
**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARA**
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**WALBER FLORENCIO DE ALMEIDA**

# COMPARAÇÃO DE CÓDIGOS CORRETORES DE ERROS EM MEMÓRIAS DE SISTEMAS EMBARCADOS

**MARACANAÚ**
**2020**

WALBER FLORENCIO DE ALMEIDA

# COMPARAÇÃO DE CÓDIGOS CORRETORES DE ERROS EM MEMÓRIAS DE SISTEMAS EMBARCADOS

Projeto de Pesquisa submetido à Coordenadoria do Curso de Bacharelado em Ciência da Computação do Instituto Federal do Ceará - Campus Maracanaú, como requisito parcial para aprovação na disciplina Trabalho de Conclusão de Curso I.

Área de pesquisa: Sistemas Embarcados

Orientador:Dr. OTÁVIO ALCÂNTARA DE LIMA JÚNIOR

Maracanaú
2020

# Sumário

# Lista de Figuras

CAPÍTULO 1

# INTRODUÇÃO

A constante evolução na microeletrônica tem proporcionado dispositivos cada vez mais eficazes e eficientes naquilo que se propõem [1]. No entanto, esse nível de integração dos circuitos acaba por causar o aumento da sensibilidade a interferências externas [2]. Segundo [3] essas interferências nas células de memória podem ocasionar Single Event Upset (SEU), quando um único bit é corrompido, ou Multiple Cell Upset (MCU), quando uma quantidade maior de bits é afetada, podendo danificar seriamente os circuitos integrados. Quando as células de memória permanecem em mau funcionamento, chama-se hard error, mas se a alteração no funcionamento é temporária, tem-se um soft error.

Nesse contexto, códigos corretores de erros são estratégias utilizadas para minimizar esses problemas, detectando e corrigindo eventuais erros através de diversos algoritmos. Muitos pesquisadores vêm utilizando, mais amplamente em sistemas embarcados, códigos que se apropriam dos conceitos de paridade e de Hamming para criar novos algoritmos.

Um dos primeiros códigos corretores de erros é o Reed-Muller, desenvolvido na década de 1950 e sendo utilizado em aplicações espaciais. Foi implementado nas sondas Mariner, que transmitia fotos da superfície de Marte [4]. Outro desses códigos capazes de lidar com MCUs em memórias é o código Matrix [5]. Esse código associa o código de Hamming com paridade, codificando uma palavra em formato matricial. O Reed-Muller é mais robusto que o Matrix e apresenta taxa de correção maior, no entanto, os custos de implementação do Matrix são muito menores.

Usando conceitos de Hamming Estendido e paridade, o Column Line Code (CLC) [6] é um código matricial que apresenta custo de implementação bem mais baixos que o Reed-Muller e taxas de correção maiores que o Matrix. Proposto em [7], o Matrix Region Section Code (MRSC) traz conceitos de paridade e intercalação, com objetivo de melhorar as taxas de correção aliadas ao custo. O Parity Hamming Interleaved Correction Code (PHICC) é um dos códigos matriciais mais recentes,

desenvolvido para ser um código de baixo custo computacional além de possuir uma grande capacidade de correção de erros, unindo paridade, Hamming e intercalação [8].

## 1.1 Objetivos

O objetivo desse trabalho é montar um quadro comparativo que sintetize o desempenho destes códigos corretores de erros, usando parâmetros como potência consumida, área ocupada e latência. Além disso, pretende-se comparar os códigos em relação à sua implementação (número de bits de redundância, número de portas lógicas na codificação) e quais técnicas são usadas (Hamming, paridade, intercalação, etc.).

### 1.1.1 Objetivos específicos

Compreender os efeitos que a exposição a ambientes hostis pode causar a dispositivos eletrônicos.

Utilizar ferramentas de simulação e desenvolvimento de sistemas digitais.

Implementar e avaliar os cógidos em VHDL e Python.

Testar aplicações com mecanismos de injeção de falhas e técnicas de análises de sistemas digitais.

Montagem dos quadros comparativos.

## 1.2 Justificativa

A radiação está presente em diversas formas e lugares, então dispositivos eletrônicos, mesmo em situações cotidianas, estão expostos a fatores que podem levá-los ao mau funcionamento. A crescente demanda por uma maior densidade e baixo consumo de potência, têm aumentado consideravelmente a sensibilidade à radiação [1].

Em aplicações mais críticas [4], onde o sistema de memória está mais suscetível a interferências provenientes do ambiente, como em aplicações aeroespaciais, a

ocorrência de MCUs é bastante comum.

