UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Augusto Gouvêa Weber

IMPLEMENTAÇÃO E AVALIAÇÃO DE DESEMPENHO DE UM CÓDIGO CORRETOR DE ERROS MULTI-BIT EM MEMÓRIAS SRAM

Santa Maria, RS
2022

Augusto Gouvêa Weber

IMPLEMENTAÇÃO E AVALIAÇÃO DE DESEMPENHO DE UM CÓDIGO CORRETOR DE ERROS MULTI-BIT EM MEMÓRIAS SRAM

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação, Área de Concentração em Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**. Defesa realizada por videoconferência.

ORIENTADOR: Prof. João Baptista dos Santos Martins

Santa Maria, RS
2022

This study was financied in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Finance Code 001

weber, Augusto Gouvea
IMPLEMENTAÇÃO E AVALIAÇÃO DE DESEMPENHO DE UM CÓDIGO CORRETOR DE ERROS MULTI-BIT EM MEMÓRIAS SRAM / Augusto Gouvea weber.- 2022.
77 f.; 30 cm

Orientador: João Baptista dos Santos Martins
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação , RS, 2022

1. ECC 2. MBU 3. SEE 4. Radiação 5. ASIC I. Martins, João Baptista dos Santos II. Título.

Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

Declaro, AUGUSTO GOUVEA WEBER, para os devidos fins e sob as penas da lei, que a pesquisa constante neste trabalho de conclusão de curso (Dissertação) foi por mim elaborada e que as informações necessárias objeto de consulta em literatura e outras fontes estão devidamente referenciadas. Declaro, ainda, que este trabalho ou parte dele não foi apresentado anteriormente para obtenção de qualquer outro grau acadêmico, estando ciente de que a inveracidade da presente declaração poderá resultar na anulação da titulação pela Universidade, entre outras consequências legais.

Augusto Gouvêa Weber

IMPLEMENTAÇÃO E AVALIAÇÃO DE DESEMPENHO DE UM CÓDIGO CORRETOR DE ERROS MULTI-BIT EM MEMÓRIAS SRAM

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação, Área de Concentração em Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**.

Aprovado em 18 de fevereiro de 2022:

_____

João Baptista dos Santos Martins, Dr. (UFSM)
(Presidente/Orientador)

_____

Cesar Augusto Prior, Dr. (UFSM)

_____

Eduardo Antonio César da Costa, Dr. (UCPel) (videoconferência)

Santa Maria, RS
2022

RESUMO

IMPLEMENTAÇÃO E AVALIAÇÃO DE DESEMPENHO DE UM CÓDIGO CORRETOR DE ERROS MULTI-BIT EM MEMÓRIAS SRAM

AUTOR: Augusto Gouvêa Weber
ORIENTADOR: João Baptista dos Santos Martins

O presente trabalho tem como objetivo apresentar a análise de desempenho e a implementação do algoritmo de ECC capaz de corrigir 4 erros de um único MBU. Para tanto foi desenvolvido seu código HDL, gerado vetores de testes para validação de seu funcionamento e implementação em FPGA e ASIC. Para sua implementação em FPGA optou-se por utilizar as famílias Spartan 3, Spartan 6 e Artix 7 da Xilinx por possuírem tecnologias de fabricação distintas, permitindo uma comparação entre elas. Para a implementação em ASIC realizou-se a síntese lógica utilizando o *design kit* XH018 da XFab e sua versão modificada tolerante a radiação (RH) da SMDH. A partir da escolha destes fizeram-se 3 sínteses: a de maior frequência sem modificações (XH018 Rápida), a de maior frequência tolerante a radiação (XH018 RH) e a sem modificação na mesma frequência que a versão modificada (XH018 Normal). Por fim realizaram-se análises de área, potência e frequência tanto dos ASICs como das FPGAs e posteriormente compararam-se os dados obtidos. Após as análises constatou-se que a Artix 7 mostrou-se uma ótima alternativa a circuitos que não utilizem RH por possuir uma frequência 17 % inferior.

Palavras-chave: MBU. SEE. ASIC. FPGA. Radiação. ECC. multi-bit. 4MBU-Rohde.

ABSTRACT

IMPLEMENTATION AND PERFORMANCE EVALUATION OF A ERROR CORRECTION CODE OF A MULTI-BIT SRAM MEMORY

AUTHOR: Augusto Gouvêa Weber
ADVISOR: João Baptista dos Santos Martins

The present work presents the performance analysis and the implementing of an ECC capable of correcting 4 bit-flip generated by a single MBU. For this, its HDL code was developed and generated the test vectors for validation of its operation and then implementation in FPGA and ASIC. For its implementation in FPGA, was used Xilinx's Spartan 3, Spartan 6 and Artix 7 families because they have different design technologies, allowing a comparison between them. For implementation in ASIC, the logic synthesis was performed using XFab's XH018 design kit and its modified radiation tolerant (RH) version of SMDH. With them, 3 syntheses were made, one with the highest frequency without modifications (XH018 Fast), one with the highest frequency radiation tolerant (XH018 RH) and the last without modification at the same frequency as the modified version (XH018 Normal), then it was performed area, power and frequency analysis between them and the FPGAs, concluding that if you are not going to make circuits using radiation tolerant transistors, it's possible to use Artix 7 because it has a 17 % lower frequency.

Keywords: MBU. SEE. ASIC. FPGA. Radiação. ECC. multi-bit. 4MBU-Rohde.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AEB	Agência Espacial Brasileira
$ASIC$	*Application-Specific Integrated Circuit*
CI	Circuito Integrado
CLB	*Configurable Logic Blocks*
$COTS$	*Commercial-Off-The-Shelf*
dk	*design kit*
DMA	*Direct Memory Access*
DSP	*Digital Signal Processing*
ECC	*Error Correcting Code*
ELT	*Edge-less Transistor*
ESA	*European Space Agency*
FIT	*Failure in time*
$FPGA$	*Field-Programmable Gate Array*
FSM	*Finite-State Machine*
HDL	*Hardware Description Language*
$INPE$	Instituto de Pesquisas Aeroespaciais
IP	*Intelectual Propriety*
MBU	*Multi Bit Upset*
PCB	*Printed Circuit Board*
PLL	*Phase-Locked Loop*
$RHBD$	*Radiation hardening By Design*
$RHBP$	*Radiation hardening By Process*
SBU	*Single Bit Upset*
SEE	*Single Event Effect*
SER	*Soft Error Rate*
SEU	*Single Event Upset*
$SMDH$	Santa Maria Design House

SoC	*System on a Chip*
SOI	*Silicon on Insulator*
SOS	*Silicon on Sapphire*
TID	*Total Ionizing Dose*
TMR	*Triple Modular Redundancy*

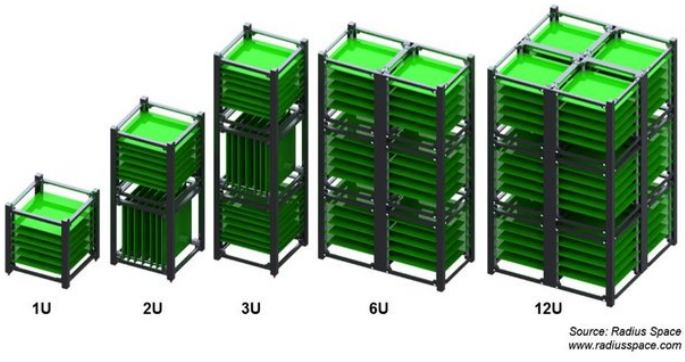
SUMÁRIO

1 INTRODUÇÃO

Durante o século XX a corrida pelo espaço foi um dos grandes desafios tecnológicos, sendo o primeiro satélite a orbitar a Terra o Sputnik 1, lançado em 1957 pela então União Soviética, desde então muitos outros lançamentos foram realizados, desbravando o espaço. Com isto novos desafios e avanços foram conquistados fazendo com que os satélites também fossem avançando conforme a finalidade requerida.

