UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

FELIPE GASPAR ALAN E SILVA

UM CÓDIGO EXTENSÍVEL PARA CORREÇÃO DE MULTIPLE BIT UPSETS EM MEMÓRIAS

FORTALEZA

2018

FELIPE GASPAR ALAN E SILVA

UM CÓDIGO EXTENSÍVEL PARA CORREÇÃO DE MULTIPLE BIT UPSETS EM MEMÓRIAS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Mestre em Engenharia de Teleinformática. Área de concentração: Sinais e Sistemas.

Orientador: Prof. Dr. Jarbas Aryel Nunes da Silveira.
Coorientador: Prof. Dr. Walter da Cruz Freitas Junior.

FORTALEZA
2018

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)

S58c Silva, Felipe Gaspar Alan e.
Um Código Extensível para Correção de Multiple Bit Upsets em Memórias / Felipe Gaspar Alan e Silva. – 2018.
79 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2018.
Orientação: Prof. Dr. Jarbas Aryel Nunes da Silveira.
Coorientação: Prof. Dr. Walter da Cruz Freitas Junior.

1. Códigos Corretores de Erros. 2. Memórias. 3. Multiple Bit Upsets (MBU). 4. Tolerância a falhas. I. Título.

CDD 621.38

FELIPE GASPAR ALAN E SILVA

UM CÓDIGO EXTENSÍVEL PARA CORREÇÃO DE MULTIPLE BIT UPSETS EM MEMÓRIAS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Mestre em Engenharia de Teleinformática. Área de concentração: Sinais e Sistemas.

Aprovada em: 29/06/2018.

BANCA EXAMINADORA

_____

Prof. Dr. Jarbas Aryel Nunes da Silveira (Orientador)

Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Walter da Cruz Freitas Junior (Coorientador)

Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Danielo Gonçalves Gomes

Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Fabian Luis Vargas

Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS)

_____

Prof. Dr. Fernando Gehm Moraes

Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS)

AGRADECIMENTOS

Primeiramente a Deus, por ter me proporcionado a vida e ter me guiado durante toda a minha vida.

À minha família pelo apoio, especialmente: aos meus pais, João e Christina, pelo amor e esforço com que me criaram e me permitiram estar onde estou hoje, e às minhas irmãs Juliana e Marília pelo amor, amizade e por terem sido minhas primeiras professoras.

Ao professor Walter, pela oportunidade e confiança na realização do mestrado.

Ao professor Jarbas, pela amizade e comprometimento com que se dedicou para a realização do Mestrado e dos trabalhos desenvolvidos.

Aos professores César Marcon e Fabian Vargas da Pontifícia Universidade Católica do Rio Grande do Sul, pelas colaborações e conselhos que tanto contribuíram para este trabalho.

Aos professores participantes da banca, pelo tempo e sugestões providas a este trabalho.

Aos meus amigos e colegas de laboratório, pela convivência e apoio nos momentos mais difíceis.

Finalmente a CAPES, pelo apoio financeiro com a manutenção da bolsa de auxílio.

“O mundo não é um mar de rosas; é um lugar sujo, um lugar cruel, que não quer saber o quanto você é durão. Vai botar você de joelhos e você vai ficar de joelhos para sempre se você deixar. Você, eu, ninguém vai bater tão forte como a vida, mas não se trata de bater forte. Se trata de quanto você aguenta apanhar e seguir em frente, o quanto você é capaz de aguentar e continuar tentando. É assim que se consegue vencer.” Rocky Balboa.

RESUMO

A diminuição em escala dos dispositivos eletrônicos, aliado ao aumento do seu processamento, deixou-os mais sensíveis a efeitos de radiação oriundos, principalmente, do espaço. Memórias são circuitos altamente suscetíveis a esses efeitos, principalmente, em ambientes com alta incidência de radiação, culminando na grande possibilidade de apresentar erros em múltiplos bits (MBU), e assim provocar falhas críticas. Códigos Corretores de Erros (CCE) são bastante aplicados para lidar com a ocorrência de erros desse tipo. No entanto, a aplicação de um CCE com grande capacidade corretiva pode acarretar um aumento de custo de área, potência e atraso em um sistema, e assim perda de desempenho. O Código CLC é um CCE que utiliza conceitos de Hamming Estendido e paridade formando uma matriz capaz de lidar com MBU em memórias, além apresentando baixo custo de implementação quando comparado com códigos mais robustos. Apesar disso, o código apresenta pontos fracos em relação aos seus concorrentes. Um deles é o número de bits de redundância gerados, que para uma palavra de 16 bits de dados, necessita 150% de bits a mais da palavra original. O código CLC não apresentou grande capacidade de corrigir padrões muito agressivos de MBU (6, 7 e 8 erros), que com a crescente miniaturização dos circuitos integrados, a probabilidade de ocorrer esses tipos de erros tornou-se uma preocupação real em ambientes com grande incidência de radiação. Esse trabalho propõe a extensão do código CLC, de modo a tratar os problemas citados através de alterações na matriz do código CLC, pela modificação do Hamming Estendido aplicado. Além disso, foi avaliada a escalabilidade do código para maiores palavras de bits de dados. Pela modificação do Hamming Estendido aplicado, obtiveram-se duas novas estruturas para o CLC: uma que apresenta menor adição de bits de redundância e menor custo de implementação, e outra que apresenta maior capacidade de detecção e correção de erros. A análise de escalabilidade mostrou que para maiores palavras de dados, a taxa de acréscimo de bits gerados pelo CLC diminui, principalmente, para uma das estruturas propostas, caindo de 143,75% para 16 bits de dados, para 82,81% para 64 bits de dados. Por fim, este trabalho também realizou testes de confiabilidade e síntese, em que o desenvolvimento das novas estruturas para o CLC apresentou melhorias substanciais, logo trouxe maior flexibilidade de aplicação para este código.

Palavras-Chave: Códigos Corretores de Erro, Memórias, *Multiple Bit Upset* (MBU), Tolerância a Falhas

ABSTRACT

The scaling down of electronic devices along with the processing increase, has made them more sensitive to the effects of radiation coming mainly from space. Memories are circuits highly susceptible to these effects, having great possibility of presenting errors in multiple bits(MBU), which can cause critical failures in the system. Error Correcting Codes(ECCs) are widely applied to deal with the occurrence of such errors. However, the application of an ECC with large corrective capacity can lead to increased area, power and delay cost, causing loss of performance. The CLC code is na ECC that uses concepts of Extended Hamming and parity code, forming a matrix capable of handling MBU in memories, though presenting low implementation cost when compared to more robust codes. Despite this, the code has weaknesses compered to its competitors. One is the number of redundancy bits generated, which for a 16-bit word of data, requires 150% more bits of the original word. The CLC code did not presented good capability to correct very aggressive MBU patterns (6, 7 and 8 errors), that with the increasing miniaturization of integrated circuits, the probability of these types of errors occurring became a real concern in environments with large radiation incidence. This work proposes to treat the problems mentioned through changes in the matrix of the CLC code, by the modification of the Extended Hamming applied. In addition, we evaluated the scalability of the code for larger data bits words. The proposed Extended Hamming modification brought two new structures for the CLC code, one that has less redundancy bit addition and lower implementation cost, and another that has bigger error detection and correction capabilities. The scalability analysis shows that for larger data words, the bit rate generated by the CLC decreases, mainly for one of the proposed structure, falling from 143,75 % for 16-bit data, to 82,81% for 64-bit data. Finally, this work also performed tests of reliability and synthesis, in which the development of the new structures for the CLC presented substantial improvements, bringing bigger flexibility of application for this code.

Keywords: Error Correction Codes (ECCs), Memory device, Multiple Bit Upset (MBU), Fault Tolerance

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Os avanços em Microeletrônica têm proporcionado o desenvolvimento de circuitos integrados (CI´s) cada vez mais eficazes, que são caracterizados por altas frequências de operação e baixos níveis de tensão, além de estarem presentes no desenvolvimento de diversos sistemas eletrônicos (HAZUCHA; SVENSSON, 2000). Esses circuitos têm sido amplamente utilizados no desenvolvimento de sistemas críticos como: missões espaciais (FERREYRA et al., 2005) e (SAKATA et al.,2007) e dispositivos médicos (GURUN et al., 2012). No entanto, o alto nível de integração dos CI's os tornou suscetíveis a efeitos causados por partículas carregadas (e.g. nêutrons e elétrons) (KING et al, 2013). Um dos principais efeitos, que podem provocar falhas em circuitos integrados, são os *Single Event Effects* (SEE). SEEs tornaram-se uma preocupação constante em aplicações espaciais, principalmente em memórias SRAMs (KOHLER et al, 2018), sendo os efeitos mais corriqueiros nestes dispositivos: *Single Bit Upset* (SBU) e *Multiple Bit Upset* (MBU). SBU e MBU são, respectivamente, a alteração de um único bit e a alteração de múltiplos bits armazenados em memórias a partir de uma única partícula carregada. A ocorrência desses eventos pode trazer consequências catastróficas em aplicações críticas (LABEL et al., 2000).

1.1 Motivação

Códigos corretores de erros (CCEs) do formato *Single Error Correction - Double Error Detection* (SEC-DED), como o código de Hamming (Ham), são bastante utilizados para lidar com SBU, no entanto não conseguem corrigir múltiplos erros (GHERMAN et al., 2011), pois, à medida que o nível de integração dos circuitos integrados cresce, a ocorrência de MBU em memórias torna-se mais frequente (CHUGG; MOUTRIE; JONES, 2004) e (RADAELLI et al.,2005). Uma das técnicas que pode ser utilizada para tratar MBU é o *interleaving* (YUANYUAN; XUNYING, 2013), que consiste em reposicionar a informação em regiões diferentes de memória, evitando a ocorrência de múltiplos erros em dados que pertencem à mesma mensagem armazenada. No entanto, essa técnica pode acarretar um atraso considerável nos processos de escrita e de leitura de memórias, o que traz grande perda de performance para o sistema (DUTTA; TOUBA, 2007). Para lidar com esse problema, torna-se necessário a utilização de CCEs mais robustos. Todavia, dependendo do código aplicado, o custo agregado de implementação pode comprometer consideravelmente os recursos do sistema

computacional. Aliar alta taxa de detecção e correção de erros com o menor custo possível é algo fundamental em aplicações críticas, como missões espaciais (SAKATA et al., 2007).

Um CCE que pode ser utilizado para lidar com MBU em memórias é o código Matrix (ARGYRIDES; ZARANDI; PRADHAN, 2007). Matrix usa os conceitos de código de Hamming e Paridade ao formar um código de estrutura matricial. Outro CCE bastante conhecido é o código Reed-Muller, que faz parte de uma família de códigos de complexidade maior que o Matrix e que foram um dos primeiros a ser utilizados em missões espaciais (VARGHESE et al., 2013), sendo ainda presente em muitos sistemas digitais. O código Reed-Muller (2,5) apresenta maior eficácia para correção de MBU que o código Matrix, entretanto sua complexidade o torna mais dispendioso.

Outro CCE é o Código Linha Coluna (CLC)(SILVA et al., 2018) que foi desenvolvido com o intuito de combinar alta capacidade de detecção e correção de erros com baixo custo de implementação. O CLC utiliza código de Hamming Estendido ($Ham - E$) e Paridade para formar uma matriz, similar ao código Matrix; e possui dois modos de correção: Padrão e Estendido. O primeiro aplica o algoritmo de correção padrão do CLC, o segundo estende esse algoritmo possibilitando a correção de mais tipos de MBU, porém com maior custo de síntese que o primeiro. Apesar dos resultados apresentados pelo CLC terem mostrado sua eficiência para correção de MBU, o código apresenta algumas deficiências em relação às técnicas mais modernos. O código CLC para 16 bits gera um total de 40 bits, o que representa o acréscimo de 150% em relação à informação original, algo bastante complicado para a sua implementação em memórias. Outro ponto que precisa ser melhorado é a sua capacidade de correção para padrões de MBUs mais agressivos. Os resultados do CLC mostram que o código, em seu modo padrão de correção, apresentou deficiência em corrigir cenários de 6,7 e 8 erros (CASTRO et al, 2016).

Como soluções para os problemas descritos, este trabalho propõe a extensão da forma de aplicação do código CLC, descrevendo novas estruturas matriciais desenvolvidas pela aplicação de diferentes tipos de $Ham - E$. Além disso, foi realizado um estudo da escalabilidade do código para palavras de dados de 16, 32 e 64 bits, visando avaliar o impacto dos novos formatos do CLC, em comparação ao formato original e aos códigos Reed-Muller(2,5) e Matrix em termos de confiabilidade e custo.

1.2 Objetivos

Dentre os objetivos propostos com esse trabalho, destacam-se:

- Desenvolver novas estruturas matriciais para o código CLC e, assim. conseguir reduzir o número de bits gerados pelo código, bem como o seu custo de síntese.
- Melhorar as taxas de detecção e correção do código para cenários mais agressivos de MBU.
- Avaliar as novas estruturas em comparação com o código CLC original, juntamente com os códigos Matrix e Reed-Muller(2,5), nos experimentos de: Injeção de Erros, Teste de confiabilidade e Implementação física, para palavras de dados de 16,32 e 64 bits.

1.3 Organização

Este trabalho estrutura-se da seguinte forma:

No capítulo 2, serão apresentados os conceitos básicos de SEEs a taxa de *Soft Error*, que ;é aplicada para determinar a sensibilidade de um CI a SEEs; bem como os principais tipos de SEEs que afetam memórias e as técnicas de proteção aplicadas para protegê-las. Dando contiuidade, o capítulo 3, apresentará exemplos de CCEs que podem ser utilizados para lidar com SBU e MBU em memórias. Já o capítulo 4 apresenta as novas estruturas propostas nesse trabalho, e a análise dos bits de redundância dos novos CLCs para palavras de dados de 16,32 e 64 bits. Em seguida, são descritos e discutidos, no capítulo 5, os experimentos realizados nesse trabalho para validar a proposta, além de serem também apresentados os resultados obtidos na comparação para palavras de dados de 16, 32 e 64 bits. Por fim, o capítulo 6 apresenta as principais conclusões e os trabalhos futuros.

2 *SINGLE EVENT EFFECTS (SEE)*

A miniaturização dos circuitos integrados proporcionou a evolução de diversos dispositivos eletrônicos, capazes de operar em alta frequência e baixos níveis de tensão. No entanto, isso também os tornou muito mais suscetíveis a falhas transientes e permanentes. Em aplicações críticas, como missões espaciais e sistemas de comunicação de alta performance, dispositivos de memória integrados em subsistemas tornaram-se mais propensos aos efeitos de raios cósmicos, partículas-alpha e interferência eletromagnética (RADAELLI et al.,2005).

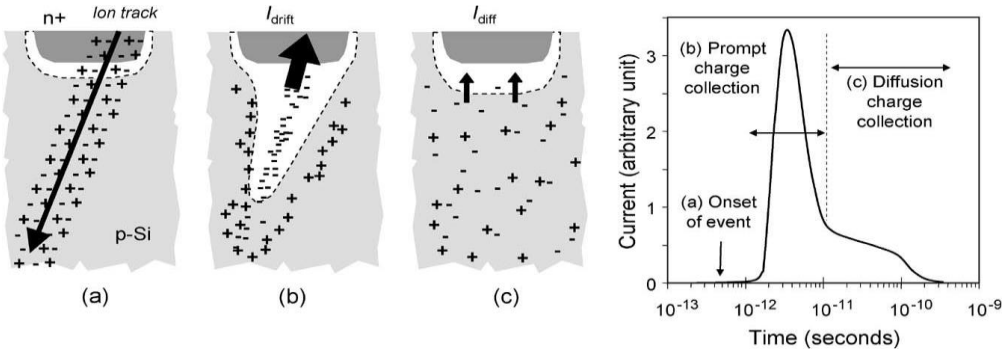
Os tipos de efeitos de radiação que mais afetam dispositivos eletrônicos são: *Total Ionizing Dose* (TID) e *Single Event Effect* (SEE). TID é o efeito que causa a degradação dos circuitos integrados devido ao acúmulo de cargas, podendo ser causada por fótons altamente carregados (i.e. prótons, elétrons e *heavy ions*) (BARNABY, 2006), e assim provocar falha permanente do dispositivo. SEEs são efeitos causados pelo impacto de uma única partícula energizada que podem acarretar falhas transientes ou permanentes (HEIJMEN, 2011).

Neste capítulo, são apresentados conceitos de SEEs que são os principais responsáveis na ocorrência de falhas em memórias aplicadas em ambientes críticos. São apresentados também seus tipos, os efeitos que causam nos dispositivos, incluindo os que afetam memórias e os métodos de proteção aplicados.

2.1 SEE: Conceitos básicos

SEEs são efeitos causados pelo impacto de uma partícula energizada nos circuitos integrados, podendo danificá-los seriamente. A Figura 2.1 mostra o efeito que o impacto de uma partícula carregada pode causar em um transistor.

Figura 2.1 – Efeito causado pelo impacto de uma partícula carregada em um transistor.



Fonte: (BAUMANN, 2005)

Na figura acima, é possível observar que a partícula carregada, ao atravessar a junção, produz um túnel de pares de elétrons buraco (Figura 2.1a). Quando a ionização resultante chega próxima à região de depleção, o desequilíbrio de cargas induz a criação de um funil temporário, em que elas são rapidamente coletadas pelo campo elétrico, formando uma alta corrente no nodo (Figura 2.1b). Em seguida, o funil é desfeito e os íons restantes são equilibrados pelo processo de difusão (Figura 2.1c). A Figura 2.1d mostra graficamente a variação da corrente apresentada, reflexo das figuras iniciais. O pico de corrente ocasionou a alteração do valor de tensão armazenado no transistor, porém uma partícula muito carregada poderá provocar falha terminal do componente.

Os efeitos que causam o mau funcionamento permanente dos CIs são chamados de *Hard Errors* e os que causam o mau funcionamento temporário são chamados de *Soft Errors* (HEIJMEN, 2011). *Soft Errors* podem ser classificados da seguinte forma:

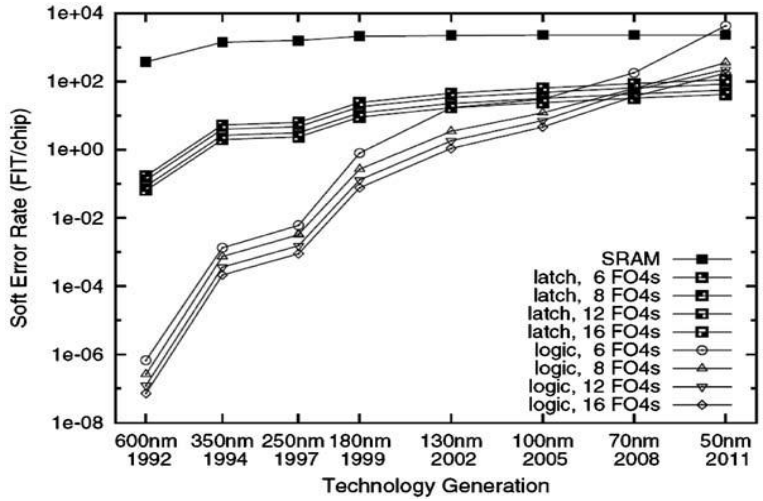
- *Single Event Transient* (SET). Esse evento causa um pico momentâneo de voltagem em um nodo do circuito. Se o valor for armazenado em uma memória, ocorrerá inversão do valor de tensão, caso contrário o sistema voltará a operar normalmente.
- *Single Event Upset*. Esse evento pode aparecer principalmente em 3 formas:
    - *Single Bit Upset* (SBU). É a inversão de um bit armazenado em uma célula de memória.
    - *Multiple Bit Upset* (MBU). É a inversão da informação armazenada em múltiplas células de memória.
    - *Single Event Functional Interrupt* (SEFI). É a interrupção da funcionalidade de um componente por *reset*, travamento ou outro meio detectável, porém não é necessário reiniciá-lo para restaurar a operabilidade.

Um exemplo de *Hard Error* causado por SEE é o *Single Event Burnout* (SEB), em que a partícula gera uma corrente de ruptura tão grande que causará uma falha catastrófica. A ocorrência de SEEs em componentes eletrônicos pode ser observada ao nível do mar, ao nível atmosférico e ao nível espacial. No entanto, por muito tempo os efeitos da radiação foram considerados problemas de confiabilidade apenas para aplicações espaciais. Em meados dos anos de 1990, mais trabalhos envolvendo efeitos de partículas de energia em dispositivos aos níveis do mar e atmosférico começaram a despontar (NORMAND, 2004) (HEIJMEN, 2011). Experimentos envolvendo memórias DRAM, mostraram que mesmo ao nível do mar, há presença significativa de raios cósmicos capazes de causar falhas em memórias (O'GORMAN,1994).

2.2 *Soft Error Rate* (SER)

SER é a taxa que um dispositivo ou sistema tem de possibilidade ou previsão de sofrer *Soft Errors.* Essa taxa é medida em unidades FIT, que significa *failures in time*, em que 1 FIT representa uma falha por bilhão de horas (aproximadamente 114,077 anos) de funcionamento do dispositivo. A faixa padrão para dispositivos eletrônicos é entre 100 e 100.000 FIT (HEIJMEN, 2011). Com a miniaturização dos circuitos integrados, a SER dos dispositivos cresceu de forma tão impactante que chips com SER acima de 50.000 FIT tornaram-se bastante comuns (BAUMANN, 2002). A Figura 2.2 apresenta o crescimento da SER para lógicas diversas.

Figura 2.2 – SER por tecnologia aplicada



Fonte: (GARG; KHATRI, 2009)

O crescimento agressivo da SER de circuitos combinacionais justifica-se pela redução da tecnologia aplicada, que resultou na criação de circuitos altamente interligados, melhorando a capacidade de processamento, porém tornando-os mais sensíveis a falhas.

Memórias, como as SRAM, desde tecnologias de maior escala já apresentavam uma SER bastante elevada. Apesar da aparente não alternância na SER dessas memórias, aplicações atuais demandam grandes quantidades desse dispositivo para realizar diversas operações, e assim potencializar a ocorrência de falhas. A próxima seção apresenta os principais tipos de SEE que podem ocorrer em memórias.

2.3 SEEs em memórias: SBU e MBU

Como apresentado na subseção anterior, a ocorrência de SEEs tornou-se uma preocupação real para circuitos integrados. Hoje, SEEs estão presentes em diversos ambientes de aplicação, cujas memórias são um dos componentes mais sensíveis à ocorrência desses efeitos; Caso esse ocorre, principalmente em SEU, uma vez que o impacto de uma única partícula carregada pode provocar falhas em todo o sistema. Nesta seção, serão apresentados os principais SEEs que afetam memórias em aplicações críticas.

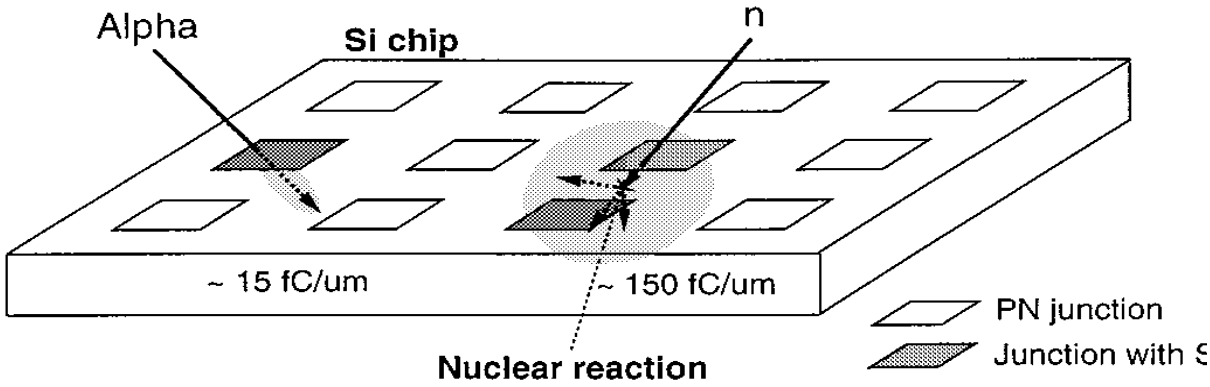
2.3.1 SBU e MBU

Como explanado na sessão 2.1, um dos tipos de SEE é o SEU, definido como evento provocado pelo impacto de uma partícula carregada no circuito integrado. Em memórias, os SEUs que predominam são: SBU e MBU.

