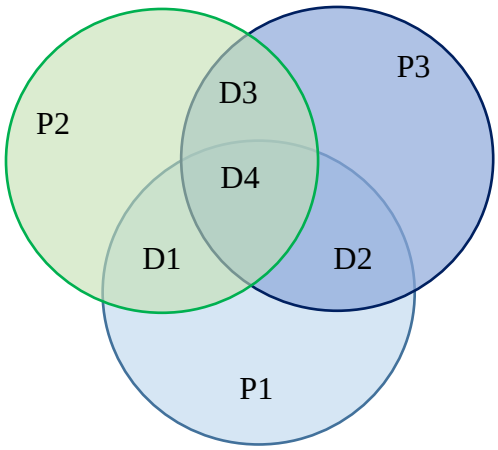
IV. A posição entre os *bits* de paridade é preenchida com *bits* de dados.

V. O *layout* faz com que cada coluna tenha uma combinação única de *bits* de paridade, para cada posição de *bits.*

VI. Essa combinação única de *bits* de paridade é chamada de valor da síndrome

Na Figura 12 é possível visualizar a relação entre os *bits* de paridade e os *bits* de dados que dependem destes.

Figura 12 – bits de paridade versus bits de dados. Esquema Hamming(7,3).



Fonte – Hellier e Balyan, 2019.

3.1.2 Construção da matriz geradora (*G*)

A matriz geradora (**G**) é usada ao codificar os dados de informação para formar a palavra-código.

A matriz $G_{(k\times n)}$ é definida como combinação de uma matriz identidade (I) de dimensão $k \times k$ e uma sub-matriz (P) de dimensão $k \times r$. A relação entre essas matrizes pode ser vista na equação (3).

$$G_{k\times n} = [P_{k\times r} \mid I_{k\times k}\ ] \tag{3}$$

Para o código de Hamming(7,4) a matriz geradora toma forma apresentada na equação (4).

$$G_{4\times 7} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

As três primeiras colunas da matriz $G_{4\times 7}$ representam a sub-matriz de paridade $P_{4\times 3}$ e as demais compõem a matriz identidade $I_{4\times 4}$.

Como o esquema não-sistemático é adotado neste trabalho, é preciso realizar uma mudança de posição nas colunas 3 e 4 da matriz G, dessa forma tem-se a equação (5).

$$G_{4\times 7} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

As colunas 1, 2 e 4 da matriz G mapeiam os *bits* de dados para os respectivos *bits* de paridade P1, P2 e P3, onde cada elemento de cada coluna que seja igual à 1 representa o bit incluído na relação, assim:

- P1 mapeia D1, D2, D4
- P2 mapeia D1, D3, D4
- P3 mapeia D2, D3, D4

3.1.3 Construção da matriz de verificação de paridade (H).

A matriz de verificação de paridade (H) é usada na decodificação e correção da palavra-código, a fim de extrair uma mensagem sem erros. A matriz H forma uma das bases do código

de Hamming. Devido à relação entre a matriz de verificação de paridade e a matriz geradora, o código de Hamming é compatível com SECSED.

A matriz $H_{(r\times n)}$ é definida como sendo uma combinação do negativo da submatriz ($P_{(k\times r)}$) transposta e uma matriz identidade $I_{(r\times r)}$, como apresentado na equação (6).

$$H_{r\times n} = [\, I_{r\times r} \mid -\ P_{k\times r}^{T}\,] \tag{6}$$

A matriz $H_{3\times 7}$ está mostrada na equação (7).

$$H_{3\times 7} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{7}$$

Pelo mesmo motivo que foi necessário inverter as colunas 3 e 4 da matriz G, também se faz necessário realizar o procedimento com a matriz H, dessa forma a equação X ganha a seguinte forma.

$$H_{3\times 7} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{8}$$

A relação entre as matrizes G e H está mostrada na equação 9.

$$G * H^{T} = 0 \tag{9}$$

3.1.4 O Codificador de Hamming.

O codificador de Hamming é responsável por gerar a palavra-código (com comprimento de $n\ bits$) a partir da mensagem denotada por (M) e da matriz geradora (G). Uma vez gerada, a palavra-código contém os *bits* de mensagem e de paridade. A palavra-código é calculada a partir do resto da divisão por dois da matriz resultante da equação (10).

$$n = [\, M_{k-bits} * G_{k\times n}] \tag{10}$$

Substituindo a equação (8) em (10) e arbitrando uma matriz M genérica, tem-se a equação (11).

$$n = [D1 \quad D2 \quad D3 \quad D4] * \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

Após extrair o resto da divisão por 2 de cada elemento da matriz resultante, tem-se a equação (12).

$$n = [P1 \quad P2 \quad D1 \quad P3 \quad D2 \quad D3 \quad D4] \tag{12}$$

A expressão matemática apresentada na equação (11) pode ser realizada essencialmente utilizando portas XOR.

3.1.5 O decodificador de Hamming.

O decodificador de Hamming é responsável por gerar a síndrome ($r - bits$ de comprimento) a partir da palavra-código ($n - bits$ de comprimento) denotada por C e a matriz de verificação de paridade (H). Uma vez gerada, a síndrome contém o padrão de erro que permite que o este seja localizado e corrigido. A síndrome é calculada usando a equação 7.

$$sindrome_r = mod_2(\, C_{n-bits} * H^T_{r \times n}) \tag{13}$$

A equação (13) pode ser elaborada utilizando portas lógicas AND e XOR.

As equações necessárias para implementar o código que Hamming podem facilmente ser representadas utilizando portas lógicas, por esse motivo foi escolhido realizar o processo de detecção e correção de erros utilizando o código de Hamming em FPGA.

3.2 CÓDIGO DE HAMMING EM CONJUNTO COM O MÉTODO DE PARIDADE

O código de Hamming é vastamente utilizado em sistemas com detecção e correção de erros e é capaz de realizar a detecção e correção de erros simples, isto é, quando há apenas um bit incorreto na informação, porém é possível adaptar o código para ser capaz de realizar a detecção de um erro duplo.

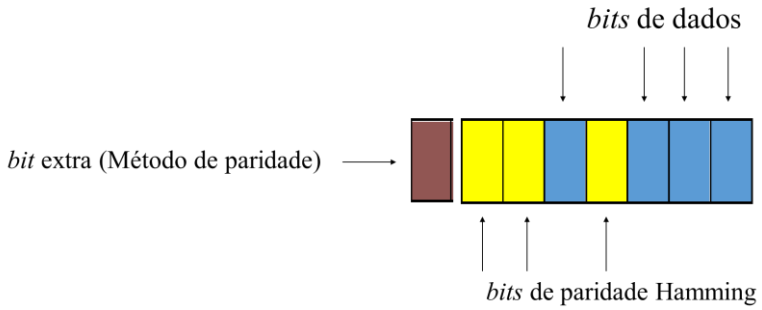
Uma das formas de realizar essa modificação é utilizando o método de paridade em conjunto com o método de Hamming.

O método de paridade consiste na adição, pelo transmissor, de um *bit* extra à informação seguindo a seguinte regra:

- Informação com número ímpar de *bits* 1, adiciona 1 na palavra.
- Informação com número par de *bits* 1, adiciona 0 na palavra.

Uma ilustração do método pode ser visualizada na Figura 13.

Figura 13 – Método de paridade + Hamming.



Fonte – Elaborado pelo autor.

