

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Philippe de Souza Magalhães

PHICC: UMA FAMÍLIA DE CÓDIGOS CORRETORES DE ERROS PARA DISPOSITIVOS DE MEMÓRIA

Fortaleza – CE

19 de novembro de 2021

Philippe de Souza Magalhães

PHICC: UMA FAMÍLIA DE CÓDIGOS CORRETORES DE ERROS PARA DISPOSITIVOS DE MEMÓRIA

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Linha de Pesquisa: Sistemas de computação

Orientador: Otávio Alcântara de Lima Júnior

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação (PRPI)

Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Fortaleza – CE

19 de novembro de 2021



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
Av. 13 de maio, 2081 - Bairro Benfica - CEP 60040-531 - Fortaleza - CE - www.ifce.edu.br

PHILIPPE DE SOUZA MAGALHÃES

PHICC: UMA FAMÍLIA DE CÓDIGOS CORRETORES DE ERROS PARA DISPOSITIVOS DE MEMÓRIA

Dissertação submetida à Coordenação do Curso de Pós-graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para a obtenção do título de mestre em Ciência da Computação, área de concentração Sistemas de Computação.

Aprovada em 19/11/2021.

BANCA EXAMINADORA

Prof. Dr. Otávio Alcântara de Lima Júnior (Orientador)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - PPGCC/IFCE

Prof. Dr. Daniel Alencar Barros Tavares
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE

Prof. Dr. Jarbas Aryel Nunes da Silveira
Universidade Federal do Ceará - UFC

Prof. Dr. Corneli Gomes Furtado Júnior
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE



Documento assinado eletronicamente por **Otávio Alcântara de Lima Junior, Professor(a) do Ensino Básico, Técnico e Tecnológico**, em 19/11/2021, às 11:29, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Daniel Alencar Barros Tavares, Professor do Ensino Básico, Técnico e Tecnológico**, em 19/11/2021, às 12:10, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Jarbas Aryel Nunes da Silveira, Usuário Externo**, em 19/11/2021, às 17:24, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Corneli Gomes Furtado Junior, Professor do Ensino Básico, Técnico e Tecnológico**, em 22/11/2021, às 11:32, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade do documento pode ser conferida no site

[https://sei.ifce.edu.br/sei/controlador_externo.php?](https://sei.ifce.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0)

[acao=documento_conferir&id_orgao_acesso_externo=0](https://sei.ifce.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0) informando o código verificador **3161376** e o código CRC **4C30581A**.

Dedico este trabalho, com todo amor e carinho, à minha família, que, sem reservas, sempre me amou, me incentivou e contribuiu para que eu me tornasse quem sou.

Agradecimentos

Primeiramente agradeço a Deus pelo dom da vida, por ter sido a base de tudo e por conceder fortaleza, coragem e equilíbrio para enfrentar todos os desafios.

Aos meus pais, João Ronaldo Vasconcelos Magalhães e Maria Lucimar de Souza Magalhães, pelo amor, pelo incentivo, pela forma como batalharam para me proporcionar tudo o que me foi necessário e por me ensinarem os valores que sempre levarei comigo. Estendo os agradecimentos ao meu irmão, Romário Souza Magalhães, pelo companheirismo e amor fraterno.

À minha esposa, Maria Janaína de Sousa Silva Magalhães, que esteve sempre comigo como uma verdadeira companheira, nos bons momentos e nos mais difíceis. Obrigado pelo amor, dedicação e compreensão.

A todos os meus familiares que me incentivaram e ajudaram de forma direta e indireta.

Ao meu orientador, professor Dr. Otávio Alcântara, que, com empenho, me direcionou para o desenvolvimento deste trabalho. Obrigado pela oportunidade, pelo aprendizado, pelas reuniões e pela confiança.

Ao Programa de Pós-Graduação em Ciência da Computação (PPGCC) do Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), pelo acolhimento da pesquisa.

A todos os professores do PPGCC do IFCE, pelo conhecimento proporcionado e pelas experiências vivenciadas.

Aos amigos que estiveram comigo e acompanharam a minha trajetória, sendo verdadeiros companheiros. Obrigado pelos momentos proporcionados, experiências compartilhadas e por todo o suporte.

Aos meus colegas de trabalho do Departamento de Engenharia de Teleinformática, por todo o incentivo.

Aos que de alguma forma contribuíram para a conclusão deste trabalho.

A todos, meu sincero muito obrigado!

“Ninguém vai bater tão forte como a vida, mas a questão não é o quão forte você consegue bater, é o quão forte você pode ser atingido e continuar seguindo em frente. É o quanto você consegue aguentar e continuar tentando. É assim que a vitória é conquistada.”
(Rocky Balboa)

Resumo

Com a evolução da tecnologia no campo da microeletrônica, circuitos integrados (CIs) têm sido desenvolvido com maior redução das dimensões e da tensão de operação. Apesar das vantagens das reduções de escala, a ocorrência de *Multiple Bits Upsets* (MBUs), causados por interferências como a radiação ionizante, tem se tornado cada vez mais comum. Códigos corretores de erros (CCEs) são possíveis soluções, capazes de aumentar a tolerância a falhas de sistemas computacionais, no entanto, deve haver equilíbrio entre a eficácia de correção de erros e o custo de implementação em silício. Os requisitos de confiabilidade e os recursos de *hardware* disponíveis são os fatores determinantes para a escolha de um CCE. Ao observar esses aspectos, desenvolvemos uma nova família de códigos corretores de erros, denominada PHICC, capaz de abranger diferentes requisitos em termos de capacidade de correção e custo de implementação. Foram desenvolvidos quatro códigos: o PHICC(44,16), que garante maior proteção dos dados na ocorrência de até 8 erros e maior confiabilidade que códigos clássicos como Reed-Muller, mantendo o custo de implementação inferior; o PHICC(40,16), que é simples e possui altas taxas de correção de múltiplos erros; o PHICC(36,16), que é o mais equilibrado em termos de capacidade de correção por custo de implementação, sendo reduzido em quantidade de bits de redundância e mantendo alta confiabilidade; e o PHICC(32,16), que apesar de possuir um número maior de operações lógicas, consome a menor quantidade de memória e mantém altas taxas de correção de MBUs. A validação dos códigos PHICC foi realizada através de uma análise comparativa com outros CCEs em relação à eficácia de correção, confiabilidade, *Mean Time to Failure* (MTTF) e custo de implementação em silício. Os resultados mostram que os códigos PHICC podem ser fortes candidatos para a utilização em aplicações críticas, com taxas de correção de erros superiores em quase todos os cenários, cobrindo até 81,74% dos padrões de 8 erros. Além disso, todas as versões do PHICC, possuem resultados de capacidade de correção no tempo e confiabilidade superiores aos demais códigos testados. Em relação ao MTTF, todas as versões do PHICC se mostraram superiores, com diferença de até 31,82% em relação ao melhor resultado dos demais códigos.

Palavras-chave: Códigos Corretores de Erros, *Multiple Bit Upsets*, Confiabilidade, Tolerância a Falhas.

Abstract

With the evolution of technology in the microelectronics field, integrated circuits (ICs) have been developed with greater reduction in dimensions and operating voltage. Despite the advantages of downscaling, the occurrence of Multiple Bit Upsets (MBUs), caused by interference such as ionizing radiation, has become increasingly common. Error correction Codes (ECCs) are possible solutions, capable of increasing the fault tolerance of computer systems, however, there must be a balance between the effectiveness of error correction and the silicon implementation cost. Reliability requirements and available *hardware* features are the determining factors in choosing a ECC. By looking at these aspects, we have developed a new family of Error Correction Codes, called PHICC, which is capable of covering different requirements in terms of correctability and implementation cost. Four codes were developed: PHICC(44,16), which guarantees greater data protection in the event of up to 8 errors and greater reliability than classic codes such as Reed-Muller, keeping the cost of implementation lower; the PHICC(40,16), which is simple and has high multiple error correction rates; the PHICC(36,16), which is the most balanced in terms of correction capability by implementation cost, being reduced in the amount of redundancy bits and maintaining high reliability; and the PHICC(32,16), which despite having a greater number of logical operations, consumes the least amount of memory and maintains high MBU correction rates. The validation of PHICC codes was performed through a comparative analysis with other ECCs in relation to correction effectiveness, reliability, Mean Time to Failure (MTTF) and silicon implementation cost. The results shows that PHICC codes can be strong candidates for use in critical applications, with higher error correction rates in almost all scenarios, covering up to 81.74% of the 8 error patterns. In addition, all PHICC versions have superior correction in the time and reliability results compared to other tested codes. Regarding the MTTF, all the PHICC versions were superior, with a difference of up to 31.82% in relation to the best result of the other codes.

Keywords: Error Correction Codes, Multiple Bits Upsets, Reliability, Fault Tolerance.

Lista de ilustrações

Figura 1 – Relação entre capacidade de correção e custo de implementação.	26
Figura 2 – Single Event Effect.	30
Figura 3 – Relação entre SER e o nível de escala.	31
Figura 4 – Palavra codificada pelo CLC	37
Figura 5 – Palavra codificada pelo MRSC	39
Figura 6 – Regiões dos bits de dados	40
Figura 7 – Palavra codificada pelo PHICC(40,16).	45
Figura 8 – Processo de codificação PHICC(40,16).	45
Figura 9 – Organização dos dados em forma de matriz.	46
Figura 10 – Bits de paridade de dados.	46
Figura 11 – Processo de <i>interleaving</i> . (a) Conjunto de dados. (b) Permutação dos bits. (c) Ocorrência de MBU. (d) Redistribuição dos erros	47
Figura 12 – Bits de paridade de linhas permutadas.	47
Figura 13 – Bits de paridade de colunas permutadas.	48
Figura 14 – Processo de decodificação do PHICC(40,16).	49
Figura 15 – Decodificação de erro simples PHICC(40,16).	50
Figura 16 – Decodificação de erro duplo PHICC(40,16).	51
Figura 17 – Decodificação de erro triplo PHICC(40,16).	51
Figura 18 – Palavra codificada pelo PHICC(44,16).	52
Figura 19 – Processo de codificação PHICC(44,16).	53
Figura 20 – Conjunto de dados.	53
Figura 21 – Padrão de <i>interleaving</i> utilizado no PHICC(44,16).	54
Figura 22 – Bits de paridade de linhas.	55
Figura 23 – Bits de paridade de coluna.	55
Figura 24 – Bits de paridade de bits de checagem.	55
Figura 25 – Processo de decodificação.	57
Figura 26 – Decodificação de erro simples PHICC(44,16).	59
Figura 27 – Decodificação de erro duplo PHICC(44,16).	59
Figura 28 – Decodificação de erro triplo PHICC(44,16).	60
Figura 29 – Palavra codificada pelo PHICC(36,16).	61
Figura 30 – Processo de codificação PHICC(36,16).	61
Figura 31 – Bits de paridade de dados.	62
Figura 32 – Bits de paridade de linhas.	63
Figura 33 – Bits de paridade de coluna.	63
Figura 34 – Processo de decodificação do PHICC(36,16).	64
Figura 35 – Decodificação de erro simples PHICC(36,16).	66

Figura 36 – Decodificação de erro duplo PHICC(36,16).	66
Figura 37 – Decodificação de erro triplo PHICC(36,16).	67
Figura 38 – Palavra codificada pelo PHICC(32,16).	68
Figura 39 – Processo de codificação PHICC(32,16).	68
Figura 40 – Bits de paridade de dados.	69
Figura 41 – Padrão de <i>interleaving</i> utilizado no PHICC(32,16)	70
Figura 42 – Bits de paridade de verificação.	70
Figura 43 – Processo de decodificação do PHICC(32,16).	71
Figura 44 – Obtenção das síndromes de paridade de linha.	72
Figura 45 – Obtenção das síndromes de paridade de coluna.	73
Figura 46 – Decodificação de erro simples PHICC(32,16).	74
Figura 47 – Decodificação de erro duplo PHICC(32,16).	75
Figura 48 – Decodificação de erro triplo PHICC(32,16).	75
Figura 49 – Etapas de desenvolvimento dos códigos PHICC	79
Figura 50 – Etapas para obtenção das taxas de correção.	82
Figura 51 – Taxas de correção de erros.	87
Figura 52 – Capacidade de correção no tempo.	89
Figura 53 – Confiabilidade de registro de memória com 16 bits de dados.	90
Figura 54 – Confiabilidade de memória contendo 16 registros.	90
Figura 55 – Resultados para cobertura de correção por custo de implementação . .	92

Lista de tabelas

Tabela 1	–	Relação entre a síndrome e a posição de erro	34
Tabela 2	–	Vetores da matriz geradora G_{RM}	36
Tabela 3	–	Critérios para seleção de região	40
Tabela 4	–	Comparação entre diferentes versões do PHICC em número de bits de redundância em relação ao PHICC(40,16).	43
Tabela 5	–	Decodificação por bits de checagem PHICC(40,16).	50
Tabela 6	–	Decodificação por bits de checagem PHICC(44,16).	58
Tabela 7	–	Decodificação por bits de checagem PHICC(40,16).	65
Tabela 8	–	Características dos códigos PHICC	77
Tabela 9	–	MTTF (<i>Mean Time to Failure</i>), em dias, para memórias de 16 registros.	91
Tabela 10	–	Custos de área, potência, latência e custo total	92

Lista de abreviaturas e siglas

SEE	<i>Single Event Effect</i>
SBU	<i>Single Bit Upset</i>
MBUs	<i>Multiple Bit Upsets</i>
CCE	Código Corretor de Erros
SEC-DED	<i>Single Error Correction - Double Error Detection</i>
FPGA	<i>Field-Programmable Gate Array</i>
SRAM	<i>Static Random Access Memory</i>
PHICC	<i>Parity Hamming Interleaving Correction Code</i>
MRSC	<i>Matrix Region Selection Code</i>
MTTF	<i>Mean Time To Failure</i>
CI	Circuito Integrado
FIT	<i>Failures In Time</i>
SER	<i>Soft Error Rate</i>
TMR	<i>Triple Modular Redundance</i>
CLC	<i>Column Line Code</i>
CCC	Cobertura de Correção por Custo
TMR	Triple Modular Redundancy

Lista de símbolos

$\oplus$	Operação lógica XOR.
$\wedge$	Operação lógica AND.
λ	Taxa de ocorrência de erros.

Sumário

1	INTRODUÇÃO	25
1.1	Motivação para a Dissertação	25
1.2	Descrição do problema	26
1.3	Objetivos Geral e Específicos	27
1.4	Produção científica	28
1.5	Estrutura da Dissertação	28
2	FUNDAMENTAÇÃO TEÓRICA	29
2.1	Ocorrência de <i>Single Event Effect</i> (SEE) e seus efeitos	29
2.2	Relação entre <i>Soft Error Rate</i> (SER) e nível de escala	30
2.3	Códigos Corretores de Erros (CCEs)	31
2.3.1	Paridade	32
2.3.2	Códigos de Hamming	32
2.3.2.1	Obtenção da matriz geradora G e Codificação de Hamming(7,4)	33
2.3.2.2	Processo de Decodificação do Código de Hamming (7, 4)	34
2.3.3	Códigos Reed Muller	35
2.3.3.1	Processo de codificação do Reed-Muller(r,m)	35
2.3.3.2	Processo de decodificação do Reed-Muller(r,m)	36
2.3.4	Código CLC	37
2.3.4.1	Processo de codificação do CLC	37
2.3.4.2	Processo de decodificação do CLC	38
2.3.5	<i>O Matrix Region Selection Code</i> (MRSC)	38
2.3.5.1	Processo de codificação do MRSC	38
2.3.5.2	Processo de decodificação do MRSC	39
2.3.6	Trabalhos relacionados	40
2.4	Conclusão	41
3	CÓDIGOS PHICC	43
3.1	O código PHICC(40,16)	44
3.1.1	Codificador PHICC(40,16)	44
3.1.2	Decodificador PHICC(40,16)	48
3.1.3	Exemplos de correções feitas através do PHICC(40,16)	50
3.2	O código PHICC(44,16)	52
3.2.1	Codificador PHICC(44,16)	52
3.2.2	Decodificador PHICC(44,16)	56
3.2.3	Exemplos de correções feitas através do PHICC(44,16)	58

3.3	Código PHICC(36,16)	60
3.3.1	Codificador PHICC(36,16)	60
3.3.2	Decodificador PHICC(36,16)	63
3.3.3	Exemplos de correções feitas através do PHICC(36,16)	65
3.4	O código PHICC(32,16)	67
3.4.1	Codificador PHICC(32,16)	68
3.4.2	Decodificador PHICC(32,16)	70
3.4.3	Exemplos de correção do PHICC(32,16)	74
3.5	Aplicabilidade dos códigos PHICC	76
3.6	Conclusão	77
4	MATERIAIS E MÉTODOS	79
4.1	Desenvolvimento dos CCEs da família PHICC	79
4.1.1	Levantamento dos requisitos	79
4.1.2	Definição das estratégias de correção	80
4.1.3	Desenvolvimento dos CCEs em Matlab e execução de testes funcionais	81
4.1.4	Desenvolvimento dos CCEs em HDL e realização da síntese	81
4.2	Validação dos códigos	81
4.2.1	Taxas de correção de erros	81
4.2.2	Análise de capacidade de correção no tempo	82
4.2.3	Análise de confiabilidade e MTTF	83
4.2.4	Análise de custo de implementação em silício	84
4.3	Conclusão	85
5	RESULTADOS	87
5.1	Análise de correção de erros	87
5.2	Análise de capacidade de correção no tempo	88
5.3	Análise de confiabilidade e MTTF	90
5.4	Análise de custo de implementação	91
5.5	Conclusão	93
6	CONCLUSÃO	95
	REFERÊNCIAS	97
	ANEXOS	101
	ANEXO A – REPOSITÓRIO COM CÓDIGOS E SCRIPTS UTILIZADOS	103

1 Introdução

Devido à evolução da tecnologia no campo da microeletrônica, tornou-se possível desenvolver dispositivos capazes de atenderem a demanda por circuitos integrados de alta performance, cujas principais características são: maior densidade de transistores, maiores frequências de operação e menor tensão de alimentação. Apesar dos benefícios dessas evoluções, uma das consequências indesejadas é o aumento da susceptibilidade a Single Event Effects (SEEs) (CHUGG; MOUTRIE; JONES, 2004), causados pela incidência de partículas energizadas, como partículas alfa e nêutrons, em unidades de memórias de sistemas embarcados (IBE et al., 2010)(DAS; TOUBA, 2020). Essa radiação ionizante é capaz de alterar os valores armazenados em células de memória, como SRAMs, podendo ocorrer em um bit (Single Bit Upset – SBU) ou em múltiplos bits (Multiple Bit Upsets – MBUs), além disso, sua ocorrência aumenta conforme o nível de escala dos circuitos integrados diminui (PÉREZ-CELIS; THURLOW; WIRTHLIN, 2021)(RAO et al., 2014)(WIRTHLIN; LEE; QUINN, 2014).

Códigos Corretores de Erros (CCEs), principalmente do tipo matriciais, têm se tornado amplamente utilizados para garantir a integridade dos conteúdos dos dispositivos de memória (LIU et al., 2017)(RAHUL; YACHARENI, 2020). CCEs são estratégias baseadas na codificação de uma mensagem com a inserção de redundâncias, que são usadas na verificação da consistência dos dados. Quando um erro eventualmente ocorre, o processo de decodificação detecta e corrige os bits afetados. Em aplicações críticas, CCEs têm tido ampla utilização, prevenindo os sistemas computacionais de graves consequências (LABEL et al., 2000)(MAGALHÃES; ALCÂNTARA; SILVEIRA, 2019).

1.1 Motivação para a Dissertação

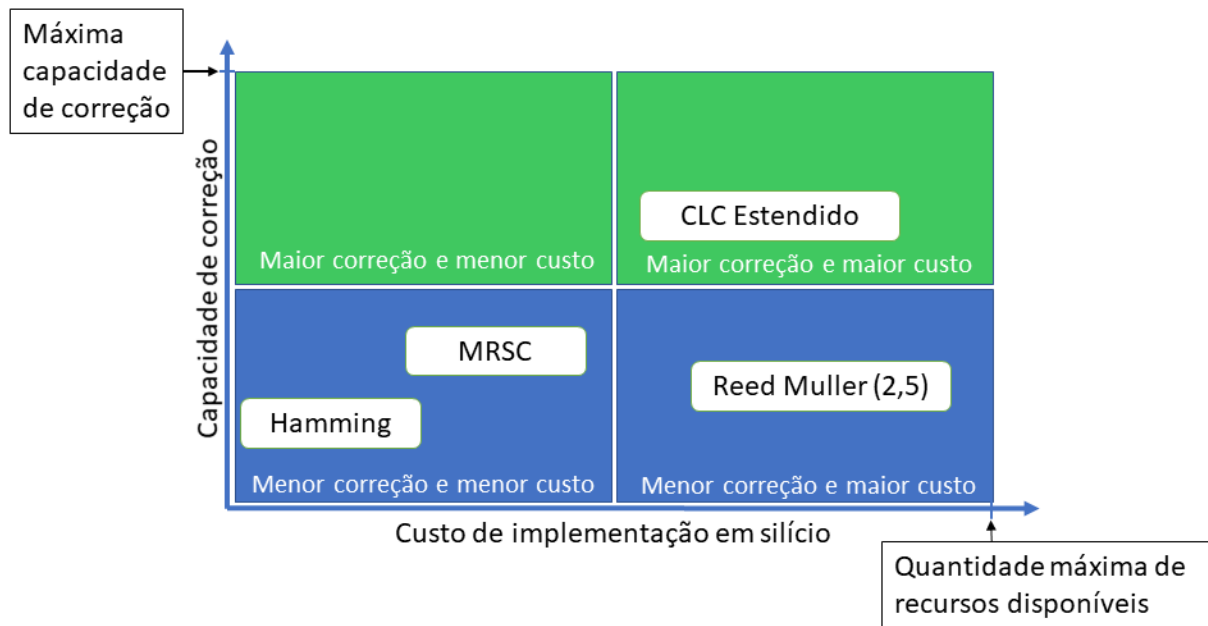
Os requisitos de confiabilidade de uma memória para sistemas críticos podem variar de acordo com a aplicação, assim como os recursos de *hardware* disponíveis. Por esse motivo, surge a necessidade de CCEs com diferentes características, no que diz respeito a capacidade de lidar com os erros e o custo de implementação em silício, para que melhor se adequem aos aspectos dos sistemas.

O custo de implementação e a capacidade de correção são os principais aspectos a serem observados ao selecionar algum CCE para a proteção de memórias (KUMAR et al., 2020). Aplicações críticas, cujos dispositivos são submetidos a uma maior exposição à radiação ionizante, exigem a utilização de CCEs mais robustos, capazes de garantir a confiabilidade dos dados, mesmo com alta incidência de MBUs. CCEs mais simples podem ser utilizados em dispositivos submetidos a condições menos hostis, com menor incidência

de MBUs, cujos padrões de erros são menos complexos. Ao observarmos esses fatos, percebemos que a utilização de um código capaz de garantir maior confiabilidade a um custo reduzido, pode permitir um melhor aproveitamento dos recursos de *hardware*. Além disso, o desenvolvimento de um código simples e menos oneroso, mas que mantenha uma alta capacidade de correção de erros, pode permitir que sistemas com menor capacidade computacional sejam utilizados em aplicações com alta incidência de MBUs.

As principais lacunas a serem preenchidas, no que diz respeito as características dos CCEs, é a combinação da capacidade de correção de MBUs, com o baixo custo de implementação e a alta capacidade de correção de MBUs mais complexos, com o custo de implementação reduzido em relação a estratégias clássicas. Na Figura 1, podemos observar como os códigos CLC Estendido (SILVA et al., 2018), MRSC (SILVA et al., 2017), Hamming e Reed Muller(2,5) estão posicionados em relação a esses aspectos, na qual as regiões em verde representam as situações a serem alcançadas.

Figura 1 – Relação entre capacidade de correção e custo de implementação.



Fonte: Autor.

1.2 Descrição do problema

CCEs do tipo *Single Error Correction, Double Error Detection* (SEC-DED) têm sido usados para corrigir padrões de erros simples e prevenir sistemas computacionais da ocorrência de falhas, entretanto este tipo de código não é eficaz na correção de MBUs (REVIRIEGO et al., 2020) (GHERMAN et al., 2011), por esse motivo, CCEs mais robustos são necessários para lidarem com padrões de erros mais complexos (LIU et al., 2017).

O código de Hamming Estendido (HAMMING, 1950) é um dos métodos essenciais a se considerar em um CCE (GOERL et al., 2018). Apesar de ser um código do tipo SECDED, quando combinado com outras estratégias, como *interleaving*, permite a correção de MBUs (LI et al., 2018) (LI; XIAO; WU, 2019)(NEALE; JONKMAN; SACHDEV, 2015). A ideia principal do *interleaving* é rearranjar as posições dos dados com a finalidade de separar os múltiplos erros no processo de decodificação, desse modo, erros de padrões mais complexos se tornam mais fáceis de corrigir.

Códigos simples, como Hamming estendido, tendem a ter um custo menor, no entanto podem não atender os critérios de confiabilidade. Por outro lado, códigos mais complexos, como o Reed-Muller(2,5), tendem a ter maior capacidade de correção, porém isso também está associado ao alto custo de implementação. De maneira geral, a redução de custo de um CCE está diretamente ligada diminuição da quantidade de operações lógicas e de bits de redundância. Já o aumento da capacidade de correção de erros está relacionado com o uso de técnicas mais complexas. A combinação de estratégias, pode permitir o desenvolvimento de CCEs mais eficazes e eficientes que soluções clássicas já utilizadas. Para que isso seja possível, é preciso combinar os pontos fortes das diferentes técnicas e buscar o equilíbrio entre o custo e os resultados obtidos.

1.3 Objetivos Geral e Específicos

Neste trabalho, temos o objetivo de propor uma família de CCEs matriciais, que, a partir da combinação de interleaving com códigos lineares, seja capaz de atender a diferentes demandas de aplicações críticas, garantindo alta capacidade de proteção dos dados em dispositivos de memória, tanto para sistemas com menores restrições de recursos quanto para sistemas com maiores limitações.

Em seu sentido mais particular, os seguintes objetivos específicos foram perseguidos:

- Desenvolver CCEs matriciais a partir da combinação de estratégias como paridade, *interleaving* e Hamming;
- Desenvolver uma versão da codificação de Hamming, capaz de corrigir erros duplos;
- Desenvolver CCEs matriciais utilizando uma codificação inspirada em Hamming, com redução nos bits de redundância;
- Desenvolver CCEs matriciais, cujos bits de redundância compartilhem os dados de paridade;
- Analisar a eficácia dos CCEs desenvolvidos na correção de MBUs;
- Realizar análises de capacidade de correção, confiabilidade e custo de implementação dos códigos desenvolvidos;
- Validar os códigos desenvolvidos a partir de uma comparação com outros CCEs.

1.4 Produção científica

Durante o curso deste mestrado, os seguintes trabalhos científicos foram aceitos e publicados, a saber:

- Magalhães, P., 2019. **PHICC: An Error Correction Code For Memory Devices**. 2019 32nd Symposium on Integrated Circuits and Systems Design (SBCCI), 2019, pp. 1-6.

1.5 Estrutura da Dissertação

O texto restante desta dissertação está organizado como se segue:

No Capítulo 2 são descritos os principais conceitos, as razões as quais os *Single Event Effects* (SEEs) ocorrem e algumas soluções já propostas para lidarem com os problemas gerados.

No Capítulo 3, a família de códigos PHICC é apresentada, juntamente com a descrição dos processos de codificação e decodificação. Também são apresentadas as principais características de cada código apresentado, bem como as justificativas para o seu desenvolvimento.