Dentre as necessidades advindas da chegada ao espaço encontravam-se desafios como os altos custos de produção de grandes satélites, a capacitação de mão de obra e a operação de circuitos em ambientes hostis. Buscando superar algumas dessas adversidades, em 1999, os professores Jordi Puig-Suari da California Polytechnic State University (Cal Poly) e Bob Twiggs da Stanford University's Space Systems Development Laboratory (SSDL), elaboraram um nanossatélite nomeado de CubeSat. Tal nome provém do acrônimo dos termos *Cube* (cubo em inglês) e da palavra satélite, já que o mesmo possui forma cúbica de 10 centímetros de arestas (1U), obedecendo o padrão descrito pelo CDS (*CubeSat Design Specification*). E também podendo ser combinadas gerando CubeSats de 2U, 3U, 6U ou 12U (Figura 1.1). Dentre as vantagens dos CubeSats encontram-se seu baixo custo e tempo de fabricação e a possibilidade de uso de componentes COTS (*Commercial Off-The-Shelf*) nos subsistemas de bordo (CENTRO DE GESTÃO E ESTUDOS ESTRATÉGICOS, 2018).

Figura 1.1 – Algumas configurações de *CuboSat*



Fonte: Adaptado de (LAHRICHI, 2017)

Dentre os subsistemas que compõe os nanossatélites podem-se mencionar os de comunicação, gerenciamento de energia, cargas úteis, sistema de altitude e computador de bordo (OBC). Este último é o subsistema principal responsável pelo gerenciamento dos demais subsistemas e armazenamento dos dados das cargas úteis. Nos nanossatélites os subsistemas são distribuídos nas PCBs e comunicam-se através de um barramento

principal, sendo cada um posicionado em uma PCB que é usualmente desenvolvida por diferentes instituições.

Após o lançamento de um satélite seus subsistemas serão expostos a um ambiente hostil (grandes variações de temperatura, altas doses de radiação, dificuldade de refrigeração, entre outros) (CENTRO DE GESTÃO E ESTUDOS ESTRATÉGICOS, 2018), sendo a radiação um dos principais desafios enfrentados pelos circuitos eletrônicos, pois esta pode causar *glitchs* de sinais nos circuitos lógicos e alteração nos *bits* de dados armazenados nas memórias, em especial nas memórias SRAM. Para mitigar os efeitos da radiação nos circuitos lógicos pode-se utilizar votadores na entrada dos registradores e nas memórias SRAM utilizam-se técnicas de redundâncias e ECCs (*Error Correcting Code*). Um caso recente onde pode ser observado a severidade que a memória SRAM de um sistema pode sofrer o efeito da radiação é o do Boeing 737 MAX, lançado no começo de 2019, no qual ocorreu um *bit-flip* no computador de bordo do piloto automático e, por não ter mitigação dos efeitos da radiação, acarretou na queda de dois aviões. Somente após a inserção de um segundo computador de bordo, foram liberados os voos com tais aeronaves (TRAVIS, 2019).

Problemas como o mencionado ocorrem pois as células de memória SRAM são as mais afetadas com o avanço dos processos de fabricação. Este avanço permite que mais transistores sejam inseridos em uma mesma área e que gera uma redução da tensão de operação. Assim, as células têm diminuído sua área, o que acarreta em um aumento da densidade das memórias, porém, a área de interação da radiação não reduz, com isso, múltiplas células são atingidas em um único evento, acarretando um aumento na quantidade de *bits* adjacentes alterados.

Em função disto, ECCs com maior capacidade de identificação e correção de erros vêm sendo utilizados. Assim, ECCs clássicos como o paridade simples e Hamming são ineficientes para correção de múltiplos erros em um dado. Sendo necessária a elaboração de técnicas voltados a correção deste tipo de erros adjacentes causados pela radiação.

Em trabalhos anteriores mostrou-se muito efetiva a técnica de correção de 4MBU de Rohde (ROHDE; MARTINS, 2020), onde a mesma tem como principal característica a correção de 4 erros adjacentes causados por uma radiação. Tal técnica foi desenvolvida e simulada em Matlab, sendo assim, neste trabalho visa-se realizar a primeira implementação da técnica em FPGA e ASIC, como proposto por Rohde (ROHDE, 2020).

Para tanto este trabalho divide-se em 5 capítulos. No primeiro capítulo apresentou-se de forma breve as motivações deste trabalho. Na sequência encontram-se uma revisão dos efeitos causados pela radiação nos circuitos CMOS, uma explanação das técnicas de ECC e uma explicação sobre o que torna um componente tolerante a radiação. Para o terceiro capítulo discorre-se sobre o processo de correção de erro da técnica 4MBU-Rohde, apresenta-se a metodologia utilizada para a criação dos códigos de descrição de *hardware*, o funcionamento dos seus blocos internos e a criação dos vetores de teste.

Por sua vez, os resultados estão dispostos no decorrer do capítulo 4. Por fim, no último capítulo, têm-se as análises e conclusões obtidas a partir dos resultados.

1.1 OBJETIVO GERAIS

Este trabalho tem como objetivo a realização da análise de desempenho do algoritmo de ECC 4MBU-Rohde através da simulação com vetores de testes de erros gerados de forma sistemática após a realização de sua implementação em linguagem de descrição de *hardware*, bem como sua implementação em FPGA e a realização da síntese lógica para um ASIC.

1.2 OBJETIVO ESPECÍFICO

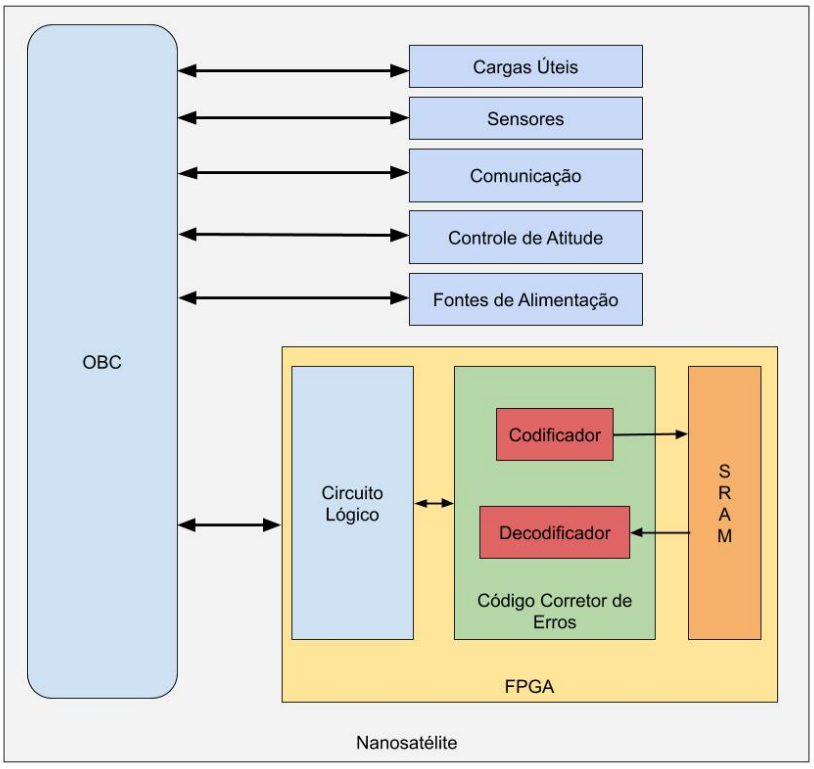
O presente trabalho visa primeiramente realizar a implementação do ECC 4MBU-Rohde em HDL. Em um segundo momento realizar a síntese em FPGA e a síntese lógica utilizando o *design kit* da XFab XH018 em 180 nm normal e a biblioteca de células tolerantes a radiação desenvolvida pela SMDH na mesma tecnologia.

Por fim, se realizará a avaliação de desempenho da implementação e uma comparação de área, potência e frequência de operação, de forma a considerar o impacto na performance do sistema.

2 REVISÃO TEÓRICA

O século XXI trouxe consigo uma grande evolução no desenvolvimento de satélites de pequeno porte. A categoria *CuboSat* possui muitas semelhanças com os demais satélites, apresentando diversos sistemas, como de energia, comunicação, controle de atitude, computador de bordo, e experimentos e/ou cargas funcionais. No entanto possuem a vantagem que seus sistemas encontram-se dispostos em módulos que se comunicam entre si através de um barramento padrão, permitindo a utilização de módulos desenvolvidos para outras missões, como observado na Figura 2.1.