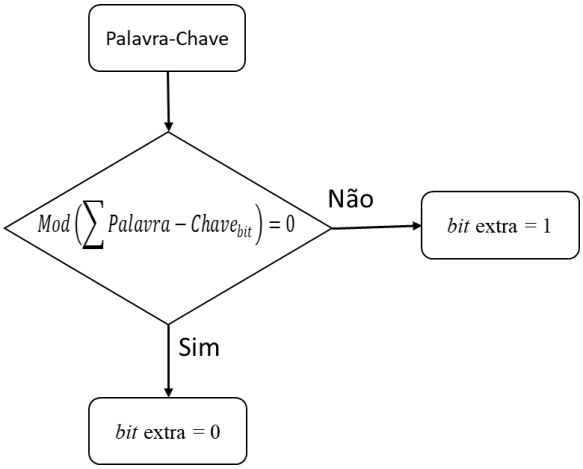
3.2.1 Codificador de Hamming com método de paridade

Após gerar a palavra código é necessário realizar a identificação da quantidade de bits 1 na palavra para, então, adicionar o *bit* extra à informação. Isso pode ser realizado com auxílio da equação 14.

$$Mod\left(\sum Palavra - Chave_{bit}\right) \tag{14}$$

O nível logico é definido após avaliar o resultado da equação 14 conforme mostrado no fluxo apresentado na Figura 14.

Figura 14 – Fluxograma para obtenção do valor do bit estra.

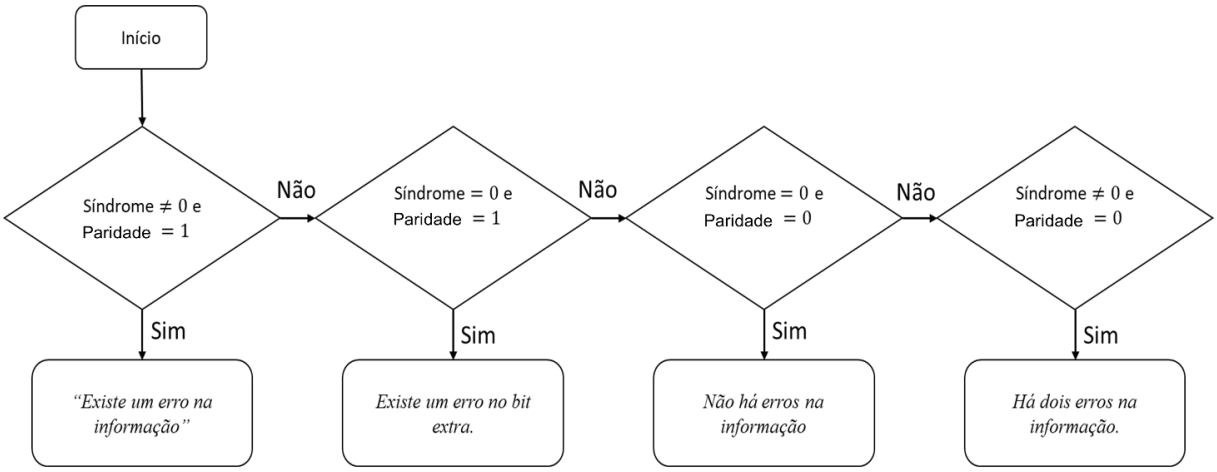


Fonte – Elaborado pelo autor.

3.2.2 Decodificador de Hamming com método de paridade

Com o *bit* extra adicionado a palavra código, se faz necessário avaliar 4 situações possíveis para o resultado do decodificador, conforme fluxograma mostrado na Figura 14.

Figura 15 – Fluxo decisório do código de correção.



Fonte – Elaborado pelo autor.

A partir do resultado do fluxo mostrado acima, o decodificador poderá corrigir o *bit* corrompido e retornar a informação original, ou descartar o dado caso este esteja corrompido em mais de um *bit*.

4 MATERIAIS E MÉTODOS

Para a concepção do projeto e protótipo propostos neste trabalho, serão utilizados os materiais e recursos apresentados na Tabela 2.

Tabela 2 – Lista de materiais utilizados do projeto.

Material	Características
Microcontrolador STM32	Modelo: STM32F411 RAM: 128 Kb ROM: 512 Kb Clock: 100 MHz Cortex-M4, 32 bits Interfaces: 36 I/O Tensão: 3,3 V
FPGA Altera Cyclone II	Modelo: EP2C5T144 Clock: 50 Mhz EEPROM: 4 Mbits Programação via JTAG ou AS Tensão: 3,3 V 112 pinos
Gravador ST-Link	Modelo: ST-LINK V2 STM8/STM32 Tensão: 5 V

Fonte - Elaborado pelo autor.

Todos os componentes foram selecionados levando em consideração os requisitos para o projeto proposto. O modelo do microcontrolador STM32 foi escolhido pois possui características robustas de Hardware para uma aplicação em nanossatélite, como memórias RAM e ROM suficientes para lidar com o grande volume de dados que passam pelo microcontrolador do OBC.

Para a escolha do FPGA levou-se em consideração a necessidade deste poder ser interligado ao microcontrolador STM32 de forma que seja possível realizar a operação de detecção e correção de erros.

O ST-Link é utilizado para realizar a gravação dos firmwares no microcontrolador SMT32.

5 IMPLEMENTAÇÃO E ANÁLISE

5.1 SIMULAÇÕES

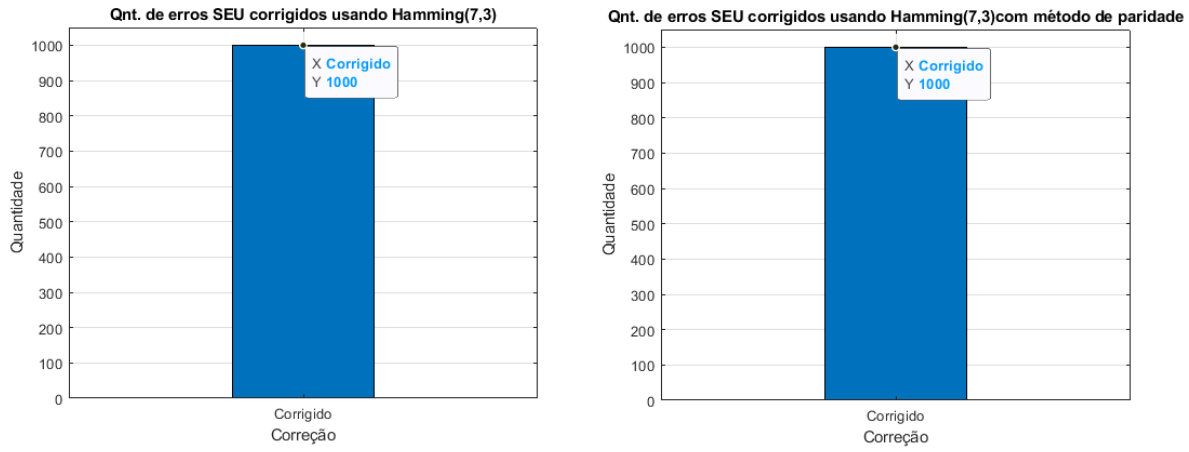
Foram implementadas duas versões do código de detecção e correção de erros utilizando o *software* MATLAB R2021b e cada algoritmo foi submetido à dois testes. O primeiro teste consistiu na avaliação do método clássico do código de Hamming(7,4) para detectar e corrigir erros causados por falhas do tipo SEU, isto é, quando o estado lógico de um único *bit* foi invertido na informação. O segundo teste consistiu avaliação do código de Hamming(7,4), em conjunto com o método do *bit* de paridade, em detectar e corrigir erros causados por falhas do tipo SEU.

Para o primeiro teste os dois algoritmos foram submetidos á uma bateria de testes com 1000 amostradas e com cada uma tendo apenas um único *bit* corrompido, isto é, simulação de evento do tipo SEU. Tanto a geração das amostras quanto escolha das posições dos *bits* a serem corrompidos ocorreram de forma aleatória.