O Capítulo 4 relata quais os materiais e métodos utilizados nos processos de desenvolvimento, teste e validação dos CCEs apresentados. São apresentados os testes para obtenção das taxas de correção, capacidade de correção no tempo, confiabilidade, MTTF e custo de implementação em silício.

O Capítulo 5 apresenta os resultados obtidos a partir da realização dos testes descritos no Capítulo 4.

A dissertação é finalizada no capítulo 6, com a apresentação das conclusões e com as propostas de trabalhos futuros.

2 Fundamentação teórica

A constante evolução da tecnologia no campo da microeletrônica tem proporcionado a concepção de dispositivos cada vez mais eficazes e eficientes no desempenho de suas funções. Vários desses dispositivos estão presentes no cotidiano das pessoas, nas mais diversas aplicações, e têm se tornado cada vez mais necessários. Contudo, o grande nível de integração dos circuitos integrados que compõem esses dispositivos tem causado um aumento na sua sensibilidade a interferências externas. Um dos principais fatores é a radiação ionizante, a qual pode ocasionar mau funcionamento desses dispositivos (KISHANI et al., 2011).

Nesse capítulo, serão abordados os fundamentos que demonstram a relevância desse trabalho. O conceito de SEE's, bem como a maneira como ocorre e os seus efeitos para sistemas embarcados será demonstrado. Também serão apresentados alguns códigos corretores de erro que são capazes minimizar os efeitos do SEE através da correção dos SBU's e MBU's.

2.1 Ocorrência de *Single Event Effect* (SEE) e seus efeitos

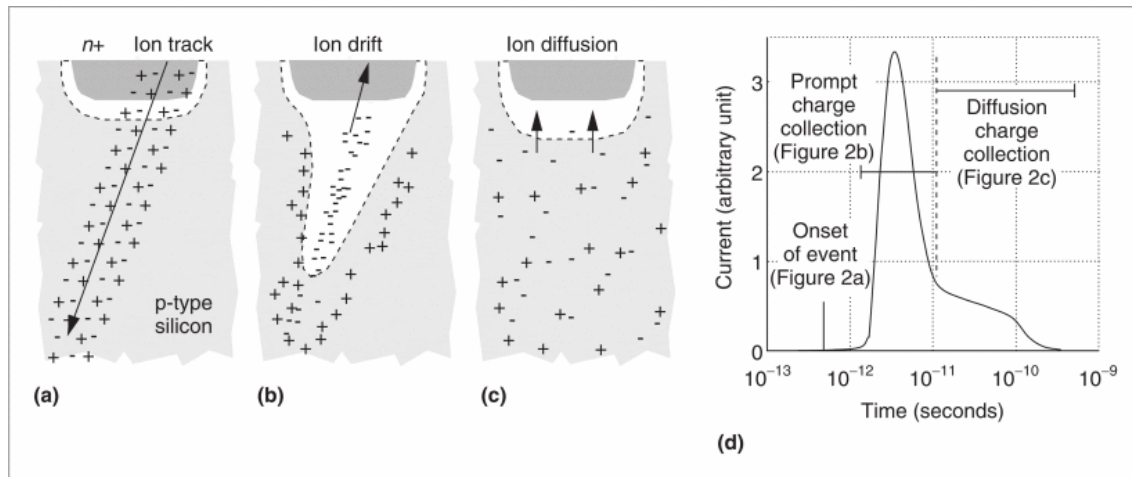
A crescente demanda por uma maior densidade e baixo consumo de potência, têm aumentado consideravelmente a sensibilidade à radiação, especialmente em aplicações espaciais. Os efeitos, denominados *Single Event Effect* (SEE), podem danificar seriamente os circuitos integrados e são causados principalmente por nêutrons atmosféricos, raios cósmicos e partículas alfa (KARAN; SRIKANTH; AGRAWAL, 2019) (SAMWEL et al., 2019) (SHARMA; VIJAYAKUMAR, 2012).

Na atmosfera, os raios cósmicos e as partículas solares incidem no topo da atmosfera e são atenuados pelas partículas de nitrogênio e oxigênio. As partículas resultantes mais importantes são os nêutrons. À medida que a altitude diminui, a densidade de nêutrons aumenta, alcançando o pico em cerca de 20 km de altitude. A partir de então, com o decréscimo da altitude, a densidade dessas partículas diminui, assumindo um valor no nível do solo de 1/500 do seu pico. Na Terra estão presentes, tanto as radiações naturais quanto as causadas por atividades humanas, dentre elas podemos citar a radiação terrestre, cósmica, as provenientes de tratamentos médicos e a nuclear (BATH; DYER; STASSINOPOULOS, 2003).

As junções de um transistor são partes muito sensíveis à radiação (BAUMANN, 2005). No início de um evento de radiação ionizante, observa-se a criação de um conjunto de pares elétrons-lacunas distribuídos em formato radial na passagem do íon (Figura

2a). Quando uma partícula energizada atravessa a região de depleção, um campo elétrico rapidamente atrai as partículas negativas para a junção $n+$, ocasionando uma diminuição na tensão. Com isso a região de depleção estende-se mais profundamente no substrato em formato de funil, cujo tamanho depende do nível de dopagem no substrato. A ampliação da região de depleção resulta no aumento considerável da eficiência no fluxo de portadores (Figura 2b). Alguns picosegundos depois o processo de difusão acontece (Figura 2c), no qual os portadores são recombinados ou difundidos na junção. A Figura 2d mostra o nível de corrente no decorrer das fases de ocorrência do SEE. Pode-se notar o aumento da corrente durante a segunda fase, nesse momento ocorre uma mudança de estado da informação armazenada.

Figura 2 – Single Event Effect.



Fonte: (BAUMANN, 2005)

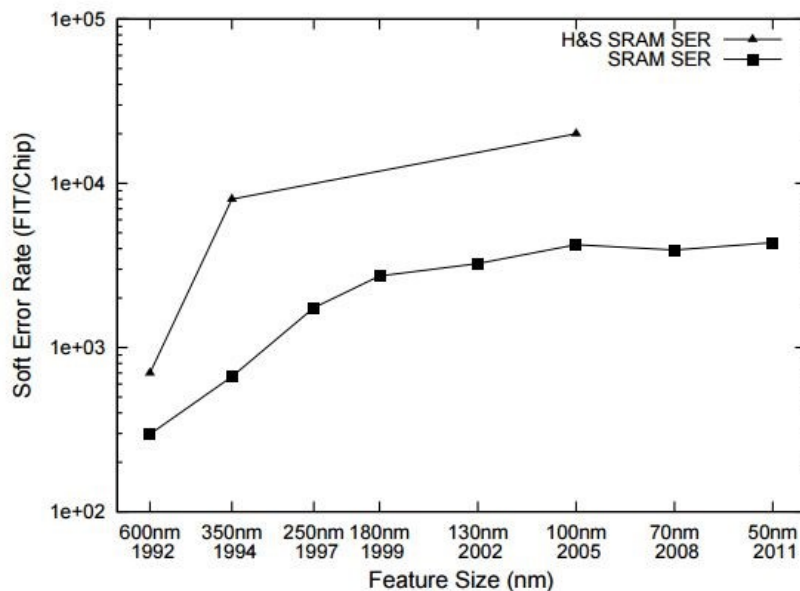
O foco desse trabalho está direcionado para dois tipos de SEE's: Single Bit Upset (SBU) e Multiple Bit Upset (MBU). SBU é a inversão do valor armazenado em uma célula de memória. MBU é a inversão de múltiplos bits armazenados em células de memória. Nos circuitos atuais, um nó nunca está isolado, e sim faz parte de um conjunto complexo de nós próximos uns dos outros. Desse modo a probabilidade de interferências parasitas influenciarem vários nós é muito maior.

2.2 Relação entre *Soft Error Rate* (SER) e nível de escala

A taxa de *soft errors* geralmente é expressa em *Failures In Time* (FIT), que indica a quantidade de falhas que podem ocorrer em um período de 10^9 horas de operação. Com a redução de escala dos transistores, a probabilidade de uma partícula incidir sobre o circuito, afetando o seu funcionamento, é reduzida devido diminuição da área ocupada. No entanto, circuitos cada vez mais complexos têm sido integrados em chip, já que se

tornou possível um melhor aproveitamento da área. No caso das memórias, em uma mesma área de chip, uma quantidade cada vez maior de células de armazenamento de dados tem sido acomodada aumentando consideravelmente a capacidade de armazenamento. Esse fator tem mantido a área utilizada em chip constante. Alguns estudos mostram que, para uma área constante de SRAM, a taxa de *soft errors* causada pela incidência de partículas energizadas aumenta linearmente conforme a escala diminui. A Figura 3 mostra os resultados obtidos através de um modelo realizado por Shivakumar et al. (2002) e Hazucha e Svensson (2000) que mostra como o nível de escala se relaciona com a taxa de *soft errors* e prevê como será o aumento de SER.

Figura 3 – Relação entre SER e o nível de escala.



Fonte: (SHIVAKUMAR et al., 2002)

A Figura 3 mostra como as taxas de *soft errors* (SER) aumentam significativamente com a redução de escala de 600nm para 50nm. A área do dreno de cada transistor, que é a área sensível à radiação, diminui a uma taxa quadrática, mas ao ocupar a mesma área, o número de bits aumenta quadraticamente, compensando esse efeito. Com a redução da tensão de alimentação e aumento da acumulação de carga no transistor, o valor da carga crítica, que é a carga mínima necessária para ocasionar um SEE em uma célula SRAM, também diminui. Esses são alguns fatores que explicam os resultados obtidos.

2.3 Códigos Corretores de Erros (CCEs)

Códigos corretores de erro são estratégias utilizadas para detectar e corrigir eventuais erros que venham acontecer nas informações armazenadas em memórias. CCEs têm se

tornado soluções bastante utilizadas em sistemas eletrônicos, em especial memórias (LI et al., 2019), devido à eficácia e o baixo custo de implementação, em comparação com outras estratégias utilizadas para tornar um sistema tolerante a falhas, tais como TMR (*Triple Modular Redundance*) (JUNG; CHOI, 2019) (JIN et al., 2018). A ideia central por trás dos códigos corretores de erro é a inserção de redundâncias na mensagem codificando-a. Essas redundâncias são utilizadas nos receptores na verificação de consistência da mensagem, havendo a detecção de um erro, é feita a correção através do processo de decodificação (SINGH; SINGH, 2012).

2.3.1 Paridade

A codificação de paridade é um esquema inteligível que pode ser utilizado para detectar um erro simples ou múltiplos erros ímpares. No entanto não é possível determinar a posição de ocorrência do erro. Quando os bits de uma mensagem são transmitidos ou armazenados, um bit adicional é acrescentado contendo o valor da paridade dos dados. O valor desse bit é obtido através de operações XOR com os bits da mensagem. Existem dois tipos de paridade, a paridade par e a paridade ímpar. Na paridade par, quando a quantidade de bits ‘1’ for ímpar, o bit adicional terá valor ‘1’, caso contrário o valor será ‘0’. Tomando por exemplo a seguinte mensagem: $M = 0\ 0\ 1\ 0$, o bit de paridade par seria obtido da seguinte forma, onde $\oplus$ representa a operação XOR

$$0 \oplus 0 \oplus 1 \oplus 0 = 1 \quad (2.1)$$

E a mensagem codificada a ser transmitida seria $MC = 0\ 0\ 1\ 0\ 1$.

A verificação de ocorrência de erros é feita recalculando a paridade dos bits de dados da palavra recebida e realizando a operação XOR com o bit de paridade transmitido junto com a mensagem, esse processo denomina-se síndrome. Suponhamos que a mensagem transmitida foi $MC=0\ 0\ 1\ 0\ 1$ e a recebida foi $MR=1\ 0\ 1\ 0\ 1$. Recalculando o bit de paridade par da mensagem recebida obteríamos:

$$1 \oplus 0 \oplus 1 \oplus 0 = 0 \quad (2.2)$$

Sendo assim a síndrome do bit de paridade recebido na mensagem com o bit de paridade recalculado seria $1\ \text{XOR}\ 0 = 1$ indicando a presença de erro.

2.3.2 Códigos de Hamming

Os códigos de Hamming(n,k) (HAMMING, 1950) compõem a primeira classe de códigos lineares a ser utilizada em correção de erros (SÁNCHEZ-MARCIÁN; REVIRIEGO;

MAESTRO, 2014). Ele é simples, do tipo SEC-DED. Isso quer dizer que ele é capaz de lidar com SBU's, corrigindo erros simples e detectando erros duplos.

Para cada inteiro positivo $m \leq 3$, existe um código de Hamming com os seguintes parâmetros:

Tamanho da palavra $n = 2^m - 1$;

Quantidade de bits de mensagem $k = 2^m - m - 1$;

Quantidade de bits de checagem $n - k = m$;

Capacidade de correção $t = 1$ ($d_{\min} = 3$).

Um código de Hamming(n, k) produz n bits na saída para cada k bits de dados de entrada.

2.3.2.1 Obtenção da matriz geradora G e Codificação de Hamming(7,4)

O processo de codificação do código de Hamming pode ser representado por uma matriz geradora (LIN; COSTELLO, 2005). O primeiro passo para a obtenção da matriz geradora G é a composição da matriz de paridade H , que possui m colunas e é arranjada conforme a Equação 2.3.

$$H = [I_m Q] \quad (2.3)$$

Onde I_m é a matriz identidade de dimensões $m \times m$ e Q consiste de uma matriz com vetores distintos entre si. Para o código de Hamming(7, 4), temos $n = 7$, $k = 4$ e $m = 3$. Desse modo, a matriz de paridade de comprimento 7 pode ser determinada como

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Dentre os possíveis vetores para compor Q , os selecionados foram: 110, 011, 111 e 101. Com a definição de H , a matriz geradora pode ser obtida, a partir da relação mostrada na Equação 2.4.

$$G = [Q^T I_{2^m - m - 1}] \quad (2.4)$$

Onde Q^T é a transposta de Q e $I_{2^m - m - 1}$ é uma matriz identidade, cujas dimensões são $(2^m - m - 1) \times (2^m - m - 1)$. O resultado obtido para a matriz geradora G é:

$$\mathbf{G} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Com a matriz G obtida, a mensagem codificada (MC) é obtida através do produto vetorial entre o vetor contendo os bits de dados e a matriz geradora G .

Tomando como exemplo a mensagem $M = [0010]$, obtemos a mensagem codificada da seguinte forma:

$$MC = (0 \ 0 \ 1 \ 0) \times \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} = (1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0)$$

2.3.2.2 Processo de Decodificação do Código de Hamming (7, 4)

O processo de decodificação dos códigos de Hamming é utilizado para determinar se houve a ocorrência de erros na mensagem recebida. Esse processo, chamado síndrome, consiste na multiplicação do vetor MC pela matriz de paridade, conforme a Equação 2.5.

$$S = [H \times MC^T] \quad (2.5)$$

O resultado dessa multiplicação indica se houve falha e qual a posição. Através da Tabela 1, observamos a relação entre o resultado do produto e a posição Pe correspondente ao erro.

Tabela 1 – Relação entre a síndrome e a posição de erro

S	Bit Pe
0 0 0	0 0 0 0 0 0 0
1 0 0	1 0 0 0 0 0 0
0 1 0	0 1 0 0 0 0 0
0 0 1	0 0 1 0 0 0 0
1 1 0	0 0 0 1 0 0 0
0 1 1	0 0 0 0 1 0 0
1 1 1	0 0 0 0 0 1 0
1 0 1	0 0 0 0 0 0 1

Fonte: Autor.

Quando o vetor Pe é nulo, significa que não houve erros. Tomando por exemplo a mensagem recebida $MR = [1\ 1\ 0\ 0\ 0\ 1\ 0]$, ao realizarmos a síndrome, obtemos $[0\ 0\ 1]$. Fazendo a correspondência na Tabela 1, temos $Pe = [0\ 0\ 1\ 0\ 0\ 0\ 0]$. A correção é feita através da seguinte operação:

$$\text{Mensagem corrigida} = Pe \oplus MR$$

$$\text{Mensagem corrigida} = [1\ 1\ 0\ 0\ 0\ 1\ 0] \oplus [0\ 0\ 1\ 0\ 0\ 0\ 0] = [1\ 1\ 1\ 0\ 0\ 1\ 0]$$

O código de Hamming possui uma variação chamada de Hamming estendido. Nessa versão, um bit adicional é utilizado, cujo valor nele armazenado é o resultado da operação de paridade par feita com os bits da mensagem codificada. Com essa variação, até duas falhas podem ser detectadas, no entanto a cobertura de correção se mantém. Para a mensagem $MC=[1\ 1\ 1\ 0\ 0\ 1\ 0]$, teríamos um bit '0' adicionado ao final da palavra, como resultado da operação XOR feita com os sete bits. Esse é o código de Hamming Ham(8,4). A detecção de duplo erro é feita quando a síndrome do bit de paridade adicional é '0', mas o vetor S é não nulo.

2.3.3 Códigos Reed Muller

Os códigos Reed-Muller são muito utilizados no envio de mensagens para longas distâncias ou através de canais onde podem ocorrer erros na mensagem. O código foi inventado em 1954 por D.E Muller e I.S. Reed. e estão entre os primeiros a serem utilizados em aplicações espaciais. Em 1972 um código Reed-Muller foi utilizado na missão Mariner 9, que transmitiu de imagens em preto e branco de Marte (VARGHESE et al., 2013) (SINGH; SINGH, 2012).

O código Reed-Muller(r,m) é um código corretor de erros antigo, mas com inúmeras aplicações, inclusive espaciais. Isso se deve ao nível de complexidade e robustez desse código que o torna capaz de corrigir inúmeros padrões de múltiplos erros. Contudo, o alto nível de complexidade, baseado em lógica de maioria, impacta diretamente no custo de implementação física.

2.3.3.1 Processo de codificação do Reed-Muller(r,m)

Na codificação do Reed-Muller(r,m), " r " indica a ordem e " m " indica o tamanho da palavra codificada, sendo igual a 2^m . De maneira geral, a dimensão da palavra codificada é dada pela Equação 2.6.

$$k = 1 + \binom{m}{1} + \binom{m}{2} + \dots + \binom{m}{r} \quad (2.6)$$

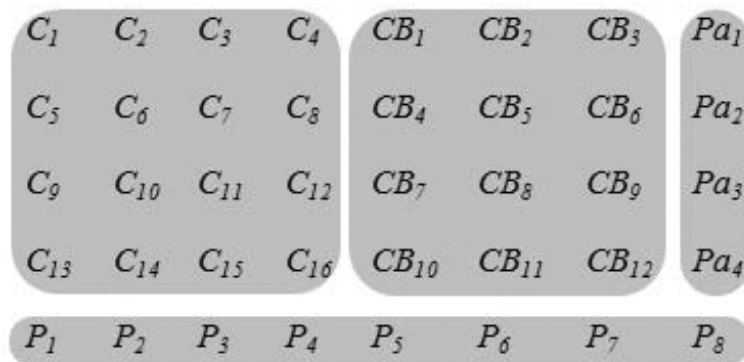
Sendo assim, para o Reed-Muller(2,5), $k=16$ e o tamanho da palavra é igual a 32 (CHEN; WANG, 2002). O ponto de partida é o cálculo de uma matriz geradora $G_{RM}(r, m)$,

2.3.4 Código CLC

O CLC é um código que possui dois modos de operação, o padrão (CASTRO et al., 2016) e o estendido (SILVA et al., 2018). Cada modo de operação é voltado para um objetivo. O modo padrão permite a correção de alguns padrões de MBU's com um custo de implementação reduzido. O nível de complexidade em relação ao código de Hamming é maior e em relação ao Reed-Muller(2,5) é inferior. No modo estendido, o CLC estendido é capaz de corrigir vários padrões de erro a mais que o modo padrão, sendo comparável a Reed-Muller(2,5). A diferença entre os dois modos de operação é a quantidade de vezes que o algoritmo de detecção e correção é aplicado à mensagem.

O código CLC é um código matricial baseado em Hamming estendido e paridade. Uma mensagem de 16 bits é codificada em uma palavra de 40 bits, cuja disposição é organizada conforme a Figura 4.

Figura 4 – Palavra codificada pelo CLC



Fonte: (CASTRO et al., 2016)

A palavra de 40 bits é composta por 16 bits de dados (C), 12 bits de checagem de Hamming (CB), 4 bits de paridade de linha (Pa) e 8 bits de paridade de coluna (P).

2.3.4.1 Processo de codificação do CLC

O processo de codificação inicia com a divisão da palavra de 16 bits em subconjuntos de 4 bits. Cada subconjunto passa pela codificação de Hamming estendido e colocado em uma linha da matriz. A informação codificada permite a correção de um erro simples e a detecção de um erro duplo.

Após a codificação dos quatro subgrupos de dados, realiza-se a composição da última linha da matriz com os bits de paridade de coluna (P), os quais são obtidos através de operações XOR entre os elementos das colunas, abrangendo tanto os bits de dados quanto os bits de checagem CBs . Depois do cálculo dos bits de paridade, o processo de codificação do CLC é concluído e a palavra é formada com um total de 40 bits.

2.3.4.2 Processo de decodificação do CLC

O primeiro passo na realização da decodificação é o cálculo das síndromes de paridade (SP) e bits de checagem (SCB). Para isso, primeiramente é realizado o recálculo dos bits de redundância CB através dos bits de dados C obtidos no receptor. Em seguida, operações XOR são realizadas entre os CB armazenados na palavra recebida e os recalculados (RCB). A Equação 2.9 demonstra como é realizada a operação.

$$SCB = CB \oplus RCB \quad (2.9)$$

De forma semelhante as síndromes dos bits de paridade SP e SPa são calculadas, conforme as Equações 2.10 e 2.11, e os resultados são utilizados para determinar a ocorrência de erros e as suas posições. A ocorrência de erro é evidenciada quando as síndromes geradas resultam em um valor diferente de zero.

$$SP = P \oplus RP \quad (2.10)$$

$$SPa = Pa \oplus RPa \quad (2.11)$$

No modo padrão, o CLC submete a palavra recebida ao processo de decodificação uma vez, enquanto que no modo estendido, a palavra passa pelo processo de correção duas vezes.

2.3.5 O Matrix Region Selection Code (MRSC)

O código MRSC (SILVA et al., 2017) é um código matricial, que permite a correção de MBUs. Sua capacidade de correção é próxima a obtida pelo CLC, no entanto o número de bits de redundância é reduzido. Este código é baseado em bits de paridade e uma lógica de seleção de regiões para determinar as posições dos erros.

2.3.5.1 Processo de codificação do MRSC

O processo de codificação do MRSC sobre um conjunto de 16 bits de dados resulta em uma palavra de 32 bits, cuja disposição dos bits pode ser observada através da Figura 5.

Figura 5 – Palavra codificada pelo MRSC

A_1	A_2	A_3	A_4	Di_1	Di_3	CbA_{13}	CbA_{24}
B_1	B_2	B_3	B_4	Di_2	Di_4	CbB_{13}	CbB_{24}
C_1	C_2	C_3	C_4	P_1	P_3	CbC_{13}	CbC_{24}
D_1	D_2	D_3	D_4	P_2	P_4	CbD_{13}	CbD_{24}

Fonte: (SILVA et al., 2017)

Os bits de dados são divididos em quatro subgrupos de quatro bits (A, B, C, D). Os bits sombreados em verde são os bits de paridade diagonais (Di), os quais são resultados de operações com bits de dados específicos dispostos diagonalmente entre si. Os bits sombreados em azul (P) são os bits de paridade de coluna, que resultam do cálculo de paridade de cada coluna dos bits de dados. Os bits em laranja (Cb) são os bits de checagem que resultam de operações XOR sobre bits de dados intercalados. Os bits de dados envolvidos nas operações são identificados em subscrito.

2.3.5.2 Processo de decodificação do MRSC

O processo de decodificação do MRSC inicia com a obtenção das síndromes dos bits de redundância, que são resultados das operações XOR entre os bits de redundância armazenados e os bits de redundância recalculados (RDi, RP e RCb). As síndromes de diagonal (SDi), paridade de coluna (P) e bits de checagem (SCb) são obtidas da seguinte forma:

$$SDi = Di \oplus RDi \quad (2.12)$$

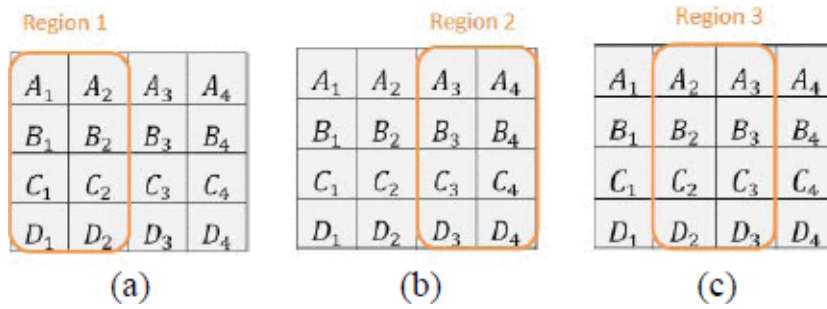
$$SP = P \oplus RP \quad (2.13)$$

$$SCb = Cb \oplus RCb \quad (2.14)$$

Após a obtenção das síndromes dos bits de redundância, o conjunto de dados é dividido em três diferentes regiões, conforme ilustrado pela Figura 6. Essas demarcações são utilizadas para limitar a atuação do código apenas nas regiões onde os erros estão presentes.

Algumas operações, baseadas em soma de inteiros, são utilizadas para determinar em qual região os erros estão posicionados. Na Tabela 3 podemos observar quais são os critérios utilizados para selecionar a região a qual o código irá realizar a correção.

Figura 6 – Regiões dos bits de dados



Fonte: (SILVA et al., 2017)

Tabela 3 – Critérios para seleção de região

Região selecionada	Critérios para seleção
Região 1	$(SDi_1 + SDi_2 + P_1 + P_2) > (SDi_3 + SDi_4 + P_3 + P_4)$
Região 2	$(SDi_1 + SDi_2 + P_1 + P_2) < (SDi_3 + SDi_4 + P_3 + P_4)$
Região 3	$(SDi_1 + SDi_2 + P_1 + P_2) = (SDi_3 + SDi_4 + P_3 + P_4)$

Fonte: (SILVA et al., 2017)

Para as regiões 1 e 2, o processo de correção é realizado através da operação XOR entre a região selecionada e a matriz de síndromes dos bits de checagem (*SCbs*). Quando a região 3 é selecionada, o procedimento de correção deve ser realizado com as posições da matriz de *SCbs* deslocadas.

2.3.6 Trabalhos relacionados

Em Karan, Srikanth e Agrawal (2019) são apresentados três CCEs capazes de corrigir até 5 erros em sequência. Esses códigos foram desenvolvidos para atuarem sobre 16, 32 e 64 bits de dados, resultando em uma palavra linear contendo 9, 10 e 11 bits de redundância respectivamente. O código se trata de uma melhoria realizada sobre a matriz de checagem de paridade desenvolvida em Li et al. (2018), que permite a correção de até quatro bits errados em sequência. Apesar da melhoria, o custo de implementação em silício foi significativamente acrescido.

Rahul e Yachareni (2020) apresentam dois códigos, um deles capaz de corrigir até dois erros e detectar até três erros adjacentes e o outro com capacidade de correção de até 3 erros adjacentes. Os códigos são comparados com outros do tipo SEC-DED. A matriz de geração de paridade apresentada permite uma redução de até 40% na quantidade de operações XOR de duas entradas, devido as regras estabelecidas.