A ocorrência de SBU em memórias é um problema já observado desde as primeiras aplicações críticas envolvendo esses dispositivos (GUENZER; CAMPBELL; SHAPIRO et al, 1981). Ele se caracteriza pela inversão de um bit (valor de tensão) armazenado em uma célula de memória. Esse foi um dos principais objetos de estudo em aplicações espaciais, inclusive no desenvolvimento de componentes totalmente tolerantes a SEU (BROWNING; KOGA; KOLASINSKI, 1985).

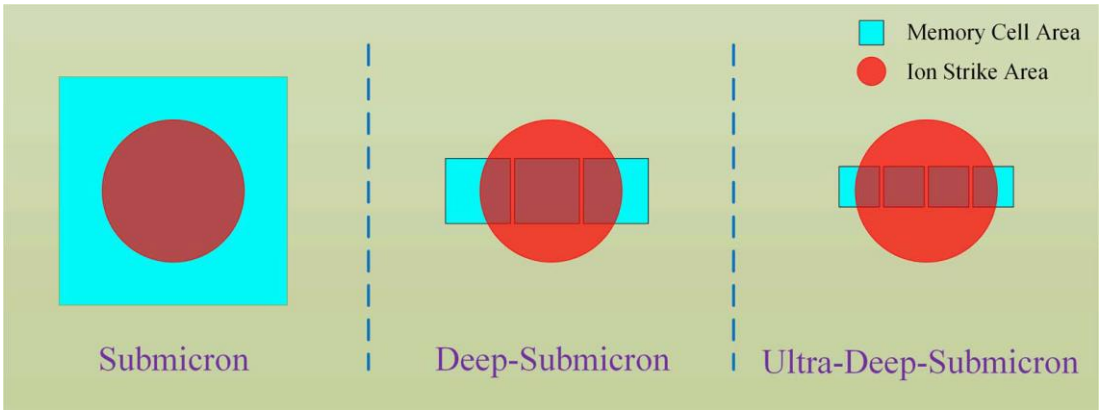
Com a diminuição da tecnologia aplicada nos circuitos integrados, uma única partícula de carga passou a afetar um número maior de células fisicamente próximas, como mostrado na Figura 2.3 e Figura 2.4.

Figura 2.3 – MBU induzido por uma partícula carregada



Fonte: (SATOH; TOSAKA; WENDER, 2000)

Figura 2.4 – Evolução do MBU para diminuição de tecnologia



Fonte: (LI et al, 2018)

Com a crescente ocorrência de MBU, estudos foram realizados com objetivo de identificar os principais padrões de células afetadas durante o choque de uma partícula carregada (RADAELLI, 2005). Kohler realizou experimentos em memórias DDR3 e DDR3L de quatro fabricantes diferentes, em que foi observada a ocorrência de SEUs nesses dispositivos com padrões similares (KOHLER et al, 2018).

Em outro caso, (OGDEN e MASCAGNI, 2017) desenvolveram simuladores de injeção de MBUs para memórias SRAM de 45nm. Os padrões de erros desses simuladores foram inspirados no experimento desenvolvido por (RAO et al, 2014), em que se observou a ocorrência de erros múltiplos em memórias com células adjacentes. Com isso, diversas técnicas foram desenvolvidas ao longo dos anos para prover confiabilidade à informação de memórias.

2.4 Técnicas para proteção de memórias contra SEE

Aplicações críticas, como as militares e as espaciais, são sistemas que demandam grande quantidade de processamento e agilidade na realização e troca de processos. Aplicações como essas utilizam grandes quantidades de memórias, por serem cruciais para o pleno funcionamento do sistema, ao armazenar rotinas e informações fundamentais de diversos processos. Além disso, essas aplicações necessitam de alta confiabilidade de operação, de modo que sejam evitadas falhas críticas. Portanto, torna-se essencial proteger sistemas integrados de possíveis falhas ocasionadas por SEE. Esta seção apresenta algumas técnicas aplicadas para lidar com erros em memórias oriundos de SEE.

2.4.1 Blindagem Eletromagnética

A Blindagem Eletromagnética consiste em bloquear ou proteger um componente com um material condutivo ou magnético, de modo que as cargas se depositem nele, e não no componente que se almeja proteger.

A aplicação de blindagem em missões espaciais foi objeto de estudo na década de 1990 (SMITH, 1994), no qual foi observado que, para uma diminuição efetiva de SEE, seria necessária uma blindagem robusta. Além disso, outra variável para a escolha da blindagem é o peso que ela irá acrescentar para todo o sistema, uma vez que missões espaciais possuem limites rigorosos de peso (FAN et al, 1996).

Mesmo com o aumento da espessura da blindagem, a miniaturização dos circuitos eletrônicos tornou essa técnica pouco eficiente para a proteção contra SEU e MBUs, pois pode ser causada por nêutrons que passariam facilmente pelo condutor (ARGYRIDES et al, 2010).

2.4.2 Aplicação de Interleaving

A aplicação de *interleaving* em memórias é uma técnica que consiste no embaralhamento dos endereços de memória, a fim de evitar a ocorrência de erros adjacentes na mesma palavra ou até mesmo múltiplos em uma mesma região de memória.

Figura 2.5 – Procedimento de *Inteleaving*

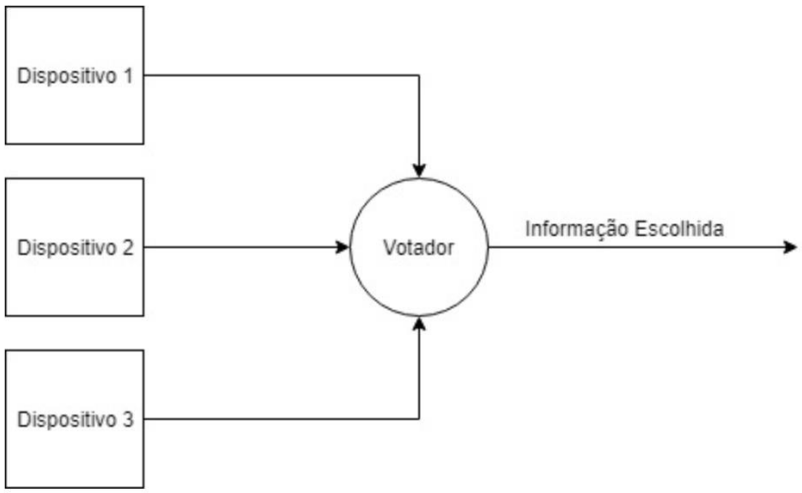
Fonte: Elaborada pelo Autor.

A partir de estudos sobre como os padrões de múltiplos erros se apresentam em memórias (SATOH; TOSAKA; WENDER, 2000), (RADAELLI, 2005), *interleaving* passou a ser uma técnica aplicada para corrigi-los em combinação com outros CCEs (LAI; CHEN; CHIOU, 2010).

2.4.3 Redundância Modular Tripla (RMT)

A Redundância Modular Tripla (RMT) é uma técnica de tolerância a falhas aplicada em circuitos eletrônicos. A RMT consiste em replicar um componente eletrônico e sujeitá-los a um votador que irá decidir qual dos dispositivos está repassando as informações corretas.

Figura 2.6 – Ilustração do RMT



Fonte: Elaborada pelo Autor.

Na figura acima, é possível observar que o votador irá transmitir somente a informação que estiver em maioria. Com a aplicação de um RMT se pode garantir a confiabilidade da informação, desde que haja mais de um registrador afetado por uma falha transiente no mesmo instante da votação ou de modo permanente. Porém, a implementação do RMT pode acrescentar de maneira significativa custo tanto físico, quanto financeiro de um projeto.

2.4.4 Códigos Corretores de Erros (CCE)

Por fim, uma das técnicas mais aplicadas em sistemas tolerantes a falhas são os Códigos Corretores de Erros (CCE). Esses códigos se caracterizam por, através de operações lógicas, detectar e corrigir erros na informação em memórias. Comparado a outros métodos, como blindagem ou RMT, os CCEs apresentam custo de aplicação inferior (CHEN; HSIAO, 1984). Antes de ter sido utilizada para proteção de informação em memórias, os CCEs já eram

aplicados em diversos sistemas de transmissão para aplicações críticas, como em satélites (VARGHESE et al., 2013).

Apesar das vantagens apresentadas pelos CCEs, códigos que têm a capacidade de corrigir múltiplos erros; ou que apresentam lógica muito complexa, isto é, que necessitam de muitos recursos de computação para operar, geralmente apresentam impactos negativos em área, potência e atraso, logo podem prejudicar o desempenho do sistema em aplicações de recursos limitados, ou com restrições temporais (CARDARILLI et al, 2005). No próximo capítulo, serão apresentados CCEs que serviram de base de estudo para o desenvolvimento deste trabalho.

3 EXEMPLOS DE CÓDIGOS CORRETORES DE ERROS

Códigos corretores de erros são códigos utilizados para detectar e corrigir eventuais erros que podem ter ocorrido na informação salva em memórias. Esses algoritmos têm sido amplamente utilizados em vários campos tecnológicos como: sistemas de comunicação, aplicações espaciais, gravação digital, entre outros. A utilização de códigos corretores de erros em sistemas embarcados tem ganhado cada vez mais espaço devido ao baixo custo de implementação, quando comparado com outros métodos para tornar sistemas tolerantes a falhas.

3.1 Paridade

A Paridade é uma técnica utilizada para detecção de erro simples. Consiste em adicionar um bit, cujo o cálculo da operação é XOR ($\oplus$), entre os elementos da mensagem binária. Tomando como exemplo $M = [1\ 0\ 0\ 0]$, temos:

- Para M, o bit adicional será calculado da seguinte forma: $1 \oplus 0 \oplus 0 \oplus 0 = 1$. A mensagem codificada será $1000\mathbf{1}$.

A verificação de ocorrência de erros é feita através do recálculo do bit de paridade. Em seguida, o bit recalculado será operado com o bit de paridade armazenado. Caso o valor da operação seja não nulo, significará que um erro foi detectado.

3.2 Código de Hamming

O Código de Hamming $Ham\ (M, N)$, desenvolvido por Richard Wesley Hamming, foi um dos primeiros CCEs desenvolvidos para aplicação em sistemas computacionais, com o objetivo de corrigir erros simples (HAMMING, 1950). A seguir são apresentados fundamentos que caracterizam um código $Ham\ (M, N)$

- M representa o número de bits de dados e N o número de bits totais.
- N é definida pela equação $N = M + k$, em que k é o número de bits de Hamming.

- A distância do código de Hamming é 3, o que significa que ele detecta e corrige um erro.
- Um código de Hamming precisa respeitar as seguintes relações apresentadas em 3.1 e 3.2.

$$2^k \geq k + M + 1 \qquad (3.1)$$

$$2^M \geq \frac{2^N}{N+1} \qquad (3.2)$$

A seguir, será descrito o desenvolvimento do $Ham\,(4{,}7)$ baseado em (NEUBAUER, FREUDENBERGER e KÜHN, 2007), (MOREIRA; FARRELL, 2006).

3.2.1 Codificação de $Ham\,(4,7)$

A codificação do código $Ham\,(4{,}7)$ começa com o cálculo da sua matriz geradora G que é encontrada a partir da Equação descrita em 3.3

$$G = [\, I_{2^k-k-1}\; Q] \qquad (3.3)$$

Onde I_{2^k-k-1} é a matriz identidade quadrada de ordem $2^k - k - 1$, Q é matriz formada por $2^k - k - 1$ colunas com dois ou mais vetores de peso dois ou maior, e k é o número de bits de redundância, no caso 3 (MOREIRA; FARRELL, 2006). Os possíveis vetores de bits de Hamming do código $Ham\,(4{,}7)$ são: 000, 001, 010, 011, 100, 101, 110, 111. Os quatro vetores colunas escolhidos para formar a matriz Q serão: 011, 101, 110, 111.

$$G(4{,}7) = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

A partir do momento em que se foi calculada a matriz G, o processo de codificação para o código $Ham\ (4,7)$ já pode ser realizado. O vetor codificado *N(1,7)* é obtido a partir do produto vetorial entre o vetor de *M(1,4)* e a matriz geradora *G(4,7)*.

$$N = M\ x\ G \qquad (3.4)$$

Usando um exemplo simples, temos para um $M = [1\ 0\ 0\ 0]$, a seguinte palavra codificada:

$$N = 1\,0\,0\,0 \quad x \quad \begin{matrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

$$N = [1\ 0\ 0\ 0\ 0\ 1\ 1]$$

3.2.2 Decodificação de $Ham(4,7)$

O processo de Decodificação consiste em verificar se houve erros na mensagem transmitida. A equação 3.6 descreve o cálculo de síndrome, cujo produto apresenta-se entre matriz de paridade H, com expressão em 3.5; e a mensagem codificada N :

$$H = [Q^T\ I_k] \qquad (3.5)$$

Com isso, a matriz H será:

$$H = \begin{matrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{matrix}$$

$$M_{Decodificada} = H\ x\ N^T \qquad (3.6)$$

Se o resultado desse produto for um vetor nulo, então a mensagem transmitida não detectou erros. Caso contrário, a mensagem N apresenta erro, e o valor da síndrome representará

um dos vetores de correção (Vc) mostrados na Tabela 3.1. A correção é feita pela operação *Xor* entre a mensagem errada e o Vc encontrado. Após a correção do erro, realizada pela Equação (3.7), a mensagem está pronta para ser transmitida.

Tabela 3.1 – Posição de correção por síndrome

Resultado do produto $H \ x \ N^T$	Vc
[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]

Fonte: Elaborada pelo Autor

$$Mcorreto = N \ XOR \ Vc \qquad (3.7)$$

3.2.3 Hamming Estendido

O código de Hamming Estendido $Ham - E(M, N+1)$ é formado ao adicionar um bit cujo valor obtêm-se a partir da paridade entre os bits da mensagem. A adição da paridade aumenta a sua distância de Hamming para 4, dessa forma o código irá corrigir um erro e será capaz de detectar dois erros. Os resultados de checagem do $Ham - E(M, N+1)$ podem apresentar os seguintes estados apresentados na Tabela 3.2.

Tabela 3.2 – Estados do $Ham - E(M, N+1)$

Síndrome Bits de Hamming	**Síndrome Paridade**	**Estado**
Sem Erro	Sem Erro	Sem Erro
Sem Erro	Erro	Erro Simples na Paridade
Erro	Sem Erro	Erro Simples
Erro	Erro	Erro Duplo

Fonte: Elaborada pelo Autor

Partindo do exemplo apresentado nas subseções anteriores, são feitas as seguintes considerações:

- Calculando a paridade dos sete bits de M, encontramos o bit de valor 1 que será adicionado ao final da mensagem M e formará a mensagem [1 0 0 0 0 1 1 1].
- Na ocorrência de 2 erros em M, por exemplo [**0 1** 0 0 0 1 1 1], ao calcular-se a síndrome da paridade apresentada na seção 3.1, veremos que o valor dará 0, Tal resultado inicialmente indica a ausência de erros, no entanto se a operação 3.6 for repetida para os sete primeiros bits de M, observa-se-á que o valor do produto será diferente de 0 e ambos os erros não poderão ser corrigidos

3.3 Código Matrix

O código Matrix (ARGYRIDES; ZARANDI; PRADHAN, 2007) é um código corretor de erros que utiliza $Ham(4,7)$ e Paridade para formar um código de estrutura matricial. como mostrado na Figura 3.1.

Figura 3.1 – Estrutura do código Matrix para 16 bits.

X_1	X_2	X_3	X_4	C_1	C_2	C_3
X_5	X_6	X_7	X_8	C_4	C_5	C_6
X_9	X_{10}	X_{11}	X_{12}	C_7	C_8	C_9
X_{13}	X_{14}	X_{15}	X_{16}	C_{10}	C_{11}	C_{12}
P_1	P_2	P_3	P_4			

Fonte: (Argyrides et, 2007).

Esse formato foi aplicado com o objetivo de corrigir padrões de erros múltiplos com custo de síntese menor que códigos mais robustos. Portanto o código utilizado neste trabalho é o Matrix(16,32).

3.3.1 Codificação Matrix(16,32)

O processo de codificação do Matrix(16,32) consiste na divisão dos 16 bits de dados para aplicação de $Ham(4,7)$, que irá compor as quatro primeiras linhas da matriz. A quinta

linha é formada pelo resultado do cálculo da paridade dos quatro primeiros bits das quatro primeiras colunas da matriz, formando a palavra codificada de 32 bits.

O símbolo X representa os bits de dados da palavra, os bits C são os de checagem gerados por codificação de Hamming e os bits P os de paridade. Ao tomar como exemplo a mensagem X = [1000 1010 0101 1111], a mensagem codificada resultante será:

1	0	0	0	0	1	1
1	0	1	0	1	0	1
0	1	0	1	0	1	0
1	1	1	1	1	1	1
1	0	0	0			

3.3.2 Decodificação Matrix(16,32)

Para a decodificação, são recalculados todos os bits de checagem C e paridade P, em seguida é realizada a síndrome com os bits originais apresentada nas equações 3.8 e 3.9

$$SC_i = C_i \oplus RC_i \tag{3.8}$$

$$SP_i = P_i \oplus RP_i \tag{3.9}$$

Em que, RC_i e RP_i são os valores de bit recalculados para cada um dos bits de checagem e de paridade respectivamente. A partir dos valores das síndromes, a equação *3.10* é aplicada para corrigir os bits errados.

$$X_{i\_correto} = (X_{i\_errado} \oplus O_i) \oplus (DED_j \cdot SP_i) \tag{3.10}$$

Onde, $X_{i\_correto}$ é o valor do bit corrigido, X_{i_errado} representa o bit que será corrigido, O_i é o elemento do vetor de correção de Hamming apresentado na Tabela 3.1, DED_j representa o bit de detecção de duplo erro na linha j, e S_p é a síndrome da paridade correspondente a coluna do bit corrigido.

3.4 Código Reed-Muller

Os códigos Reed-Muller foram uma das primeiras famílias de códigos a serem utilizadas em aplicações espaciais, em que é necessária alta cobertura na detecção e correção de erros.

Apesar de sua primeira aplicação em missões espaciais ter ocorrido há aproximadamente 40 anos, essa família de códigos continua sendo bastante utilizada em sistemas digitais (MOON, 2005).

Neste trabalho, o código utilizado foi o Reed-Muller (2,5). Um código Reed-Muller(2,5) é definido de acordo com:

- Sua ordem: r
- O tamanho da palavra codificada: 2^m
- Sua dimensão: $k(r,m) = 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r}$
- Sua distância mínima: $dmin = 2^{m-r}$.

3.4.1 Codificação Reed-Muller(2,5)

Para o código Reed-Muller(2,5), teremos tamanho igual a 32, $k(2,5)$ = 16 e $dmin$ = 8. Para calcular a matriz geradora $G_{RM}\ (r,m)$ partimos para a seguinte análise.

I. Para $1 \leq i \leq m$ teremos um vetor V_i seguindo a lógica:

$$V_i = [0\ldots0,1\ldots1,0\ldots0\ldots,1\ldots1]$$

Em que, V_i terá 2^{m-i+1} elementos de 0 e 2^{i-1} termos de um. Para o caso citado anteriormente, em que m = 5, teremos:

$$V_5 = 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1$$
$$V_4 = 0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1$$
$$V_3 = 0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1$$
$$V_2 = 0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1$$
$$V_1 = 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1$$

Temos também $V_0 = (1,1,\ldots,1)$.

II. Sendo $X = (x_0, x_1, x_2, \ldots, x_{n-1})$ e $Y = (y_0, y_1, y_2, \ldots, y_{n-1})$ dois vetores que contém *n* números binários, definimos a operação AND (&) entre os dois como:

$$XY = (x_0 \,\&\, y_0, x_1 \,\&\, y_1, x_2 \,\&\, y_2, \ldots, x_{n-1} \,\&\, y_{n-1})$$

III. Temos que a expressão geral para a matriz geradora $G_{RM}\ (r, m)$ é dada por:

$$G_{RM}\ (r, m) = (V_0, V_1, \ldots, V_m, V_1V_2, V_1V_3, \ldots, V_{m-1}V_m) \qquad (3.11)$$

Para o caso a seguir, temos que $G_{RM}(2, 5)$ é dada por:

$$G_{RM}(2,5) = (V_0, V_1, V_2, V_3, V_4, V_5, V_1V_2, V_1V_3, V_1V_4, V_1V_5, V_2V_3, V_2V_4, V_2V_5, V_3V_4, V_3V_5, V_4V_5) \longrightarrow$$

$$\begin{aligned}
V_0 &= 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\\
V_1 &= 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\\
V_2 &= 0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\\
V_3 &= 0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\\
V_4 &= 0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\\
V_5 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\\
V_1V_2 &= 0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1\\
V_1V_3 &= 0\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1\\
V_1V_4 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\\
V_1V_5 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\\
V_2V_3 &= 0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,1\,1\\
V_2V_4 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1\\
V_2V_5 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\\
V_3V_4 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\\
V_3V_5 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\\
V_4V_5 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1
\end{aligned}$$

Essa matriz geradora é formada pelo vetor unitário V_0, pelos vetores de primeira ordem (V_1, V_2, V_3, V_4, V_5) e pelos vetores de segunda ordem (V_1V_2, V_1V_3, V_1V_4, V_1V_5, V_2V_3, V_2V_4, V_2V_5, V_3V_4, V_3V_5, V_4V_5). Com isso, é possível codificar uma palavra A de 16 bits em uma B de 32 bits a partir da expressão:

$$B = A\ x\ G_{RM}(2,5) \qquad (3.12)$$

3.4.2 Decodificação Reed-Muller(2,5)

Para mostrar a decodificação do Reed-Muller, utilizaremos um exemplo arbitrário. Para o Reed-Muller(2,5), suponha que ($A_0, A_1, A_2, A_3, A_4, A_5, A_{12}, A_{13}, A_{14}, A_{15}, A_{23}, A_{24}, A_{25}, A_{34}, A_{35}, A_{45}$) é a mensagem que será codificada.