Em ambos os métodos foi possível identificar e corrigir com 100% de precisão todos os erros nos dados simulados de forma aleatória, como apresentado na Figura 16.

Na Figura 16(a) estão apresentados os resultados dos testes utilizando o método de Hamming(7,3) e na Figura 16(b) é possível avaliar os resultados do teste utilizando o código de Hamming(7,3) em conjunto com o método do *bit* de paridade.

Figura 16 – Experimento para detecção e correção de eventos de efeito único.



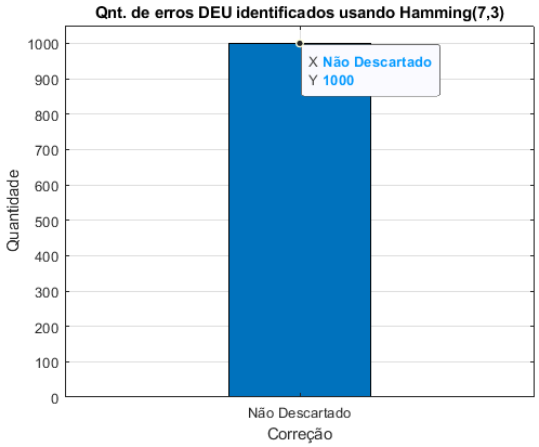
(a) (b)

Fonte – Elaborado pelo autor.

O intuito do segundo teste foi avaliar a capacidade dos algoritmos em detectar a presença de dois erros na mesma informação, isto é, a inversão do estado lógico de até dois *bits*. Para isso foram geradas de forma aleatória um conjunto de dados contendo 1000 informações com 4 *bits* de comprimento, e em seguida cada informação teve dois *bits* corrompidos de forma randômica.

Para o código de Hamming clássico foi possível constatar que dos 1000 erros do tipo DEU simulados, o algoritmo não foi capaz de determinar se cada uma das informações continha um ou dois *bits* corrompidos, desta forma o algoritmo não sinaliza se o dado precisa ser descartado. O resultado deste experimento pode ser visualizado na Figura 17.

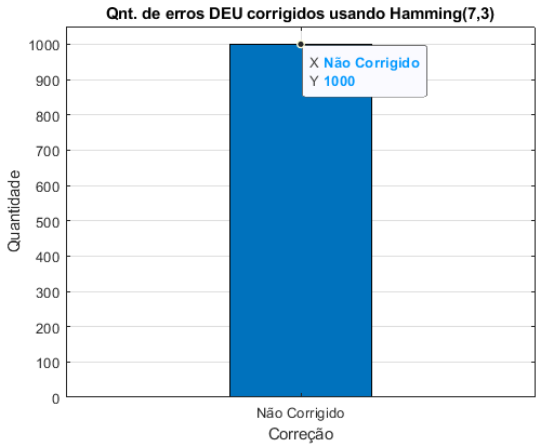
Figura 17 - Avaliação da capacidade do método para detectar eventos do tipo DEU.



Fonte – Elaborado pelo autor.

Como o algoritmo de Hamming clássico não realiza a detecção da presença de mais de um *bit* corrompido na informação, este trata o dado inserido como contendo um erro e realiza a tentativa de correção. Para o teste anterior a realização da tentativa de correção dos erros resultou em 0% de dados corrigidos, como mostrado na Figura 18.

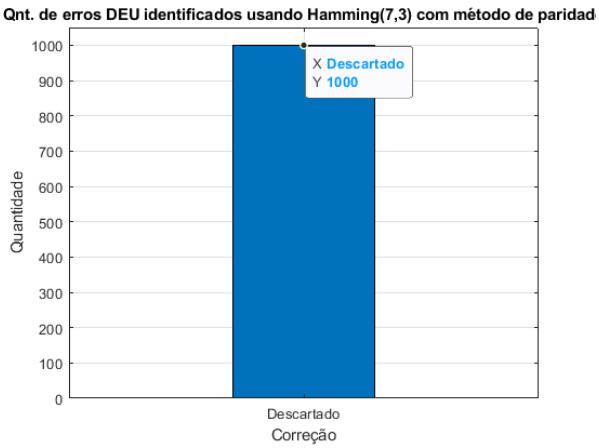
Figura 18 – Incapacidade de correção de evento DEU com o método de Hamming.



Fonte – Elaborado pelo autor.

O mesmo experimento anterior foi realizado com o método do código de Hamming em conjunto com o método do *bit* de paridade. O resultado está mostrado na Figura 19 e é possível constatar que tal método permite a detecção da presença de dois *bits* alterados e então sinaliza a necessidade de descarte da informação corrompida.

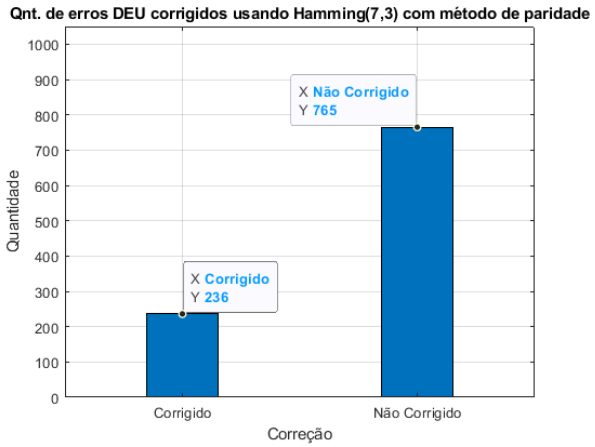
Figura 19 – Detecção de eventos DEU pelo método de Hmming + *bit* de paridade.



Fonte – Elaborado pelo autor.

Ainda foi realizado o teste de correção utilizando o método proposto e foi possível verificar que para o conjunto de dados gerados o algoritmo foi capaz de corrigir 23,6% dos dados corrompidos, como mostrado na Figura 20.

Figura 20 – Correção de erros DEU com método de Hamming + *bit* de paridade.



Fonte – Elaborado pelo autor.

A partir dos resultados analisados, foi possível verificar que ambos os métodos realizam a detecção de erros do tipo SEU com 100% de precisão, porém somente o segundo método é capaz de realizar a detecção de erros do tipo DEU. O resumo do experimento está mostrado na Tabela 3.

Tabela 3 – Resumo das simulações dos métodos.

Método	**SEU** (detecção)	**SEU** (correção)	**DEU** (detecção)	**DEU** (correção)
Hamming	100%	100%	0%	0%
Hamming + *bit* de paridade	100%	100%	100%	23,6%

Fonte – Elaborado pelo autor.

O objetivo do método proposto é realizar a detecção e correção de erros simples (SEU), além de detectar a presença de erro duplo (DEU), porém o método provou ser capaz de corrigir uma parte dos dados corrompidos por evento DEU. A explicação para essa característica é que quando um dos *bits* alterados é o *bit* de paridade, este pode ser ignorado e a palavra-código fica contendo apenas um erro e este é tratado e corrigido corretamente, porém não é possível determinar quando isso ocorre na presença de um segundo erro e cabe ao projetista decidir se após detecção de erro do tipo DEU a informação é descartada ou realiza-se a tentativa de correção.