Em (FARHEEN; PANDE, 2020) é realizada a detecção e correção de erros sobre um conjunto de dados de 32 bits. São utilizados três blocos contendo códigos de Hamming(16,11) reduzidos em número de bits de paridade. Nos dois primeiros blocos é possível detectar 4 erros e corrigir 2 erros. O terceiro bloco é capaz de atuar em apenas um erro.

Em (REVIRIEGO et al., 2013) uma estratégia de compartilhamento de paridade é demonstrada sobre um código matricial do tipo SEC-DED. A técnica consiste em realizar uma primeira codificação horizontal e em seguida realizar uma segunda codificação vertical sobre os bits de redundância do primeiro código. Somente os bits de redundância do segundo código são armazenados. O processo de correção é realizado com o recálculo da codificação horizontal e da detecção de erros a partir da decodificação vertical. desse modo, os bits de redundância do primeiro código são reconstruídos e a correção é realizada sobre os bits de dados.

Com o objetivo de criar um código, reduzido em número de bits de paridade, Kumar et al. (2020) utilizaram um artifício semelhante ao *interleaving* e uma estratégia de compartilhamento de paridade (REVIRIEGO et al., 2013). O código é do tipo matricial e a codificação de cada bit de redundância engloba bits de dados de várias linhas, em um formato que lembra a dupla hélice de DNA.

O CCE matricial PCoSA, apresentado em (FREITAS et al., 2020a), atua através de bits de paridade e da codificação e decodificação de Hamming Estendido, em duas dimensões. Para 16 bits de dados, uma palavra codificada pelo PCoSA possui um total de 48 bits de redundância, o que o torna capaz de corrigir, em média, 78% dos casos de MBUs com erros em até sete bits. As estratégias utilizadas refletem diretamente na alta capacidade de correção de erros e no alto custo de implementação associado.

Freitas et al. (2020b) apresentam um CCE matricial, que atua em memórias 3D (multicamadas), levando em conta a diferença de susceptibilidade a falhas de cada camada. O código é capaz de lidar com vários padrões de MBUs, com altas taxas de sucesso ao utilizar a codificação de Hamming em duas dimensões. No entanto, o modelo de codificação adotado eleva o custo de implementação consideravelmente, por necessitar de 48 bits de redundância para 16 bits de dados e por possuir uma grande quantidade de operações lógicas nos processos de codificação e decodificação.

2.4 Conclusão

Diante da crescente necessidade de estratégias capazes de lidar com MBUs em dispositivos de memória, vários códigos corretores de erros têm sido desenvolvidos. Alguns desses códigos são do tipo lineares e outros do tipo matriciais. A grande maioria não é capaz de lidar com MBUs mais complexos, podendo corrigir 2, 3, 4 ou até 5 erros de padrões específicos. Os códigos mais robustos, como Reed-Muller e CLC estendido, podem

corrigir uma maior quantidade de erros, no entanto o custo de implementação pode tornar o uso inviável em sistemas computacionais com restrições de recursos.

Entre os trabalhos analisados, percebemos que existem duas principais linhas de pensamento para o desenvolvimento de CCEs. O primeiro deles é voltado para a redução do custo de implementação em detrimento da capacidade de correção e o segundo é o aumento da capacidade de correção de erros tendo como consequência o aumento no consumo de recursos. Outras possibilidades podem ser exploradas, ao associar as vantagens de ambas as ideias. O desenvolvimento de códigos simples, mas com altas capacidades de correção, pode permitir o aumento da tolerância a falhas de sistemas limitados em recursos. Já o desenvolvimento de códigos mais robustos, com redução de custo em relação a códigos já utilizados, pode garantir o aumento da confiabilidade de sistemas críticos e permitir o melhor aproveitamento dos recursos de *hardware*.

No capítulo a seguir, propomos uma família de CCEs matriciais denominada PHICC, que foi desenvolvida com o objetivo de atender sistemas com diferentes requisitos e explorar as principais lacunas não solucionadas por CCEs já desenvolvidos.

3 Códigos PHICC

Devido ao aumento da susceptibilidade a erros transitórios em SRAMs, causados principalmente pela radiação ionizante, há uma crescente necessidade de estratégias que possam minimizar os efeitos e as consequências geradas por essas perturbações. A escolha de uma estratégia é determinada por diversos fatores, entre eles: o nível de proteção desejada, os recursos de hardware disponíveis e critérios de temporização. Em ambientes hostis, como o espaço por exemplo, estratégias mais eficazes na proteção dos dados são necessárias, no entanto em algumas aplicações, como pequenos satélites, os recursos de hardware são limitados e isso faz com que seja necessário haver equilíbrio entre o nível de proteção e os recursos demandados.

Com o objetivo de proporcionar soluções que melhor se adequem aos diferentes requisitos de capacidade de proteção e consumo de recursos de hardware, desenvolvemos uma nova família de códigos denominada PHICC (*Parity Hamming Interleaving Correction Code*), a qual possui quatro diferentes versões de CCEs. A primeira delas, o PHICC(40,16) (MAGALHÃES; ALCÂNTARA; SILVEIRA, 2019), possui 16 bits de dados, 24 bits de redundância e foi criado com a proposta de ser um CCE de elevada capacidade de correção de MBUs. A segunda versão é o PHICC(44,16), que possui 28 bits de redundância para 16 bits de dados e foi criado para atender a necessidade de elevada confiabilidade a partir de altos índices de correção de múltiplos erros. O PHICC(36,16) é a terceira versão dessa classe de códigos, que para 16 bits de dados, possui 20 bits de redundância, o que representa uma redução de 16,66% em relação ao PHICC(40,16). A quarta versão é o PHICC(32,16), que para 16 bits de dados, possui 16 bits de redundância, o que significa uma redução de 33,33% em relação ao PHICC(40,16). A Tabela 4 apresenta a diferença na quantidade de bits de redundância das novas versões do PHICC em comparação com o PHICC(40,16).

Tabela 4 – Comparação entre diferentes versões do PHICC em número de bits de redundância em relação ao PHICC(40,16).

Código	Bits de redundância	%
PHICC (40,16)	24	-
PHICC (44,16)	28	↑ 16, 66%
PHICC (32,16)	16	↓ 33, 34%
PHICC (36,16)	20	↓ 16, 66%

Fonte: Autor.

Os códigos PHICC são códigos matriciais semelhantes, com dois processos de correção. O primeiro deles é a correção por bits de paridade, no qual o erro é localizado

e corrigido através do cruzamento das informações dos bits que indicam a linha e a coluna errada. O segundo processo é a correção por bits de checagem, em que os erros remanescentes da primeira correção são localizados a partir do padrão obtido nas síndromes. Apesar das semelhanças, algumas características os diferenciam em relação as estratégias utilizadas, uso de memória, nível de complexidade e capacidade de correção.

Este capítulo está dividido em cinco seções. A Seção 3.1 aborda as razões pelas quais o código PHICC(40,16) foi desenvolvido, o processo de codificação, a decodificação e dois exemplos de correção. A Seção 3.2 relata as melhorias realizadas a fim de prover ao PHICC(44,16) maior capacidade de correção de erros. Essas melhorias são evidenciadas conforme os processos de codificação e decodificação são explicados. A Seção 3.3 mostra como é realizada a codificação e a decodificação do PHICC(36,16). Além disso, um novo método de codificação baseado em Hamming é apresentado e demonstrado através de exemplos de correção. Na Seção 3.4, os métodos de codificação e decodificação do PHICC(32,16) são explicados e justificados e assim como nas demais subseções, exemplos de correção são demonstrados. Por fim, a Seção 5 mostra a conclusão do capítulo.

3.1 O código PHICC(40,16)

O PHICC(40,16) foi criado com a proposta de ser um código robusto, como o Reed-Muller(2,5), mas com o custo de implementação inferior, próximo ao CLC Estendido. Essas qualidades permitem ao PHICC(40,16) ser um forte candidato para utilização em aplicações críticas. O código PHICC(40,16), além de associar o código de Hamming e paridade, também utiliza conceitos de *interleaving* (BAEG; WEN; WONG, 2009). Essa estratégia consiste na permutação de uma sequência de símbolos antes de enviá-los. No receptor, o processo inverso é realizado e a sequência original é obtida. Com isso, erros adjacentes que ocorrem durante transmissão dos dados são reposicionados de forma a favorecer o processo de correção.

Para um conjunto de dados de 16 bits, a palavra codificada através do PHICC possui um total de 40 bits, assim como o código CLC. Uma das principais diferenças entre o código PHICC e os demais, é o fato de haver a permutação dos bits de dados antes da transmissão. O acréscimo dessa estratégia eleva significativamente a eficácia no processo de correção. Além disso, a sequência dos passos de decodificação permite resolver padrões de erro complexos a partir da fragmentação em padrões mais simples.

3.1.1 Codificador PHICC(40,16)

O codificador do PHICC(40,16) constroi uma palavra, cuja a organização pode ser visualizada através da Figura 7, os quais C_{ij} são os bits de dados, Ppl_i são os bits de paridade de linha permutada, Ppc_j são os bits de paridade de coluna permutada, Dp_i são

os bits de paridade de linha dos dados e CB_k são os bits de checagem da codificação de Hamming(7,4).

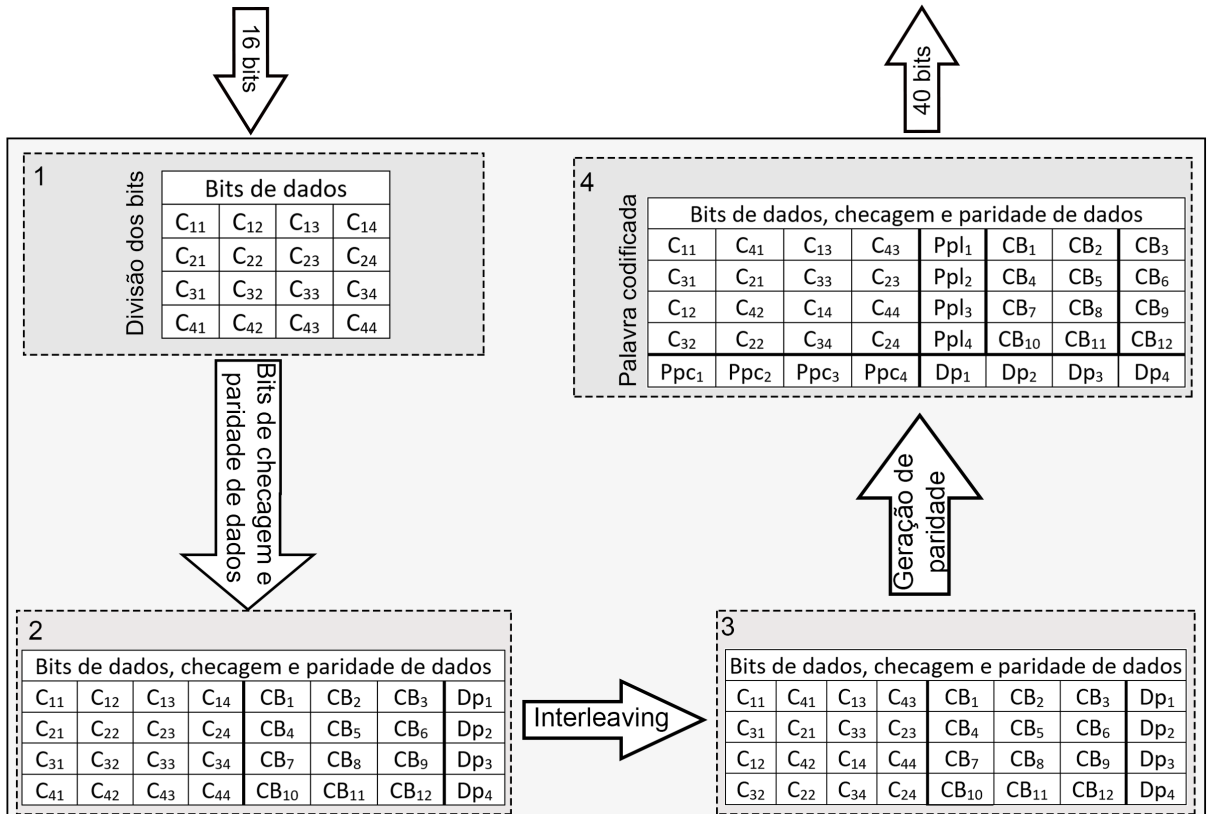
Figura 7 – Palavra codificada pelo PHICC(40,16).

C_{11}	C_{41}	C_{13}	C_{43}	Ppl_1	CB_1	CB_2	CB_3
C_{31}	C_{21}	C_{33}	C_{23}	Ppl_2	CB_4	CB_5	CB_6
C_{12}	C_{42}	C_{14}	C_{44}	Ppl_3	CB_7	CB_8	CB_9
C_{32}	C_{22}	C_{34}	C_{24}	Ppl_4	CB_{10}	CB_{11}	CB_{12}
Ppc_1	Ppc_2	Ppc_3	Ppc_4	Dp_1	Dp_2	Dp_3	Dp_4

Fonte: (MAGALHÃES; ALCÂNTARA; SILVEIRA, 2019)

O processo de codificação do PHICC(40,16) é realizado em quatro etapas: a primeira delas é a organização dos bits de dados em forma de matriz 4x4, a segunda é a codificação de Hamming(8,4), a terceira etapa consiste no processo de *interleaving* e a quarta é a geração dos bits de paridade. Através da Figura 8 podemos conferir, resumidamente, o passo a passo do processo de codificação.

Figura 8 – Processo de codificação PHICC(40,16).



Fonte: Autor.

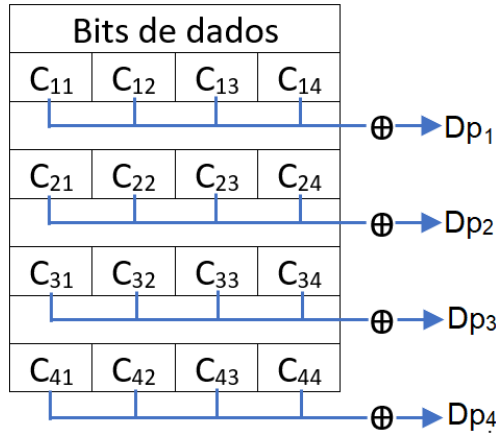
No primeiro estágio da codificação, os bits de dados são organizados em forma de matriz 4x4, cujas posições são especificadas como na Figura 9. Em um segundo momento, conforme demonstradas pela Figura 10, operações XOR são realizadas sobre os bits de cada linha da matriz de dados, para a obtenção dos bits de paridade de dados.

Figura 9 – Organização dos dados em forma de matriz.

C_{11}	C_{12}	C_{13}	C_{14}
C_{21}	C_{22}	C_{23}	C_{24}
C_{31}	C_{32}	C_{33}	C_{34}
C_{41}	C_{42}	C_{43}	C_{44}

Fonte: Autor.

Figura 10 – Bits de paridade de dados.



Fonte: Autor.

O segundo estágio da codificação é concluído com a obtenção dos bits de checagem da codificação de Hamming, que é regida pelas Equações 3.1 a 3.3.

$$CB_{(i=1,4,7,10)} = C_{[(i+2)/3],2} \oplus C_{[(i+2)/3],3} \oplus C_{[(i+2)/3],4} \quad (3.1)$$

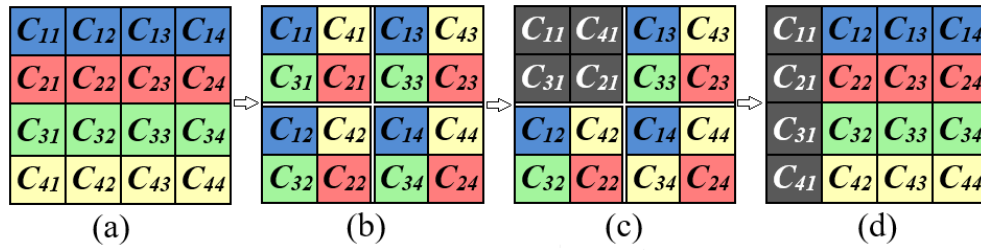
$$CB_{(i=2,5,8,11)} = C_{[(i+1)/3],1} \oplus C_{[(i+1)/3],3} \oplus C_{[(i+1)/3],4} \quad (3.2)$$

$$CB_{(i=3,6,9,12)} = C_{(i/3),1} \oplus C_{(i/3),2} \oplus C_{(i/3),4} \quad (3.3)$$

A terceira etapa é a permutação do conjunto de dados, na qual os bits de cada coluna são agrupados em quatro quadrantes. Este procedimento é executado para evitar vários erros na mesma linha durante o processo de decodificação. Ocorrendo até quatro

bit-flips em células próximas, o padrão de permutação permite a redistribuição dos bits afetados em quatro linhas diferentes. Isso permite que as síndromes dos bits de paridade de dados detectem as linhas que contêm erros e que os bits de checagem realizem a correção. A Figura 11 ilustra como os MBUs (células sombreadas) podem ser separados.

Figura 11 – Processo de *interleaving*. (a) Conjunto de dados. (b) Permutação dos bits. (c) Ocorrência de MBU. (d) Redistribuição dos erros

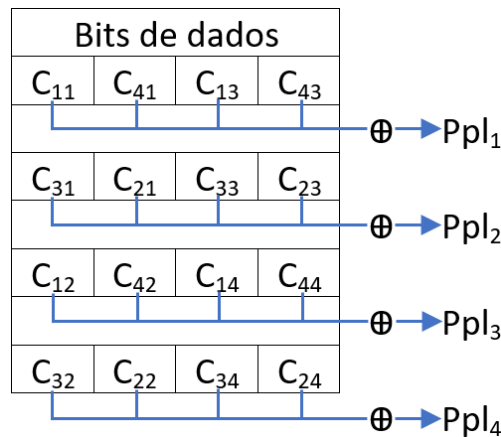


Fonte: (MAGALHÃES; ALCÂNTARA; SILVEIRA, 2019)

Na Figura 11a, o conjunto de dados são agrupados por linhas. Na Figura 11b, os bits de dados mudam suas posições, de forma que cada quadrante da matriz de dados seja ocupado por um elemento de cada linha. Na Figura 11c, podemos visualizar a ocorrência de 4 erros adjacentes nas células sombreadas em cinza. Na Figura 11d, podemos perceber que, quando os bits de dados retornam para as suas posições de origem, os erros são separados, com cada um ocupando uma das linhas da matriz de dados.

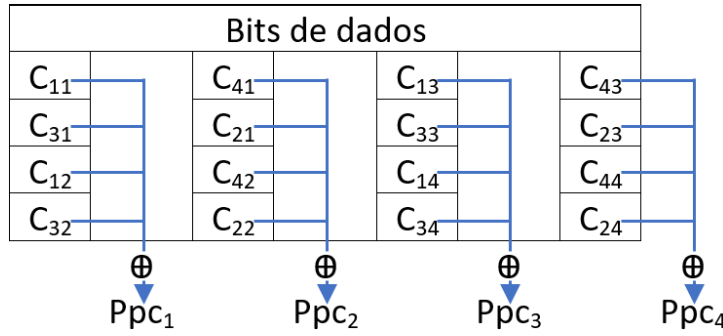
A quarta etapa consiste na geração dos bits de paridade de linhas e de colunas permutadas, cujas operações são demonstradas nas Figuras 12 e 13.

Figura 12 – Bits de paridade de linhas permutadas.



Fonte: Autor.

Figura 13 – Bits de paridade de colunas permutadas.



Fonte: Autor.

Após a realização de todas as etapas de codificação, a palavra codificada, contendo 40 bits, é colocada na saída do codificador.

3.1.2 Decodificador PHICC(40,16)

O decodificador do PHICC(40,16) atua através da correção por bits de paridade e da correção por codificação de Hamming(8,4). Os passos realizados durante o processo e correção de erros podem ser visualizados através da Figura 14.

Ao receber uma mensagem codificada, primeiramente é feita verificação de ocorrência de erros através dos bits de paridade Plp e Pcp . Para isso, as síndromes de paridade de linha permutada ($SPlp$) e as síndrome de paridade de coluna permutada ($SPcp$) são calculadas a partir das Equações 3.4 e 3.5, onde Plp' são os bits de paridade de linha permutada, recalculados utilizando os bits de dados (C) da mensagem recebida e Pcp' são os bits de paridade de coluna permutada, obtidos de forma similar.

$$SPpl = Ppl' \oplus Ppl \quad (3.4)$$

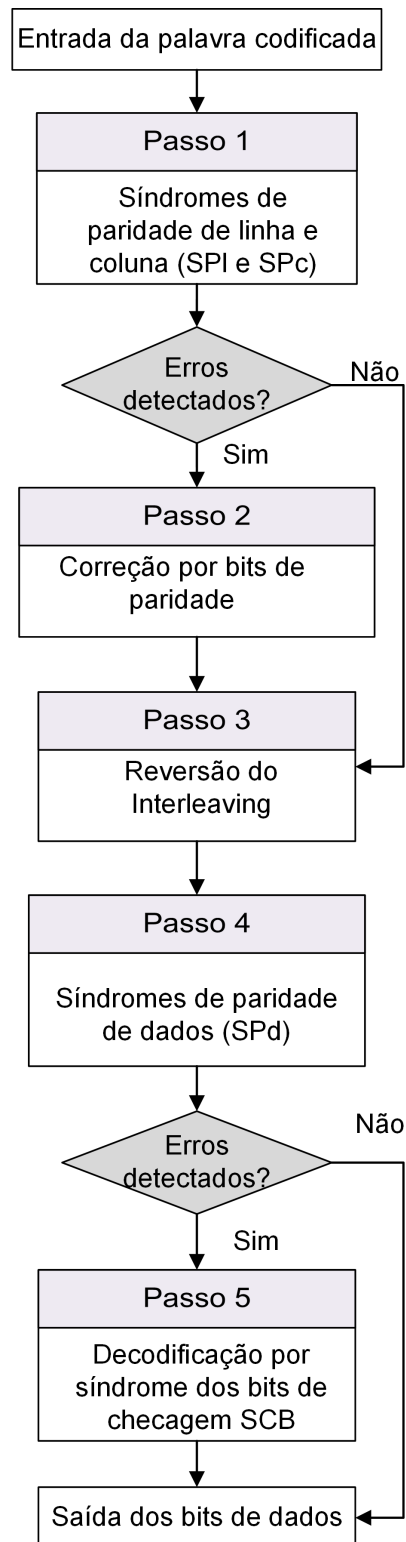
$$SPpc = Ppc' \oplus Ppc \quad (3.5)$$

Após o primeiro processo de correção por bits de paridade, o processo inverso do *interleaving* é realizado, a fim de permitir a detecção de erros remanescentes. Realiza-se então o cálculo das síndromes de paridade de dados (SDp), através da Equação 3.6. Caso algum erro seja detectado, a decodificação de Hamming é executada com das síndromes dos bits de checagem (SCB), conforme a Equação 3.7.

$$SDp = Dp' \oplus Dp \quad (3.6)$$

$$SCB = CB' \oplus CB \quad (3.7)$$

Figura 14 – Processo de decodificação do PHICC(40,16).



Fonte: Autor.

A Tabela 5 mostra a relação entre os resultados das síndromes dos bits de checagem e as posições dos erros. Ao final de todo o processo, os bits de dados são colocados na saída do decodificador.

Tabela 5 – Decodificação por bits de checagem PHICC(40,16).

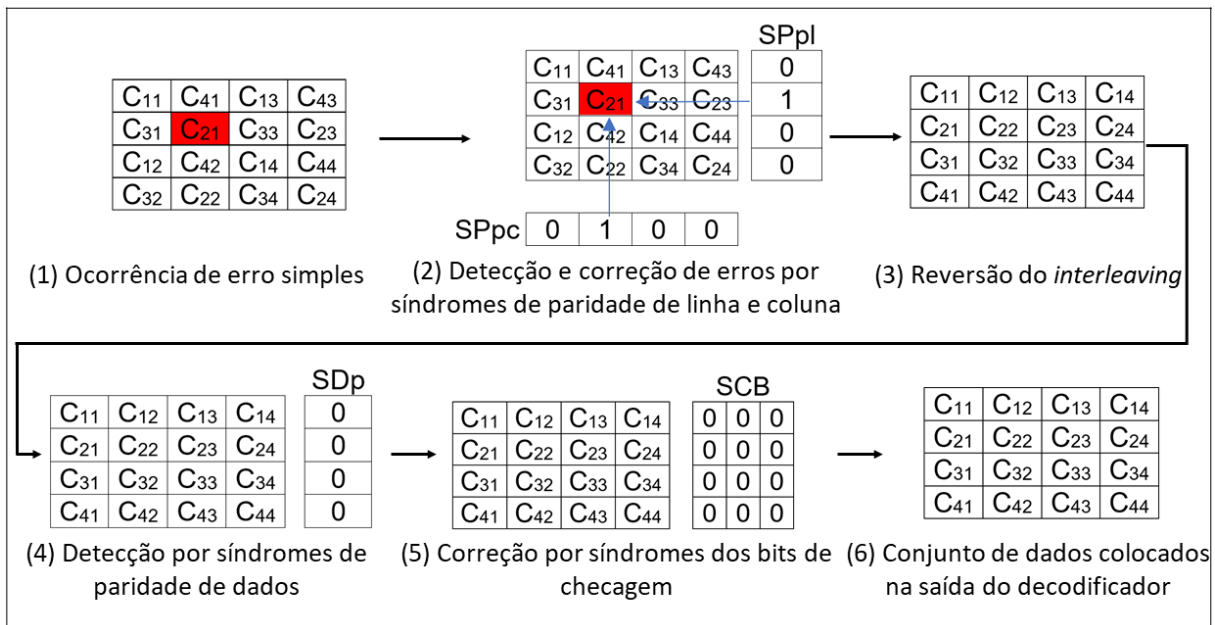
SCB ₁	SCB ₂	SCB ₃	Bits afetados
0	0	0	Sem erro
0	1	1	1
1	0	1	2
1	1	0	3
1	1	1	4

Fonte: Autor.

3.1.3 Exemplos de correções feitas através do PHICC(40,16)

Nesta subseção, três exemplos do processo de correção de erros através do PHICC(40,16) serão demonstrados. O primeiro exemplo, mostrado na Figura 15, consiste de um erro simples sobre os bits de dados. Podemos notar que para este padrão de erro as síndromes dos bits de paridade de linha e coluna são capazes de apontar a posição do erro, permitindo a correção. Após a primeira etapa de correção, nenhum erro remanescente é deixado para o segundo processo de correção e os bits de dados são colocados na saída do decodificador.

Figura 15 – Decodificação de erro simples PHICC(40,16).