A palavra codificada resultante será $B = (B_0, B_1, B_2, ..., B_{31}) = (A_0V_0 + A_1V_1 + A_2V_2 + \ldots + A_{45}V_4V_5)$. Note que a soma dos primeiros quatro elementos de todas as linhas de vetores V, com exceção da linha V_1V_2, será 0. Isso também será verdade para os outros sete grupos de quatro elementos. Nesse caso obtêm-se as seguintes somas:

$$A_{12}(0) = B_0 \oplus B_1 \oplus B_2 \oplus B_3$$
$$A_{12}(1) = B_4 \oplus B_5 \oplus B_6 \oplus B_7$$
$$A_{12}(2) = B_8 \oplus B_9 \oplus B_{10} \oplus B_{11}$$
$$A_{12}(3) = B_{12} \oplus B_{13} \oplus B_{14} \oplus B_{15}$$
$$A_{12}(4) = B_{16} \oplus B_{17} \oplus B_{18} \oplus B_{19}$$
$$A_{12}(5) = B_{20} \oplus B_{21} \oplus B_{22} \oplus B_{23}$$
$$A_{12}(6) = B_{24} \oplus B_{25} \oplus B_{26} \oplus B_{27}$$
$$A_{12}(7) = B_{28} \oplus B_{29} \oplus B_{30} \oplus B_{31}$$

Essas somas fornecem oito formas independentes de reconstruir a informação de A_{12}. Caso um erro ocorra em um dos bits de B, esse bit irá afetar apenas uma das oito somas, dessa forma as outras sete darão o valor correto de A_{12} analisando qual valor está em maioria.

$$\boldsymbol{A_{12}} = \boldsymbol{maioria}(A_{12}(0), A_{12}(1), A_{12}(2), A_{12}(3), A_{12}(4), A_{12}(5), A_{12}(6), A_{12}(7))$$

Analogamente à expressão para o valor A_{12}, também é possível determinar as somas para os bits A_{13}, A_{14}, A_{15}, A_{23}, A_{24}, A_{25}, A_{34}, A_{35}, A_{45}.

$$A_{13}(0) = B_0 \oplus B_1 \oplus B_4 \oplus B_5$$
$$A_{13}(1) = B_2 \oplus B_3 \oplus B_6 \oplus B_7$$
$$A_{13}(2) = B_8 \oplus B_9 \oplus B_{12} \oplus B_{13}$$
$$A_{13}(3) = B_{10} \oplus B_{11} \oplus B_{14} \oplus B_{15}$$
$$A_{13}(4) = B_{16} \oplus B_{17} \oplus B_{20} \oplus B_{21}$$
$$A_{13}(5) = B_{18} \oplus B_{19} \oplus B_{22} \oplus B_{23}$$
$$A_{13}(6) = B_{24} \oplus B_{25} \oplus B_{28} \oplus B_{29}$$
$$A_{13}(7) = B_{26} \oplus B_{27} \oplus B_{30} \oplus B_{31}$$

$$A_{14}(0) = B_0 \oplus B_1 \oplus B_8 \oplus B_9$$
$$A_{14}(1) = B_2 \oplus B_3 \oplus B_{10} \oplus B_{11}$$
$$A_{14}(2) = B_4 \oplus B_5 \oplus B_{12} \oplus B_{13}$$
$$A_{14}(3) = B_6 \oplus B_7 \oplus B_{14} \oplus B_{15}$$
$$A_{14}(4) = B_{16} \oplus B_{17} \oplus B_{24} \oplus B_{25}$$
$$A_{14}(5) = B_{18} \oplus B_{19} \oplus B_{26} \oplus B_{27}$$
$$A_{14}(6) = B_{20} \oplus B_{21} \oplus B_{28} \oplus B_{29}$$
$$A_{14}(7) = B_{22} \oplus B_{23} \oplus B_{30} \oplus B_{31}$$

$$A_{15}(0) = B_0 \oplus B_1 \oplus B_{16} \oplus B_{17}$$
$$A_{15}(1) = B_2 \oplus B_3 \oplus B_{18} \oplus B_{19}$$
$$A_{15}(2) = B_4 \oplus B_5 \oplus B_{20} \oplus B_{21}$$
$$A_{15}(3) = B_6 \oplus B_7 \oplus B_{22} \oplus B_{23}$$
$$A_{15}(4) = B_8 \oplus B_9 \oplus B_{24} \oplus B_{25}$$
$$A_{15}(5) = B_{10} \oplus B_{11} \oplus B_{26} \oplus B_{27}$$
$$A_{15}(6) = B_{12} \oplus B_{13} \oplus B_{28} \oplus B_{29}$$
$$A_{15}(7) = B_{14} \oplus B_{15} \oplus B_{30} \oplus B_{31}$$

$$A_{23}(0) = B_0 \oplus B_2 \oplus B_4 \oplus B_6$$
$$A_{23}(1) = B_1 \oplus B_3 \oplus B_5 \oplus B_7$$
$$A_{23}(2) = B_8 \oplus B_{10} \oplus B_{12} \oplus B_{14}$$
$$A_{23}(3) = B_9 \oplus B_{11} \oplus B_{13} \oplus B_{15}$$
$$A_{23}(4) = B_{16} \oplus B_{18} \oplus B_{20} \oplus B_{22}$$
$$A_{23}(5) = B_{17} \oplus B_{19} \oplus B_{21} \oplus B_{23}$$
$$A_{23}(6) = B_{24} \oplus B_{26} \oplus B_{28} \oplus B_{30}$$
$$A_{23}(7) = B_{25} \oplus B_{27} \oplus B_{29} \oplus B_{31}$$

$$A_{24}(0) = B_0 \oplus B_2 \oplus B_8 \oplus B_{10}$$
$$A_{24}(1) = B_1 \oplus B_3 \oplus B_9 \oplus B_{11}$$
$$A_{24}(2) = B_4 \oplus B_6 \oplus B_{12} \oplus B_{14}$$
$$A_{24}(3) = B_5 \oplus B_7 \oplus B_{13} \oplus B_{15}$$
$$A_{24}(4) = B_{16} \oplus B_{18} \oplus B_{24} \oplus B_{26}$$
$$A_{24}(5) = B_{17} \oplus B_{19} \oplus B_{25} \oplus B_{27}$$
$$A_{24}(6) = B_{20} \oplus B_{22} \oplus B_{28} \oplus B_{30}$$
$$A_{24}(7) = B_{21} \oplus B_{23} \oplus B_{29} \oplus B_{31}$$

$$A_{25}(0) = B_0 \oplus B_2 \oplus B_{16} \oplus B_{18}$$
$$A_{25}(1) = B_1 \oplus B_3 \oplus B_{17} \oplus B_{19}$$
$$A_{25}(2) = B_4 \oplus B_6 \oplus B_{20} \oplus B_{22}$$
$$A_{25}(3) = B_5 \oplus B_7 \oplus B_{21} \oplus B_{23}$$
$$A_{25}(4) = B_8 \oplus B_{10} \oplus B_{24} \oplus B_{26}$$
$$A_{25}(5) = B_9 \oplus B_{11} \oplus B_{25} \oplus B_{27}$$
$$A_{25}(6) = B_{12} \oplus B_{14} \oplus B_{28} \oplus B_{30}$$
$$A_{25}(7) = B_{13} \oplus B_{15} \oplus B_{29} \oplus B_{31}$$

$$A_{34}(0) = B_0 \oplus B_4 \oplus B_8 \oplus B_{12}$$
$$A_{34}(1) = B_1 \oplus B_5 \oplus B_9 \oplus B_{13}$$
$$A_{34}(2) = B_2 \oplus B_6 \oplus B_{10} \oplus B_{14}$$
$$A_{34}(3) = B_3 \oplus B_7 \oplus B_{11} \oplus B_{15}$$
$$A_{34}(4) = B_{16} \oplus B_{20} \oplus B_{24} \oplus B_{28}$$
$$A_{34}(5) = B_{17} \oplus B_{21} \oplus B_{25} \oplus B_{29}$$
$$A_{34}(6) = B_{18} \oplus B_{22} \oplus B_{26} \oplus B_{30}$$
$$A_{34}(7) = B_{19} \oplus B_{23} \oplus B_{27} \oplus B_{31}$$

$$A_{35}(0) = B_0 \oplus B_4 \oplus B_{16} \oplus B_{20}$$
$$A_{35}(1) = B_1 \oplus B_5 \oplus B_{17} \oplus B_{21}$$
$$A_{35}(2) = B_2 \oplus B_6 \oplus B_{18} \oplus B_{22}$$
$$A_{35}(3) = B_3 \oplus B_7 \oplus B_{19} \oplus B_{23}$$
$$A_{35}(4) = B_8 \oplus B_{12} \oplus B_{24} \oplus B_{28}$$
$$A_{35}(5) = B_9 \oplus B_{13} \oplus B_{25} \oplus B_{29}$$
$$A_{35}(6) = B_{10} \oplus B_{14} \oplus B_{26} \oplus B_{30}$$
$$A_{35}(7) = B_{11} \oplus B_{15} \oplus B_{27} \oplus B_{31}$$

$$A_{45}(0) = B_0 \oplus B_8 \oplus B_{16} \oplus B_{24}$$
$$A_{45}(1) = B_1 \oplus B_9 \oplus B_{17} \oplus B_{25}$$
$$A_{45}(2) = B_2 \oplus B_{10} \oplus B_{18} \oplus B_{26}$$
$$A_{45}(3) = B_3 \oplus B_{11} \oplus B_{19} \oplus B_{27}$$
$$A_{45}(4) = B_4 \oplus B_{12} \oplus B_{20} \oplus B_{28}$$
$$A_{45}(5) = B_5 \oplus B_{13} \oplus B_{21} \oplus B_{29}$$
$$A_{45}(6) = B_6 \oplus B_{14} \oplus B_{22} \oplus B_{30}$$
$$A_{45}(7) = B_7 \oplus B_{15} \oplus B_{23} \oplus B_{31}$$

Juntando as expressões, podemos utilizar a mesma lógica de maioria aplicada em $\boldsymbol{A_{12}}$ para calcular a maioria de $\boldsymbol{A_{13}, A_{14}, A_{15}, A_{23}, A_{24}, A_{25}, A_{34}, A_{35}, A_{45}}$. Para calcular os bits decodificados a partir dos vetores de segunda ordem é utilizada parte da matriz $G_{RM}(2,5)$ formada apenas pelos componentes de segunda ordem. Com isso, são obtidos os bits decodificados para os vetores de segunda ordem (B') pela expressão:

$$B' = B - (\boldsymbol{A_{13}, A_{14}, A_{15}, A_{23}, A_{24}, A_{25}, A_{34}, A_{35}, A_{45}})\ x\ G_{RM}\ (2,5)'$$

Onde $G_{RM}\ (2,5)'$ representa a matriz:

$$G_{RM}\ (2,5)' = (V_1V_2, V_1V_3, V_1V_4, V_1V_5, V_2V_3, V_2V_4, V_2V_5, V_3V_4, V_3V_5, V_4V_5)$$

Repetindo o processo com o fito de encontrar as equações para os bits de primeira ordem da mensagem, encontramos as seguintes equações para A_1.

$A_1(0) = B'_0 \oplus B'_1$	$A_1(4) = B'_8 \oplus B'_9$	$A_1(8) = B'_{16} \oplus B'_{17}$	$A_1(12) = B'_{24} \oplus B'_{25}$
$A_1(1) = B'_2 \oplus B'_3$	$A_1(5) = B'_{10} \oplus B'_{11}$	$A_1(9) = B'_{18} \oplus B'_{19}$	$A_1(13) = B'_{26} \oplus B'_{27}$
$A_1(2) = B'_4 \oplus B'_5$	$A_1(6) = B'_{12} \oplus B'_{13}$	$A_1(10) = B'_{20} \oplus B'_{21}$	$A_1(14) = B'_{28} \oplus B'_{29}$
$A_1(3) = B'_6 \oplus B'_7$	$A_1(7) = B'_{14} \oplus B'_{15}$	$A_1(11) = B'_{22} \oplus B'_{23}$	$A_1(15) = B'_{30} \oplus B'_{31}$

A expressão de maioria para $\boldsymbol{A_1}$ será:

$$\boldsymbol{A_1} = \boldsymbol{maioria}(A_1(0), A_1(1), A_1(2), A_1(3), .., A_1(15))$$

De modo análogo à $\boldsymbol{A_1}$, os bits A_2, A_3, A_4 e A_5 são encontrados. Para os vetores de primeira ordem, é utilizada parte da matriz $G_{RM}(2,5)$ composta pelos componentes de primeira ordem. Assim, são obtidos os bits decodificados para os vetores de primeira ordem (B'') pela expressão:

$$B'' = B' - (\boldsymbol{A_1}, \boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4}, \boldsymbol{A_5})\ x\ G_{RM}(2,5)''$$

Onde $G_{RM}(2,5)''$ representa a matriz:

$$G_{RM}(2,5)'' = (V_1, V_2, V_3, V_4, V_5)$$

Para encontrar o bit $\boldsymbol{A_0}$, é necessário fazer a maioria dos bits B'':

$$\boldsymbol{A_0} = \boldsymbol{maioria}(B''_1, B''_2, B''_3, \ldots, B''_{32})$$

Por fim, a mensagem decodificada será: $(\boldsymbol{A_0}, \boldsymbol{A_1}, \boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4}, \boldsymbol{A_5}, \boldsymbol{A_{12}}, \boldsymbol{A_{13}}, \boldsymbol{A_{14}}, \boldsymbol{A_{15}}, \boldsymbol{A_{23}}, \boldsymbol{A_{24}}, \boldsymbol{A_{25}}, \boldsymbol{A_{34}}, \boldsymbol{A_{35}}, \boldsymbol{A_{45}})$**.** Caso não seja definida maioria nas expressões realizadas, não será possível decodificar a mensagem. Observamos nesse processo que o código Reed-Muller, diferente dos outros analisados, não se preocupa em descobrir a posição em que o erro aparece na mensagem para corrigi-lo, mas utiliza-se de expressões algébricas e da lógica de maioria para conseguir ter uma cobertura de múltiplos erros.

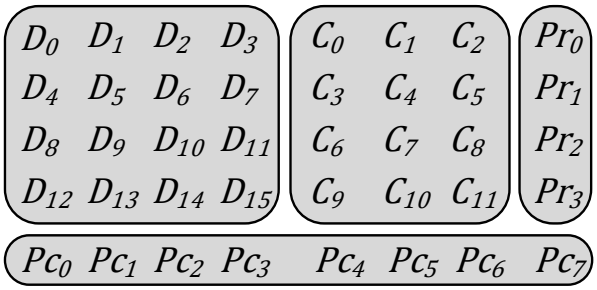
3.5 Código Linha Coluna (CLC)

O código CLC (CASTRO et al, 2016; SILVA et al, 2017; SILVA et al, 2018) foi desenvolvido utilizando conceitos de $Ham - E$ e Paridade para formar um código matricial capaz de corrigir múltiplos erros em bits. Este código utiliza uma tabela de correção que determina, a partir do valor coletado das síndromes, o tipo de erro que ele irá corrigir. O CLC apresenta 2 modos de correção: Padrão e Estendido. O modo Padrão realiza a correção ao verificar síndromes apenas uma vez; já o modo Estendido realiza dupla correção, ou seja, sua correção é dividida em duas partes, dessa maneira há a verificação de síndromes duas vezes e com isso permite a correção de padrões mais agressivos de erro. É importante salientar que o modo Estendido apresentará acréscimo de custo para sua operação.

3.5.1 Codificação do CLC(16,40)

O código CLC (16,40) codifica 16 bits de dados em uma mensagem de 40 bits. Na figura a seguir, é mostrado o formato de uma mensagem codificada utilizando o CLC.

Figura 3.2 – Estrutura do código CLC para 16 bits.



Fonte: Elaborada pelo Autor.

Na Figura 3.2, temos que os 40 bits da mensagem codificada consistem em 16 bits de dados (D), oito bits de paridade de coluna (Pc), quatro bits de paridade da palavra (Pr) e 12 bits de checagem (C). A disposição dos bits codificados da mensagem apresentada foi inspirada no código Matrix (16,32) apresentada na subseção 3.3.

Para a codificação do código CLC, os 16 bits de dados são divididos em quatro grupos de quatro bits. Cada grupo é codificado utilizando $Ham - E(4,8)$. Dos bits gerados pela codificação, em cada linha haverá três bits C e um bit Pr. O símbolo $\oplus$ representa a

operação lógica XOR. Para o cálculo dos bits de checagem, em que i varia de 1 a 4, utilizamos as relações:

$$C_{(0,3,6,9)} = D_{4\cdot(i-1)\ +2} \oplus D_{4\cdot(i-1)\ +3} \oplus D_{4\cdot(i-1)\ +4} \quad (3.13)$$

$$C_{(1,4,7,10)} = D_{4\cdot(i-1)\ +1} \oplus D_{4\cdot(i-1)\ +3} \oplus D_{4\cdot(i-1)\ +4} \quad (3.14)$$

$$C_{(2,5,8,11)} = D_{4\cdot(i-1)\ +1} \oplus D_{4\cdot(i-1)\ +2} \oplus D_{4\cdot(i-1)\ +4} \quad (3.15)$$

Já os bits Pr são calculados da seguinte forma:

$$Pr_{i-1} = D_{4\cdot(i-1)\ +1} \oplus D_{4\cdot(i-1)\ +2} \oplus D_{4\cdot(i-1)\ +3} \oplus D_{4\cdot(i-1)\ +4} \oplus C_{3\cdot(i-1)\ +1} \oplus C_{3\cdot(i-1)\ +2} \oplus C_{3\cdot(i-1)\ +3} \quad (3.16)$$

Após a codificação dos quatro grupos de bits de dados, a última linha será formada pelos bits de paridade de coluna, que são gerados por operações XOR quatro bits de cada coluna na matriz.

$$Pc_0 = D_0 \oplus D_4 \oplus D_8 \oplus D_{12} \quad (3.17)$$

$$Pc_1 = D_1 \oplus D_5 \oplus D_9 \oplus D_{13} \quad (3.18)$$

$$Pc_2 = D_2 \oplus D_6 \oplus D_{10} \oplus D_{14} \quad (3.19)$$

$$Pc_3 = D_3 \oplus D_7 \oplus D_{11} \oplus D_{15} \quad (3.20)$$

$$Pc_4 = C_0 \oplus C_3 \oplus C_6 \oplus C_9 \quad (3.21)$$

$$Pc_5 = C_1 \oplus C_4 \oplus C_7 \oplus C_{10} \quad (3.22)$$

$$Pc_6 = C_2 \oplus C_5 \oplus C_8 \oplus C_{11} \quad (3.23)$$

$$Pc_7 = Pr_0 \oplus Pr_1 \oplus Pr_2 \oplus Pr_3 \quad (3.24)$$

Os bits Pc são de grande importância para detecção de padrões de erro mais complexos que serão mostrados no decorrer do capítulo.

3.5.2 Decodificação do CLC(16,40)

O processo de decodificação do CLC consiste no cálculo das síndromes dos bits de redundância. O primeiro passo é gerar as síndromes dos bits de checagem ($SC's$), de paridade da coluna ($SPc's$) e de paridade de linha (SPr). Para isso, são recalculados os $C's$, $Pc's$ e

$Pr's(RC, RPc$ e $RPr)$ usando os bits $C's$ na mensagem. As equações 3.25 a 3.27 apresentam as equações das síndromes dos bits de redundância.

$$SC = C \oplus RC \quad (3.25)$$

De modo similar, calculamos SPc, que representa a síndrome dos bits de paridade da coluna.

$$SPc = Pc \oplus RPc \quad (3.26)$$

Essas duas síndromes são utilizadas para detectar se houve erro(s) na linha correspondente aos bits da síndrome. A síndrome SC é fundamental para corrigir as linhas utilizando o algoritmo de Hamming, já a síndrome SPc será utilizada para detectar erros nas colunas da matriz.

Além dessas duas, outra síndrome é calculada para detectar e corrigir erros triplos numa mesma linha. Essa é a síndrome da paridade da palavra (SPr), e seu cálculo segue o mesmo raciocínio das outras síndromes calculadas.

$$SPr = Pr \oplus RPr \quad (3.27)$$

A Tabela 3.3 apresenta a tabela de correção que será utilizada para corrigir os erros que ocorrerem em cada linha da matriz na mensagem codificada do CLC. O valor 0 na tabela representa que todos os bits de síndrome são 0, o que significa que não foram detectados erros por essa síndrome; já o valor 1 representa que pelo menos um dos bits de síndrome é diferente de 0, o que já é suficiente para afirmar que erros foram detectados. Por exemplo, o valor 1 na tabela para a síndrome SPr significa que um dos bits de síndrome SPr (SPr_1 ou SPr_2 ou SPr_3 ou SPr_4) apresentou valor igual a 1, caso SPr seja 0 isso significa que todos os bits apresentaram valor 0. O mesmo raciocínio é valido para as síndromes SC e SPc.

Tabela 3.3 – Tabela de correção do CLC

SC	SPr	SPc	Tipo de erro	Método de correção
0	0	0	Sem erro	Nenhum método aplicado
0	0	1	Erro detectado	Nenhum método aplicado

0	1	0	Erro detectado	Nenhum método aplicado
0	1	1	Erro triplo detectado	SPc bits
1	0	0	Erro detectado	Nenhum método aplicado
1	0	1	Número par de erros detectados	SPc bits
1	1	0	Número ímpar de erros detectados	Hamming
1	1	1	Número ímpar de erros detectados (Caso especial)	Hamming/SPc

Fonte: Elaborada pelo Autor.

A Tabela 3.3 apresenta dois tipos de correção que podem ser aplicados pelo CLC: utilização dos bits SPc e o algoritmo de Hamming. A utilização dos bits SPc ocorre para a correção de múltiplos erros em uma mesma linha, já a correção utilizando algoritmo de Hamming é aplicado para situações em que um único erro é detectado.

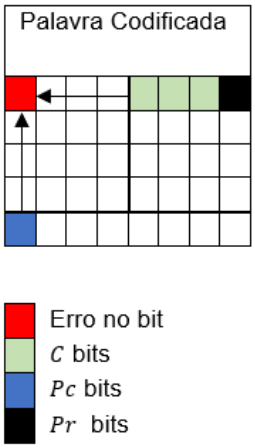
A tabela de correção do CLC apresenta o caso especial ($SC = 1$, $SPr = 1$ e $SPc = 1$), em que nessa situação podem ser utilizados os dois métodos de correção. Para o caso em questão, se somente uma linha apresentar erros, ela será corrigida pelos bits SPc, caso contrário será utilizado Hamming Estendido. Em seguida, são mostrados alguns exemplos de como os dois modos de operação do CLC trabalham para a correção dos erros.

3.5.3 Exemplos de correção do CLC

Nesta secção, serão apresentados alguns exemplos de correção dos bits de dados utilizando o CLC. Os primeiros dois exemplos mostram o funcionamento do CLC padrão nas Figuras 3.3 e 3.4. As figuras são compostas por um desenho da palavra codificada e por uma tabela ao lado que informa quais síndromes tiveram valores iguais a 1 ou 0.

Na Figura 3.3 observa-se que é um caso bem simples de erro. Na tabela ao lado da figura, temos que todas síndromes apresentaram valor 1 para a primeira linha. Nesse caso, o bit errado será corrigido usando código de Hamming.

Figura 3.3 – Erro simples corrigido pelo CLC

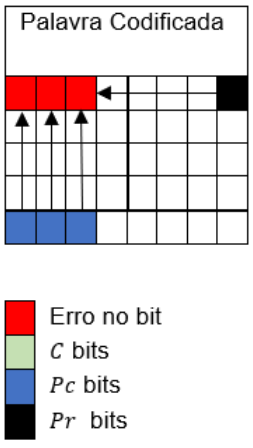


SC	SPr	SPc	Linha (n°)	Método de Correção
1	1	1	Linha 1	Hamming
0	0	0	Linha 2	Linha Ok
0	0	0	Linha 3	Linha Ok
0	0	0	Linha 4	Linha Ok

Fonte: Elaborada pelo Autor.

Na Figura 3.4 é apresentado o caso em que há triplo erro em uma linha. Observe que nas equações 3.15 à 3.17, a inversão dos bits D de 1 a 3 em todas as linhas não trará alterações no recálculo dos bits C, o que torna os SC incapazes de detectar a ocorrência de erros. Para corrigir esse erro, são utilizadas as síndromes SPc e SPr. Observamos que apenas um bit de SPr é diferente de 0, indicando que uma linha está errada, -no caso a primeira - e vemos que há três bits de SPc que são iguais a 1, o que significa que nas respectivas colunas há erros. Com essas duas informações, conclui-se que na primeira linha há três erros nas colunas em que os bits SPc são iguais a um, no caso os três primeiros, que, em seguida, são invertidos, corrigindo a palavra codificada.

Figura 3.4 – Erro triplo corrigido pelo CLC.