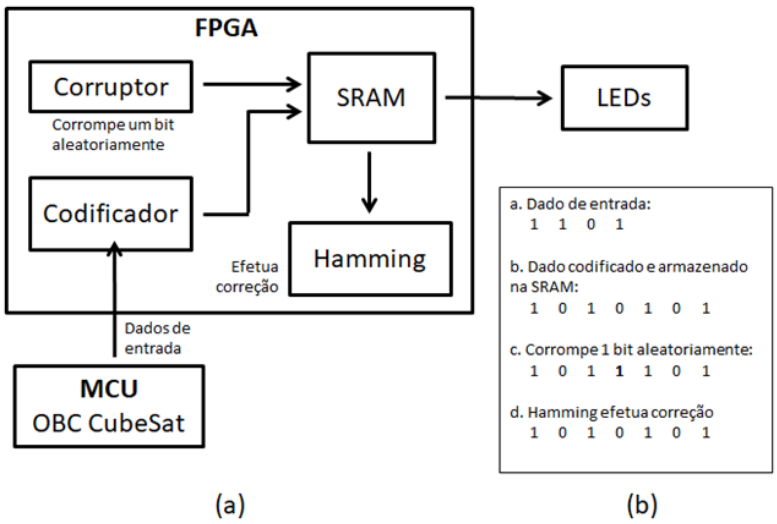
Para mais detalhes sobre os códigos desenvolvidos consultar o APÊNDICE A.

5.2 EXPERIMENTAÇÃO

A abordagem aqui proposta objetiva tornar os dispositivos FPGA capazes de detectar e reparar falhas de configuração induzidas por SEU, em bancos de memória internos, sem recorrer a circuitos externos de armazenamento. A intenção é que a detecção e correção de SEUs sejam realizadas através de uma verificação de integridade da memória SRAM implementada no próprio chip FPGA, usando para isso um Circuito de Detecção e Correção de Erros (EDAC).

Na Figura 21, é apresentado o funcionamento básico do método proposto. Na Figura 21(a) pode-se observar os blocos básicos constituintes do sistema. Os blocos principais são formados por um microcontrolador (MCU) e um FPGA. O MCU irá assumir o papel do OBC do CubeSat, enquanto que o FPGA será responsável pelo armazenamento dos dados em uma memória SRAM, bem como por comportar o circuito para codificação dos dados enviados a partir do MCU, assim como o circuito para detecção e correção de erros simples e o circuito para corromper os dados armazenados na SRAM. O circuito corruptor de *bits* atuará através de mudanças de nível lógico dos *bits* de forma aleatória. O conteúdo da memória SRAM atualmente acessada pelo MCU é exibido no conjunto de LEDs para que possa ser verificado o processo de correção utilizando o código de Hamming. Na Figura 21(b) é possível ver de forma resumida e através de um exemplo, o funcionamento do método. Após o fornecimento do dado de entrada a partir do MCU, este é codificado e armazenado na SRAM; em seguida, é feita a simulação de um SEU através da inversão do estado de um *bit*, de forma aleatória; O circuito com a implementação do código de Hamming atuará, efetuando a correção dos *bits* armazenados e retornando à informação original.

Figura 21 – Esquema proposto para experimentação.

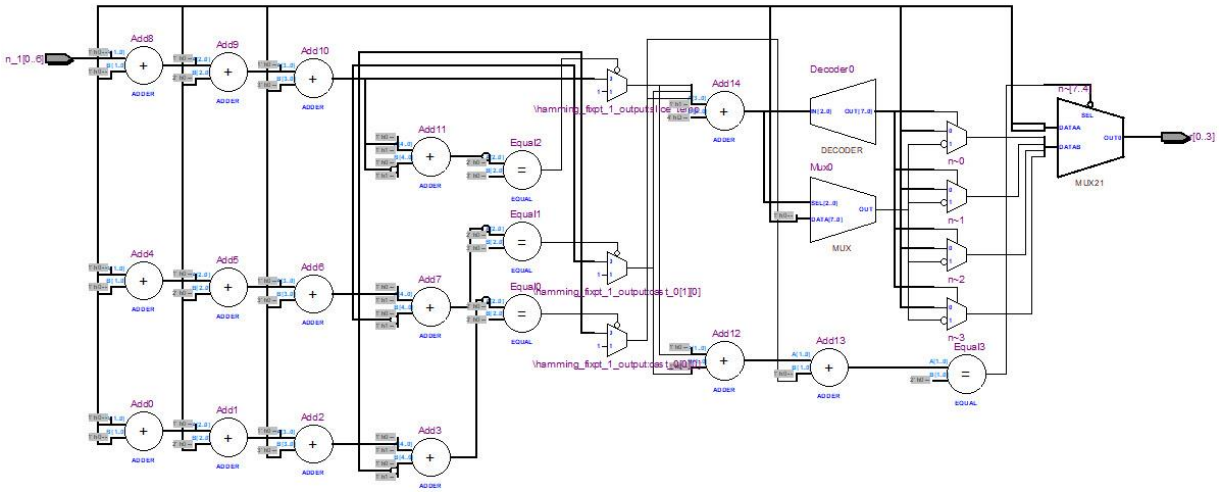


Fonte – Elaborado pelo autor.

Usando os *softwares* Altera Quartus II *Web Edition* e o MATLAB R2021b, o método de detecção e correção de erros foi desenvolvido e simulado. As funções de codificação e correção foram convertidas para VHDL usando a *ToolBox* HDL *Coder*, do MATLAB. Foram usadas as entradas de dados em ponto fixo e os arquivos VHDL exportados foram incorporados ao projeto dos experimentos no Altera Quartus II.

Usando o visualizador RTL do Altera Quartus II, é possível gerar a visão RTL do circuito gerados a partir do projeto criado. Pode-se observar na Figura 22, o RTL do circuito de correção através do código de Hamming gerado a partir do MATLAB. É possível perceber no RTL do circuito os componentes somadores, comparadores, multiplexadores e codificadores que foram utilizados no processo de geração de código em VHDL através do MATLAB. Percebe-se que as ferramentas de geração de código alcançaram um nível de maturidade ao ponto de acelerar o processo de desenvolvimento de aplicações em FPGA aplicados a sistemas embarcados espaciais.

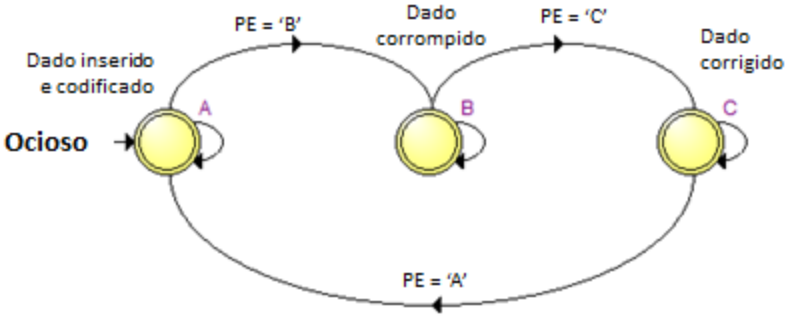
Figura 22 – Circuito RTL do código de Hamming.



Fonte – Elaborado pelo autor.

Foi realizado um teste de bancada para simular a entrada de dados e o armazenamento de alguns valores na memória SRAM, a codificação e a correção de eventos SEU gerados aleatoriamente. Após a compilação do projeto do circuito síncrono digital (*Register Transfer Level*, RTL), foi simulado um SEU no valor de entrada armazenado. Pode-se observar na Figura 23, a máquina de estados implementada para a realização do teste. O sistema inicia no estado ocioso (estado A), em que não há entrada de informações na memória. Após o dado ser inserido e codificado, a máquina de estados avança para o estado seguinte (estado B), onde ocorre a corrupção de um bit nos dados armazenados. Após, a máquina de estados avança para o último estado (estado C), onde o circuito responsável pela correção dos SEUs gerados é executado efetuando a correção dos erros provocados. A máquina de estados é então reiniciada para começar novamente o processo.