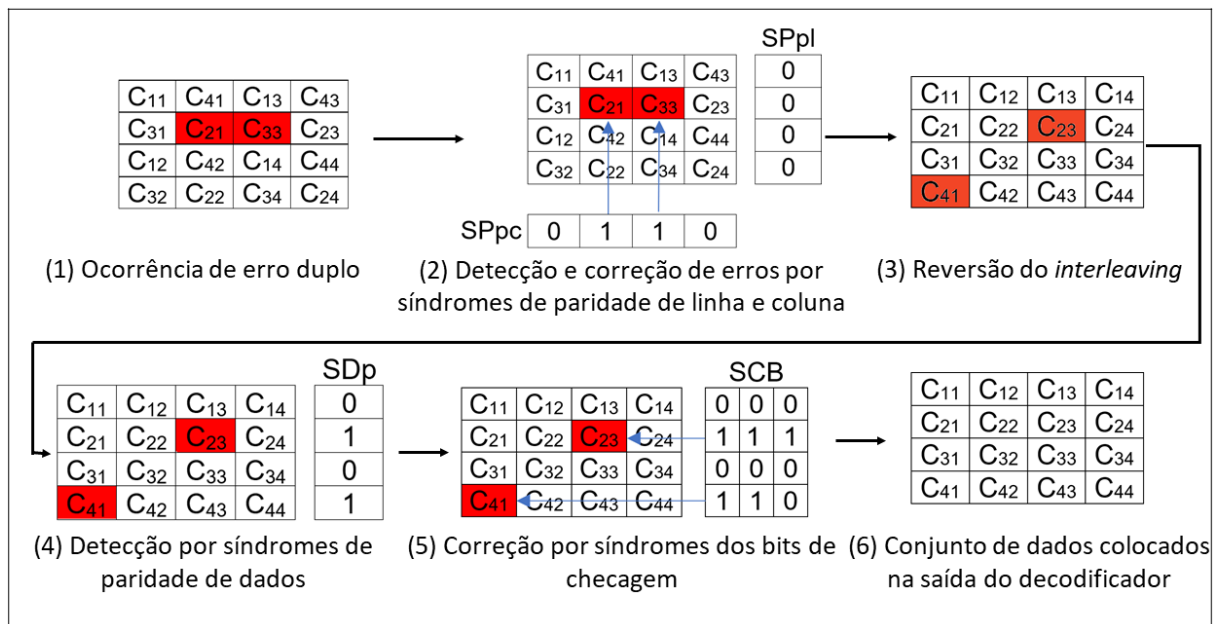


Fonte: Autor.

Na Figura 16 podemos visualizar o processo de correção de um padrão de erro duplo. No segundo estágio, o processo de correção por bits de paridade não é capaz de determinar as posições dos erros, portanto a correção não é realizada. Após a reversão do *interleaving*, os erros são separados, permitindo a detecção por síndrome de paridade

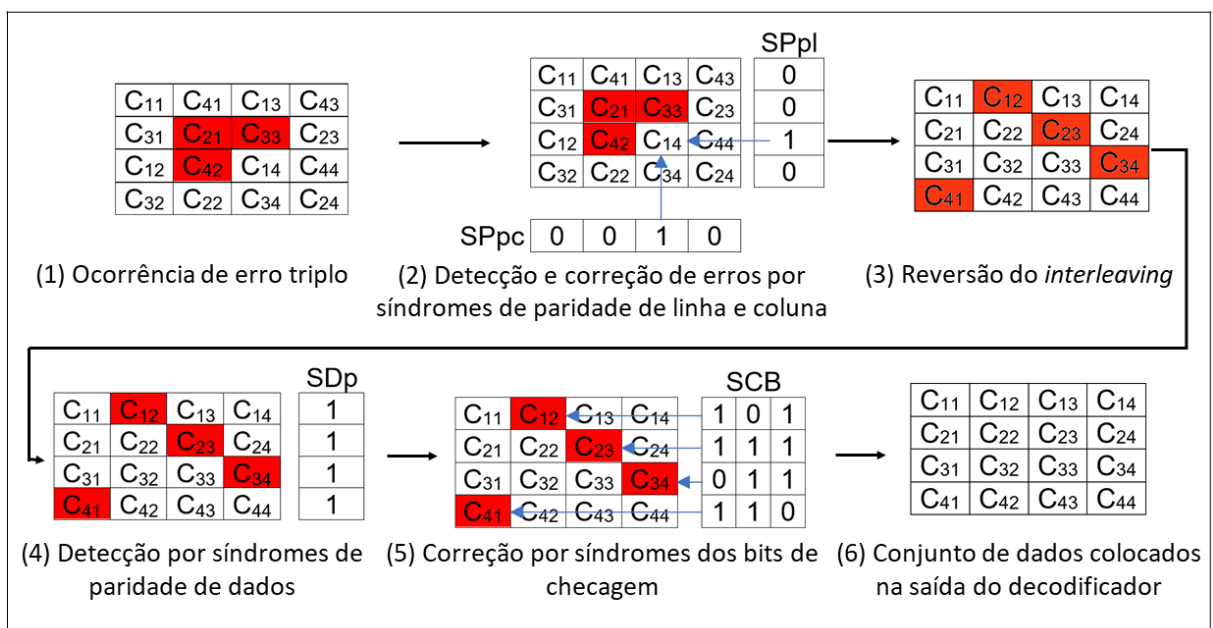
de dados, como mostrado na quarta parte do exemplo. Em seguida, a decodificação de Hamming(7,4) é realizada, apontando para as posições dos erros. Então, correção é feita com a inversão dos valores armazenados e os bits de dados são colocados na saída do decodificador.

Figura 16 – Decodificação de erro duplo PHICC(40,16).



Fonte: Autor.

Figura 17 – Decodificação de erro triplo PHICC(40,16).



Fonte: Autor.

O exemplo ilustrado pela Figura 17, mostra o processo de correção de um padrão de erro triplo sobre os bits de dados. Na segunda parte da figura, podemos perceber que este padrão de erro força as síndromes de paridade de linha e coluna a apontarem para uma posição que não contém erro, levando a uma correção errônea. No entanto, após a reversão do *interleaving* na parte 3, as síndromes de paridade de dados detectam a presença de todos os erros e a correção pela decodificação de Hamming é realizada, como mostrado na parte 5. Ao final do processo de detecção, os bits de dados são colocados na saída do decodificador.

3.2 O código PHICC(44,16)

O código PHICC(44,16) foi criado para proporcionar maior capacidade de correção em comparação com o PHICC(40,16), podendo ser aplicado em situações de maior exposição dos sistemas computacionais à condições adversas, como a exposição a raios cósmicos. Como o código PHICC(44,16) requer uma quantidade maior de memória para o armazenamento das palavras codificadas, sua aplicação é recomendada a sistemas computacionais com baixa restrição de armazenamento. A estratégia de correção utilizada no PHICC(44,16) se assemelha com a utilizada no PHICC(40,16), na qual são utilizados conceitos de bits de paridade, interleaving e código de Hamming. O processo de codificação e decodificação pode ser conferido nas subseções a seguir.

3.2.1 Codificador PHICC(44,16)

O codificador do PHICC(44,16) atua sobre um conjunto de dados de 16 bits formando uma palavra composta de 44 bits organizados conforme ilustrado pela Figura 18, onde podemos visualizar os bits de dados agrupados mais à esquerda, enquanto que os bits de redundância podem ser vistos nos agrupamentos mais à direita.

Figura 18 – Palavra codificada pelo PHICC(44,16).

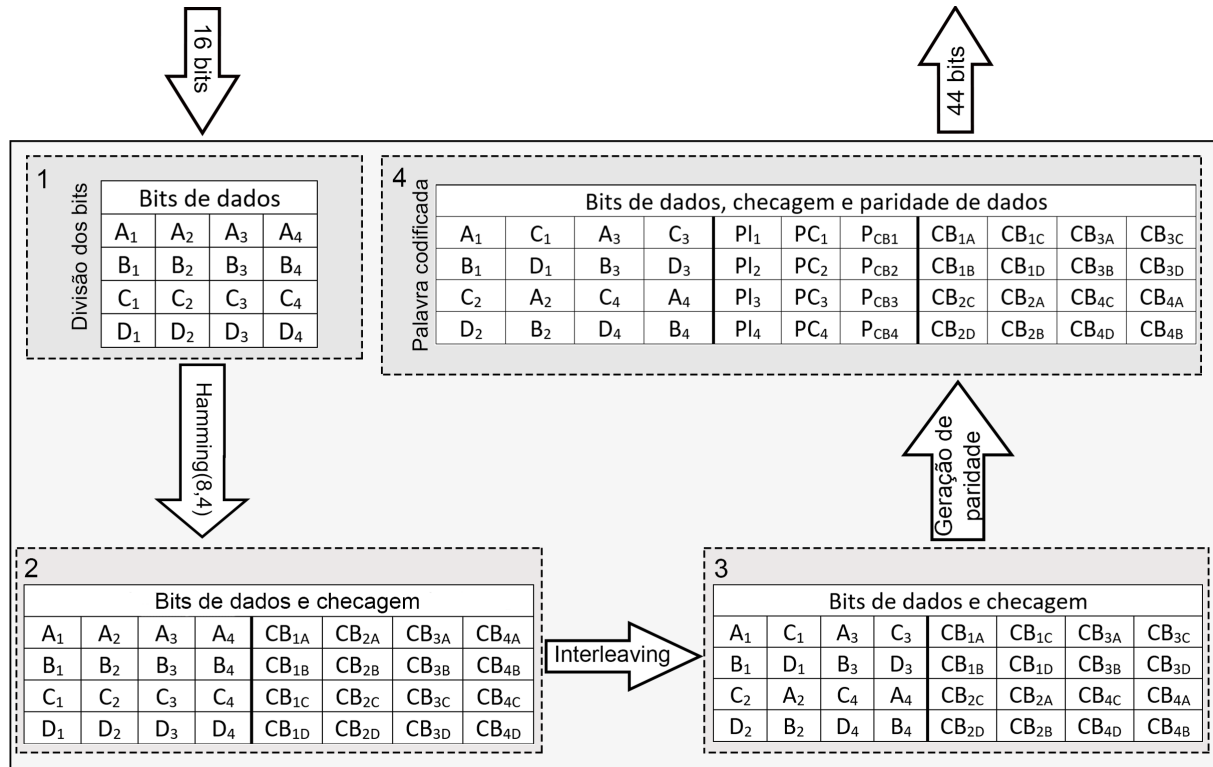
A ₁	C ₁	A ₃	C ₃	Pl ₁	Pc ₁	P _{CB1}	CB _{1A}	CB _{1C}	CB _{3A}	CB _{3C}
B ₁	D ₁	B ₃	D ₃	Pl ₂	Pc ₂	P _{CB2}	CB _{1B}	CB _{1D}	CB _{3B}	CB _{3D}
C ₂	A ₂	C ₄	A ₄	Pl ₃	Pc ₃	P _{CB3}	CB _{2C}	CB _{2A}	CB _{3C}	CB _{4A}
D ₂	B ₂	D ₄	B ₄	Pl ₄	Pc ₄	P _{CB4}	CB _{2D}	CB _{2B}	CB _{4D}	CB _{4B}

Fonte: Autor.

O processo de codificação do PHICC(44,16) é realizado em quatro etapas: a primeira delas é a organização dos bits de dados em forma de matriz 4x4, a segunda é a codificação de Hamming(8,4), a terceira etapa consiste no processo de *interleaving* e a quarta é a

geração dos bits de paridade. Através da Figura 19 podemos conferir, resumidamente, o passo a passo do processo de codificação.

Figura 19 – Processo de codificação PHICC(44,16).



Fonte: Autor.

No primeiro estágio os bits de dados são organizados em forma de matriz 4x4, cujas posições dos bits podem ser visualizadas através da Figura 20.

Figura 20 – Conjunto de dados.

A ₁	A ₂	A ₃	A ₄
B ₁	B ₂	B ₃	B ₄
C ₁	C ₂	C ₃	C ₄
D ₁	D ₂	D ₃	D ₄

Fonte: Autor.

Uma das principais diferenças entre o PHICC(40,16) e o PHICC(44,16) é a adição de um bit de checagem da codificação de Hamming, para cada linha da matriz de dados. Essa diferença aumenta cobertura de correção para padrões de erros mais complexos, pois permite a correção de erros duplos na mesma linha caso seja necessário. As Equações 3.8 a 3.11 descrevem como são obtidos os bits de checagem da primeira linha de dados.

As mesmas operações são realizadas sobre as demais linhas para a realização da segunda etapa do processo de codificação.

$$CB_{1A} = A_1 \oplus A_2 \oplus A_3 \quad (3.8)$$

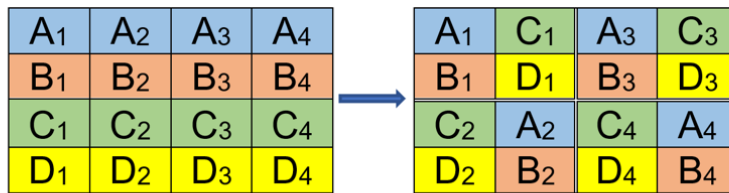
$$CB_{2A} = A_1 \oplus A_2 \oplus A_4 \quad (3.9)$$

$$CB_{3A} = A_1 \oplus A_3 \oplus A_4 \quad (3.10)$$

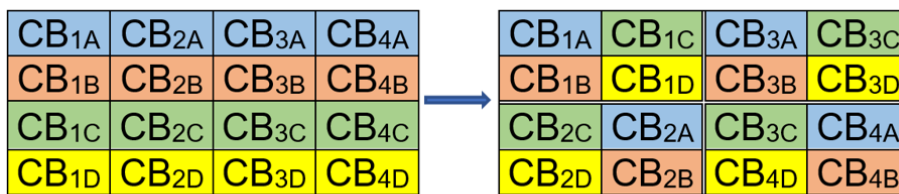
$$CB_{4A} = A_2 \oplus A_3 \oplus A_4 \quad (3.11)$$

O *interleaving* realizado na codificação da palavra ocorre tanto sobre os bits de dados quanto sobre os bits de checagem, obedecendo o padrão demonstrado pela Figura 21.

Figura 21 – Padrão de *interleaving* utilizado no PHICC(44,16).



(a) *Interleaving* sobre os bits de dados

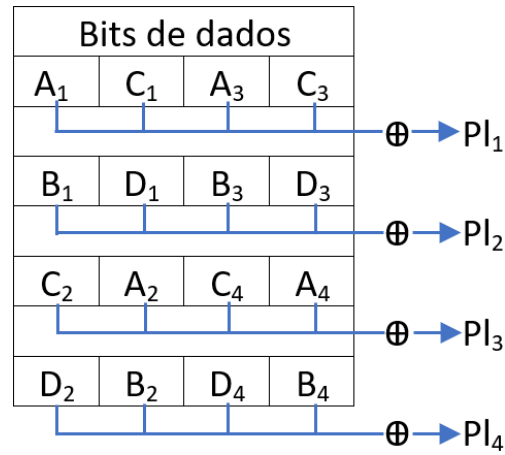


(b) *Interleaving* sobre os bits de checagem

Fonte: Autor.

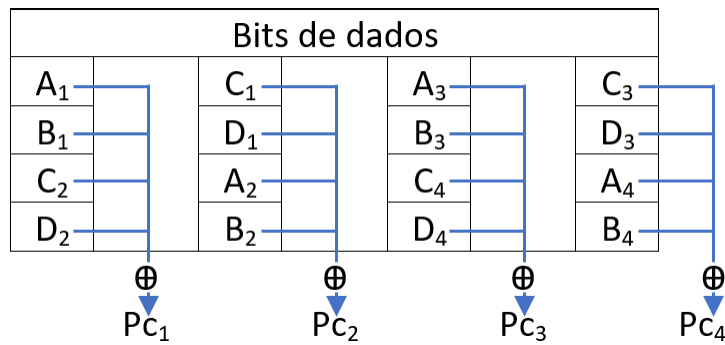
Após o processo de *interleaving*, são calculados os bits de paridade de linha (Pl) e os bits de paridade de coluna (Pc). As operações utilizadas para a obtenção desses bits de paridade são demonstradas pelas Figuras 22 e 23.

Figura 22 – Bits de paridade de linhas.



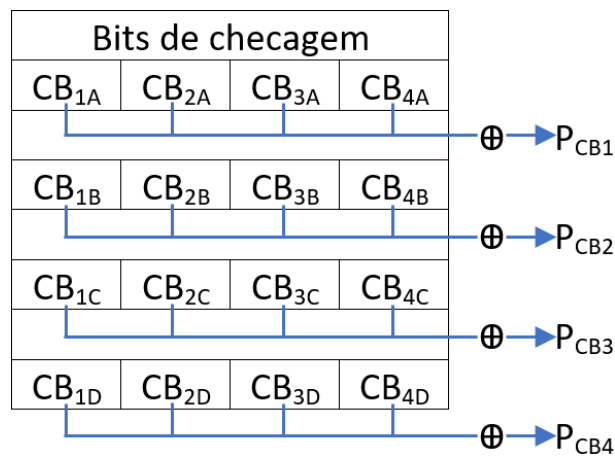
Fonte: Autor.

Figura 23 – Bits de paridade de coluna.



Fonte: Autor.

Figura 24 – Bits de paridade de bits de checagem.



Fonte: Autor.

Um outro aspecto que diferencia o PHICC(44,16) do PHICC(40,16) é a utilização de bits de paridade para a verificação da integridade dos bits de checagem (P_{CB}). Esses bits de paridade são obtidos conforme mostrado na Figura 24 e são utilizados para evitar a realização de uma correção errônea devido a presença de erros nos bits de checagem. O processo de *interleaving* realizado sobre os bits de checagem, torna a detecção de erros mais eficaz, pois permite a separação de erros pares adjacentes, que normalmente não seriam detectados. Dessa forma, os erros múltiplos podem ser reconhecidos como erros simples em cada linha dos bits de checagem.

A composição da palavra codificada é concluída com a inclusão dos bits de paridade. Ao final de todo o processo a palavra de 44 bits é colocada na saída do codificador.

3.2.2 Decodificador PHICC(44,16)

O decodificador do PHICC(44,16) atua através da correção por bits de paridade e da correção por codificação de Hamming(8,4). Os passos realizados durante o processo e correção de erros podem ser visualizados através da Figura 25.

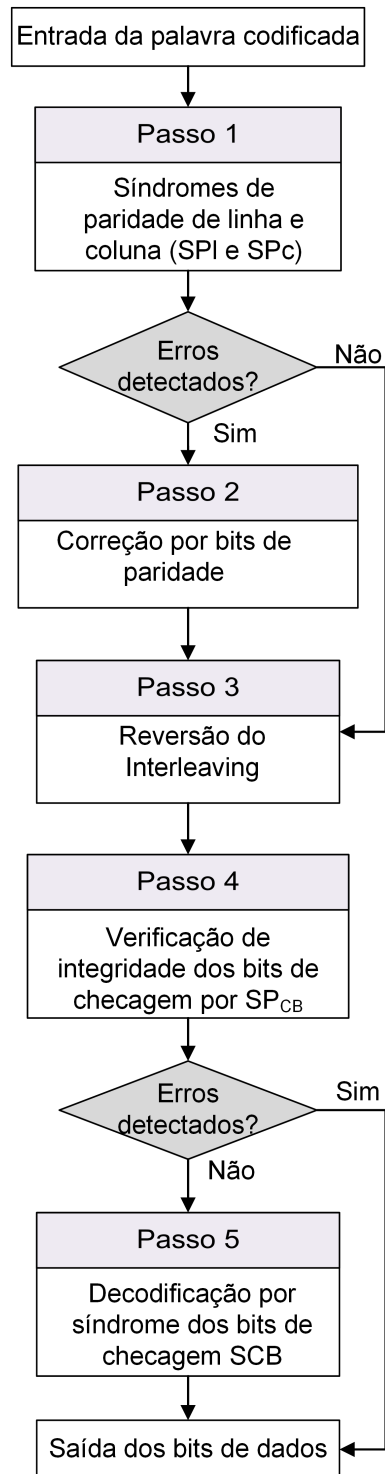
O passo 1 consiste de um processo no qual os bits de paridade de linha e coluna são recalculados a partir dos dados recebidos (Pl' e Pc') e em seguida são realizadas operações lógicas XOR com os valores de Pl e Pc já armazenados. As Equações 3.12 e 3.13 descrevem como essa operação é feita.

Essas síndromes são utilizadas para a detecção de erros ímpares, nas linhas e nas colunas do conjunto de dados. O cruzamento dessas informações permite determinar as posições afetadas e a correção é realizada com a inversão do valor armazenado.

$$SPl = Pl' \oplus Pl \quad (3.12)$$

$$SPc = Pc' \oplus Pc \quad (3.13)$$

Figura 25 – Processo de decodificação.



Fonte: Autor.

Com a reversão do *interleaving*, os erros remanescentes da correção por paridade são separados em diferentes linhas. Então as síndromes de paridade dos bits de checagem (SP_{CB}) são calculadas conforme a Equação 3.14, onde P_{CB}' é o resultado da paridade dos bits de checagem recalculada a partir da palavra código recebida e P_{CB} é o valor da

paridade armazenada. Caso nenhum erro seja detectado, é executada a decodificação de Hamming(8,4) a partir das síndromes dos bits de checagem (SCB), conforme a Equação 3.15, onde CB' corresponde ao valor recalculado enquanto CB é o valor armazenado. Após a correção, os dados são colocados na saída do decodificador. Quando erros são detectados nos bits de checagem, o processo de correção é interrompido e os bits de dados são colocados na saída do decodificador. A Tabela 6 mostra detalhes de como é realizada a decodificação pelas síndromes dos bits de checagem, indicando as posições dos erros com base nos resultados das síndromes.

$$SP_{CB} = P'_{CB} \oplus P_{CB} \quad (3.14)$$

$$SCB = CB' \oplus CB \quad (3.15)$$

Tabela 6 – Decodificação por bits de checagem PHICC(44,16).

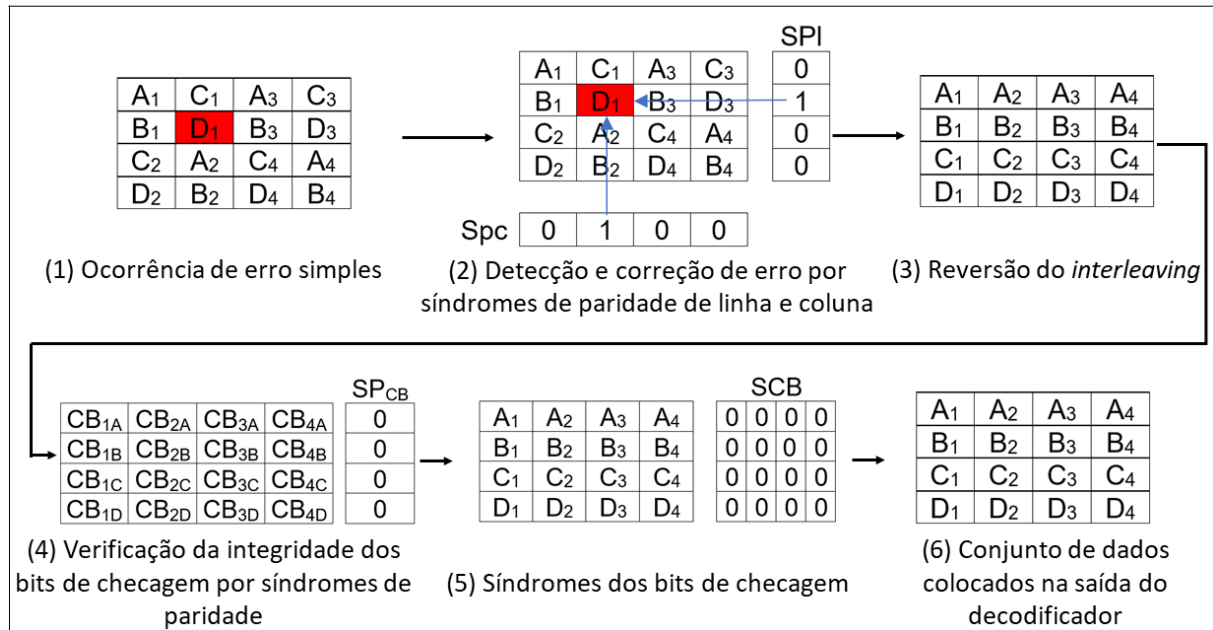
SCB ₁	SCB ₂	SCB ₃	SCB ₄	Bits afetados
0	0	0	0	Sem erro
1	1	1	0	1
1	1	0	1	2
1	0	1	1	3
0	1	1	1	4
0	0	1	1	1, 2
0	1	0	1	1, 3
0	1	1	0	1, 4
1	0	0	1	2, 3
1	0	1	0	2, 4
1	1	0	0	3, 4

Fonte: Autor.

3.2.3 Exemplos de correções feitas através do PHICC(44,16)

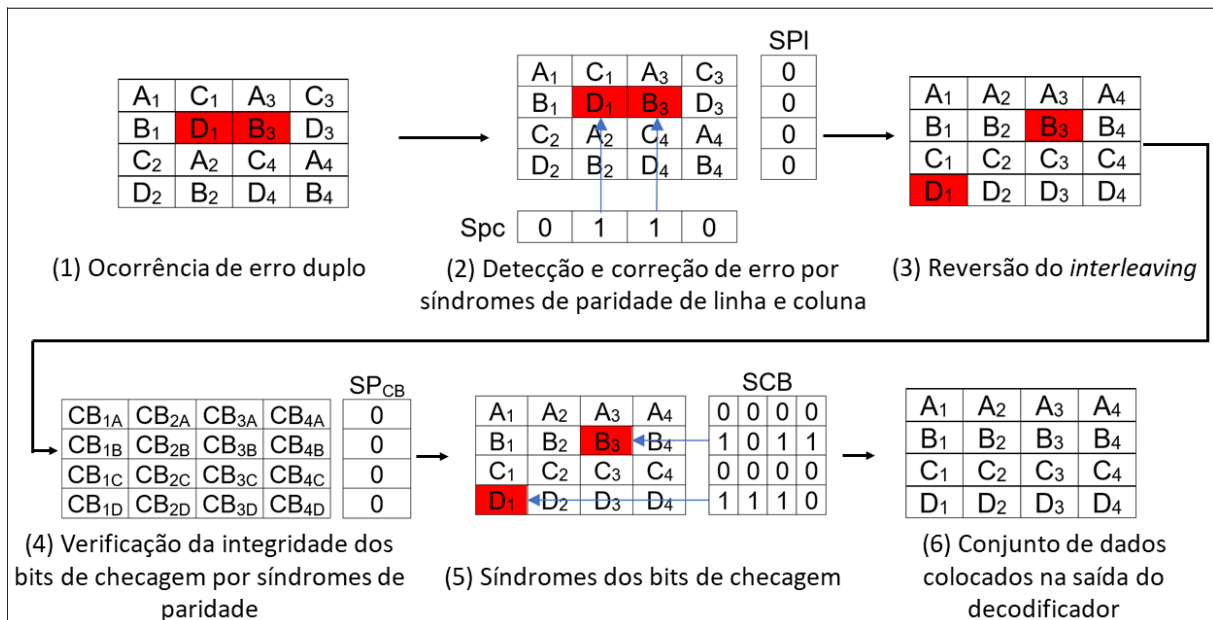
Nesta subseção, três exemplos do processo de correção de erros através do PHICC(44,16) serão demonstrados. O primeiro exemplo, mostrado na Figura 26, consiste de um erro simples sobre os bits de dados. Podemos notar que para este padrão de erro as síndromes dos bits de paridade de linha e coluna são capazes de apontar a posição do erro, permitindo a correção. Após a primeira etapa de correção, nenhum erro remanescente é deixado para o segundo processo de correção e os bits de dados são colocados na saída do decodificador.

Figura 26 – Decodificação de erro simples PHICC(44,16).



Fonte: Autor.

Figura 27 – Decodificação de erro duplo PHICC(44,16).