SC	SPr	SPc	Linha (n°)	Método de Correção
0	1	1	Linha 1	SPc bits
0	0	0	Linha 2	Linha Ok
0	0	0	Linha 3	Linha Ok
0	0	0	Linha 4	Linha Ok

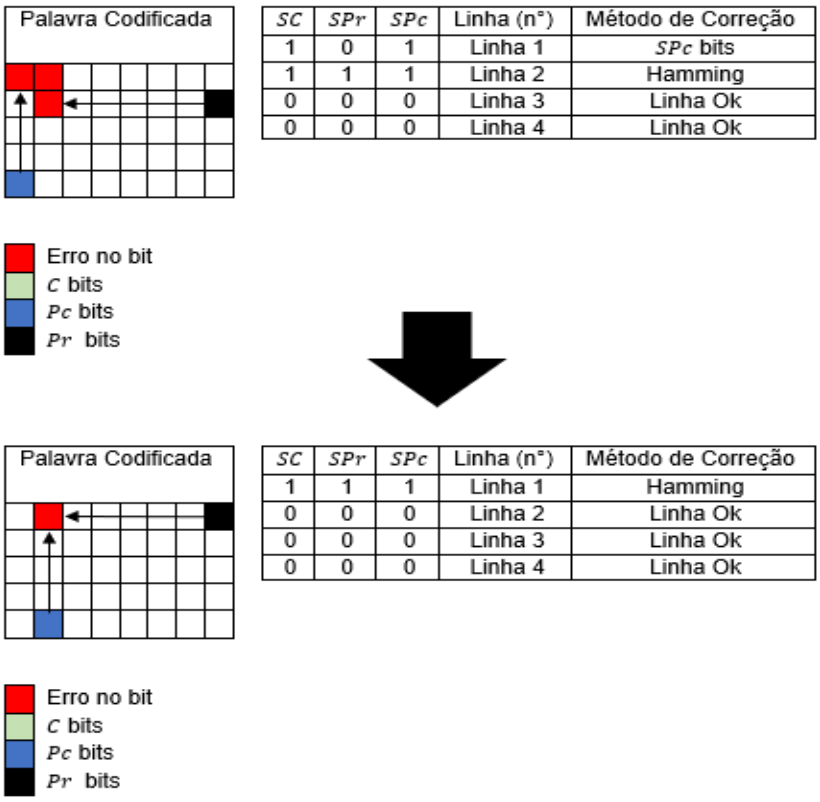
Fonte: Elaborada pelo Autor.

Há casos de múltiplos de erros em que o CLC Padrão não consegue corrigir totalmente os bits de dados presentes na palavra codificada. Para solucionar esse problema, uma segunda verificação das síndromes torna-se necessária. Dessa forma, o modo Estendido é utilizado para corrigir múltiplos erros.

Na Figura 3.5, temos um caso de erro triplo, mas diferentemente da Figura 3.4, esse erro triplo encontra-se dividido nas duas primeiras linhas. Observamos na primeira linha do exemplo que $SC = 1$, $SPr = 0$ e $SPc = 1$, o que significa que há número par de erros nessa linha. Os bits errados são corrigidos pelos bits SPc, no caso, o bit corrigido será o segundo da primeira linha. Na segunda linha, temos que $SC = 1$, $SPr = 1$ e $SPc = 1$. De acordo com a Tabela 3.3, esse tipo de erro é corrigido por Hamming.

Por fim, observa-se que o primeiro bit da primeira linha continua errado, tornando necessário calcular as síndromes novamente (Segundo passo ilustrado na Figura 3.5). Com os novos valores de síndromes, observamos que os valores apresentados para a primeira linha serão $SC = 1$, $SPr = 1$ e $SPc = 1$. Logo, a primeira linha será corrigida por Hamming e todos os bits de dados estarão corretos.

Figura 3.5 – Erro triplo corrigido pelo CLC Estendido.



SC	SPr	SPc	Linha (n°)	Método de Correção
1	0	1	Linha 1	SPc bits
1	1	1	Linha 2	Hamming
0	0	0	Linha 3	Linha Ok
0	0	0	Linha 4	Linha Ok

SC	SPr	SPc	Linha (n°)	Método de Correção
1	1	1	Linha 1	Hamming
0	0	0	Linha 2	Linha Ok
0	0	0	Linha 3	Linha Ok
0	0	0	Linha 4	Linha Ok

Fonte: Elaborada pelo Autor.

3.5.4 Conclusões sobre o CLC

O código CLC apresentado utiliza Hamming Estendido e paridade para formar uma palavra codificada de 40 bits em formato de matriz. Os resultados do CLC(16,40) em (CASTRO et al, 2016) mostraram que o código apresenta bom potencial para aplicações críticas. No entanto, o número de bits de redundância adicionados para uma palavra de 16 bits é grande, tornando difícil sua aplicação em aplicações com recursos limitados. Além disso, as taxas de correção para situações de 6,7 e 8 erros foram bastante discretas para esse código.

O capítulo a seguir propõe novas estruturas de matriz para o código CLC, a partir da substituição do $Ham - E(4,8)$, aplicado no CLC original, pela utilização dos $Ham - E(8,13)$ e $Ham - E(2,6)$, com o objetivo de atenuar os problemas apresentados pela primeira versão do código. Para avaliar o real impacto do acréscimo de redundância do CLC, também é feita uma análise do acréscimo de bits necessário para as novas estruturas com palavras de dados de 16,32 e 64 bits.

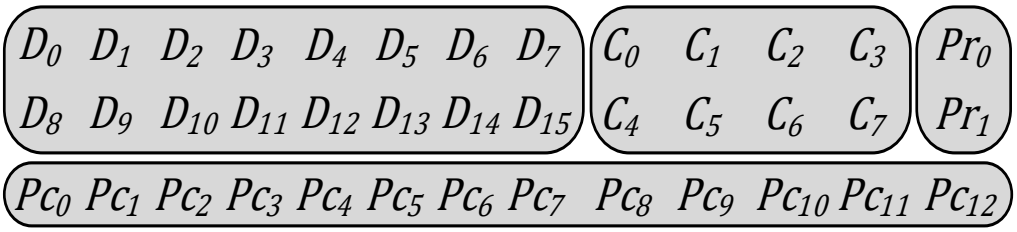
4 METODOLOGIA

No capítulo anterior, foi discutida a implementação do CLC(16,40), que codifica 16 bits de informação, gerando 40 bits. Esse CLC divide a informação original em uma estrutura 2-D, aplicando $Ham - E(4,8)$ e paridade na codificação. O CLC em questão divide a informação em grupos de 4 bits, que são codificados utilizando $Ham - E(4,8)$. Neste capítulo, é proposta a utilização de Hamming Estendido para 2 e 8 bits, modificando a palavra codificada e o número de bits gerados após a codificação. A partir disso, é calculado o impacto em termos de bits de redundância gerados pelos CLCs com a variação do Hamming Estendido utilizado.

4.1 CLC utilizando $Ham - E\ (8,13)$

A Figura 4.1 mostra a estrutura do CLC utilizando $Ham - E(8,13)$. Note que a estrutura 2-D deste CLC, para 16 bits de dados, é uma matriz 3x13. Os bits codificados pelo CLC(16,39) estão divididos em quatro campos: 16 bits de dados (D), 8 bits de checagem (C), 13 bits de paridade de coluna (P_C) e 2 bits de paridade de linha (P_r).

Figura 4.1. Estrutura do código CLC com $Ham - E(8,13)$ para 16 bits.



Fonte: Elaborada pelo Autor.

A utilização de $Ham - E(8,13)$ ao invés de $Ham - E(4,8)$ na codificação do CLC trouxe uma pequena redução de redundância, o que deverá também representar uma diminuição do custo geral de sintetização. As Equações 4.1 a 4.8 apresentam como são calculados os bits C,P_C e P_r do CLC(16,39) para uma palavra de 16 bits, em que $\oplus$ representa a operação lógica XOR (ou exclusivo) .

As Equações 4.1 a 4.4 apresentam o cálculo dos C bits.

$$C_{4q} = D_{8q} \oplus D_{8q+1} \oplus D_{8q+3} \oplus D_{8q+4} \oplus D_{8q+6} \qquad \forall\ q \in \{0, 1\} \qquad (4.1)$$

$$C_{4q+1} = D_{8q} \oplus D_{8q+2} \oplus D_{8q+3} \oplus D_{8q+5} \oplus D_{8q+6} \qquad \forall\ q \in \{0, 1\} \qquad (4.2)$$

$$C_{4q+2} = D_{8q+1} \oplus D_{8q+2} \oplus D_{8q+3} \oplus D_{8q+7} \qquad \forall\ q \in \{0, 1\} \qquad (4.3)$$

$$C_{4q+3} = D_{8q+4} \oplus D_{8q+5} \oplus D_{8q+6} \oplus D_{8q+7} \qquad \forall\ q \in \{0, 1\} \qquad (4.4)$$

A Equação 4.5 apresenta o cálculo dos bits *Pr* para cada linha.

$$Pr_q = D_{8q} \oplus \ldots \oplus D_{8q+7} \oplus C_{4q} \oplus \ldots \oplus C_{4q+3} \qquad \forall\ q \in \{0, 1\} \qquad (4.5)$$

As Equações 4.6 a 4.8 calculam os valores dos bits P_C de cada coluna.

$$P_{Cq} = D_q \oplus D_{q+8} \qquad \forall\ 0 \le q \le 7 \qquad (4.6)$$

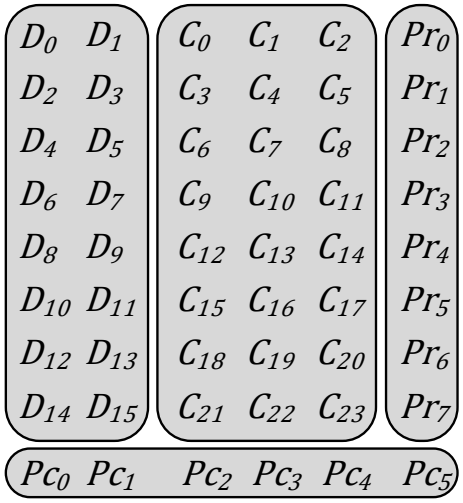
$$P_{Cq} = D_q \oplus D_{q+4} \qquad \forall 0 \le q \le 3 \qquad (4.7)$$

$$P_{C12} = P_{r0} \oplus P_{r1} \qquad (4.8)$$

4.2 CLC utilizando $Ham - E\ (2, 6)$

A Figura 4.2 mostra a estrutura do CLC utilizando $Ham - E(2,6)$. A estrutura 2-D deste CLC, para 16 bits de dados, é uma matriz 9x6. Os bits estão divididos em: 16 bits de dados (*D*), 24 bits de checagem (*C*), 6 bits de paridade de coluna (P_C) e 8 bits de paridade de linha (P_r).

Figura 4.2. Estrutura do código CLC com $Ham - E(2,6)$ para 16 bits.



Fonte: Elaborada pelo Autor.

Note que o CLC(16,54) apresenta mais bits de redundância que os outros formatos do código. Com o acréscimo de redundância, é esperado que o CLC(16,54) consiga corrigir

mais padrões de MBUs, em contrapartida, ele também apresentará maior custo de implementação que seus concorrentes. As Equações 4.9 a 4.15 apresentam como são calculados os bits C,P_C e P_r do CLC(16,54) para uma palavra de 16 bits, em que $\oplus$ representa a operação lógica XOR.

As Equações 4.9 a 4.11 apresentam o cálculo dos C bits.

$$C_{3q} = D_{2q} \oplus D_{2q+1} \qquad \forall\ 0 \le q \le 7 \qquad (4.9)$$

$$C_{3q+1} = D_{2q} \qquad \forall\ 0 \le q \le 7 \qquad (4.10)$$

$$C_{3q+2} = D_{2q+1} \qquad \forall\ 0 \le q \le 7 \qquad (4.11)$$

A Equação 4.12 apresenta o cálculo dos bits *Pr* para cada linha.

$$Pr_q = D_{2q} \oplus D_{2q+1} \oplus C_{3q} \oplus C_{3q+1} \oplus C_{3q+2} \qquad \forall\ 0 \le q \le 7 \qquad (4.12)$$

As Equações 4.13 a 4.15 calculam os valores dos bits P_C de cada coluna.

$$Pc_q = D_q \oplus D_{2+q} \oplus D_{4+q} \oplus D_{6+q} \oplus \ldots \oplus D_{12+q} \oplus D_{14+q} \qquad \forall\ q \in \{0, 1\} \qquad (4.13)$$

$$Pc_{q+2} = C_q \oplus C_{3+q} \oplus C_{6+q} \oplus C_{9+q} \oplus \ldots \oplus C_{18+q} \oplus C_{21+q} \qquad \forall\ q \in \{0, 1, 2\} \qquad (4.14)$$

$$Pc_5 = Pr_0 \oplus Pr_1 \oplus Pr_2 \oplus \ldots \oplus Pr_6 \oplus Pr_7 \qquad (4.15)$$

Os CLCs propostos possuem o mesmo algoritmo de correção de erros do CLC(16,40), que foi apresentado no capítulo 3, tanto para o modo Padrão quanto para o modo Estendido de correção.

4.3 Análise de redundância para 16, 32 e 64 bits de dados

Neste capítulo, foram apresentados dois novos formatos do código CLC, variando o Hamming Estendido utilizado para a composição das linhas da matriz. Percebe-se que os novos formatos já trouxeram alterações na quantidade de bits gerados pela codificação. Este subcapítulo analisa a aplicação das versões do CLC para 32 e 64 bits de dados. Esse estudo tem como finalidade observar o impacto de redundância que o CLC trará ao ser utilizado Hamming Estendido de 2, 4 e 8 bits. A Equação 4.16 apresenta o número de bits gerados pela codificação do CLC (N_{TB}).

$$N_{TB} = \left(\frac{N_{Bits\ de\ dados}}{HP} + 1\right) * (HP + RHP), \quad (4.16)$$

em que, $N_{Bits\ de\ dados}$ é o número de bits que são codificados, HP é o número de bits codificados por Hamming Estendido e RHP é o número de bits de redundância gerados pelo Hamming Estendido aplicado. A Equação 4.17 apresenta a porcentagem de bits adicionados pela codificação do CLC (R_{RTB}).

$$R_{RTB} = \left(\frac{N_{TB} - N_{Bits\ de\ dados}}{N_{TB}}\right) \quad (4.17)$$

A Tabela 4.1 apresenta os valores de N_{TB} e R_{RTB} calculados para 16, 32 e 64 bits de dados codificados com CLC utilizando Hamming Estendido de 2, 4 e 8 bits.

Tabela – 4.1. Valores de N_{TB} para 16, 32 e 64 bits de dados.

$N_{Bits\ de\ dados}$(bits)	HP(bits)	RHP(bits)	N_{TB}(bits)	R_{RTB} (%)
16	2	4	54	237,5
	4	4	40	150
	8	5	39	143,75
32	2	4	102	218,5
	4	4	72	125
	8	5	65	103,1
64	2	4	198	209,4
	4	4	136	112,5
	8	5	117	82,6

Fonte: Elaborada pelo Autor.

A Tabela 4.1 mostra que à medida que $N_{Bits\ de\ dados}$ cresce, a R_{RTB} de cada código diminui, ou seja, o número de bits de redundância necessários para realizar a codificação com CLC atenua. Também pode ser visto que a utilização do CLC com Hamming Estendido de 8 bits traz o menor acréscimo de redundância. Não obstante, esse tipo de CLC trouxe decréscimo de R_{RTB} entre 16 e 64 bits de dados de 61,15%, -o maior dentre todas as outras versões-. Para avaliar os impactos que a utilização de $Ham - E(2,6)$ e $Ham - E(8,13)$ trazem em cobertura contra erros, confiabilidade e implementação física, serão apresentados os experimentos realizados, bem como as conclusões adquiridas sobre os resultados no próximo capítulo.

5 EXPERIMENTOS E ANÁLISE DOS RESULTADOS

Neste capítulo são apresentados três experimentos que serviram como base para avaliar o impacto que aplicação de diferentes tipos de $Ham - E$ trouxe para a implementação do código CLC. Os experimentos estão divididos da seguinte forma:

I. Experimento de Injeção de Erros.
II. Estimativa de Confiabilidade.
III. Análise de Implementação Física.

Para os experimentos, foram implementados os códigos do CLC com $Ham - E(2,6)$, $Ham - E(4,8)$ e $Ham - E(8,13)$ nos modos Padrão e Estendido de correção, e comparados com os códigos Matrix e Reed-Muller(2,5). Os CCEs foram desenvolvidos para palavras de dado de 16,32 e 64 bits.

As versões dos CLCs implementadas em $Ham - E(8,13)$, $Ham - E(4,8)$, e $Ham - E(2,6)$, e dos códigos Reed-Muller(2,5) e Matrix são mostrados na Tabela 5.1.

Tabela 5.1. CCEs utilizados nos experimentos.

CCE	16 bits	32 bits	64 bits
CLC com $Ham - E$ (8,13)	CLC-S(16,39) CLC-E(16,39)	CLC-S(32,65) CLC-E(32,65)	CLC-S(64,117) CLC-E(64,117)
CLC com $Ham - E$ (4,8)	CLC-S(16,40) CLC-E(16,40)	CLC-S(32,72) CLC-E(32,72)	CLC-S(64,136) CLC-E(64,136)
CLC com $Ham - E$ (2,6)	CLC-S(16,54) CLC-E(16,54)	CLC-S(32,102) CLC-E(32,102)	CLC-S(64,198) CLC-E(64,198)
Matrix	Matrix(16,32)	Matrix(32,60)	Matrix(64,116)
Reed-Muller(2,5)	Reed-Muller(2,5)-16	Reed-Muller(2,5)-32	Reed-Muller(2,5)-64

Fonte: Elaborada pelo Autor.

Para o código Reed-Muller(2,5), foi utilizado *interleaving* dessa versão, que codifica 16 bits de dados, para a codificação de palavras de dados de 32 e 64 bits. Para isso, em 32 e 64 bits os dados foram divididos igualmente em grupos de informação de 16 bits. Logo, para 32 bits; os dados são divididos em 2 grupos; já para 64 bits, os dados são divididos em 4 grupos. Cada grupo é codificado utilizando Reed-Muller(2,5) e, ao final, são repassados os dados em conjunto.

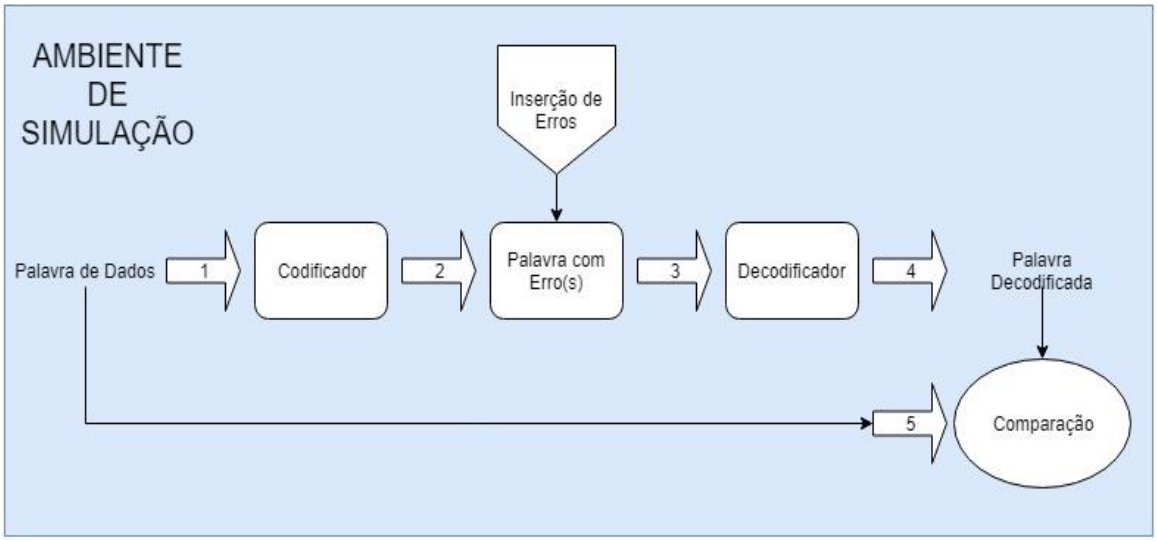
Na decodificação, são divididos grupos de 32 bits, que é o resultado de uma codificação do Reed-Muller(2,5), em que cada grupo será decodificado separadamente

resultado nas palavras originais de 16 bits. Os Reed-Muller(2,5) desenvolvidos para 16,32 e 64 bits foram chamados respectivamente de: Reed-Muller(2,5)-16, Reed-Muller(2,5)-32 e Reed-Muller(2,5)-64.

5.1 Experimento de Injeção de Erros

Para esse experimento, os códigos foram colocados em um ambiente de simulação projetado em MATLAB para calcular as taxas de correção e detecção. A Figura 5.1 apresenta como foram desenvolvidos os testes para esse experimento.

Figura 5.1. Ambiente de simulação projetado.



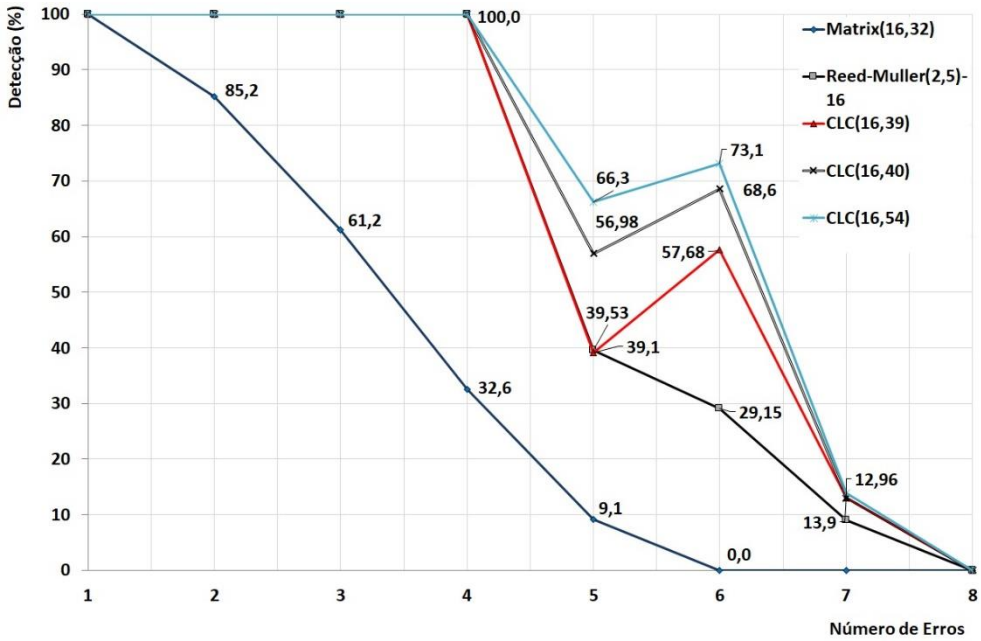
Fonte: Elaborada pelo Autor.

Para o algoritmo apresentado na Figura 5.1, uma Palavra de Dados (PDA) é codificada (Passo 1) de acordo com o seu tamanho (16, 32 ou 64 bits) e o CCE escolhido. Em seguida, na palavra codificada, são inseridos de 1 a 8 erros adjacentes (Passo 2), com objetivo de simular MBUs em memórias, pois os erros nessas situações tendem a ser fisicamente próximos (SATOH; TOSAKA; WENDER, 2000).