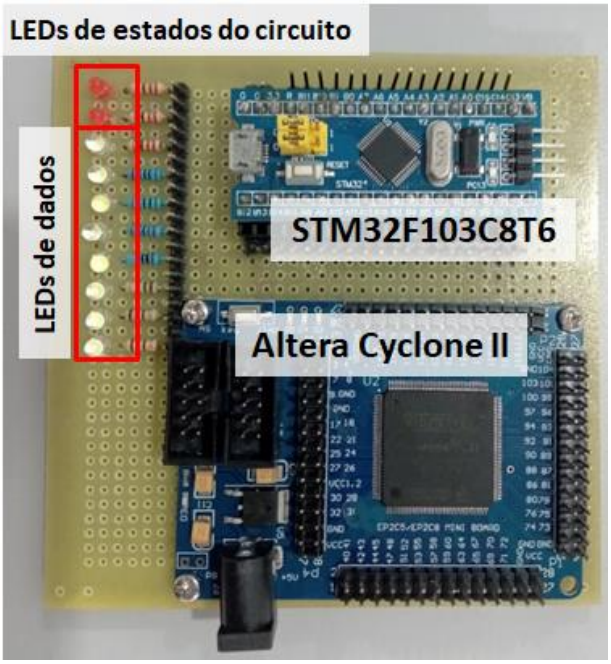
Figura 23 – Representação da máquina de estados projetada.



Fonte – Elaborado pelo autor.

Para os experimentos implementados fisicamente, foram usados os componentes FPGA Altera Cyclone II EP2C5T144C8, disponível em uma placa Cyclone II EP2C5 *Mini Dev Board*, o microcontrolador (*Microcontroller Unit*, MCU) STMF103C8T6 da STMicroeletronics. Tanto o FPGA quanto o MCU foram embarcados em uma placa de circuito impresso universal, onde também foram acoplados LEDs para visualização de dados do sistema e os estados do circuito armazenados no FPGA. Na Figura 24 é possível observar a placa desenvolvida para os experimentos. Optou-se neste protótipo experimental por confeccionar em uma placa universal pelo baixo custo e rápido desenvolvimento. No entanto, a ideia é elaborar uma placa de circuito impresso no padrão PC/104, de modo a ser embarcada em um CubeSat.

Figura 24 – Circuito desenvolvido para os experimentos.



Fonte – Elaborado pelo autor.

Após a confecção da placa, foram definidos os pinos do FPGA que seriam interligados ao MCU, bem como os pinos que seriam ligados aos LEDs de estado e aos LEDs de dados. Os pinos conectados ao MCU foram utilizados para endereçamento da memória SRAM no FPGA, realizar a entrada de dados na memória, realizar a saída de dados após a correção de *bits* usando código de Hamming, envio de um sinal de *clock*, envio de um sinal de entrada (*enable)* e envio de um sinal de *reset*. Ao todo, foram utilizados 15 pinos do MCU. Os pinos do MCU conectados à saída de dados do FPGA tiveram resistores de *PULL UP* internos habilitados, impedindo que

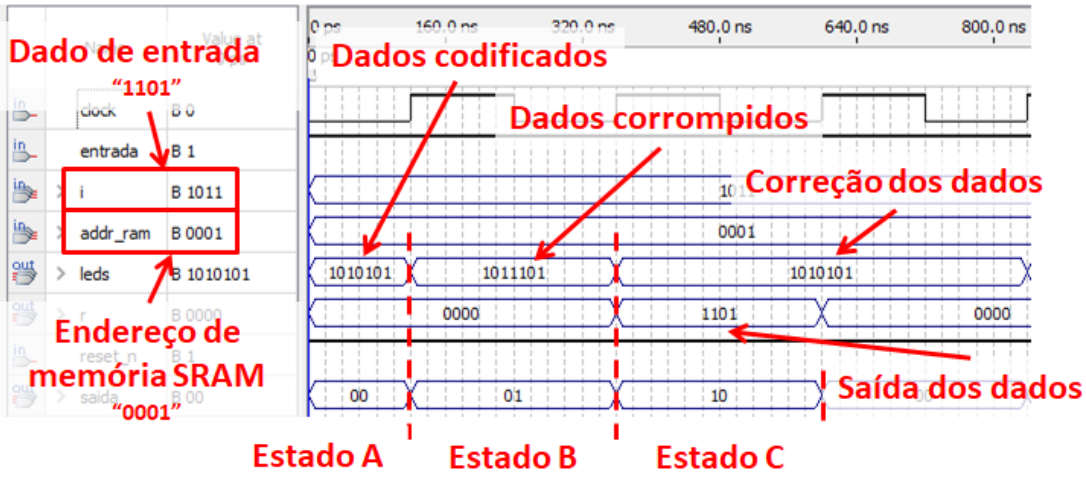
valores flutuantes nos pinos ocasionassem erros de leitura no MCU. O MCU foi programado usando o programador ST Link V2 e a codificação do *firmware* foi feita utilizando a IDE Arduino. A depuração do sistema foi realizada através de um módulo conversor USB-Serial, conectado à porta USB de um computador pessoal.

5.3 RESULTADOS

Foram realizados dois experimentos para avaliar a acurácia do método proposto e sua implementação. No primeiro experimento foi avaliada a capacidade de identificar e corrigir erros do tipo SEU usando apenas o código de Hamming. No segundo experimento, foi avaliada a capacidade de identificar, corrigir erros do tipo SEU além de identificar e descartar dados que contivessem erros do tipo DEU usando o código de Hamming em conjunto com o método do *bit* de paridade. Para ambos os experimentos, foi usada a mesma placa desenvolvida, conectada a um computador pessoal (PC) através de uma interface de conversão serial/USB. Foi desenvolvido um programa no MATLAB capaz de codificar, simular eventos SEU, identificar, corrigir e descartar dados usando os dois métodos propostos.

Na Figura 25 é possível verificar a captura de tela do diagrama de sinais da simulação, executada a partir do editor de forma de ondas de simulação (*Simulation Waveform Editor*), disponível no Altera Quartus II. Esta captura foi obtida a partir de uma das simulações executadas para método que identifica e corrige erros usando apenas a codificação de Hamming. Na captura de tela é possível identificar o dado de entrada e o endereço de memória SRAM do dado, dentro do FPGA. Além disso, é possível observar o dado codificado e este corrompido através com erro do tipo SEU, o dado corrigido e a saída do dado após o processo de correção e leitura da memória. A posição do *bit* que contém o erro foi informada como parâmetro, a partir do microcontrolador, de forma aleatória. Na parte inferior da figura, pode-se visualizar as transições dos estados, indicando o momento em que cada operação da máquina de estados ocorre.

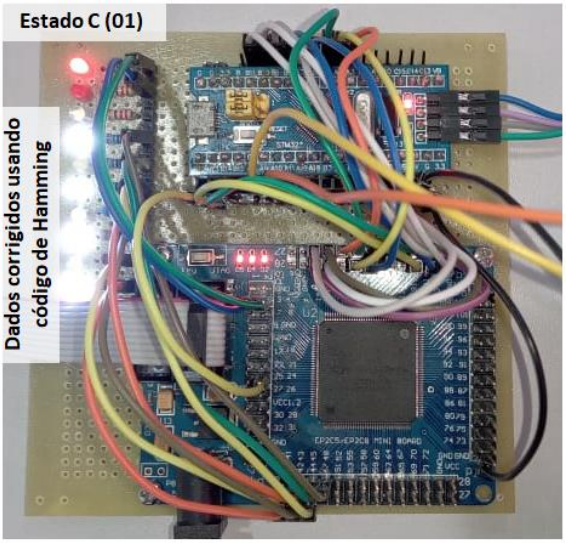
Figura 25 – Simulação do código de Hamming em VHDL.