Figura 2.1 – Subsistemas de um nanossatélite com ECC



Fonte: Adaptado de (ROHDE, 2020)

Outro diferencial na elaboração destes satélites é a utilização de COTS em sua produção, gerando custos inferiores aos de satélites convencionais por serem componentes produzidos em grande escala para diversas aplicações. No entanto a utilização de COTS trouxe consigo seus próprios desafios, tais como a baixa tolerância a radiação, TID (*Total Ionizing Dose*), e susceptibilidade a SEEs (*Single Event Effect*) que culminam em um tempo de missão limitado comparado a satélites com componentes resistentes a radiação.

Se, por um lado, a utilização de COTS é um desafio, por outro, a escolha por este tipo de componente é realizada devido a uma maior variedade e disponibilidade de compo-

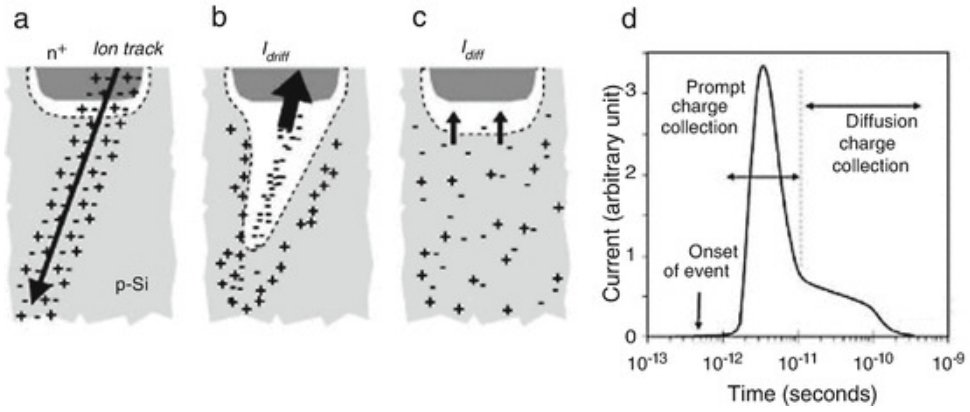
nentes, acarretando em um menor tempo de aquisição e redução considerável dos custos. Contudo, é necessário levar em consideração os efeitos que a radiação podem causar nos circuitos CMOS.

2.1 EFEITOS DA RADIAÇÃO

A radiação pode infringir diversos efeitos a curto e longo prazo nos circuitos tanto digitais, quanto analógicos. Como efeito de longo prazo tem-se a TID (*Total Ionizing Dose*), que é o acúmulo de carga nos *gates* dos transistores. Ela afeta o funcionamento tanto dos circuitos analógicos, causando o incorreto funcionamento de sensores, fontes de tensão, fontes de corrente e outros; como os circuitos digitais que podem vir a causar falha de funcionamento nos transistores deixando a porta lógica com saída fixa em "1" ou "0" (JOHNSTON, 1998).

Já os efeitos de curto prazo são conhecidos como SEE (*Single Event Effect*) e são resultados da interação de uma partícula carregada (íon) com os átomos do transistor (Figura 2.2). Essa reação gera uma deposição de carga responsável por *Hard Errors* (eventos destrutivos) ou *Soft Errors* (eventos não destrutíveis).

Figura 2.2 – Deposição de cargas em um transistor devido a um SEE



Fonte: Adaptado de (PRINZIE; STEYAERT; LEROUX, 2018)

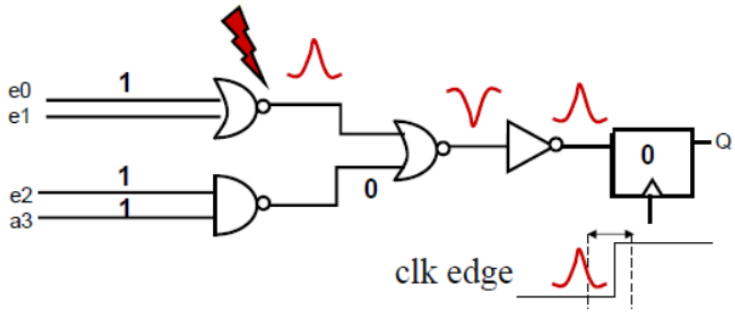
O *Hard Error* leva a inoperabilidade do circuito causado por *Latchup*, efeito que gera uma passagem excessiva de corrente pelo circuito, sobrecarregando e queimando o mesmo ou o componente. Se o mesmo não for prevenido, pode levar a interrupção do funcionamento do sistema.

O *Soft Error* é dividido em eventos distintos dependendo de qual circuito é atingido, podendo ser SETs (*Single Event Transient*) e SEUs (*Single Event Upset*), este dividido em SBUs (*Single Bit Upset*) e MBUs (*Multi Bit Upset*).

2.1.1 Single Event Transient (SET)

Caso a radiação atinja uma porta lógica o SEE pode ocasionar um SET, também conhecido como *glitch*, que é a propagação de um sinal incorreto através de uma sequência de portas lógicas (Figura 2.3). Se esta transição indesejada chegar a um registrador junto a um sinal de relógio pode causar o efeito de metaestabilidade ou armazenar o valor incorreto (AZAMBUJA; KASTENSMIDT; BECKER, 2014).

Figura 2.3 – Exemplo de SET causado por um SEE



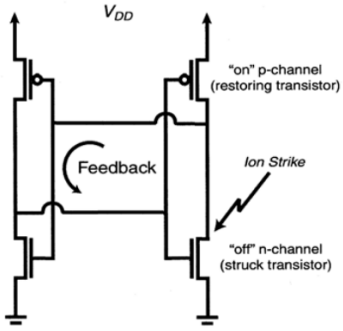
Fonte: Adaptado de (AZAMBUJA; KASTENSMIDT; BECKER, 2014)

2.1.2 Single Event Upset (SEU)

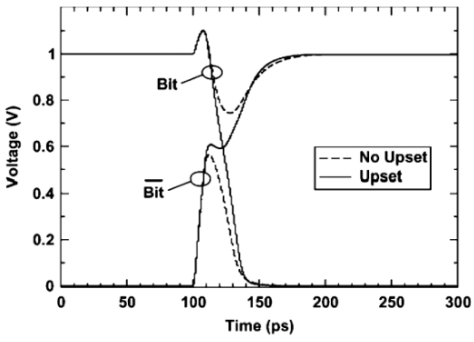
Quando um SEE atinge um circuito de memória (*flip-flop*, registradores, células de memórias SRAMs ou DRAMs) e ocasionam um *bit-flip* (SEU), isto é, a mudança do dado armazenado de "1" para "0" ou de "0" para "1" (Figura 2.4). Quando apenas uma mudança de *bit* é ocasionada em uma célula de memória, esse evento chama-se SBU, todavia, quando duas ou mais células de memória são afetadas por um único SEE e ocorrem *bit-flips*, nomeia-se MBU (JOHNSTON, 1998).

Figura 2.4 – *Bit-flip* Memória SRAM

(a) circuito



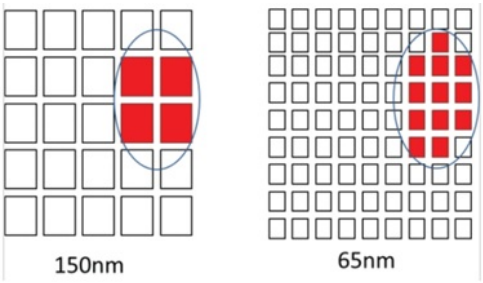
(b) SEE



Fonte: Adaptado de (QUISPE, 2014)

Os MBUs têm aumentado devido ao melhoramento dos processos de fabricação dos transistores que se tornaram cada vez menores, permitindo assim que fossem colocados mais em uma mesma área (lei de Moore). Devido a isso, circuitos mais complexos puderam ser elaborados e integrados em chips cada vez menores. Não obstante, a área de interação entre as partículas radioativas e o circuito se manteve constante, causando um aumento na quantidade MBUs (Figura 2.5).