Após a inserção de erros, a Palavra com Erros é passada para o decodificador (Passo 3), que irá calcular os bits de checagem e tentar corrigi-los. Após a tentativa de correção dos erros, é gerada a Palavra Decodificada (Passo 4), que finalmente é comparada com a PDA original (Passo 5), em que será verificado se os erros foram corrigidos e detectados, encerrando uma simulação. Para cada situação de erro (i.e. 1, 2, 3, ...,7 e 8), foram gerados 1 milhão de simulações. Por fim, são calculadas as taxas de correção e detecção para cada situação de erro

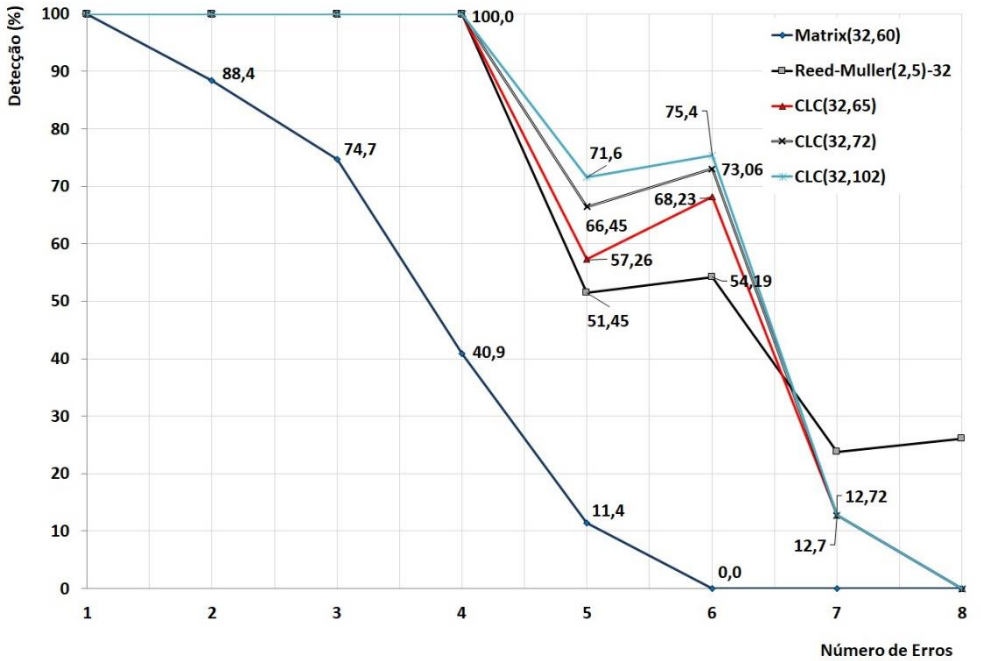
testado. É importante salientar que os modos Padrão e Estendido do CLC afetarão somente a correção do CCE. As Figuras 5.2, 5.3 e 5.4 apresentam os resultados de detecção dos códigos implementados para 16, 32 e 64 bits respectivamente.

Figura 5.2. Resultados de Detecção para CCEs de 16 bits



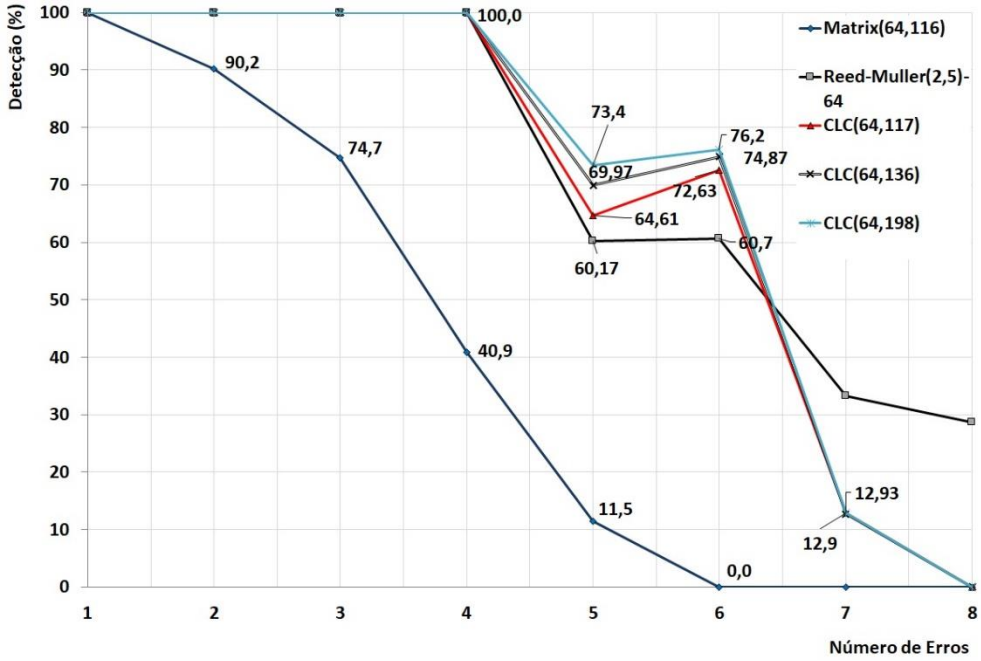
Fonte: Elaborada pelo Autor.

Figura 5.3. Resultados de Detecção para CCEs de 32 bits



Fonte: Elaborada pelo Autor.

Figura 5.4. Resultados de Detecção para CCEs de 64 bits



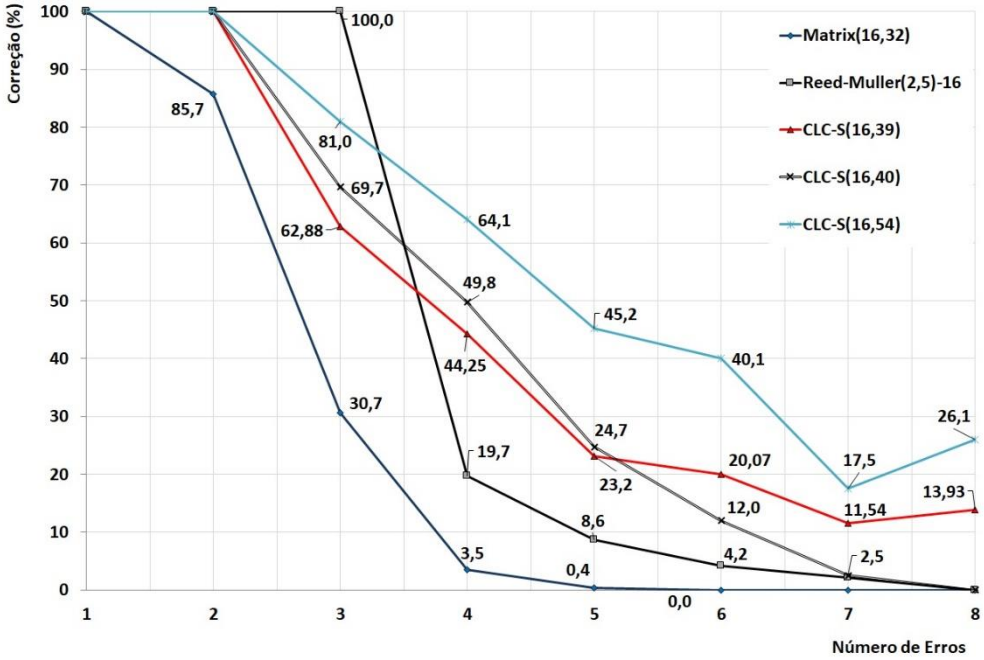
Fonte: Elaborada pelo Autor.

A aplicação do *interleaving* no Reed-Muller(2,5) para as PDAs de 32 e 64 aumentou as taxas de detecção. Isso ocorreu pois, em algumas situações de múltiplos erros, houve divisão de erros entre os grupos de dados do Reed-Muller(2,5)-32, de maneira que amplificou a ocorrência no Reed-Muller(2,5)-64, assim torna-os mais fáceis de serem detectados pelo código. O código Matrix apresentou as menores taxas de detecção por ser o código menos robusto dos apresentados. Quando houve um aumento da PDA aplicada, os CCEs tiveram um acréscimo em suas taxas de detecção. Além disso, os múltiplos erros afetaram proporcionalmente uma menor quantidade de bits da palavra codificada, o que os tornou mais simples de serem detectados e impulsionou as taxas de detecção.

Devido aos CLCs formados por $Ham - E(2,6)$ apresentarem mais bits de redundância que os demais códigos, suas taxas de detecção foram as melhores na maioria dos cenários testados, seguido pelos CLCs formados por $Ham - E(4,8)$ e CLCs formados por $Ham - E(8,13)$. Um resultado de destaque é que de 5 para 6 erros, em que houve um acréscimo na taxa de detecção para todos os tamanhos de PDAs. Isso pode ser explicado pelo fato de que o código CLC utiliza $Ham - E(n,k)$ no seu algoritmo permitindo a detecção de erros pares em uma linha, enquanto ele não consegue diferenciar erros ímpares em uma linha (1 e 3 erros por exemplo).

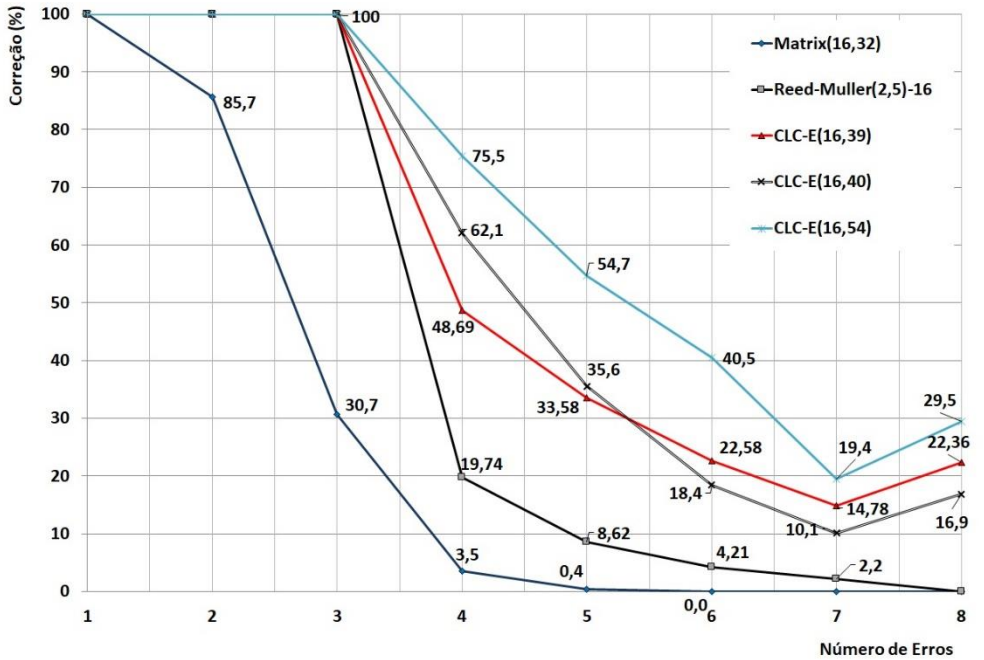
Para os resultados de correção, foram realizados testes dos CLCs em seus dois modos de correção: Padrão (CLC-S) e Estendido (CLC-E). Os grupos de Figuras 5.5, 5.6 e 5.7 apresentam os resultados comparativos para PDAs de 16, 32 e 64 bits.

Figura 5.5a. Resultados de Correção para CCEs de 16 bits.



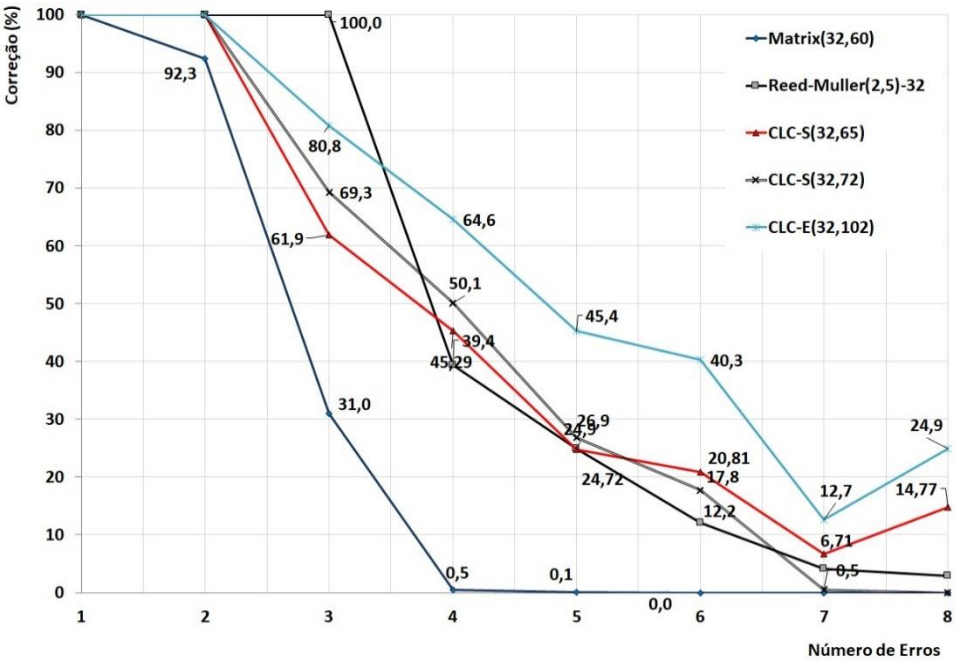
Fonte: Elaborada pelo Autor.

Figura 5.5b. Resultados de Correção para CCEs de 16 bits.



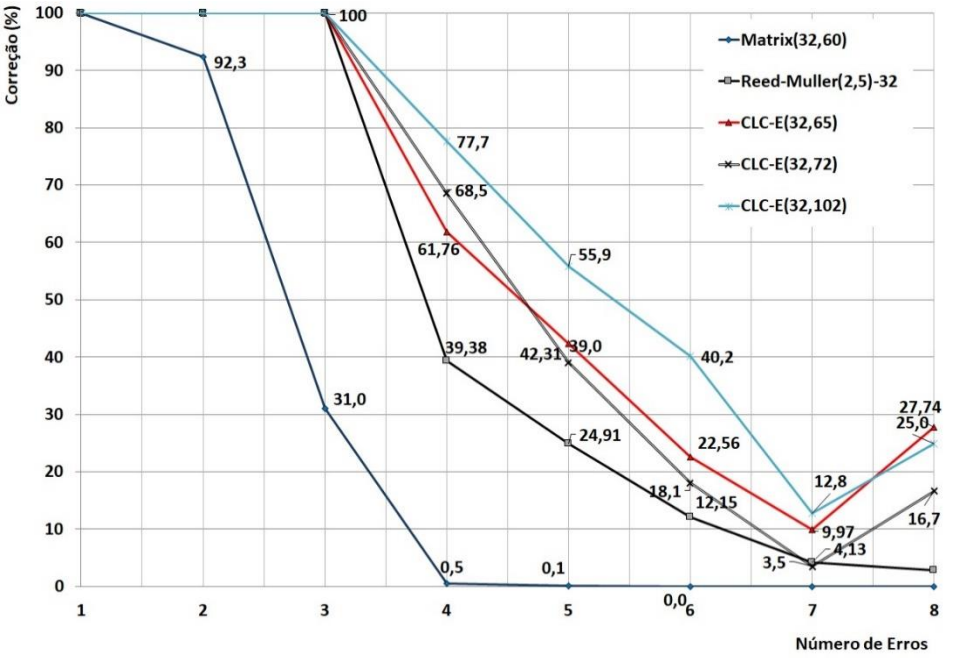
Fonte: Elaborada pelo Autor.

Figura 5.6a. Resultados de Correção para CCEs de 32 bits.



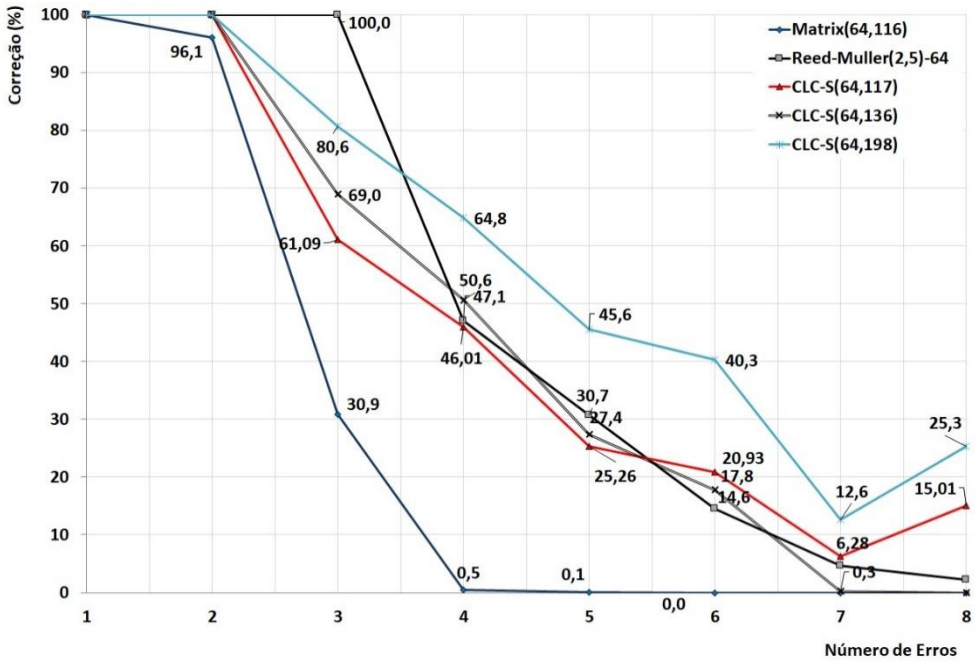
Fonte: Elaborada pelo Autor.

Figura 5.6b. Resultados de Correção para CCEs de 32 bits.



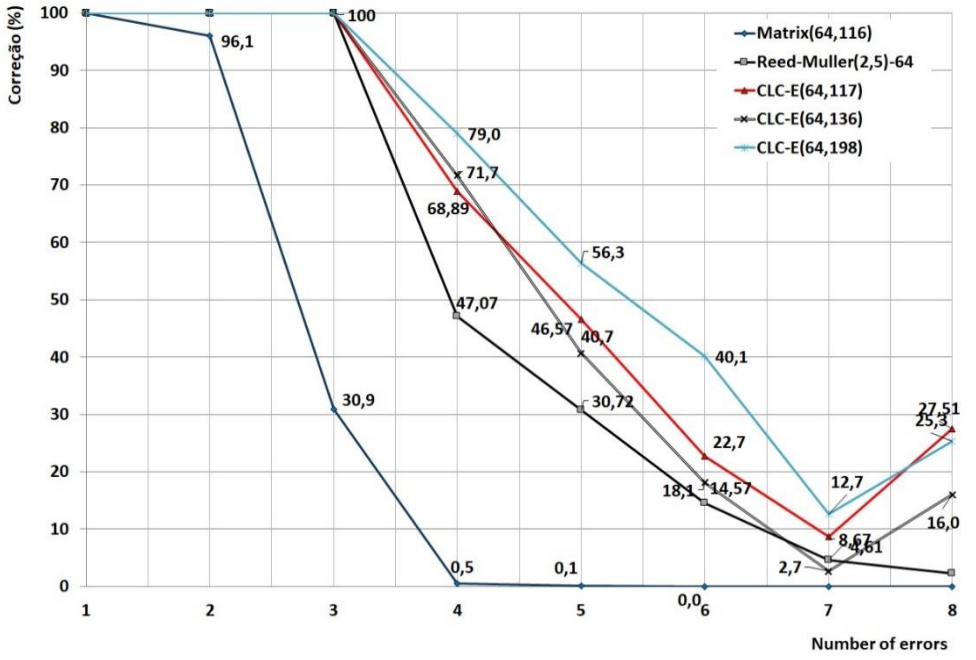
Fonte: Elaborada pelo Autor.

Figura 5.7a. Resultados de Correção para CCEs de 64 bits.



Fonte: Elaborada pelo Autor.

Figura 5.7b. Resultados de Correção para CCEs de 64 bits.



Fonte: Elaborada pelo Autor.

Devido à implementação de *interleaving* no Reed-Muller(2,5), as taxas de correção aumentam à medida que as PDAs crescem, pois cenários de múltiplos erros podem se espalhar por grupos de dados que estão divididos pelo *interleaving*. O código Matrix obteve dados de

correção inferiores aos demais códigos. A variação das PDAs trouxe um leve aumento de correção para o cenário de 2 erros, porém não houve ganho significativo nos demais. Com relação aos novos CLCs, assim como nos testes de detecção, os CLCs formados por $Ham - E(2,6)$ tiveram as melhores taxas de correção na maioria dos cenários testados, seguidos pelos CLCs formados por $Ham - E(4,8)$ e CLCs formados por $Ham - E(8,13)$.

Verificou-se que tanto para as análises de detecção, quanto de correção, os CLCs formados por $Ham - E(8,13)$ apresentaram o menor acréscimo de redundância entre os CLCs, porém sua cobertura contra erros apresentou taxas comparáveis aos das outras versões dos códigos, nos dois modos de correção. Além disso, as novar versões do CLC propostas (CLC com $Ham - E(8,13)$ e CLC com $Ham - E(2,6)$) foram capazes de corrigir mais padrões de erros para os cenários de 6, 7 e 8 erros. Isso pode ser justificado pelo fato de que, para as novas estruturas, a chance de todos ou partes dos erros ocorrerem nos bits de redundância é maior que para o CLC com $Ham - E(4,8)$.

5.2 Estimativa de Confiabilidade

Em uma aplicação real de um específico CCE, os bits codificados por ele são armazenados em uma memória para ser eventualmente decodificado e retransmitido internamente num sistema tolerante a falhas. Considerando que a incidência de radiação em memórias pode ocasionar MBUs, número de bits codificados e taxas de cobertura contra erros (detecção e correção) serão parâmetros-chave para a determinação da confiabilidade de um CCE. Nessa subseção, são analisados os resultados dos testes de confiabilidade dos códigos implementados considerando PDAs de 16,32 e 64 bits.