Fonte – Elaborado pelo autor.

Na Figura 26 é possível observar a placa de prototipagem confeccionada em funcionamento, apresentando nos LEDs o estado atual e os dados corrigidos após o circuito do código de Hamming ser executado. Nos LEDs são apresentados o estado atual do circuito no FPGA e o conteúdo da memória RAM acessada no momento do teste.

Figura 26 – Protótipo para experimentação.



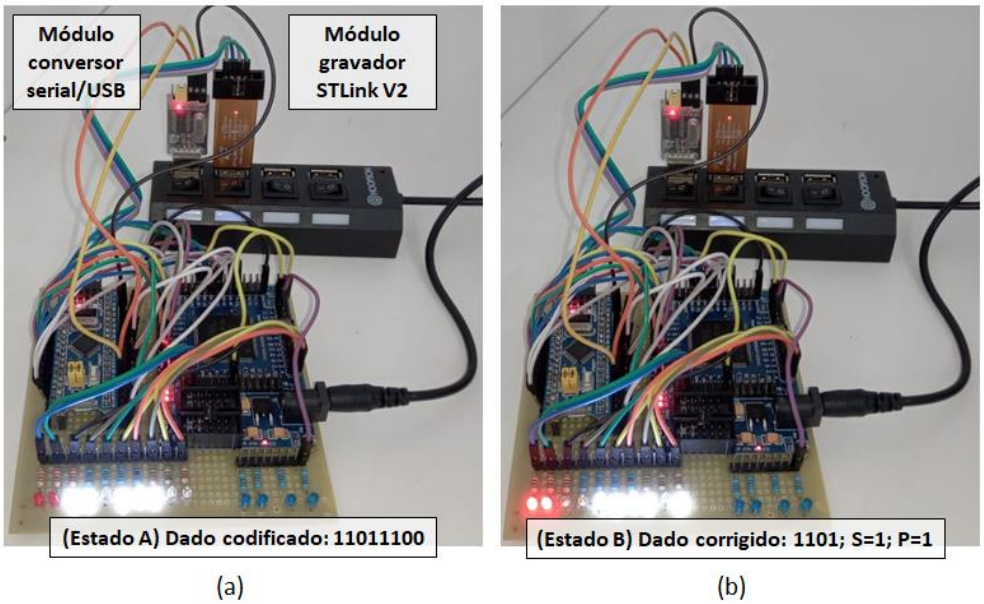
Fonte – Elaborado pelo autor.

No segundo experimento, além dos dados informados no primeiro experimento, foram informados dois valores de posições de *bits* que contém erro como parâmetro, a partir do microcontrolador, isto é, foi simulado a corrupção de até dois *bits* na informação. Desta forma,

testou-se a capacidade do sistema identificar, além de erros do tipo SEU, erros do tipo DEU. Os resultados de todos os testes realizados foram comparados com os resultados das simulações desenvolvidas no MATLAB para checagem e comparação de desempenho. Foi usado o programa de computador *AccessPort* para se ter acesso à saída dos resultados do experimento, através do módulo conversor serial/USB. Na Figura 27 é possível observar um dos vários testes efetuados. Na Figura 27 (a) é apresentada a configuração inicial da máquina de estados, com a exibição dos dados codificados e armazenados na RAM, exibidos a partir dos LEDs de dados. Estes dados são fornecidos pelo microcontrolador para o FPGA e também são enviados para o PC usando o módulo conversor serial/USB. Desta forma, o processamento realizado no FPGA pode ser conferido a partir dos LEDs da placa e também no PC, a partir do programa *AccessPort*. Na Figura 27 (b) é possível observar o resultado do processamento do método de correção usando o código de Hamming combinado com método de verificação de erro de paridade.

Pode-se perceber o dado corrigido, a informação sobre a existência de erro de síndrome e a existência de erro de paridade. Essas informações são repassadas para o microcontrolador, que pode então decidir se o dado pode ser lido corretamente ou descartado, em caso de erro de síndrome e paridade simultaneamente.

Figura 27 – Teste de correção de erros do tipo SEU.

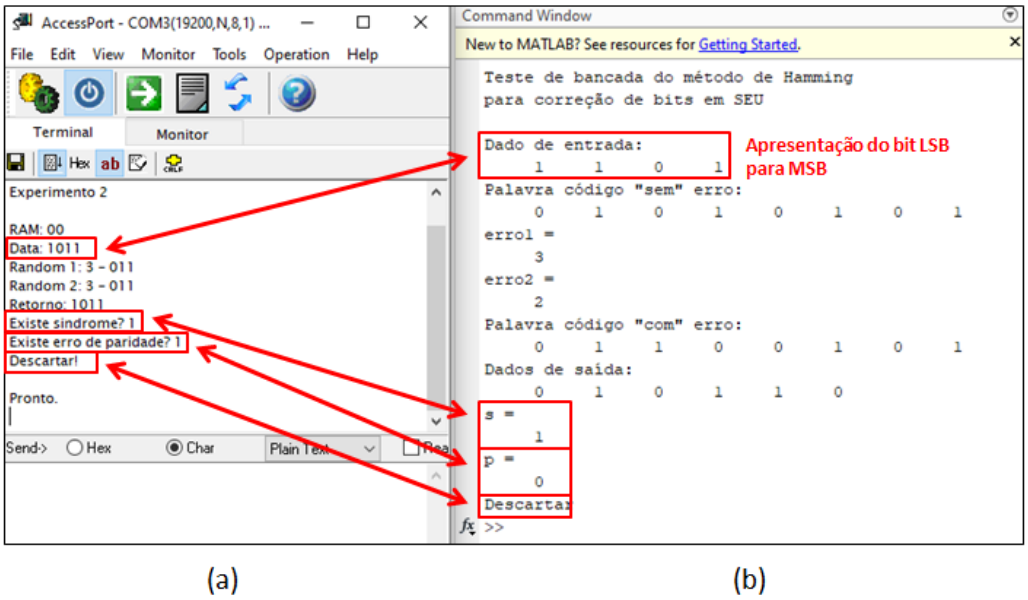


Fonte – Elaborado pelo autor.

Na Figura 28 (a) é possível observar a saída enviada pelo microcontrolador para o PC através do módulo conversor serial/USB, na tela do programa *AccessPort*. Na Figura 28 (b)

pode-se ver o resultado de uma das simulações efetuadas no MATLAB. É possível perceber em ambas as figuras que os resultados são equivalentes, ou seja, os resultados simulados no PC são os mesmos executados no FPGA, com os dados fornecidos pelo microcontrolador.

Figura 28 - Comparação entre o experimento com as simulações.



Fonte – Elaborado pelo autor.

A capacidade de detecção de falhas da técnica foi analisada através de duas baterias de teste, uma para cada experimento. Em cada uma das baterias, foram armazenados dados na memória SRAM do FPGA e definidas posições aleatórias de *bits* que deveriam ser trocados, ou seja, uma injeção de falha nos *bits* de dados. No primeiro experimento, foram executados 100 procedimentos de injeção de falhas, com posição definida de forma aleatória, em cada um dos 4 *bits* de informação, também aleatórios. No segundo experimento, foram executados 200 procedimentos de injeção de falhas com duas posições definidas de forma aleatórias, em cada um dos 4 bits de informação, também aleatórios. Os resultados das simulações foram, em seguida, comparados com as simulações realizadas no MATLAB.