Figura 2.5 – Área de iteração da radiação em memórias



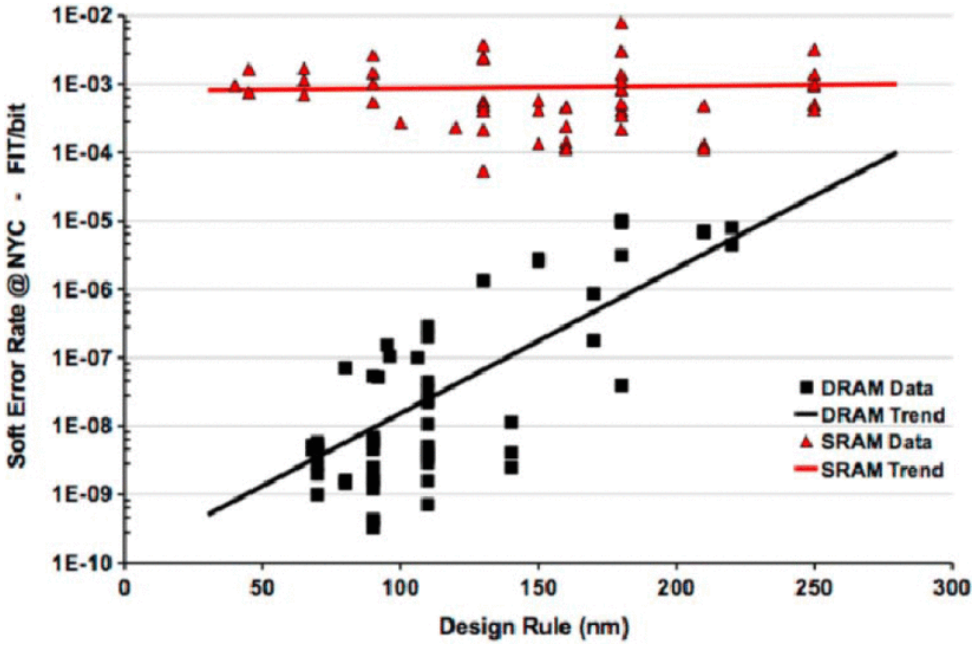
Fonte: Adaptação de (Amagasaki et al., 2016)

2.1.3 Failure In Time (FIT)

O FIT(*Failure In Time*) é definido como 1 evento a cada 10^9 horas ao nível do mar de Nova Iorque. Ele é utilizado como métrica do tempo necessário para a ocorrência de *bit-flip* para uma célula de memória.

Como observa-se na Figura 2.6 quase não há variação de SER (*Soft Error Rate*) para memórias SRAM com *design* de $250nm$ a $50nm$, indicando que a carga crítica (Q_{crit}), carga das partículas necessária para que ocorra um SEU, está diminuindo e, como consequência, aumentando assim o SER. Pode-se afirmar que a medida que o tamanho da célula de memória diminui a área para SER diminui. Assim, combinados os dois resultados os efeitos serão cancelados.

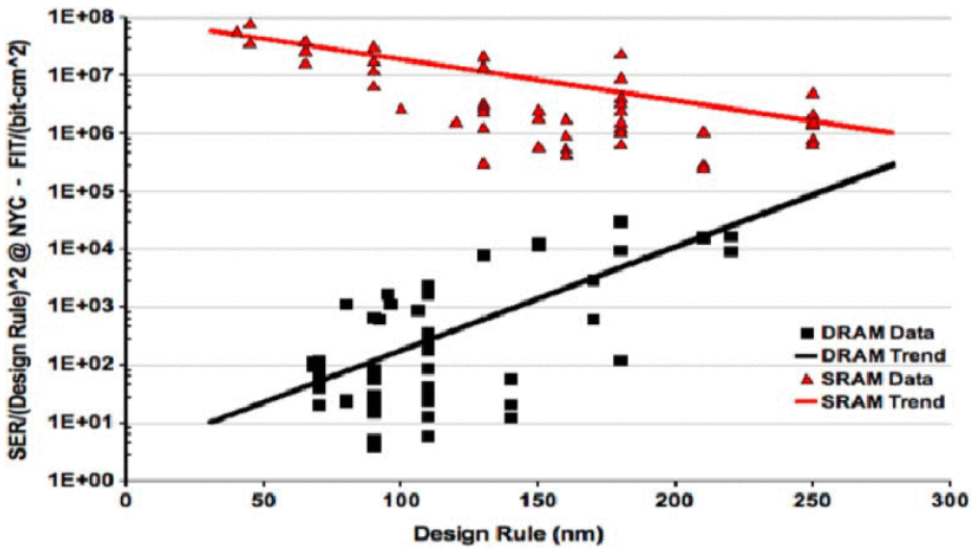
Figura 2.6 – SER x Design



Fonte: Adaptado de (SLAYMAN, 2011)

Deste modo, ao considerar-se que o aumento da densidade da memória SRAM é $design(nm)^{-2}$, obtém-se a anulação da redução da área de ação do SER. Tal redução é responsável por gerar um aumento do SER à medida que a tecnologia do processo diminui (Figura 2.7) (SLAYMAN, 2011).

Figura 2.7 – SER x $Design(nm)^{-2}$



Fonte: Adaptado de (SLAYMAN, 2011)

Ainda com relação ao FIT, como observado na Figura 2.6 (*SRAM Trend*), o mesmo aumenta à medida que o fluxo de radiação que passa pela memória aumenta ou que as

partículas possuam mais energia. Uma forma de aumentar o FIT é através da mitigação dos SEUs, para isto utilizam-se técnicas de correção de erro (ECC), tais como: TMR (*Triple Modular Redundancy*), paridade simples, código de Hamming, *Matrix*, *Checksum* e 4MBU-Rohde, que serão apresentadas a seguir.

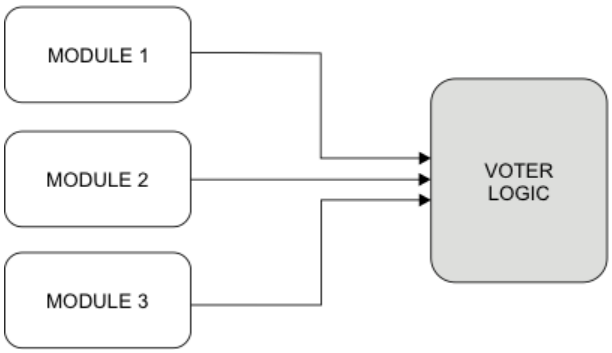
2.2 ERROR CORRECTING CODE (ECC)

Os ECCs são técnicas capazes de identificar os erros nos *bits* do dado e corrigi-los. Para isto, realizam-se operações lógicas para geração de *bits* extras (*bits* de paridade) a serem armazenados junto ao dado e que posteriormente serão utilizados para verificar sua integridade. A seguir serão apresentado os ECCs mais implementados e seus funcionamentos.

2.2.1 Triple Modular Redundancy (TMR)

O TMR é uma técnica onde o dado é triplicado e armazenado em três partes da memória ou em três memórias. Quando acontece a leitura ele é obtido destas memórias e passa por um votador onde o valor a ser transmitido é a maioria entre os lidos, fazendo com que seja necessária a alteração de dois dados ou mais para que a informação resultante seja adulterada. Esta característica traz ganhos de segurança e integridade, todavia é necessário três vezes mais memória para armazenar o dado (SINGH; PASHAIE; KUMAR, 2015).

Figura 2.8 – Exemplo de TMR



Fonte: Adaptado de (SINGH; PASHAIE; KUMAR, 2015)

2.2.2 Paridade Simples

A técnica de paridade simples, por sua vez, consiste em verificar a quantidade de "1" que existe em uma sequência de dado. Se for par o dado de paridade é "0", caso contrário será "1". Por exemplo, caso seja armazenado um dado de 8 *bits* 00111011 e deseja-se ter a paridade simples dele, então realiza-se a operação XOR entre os bits e obtém-se o valor "1"(Equação 2.1).

$$P(00111011) = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1 \tag{2.1}$$

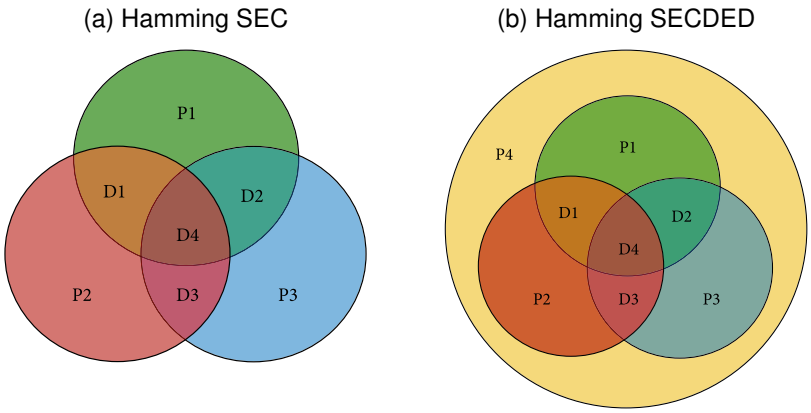
Quando é realizado o processo de leitura do dado repete-se a operação de geração da paridade, fazendo-se XOR da paridade lida e do dado. Em caso do resultado ser "0" não existe erro na informação e em caso do resultado ser "1" informa que ocorreu a alteração de 1 *bit*. Contudo, caso aconteça a mudança de valor par de *bits*, o resultado será "0", não indicando a presença de erro (Ravi; Pargunarajan, 2017).