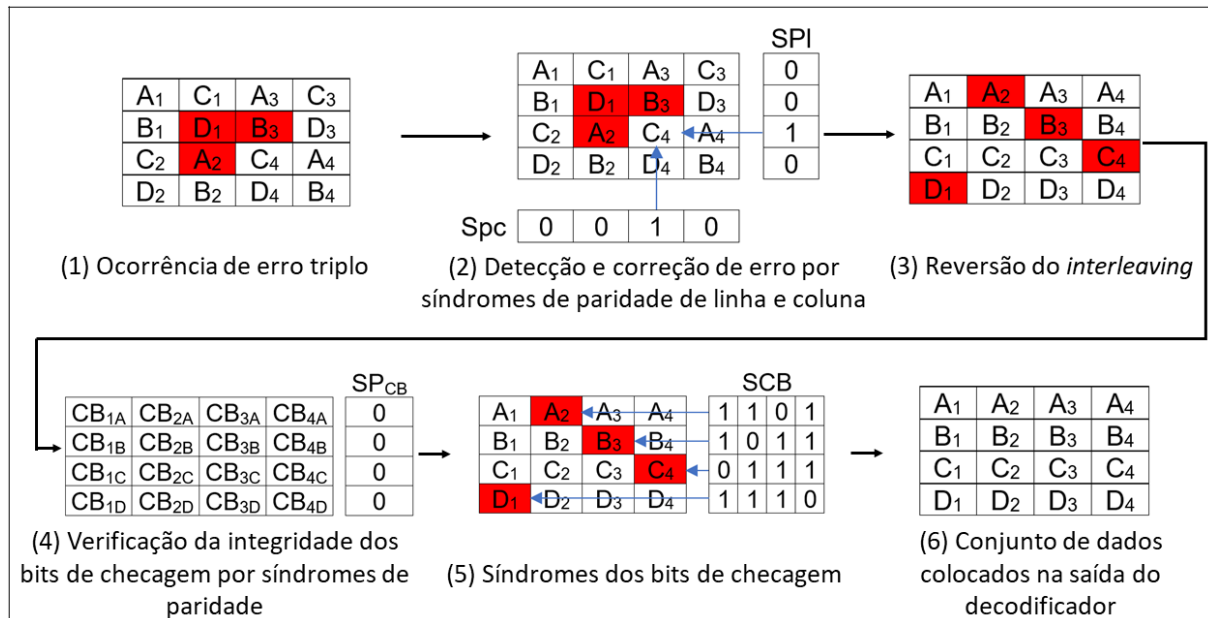
Fonte: Autor.

Na Figura 27 podemos visualizar o processo de correção de um padrão de erro duplo. É possível observarmos que, como os bits afetados pertencem a mesma linha, as síndromes de paridade de linha não são capazes de detectá-los. Após a etapa de correção por bits de paridade, a reversão do *interleaving* separa os erros não corrigidos em diferentes linhas. Em seguida a integridade dos bits de checagem é verificada através das síndromes

de paridade dos bits de checagem (SP_{CB}). Como não são detectados erros sobre os bits de checagem, a decodificação de Hamming(8,4) é realizada, apontando para as posições dos erros. Então, correção é feita com a inversão dos valores armazenados e os bits de dados são colocados na saída do decodificador.

A Figura 28 ilustra as etapas de correção de um padrão de erro triplo. Ao observarmos a figura, podemos perceber que esse padrão causa uma correção errônea devido a posição apontada pelas síndromes dos bits de paridade de linha e coluna. No entanto, nos passos restantes, os erros são separados em diferentes linhas, permitindo a correção total através das síndromes dos bits de checagem.

Figura 28 – Decodificação de erro triplo PHICC(44,16).



Fonte: Autor.

3.3 Código PHICC(36,16)

O PHICC(36,16) foi desenvolvido com o intuito de reduzir o uso de memória em relação ao PHICC(40,16), mas sem prejudicar de maneira expressiva a sua capacidade de correção de erros. Apesar das principais características do PHICC(40,16) terem sido mantidas, realizamos mudanças nas operações para a obtenção dos bits de redundância.

3.3.1 Codificador PHICC(36,16)

O codificador do PHICC(36,16) atua sobre um conjunto de dados de 16 bits, formando uma palavra composta de 36 bits organizados conforme ilustrado pela Figura 29, na qual vemos A , B , C e D , que são os bits de dados, Pl e Pc , que são os bits de

paridade de linha e coluna, respectivamente, CB que são bits de checagem e Pd que são os bits de paridade de dados não permutados.

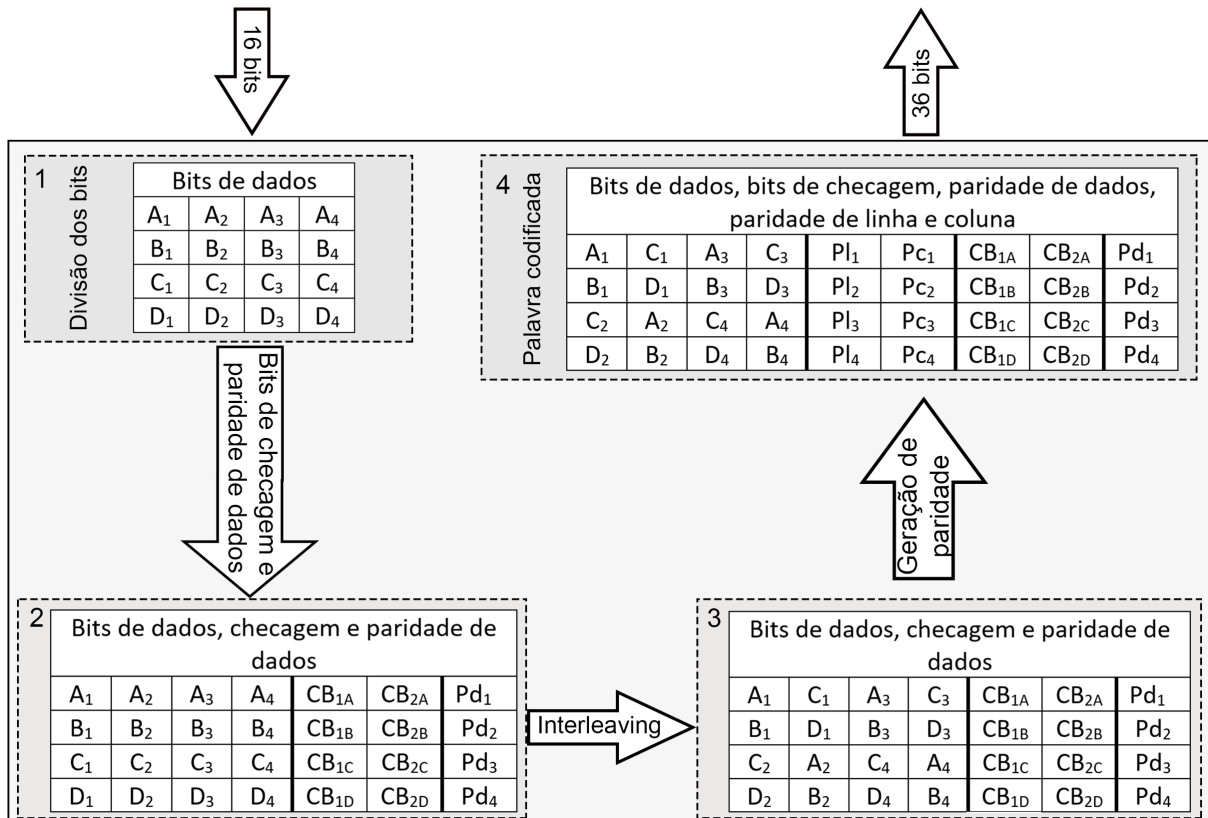
Figura 29 – Palavra codificada pelo PHICC(36,16).

A_1	C_1	A_3	C_3	Pl_1	Pc_1	CB_{1A}	CB_{2A}	Pd_1
B_1	D_1	B_3	D_3	Pl_2	Pc_2	CB_{1B}	CB_{2B}	Pd_2
C_2	A_2	C_4	A_4	Pl_3	Pc_3	CB_{1C}	CB_{2C}	Pd_3
D_2	B_2	D_4	B_4	Pl_4	Pc_4	CB_{1D}	CB_{2D}	Pd_4

Fonte: Autor.

A Figura 30 ilustra resumidamente o passo a passo do processo de codificação, no qual os 16 bits de dados são inseridos e divididos em quatro subgrupos de quatro bits, de forma semelhante a que vimos na Figura 20, na codificação do PHICC(44,16). Em seguida, para cada um desses subgrupos, os bits paridade e de checagem são calculados, totalizando 7 bits por grupo. Logo depois as posições dos bits são permutadas, ao passarem pelo processo de *interleaving*. Finalmente, os bits de paridade de linha e coluna são calculados, compondo a palavra codificada, que logo após é colocada na saída do codificador.

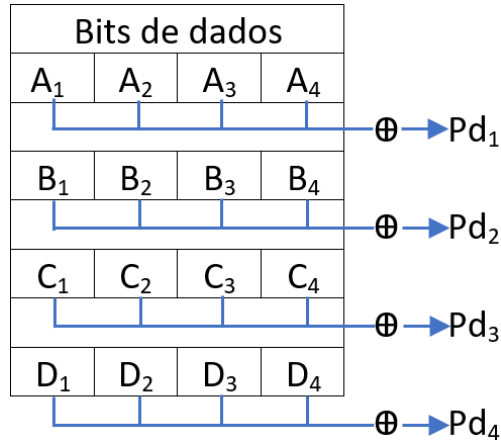
Figura 30 – Processo de codificação PHICC(36,16).



Fonte: Autor.

Os bits de paridade de dados Pd , que são os resultados das paridades de cada linha da matriz de dados, são obtidos como demonstrado na Figura 31.

Figura 31 – Bits de paridade de dados.



Fonte: Autor.

A codificação por bits de checagem é inspirada no código de Hamming, no entanto, utilizamos outras operações para que pudéssemos reduzir o número de bits de redundância. As operações que regem a obtenção dos bits de checagem podem ser conferidas através das equações 3.16 a 3.23.

$$CB_{1A} = A_1 \oplus A_2 \quad (3.16)$$

$$CB_{2A} = A_1 \oplus A_3 \quad (3.17)$$

$$CB_{1B} = B_1 \oplus B_2 \quad (3.18)$$

$$CB_{2B} = B_1 \oplus B_3 \quad (3.19)$$

$$CB_{1C} = C_1 \oplus C_2 \quad (3.20)$$

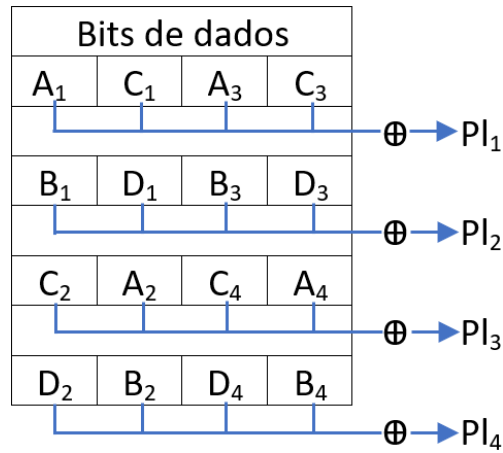
$$CB_{2C} = C_1 \oplus C_3 \quad (3.21)$$

$$CB_{1D} = D_1 \oplus D_2 \quad (3.22)$$

$$CB_{2D} = D_1 \oplus D_3 \quad (3.23)$$

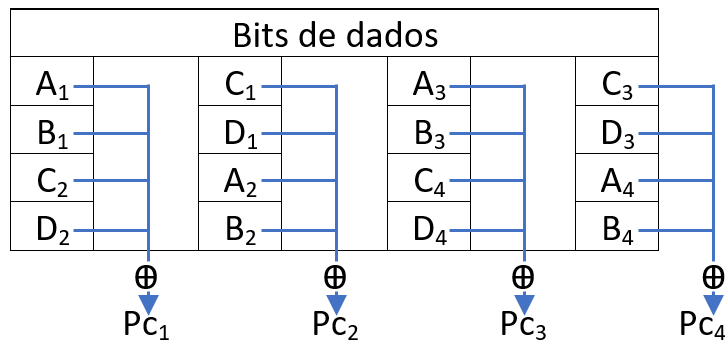
O padrão de *interleaving* utilizado na codificação é semelhante ao visto no PHICC(40,16), no qual os bits de cada coluna são colocados em um dos quadrantes da matriz de dados. Na última etapa da codificação, os bits de paridade de linha e coluna são calculados de acordo com o demonstrado nas Figuras 32-33.

Figura 32 – Bits de paridade de linhas.



Fonte: Autor.

Figura 33 – Bits de paridade de coluna.



Fonte: Autor.

3.3.2 Decodificador PHICC(36,16)

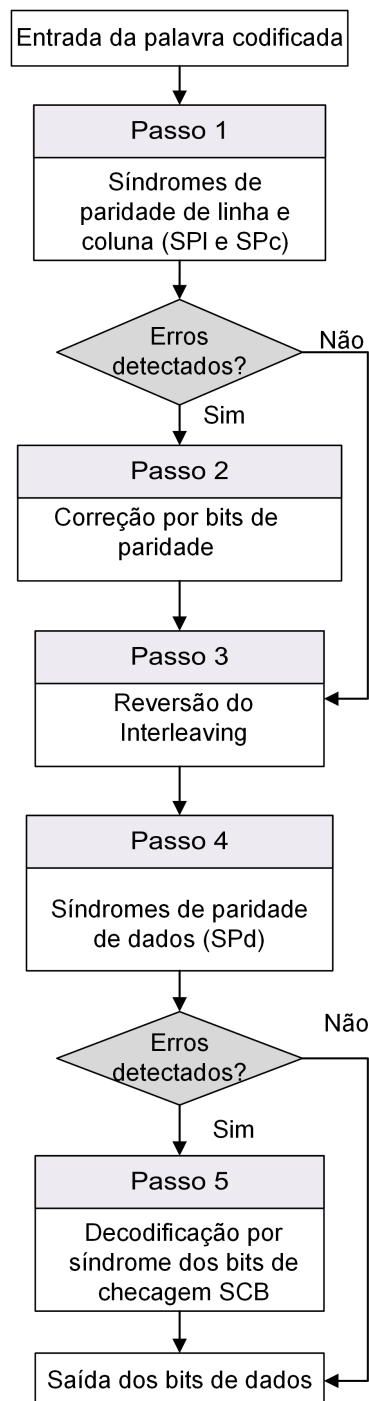
O decodificador do PHICC(36,16) atua em cinco passos, conforme podemos ver na Figura 34. O primeiro passo é o cálculo das síndromes de paridade de linha e coluna que é realizado através das Equações 3.24 e 3.25.

$$SPl = Pl' \oplus Pl \quad (3.24)$$

$$SPc = Pc' \oplus Pc \quad (3.25)$$

No passo 2, os erros detectados por bits de paridade são corrigidos a partir do cruzamento das informações de linhas e colunas erradas. Com a reversão do *interleaving* os erros não corrigidos são separados, permitindo que as síndromes de paridade de dados os detectem no passo 4. Após a detecção, os erros são corrigidos pelos bits de checagem e os dados são colocados na saída do decodificador.

Figura 34 – Processo de decodificação do PHICC(36,16).



Fonte: Autor.

A decodificação por bits de checagem, atua em conjunto com as síndromes de paridade de dados. Como os bits de checagem não possuem todas as informações necessárias para a realização das correções, elas são complementadas pelos bits de paridade de dados. A Tabela 3 demonstra como é realizada a decodificação por bits de checagem.

Tabela 7 – Decodificação por bits de checagem PHICC(40,16).

SCB ₁	SCB ₂	SPd	Posição do erro	Bits afetados
0	0	0	Sem erro	-
0	0	1	Conjunto de dados	4
0	1	1	Conjunto de dados	3
1	0	1	Conjunto de dados	2
1	1	1	Conjunto de dados	1
1	0	0	Bits de redundância	-
0	1	0	Bits de redundância	-
1	1	0	Bits de redundância	-

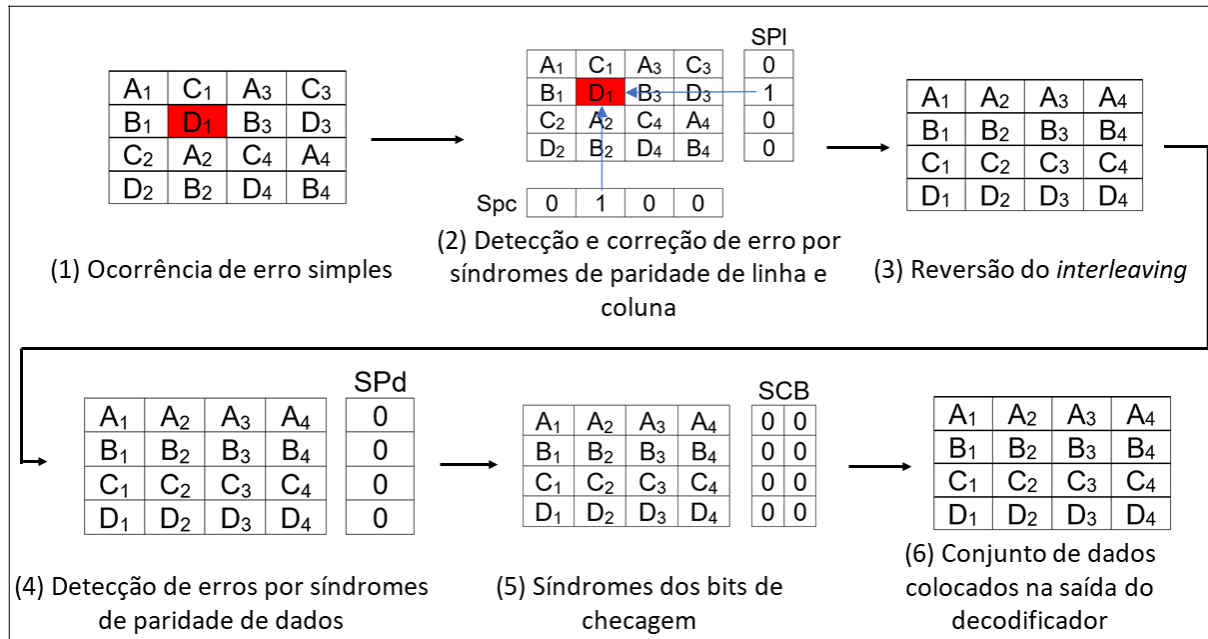
Fonte: Autor.

3.3.3 Exemplos de correções feitas através do PHICC(36,16)

O primeiro exemplo, ilustrado pela Figura 35, se trata de um erro simples sobre os bits de dados. Este padrão de erro é facilmente detectado e corrigido pelas síndromes de paridade de linha e coluna, como pode ser visto na parte (2) da figura. Após a reversão do *interleaving*, nenhum outro erro é detectado e os bits de dados são colocados na saída do decodificador.

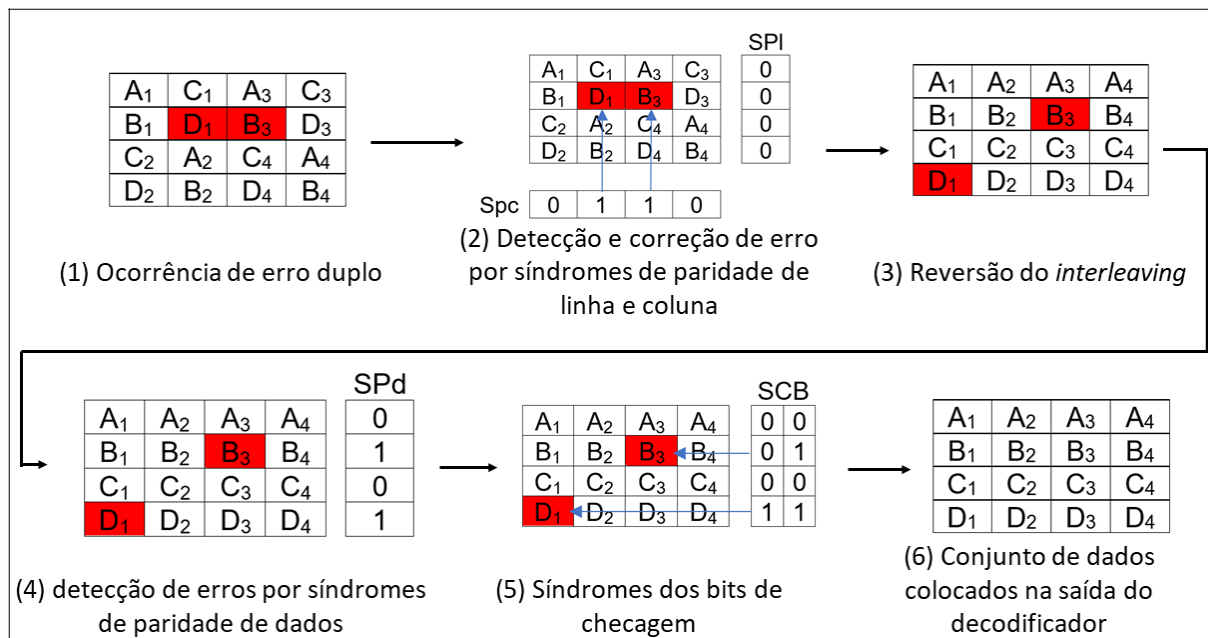
O segundo exemplo mostra a ocorrência de um erro duplo. Na parte (2) da Figura 36, podemos perceber que este padrão de erro dificilmente pode ser corrigido pelas síndromes de paridade de linha e coluna, o que leva a necessidade da utilização das outras estratégias do código. A reversão do *interleaving*, em (3), separa os erros adjacentes em diferentes linhas, permitindo a detecção por síndromes de paridade de dados, como mostrado na parte (4). Então, a decodificação por bits de checagem é realizada na parte (5), corrigindo os erros com a inversão dos valores armazenados. Em seguida, os dados são colocados na saída do decodificador.

Figura 35 – Decodificação de erro simples PHICC(36,16).



Fonte: Autor.

Figura 36 – Decodificação de erro duplo PHICC(36,16).

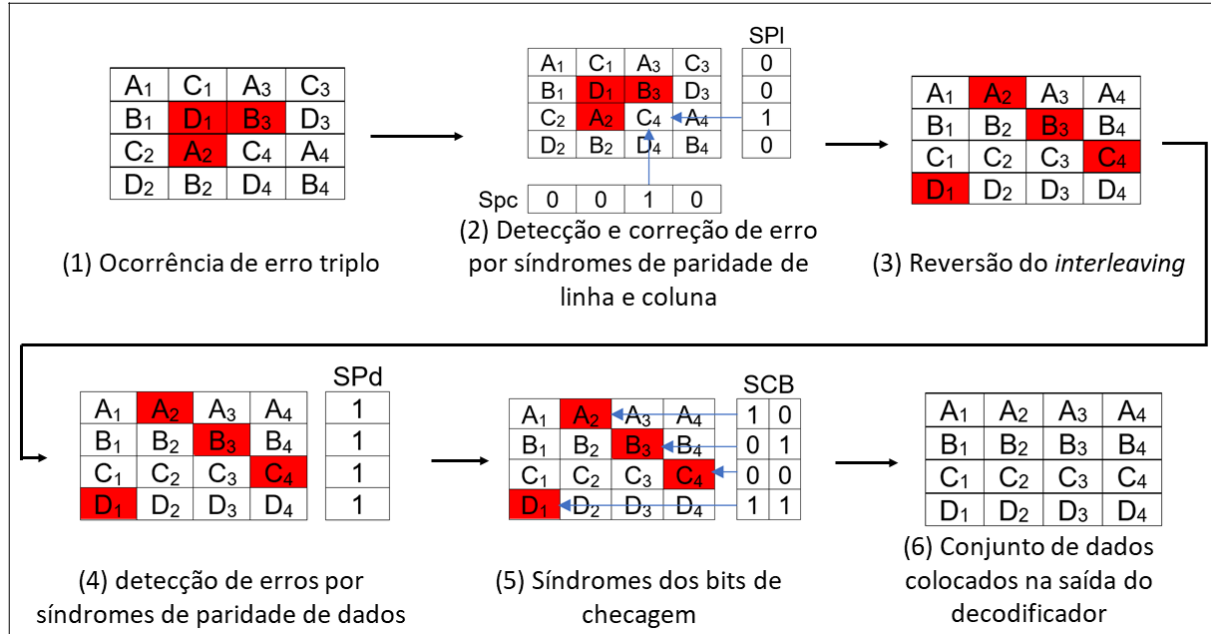


Fonte: Autor.

O terceiro exemplo, demonstrado pela Figura 37, se trata de um erro triplo. Apesar de alguns padrões de erros triplos serem corrigidos pelas síndromes de paridade de linha e coluna, o padrão em demonstração não permite essa correção, fazendo com que as síndromes de paridade de linha e coluna apontem para uma posição que não contém

erro, como visto na parte (2). Apesar disso, a separação dos erros, com a reversão do *interleaving*, permite a detecção correta em (4) e a correção por bits de checagem em (5).

Figura 37 – Decodificação de erro triplo PHICC(36,16).



Fonte: Autor.

3.4 O código PHICC(32,16)

O código PHICC(32,16) foi criado com o propósito de permitir a correção de MBUs com alto padrão de complexidade, mesmo com uma grande redução na quantidade de bits de redundância, em comparação com o PHICC(40,16). A redução do uso de memória pode permitir a utilização do PHICC(32,16) em sistemas embarcados, cujos recursos são mais limitados.

Buscamos utilizar as mesmas etapas de correção do PHICC(40,16), a fim de mantermos ao máximo as características de capacidade de correção. No entanto, adotamos duas novas estratégias: a codificação por bits de checagem usada no PHICC(36,16) e o compartilhamento de paridade, inspirado no conceito de Reviriego et al. (2013). No PHICC(32,16), os valores das paridades de linha e coluna são obtidos de forma indireta, a partir de combinações dos bits de checagem que envolvam os bits de interesse. Dessa forma, as síndromes de paridade de linha e coluna podem ser calculadas, sem a necessidade de utilizar bits dedicados para essa finalidade. Este CCE pode ser utilizado em sistemas com restrições na capacidade de armazenamento por possuir uma palavra código mais compacta.

3.4.1 Codificador PHICC(32,16)

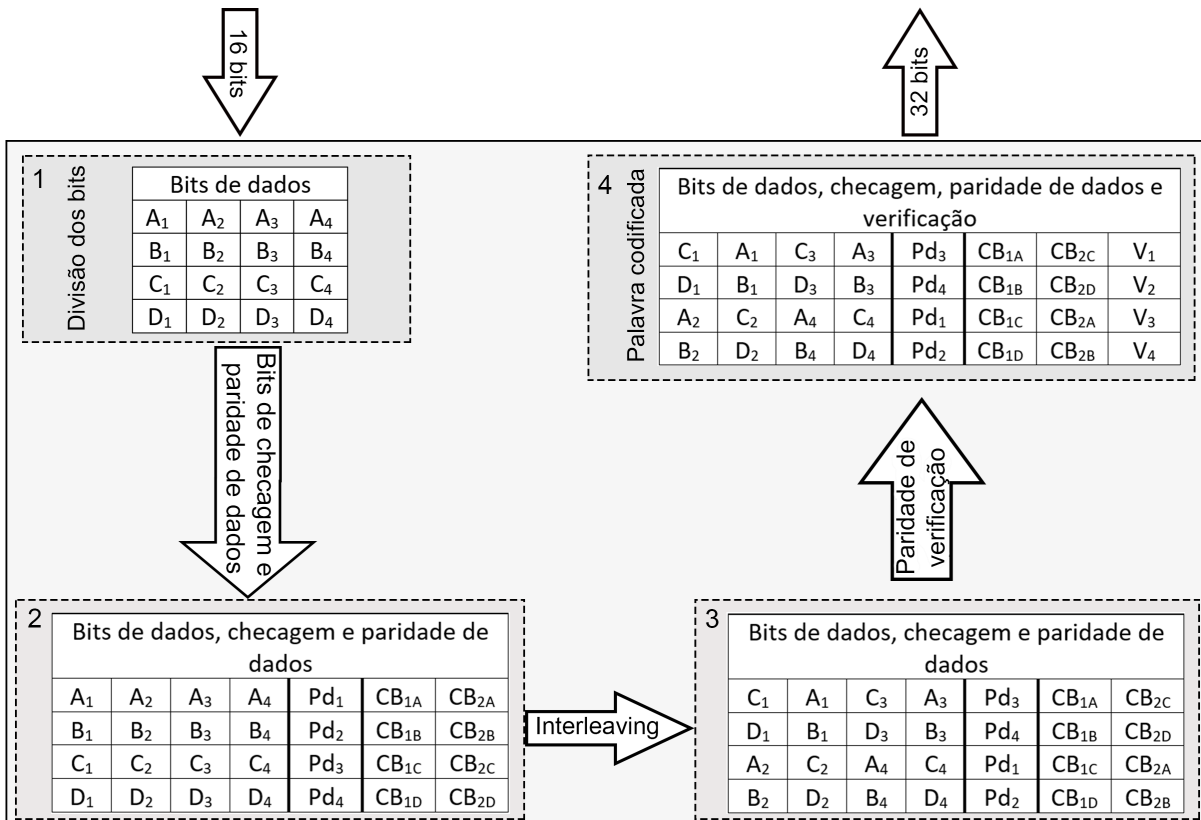
Assim como nas demais implementações, o PHICC(32,16) utiliza conceitos de Paridade, Hamming e *interleaving* para codificar um conjunto de dados de 16 bits, no entanto algumas estratégias foram modificadas. A Figura 38 mostra o formato da palavra codificada pelo PHICC(32,16) e a Figura 39 mostra quais são as etapas envolvidas no processo de codificação.