Como propostas de trabalhos futuros, pretende-se modificar o circuito sintetizado em FPGA para realizar de forma cíclica constantes verificações e correções em todos os endereços da memória RAM disponível no sistema. A ideia é desenvolver um sistema autônomo que sinalize para o computador de bordo as regiões de memória que estão sendo manipuladas a fim de evitar conflitos de acesso entre o FPGA e o microcontrolador. Além disso, será projetada uma placa de circuito impresso no padrão PC/104 para CubeSats, a fim de testar o sistema

integrado aos demais subsistemas, incluindo energia, telecomunicações e possíveis cargas úteis, tanto em bancada quanto integrados em uma estrutura de tamanho 1U ou maior. Dessa forma, o sistema pode funcionar como um sistema redundante de armazenamento de dados de missão, em conjunto com o OBC do CubeSat, para que o método proposto possa ser validado e futuramente embarcado como dispositivo de apoio ao OBC principal.

6 CONSIDERAÇÕES FINAIS

O crescente interesse pelo desenvolvimento de tecnologias espaciais por parte de instituições de pesquisa e universidades do mundo todo, em especial pela área de nanossatélites, torna necessário a concepção de tecnologias cada vez mais robustas nesta área que proporcionem um aumento da confiabilidade dos sistemas de bordo e aumento da vida útil dos satélites.

Neste trabalhou foi proposto um novo método de detecção e correção de erros causados por eventos do tipo SEU, além de ser possível detectar eventos do tipo DEU. Dessa forma é possível desenvolver nanossatélites utilizando componentes de prateleira e ainda ser possível garantir a confiabilidade das informações coletadas através da utilização da técnica EDAC proposta. O principal ganho do método aqui desenvolvido está no fato de que a maior parte dos algoritmos utilizados na indústria para o mesmo fim é capaz de detectar e corrigir a presença de eventos SEU, enquanto eventos do tipo DEU passam desapercebido pelo sistema de detecção o que acarreta na utilização de informações corrompidas pelo nanossatélite.

Ainda é possível destacar que o método proposto neste trabalho identifica a presença de erros do tipo DEU dando a possibilidade ao sistema descartar o dado corrompido. Outro fator a ser notado é que o algoritmo desenvolvido é capaz de corrigir parcialmente os dados corrompidos por eventos tipo DEU, isso se dá pelo fato que a corrupção do *bit* de paridade não acarreta em corrupção de informação útil, sendo assim, no processo de correção, informações com dados afetados no primeiro *bit* podem ser recuperados e corrigidos.

REFERÊNCIAS

ALÉN SPACE - **A Basic Guide to Nanosatellites.** Disponível em: <https://alen.space/basic-guide-nanosatellites/>, em 10 de agosto de 2020.

Bentoutou, Youcef, Bensikaddour,El-Habib. **Analysis of radiation induced effects in high-density commercial memories on-board Alsat-1: The impact of extreme solar particle events.** Advances in Space Research 55, pags 2820-2832, 2015.

D. Burlyaev, R. van Leuken. **System Fault-Tolerance Analysis Of COTS-Based Satellite On-Board Computers.** Microelectron Journal, Elsevier, 2014.

Dug, M., Krstic, M. **Implementation and Analysis of Methods for Error Detection and Correction on FPGA.** IFAC – International Federation of Automatic Control, Elsevier Ltd. PapersOnLine 51-6 (2018), pags 348-353.

Eickhoff, Jens. **Onboard computers, Oboard Software and Satellite Operations.** Spring Heidelberg Dordrecht London New York, 2011.

Fauzi, Achmad., Nurhayati, Rahim, Robbi. **Bit Error Detection and Correction with Hamming Code Algorithm.** IJSRSET, Volume 3, Issue 1, Theme Section: Engineering and Technology, 2017, pags 76-81.

Goel, Roger C., Villa, Paulo R. C., Poehls, Letícia B., Bezerra, Eduardo A., Vargas, Fabian L. **An efficient EDAC approach for handling multiple bit upsets in memory array.** Microelectronics Reliability 88-90, pags 214-218, 2018.

Han, J.H. & Kim, C.G. 2006. **Low earth orbit space environment simulation and its effects on graphite/epoxy composites.** Composie Structures. 72(7): pp. 218-226. Elsevier Ltd.

Hardy, J. 2009**. Implementation du Protocol AX.25 a Bord du Nanosatellite OUFTI-1**. Master's thesis, University of Liege, Liege.

Hillier, Caleb, Balyan, Vipin. **Error Detection and Correction On-Board Nanosatellites Using Hamming Codes.** Journal of Electrical and Computer Engineering, Volume 2019, Article ID 3905094, 15 pages – Hindawi, 2019.

Holman, G.& Benedict, S. 2007. **Solar Flare Theory Educational Web Pages.Nasa's Goddard Space Flight Center**. Disponível em: <http://hesperia.gsfc.nasa.gov/sftheory/frame1.htm>, acesso em 20 de agosto de 2020.

J. Benfica, B. Green, B.C. Porcher, L.B. Poehls, F. Vargas, N. Medina, N. Added, V.A.P. de Aguiar, E.L.A. Macchione, F. Aguirre, M.A.G. Silveira, M. Perez, M. Sofo Haro, I. Sidelnik, J. Blostein, J. Lipovetzky, E.A. Bezerra, **Analysis of SRAM-based FPGA SEU sensitivity to combined EMI and TID-imprinted effects**, IEEE Trans. Nucl. Sci. 63 (2016) 1294–1300.

Kulu, Erik. **Small Launchers - 2021 Industry Survey and Market Analysis.** 72nd International Astronautical Congress (IAC 2021), Dubai, United Arab Emirates, 25-29 October 2021.

L. Ciani, M. Catelani, **A Fault Tolerant Architecture To Avoid The Effects Of Single Event Upset (SEU) Inavionics Applications.** Measurement Journal, Elsevier, 2014.

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

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

Lumbwe, Lwabanji Tony. **Development of an Onboard Computer (OBC) for a Cubesat**. Thesis of a Master degree in Electrical Engineering at the Cape Peninsula University of Technology, Bellville, May 2013.

Marilia Hagen, Anibal. **South Atlantic Anomaly Seasonal Seismicity during Two Solar Cycles.** Scientific Research Publishing, Open Journal of Earthquake Research, 2020, 9, 307-322

Krstic, M. Dug; Jokie, D. **Implementation and Analysis of Methods for Error Detection and Correction on FPGA**. InternationalFederation os Automatic Control – IFAC, Elsevier Ltd, 2018.

NASA, 2017. **CubeSat101:** Basic concepts and processes of first-time CubeSat developers. NASA CubeSat Launch Initiative. October 2017.

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

Reviriego, P., Argyrides, C., Maestro, J. A. **Efficient error detection and Double Error Correction BCH codes for memory applications.** Elsevier, Microelectronics Reliability 52 (2012), pags 1528-1530.

Seunghwa Jung , Jihwan P. Choi , **Predicting System Failure Rates of SRAM-based FPGA On-Board Processors in Space Radiation Environments**, Reliability Engineering and System Safety (2018).

Shufan, W., Wen, C., Cai Xia, C., Chuanxing, Z., Zhongcheng, M., et al., 2015. **Earth observation and marine/air traffic monitoring with a multiple CubeSat constellation**. IAC.15.B4.4.3, 66th International Astronautical Congress, 12-16, Jerusalem, Israel.

Tolentino, Lean Karlo S., Padilha, Maria Victoria C., Juan, Ronnie O. Serfa. **FPGA-based redundancy bits reduction algorithm using the enhanced error detection correction code.** International Journal of Engineering & Technology, 7 (3), SPC, (2018), pags 1008-1013.