O trabalho em [3] diz que a probabilidade de duas células vizinhas serem afetadas por um erro é dez vezes menor que uma única célula. Assim como um erro que afeta três células, é cem vezes menor que um erro único. No entanto, após teste com sistemas de memória usando interferência externa espalhadas aleatoriamente pelas células, [9] mostra que as células afetadas são, na maioria das vezes, fisicamente próximas ou até mesmo adjacentes. Estes trabalhos mostram a importância de proteger as memórias contra erros múltiplos.

Sabendo disso, e que os recursos de energia nessas aplicações são limitados, os códigos de correção e detecção de erros que vêm sendo propostos, se preocupam com parâmetros como área ocupada e potência consumida.

No entanto, outro importante parâmetro a ser analisado é o custo computacional de implementação, pois códigos mais robustos, embora protejam melhor que os mais simples, acabam por ser mais difíceis de implementar.

Com todas essas comparações organizadas, será possível analisar quais os pontos fortes de cada código e qual o mais indicado para diferentes tipos de sistemas embarcados.

CAPÍTULO 2

# TRABALHOS RELACIONADOS

O trabalho em [2], apresenta uma técnica para encontrar a proporção de MBUs (multiple bit upsets) em testes que medem os efeitos de single events. Os resultados dos experimentos comprovam que a quantidade de MBUs aumenta de acordo com a complexidade e a diminuição dos circuitos integrados.

[13] apresenta códigos de correção e detecção de erros que utilizam Hamming e Hamming Estendido e mostra suas taxas de correção e detecção para diferentes aplicações. O trabalho cita que os códigos apresentados podem ser usados na detecção de MCUs em SRAMs. [12] testou SRAMs para mostrar que single events upsetes podem também afetar múltplos bits, após ionizar sistemas de memórias em diferentes aplicações.

[3], [9] e [10] focam mais na necessidade de utilizar códigos de correção de erros em aplicações sujeitas à interferências causadas por fatores ambientais, como as aplicações aeroespaciais, onde o risco de falha pode ser catastrófico. [3] compara a aplicação do código Reed-Muller com códigos mais tradicionais e mostra que seu desempenho em relação à velocidade de processamento é melhor.

A abordagem de [5], cujo método apresentado foi chamado Matrix devido ao seu formato matricial, combina Hamming e paridade. O código é avaliado utilizando testes de injeção de falhas para calcular o MTTF e comparado com outros métodos anteriores em relação às taxas de correção e detecção. O método possui uma taxa de correção menor que o Reed-Muller, mas devido a robustez deste, os custos de implementação do Matrix se mostram mais vantajosos.

Em [11], os autores avaliam códigos de correção de erros em sistemas com alto grau de parelelismo, os NoCs (Network-on-Chip's) contra MCUs. Os experimentos envolvem três códigos e dois parâmetros de buffer diferentes. Ele apresenta uma metodologia boa de comparação entre códigos em aplicações mais específicas.

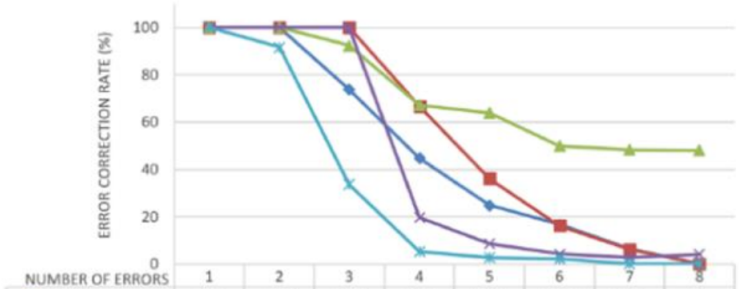
Após os trabalhos citados anteriormente, uma série de abordagens que utilizam

ECCs matriciais foram sendo apresentadas. [6] apresenta o Column Line Code (CLC), um código matricial que se apropria dos conceitos de paridade e Hamming Estendido. Os autores avaliam o código em relação a área ocupada, potência consumida e latência. Há também comparação entre os códigos para diferentes quantidades de bits de erro, apontando que o CLC corrige e detecta mais erros que os códigos com os quais foi comparado.

[7] apresenta o Matrix Region Section Code (MRSC), um código matricial que também se apropria dos conceitos de paridade, mas que usa intercalação. Os autores avaliam o código em relação a área ocupada, potência consumida e latência, concluindo num cálculo baseado nesses fatores e nas taxas de correção e detecção de erros, que o MRSC apresenta baixo custo de implementação se comparado às abordagens anteriores.

Assim como os artigos que apresentam o CLC e o MRSC, o trabalho em [8] apresenta o PHICC, Parity Hamming Interleaving Correction Code, que como o nome sugere, usa conceitos de paridade, Hamming e intercalação. Nesse trabalho, os autores também fazem comparações com outros códigos em relação a área, potência e latência, mas também se utiliza de cálculos mais complexos, como o MTTF e a confiabilidade.