Figura 38 – Palavra codificada pelo PHICC(32,16).

C ₁	A ₁	C ₃	A ₃	Pd ₃	CB _{1A}	CB _{2C}	V ₁
D ₁	B ₁	D ₃	B ₃	Pd ₄	CB _{1B}	CB _{2D}	V ₂
A ₂	C ₂	A ₄	C ₄	Pd ₁	CB _{1C}	CB _{2A}	V ₃
B ₂	D ₂	B ₄	D ₄	Pd ₂	CB _{1D}	CB _{2B}	V ₄

Fonte: Autor.

Figura 39 – Processo de codificação PHICC(32,16).



Fonte: Autor.

Na primeira etapa, os bits de dados são separados em quatro subgrupos de quatro bits, assim como nas demais implementações do PHICC. Em seguida, através das Equações

3.26 a 3.33, os bits de checagem são obtidos. Neste momento, os bits de paridade de dados também são obtidos, assim como demonstrado pela Figura 40.

$$CB_{1A} = A_1 \oplus A_2 \quad (3.26)$$

$$CB_{2A} = A_1 \oplus A_3 \quad (3.27)$$

$$CB_{1B} = B_1 \oplus B_2 \quad (3.28)$$

$$CB_{2B} = B_1 \oplus B_3 \quad (3.29)$$

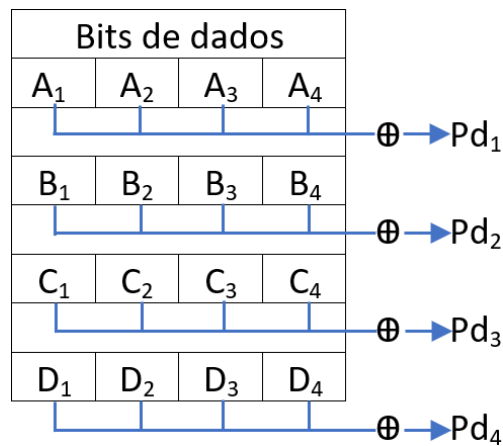
$$CB_{1C} = C_1 \oplus C_2 \quad (3.30)$$

$$CB_{2C} = C_1 \oplus C_3 \quad (3.31)$$

$$CB_{1D} = D_1 \oplus D_2 \quad (3.32)$$

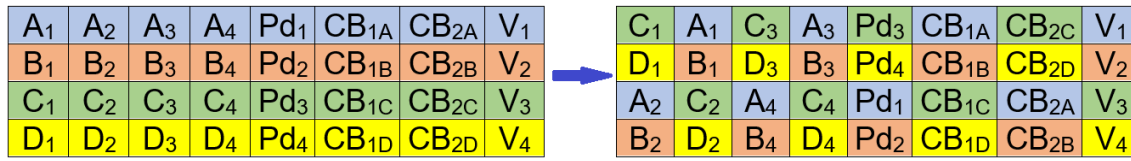
$$CB_{2D} = D_1 \oplus D_3 \quad (3.33)$$

Figura 40 – Bits de paridade de dados.



Fonte: Autor.

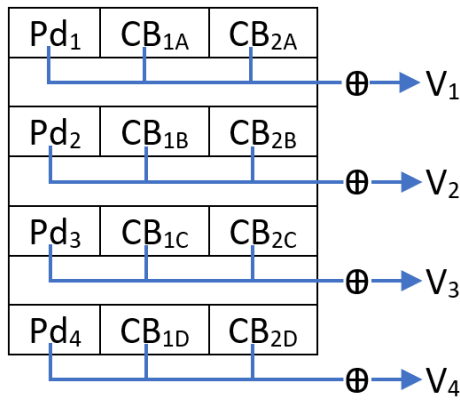
Na etapa 3, ocorre o *interleaving*, no qual tanto os bits de dados quanto os bits de redundância são submetidos. O padrão de permutação dos bits pode ser observado na Figura 41.

Figura 41 – Padrão de *interleaving* utilizado no PHICC(32,16)

Fonte: Autor.

Na etapa 4, os bits de paridade de verificação, que permitem a detecção de erros nos bits de redundância, são calculados. Na Figura 42, podemos observar as operações utilizadas neste processo. Após todo o processo, a palavra codificada é colocada na saída do codificador.

Figura 42 – Bits de paridade de verificação.



Fonte: Autor.

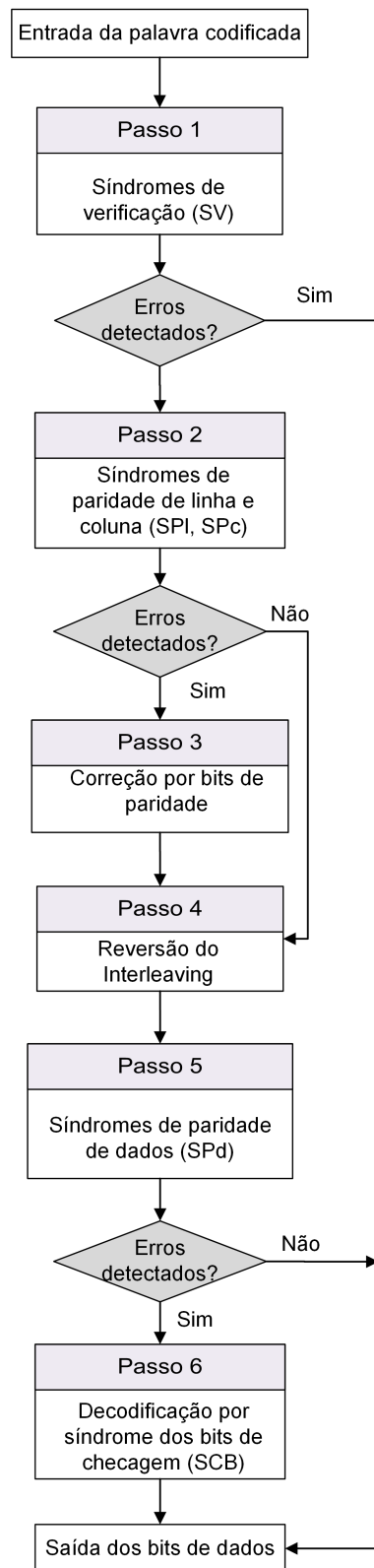
3.4.2 Decodificador PHICC(32,16)

A decodificação do PHICC(32,16), como podemos ver na Figura 43, possui 6 passos. O primeiro é a obtenção das síndromes dos bits de verificação (SV), que são calculadas através da Equação 3.34, na qual V' é o valor da paridade recalculada e V é o valor da paridade armazenada.

$$SV = V' \oplus V \quad (3.34)$$

As síndromes dos bits de verificação são utilizadas para detectar a ocorrência de erros sobre os bits de redundância. Esse tipo de erro é um indício de que os bits de dados não foram afetados, portanto, os bits de dados são colocados diretamente na saída do decodificador.

Figura 43 – Processo de decodificação do PHICC(32,16).

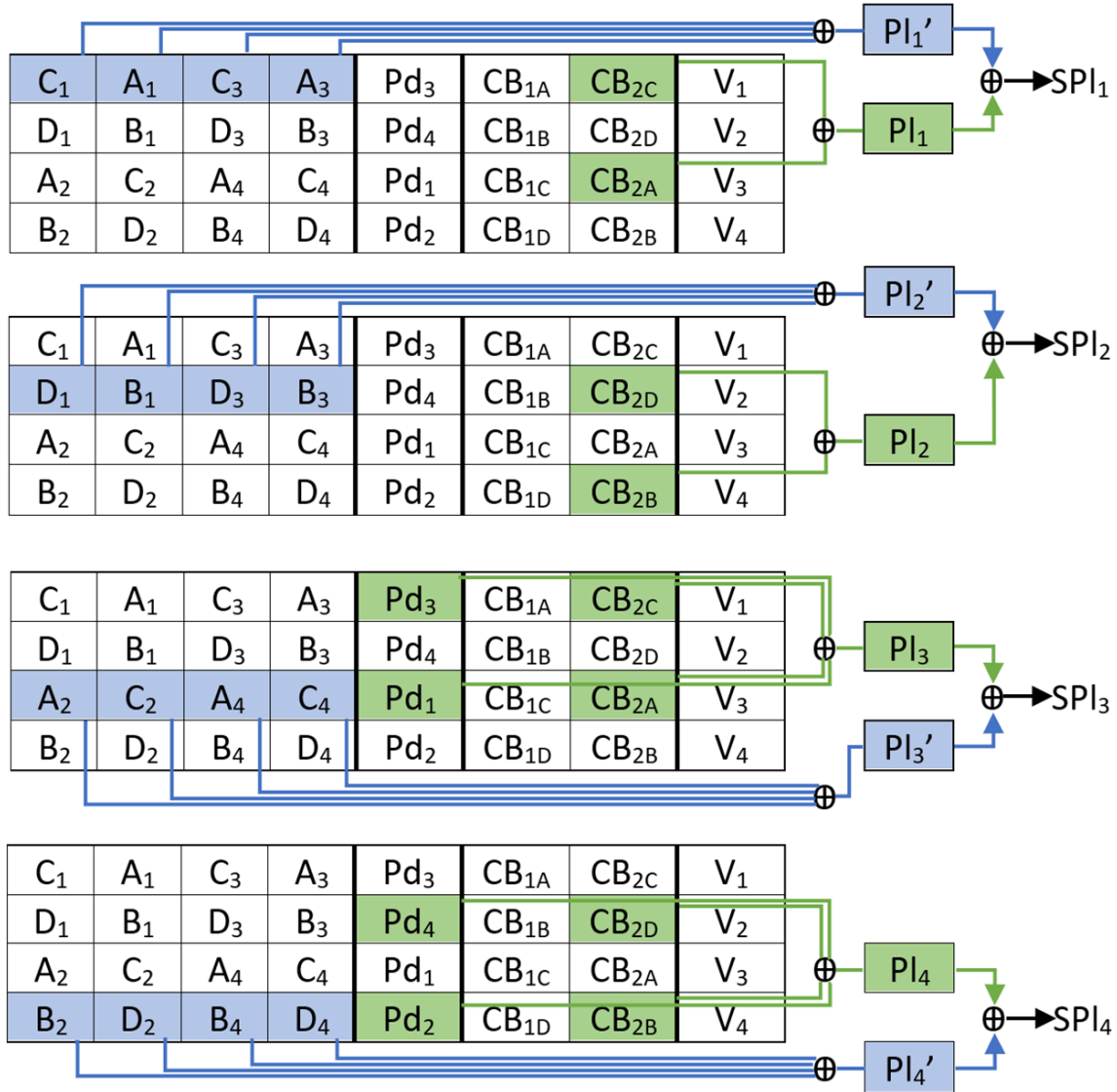


Fonte: Autor.

Nos casos em que nenhum erro é detectado nos bits de redundância, o passo 2, que consiste no cálculo das síndromes de paridade de linha e coluna, é realizado. Apesar de

não haver bits dedicados para o armazenamento dessas paridades, é possível obtermos essas síndromes através das informações contidas nos bits de paridade de dados e nos bits de checagem. As Figuras 44 e 45 demonstram como ocorre o processo.

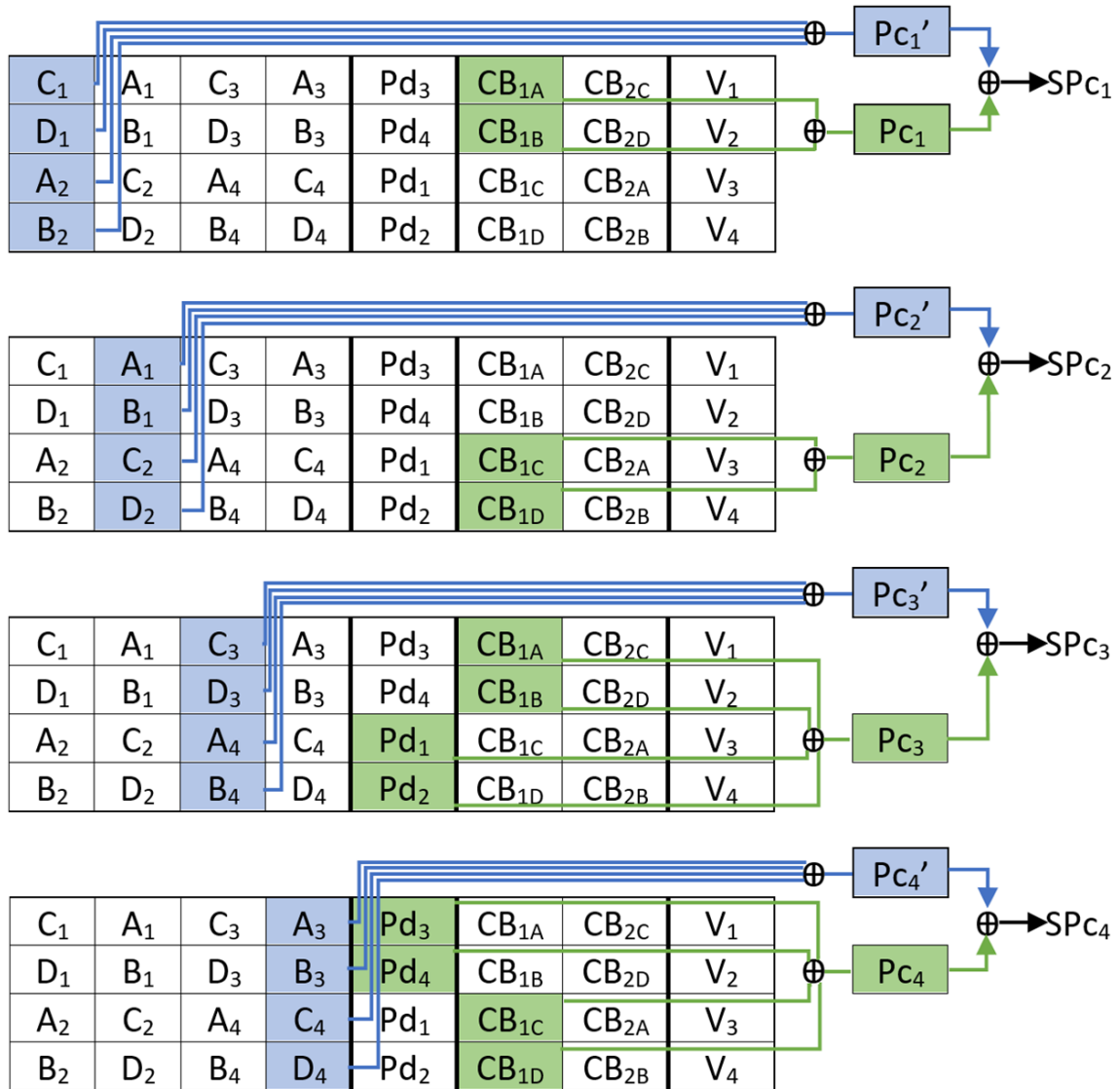
Figura 44 – Obtenção das síndromes de paridade de linha.



Fonte: Autor.

Na Figura 44, podemos ver, em verde, como são obtidas as paridades de linha a partir dos demais bits de redundância. É possível notar que, para as linhas 3 e 4, os bits de paridade de dados são utilizados para complementar as informações que não estão presentes nos bits de checagem. Em azul, podemos visualizar como são obtidas as paridades de linha recalculadas a partir dos dados obtidos, então as síndromes de paridade de linha são calculadas através de operações XOR entre a paridade de linha recebida e a paridade de linha recalculada.

Figura 45 – Obtenção das síndromes de paridade de coluna.



Fonte: Autor.

Na Figura 45, visualizamos em verde, como são obtidas as paridades de coluna a partir dos demais bits de redundância. Para as colunas 3 e 4, os bits de paridade de dados são utilizados para complementar as informações que não estão presentes nos bits de checagem. Em azul, podemos visualizar como são obtidas as paridades de coluna recalculadas a partir dos dados recebidos. As síndromes de paridade de coluna são calculadas através de operações XOR entre a paridade de coluna recebida e a paridade de coluna recalculada.

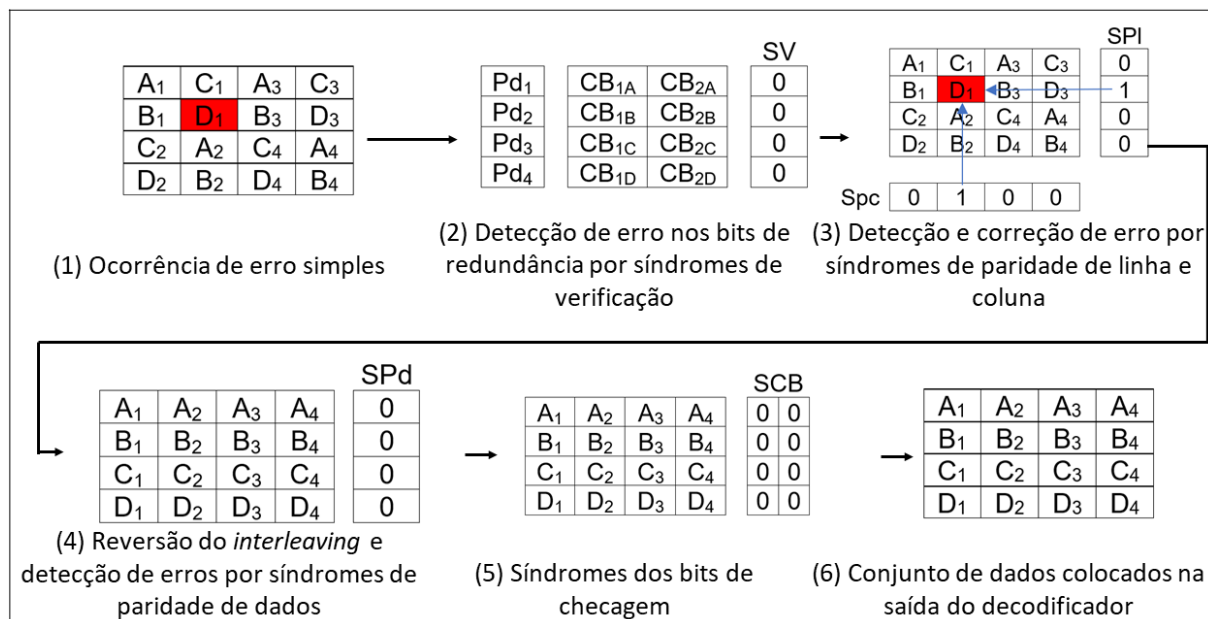
Após a correção realizada no passo 3, os bits de dados retornam para as posições originais no passo 4 e as síndromes SPd são executadas no passo 5, a fim de sabermos quais as linhas que ainda contêm erros. Caso algum erro seja detectado, a síndrome SV correspondente a linha é verificada novamente para assegurar que os bits de checagem responsáveis pela correção não estão comprometidos, causando uma correção errada. O

passo 6 ocorre após a detecção de erros por síndromes de paridade de dados, com a decodificação por bits de checagem. Então os bits de dados são colocados na saída do decodificador.

3.4.3 Exemplos de correção do PHICC(32,16)

O processo de decodificação do PHICC(32,16) possui operações mais complexas que as demais versões. Por esse motivo, usaremos os exemplos a seguir para mostrar os passos necessários para a decodificação de uma palavra recebida. O primeiro exemplo é de um erro simples sobre os bits de dados, como mostrado na Figura 46. O processo de decodificação inicia com as síndromes de verificação. Como podemos ver em (2), nenhum erro é detectado sobre os bits de redundância. Na parte (3) da figura, podemos ver como as síndromes de paridade de linha e coluna atuam, apontando a posição do erro, que facilmente é corrigido com a inversão do valor armazenado. Após a reversão do *interleaving*, nenhum erro é detectado e dos dados são colocados na saída do decodificador.

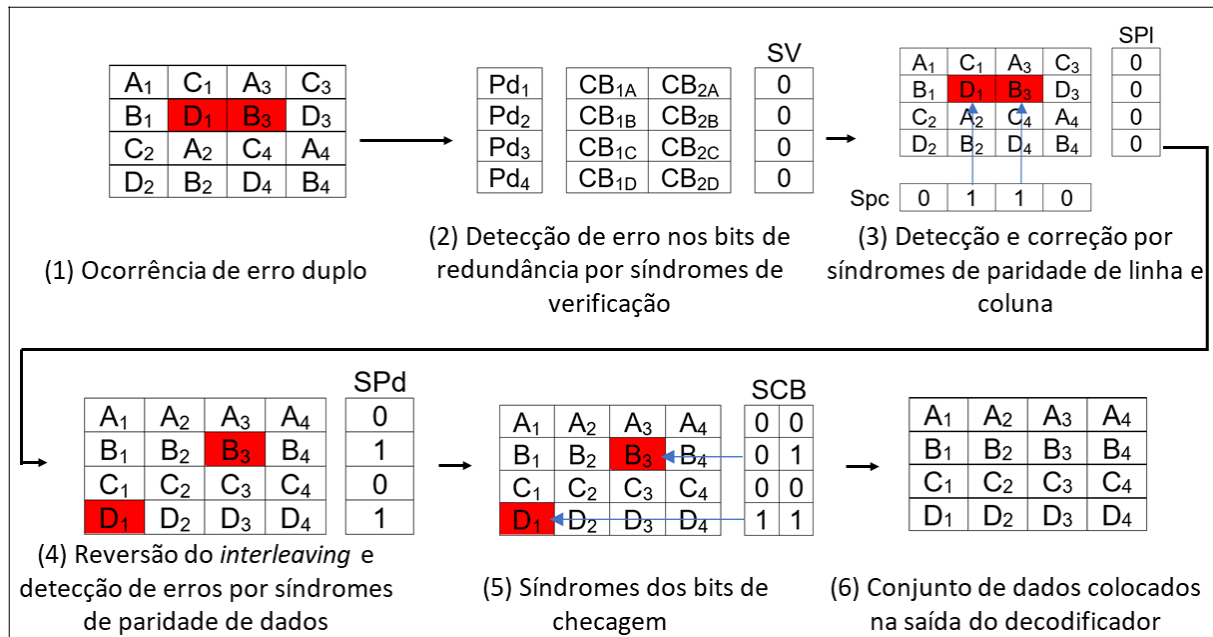
Figura 46 – Decodificação de erro simples PHICC(32,16).



Fonte: Autor.

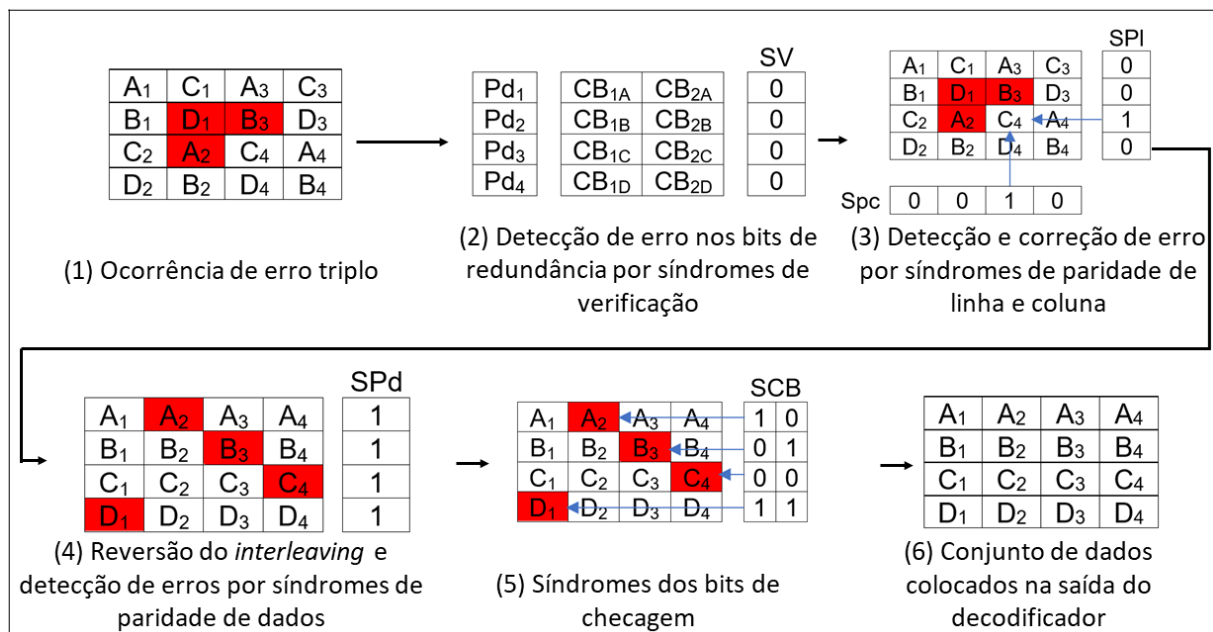
A Figura 47, mostra um padrão de erro duplo sobre os bits de dados. Na parte (2) percebemos que as síndromes de verificação não apontam para erros nos bits de redundância. Na parte (3) as síndromes de paridade de linha e coluna são calculadas, mas não são capazes de apontar as posições dos erros. Na parte (4), com a reversão do *interleaving* e as síndromes de paridade de dados, os erros são detectados para que, através das síndromes dos bits de checagem, possa ocorrer a correção. Após este processo, os bits de dados são colocados na saída.

Figura 47 – Decodificação de erro duplo PHICC(32,16).



Fonte: Autor.

Figura 48 – Decodificação de erro triplo PHICC(32,16).



Fonte: Autor.

A Figura 48 mostra o mesmo padrão de erro triplo demonstrado nas subseções anteriores. É possível notar, através da parte (3) da figura, que as síndromes de paridade de linha também não são capazes de apontar as posições dos erros. Apenas após a reversão do *interleaving* e o cálculo das síndromes de paridade de dados, na parte (4), os erros são

detectados. Na parte (5) ocorre a decodificação por bits de checagem, onde as posições dos erros são apontadas. Após a correção os bits de dados são colocados na saída.

3.5 Aplicabilidade dos códigos PHICC

Em relação capacidade de correção do PHICC, um dos fatores que mais influenciam é a quantidade de bits de redundância. Ao ser utilizada uma quantidade menor de bits, as informações se tornam mais condensadas, ocasionando um impacto maior na ocorrência de erros e dificultando a correção. Com uma quantidade maior de bits, a capacidade de recuperação é aprimorada.