2.2.3 Hamming

A técnica de Hamming consiste na geração de bits de paridade onde o resultado da mesma é a posição do erro, sendo assim, quando não existe erro o resultado da paridade são todos os bits "0". Esta técnica é SEC (*Single Error Correction*) quando é capaz de corrigir somente um *bit flip* de dado e paridade, já a SEC-DED (*Single Error Correction, Double Error Detection*) ou Hamming Estendido é capaz de corrigir um *bit flip* e identificar caso tenham ocorrido dois *bit flips*, para isto a técnica de paridade simples é integrada, aumentando em um *bit* o tamanho da paridade (MILIES, 2009).

A notação para a técnica SEC é *Hamming* (k, m) e SEC-DED é *Hamming-E* (k, m) onde m é a quantidade de *bits* de dado e k é a quantidade total de *bits* de (paridade + dado). A seguir, tem-se *Hamming(7,4*) na Figura 2.9a e *Hamming-E(8,4)* na Figura 2.9b.

Figura 2.9 – Diagramas de Hamming



Fonte: Adaptado de (HILLIER; BALYAN, 2019)

Para a geração dos bits de paridade faz-se operações de XOR entre os bits de dado, assim para o Hamming(7,4) temos:

$$\begin{aligned} P1 &= D1 \oplus D2 \oplus D4 \\ P2 &= D1 \oplus D3 \oplus D4 \\ P3 &= D2 \oplus D3 \oplus D4 \end{aligned} \tag{2.2}$$

Para a decodificação faz-se as seguintes operações:

$$\begin{aligned} S1 &= P1 \oplus D1 \oplus D2 \oplus D4 \\ S2 &= P2 \oplus D1 \oplus D3 \oplus D4 \\ S3 &= P3 \oplus D2 \oplus D3 \oplus D4 \end{aligned} \tag{2.3}$$

Com o Hamming Estendido é semelhante. Para o Hamming-E(8,4) têm-se as seguintes operações para geração dos *bits* de paridade:

$$\begin{aligned} P1 &= D1 \oplus D2 \oplus D4 \\ P2 &= D1 \oplus D3 \oplus D4 \\ P3 &= D2 \oplus D3 \oplus D4 \\ P4 &= D1 \oplus D2 \oplus D3 \oplus D4 \oplus P1 \oplus P2 \oplus P3 \end{aligned} \tag{2.4}$$

E para a decodificação faz-se as seguintes operações:

$$
\begin{aligned}
S1 &= P1 \oplus D1 \oplus D2 \oplus D4 \\
S2 &= P2 \oplus D1 \oplus D3 \oplus D4 \\
S3 &= P3 \oplus D2 \oplus D3 \oplus D4 \\
S4 &= D1 \oplus D2 \oplus D3 \oplus D4 \oplus P1 \oplus P2 \oplus P3 \oplus P4
\end{aligned}
\tag{2.5}
$$

Assim, com S1, S2 e S3 tem-se a posição em que ocorreu o erro caso sejam diferentes de "0" e com S4 a indicação se ocorreram 2 erros caso S1, S2 e S3 não sejam "0"e S4 seja igual a P4.

Deste modo pode-se concluir, que a técnica de Hamming tem como principais vantagens a velocidade de execução, simplicidade por ter poucas portas lógicas em seu caminho crítico e eficiência, pois, quanto maior o dado menor é a relação $(paridade + dado)/dado$.

2.2.4 Matrix

Outra técnica é a *Matrix* que consiste na combinação da técnica de paridade simples e da técnica de Hamming. Nessa, o dado é disposto em uma matriz onde tem-se o código de Hamming para cada linha da matriz e paridade simples para cada coluna da matriz (SILVA, 2018) (Figura 2.10).

Figura 2.10 – Código Matrix

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

Fonte: Adaptado de (Argyrides; Zarandi; Pradhan, 2007)

Assim sendo, é possível que ocorra um erro por linha, visto que a técnica de Hamming será capaz de corrigi-los. Caso ocorram 3 erros em uma linha da matriz pode-se detectar que houve uma incorreta correção através da paridade simples das colunas que será diferente.

2.2.5 Checksum

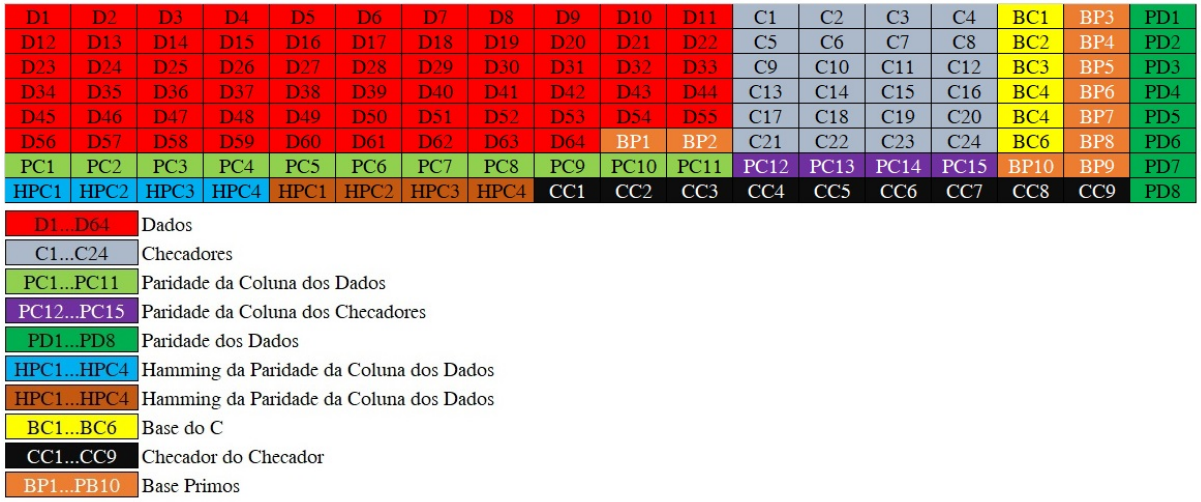
O *Checksum* ou soma de verificação é um código utilizado para verificar a integridade do dado através de soma de valores de acordo com a posição do dado e seu valor. Por exemplo, caso tenha-se o dado 00111011 e o mapa de valores $\{1, 2, 3, 4, 5, 6, 7, 8\}$ obtém-se o C_s com valor igual a 27 (Equação 2.6). Assim, se ao realizar a leitura do dado e a geração do *Checksum* for 30, diferente do armazenado, pode-se constatar que o mesmo está incorreto e não há como recuperá-lo.

$$C_s = (0 * 1) + (0 * 2) + (1 * 3) + (1 * 4) + (1 * 5) + (0 * 6) + (1 * 7) + (1 * 8) = 27 \quad (2.6)$$

2.2.6 4MBU-Rohde

A técnica 4MBU-Rohde foi desenvolvida recentemente com a corregibilidade de 100% para 4 erros quaisquer em uma área de 3x3 células de memória SRAM. Ela consiste na combinação das técnicas citadas anteriormente para um conjunto de 64 *bits* de dado (Figura 2.11) (ROHDE, 2020).

Figura 2.11 – Matriz 4MBU-Rohde



Fonte: Adaptado de (ROHDE, 2020)

Faz-se necessária a menção de que a técnica tem a numeração de seus vetores de elementos começando em 1, onde o dado tem seu índice inicial em 1 e final em 64. Todavia neste trabalho optou-se pela iniciação da contagem de seus vetores em 0, devido sua ampla utilização nas linguagens de descrição de *hardware*, assim o índice inicial do

dado torna-se 0 e o final 63.

Para gerar a paridade inicialmente distribui-se o dado em uma matriz 6 x 11 e aplica-se a técnica *Matrix*, onde para cada linha gera-se o Hamming-E (16,11) e para cada coluna aplica-se a técnica de paridade simples.