A Figura 1 é um gráfico onde podem ser visto as diferenças entre alguns dos códigos citados anteriormente, quanto às taxas de correção de erros. Foram testados cenários de 1 a 8 erros para um total de 1 milhão de palavras pseudo-aleatórias injetadas nas implementações.




| NUMBER OF ERRORS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| MRSC | 100 | 100 | 73,74 | 44,7 | 24,81 | 16,79 | 6,34 | 0 |
| CLC | 100 | 100 | 100 | 66,84 | 36,05 | 16,21 | 6,25 | 0 |
| PHICC | 100 | 100 | 92,47 | 67,24 | 63,87 | 49,83 | 48,24 | 48,07 |
| Reed-Muller(2,5) | 100 | 100 | 100 | 19,74 | 8,62 | 4,21 | 2,65 | 4,12 |
| Matrix | 100 | 91,72 | 33,64 | 5,23 | 2,65 | 2,15 | 0 | 0 |

**Figura 1:** Taxa de correção de erros [8]

Na Figura 2, os autores comparam os parâmetros de área (medida em

micrometros) e potência (medida em miliwatts) consumidas, e atraso no tempo de execução (latência, dada em nanosegundos). O custo total é calculado usando as taxas de correção e detecção de erros inversamente proporcionais às informações de área, potencia e latência.

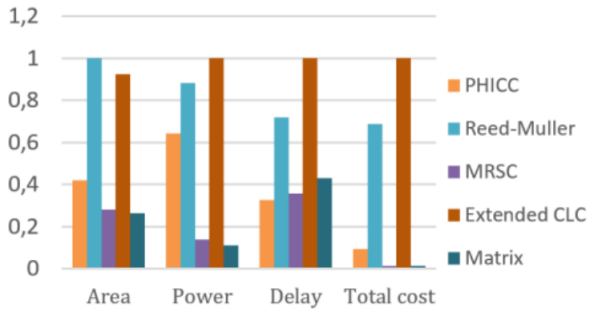



**Figura 2:** Parâmetros para medir custo total de implementação [8]

CAPÍTULO 3

# METODOLOGIA

O desenvolvimento deste trabalho deve ocorrer em cinco etapas: escolha e uso da ferramenta de simulação de sistemas digitais, implementação dos códigos, escolha da ferramenta de injeção de falhar, experimentação e montagem dos quadros comparativos.

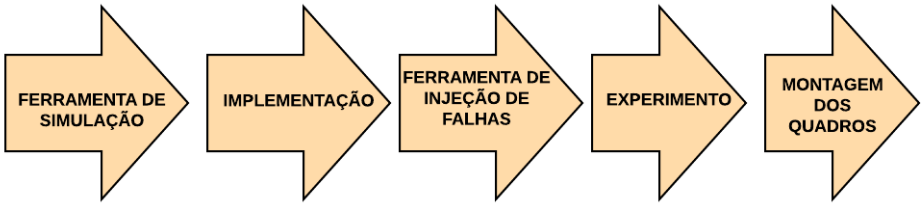



**Figura 3:** Etapas para o desenvolvimento do trabalho

Etapa 1: Escolha e uso da ferramenta de simulação de sistemas digitais.

Nesta primeira etapa, serão estudadas algumas ferramentas de simulação para sistemas digitais com o objetivo de escolher uma que possa contemplar a todos os parâmetros que se pretende utilizar.

Etapa 2: Implementação dos códigos.

Neste etapa, será feita dois de tipos de implementação dos códigos a serem comparados: a da matemática por trás dos códigos, que será feita em Python e/ou Matlab, e a implementação em VHDL, na ferramenta de simulação escolhida na etapa anterior. Neste segundo tipo de implementação já poderão ser calculados alguns parâmetros, como potência e area ocupada.

Etapa 3: Escolha da ferramenta de injeção de falhas.

Serão analisadas algumas ferramentas de injeção de falhas disponíveis do mercado, assim como as utilizadas pelos autores dos trabalhos estudados. Definida qual a ferramenta, podemos ir para a próxima etapa.

Etapa 4: Experimentos.

A etapa de experimentação é bastante importante, pois é onde podemos obter os dados das taxas de correção e detecção de falhas. A ferramenta escolhida, irá injetar milhares de falhas pseudo-aleatórias nos sistemas de memória, afim de que os códigos corretores de erro encontrem e corrijam estas falhas.

Etapa 5: Montagem dos quadros comparativos.

Esta é a última etapa do trabalho, quando já dispormos de todos os dados necessários para a comparação entre os códigos. Serão montados três quadros: desempenho, implementação e abordagem.