O PHICC(44,16) é o mais eficaz na correção de MBUs, no entanto tende a elevar o consumo de memória devido a necessidade de armazenar uma maior quantidade de bits de redundância. Por esse motivo, a sua utilização é recomendada para aplicações críticas em ambientes de maior exposição à radiação ionizante, que necessitem garantir a integridade dos dados e que possuam maior capacidade de armazenamento, como missões espaciais e satélites, por exemplo.

O PHICC(40,16) e o PHICC(36,16) tendem a ser os códigos mais equilibrados em relação à capacidade de correção e uso de memória. A principal diferença entre esses dois códigos são as estratégias utilizadas nos bits de checagem, sendo o código de Hamming(7,4) empregado no PHICC(40,16) e uma codificação própria, com redução de 4 bits, aplicada ao PHICC(36,16). A capacidade de correção desses códigos é semelhante, sendo o PHICC(36,16) ligeiramente inferior. Esses códigos podem ser aplicados a sistemas críticos que necessitem de alta cobertura de correção, mas que possuam uma certa restrição na capacidade de armazenamento e massa limitada, como sistemas embarcados de aplicações críticas e CubeSats.

O PHICC(32,16) é o código com menor consumo de memória e o mais complexo em termos de quantidade de operações lógicas. As melhorias aplicadas a esse código o tornam apto para aplicações com maiores restrições na capacidade de armazenamento. Devido a quantidade de operações lógicas, o PHICC(32,16) possui uma tendência de possuir maior latência e suas características o tornam recomendado para sistemas embarcados menos suscetíveis a MBUs. Através da Tabela 8 podemos observar um resumo das principais características dos códigos PHICC.

Todas as versões do PHICC são extensíveis, o que torna possível a sua utilização em palavras que contenham diferentes quantidades de bits de dados, a partir do ajuste na quantidade de bits de redundância. Além disso, a alta capacidade de evitar correções errôneas, as adaptações realizadas sobre a codificação de Hamming e o compartilhamento de paridade são alguns dos principais diferenciais dos códigos PHICC em relação aos demais códigos.

Tabela 8 – Características dos códigos PHICC

Código	Percentual dos bits de dados	Estratégia utilizada	Cobertura de correção.
$PHICC(44, 16)$	36,36%	Paridade, Interleaving sobre bits de dados e checagem e Hamming(8,4).	Altíssima
$PHICC(40, 16)$	40,00%	Paridade, Interleaving sobre bits de dados e Hamming (7,4).	Alta
$PHICC(36, 16)$	44,44%	Paridade, Interleaving sobre bits de dados e codificação por bits de checagem.	Alta
$PHICC(32, 16)$	50,00%	Paridade, Interleaving sobre todos os bits da palavra e codificação por bits de checagem.	Média

Fonte: Autor.

3.6 Conclusão

Neste capítulo foram apresentadas as metodologias de correção dos códigos $PHICC(32,16)$, $PHICC(36,16)$, $PHICC(40,16)$ e $PHICC(44,16)$. Todos os códigos mencionados utilizam conceitos de paridade, Hamming e *interleaving*. O código $PHICC(32,16)$ foi apresentado com uma estratégia a mais em relação as outras versões do PHICC, denominada de compartilhamento de paridade. Foram apresentados também alguns exemplos do processo de correção de cada CCE. Cada código possui características que o diferencia dos demais, sendo as principais delas: a quantidade de bits de redundância e a capacidade de correção.

Para uma melhor avaliação dos códigos PHICC como possíveis candidatos a utilização em sistemas tolerante a falhas, alguns testes de detecção, correção, custo de implementação e confiabilidade são necessários. No capítulo a seguir essas análises serão demonstradas e uma comparação entre os códigos até então estudados é feita em relação ao tratamento de MBU's.

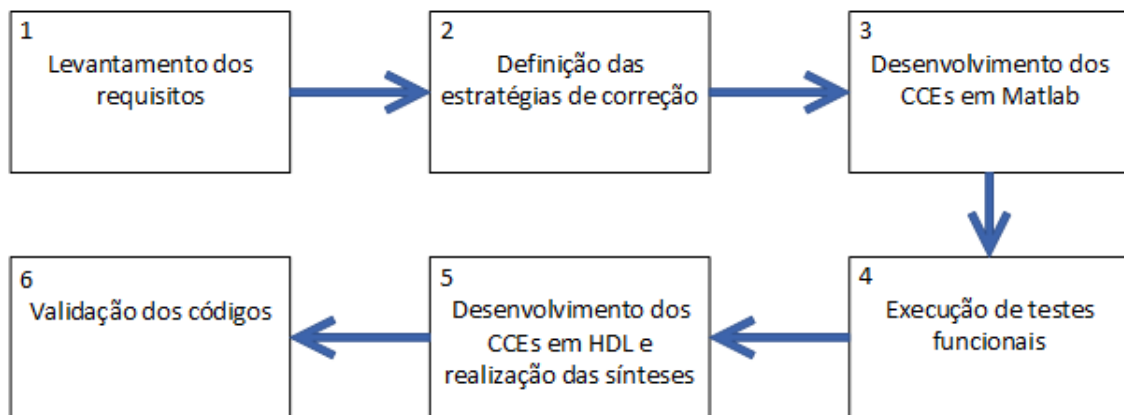
4 Materiais e métodos

Ao analisarmos as estratégias comumente utilizadas em sistemas tolerantes a falhas, percebemos que CCEs são alternativas eficazes e de baixo custo em comparação com outros métodos como TMR (*Triple Modular Redundance*) e blindagem. Por esse motivo, decidimos explorar o estudo de CCEs, especialmente os do tipo matriciais. Neste capítulo, estão descritos os processos de desenvolvimento dos CCEs da família PHICC. Também são detalhados os recursos utilizados em cada etapa deste trabalho.

4.1 Desenvolvimento dos CCEs da família PHICC

O desenvolvimento dos códigos PHICC foi realizado através de ferramentas computacionais de desenvolvimento e simulação. Para que os objetivos fossem alcançados, seguimos as etapas mostradas na Figura 49 e detalhadas nas subseções a seguir.

Figura 49 – Etapas de desenvolvimento dos códigos PHICC



Fonte: Autor.

4.1.1 Levantamento dos requisitos

A elicitação dos requisitos foi realizada na fase inicial de concepção dos códigos PHICC. Nesta etapa foram definidos os objetivos gerais da família de códigos, bem como os propósitos os quais cada código se aplica. Os pontos fortes de trabalhos já desenvolvidos foram levados em conta no levantamento dos requisitos, assim como as aplicações as quais cada código foi submetido.

O primeiro CCE a ser desenvolvido foi o PHICC(40,16), que foi criado com o objetivo de possuir maior capacidade de correção de múltiplos erros que códigos como Reed-Muller(2,5) e CLC Estendido, mantendo o custo de implementação em silício inferior.

O PHICC(36,16) e o PHICC(32,16) foram criados com o intuito de atenuar uma das desvantagens do uso de CCEs, que é o aumento do consumo de memória, o que permite a sua utilização em sistemas com restrições de armazenamento, assim como o código MRSC. Os requisitos estabelecidos sobre essas versões do PHICC foram: o consumo de memória reduzido, altas taxas de correção de erros e a conservação das principais características da primeira versão.

O PHICC(44,16) foi desenvolvido com o intuito de atender a sistemas expostos a condições, nas quais a taxa de ocorrência de MCUs é elevada. Para esse caso, necessita-se de um código com capacidade de correção de erros mais elevada. Além de buscarmos atender a esse requisito, também mantivemos as principais características do PHICC(40,16), no que diz respeito ao processo utilizado na correção e a economia de recursos.

4.1.2 Definição das estratégias de correção

Para o iniciarmos a concepção do PHICC(40,16), associamos paridade com código de Hamming estendido. No entanto, a correção apenas por esses dois métodos não foi capaz de resolver grande parte dos padrões de múltiplos erros adjacentes. Por esse motivo, resolvemos utilizar o *interleaving* para separar os erros, tornando mais fácil a correção.

Apesar de todas as versões do PHICC possuírem os processos de correção semelhantes, as estratégias utilizadas são diferentes. No caso do PHICC(36,16), que foi desenvolvido para reduzir o consumo de memória, foi necessário desenvolver uma nova estratégia de correção para substituir a codificação de Hamming utilizada no PHICC(40,16). Essa nova codificação demanda uma menor quantidade de bits de checagem e possui a capacidade de correção semelhante.

No PHICC(32,16) obtivemos uma redução do consumo de memória ainda maior em relação ao PHICC(40,16). Percebemos que, devido ao padrão de *interleaving* sobre os bits de dados, era possível que realizássemos uma lógica sobre os bits de checagem para obtermos as mesmas informações de outros bits de redundância. Com isso, tornou-se possível efetuarmos todos os processos de correção sem a utilização alguns bits.

Mesmo com a combinação de estratégias no processo de correção, observamos que alguns padrões de MCUs não eram corrigidos, pois, mesmo depois da correção por bits de paridade e da reversão do *interleaving*, ainda restavam erros duplos nas linhas dos bits de dados. Logo percebemos que precisávamos de uma codificação substituta ao código de Hamming que tornasse possível a correção de erros duplos. Então, adaptamos o código de Hamming, adicionando mais um bit de redundância, o que tornou possível a cobertura desse tipo de erro.

4.1.3 Desenvolvimento dos CCEs em Matlab e execução de testes funcionais

Após a definição das estratégias de correção, escrevemos *scripts* no *software* Matlab para simularmos os processos de codificação e decodificação de cada versão do PHICC. O *script* codificador recebe na entrada um conjunto de 16 bits de dados, realiza as operações para a obtenção dos bits de redundância, agrupa os bits de dados juntamente com os bits de redundância em forma de matriz, que, por sua vez, são colocados na saída. O *script* decodificador recebe na sua entrada uma palavra codificada. Em seguida, realiza as operações de decodificação a fim de detectar e corrigir erros. Após a realização das operações, o conjunto de 16 bits de dados é colocado na saída do decodificador.

Para verificar o funcionamento dos códigos, um *script* de teste foi desenvolvido. A sua principal função foi conferir as saídas, conforme dados eram inseridos nas entradas dos codificadores e decodificadores. Dessa maneira foi possível determinar o comportamento de cada CCE, além de verificar os padrões de erros corrigíveis.

4.1.4 Desenvolvimento dos CCEs em HDL e realização da síntese

Após o desenvolvimento e a realização de testes em Matlab, cada código da família PHICC foi escrito na linguagem de descrição de *hardware* VHDL. O ambiente de desenvolvimento utilizado foi o *software* Modelsim em conjunto com Quartus II Web Edition, da Altera. Além dos códigos da família PHICC, os códigos Reed-Muller(2,5), CLC Estendido e MRSC também foram escritos em HDL. A fim de verificarmos os valores de custo de implementação em silício, todos os códigos foram submetidos a testes de síntese, realizados na ferramenta Cadence GENUS, utilizando a tecnologia CMOS de 28 nm.

4.2 Validação dos códigos

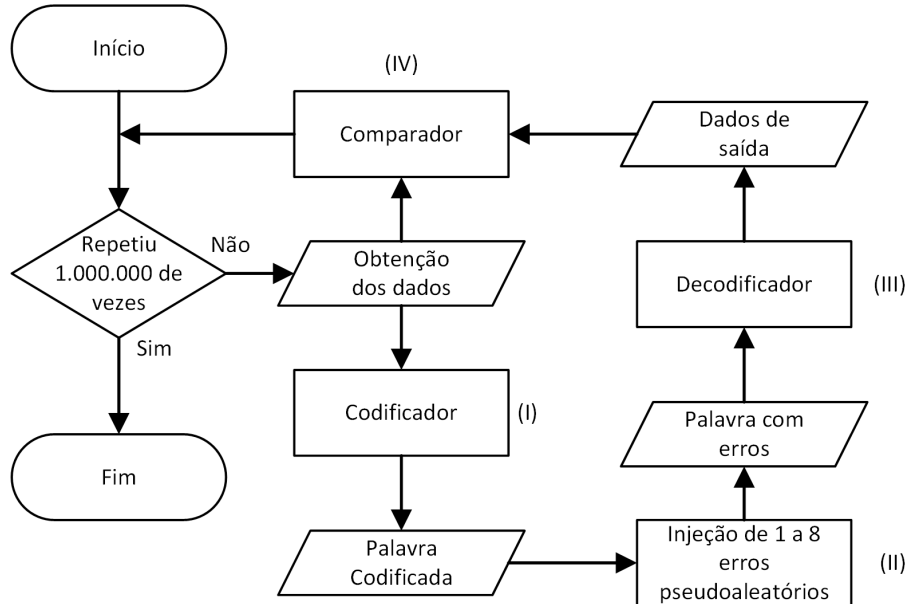
A validação dos códigos PHICC foi realizada através de testes comparativos com os códigos Reed Muller (2,5), MRSC e CLC Estendido. O código Reed Muller(2,5) foi escolhido por ser considerado um código clássico robusto e amplamente utilizado. Já os códigos MRSC e CLC Estendido foram escolhidos para fins de comparação por possuírem características semelhantes as dos códigos PHICC, em termos de quantidade de bits de redundância e estratégias utilizadas. Todos os códigos foram submetidos aos mesmos testes e os parâmetros analisados foram: taxas de correção, capacidade de correção no tempo, confiabilidade, Mean Time to Failure (MTTF) e custo de implementação em silício.

4.2.1 Taxas de correção de erros

A análise das taxas de correção consiste na simulação da ocorrência de eventuais erros e na análise de capacidade de cada código em corrigi-los. Foram utilizados os *scripts*

desenvolvidos em Matlab e em cada cenário de teste, o processo ilustrado pela Figura 50 foi realizado.

Figura 50 – Etapas para obtenção das taxas de correção.



Fonte: Autor.

Na primeira parte, um conjunto de 16 bits de dados é obtido de forma pseudoaleatória, em seguida passa pelo codificador, que gera uma palavra codificada, e os mesmos dados são inseridos em um comparador. No segundo estágio, são inseridos de 1 a 8 erros adjacentes, pois, de acordo com Satoh, Tosaka e Wender (2000) Liu et al. (2017) e Radaelli et al. (2005), múltiplos erros tendem a ocorrer em células vizinhas. No terceiro estágio, a palavra contendo erros é recebida pelo decodificador e o processo de correção é realizado. No quarto estágio é realizada a comparação entre a saída do decodificador e os dados de entrada e os resultados são contabilizados. Para cada quantidade de erros, o teste é realizado um milhão de vezes, para garantir a abrangência de todos os padrões de erros possíveis.

4.2.2 Análise de capacidade de correção no tempo

De acordo com Argyrides, Zarandi e Pradhan (2007) e Miremadi e Zarandi (2005), a ocorrência de SEEs pode ser representada por uma distribuição de Poisson, sendo cada falha estatisticamente independente. Neste trabalho, a mesma distribuição de probabilidade é utilizada, de forma que a capacidade de correção em relação ao tempo, possa ser regida pela Equação 4.1. Através desta equação, é possível obtermos o valor da probabilidade de correção, para cada instante de tempo t , como uma função de probabilidade de correção (FC) dada uma possibilidade de ocorrência de falha na memória (MF), no qual Me é o

número máximo de erros, que no caso deste experimento, foi considerado a ocorrência de até oito erros. Para a realização dos experimentos foram utilizados *scripts* no *software* Matlab.

$$f_{Correction}(t) = P\{FC|MF\} = \sum_{i=1}^{Me} P\{FC|iE\} * P\{iE|MF\} \quad (4.1)$$

A probabilidade de ocorrência de i erros ($P(iE)$), no caso de uma memória em falha (MF), pode ser obtida através da Equação 4.2

$$P\{iE|MF\} = \frac{P\{iE\}}{P\{MF\}} \quad (4.2)$$

Na Equação 4.3 obtemos o valor da probabilidade da ocorrência de i erros em uma palavra contendo d bits de dados e p bits de redundância, sendo λ a taxa de erros em bits por dia e t o tempo. A taxa de erros utilizada neste teste foi $\lambda = 10^{-5}$.

$$P\{iE\} = \binom{d+p}{i} * (1 - e^{-\lambda t})^i * e^{-\lambda(d+p-i)t} \quad (4.3)$$

A probabilidade de a memória entrar em falha é obtida através Equação 4.4, a partir dos mesmos parâmetros utilizados na Equação 4.3.

$$P(MF) = 1 - e^{-\lambda(d+p)t} \quad (4.4)$$

Por fim, a função que determina a capacidade de correção no tempo é dada pela Equação 4.5, que relaciona as equações anteriormente descritas com as taxas de correção de erros, sendo estas consideradas como as probabilidades de correção de falha, dada a ocorrência de i erros ($P(FC|iE)$).

$$P(FC|MF) = P(FC|iE) \frac{\binom{d+p}{i} * (1 - e^{-\lambda t})^i * e^{-\lambda(d+p-i)t}}{1 - e^{-\lambda(d+p)t}} \quad (4.5)$$

4.2.3 Análise de confiabilidade e MTTF

A análise de confiabilidade foi realizada através de *scripts* escritos no *software* Matlab. Os experimentos descritos a seguir foram realizados sobre dois conjuntos de dados: o primeiro deles, uma palavra codificada, no qual chamamos de registro; e o segundo, um conjunto de 16 registros. A confiabilidade de cada código pode ser estimada através da probabilidade de não haver de erros ($P(NE)$) mais a probabilidade de, na ocorrência de

i erros, a correção ser realizada com sucesso ($P(iCS)$). Desse modo, podemos avaliar a confiabilidade de cada código no tempo através dos resultados da Equação 4.6.

$$r(t) = P\{NE\} + \sum_{i=1}^{Me} P\{iE\} * P\{iCS\} \quad (4.6)$$

Uma memória, que pode armazenar uma quantidade M de registros, possui um valor de confiabilidade para cada um deles, obtido pela Equação 4.6. O valor resultante para a confiabilidade da unidade de memória como um todo é igual a multiplicação dos valores obtidos para cada registro. Como os valores dos registros em termos de taxa de correção e quantidade de bits de redundância devem ser os mesmos, as confiabilidades também devem permanecer com os mesmos valores, isso implica na possibilidade de calcular a confiabilidade da unidade de memória através de uma potência de ordem M do resultado obtido para uma palavra, como mostrado na Equação 4.7.

$$R(t) = r^M(t) \quad (4.7)$$

O tempo médio para que uma memória entre em falha pode ser estimado através da área sob a curva gerada pela confiabilidade no tempo. Desse modo, é possível calcular o MTTF das memórias protegidas com cada código através da Equação 4.8. Quanto mais robusto o código, maior a área sob a curva de confiabilidade e, conseqüentemente, maior o MTTF.

$$MTTF = \int_0^{\infty} R(t)dt \quad (4.8)$$

4.2.4 Análise de custo de implementação em silício

A análise de custo de implementação em silício é realizada levando em conta a área consumida em chip, a potência dissipada e a latência como critérios. Esses dados foram obtidos na realização da síntese dos códigos escritos em HDL. Para a obtenção desses dados, cada CCE foi sintetizado através da ferramenta Cadence GENUS, utilizando a tecnologia CMOS de 28nm. Os resultados de cada CCE equivalem às somas dos dados obtidos para o codificador e decodificador. O valor correspondente ao custo total é obtido através da Equação 4.9.

$$Custo\ total = \acute{A}rea * Pot\ência * Lat\ência \quad (4.9)$$

Para melhor apresentar os resultados baseados no equilíbrio entre o custo e a cobertura de correção, criamos a métrica CCC (Cobertura de Correção por Custo) que

relaciona a taxa de correção com o custo total de cada código. Essa métrica pode ser obtida através da Equação 4.10.

$$CCC = \frac{\text{Cobertura de correção}}{\text{Custo}} \quad (4.10)$$

4.3 Conclusão

Este capítulo apresentou os métodos e os recursos utilizados no processo de desenvolvimento e teste da família PHICC. Os métodos adotados em cada etapa, foram os meios pelos quais se tornou possível alcançarmos os objetivos definidos. A ferramenta utilizada para síntese permitiu a verificação dos principais aspectos relacionados ao custo de implementação. Os resultados obtidos através da realização dos testes descritos são mostrados no capítulo a seguir.

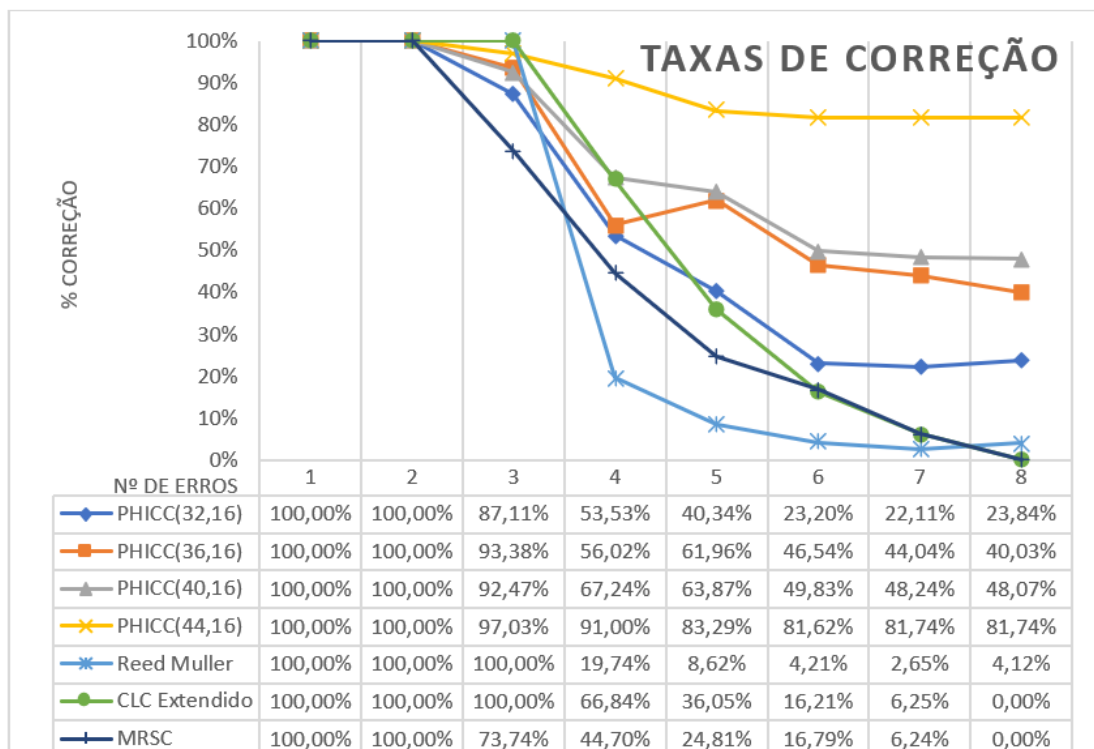
5 Resultados

Este capítulo mostra os resultados obtidos através dos experimentos descritos no capítulo anterior, sendo dividido em cinco seções: a Seção 5.1 mostra os resultados das taxas de correção de erros dos CCEs; a Seção 5.2 expõe os resultados da capacidade de correção no tempo; na Seção 5.3, os resultados de confiabilidade e MTTF são mostrados; a Seção 5.4 mostra os dados de síntese em silício; e, na Seção 5.5, é feita a conclusão do capítulo.

5.1 Análise de correção de erros

Através da Figura 51 é possível observar um impacto gradativo sobre a taxa de correção de erros de cada CCE analisado, conforme o número de bits errados aumenta. Em relação ao código PHICC(44,16), é possível notar que a inclusão de 4 bits de redundância a mais, em comparação com o código original, permitiu um aumento de em média de 28,43% das taxas de correção a partir de 4 erros. As demais versões do código PHICC seguem uma tendência de redução das taxas de correção a partir de 4 erros, conforme o número de bits de redundância diminui.

Figura 51 – Taxas de correção de erros.



Fonte: Autor.

Ao realizarmos uma comparação dos códigos PHICC com as demais codificações, podemos perceber que os resultados são iguais até 2 erros. Para 3 erros, os códigos Reed Muller (2,5) e CLC Estendido possuem melhores resultados, seguidos de perto pelo código PHICC(44,16) por uma diferença de 2,97%. A partir de 4 erros, os códigos PHICC(44,16) e PHICC(40,16) passam a ter as melhores taxas de correção, com médias de 83,87% e 55,45%, respectivamente. A partir de 5 erros, o PHICC(36,16) permaneceu superior ao código Reed Muller (2,5), com taxas de correção em média nove vezes maiores; ao CLC Estendido, corrigindo em média três vezes mais erros; e ao MRSC, corrigindo em média quatro vezes mais erros. O PHICC(40,16) possui resultados semelhantes ao PHICC(36,16), sendo ligeiramente melhores. O PHICC(32,16) foi superior ao código Reed Muller (2,5), com taxas de correção em média cinco vezes maiores; ao CLC Estendido, corrigindo em média duas vezes mais erros; e ao MRSC, corrigindo em média duas vezes mais erros.

Apesar de ser amplamente utilizado, o código Reed Muller (2,5) apresentou uma queda de 80,26% na taxa de correção, a partir de 4 erros, o que evidencia uma deficiência na correção de MBUs de padrões mais complexos. Os códigos MRSC e PHICC(32,16) são semelhantes em relação ao número de bits de redundância, no entanto, o código PHICC(32,16) possui taxas de correção em média 50,43% superiores. De maneira semelhante, os códigos CLC Estendido e PHICC(40,16) se assemelham em número de bits de redundância, mas o código PHICC(40,16) se sobressai a partir de 4 erros, com valores em média duas vezes maiores.

As versões dos códigos PHICC se assemelham em relação à estratégia utilizada, no entanto, a capacidade de correção diminui nas versões com redução de bits de redundância. O PHICC(44,16) corrige, em média, 25,75% a mais que o PHICC(40,16), 32,19% a mais que o PHICC(36,16) e 59,16% a mais que o PHICC(32,16). Esses resultados podem ser explicados devido ao fato de uma menor quantidade de bits de redundância concentrarem uma maior carga de informações sobre os dados. Desse modo, os erros que ocorrem nesses bits geram consequências mais duras, aumentando a dificuldade no processo de correção.

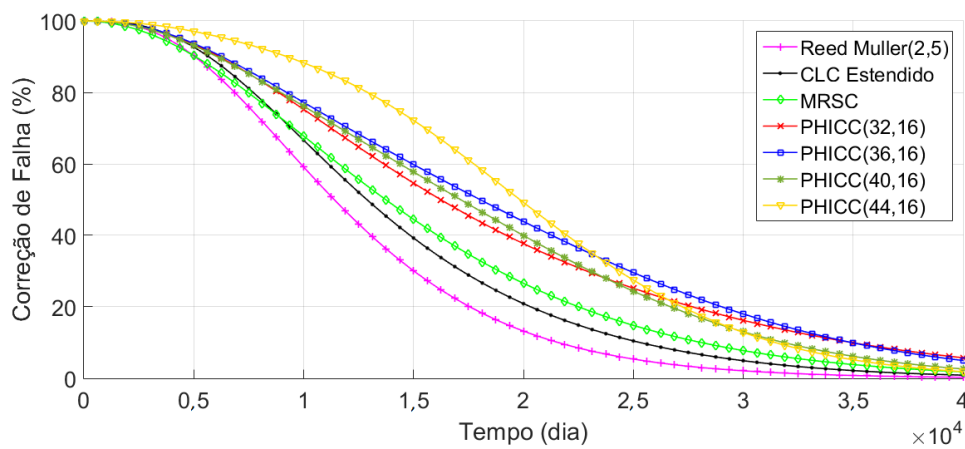
Os resultados expressivos dos códigos PHICC são justificados pela sua capacidade de dividir um erro múltiplo em erros de padrões mais simples de serem corrigidos. Além disso, sua capacidade de detectar a ocorrência de erros sobre os bits de redundância permite que correções errôneas sejam evitadas. Para o PHICC(44,16), a capacidade de corrigir erros duplos através dos bits de checagem tornou possível a correção de padrões de múltiplos erros não cobertos pelas demais versões do PHICC.