A paridade da linha gerada pela *Matrix* é chamada de **C** e o *bit* extra é nomeado de **PD**. O primeiro é composto por 24 bits de paridade distribuídos em uma matriz 6x4 e o segundo é composto por 6 *bits*. Já a paridade da coluna gerada pela *Matrix* é chamada de **PC**, todavia não limita-se apenas aos *bits* dos dados, estendendo-se para o **C** também.

A partir da geração do **C** faz-se: o **BC**, que é a paridade simples das linhas do **C**; o **CC**, que é o *Checksum* dos *bits* do **C**, onde cada posição tem seu valor respectivo a sua posição; e o **BP**, que é semelhante ao **CC**, porém utiliza a seguinte sequência de números primos: 2, 89, 3, 83, 5, 79, 7, 73, 11, 71, 13, 67, 17, 61, 19, 59, 23, 53, 29, 47, 31, 43, 37 e 41.

Com o **PC**, referente as colunas da matriz do dado (colunas de 0 a 10), gera-se o **HPC**, sendo ele os quatro bits de paridade para Hamming(15,11). Então cria-se uma cópia de **HPC** para ser armazenado na memória. Assim conclui-se a geração dos 80 *bits* de paridade da técnica 4MBU-Rohde (ROHDE; MARTINS, 2020).

No momento em que a matriz (8x18) de dado e paridade é armazenado na memória, o mesmo é embaralhado de forma que certas posições fiquem afastadas entre si, garantindo assim uma maior cobertura de correção. Um exemplo é o **PC**(0 a 10) e ambos **HPCs**, onde o **PC** é afastado dos **HPC**, assim, caso ocorra um erro **PC** realize-se a técnica de Hamming para corrigi-lo ou caso o erro ocorra em **HPC** (detectado pela diferença entre as cópias de **HPC**) infere-se que **PC** está correto. Este padrão de embaralhamento (*interleaving*) foi elaborado em trabalhos anteriores, onde teve o aumento na eficiência do ECC comprovado (Figura 2.12).

Figura 2.12 – Dado com *interleaving* (embaralhamento)

D28	PC1	C24	C5	PC4	D27	D35	HPC3	HPC4	BC1	BC2	BC3	HPC1	HPC2	HPC3	CC1	BP3	PD1
C6	D2	C19	C10	D58	D15	D34	HPC1	HPC2	BC4	BC5	BC6	D3	D16	HPC4	CC2	BP4	PD2
C11	D48	C14	C20	D17	D51	D52	D31	D43	D33	D53	D21	D11	D59	CC9	CC3	BP5	PD3
C16	PC2	C9	C15	PC5	D32	D55	D36	D26	D49	D1	PC9	D24	D39	D29	CC4	BP6	PD4
C21	D5	C18	C23	D47	D4	D38	D61	D18	D8	C1	D7	D41	D9	PC7	CC5	BP7	PD5
C2	D30	C7	C12	D6	D62	D19	D42	D54	D22	D25	D37	D60	D14	D56	CC6	BP8	PD6
C4	D40	C13	C3	D64	D10	D44	PC8	D23	D57	PC11	D45	D13	D50	BP2	CC7	BP9	PD7
C22	PC3	C17	C8	PC6	D12	D46	PC12	PC13	PC14	PC15	D63	D20	BP1	PC10	CC8	BP10	PD8

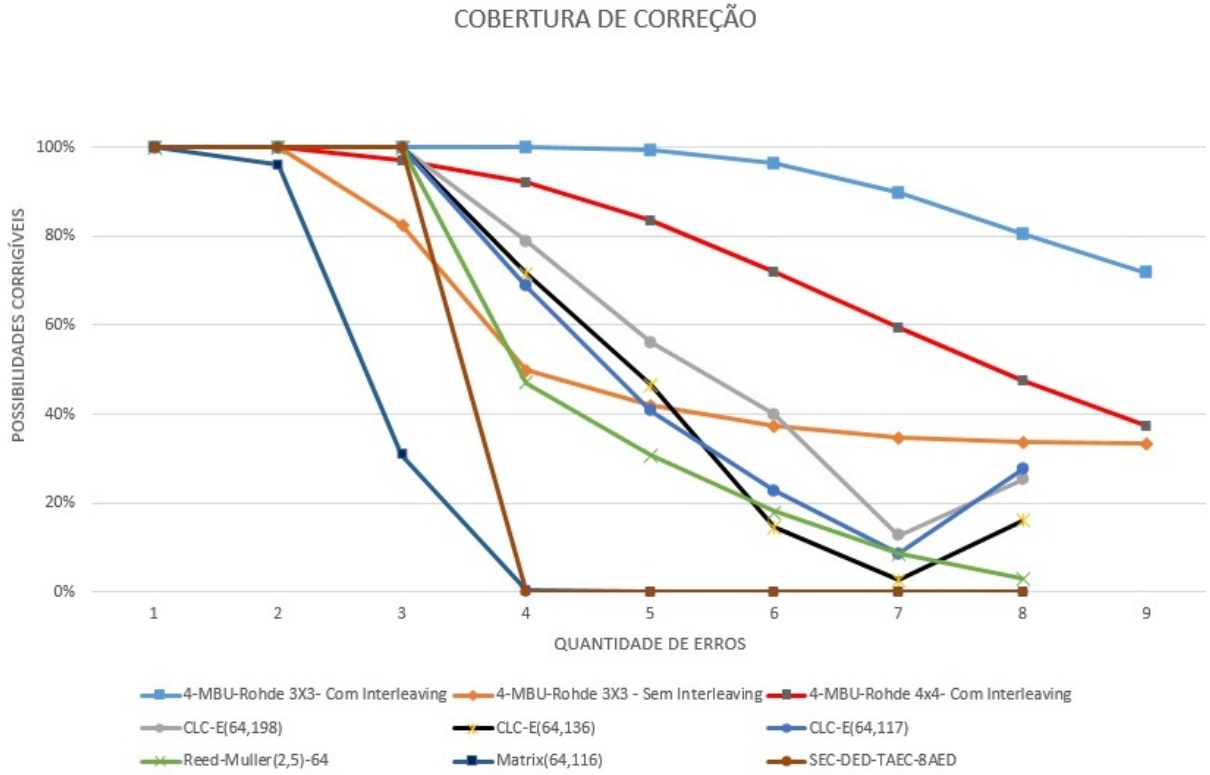
Fonte: Adaptado de (ROHDE, 2020)

Outro exemplo da melhora do ECC com o *interleaving* pode ser apresentado com as posições do dado, deixando afastadas as linhas e colunas para que a técnica de Hamming-E possa corrigi-las.Desta forma o algoritmo de correção aumenta sua cobertura em 100 %, de 2 *bits* quaisquer para 4 *bits* adjacentes.

2.2.7 Comparação da cobertura dos ECCs

Ao realizar a comparação da técnica 4MBU Rohde com outras técnicas já existentes, pode-se observar uma cobertura superior as demais a partir de 4 erros adjacentes (Figura 2.13).