5.2.1 Confiabilidade de Detecção e Correção

Com base nos experimentos realizados no trabalho (ARGYRIDES et al, 2007), em que se assume que MBUs ocorrem com a distribuição de Poisson. A Equação 5.1 apresenta a equação da confiabilidade da correção de erros ($CE_{Correção}$) em cada instante de tempo(t) como função da: probabilidade de correção de erros (*ECorreção*), da probabilidade de falha da memória (*MF*).

$$E_{Correção}(t) = \sum_{i=1}^{Me} P(ECorreção/MF) = \sum_{i=1}^{Me} P(ECorreção/iE) \times P(iE/MF), \qquad (5.1)$$

em que: *Me* é o número máximo de erros, e iE é a probabilidade de erros em *MF*. Equação 5.2 mostra a probabilidade de ocorrência erros em uma memória ($P(iE/MF)$), enquanto que a Equação 5.3 fornece a probabilidade de ocorrer i erros numa palavra de $(d + p)$ bits.

$$P\left(\frac{iE}{MF}\right) = \frac{P(iE)}{P(MF)}, \qquad (5.2)$$

$$P(iE) = \binom{d+p}{i} \times \left(1 - e^{-\lambda t}\right) \times e^{-\lambda(d+p-i)t}, \qquad (5.3)$$

em que, d é o número de bits de dados, p é o número de bits de redundância e λ é a taxa de ocorrência de erros. A Equação 5.4 dá a probabilidade de uma memória falhar.

$$P(MF) = 1 - e^{-\lambda(d+p)t} \qquad (5.4)$$

Substituindo na Equação 5.1 as Equações 5.2, 5.3 e 5.4, temos:

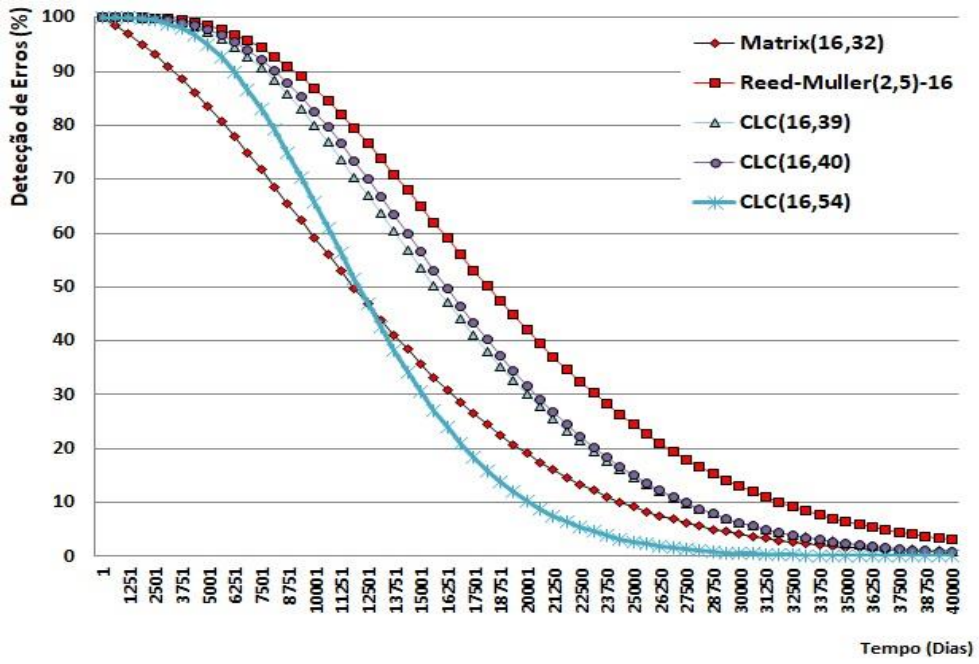
$$E_{Correção}(t) = \sum_{i=1}^{Me} P\left(\frac{ECorreção}{iE}\right) \times \frac{\binom{d+p}{i} \times \left(1 - e^{-\lambda t}\right) \times e^{-\lambda(d+p-i)t}}{1 - e^{-\lambda(d+p)t}} \qquad (5.5)$$

A probabilidade $P(ECorreção/iE)$ representa os resultados obtidos das taxas de correção (Figuras 5.5, 5.6 e 5.7). Analogamente, a Equação 5.6 é gerada para ser aplicada às taxas de detecção (Figuras 5.2,5.3 e 5.4).

$$E_{Detecção}(t) = \sum_{i=1}^{Me} P\left(\frac{EDetecção}{iE}\right) \times \frac{\binom{d+p}{i} \times \left(1 - e^{-\lambda t}\right) \times e^{-\lambda(d+p-i)t}}{1 - e^{-\lambda(d+p)t}} \qquad (5.6)$$

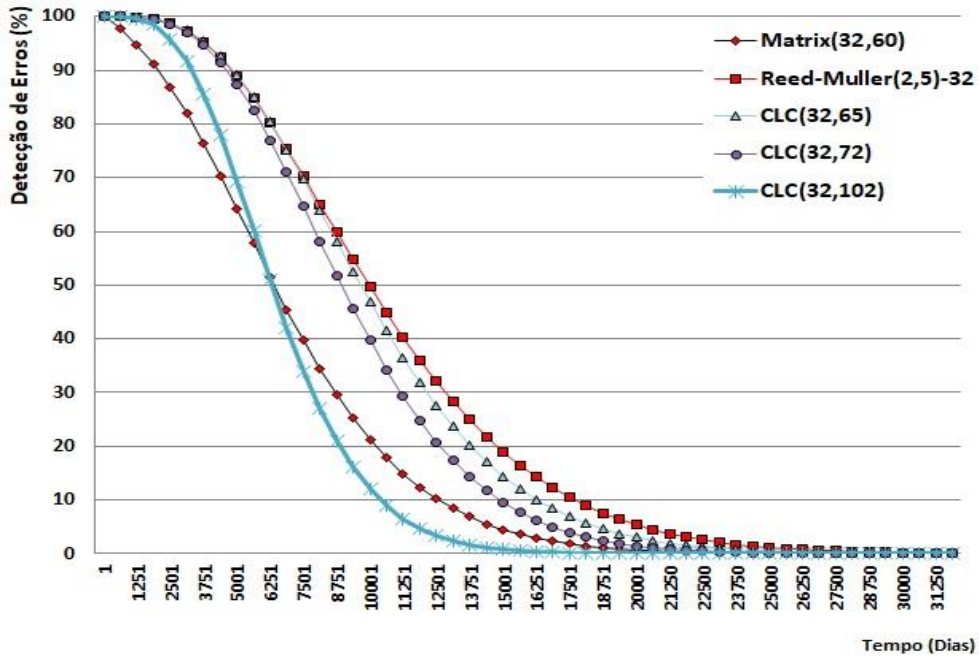
As Figuras 5.8, 5.9 e 5.10 apresentam os resultados de confiabilidade de detecção dos códigos analisados para as PDAs.

Figura 5.8. Confiabilidade de Detecção dos CCEs para 16 bits.



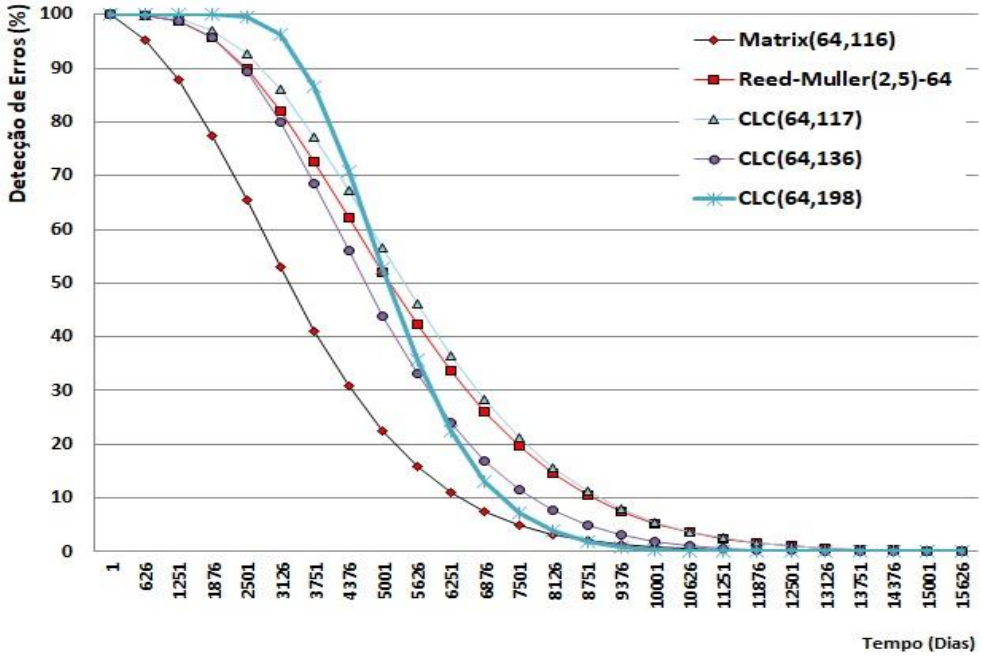
Fonte: Elaborada pelo Autor.

Figura 5.9. Confiabilidade de Detecção dos CCEs para 32 bits.



Fonte: Elaborada pelo Autor.

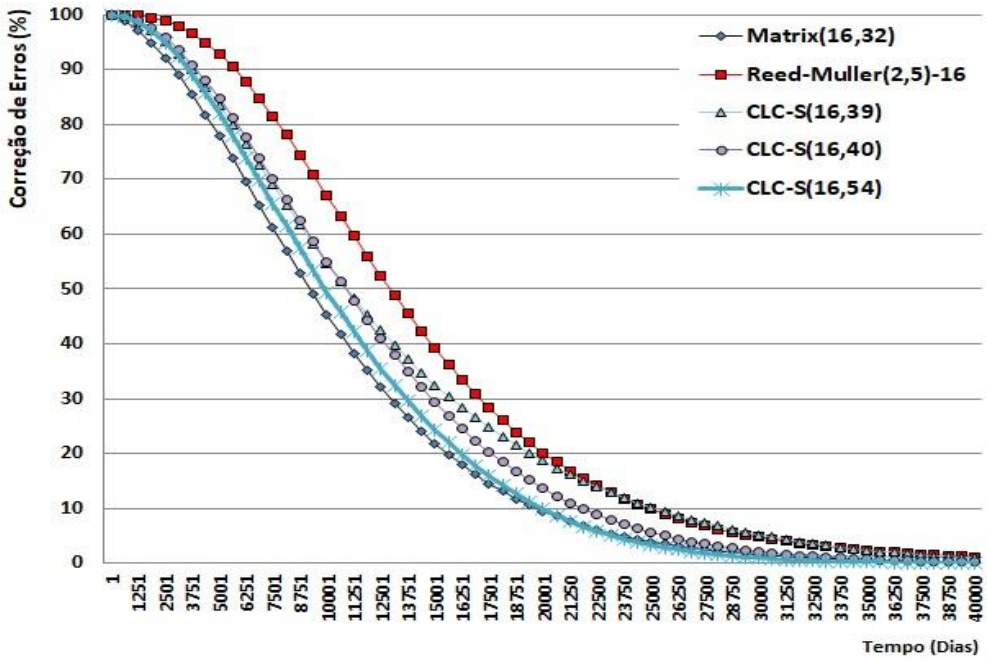
Figura 5.10. Confiabilidade de Detecção dos CCEs para 64 bits.



Fonte: Elaborada pelo Autor.

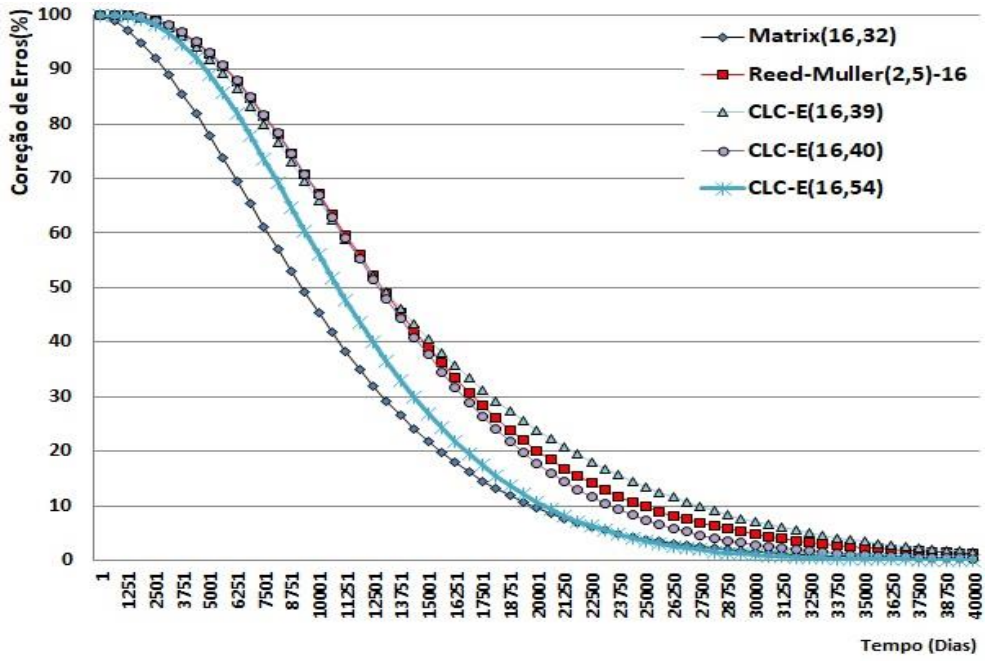
Os grupos de Figuras 5.11, 5.12 e 5.13 apresentam os resultados obtidos de confiabilidade de correção pelos CCEs para 16, 32 e 64 bits com os CLCs nos modos Padrão e Estendido de correção.

Figura 5.11a. Confiabilidade de Correção para 16 bits.



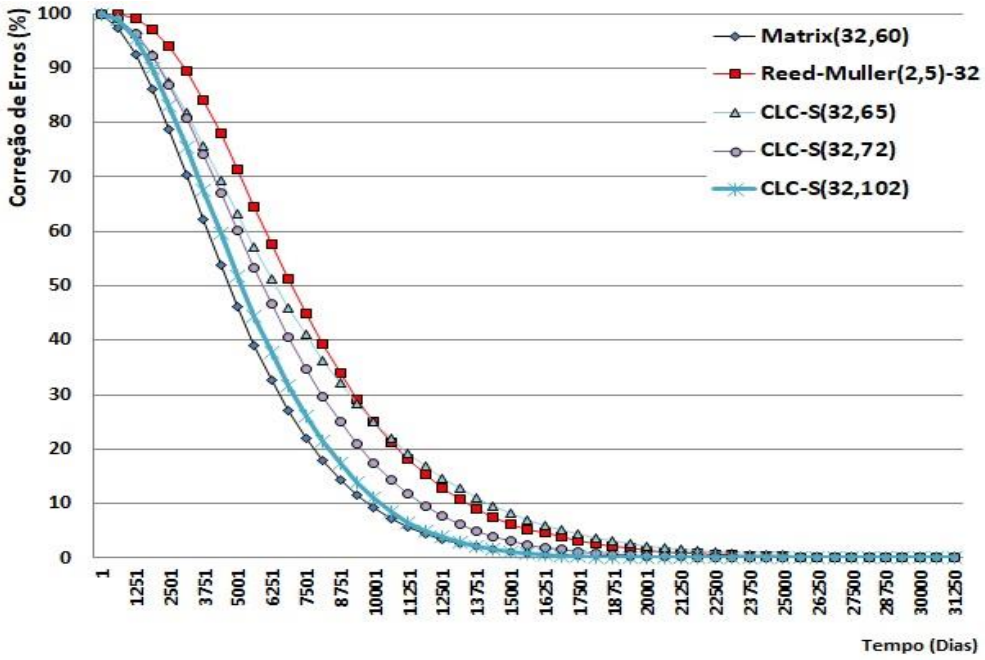
Fonte: Elaborada pelo Autor.

Figura 5.11b. Confiabilidade de Correção para 16 bits.



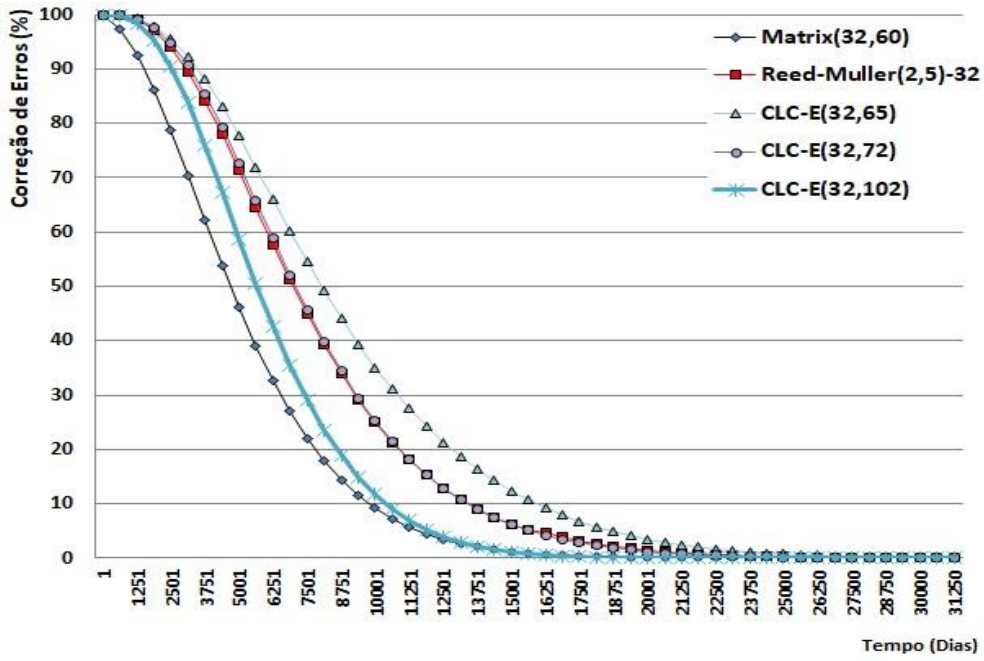
Fonte: Elaborada pelo Autor.

Figura 5.12a. Confiabilidade de Correção para 32 bits.



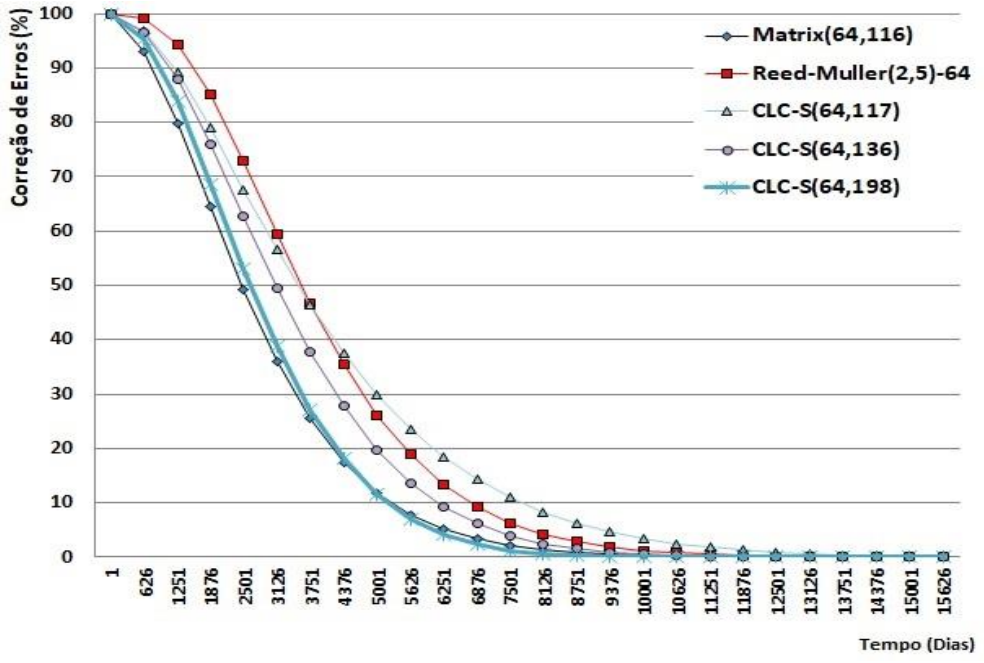
Fonte: Elaborada pelo Autor.

Figura 5.12b. Confiabilidade de Correção para 32 bits.



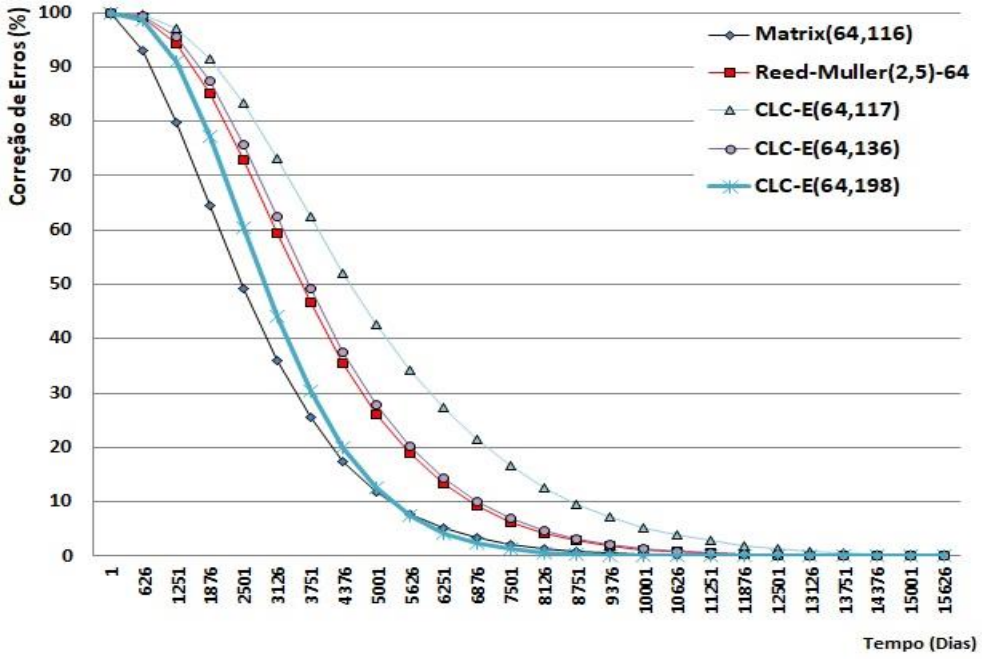
Fonte: Elaborada pelo Autor.

Figura 5.13a. Confiabilidade de Correção para 64 bits.



Fonte: Elaborada pelo Autor.

Figura 5.13b. Confiabilidade de Correção para 64 bits.



Fonte: Elaborada pelo Autor.

Note que, no eixo horizontal do gráfico, a confiabilidade é variada em função dos dias em que essa memória opera. Quanto mais tempo passar, mais críticos são os tipos de MBUs apresentados no dispositivo.

Dito isso, observe que a variação do $Ham - E$ aplicado no CLC trouxe resultados significativos à medida que são aplicadas PDAs diferentes. Para 16 bits de dados, o código Reed-Muller(2,5)-16 obteve os melhores resultados de confiabilidade tanto em detecção quanto em correção em comparação com todos os códigos analisados. Porém, já para 32 bits de dados é nítido o crescimento da confiabilidade das versões do CLC, tanto no modo Estendido quanto no modo Padrão de correção: o CLC-E com $Ham - E(8,13)$ foi o que conseguiu ultrapassar a confiabilidade do Reed-Muller(2,5)-32, para 64 bits, a versão do CLC com $Ham - E(8,13)$ superou o Reed-Muller(2,5) em ambos os modos de correção.

Para testes de confiabilidade em memória, quanto maior o número de bits aplicados, maior será chance de MBUs afetarem bits codificados pelo CCE, uma vez que as taxas de correção e detecção obtidas pelo CLC com $Ham - E(8,13)$ e Reed-Muller(2,5) para PDAs de 32 e 64 bits foram altas, o número de bits de redundância teve impacto significativo nos resultados de confiabilidade. Por outro lado, a versão do CLC com $Ham - E(2,6)$, mesmo com melhores taxas de correção e detecção, apresentou resultados de confiabilidade pouco

superiores aos do código Matrix, devido ao grande número de bits de redundância necessários para sua codificação em comparação aos outros CCEs.

5.2.2 Confiabilidade e Estimativa de Mean Time To Failure (MTTF)

Nesta seção são apresentados os resultados de confiabilidade e MTTF dos CCEs levando em consideração taxa de correção em memórias de 16,32 e 64 bits. A equação 5.7 apresenta o cálculo da confiabilidade de uma memória em relação ao tempo.

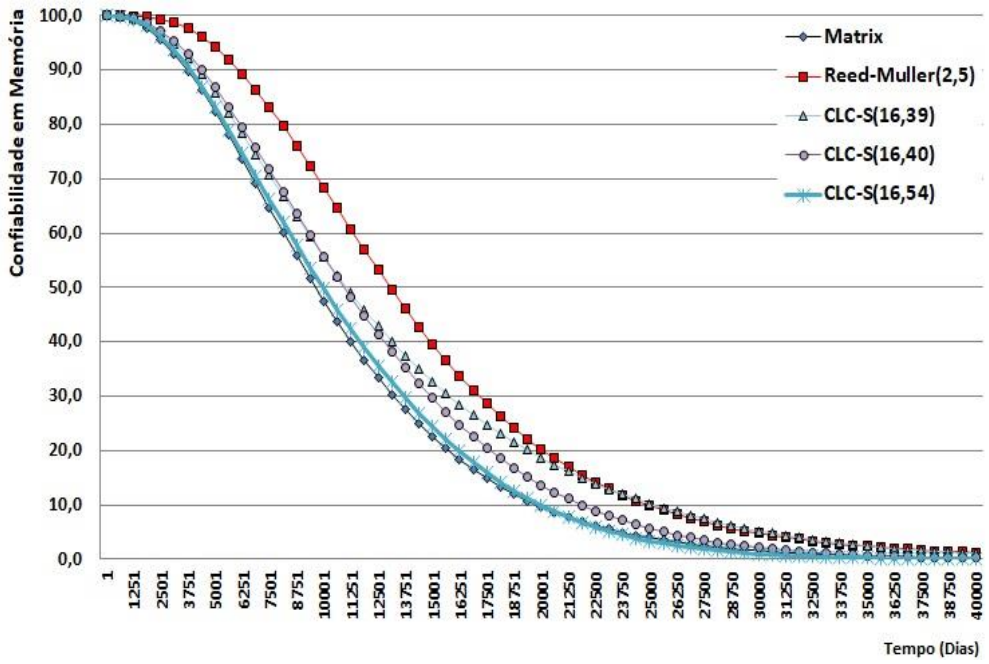
$$r(t) = 1 - P(MF) + \frac{\sum_{i=1}^{Me} P(iE) \times P(iCI)}{Me}, \qquad (5.7)$$

na qual, $1 - P(MF)$ é a probabilidade de não ocorrer erros, iCI representa a correção de i erros, e $\frac{\sum_{i=1}^{Me} P(iE) \times P(iCI)}{Me}$ representa a probabilidade de ocorrência de erros que podem ser corrigidos pelo código avaliado. A Equação 5.8 apresenta que o cálculo do MTTF consiste na integral do cálculo da confiabilidade de uma memória com n registros em relação ao tempo.

$$MTTF = \int_0^{\infty} r(t)^n \, dt \, , \qquad (5.8)$$

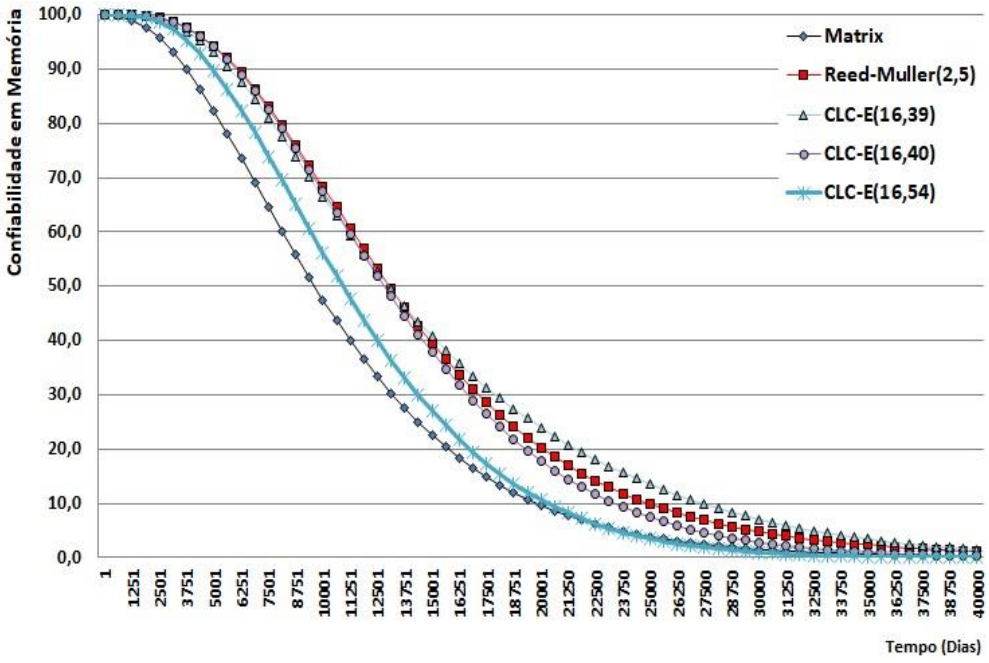
Os grupos de Figuras 5.14, 5.15 e 5.16 apresentam os resultados de confiabilidade em função do tempo comparando todos os CCEs apresentados para um registro de 16,32 e 64 bits.