5.2 Análise de capacidade de correção no tempo

Ao observarmos a Figura 52, podemos notar que, com o passar do tempo, todos os códigos apresentam um decaimento na capacidade de correção de erros. O resultado

pode ser explicado devido ao fato de a probabilidade da ocorrência de MBUs aumentar em uma janela de tempo maior. No entanto, a maneira como cada código consegue lidar com a ocorrência de erros mais complexos é diferente. O código Reed Muller(2,5) apresenta o decaimento mais acentuado, obtendo os menores resultados entre todos os códigos. Até cerca de 8.000 dias, o código CLC Estendido possui resultados semelhantes aos do MRSC, no entanto, o código MRSC passa a obter melhores resultados a partir desse ponto, chegando a diferença de cerca de 6%.

Figura 52 – Capacidade de correção no tempo.



Fonte: Autor.

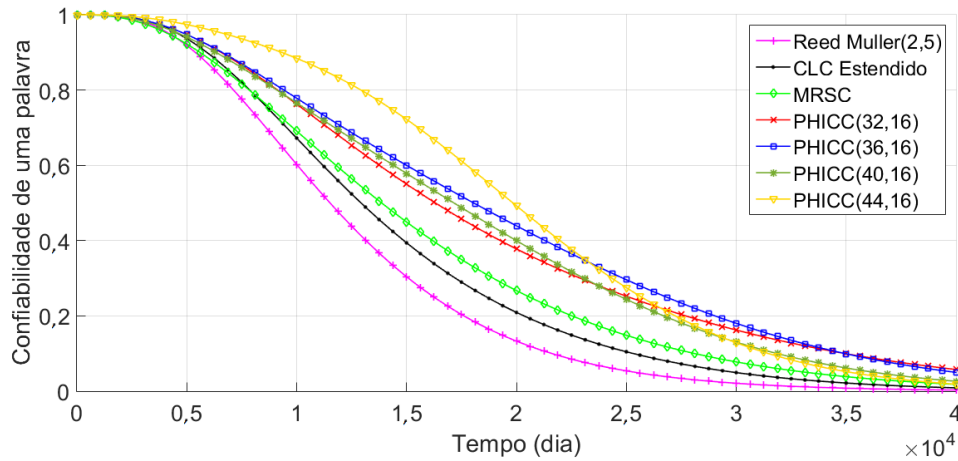
Todas as versões do código PHICC, possuem melhores resultados em comparação com os demais códigos, corrigindo entre 39% e 49% dos casos de MCUs após 20.000 dias, enquanto que os códigos Reed-Muller(2,5), CLC Estendido e MRSC ficam com resultados de 15%, 22% e 28%, respectivamente. A diferença entre os códigos PHICC e os demais códigos é de pelo menos 11% neste período de tempo. Até cerca de 15.000 dias, o PHICC(44,16) se destaca com resultado de 74%, seguido pelo PHICC(36,16) com 61%, PHICC(40,16) com 60%, PHICC(32,16) com 57%, MRSC com 47%, CLC Estendido com 42% e Reed-Muller(2,5) com 33%. A partir de 22.500 dias, os códigos PHICC alternam entre si com os melhores resultados.

Os dois principais fatores que explicam as diferenças nos resultados dos códigos são as taxas de correção de erros e a quantidade de bits de redundância por palavra. Maiores taxas de correção influenciam positivamente permitindo uma maior capacidade de correção. Por outro lado, a maior quantidade de bits de redundância leva a maior susceptibilidade de ocorrência de erros, principalmente devido ao aumento da área de cobertura em circuitos integrados.

5.3 Análise de confiabilidade e MTTF

Através das Figuras 53 e 54 podemos observar os resultados obtidos ao submetermos os códigos aos experimentos descritos anteriormente. A taxa de ocorrência de erros utilizada foi $\lambda = 10^{-5}$.

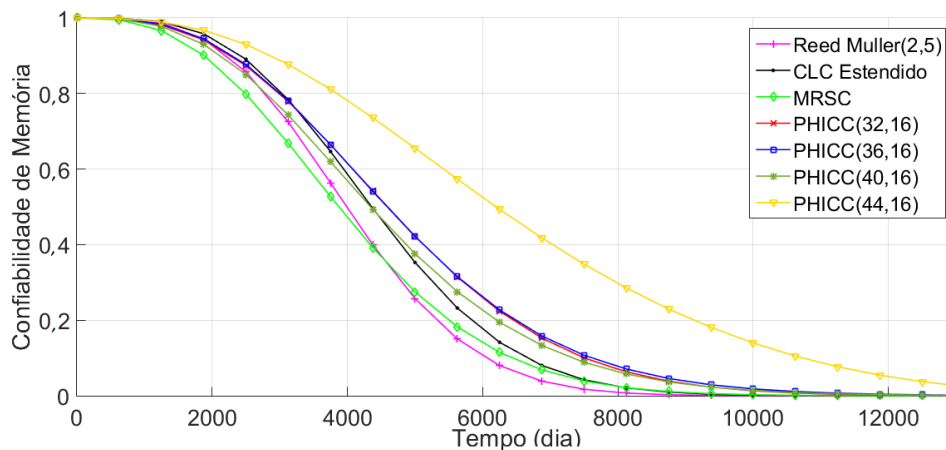
Figura 53 – Confiabilidade de registro de memória com 16 bits de dados.



Fonte: Autor.

Na Figura 53 podemos perceber que, para cada código, a curva de confiabilidade no tempo de um registro contendo 16 bits de dados e bits de redundância se assemelha com a curva obtida na análise de capacidade de correção no tempo. Isso ocorre devido ao fato de a capacidade de correção influenciar diretamente na confiabilidade. Assim como na análise anterior, todas as versões do código PHICC obtiveram melhores resultados, sendo em média 14% superiores. O PHICC(44,16) se destaca até cerca de 22.500 dias, tendo resultado médio, no mínimo, 28% superior aos dos outros códigos estudados.

Figura 54 – Confiabilidade de memória contendo 16 registros.



Fonte: Autor.

A Figura 54 mostra o resultado do teste de confiabilidade no tempo de uma memória contendo 16 registros, nos quais cada um deles possui 16 bits de dados e bits de redundância. Nota-se que a confiabilidade de uma memória decai mais rapidamente em comparação com os resultados mostrados na Figura 53, para um único registro. Isso ocorre devido ao aumento da susceptibilidade a falhas, causado pelo acréscimo de células de armazenamento. Uma quantidade maior de células de memória implica em um maior consumo de área em chip, logo é prevista uma maior exposição a interferências e, conseqüentemente, a ocorrência de erros. Até cerca de 4.500 dias o código Reed Muller(2,5) se mostra em média 4% superior ao MRSC, no entanto, com o passar do tempo o MRSC passa a ter melhores resultados. Embora o CLC Estendido se mostre ligeiramente superior ao PHICC(40,16) até cerca de 4.000 dias, ele é ultrapassado, mantendo em média o resultado 8% inferior. Os códigos PHICC(36,16) e PHICC(32,16) obtiveram resultados semelhantes, o que nos leva a crer que a menor quantidade de bits de redundância do PHICC(32,16) reduziu a sua susceptibilidade a falhas a ponto de compensar as menores taxas de correção. Assim como nos demais estudos, o PHICC(44,16) alcançou os melhores resultados, sendo em média superior ao PHICC(40,16) em 29%; ao PHICC(36,16) e PHICC(32,16) em 26%; ao Reed-Muller(2,5) e MRSC em 37%; e ao CLC Estendido em 31%.

A partir dos dados de confiabilidade de uma memória com 16 palavras codificadas, a Equação 4.8 foi utilizada a fim de obtermos o MTTF de cada código. Os valores obtidos podem ser conferidos através da Tabela 9.

Tabela 9 – MTTF (*Mean Time to Failure*), em dias, para memórias de 16 registros.

Taxa de erros por dia	Reed-Muller (2,5)	MRSC	CLC Estendido	PHICC (32,16)	PHICC (36,16)	PHICC (40,16)	PHICC (44,16)
$\lambda = 10^{-4}$	408,18	404,54	446,99	477,98	481,95	458,14	654,38
$\lambda = 10^{-5}$	4.081,88	4.045,46	4.469,99	4.779,84	4.819,52	4.581,41	6.543,83
$\lambda = 10^{-6}$	40.818,89	40.454,67	44.699,96	47.798,47	48.195,25	45.814,17	65.438,38

Fonte: Autor.

5.4 Análise de custo de implementação

Os resultados das análises de custo de implementação podem ser conferidos através da Tabela 10. Ao observarmos o custo total de cada CCE, notamos que o Reed Muller(2,5) possui o maior valor, enquanto o MRSC possui o menor. O código PHICC(36,16) possui o segundo menor custo de implementação, seguido pelo PHICC(32,16), PHICC(40,16) e PHICC(44,16). Apesar do PHICC(32,16) possuir menor quantidade de bits de redundância, o seu custo de implementação é maior em comparação com o PHICC(36,16). Isso pode

ser explicado pelo fato de haver uma maior quantidade de operações lógicas entre os bits de redundância do PHICC(32,16), o que pode aumentar o tamanho do circuito e, por consequência, aumentar a potência dissipada e a latência.

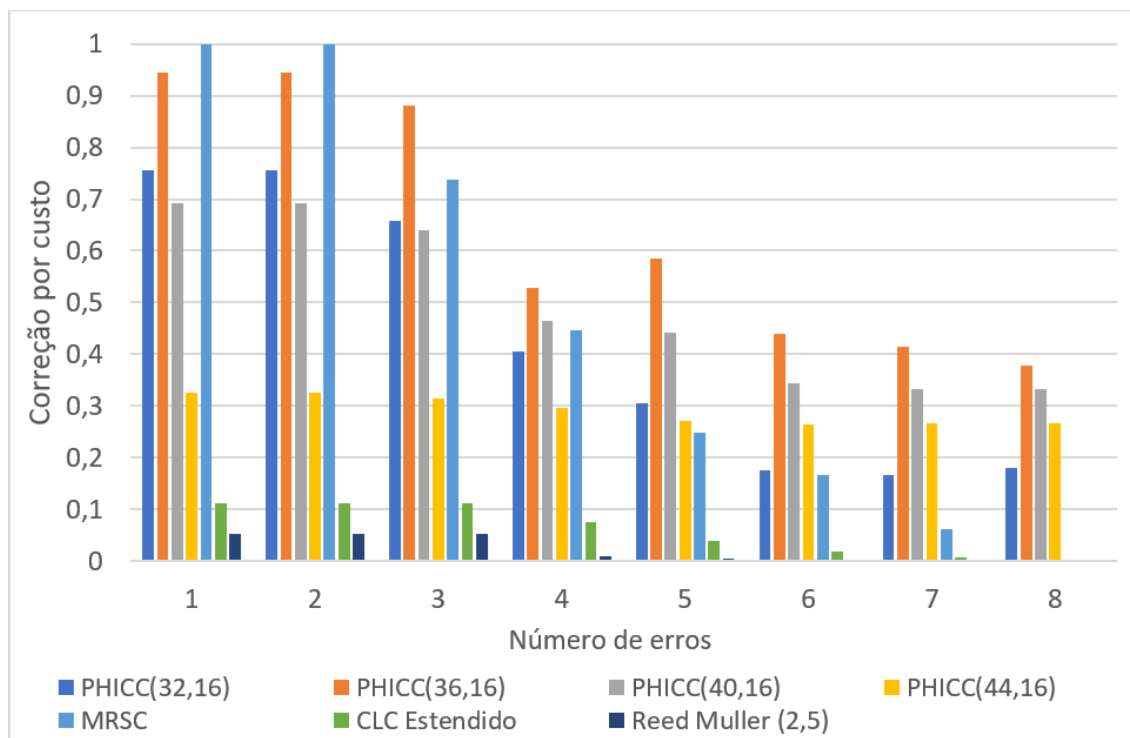
Tabela 10 – Custos de área, potência, latência e custo total

Código	Área (μm^2)	Potência (mW)	Latência (ns)	Total ($\mu m^2 mW ns$)
<i>PHICC</i> (32, 16)	419	0,137	1,276	73,194
<i>PHICC</i> (36, 16)	383	0,154	0,998	58,684
<i>PHICC</i> (40, 16)	437	0,177	1,033	80,105
<i>PHICC</i> (44, 16)	591	0,217	1,326	170,163
<i>MRSC</i>	363	0,145	1,050	55,368
<i>CLCEstendido</i>	814	0,244	2,481	491,997
<i>Reed Muller</i> (2, 5)	1090	0,424	2,272	1049,215

Fonte: Autor.

Na Figura 55 podemos observar os resultados obtidos através da métrica CCC. Para facilitar a compreensão, os resultados foram normalizados, sendo 1 o valor que representa o melhor equilíbrio entre cobertura de correção e custo de implementação.

Figura 55 – Resultados para cobertura de correção por custo de implementação



Fonte: Autor.

Os resultados mostram que, para até 2 erros, o código MRSC possui o melhor

equilíbrio, devido ao fato de possuir menor custo. No entanto, como a cobertura de correção do MRSC diminui consideravelmente com a ocorrência de MBUs, a partir de 3 erros ele é superado. O PHICC(36,16) supera todos os demais códigos a partir de 3 erros, além de seguir de perto o MRSC para até 2 erros. Isso demonstra grande versatilidade para utilização em todos os cenários.

É possível notar que entre o CLC Estendido, Reed Muller(2,5) e PHICC(44,16), considerados os CCEs mais robustos neste trabalho, O PHICC(44,16) se destaca, pois possui maiores taxas de correção e menor custo de implementação, o que eleva o valor de CCC. Com isso, para aplicações que exijam alta capacidade de correção de MBUs, o PHICC(44,16) é um forte candidato para uso. O PHICC(32,16) e o PHICC(40,16), apresentaram valores intermediários para CCC.

5.5 Conclusão

Neste capítulo, foram mostrados os dados obtidos a partir das análises realizadas para validação dos códigos PHICC. Conforme podemos verificar, cada versão do PHICC obteve os resultados desejados, com valores que superam os demais códigos de características semelhantes, na maioria dos cenários de teste. É possível perceber que o nível de complexidade, a quantidade de bits de redundância e a combinação de estratégias foram fatores determinantes para esses resultados.

6 Conclusão

É apresentada e descrita neste trabalho, uma nova família de códigos corretores de erros matriciais capazes de lidar com MBUs em memórias do tipo SRAM, tendo como principais objetivos a utilização em aplicações críticas e atender a diferentes requisitos de confiabilidade e custo de implementação. Os códigos PHICC são baseados paridade e *interleaving* e Hamming, cujas implementações possuem três versões diferentes: uma original, outra com redução no número de bits de redundância e outra capaz de corrigir erros duplos. As maneiras como esses conceitos de correção foram utilizados permitiu a criação de códigos robustos, cujas capacidades foram validadas através de experimentos e análises comparativas. Os experimentos realizados para validação dos códigos foram: obtenção das taxas de correção, análise de custo de área, potência e tempo de execução, taxa de correção por custo, capacidade de correção por tempo, confiabilidade por tempo e MTTF.

Os resultados obtidos nos experimentos demonstram que os códigos PHICC são capazes de atender critérios de confiabilidade mais exigentes em comparação aos demais códigos mostrados. Na ocorrência de quatro a oito erros, suas taxas de correção são mais expressivas, sendo o PHICC(32,16) e o PHICC(44,16) superiores em até 19,72% e 77,62%, respectivamente. A partir de cinco erros, todas as versões do PHICC possuem melhor equilíbrio em termos de capacidade de correção por custo. Em relação ao MTTF, todos os códigos PHICC obtiveram resultados superiores, com destaque para o PHICC(44,16), que é capaz de manter o sistema sem falhas por um tempo 31,69% maior que o melhor resultado dos demais códigos. O PHICC(44,16) obteve grande destaque por sua capacidade de correção muito superior, apesar do custo 65,41% inferior ao do CLC Estendido e 83,77% ao do Reed-Muller(2,5). O PHICC(40,16) é um código de resultados intermediários em relação às demais versões. Já o PHICC(36,16) se sobressai por sua alta capacidade de correção de erros apesar do custo total equivalente ao do MRSC, com diferença de apenas 5,6%. O PHICC(32,16) é equivalente ao MRSC em consumo de memória, no entanto possui taxas de correção em média 10% melhores e resultados de confiabilidade em média 12% superiores.

Podemos dizer que os códigos PHICC atingiram o propósito de serem códigos de grande robustez a um baixo custo de implementação em silício. As diferentes estratégias utilizadas nas codificações possibilitaram que cada código assumisse características próprias, capazes de atender a diferentes quesitos. Conclui-se então, baseado nos experimentos, que os códigos PHICC são alternativas viáveis para a proteção de dados, tornando dispositivos de memória tolerantes a falhas.

Os códigos PHICC apresentaram resultados satisfatórios, no entanto, para trabalhos futuros, desejamos alcançar algumas melhorias nos dados de correção, principalmente para a ocorrência de três erros. Também pretendemos desenvolver versões estendidas de cada código, tornando-os capazes de proteger uma quantidade maior de bits de dados. Com isso, os códigos PHICC serão mais confiáveis e versáteis.

Referências

- 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*, p. 340–348, 2007. Citado na página 82.
- BAEG, S.; WEN, S.; WONG, R. Sram interleaving distance selection with a soft error failure model. *IEEE Transactions on Nuclear Science*, v. 56, n. 4, p. 2111–2118, 2009. Citado na página 44.
- BATH, J. L.; DYER, C. S.; STASSINOPOULOS, E. G. Space, atmospheric, and terrestrial radiation environments. *IEEE Transactions on Nuclear Science*, v. 50, n. 3, p. 466–482, 2003. Citado na página 29.
- BAUMANN, R. Soft errors in advanced computer systems. *IEEE Design Test of Computers*, v. 22, n. 3, p. 258–266, 2005. Citado 2 vezes nas páginas 29 e 30.
- CASTRO, H. et al. A correction code for multiple cells upsets in memory devices for space applications. *2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)*, p. 1–4, 2016. Citado na página 37.
- CHEN, J.; WANG, J. Research and implementation of an improved reed decoding algorithm. *2002 6th International Conference on Signal Processing*, v. 2, p. 1763–1765, 2002. Citado na página 35.
- 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, 2004. Citado na página 25.
- DAS, A.; TOUBA, N. A. A new class of single burst error correcting codes with parallel decoding. in *IEEE Transactions on Computers*, v. 69, n. 2, p. 253–259, 2020. Citado na página 25.
- FARHEEN, N.; PANDE, K. S. Error detection and correction using rp sec-ded. *2020 4th International Conference on Electronics, Materials Engineering Nano-Technology (IEMENTech)*, p. 1–6, 2020. Citado na página 41.
- FREITAS, D. et al. Pcosa: A product error correction code for use in memory devices targeting space applications. *Integration, the VLSI Journal(2020)*, v. 74, p. 71–80, 2020. Citado na página 41.
- FREITAS, D. et al. Lpc: An error correction code for mitigating faults in 3d memories. *IEEE Transactions on Computers*, p. 1–12, 2020. Citado na página 41.
- GHERMAN, V. et al. Programmable extended sec-ded codes for memory errors. *VLSI Test Symposium (VTS)*, p. 140–145, 2011. Citado na página 26.
- GOERL, R. C. et al. An efficient edac approach for handling multiple bit upsets in memory array. *Microelectronics Reliability*, v. 88-90, p. 214–218, 2018. Citado na página 27.

HAMMING, R. W. Error detecting and error correcting codes. *The Bell System Technical Journal*, v. 29, n. 2, p. 147–160, 1950. Citado 2 vezes nas páginas 27 e 32.

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, 2000. Citado na página 31.

IBE, E. et al. Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule. *IEEE Transactions on Electron Devices*, v. 57, n. 7, p. 1527–1538, 2010. Citado na página 25.

JIN, Y. et al. Tmr group coding method for optimized seu and mbu tolerant memory design. *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, p. 1–5, 2018. Citado na página 32.

JUNG, S. S.; CHOI, J. P. Predicting system failure rates of sram-based fpga on-board processors in space radiation environments. *Reliability Engineering and System Safety*, p. 374–386, 2019. Citado na página 32.

KARAN, K. S.; SRIKANTH, N.; AGRAWAL, S. A robust code for mbu correction till 5-bit error. *2019 International Conference on Communication and Electronics Systems (ICCES)*, p. 1524–1529, 2019. Citado 2 vezes nas páginas 29 e 40.

KISHANI, M. et al. Hvd: Horizontal-vertical-diagonal error detecting and correcting code to protect against with soft errors. *Design Autom. for Emb. Sys.*, p. 289–310, 2011. Citado na página 29.

KUMAR, K. N. et al. Matrix based error detection and correction using minimal parity bits for memories. *2020 IEEE International Conference on Distributed Computing*, p. 100–104, 2020. Citado 2 vezes nas páginas 25 e 41.

LABEL, K. A. et al. A roadmap for nasa’s radiation effects research in emerging microelectronics and photonics. *2000 IEEE Aerospace Conference Proceedings*, v. 5, p. 535–545, 2000. Citado na página 25.

LI, J. et al. Extending 3-bit burst error-correction codes with quadruple adjacent error correction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 26, n. 2, p. 221–229, 2018. Citado na página 40.

LI, J. et al. Protecting memories against soft errors: The case for customizable error correction codes. *IEEE Transactions on Emerging Topics in Computing*, p. 1–1, 2019. Citado na página 32.

LI, J. et al. Efficient implementations of 4-bit burst error correction for memories. *IEEE Transactions on Circuits and Systems II: Express Briefs*, v. 65, n. 12, p. 2037–2041, 2018. Citado na página 27.

LI, J.; XIAO, L. H. L.; WU, H. A method to design 5-bit burst error correction code against the multiple bit upset (mbu) in memories. *2019 IEEE 13th International Conference on ASIC (ASICON)*, v. 88-90, p. 1–4, 2019. Citado na página 27.

LIN, S.; COSTELLO, D. J. *Error control coding: fundamentals and Applications*. 2. ed. New Jersey: Prentice-Hall, 2005. Citado na página 33.

- LIU, S. et al. Low redundancy matrix-based codes for adjacent error correction with parity sharing. *18th International Symposium on Quality Electronic Design (ISQED)*, p. 76–80, 2017. Citado 3 vezes nas páginas 25, 26 e 82.
- MAGALHÃES, P.; ALCÂNTARA, O.; SILVEIRA, J. Phicc: An error correction code for memory devices. *2019 32nd Symposium on Integrated Circuits and Systems Design (SBCCI)*, p. 1–6, 2019. Citado 4 vezes nas páginas 25, 43, 45 e 47.
- MIREMADI, S. G.; ZARANDI, H. Reliability of protected techniques used in fault-tolerant cache memories. *18th IEEE Annual Canadian Conference on Electrical and Computer Engineering*, p. 776–779, 2005. Citado na página 82.
- MOON, T. *Error Correction Coding, Mathematical Methods e lgorithms*. New Jersey: Wiley, 2005. Citado na página 36.
- NEALE, A.; JONKMAN, M.; SACHDEV, M. Adjacent-mbu-tolerant sec-ded-taec-yaed codes for embedded srams. in *IEEE Transactions on Circuits and Systems II: Express Briefs*, v. 62, n. 4, p. 387–391, 2015. Citado na página 27.
- PÉREZ-CELIS, A.; THURLOW, C.; WIRTHLIN, M. Emulating radiation induced multi-cell upset patterns in sram fpgas with fault injection. *IEEE Transactions on Nuclear Science*, 2021. Citado na página 25.
- RADAELLI, D. et al. Investigation of multi-bit upsets in a 150 nm technology sram device. *IEEE Transactions on Nuclear Science*, v. 52, n. 6, p. 2433–2437, 2005. Citado na página 82.
- RAHUL, K.; YACHARENI, S. Area and power efficient ecc for multiple adjacent bit errors in srams. *2020 IEEE International Conference on Consumer Electronics (ICCE)*, p. 1–4, 2020. Citado 2 vezes nas páginas 25 e 40.
- RAO, P. M. B. et al. Protecting sram-based fpgas against multiple bit upsets using erasure codes. *IEEE Design Automation Conference(DAC)*, p. 1–6, 2014. Citado na página 25.
- REVIRIEGO, P. et al. Reduction of parity overhead in a subset of orthogonal latin square codes. *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, p. 1–5, 2020. Citado na página 26.
- REVIRIEGO, P. et al. Reducing the cost of single error correction with parity sharing. *IEEE Transactions on Device and Materials Reliability*, v. 13, n. 3, p. 420–422, 2013. Citado 2 vezes nas páginas 41 e 67.
- SAMWEL, S. W. et al. Space radiation impact on smallsats during maximum and minimum solar activity. *Advances in Space Research*, v. 64, p. 239–251, 2019. Citado na página 29.
- SÁNCHEZ-MARCIÁN, A.; REVIRIEGO, P.; MAESTRO, J. A. Hamming sec-daed and extended hamming sec-ded-taed codes through selective shortening and bit placement. *IEEE Transactions on Device and Materials Reliability*, v. 14, p. 574–576, 2014. Citado na página 33.
- SATOH, S.; TOSAKA, Y.; WENDER, S. A. Geometric effect of multiple-bit soft errors onduced by cosmic ray neutrons on dram's. *IEEE Electron Device Letters*, v. 21, n. 6, 2000. Citado na página 82.

SHARMA, S.; VIJAYAKUMAR, P. An hvd based error detection and correction of soft errors in semiconductor memories used for space applications. *2012 International Conference on Devices, Circuits and Systems (ICDCS)*, p. 563–567, 2012. Citado na página 29.

SHIVAKUMAR, P. et al. Modeling the effect of technology trends on the soft error rate of combinational logic. *Proceedings International Conference on Dependable Systems and Networks*, p. 389–398, 2002. Citado na página 31.

SILVA, F. et al. An efficient, low-cost ecc approach for critical-application memories. *2017 30th Symposium on Integrated Circuits and Systems Design (SBCCI)*, p. 198–203, 2017. Citado 4 vezes nas páginas 26, 38, 39 e 40.

SILVA, F. et al. An extensible code for correcting multiple cell upset in memory arrays. *2018 Journal of Electronic Testing*, v. 34, n. 4, p. 417–433, 2018. Citado 2 vezes nas páginas 26 e 37.

SINGH, J.; SINGH, J. A comparative study of error detection and correction coding techniques. *2012 Second International Conference on Advanced Computing Communication Technologies*, p. 187–189, 2012. Citado 2 vezes nas páginas 32 e 35.

VARGHESE, B. et al. Multiple bit error correction for high data rate aerospace applications. *2013 IEEE Conference on Information Communication Technologies*, p. 1086–1090, 2013. Citado na página 35.

WIRTHLIN, M.; LEE, G. S. D.; QUINN, H. A method and case study on identifying physically adjacent multiple-cell upsets using 28-nm, interleaved and secded-protected arrays. *IEEE Transactions on Nuclear Science*, v. 61, n. 6, p. 3080–3087, 2014. Citado na página 25.

Anexos

ANEXO A – Repositório com códigos e scripts utilizados

<https://github.com/smphilippe/phicc>