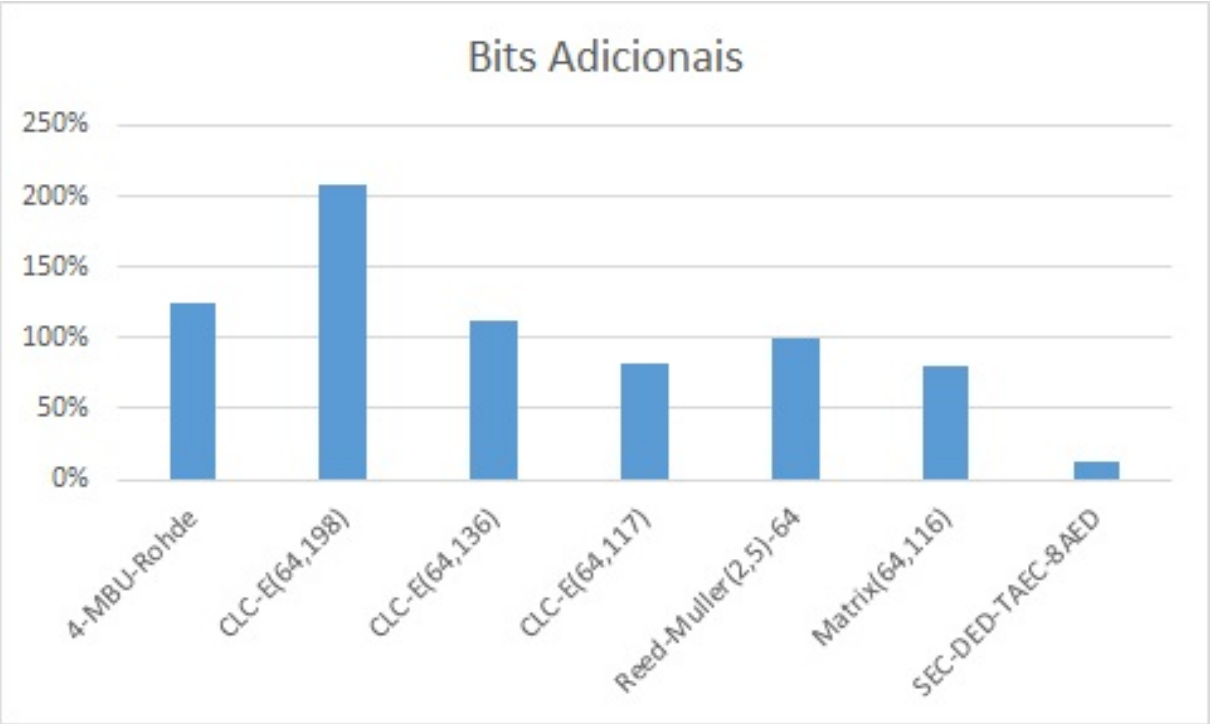
Figura 2.13 – Comparação da cobertura dos ECCs



Fonte: (ROHDE, 2020)

Também observa-se que a quantidade percentual de bits de paridade em relação ao tamanho do dado (Figura 2.14) para a técnica de Rohde é semelhante às demais. Assim sendo, tem-se uma técnica que apresenta uma ótima relação de cobertura de correção e quantidade de bits de paridade.

Figura 2.14 – Comparação do tamanho da paridade dos ECCs



Fonte: (ROHDE, 2020)

Assim, neste trabalho optou-se pela implementação da técnica 4MBU-Rohde devido a que a mesma é capaz de corrigir 4 erros adjacentes, sendo esta a primeira implementação em FPGA e ASIC.

2.3 ASICS E FPGAS

Os ECCs apresentados anteriormentes são implementados de forma integrada em ASICs, FPGAs ou memórias. Os ECCs são usualmente posicionados entre o processador e as memórias, propiciando a correção e proteção de todo dado que seja transmitido entre estes elementos e, assim, aumentando o FIT.

Os ASICs (*Application-Specific Integrated Circuit*), como o nome indica, são circuitos criados com propósito específico, foram muito populares no início da era do silício, onde poucos transistores cabiam em um chip, assim era necessário criar grandes PCBs com diversos CIs (Circuitos Integrados) para a criação do circuito. Por tratar-se de circuitos com baixa complexidade, se comparados com os atuais, seu tempo de elaboração e testes era inferior e a motivação para sua criação era importante, pois a utilização de componentes discretos tinha muitas limitações.

As FPGAs, por sua vez, surgiram com a miniaturização dos circuitos eletrônicos, permitindo que cada vez mais lógica fosse inserida em uma mesma área de silício, possibi-

litando que circuitos mais complexos surgissem. Elas são um conjunto de circuitos lógicos com função específica em uma rede de interconexões reconfiguráveis. Nelas existem:

- CLBs (*Configurable Logic Blocks*), que contém LUTs (*lookup tables*), que ao invés de ter portas lógicas, consiste em uma memória com valor de saída predeterminada para dado endereço na entrada, simulando assim uma função lógica ou elementos de armazenamento (*latchs* e *flip-flops*);
- Blocos de entrada e saída, que são responsáveis pela administração dos pinos, que, assim como em microcontroladores, podem ter múltiplas funções;
- Blocos de DSP, nele existem principalmente multiplicadores, uma operação muito utilizada em processamento de sinais;
- Controle de sinal de relógio, sendo responsável por gerenciar o sinal de relógio para diferentes frequências e fases através de PLL ou outro circuito;
- E blocos de memória RAM, entre outros.

Este avanço permitiu que fossem criados componentes com lógica reconfigurável, as FPGAs permitem a criação de circuitos lógicos de forma mais rápidas e com menor custo, visto que não era mais necessária a criação de um ASIC ou uma PCB grande para tais funções.

Além de possuírem diversos blocos já implementados, as FPGAs permitem que as conexões entre seus componentes ocorram de forma programada que possibilita a criação de diversos circuitos capazes de executar operações muito mais rápidas que processadores e *software*.

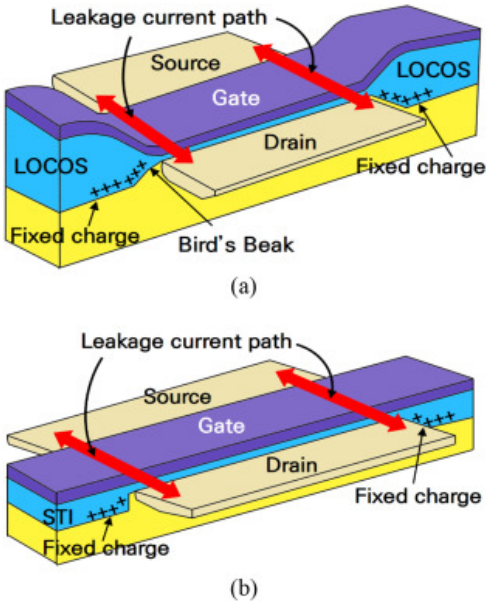
Ao migrar o projeto do FPGA para o ASIC permite-se que otimizações sejam feitas, fazendo com que o circuito possa operar em menos ciclos ou em frequências mais elevadas e com um consumo energético inferior. Contudo, para tal, faz-se necessário a elaboração de todo o circuito, demandando mais tempo, profissionais especializados para trabalhar com ASIC, testes mais sofisticados que culminam em custos muito mais elevados se comparado com FPGAs e microcontroladores.

2.3.1 Mitigando a radiação nos CMOS

Com relação as suas tolerâncias, tanto as FPGAs como os ASICs podem ser tolerantes a radiação, resistentes ou não testados. Os circuitos tolerantes são aqueles que têm seu funcionamento até uma TID de aproximadamente $30krad(Si)$, já os resistentes são desenvolvidos utilizando duas metodologias principais a RHBP (*Radiation Hardening By Process*) e a RHBD (*Radiation Hardening By Design*).

A RHBP divide-se em duas técnicas, a primeira, consiste em utilizar variações no processo de fabricação do chip para inserir tecnologias de mitigação dos efeitos de radiação, como o SOI (*Silicon on Insulator*) ou SOS (*Silicon on Sapphire*), onde o substrato dos transistores é isolado, evitando assim o acúmulo de cargas e tornando os circuitos tolerantes a *Latchup* e a TID de 100 a 300 krad, porém esses processos não garantem uma redução na quantidade de SEEs. Por sua vez, a segunda, decorre da implementação do bico de pássaro (*bird's Beak*), onde as cargas dos *gates* são deslocadas evitando, deste modo, a criação do canal por carga acumulada (LEE et al., 2018) (Figura 2.15).

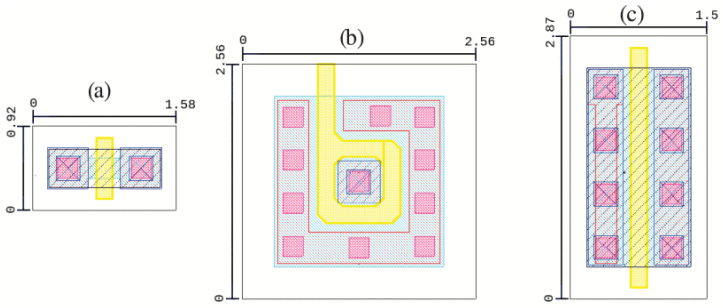
Figura 2.15 – Comparação dos transistores RHBP



Fonte: Adaptado de (LEE et al., 2018)

A RHBD permite a utilização dos processos de fabricação convencional através da utilização de ELT (*Enclosed Layout Transistors*) (Figura 2.16) que não contém cantos, evitando assim o acúmulo de cargas, desta forma, pode-se enviar o circuito para fabricação, sendo necessário apenas a implementação destes transistores no *design kit* para sua utilização.