Figura 5.14a. Confiabilidade para registro de 16 (λ=10^{-5} upsets/bit/dia).



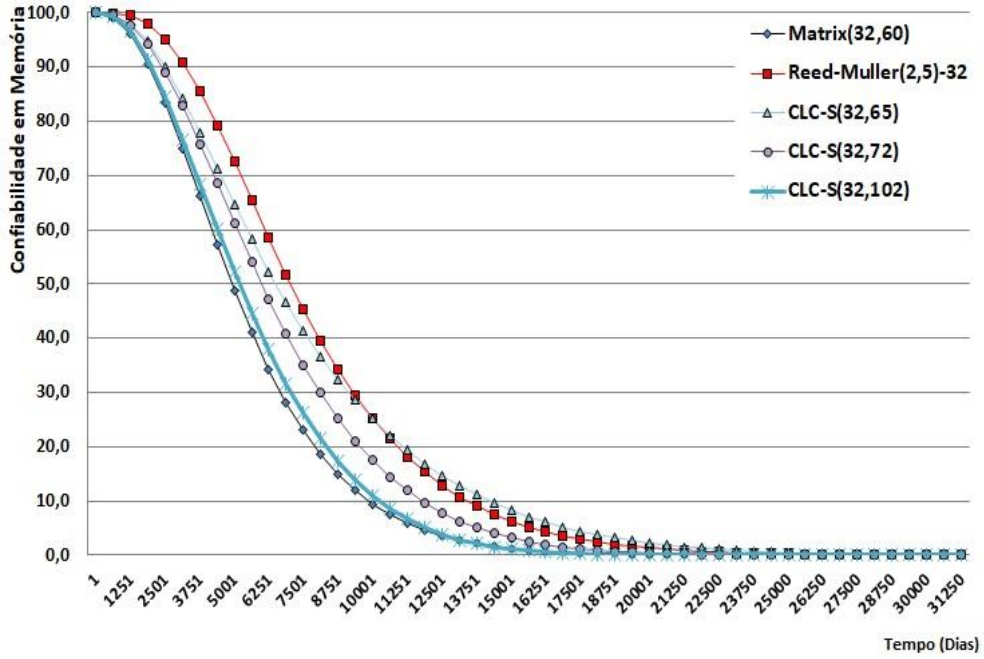
Fonte: Elaborada pelo Autor.

Figura 5.14b. Confiabilidade para registro de 16 (λ=10^{-5} upsets/bit/dia).



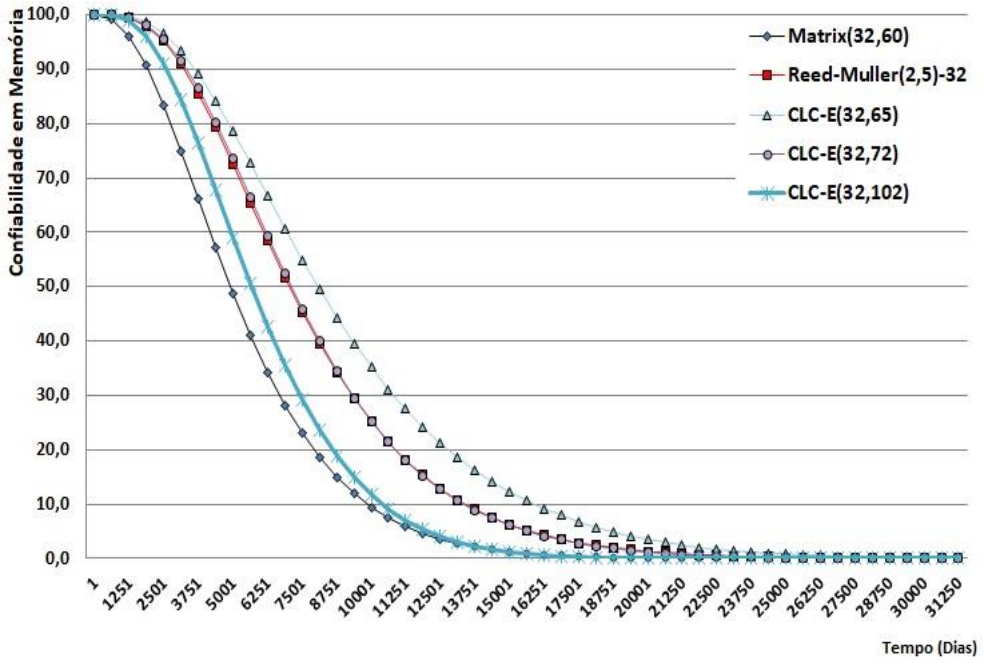
Fonte: Elaborada pelo Autor.

Figura 5.15a. Confiabilidade para registro de 32 (λ=10^{-5} upsets/bit/dia).



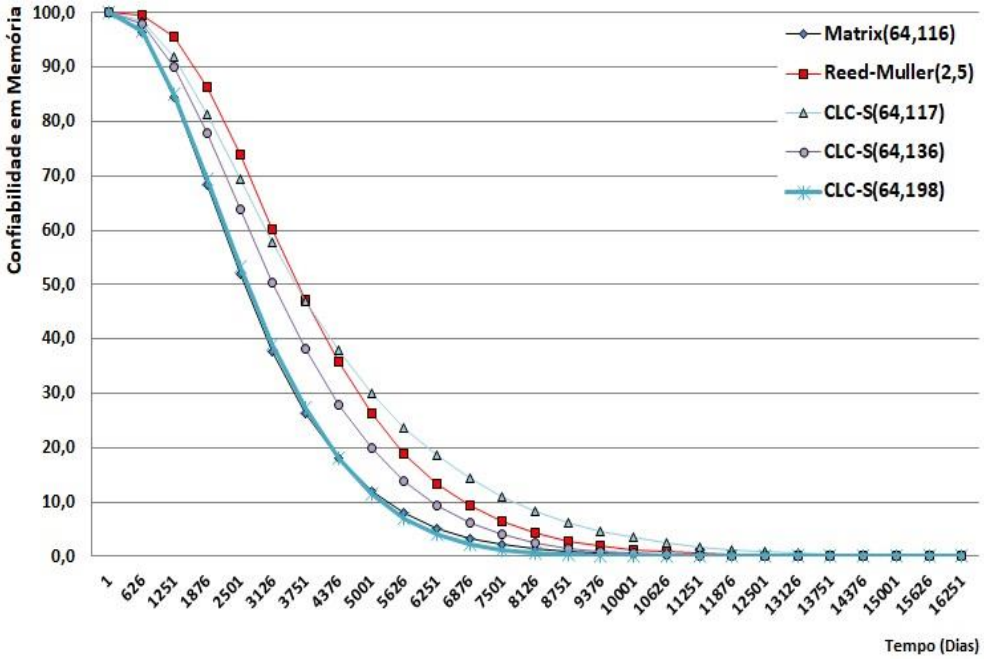
Fonte: Elaborada pelo Autor.

Figura 5.15b. Confiabilidade para registro de 32 (λ=10^{-5} upsets/bit/dia).



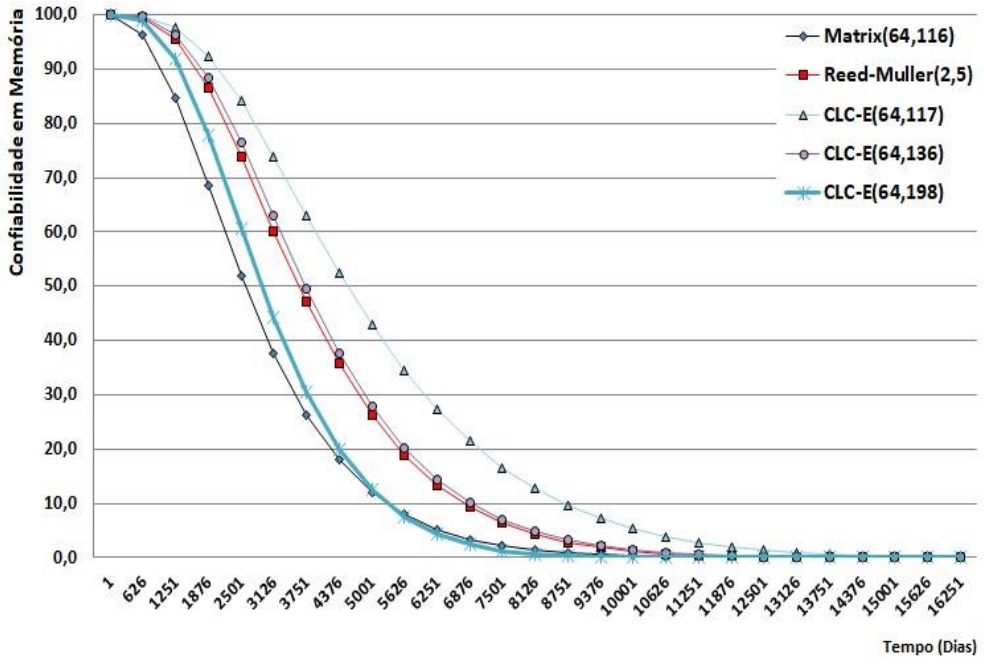
Fonte: Elaborada pelo Autor.

Figura 5.16a. Confiabilidade para registro de 64 (λ=10^{-5} upsets/bit/dia).



Fonte: Elaborada pelo Autor.

Figura 5.16b. Confiabilidade para registro de 64 (λ=10^{-5} upsets/bit/dia).



Fonte: Elaborada pelo Autor.

Assim como nos resultados de estimativa de correção e detecção, é possível perceber que a confiabilidade do CLC cresceu devido à manutenção das taxas de correção juntamente com a diminuição da redundância necessária para código, com o aumento da PDA aplicada. Novamente o CLC com $Ham - E(8,13)$ obteve os melhores resultados entre as

versões do CLC aplicadas. A Tabela 5.2 apresenta os dados de MTTF coletados a partir dos resultados apresentados nas Figuras 5.14, 5.15 e 5.16.

Tabela 5.2. MTTF para um registro de 16,32 e 64 bits.

CCE	**MTTF para 16 bits (10^4)**	**MTTF para 32 bits (10^4)**	**MTTF para 64 bits (10^4)**
Matrix	1,0923	0,55065	0,28531
Reed-Muller(2,5)	1,4514	0,78138	0,39680
CLC-S $Ham-E(8,13)$	1,3081	0,75686	0,41383
CLC-E $Ham-E(8,13)$	1,4969	0,90597	0,50247
CLC-S $Ham-E(4,8)$	1,2235	0,66447	0,34725
CLC-E $Ham-E(4,8)$	1,3938	0,78524	0,40942
CLC-S $Ham-E(2,6)$	1,1099	0,57317	0,29158
CLC-E $Ham-E(2,6)$	1,1931	0,61671	0,31437

Fonte: Elaborada pelo Autor.

Da Tabela 5.2 observa-se que o CLC-E com $Ham-E(8,13)$ obteve os melhores resultados de confiabilidade para todas as palavras de dados. Além disso, também pode-se extrair que a diferença percentual entre as versões do CLC com o Reed-Muller(2,5) diminuíram à medida que a PDA aplicada cresceu. A Tabela 5.3 mostra a diferença percentual entre as versões do CLC e Matrix para o código Reed-Muller(2,5).

Tabela 5.3. Diferença percentual para o Reed-Muller(2,5) para 16,32 e 64 bits.

CCE	**ΔMTTF (%)**	**ΔMTTF (%)**	**ΔMTTF (%)**
Matrix	24,74	29,52	28,09
CLC-S $Ham-E(8,13)$	9,87	3,31	-4,29*
CLC-E $Ham-E(8,13)$	-3,13*	-15,94*	-26,63*
CLC-S $Ham-E(4,8)$	15,7	14,96	12,48
CLC-E $Ham-E(4,8)$	3,96	-0,494*	-3,18*
CLC-S $Ham-E(2,6)$	23,52	26,64	26,51
CLC-E $Ham-E(2,6)$	17,79	21,07	20,77

Fonte: Elaborada pelo Autor.

Os valores com (*) significam que a diferença foi positiva para o código comparado ao Reed-Muller(2,5). Verifica-se que, para ambos os modos de correção dos CLCs com $Ham - E(8,13)$ e $Ham - E(4,8)$, o aumento da PDA aplicada contribuiu para melhorar a confiabilidade do Código. Somente para o CLC com $Ham - E(2,6)$, a variação da PDA não foi tão impactante na confiabilidade, logo a redução de redundância para essa versão do CLC se mostrou menos significativa.

5.3 Análise de Implementação Física

Por fim, esta seção apresenta e discute os resultados de implementação física dos CCEs apresentados. Os testes apresentados a seguir foram realizados a partir da sintetização dos CCEs descritos em Verilog. A ferramenta utilizada foi o *GENUS* (Cadance) com biblioteca de síntese de 65 nm. As Tabelas 5.4 e 5.5 apresentam os resultados de síntese dos codificadores e decodificadores dos ECCs para 16,32 e 64 bits, respectivamente.

Tabela 5.4. Dados de síntese dos codificadores dos CCEs para 16,32 e 64 bits.

PDA	**CCE**	**Área**		**Potência**		**Atraso**	
(bits)		**(µm²)**	**(%)**	**(mW)**	**(%)**	**(ns)**	**(%)**
16	**Reed-Muller (2,5)-16**	504	100	0,037	100	0,74	100
	Matrix(16,32)	298	59,1	0,01	27	0,15	20,3
	CLC(16,39)	402	79,7	0,023	62,2	0,39	52,7
	CLC(16,40)	435	86,3	0,024	64,8	0,35	47,3
	CLC(16,54)	258	51,2	0,011	29,7	0,41	55,4
32	**Reed-Muller (2,5)-16**	1.041	100	0,078	100	0,72	100
	Matrix(32,60)	622	59,76	0,025	32,06	0,27	37,5
	CLC(32,65)	754	72,44	0,048	61,54	0,53	73,62
	CLC(32,72)	762	73,2	0,046	58,98	0,66	91,67
	CLC(32,102)	511	49,09	0,025	32,06	0,53	73,62
64	**Reed-Muller (2,5)-64**	2.091	100	0,15	100	0,82	100
	Matrix(64,116)	1.259	60,2	0,054	36	0,4	48,78
	CLC(64,117)	1.601	76,6	0,107	71,3	0,64	78,04
	CLC(64,136)	1.485	71,1	0,103	68,6	1,16	141,6
	CLC(64,198)	1.023	48,9	0,052	34,6	0,66	80,5

Fonte: Elaborada pelo Autor.

Tabela 5.5. Dados de síntese dos decodificadores dos CCEs para 16,32 e 64 bits.

PDA	CCE	Área		Potência		Atraso	
(bits)		(μm²)	(%)	(mW)	(%)	(ns)	(%)
16	**Reed-Muller (2,5)-16**	4.312	100	0,737	100	2,12	100
	Matrix(16,32)	1.090	25,2	0,050	6,7	1,010	47,5
	CLC-S(16,39)	1.194	27,7	0,085	11,5	1,370	64,5
	CLC-E(16,39)	2.612	60,5	0,268	36,4	2,550	120
	CLC-S(16,40)	1.351	31,3	0,076	10,3	1,326	62,4
	CLC-E(16,40)	3.360	77,9	0,331	44,9	2,505	117
	CLC-S(16,54)	1.665	38,6	0,096	13,02	1,350	63,5
	CLC-E(16,54)	4.418	102,4	0,490	66,4	2,530	119
32	**Reed-Muller (2,5)-32**	8.724	100	1,45	100	2,42	100
	Matrix(32,60)	2.003	22,9	0,091	6,28	1,59	65,7
	CLC-S(32,65)	2.506	28,7	0,179	12,3	1,61	66,3
	CLC-E(32,65)	5.607	64,3	0,728	50,2	2,89	119,8
	CLC-S(32,72)	2.843	32,6	0,193	13,3	1,39	57,5
	CLC-E(32,72)	6.646	76,2	0,851	58,7	2,91	120
	CLC-S(32,102)	3.173	36,4	0,233	16,1	1,77	73,2
	CLC-E(32,102)	8.964	102,7	1,32	91,1	3,39	139,9
64	**Reed-Muller (2,5)-64**	18.324	100	3,05	100	2,57	100
	Matrix(64,116)	4.087	22,3	0,239	7,83	1,85	71,9
	CLC-S(64,117)	5.255	28,6	0,438	14,4	1,71	66,5
	CLC-E(64,117)	9.992	54,5	1,71	56,1	3,72	144,8
	CLC-S(64,136)	5.552	30,3	0,462	15,1	1,61	62,3
	CLC-E(64,136)	13.349	72,8	2,31	75,7	3,76	146,4
	CLC-S(64,198)	6.583	35,9	0,615	20,2	1,77	68,9
	CLC-E(64,198)	17.972	98,1	3,79	124,3	3,712	144,4

Fonte: Elaborada pelo Autor.

Dos resultados apresentados, temos que para todas as PDAs, o código Matrix obteve as menores taxas de síntese, também foi o código com menor capacidade de correção e detecção entre os analisados. Dentre as versões do CLC, as que apresentaram menores taxas foram as do CLC-S com $Ham - E(8,13)$ (CLC-S(16,39), CLC-S(32,65) e CLC-S(64,117)), pois utilizaram menos bits de redundância que as outras versões do CLC. O código Reed-Muller

e as versões do CLC-E com $Ham - E(2,6)$ (CLC-S(16,54), CLC-S(32,102) e CLC-S(64,198)) foram os códigos mais pesados em termos de síntese. O primeiro devido a utilização de lógica majoritária, dessa forma foi necessária a sintetização de muitas equações, elevando o custo. O segundo é a versão do CLC que gera mais bits de redundância entre todos os códigos, apresentando um impacto significativo para o custo.

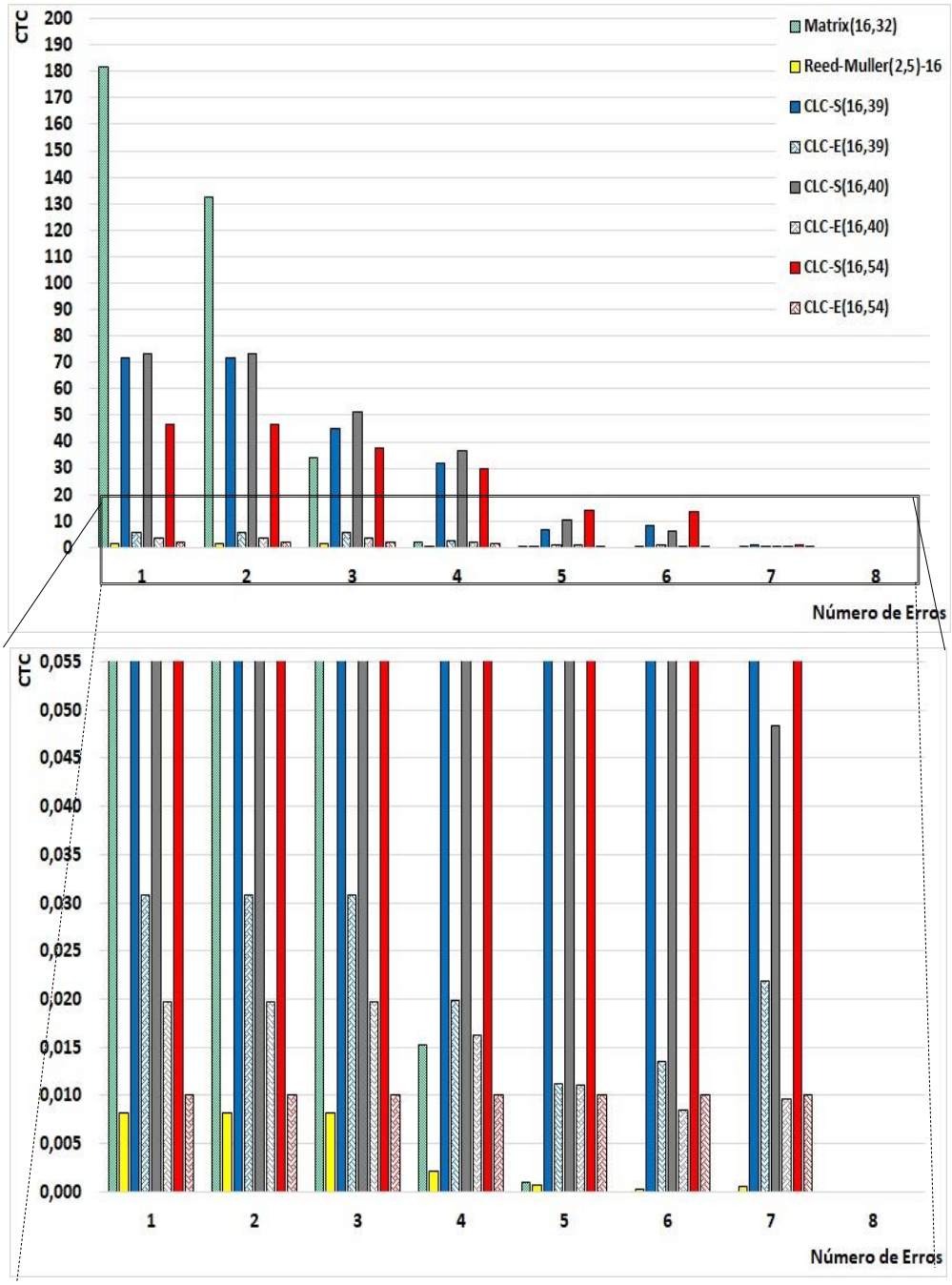
5.3.1 Análise de Cobertura Total por Custo (CTC)