Wayne A. Scales, Chair, Joseph J. Wang, Co-chair, C. Robert Clauer. **Numerical Simulation of Ion-Cyclotron Turbulence Generated by Artificial Plasma Cloud Release.** Thesis, Virginia Polytechnic Institute and State University. Master of Science in Electrical Engineering, July, 01, 2009. Blacksburg, Virginia.

APÊNDICE A – Códigos desenvolvidos em MATLAB 2021b

O Código de Hamming(7,3), apresentado a seguir, foi desenvolvido em MATLAB 2021b e utilizado para a realização dos testes de detecção e correção de erros tipo SEU.

```
%% Título: Código de hamming(7,4) EDAC
%% author: Edeilson Pestana
%% Data: 10/04/2021

clc
clear all

% definir o nome do arquivo
%% Experimento1 = 'ExperimentoHammingClassicoSEU';
Experimento1 = 'ExperimentoHammingClassicoDEU';

%% Sub-matriz de paridade
P = [1 1 0 1;1 0 1 1; 0 1 1 1]; P = P';
%% Matriz Geradora (G) do código linear (n,k)
G = [P  eye(4)];              % G = [P | I]
G(:,[3 4]) = flip(G(:,[3 4]),2); % Inverte as linhas 3 e 4 de G

%% Matriz de verificação
 H = [eye(3) P'];                  % H = [I | P']
 H(:,[3 4]) = flip(H(:,[3 4]),2); % inverte as colunas 3 e 4

for i=2:1:1001
%% Dados de Informação
%% INFORMAÇÂO ALEATÓRIA
m = randi([0 1],1,4);
% salvar informação original
  xlswrite(Experimento1, {num2str(m)},1,strcat('A',num2str(i)));

%% Palavra-código (o codificador de Hamming)
n = mod(m*G,2);
 xlswrite(Experimento1, {num2str(n)},1,strcat('B',num2str(i)));


%% Inserção do erro aleatório erro 1
erro1 = randi([1 7],1);
% salvar posição do erro 1
  xlswrite(Experimento1, {num2str(erro1)},1,strcat('C',num2str(i)));


%% Inserção do erro aleatório erro 2
 erro2 = randi([1 7],1);
if(erro2 == erro1 && erro1<7)
  erro2 = erro2 + 1;
end
if(erro2 == erro1 && erro1==7)
    erro2 = erro2 -1;
end

% salvar erro 2
```

```
 xlswrite(Experimento1, {num2str(erro2)},1,strcat('D',num2str(i)))

n(erro1) = ~n(erro1);

%% **** COMENNTE ESSAS LINHAS PARA UM UNICO ERRO **********
   n(erro2) = ~n(erro2);

%%   salvar palavra chave com erro 2
  xlswrite(Experimento1, {num2str(n)},1,strcat('E',num2str(i)));

%% Decoder. Calculo da "sindrome"
sind = mod(n*H',2);
sind = flip(sind);


if (sum(sind)~= 0)
    % converte a posição em binário para decimal
    posicao = sind(1)*2^2 + sind(2)*2^1 + sind(3)*2^0;

    %% Correção (inverte o bit na posição identificada
    n(posicao)=~n(posicao);
     xlswrite(Experimento1, {'Detectado'},1,strcat('H',num2str(i)));
else

    xlswrite(Experimento1, {'Não Detectado'},1,strcat('H',num2str(i)));

end

%% Informação original

m = [n(3) n(5) n(6) n(7)];
% disp(m)

  xlswrite(Experimento1, {num2str(m)},1,strcat('F',num2str(i)));
i
end
```

O código apresentado abaixo foi proposto como método para realizar detecção e correção de erros SEU e DEU.

```
%% Título: Código de hamming(7,4)+ bit de Paridade EDAC
%% author: Edeilson Pestana
%% Data: 13/04/2021

clc
clear all

% definir o nome do arquivo
% Experimento1 = 'ExperimentoHammingComParidadeSEU';
Experimento1 = 'ExperimentoHammingComParidadeDEU';
```

```
%% Sub-matriz de paridade
P = [1 1 0 1;1 0 1 1; 0 1 1 1]; P = P';
%% Matriz Geradora (G) do código linear (n,k)
G = [P  eye(4)];              % G = [P | I]
G(:,[3 4]) = flip(G(:,[3 4]),2); % Inverte as linhas 3 e 4 de G

%% Matriz de verificação
 H = [eye(3) P'];                  % H = [I | P']
 H(:,[3 4]) = flip(H(:,[3 4]),2); % inverte as colunas 3 e 4

for i=2:1:1001
indice = num2str(i)
%% Dados de Informação
%% INFORMAÇÂO ALEATÓRIA
m = randi([0 1],1,4);
xlswrite(Experimento1, {num2str(m)},1,strcat('A',indice));

%% Palavra-código (o codificador de Hamming)

n = mod(m*G,2);

%%     ## Hamming + método do bit de paridade  extra  ##
if mod(sum(n),2)== 0
    n = [0 n];
else
    n = [1 n];
end
xlswrite(Experimento1, {num2str(n)},1,strcat('B',indice));

%% Inserção do erro
erro1 = randi([1 8],1);
n(erro1) = ~n(erro1);
xlswrite(Experimento1, {num2str(erro1)},1,strcat('C',indice));
%% SEGUNDO ERRO ***************
 erro2 = randi([1 8],1);
 if(erro2 == erro1 && erro1<8)
  erro2 = erro2 + 1;
end
if(erro2 == erro1 && erro1==8)
    erro2 = erro2 -1;
end

 n(erro2) = ~n(erro2);
 xlswrite(Experimento1, {num2str(erro2)},1,strcat('D',indice));

% Palavara-codigo com erro
 xlswrite(Experimento1, {num2str(n)},1,strcat('E',indice));


% calculo do bit extra
%p0 = n(1)
pch = mod(sum(n),2);
n(1) = [];


%% Decoder. Calculo da "sindrome"
sind = mod(n*H',2);
sind = flip(sind);
```

```
%% Verificação da quantidade de erros, e em caso de 1 erro simples,
localização e correção do mesmo

if (sum(sind)~= 0 && pch == 1)
       %% Existe um erro na posição:
       % converte a posição em binário para decimal
        posicao = sind(1)*2^2 + sind(2)*2^1 + sind(3)*2^0;

        %% Correção (inverte o bit na posição identificada
        n(posicao)=~n(posicao);

        %% Informação original
        m = [n(3) n(5) n(6) n(7)];
        xlswrite(Experimento1, {num2str(m)},1,strcat('F',indice));
    elseif (sum(sind)== 0 && pch == 1)

        %% Existe um Erro no bit extra
        m = [n(3) n(5) n(6) n(7)];
        xlswrite(Experimento1, {num2str(m)},1,strcat('F',indice));
        xlswrite(Experimento1, {'Detectado'},1,strcat('J',indice));
    elseif (sum(sind)== 0 && pch == 0)
        %% Não há erros na informação
        m = [n(3) n(5) n(6) n(7)];
        xlswrite(Experimento1, {num2str(m)},1,strcat('F',indice));
    elseif (sum(sind)~= 0 && pch == 0)

      %% "Há dois erros na informação
       xlswrite(Experimento1, {'Descartar'},1,strcat('F',indice));

       %% **********  TENTATIVA DE CORREÇAO **********
        %% "Há dois erros na informação
        posicao = sind(1)*2^2 + sind(2)*2^1 + sind(3)*2^0;

        %% Correção (inverte o bit na posição identificada
        n(posicao)=~n(posicao);
        m = [n(3) n(5) n(6) n(7)];
        xlswrite(Experimento1, {num2str(m)},1,strcat('H',indice));
        xlswrite(Experimento1, {'Detectado'},1,strcat('J',indice));
end


end
```