De modo a melhor avaliar a relação entre cobertura contra erros com o custo de síntese do código, este trabalho utiliza a métrica proposta no trabalho (SILVA et al, 2018) chamada Cobertura Total por Custo (CTC). Nela são consideradas as taxas de correção e detecção dos CCEs e o custo total do decodificador, que será representado pelo produto entre os dados de área, potência e atraso. A Equação 5.9 representa o cálculo do CTC.

$$CTC = \frac{Taxa\ de\ Correção\ x\ Taxa\ de\ Detecção}{Área\ x\ Potência\ x\ Atraso} \tag{5.9}$$

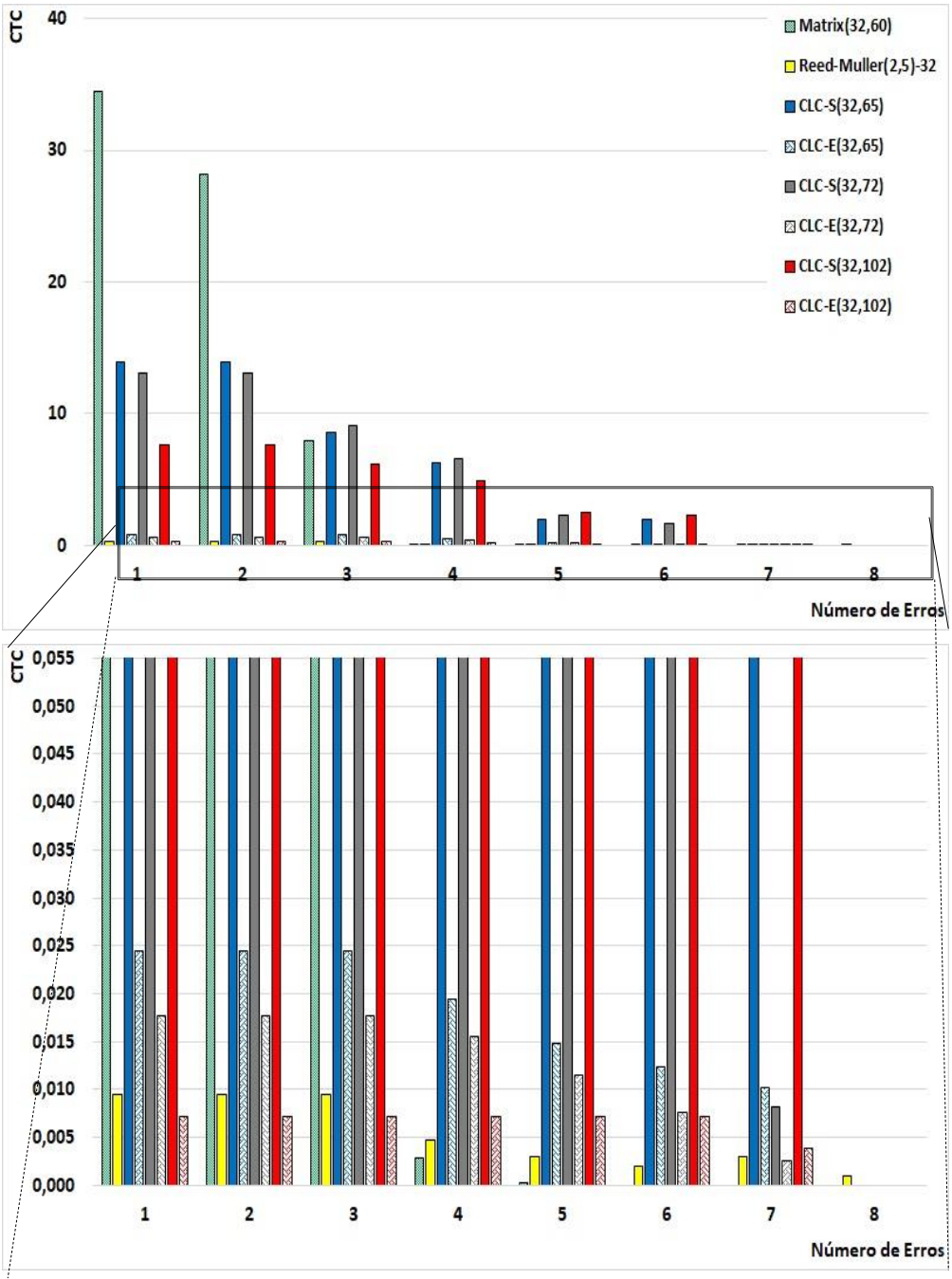
A partir dessa expressão, procura-se estimar qual o CCE apresenta a melhor relação custo-benefício para cada cenário de erro testado. As Figuras 5.17, 5.18 e 5.19 apresentam os gráficos dos CTCs dos CCEs para 16,32 e 64 bits respectivamente. Para melhor observação, a figura subsequente apresenta uma ampliação do CTC nos cenários mais críticos.

Figura 5.17. CTC de todos os CCEs para 16 bits.



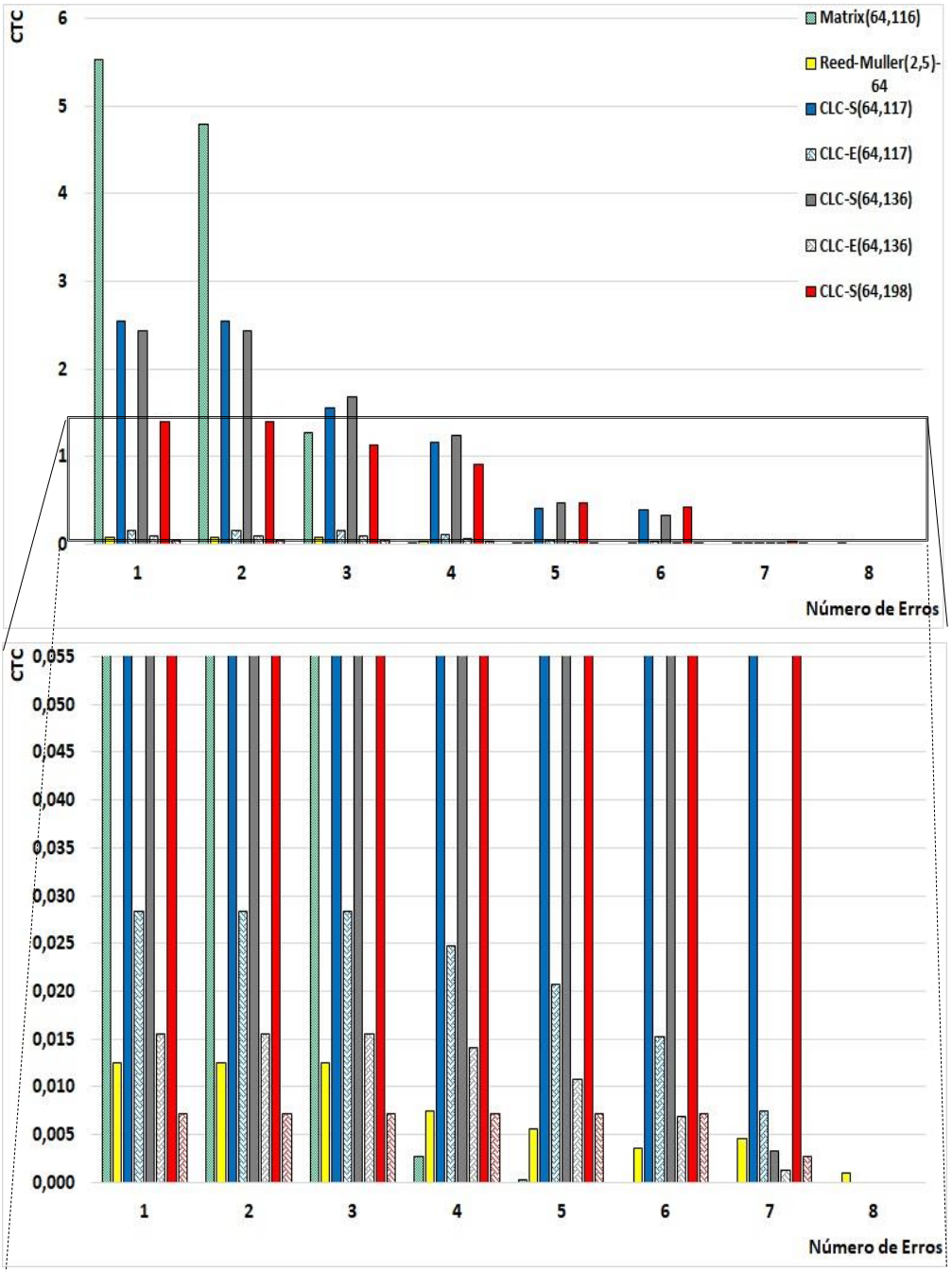
Fonte: Elaborada pelo Autor.

Figura 5.18. CTC de todos os CCEs para 32 bits.



Fonte: Elaborada pelo Autor.

Figura 5.19. CTC de todos os CCEs para 64 bits.



Fonte: Elaborada pelo Autor.

Das Figuras 5.17 a 5.19, pode-se concluir que o código Matrix, devido ao seu baixo custo e altas taxas de correção para 1 e 2 erros, apresentou os melhores CTC nesses 2 cenários. Porém, em cenários mais agressivos de erros (3 a 6 erros), as versões do código CLC-S apresentaram melhores resultados. Observe que há uma alternância entre qual dos CLC-S desempenham melhor nos gráficos apresentados. Devido ao custo dos CLC com $Ham - E(8,13)$ ser menor que das outras versões com o modo de correção equivalente, eles

apresentaram melhor custo benefício para situações de erros mais simples (1 e 2 erros) e para os últimos cenários de erros (6 e 7 erros). Os CLC com $Ham - E(4,8)$ obtiveram melhor desempenho em cenários intermediários (3 e 4 erros) por apresentarem melhores taxas de correção que os CLC com $Ham - E(8,13)$ para esses casos e menor custo que os CLC com $Ham - E(2,6)$.

Entre as versões do CLC, a formada por $Ham - E(2,6)$ apresentou os menores resultados de CTC de 1 a 4 erros, assim equiparou-se às outras versões em cenários muito críticos de erros (5,6 e 7), apesar de ter sido a versão com as melhores taxas de correção e detecção, seu alto custo de síntese pesou nessa análise. O código Reed-Muller, apesar de ter apresentado evolução nas taxas de correção e detecção, principalmente em função da aplicação de *interleaving* para os códigos de 32 e 64 bits, ofereceu resultados de CTC piores às versões do CLC-S e equiparáveis aos CLC-E com $Ham - E(4,8)$ e com $Ham - E(8,13)$ somente em cenários de 6 e 7 erros, porém superando a versão com $Ham - E(2,6)$ de 1 a 4 erros para os códigos de 64 bits.

6 CONCLUSÃO

Este trabalho apresentou a variação de estrutura do Código Linha Coluna(CLC), um CCE projetado para lidar com situações de múltiplos erros em memórias de aplicações críticas. A estrutura original do CLC consiste na aplicação de $Ham - E(4,8)$ e Paridade para formar uma matriz. Foram apresentadas duas novas estruturas desenvolvidas a partir da aplicação de $Ham - E(2,6)$ e $Ham - E(8,13)$. A aplicação do $Ham - E(2,6)$ objetiva aumentar a capacidade de correção do CLC, aproveitando-se do acréscimo de redundância maior que nas outras versões do CLC. Já a utilização do $Ham - E(8,13)$ propõe reduzir o número de bits de redundância gerados, bem como o custo geral de implementação do código.

As estruturas propostas foram implementadas para palavras de dados de 16, 32 e 64 bits. Para a validação delas, foram realizados: experimento de injeção de erros, que consistiu na estimação das taxas de correção e detecção; estimativa de confiabilidade, em que foi determinado taxas de confiabilidade e MTTF dos códigos e análise de implementação física, no qual os códigos foram sintetizados para a tecnologia de 65 nm e coletados seus dados de área, potência e atraso. No final dos experimentos, foi aplicada a métrica CTC, que foi desenvolvida com o propósito de estimar o limiar entre cobertura de erros (detecção e correção) e custo de síntese.

Os resultados coletados do experimento de injeção de erros mostraram que os códigos CLC, em especial os códigos CLC-E, apresentaram as melhores taxas de correção e detecção, principalmente para cenários agressivos de múltiplos erros (de 3 a 8 erros). Dentre os CLCs propostos, em ambos os modos de correção, os CLCs com $Ham - E(2,6)$ foram os que apresentaram os melhores resultados entre todos os códigos, mesmo com a melhoria dos resultados coletados pelo Reed-Muller(2,5) para palavras maiores de dados. Os resultados de estimativa de confiabilidade mostraram que o crescimento da palavra de dados favoreceu os resultados dos CLCs com $Ham - E(8,13)$. Isso ocorreu devido a dois fatores: esse código gerar menos bits de redundância que as demais versões apresentadas do CLC, e as taxas de correção e detecção terem sido melhores na maioria dos cenários testados em relação aos outros códigos. Observou-se que o crescimento da palavra de dados aplicada impulsionou os resultados de confiabilidade de todos os CLCs. Os resultados de implementação física mostraram que as novas estruturas do CLC obtiveram valores bem distintos. Para todas as palavras de dados testadas o CLC-S com $Ham - E(8,13)$ apresentou o menor custo dentre os CLCs, enquanto o CLC-E com $Ham - E(2,6)$ obteve o maior custo. Isso ocorreu como esperado, pois, o primeiro foi desenvolvido com o objetivo de reduzir o número de bits de

redundância gerados, dessa forma, diminuindo seu custo de implementação. Já o segundo, o grande número de bits gerados impulsionou o custo geral do código.

Esses resultados influenciaram decisivamente os resultados do CTC, pois observou-se que para múltiplos erros os códigos CLC-S superaram seus concorrentes, porém, houve alternância na liderança dentre os CLC-S. Para 3 e 4 erros, os CLC-S com $Ham - E(8,13)$ e $Ham - E(4,8)$ apresentaram resultados equivalentes. Para 5 e 6 erros, todos os CLC-S ficaram tecnicamente empatados, com pequena margem de liderança para a versão com $Ham - E(2,6)$ devido à sua grande capacidade de corrigir e detectar erros. Para 7 erros, os novos CLCs propostos superaram o CLC com $Ham - E(4,8)$ por terem conseguido corrigir e detectar mais erros.

Por fim, este trabalho atingiu os objetivos propostos que são a análise e desenvolvimento de novas versões do CLC, uma que apresentasse redução de número de bits gerados e de custo de síntese, e outra que obtivesse melhores taxas de correção para situações de múltiplos erros mais agressivas, proporcionando flexibilidade de aplicação do código CLC, além de, com as novas versões propostas, aumentar a capacidade de correção e detecção de erros para padrões mais agressivos de MBUs. Como trabalhos futuros, os novos formatos propostos podem ser aplicados em um sistema visando dinamizar a capacidade de correção ao selecionar o formato do CLC; e ao levar em consideração os recursos do sistema, e a implementação das novas versões do CLC em um ambiente de testes real, visando verificar a sua capacidade de correção em aplicações críticas.

REFERÊNCIAS

ARGYRIDES, C.; REVIRIEGO, P.; PRADHAN, D. K.; MAESTRO, J. A. Matrix-Based Codes for Adjacent Error Correction. **IEEE Trans. Nucl. Sci.,** v. 57, n. 4, p. 1, Ago. 2010.

ARGYRIDES, C.; ZARANDI, H. R.; PRADHAN, D. K. Matrix Codes: Multiple Bit Upsets Tolerant Method for SRAM Memories. **22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems**, 2007, Roma, p. 340–348.

BARNABY, H. J. Total-Ionizing-Dose Effects in Modern CMOS Technologies. **IEEE Transactions on Nuclear Science**, v. 53, n. 6, p. 3113–3121, Dec. 2006.

BAUMANN, R. The Impact of Technology Scaling on Soft Error Rate Performance and Limits to the Efficacy of Error Correction. **International Electron Devices Meeting**, 2002, São Francisco, p. 329-332.

BAUMANN, R. Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. IEEE Transactions on Device and Materials Reliability, v. 5, n. 3, p. 305–316, Set. 2005.

BROWNING, J. S.; KOGA, R.; KOLASINSKI, W. A. Single Event Upset Rate Estimates for A 16-K CMOS SRAM. **IEEE Transactions on Nuclear Science**, v. 32, n. 6, p. 4113–4119, Dec. 1985.

BOWMAN, R.C. Soft errors in commercial integrated circuits. **International Journal of High Speed Computing**. v. 14, n. 2, p. 299–309, 2004.

CARDARILI, G. C.; OTTAVI, M.; PONTARELLI, S.; RE, M.; SALSANO, A. Fault tolerant solid-state mass memory for space applications. **IEEE Trans. Aerosp. Electron. Syst.**, v. 41, n. 4, p. 1353–1372, Out. 2005.

CASTRO, H.; SILVEIRA, J.; COELHO, A.; SILVA, F.; MAGALHÂES, P.; LIMA, O. A Correction Code for Multiple Cells Upsets in Memory Devices for Space Applications. **14th IEEE International New Circuits and Systems Conference (NEWCAS)**, Vancouver, 2016.

CHEN, C. L.; HSIAO, M. Y.; Error-correcting codes for semiconductor memory applications: A state-of-the-art review. **IBM J. Res. Develop**. v. 28, n. 2, p. 124–134, Mar. 1984.

CHUGG, A. M.; MOUTRIE, M. J.; JONES, R. Broadening of the variance of the number of upsets in a read-cycle by MBU's. **IEEE Transactions on Nuclear Science**, v. 51, n. 6, p. 3701–3707, Dec. 2004.

DUTTA, A.; TOUBA, N. A. Multiple Bit Upset Tolerant Memory Using a Selective Cycle Avoidance Based SEC-DED-DAEC Code. **25th IEEE VLSI Test Symposium**, 2007, California, p. 349–354.

FAN, W. C.; DRUMM, C. R.; ROESKE, S. B.; SCRIVNER, G. J. Shielding Considerations for Satellite Microelectronics**. IEEE Transactions on Nuclear Science**, v. 43, n. 6, p. 2790-2796, Dec. 1996.

FERREYRA, P.; MARQUES, C.; FERREYRA, R.; GASPAR, J. Failure map functions and accelerated mean time to failure tests: New approaches for improving the reliability estimation in systems exposed to single event upsets. **IEEE Transactions on Nuclear Science**, v. 52, n. 1, p. 494–500, Feb. 2005.

GAILLARD, R. Soft Errors in Modern Electronic Systems. v. 41. **SPRINGER**, p. 26-53, 2011.

GARG, R.; KHATRI, S. P. Analysis and Design of Resilient VLSI Circuits. **SPRINGER**, p. 8-9, 2009.

GUENZER, C. S.; CAMPBELL, A. B.; SHAPIRO, P. Single Event Upsets in NMOS Microprocessors**. IEEE Transactions on Nuclear Science**, v. 28, n. 6, p. 3955-3958, Dec. 1981.

GURUN, G.; ZAHORIAN, J.; SISMAN, A.; KARAMAN, M.; HASLER, P.; DEGERTEKIN, F. An Analog Integrated Circuit Beamformer for High-Frequency Medical Ultrasound Imaging. **IEEE Transactions on Biomedical Circuits and Systems**, v. 6, n. 5, p. 454, Out. 2012.

GHERMAN, V.; EVAIN, S.; AUZANNEAU, F.; BONHOMME, Y. Programmable extended SEC-DED codes for memory errors. **29th IEEE VLSI Test Symposium (VTS)**, 2011, California, p. 140–145.

HAMMING, R. W. Error detecting and error correcting codes. **Bell Syst. Tech. Journal**, v. 29, n. 2, p. 147–160, Abr. 1950.

HAZUCHA, P.; SVENSSON, C. Impact of CMOS technology scaling on the atmospheric neutron soft error rate. **IEEE Transactions on Nuclear Science**, v. 47, n. 6, p. 2586–2594, Dec. 2000.

HEIJMEN, T. Soft Errors in Modern Electronic Systems. v. 41, SPRINGER, p. 1-25, 2011.
O'GORMAN, T. J. The Effect of Cosmic Rays on the Soft Error Rate of DRAM at Ground Level. **IEEE Transactions on Electron Devices**, v. 41, n. 4, Abr. 1994.

KING, M. P. et al. Electron-Induced Single-Event Upsets in Static Random-Access Memory. **IEEE Transactions on Nuclear Science**, v. 60, n. 6, p. 4122, Dez. 2013.

KOHLER, P.; POUGET, V.; WROBEL, F.; SAIGNÉ, F.; WANG, P. X., VASSAL, M. C. Analysis of Single-Event Effects in DDR3 and DDR3L SDRAMs Using Laser Testing and Mont-Carlo Simulations. **IEEE Transactions on Nuclear Science**, v. 65, n. 1, Jan. 2018.

LABEL, K. A.; BARNES, C. E.; MARSHALL, C. J.; JOHNSTON, A. H.; REED, R. A.; BARTH, J. L.; SEIDLECK, C. M.; KAVALI, S. A.; O'BRYAN, M. V. A roadmap for NASA's radiation effects research in emerging microelectronics and photonics. **IEEE Aerospace Conference**, 2000, Big Sky, p. 535-545.

LAI, Y.; CHEN, L.; CHIOU, W. A Memory Interleaving and Interlacing Architecture for Deblocking Filter in H.264/AVC. **IEEE Transactions on Consumer Electronics**, v. 56, n. 4, p. 2812-2818. Nov. 2010.

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)**, v. 26, n. 2, p. 221-222, Fev. 2018

MOON, T. Error Correction Coding, Mathematical Methods e Algorithms. **Wiley**. Hoboken, Nova Jersey. 2005.

MOREIRA, J.; FARRELL, P. Essentials of Error Control Coding. **Wiley**. The Atrium, Southern Gate, Chichester, West Sussex. 2006. p 64-65.

NEUBAUER, A.; FREUDENBERGER, J.; KÜHN, V. Coding theory: Algorithms, Architectures and Applications. **Wiley**. The Atrium, Southern Gate, Chichester, West Sussex. 2007. P. 27-31

NORMAND, E. Single event effects in avionics and on the ground. **Int. J. High Speed Electron. Syst.**, v. 14, no. 2, p. 285–298, 2004.

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, Dec. 2017.

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, Dec. 2005.

RAO, P.; EBRAHIMI, M. SEYYEDI, R.; TAHOORI, M. Protecting SRAM-based FPGAs Against Multiple Bit Upsets Using Erasure Codes. **51th ACM/EDAC/IEEE Autom. Conf.**, Jun. 2014, p. 1-2.

SAKATA, T.; HIROTSU, T.; YAMADA, H.; KATAOKA, T. A cost-effective dependable microcontroller architecture with instruction-level rollback for soft error recovery. **37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**, 2007, Edinburgh, p. 256-265.

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

SILVA, F.; FREITAS, W.; SILVEIRA, J.; FERREIRA, J.; MAGALHÃES, P.; LIMA JR, O., MARCON, C. Evaluation of Multiple Bit Upset Tolerant Codes For NoCs Buffering. **8th Latin American Symposium on Circuits and Systems (LASCAS)**, Bariloche, 2017.

SILVA, F.; SILVEIRA, J.; SILVEIRA, J.; MARCON, J.; VARGAS, F.; LIMA, O. An Extensible Code For Correcting Multiple Cell Upset in Memory Arrays. **Journal of Electronic Testing**, v. 34, n. 4, p. 417–433, Ago. 2018.

SMITH, E. C. Effects of Realistic Satellite Shielding on SEE Rates. **IEEE Transactions on Nuclear Science**, v. 41, n. 6, p. 2396–2399, Dec. 1994.

VARGHESE, B.; SREELAL, S.; VINOD, P.; KRISHNAN, A. R. Multiple bit error correction for high data rate aerospace applications. **IEEE Conference on Information and Communication Technologies (ICT)**, 2013, Choroni, p. 1086–1090.

YUANYUAN, C.; XUNYING, Z. Research and implementation of interleaving grouping hamming code algorithm. **International Conference on Signal Processing, Communication and Computing**, 2013, Kunming, p. 1–4.

.