Figura 2.16 – Transistor ELT a) Menor tamanho transistor padrão. b) Menor transistor ELT. c) Menor transistor padrão equivalente ao menor ELT.



Fonte: Adaptado de (VAZ et al., 2021)

3 IMPLEMENTAÇÃO DO ECC 4MBU-RHODE

No presente capítulo tem-se por objetivo implementar o ECC 4MBU-Rohde que havia sido validado em MATLAB. Para tanto, realiza-se a implementação utilizando FPGA e a síntese lógica utilizando os *design kits* XFab180 e XFab180 RHBD. Posterior a implementação apresenta-se a análise dos resultados para averiguar o desempenho da implementação, bem como a comparação de área, potência e frequência de forma a considerar o impacto na performance do sistema.

Para dar início ao trabalho faz-se necessário realizar a conversão para HDL (*Hardware Description Language*), para isto analisam-se e criam-se fluxogramas para uma representação da sequência de operações necessárias para a codificação e decodificação do ECC.

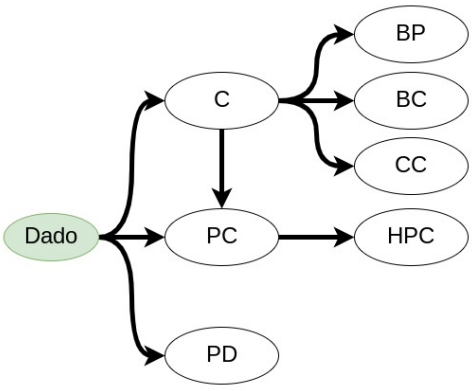
3.1 FLUXOS DE DADOS NO ECC

A seguir, serão apresentados os fluxogramas do codificador e decodificador, mostrando passo a passo as tarefas e decisões para codificar o dado e solucionar SEU.

3.1.1 Codificador

O codificador funciona a partir da entrada dos dados que através de ECCs geram as paridades **C**, **PC**, **PD**, **BC**, **HPC**, **CC** e **BP** (Seção 2.2.6), que em conjunto com os dados são salvos na memória após a passagem pelo *interleaving* (Figura 3.1).

Figura 3.1 – Fluxograma para gerar a paridade e armazenar na memória



Fonte: Autoria própria

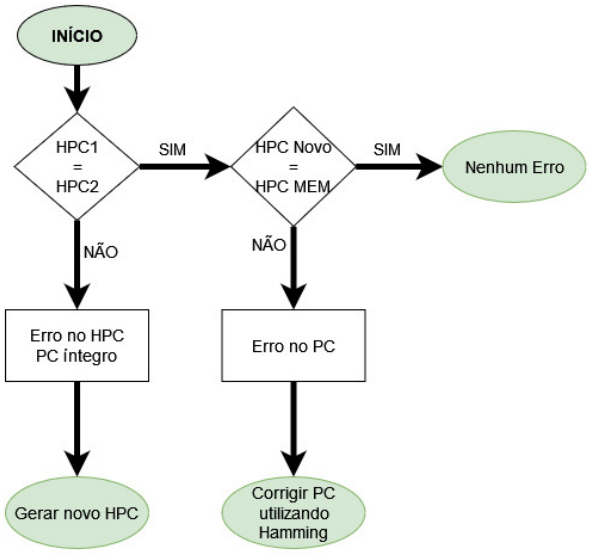
3.1.2 Decodificador

A operação de obtenção do dado consiste em obter o dado e a paridade da memória, desembaralha-los (desfazer o *interleaving*), passa-los pelo decodificador que corrige-os e valida o dado. Após sua validação, caso não haja erro, o dado é enviado para a CPU. Caso tenha algum erro, codifica-se o dado validado, aplica-se o *interleaving*, armazena-o novamente na memória e o dado é enviado Para a CPU. A seguir se apresentarão todos os passos para a verificação e correção dos erros.

Para realizar a decodificação do dado geram-se novos bits de paridade com o dado obtido da memória. Então comparam-se ambas as paridades, caso não sejam iguais começa-se o processo de correção. Este processo permite que se faça a verificação do **PC** e do **C** de forma simultânea.

Para verificação de **PC**, faz-se necessária a análise onde ambos HPCs devem ser iguais, isto indica a ocorrência de erro em **HPC**. Caso não sejam iguais, sabe-se que ocorreu um erro em **HPC** e não em **PC**. Neste caso, por encontrarem-se afastados na memória gera-se um novo **HPC** para ser salvo, substituindo o seu anterior o qual apresentava erro. Caso sejam iguais, compara-se ao gerado e, sendo iguais constata-se a inexistência de erro no **PC** (0-10). Todavia, caso sejam diferentes usa-se Hamming para corrigir o **PC** usando o **HPC** da memória (Figura 3.2).

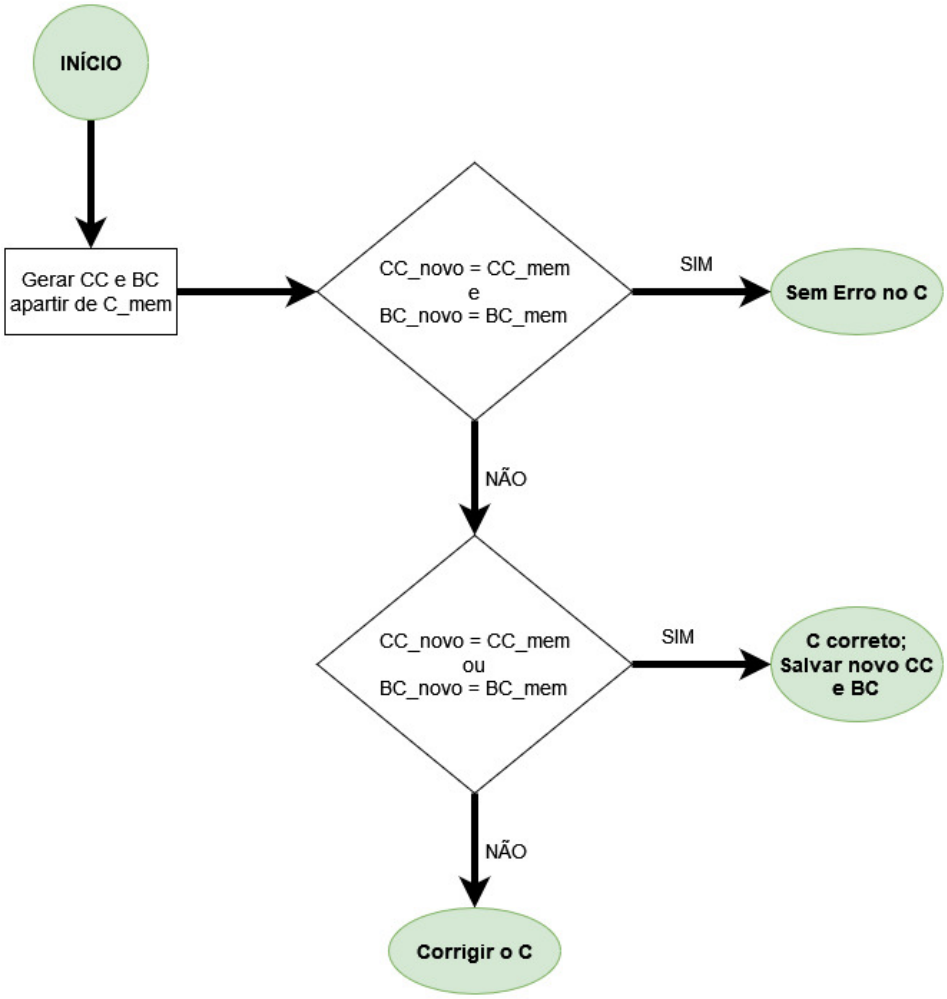
Figura 3.2 – Fluxograma Erro PC(0 - 10)



Fonte: Autoria própria

Para verificação de **C**, primeiramente, averígua-se a existência de discrepâncias no **C**. Faz-se isso comparando **BP** e **CC** gerados a partir do **C**, **BP** e **CC** provenientes da memória. Caso sejam iguais pode-se aplicar a técnica de Hamming e corrigir as linhas dos dados.

Figura 3.3 – Verificar C correto



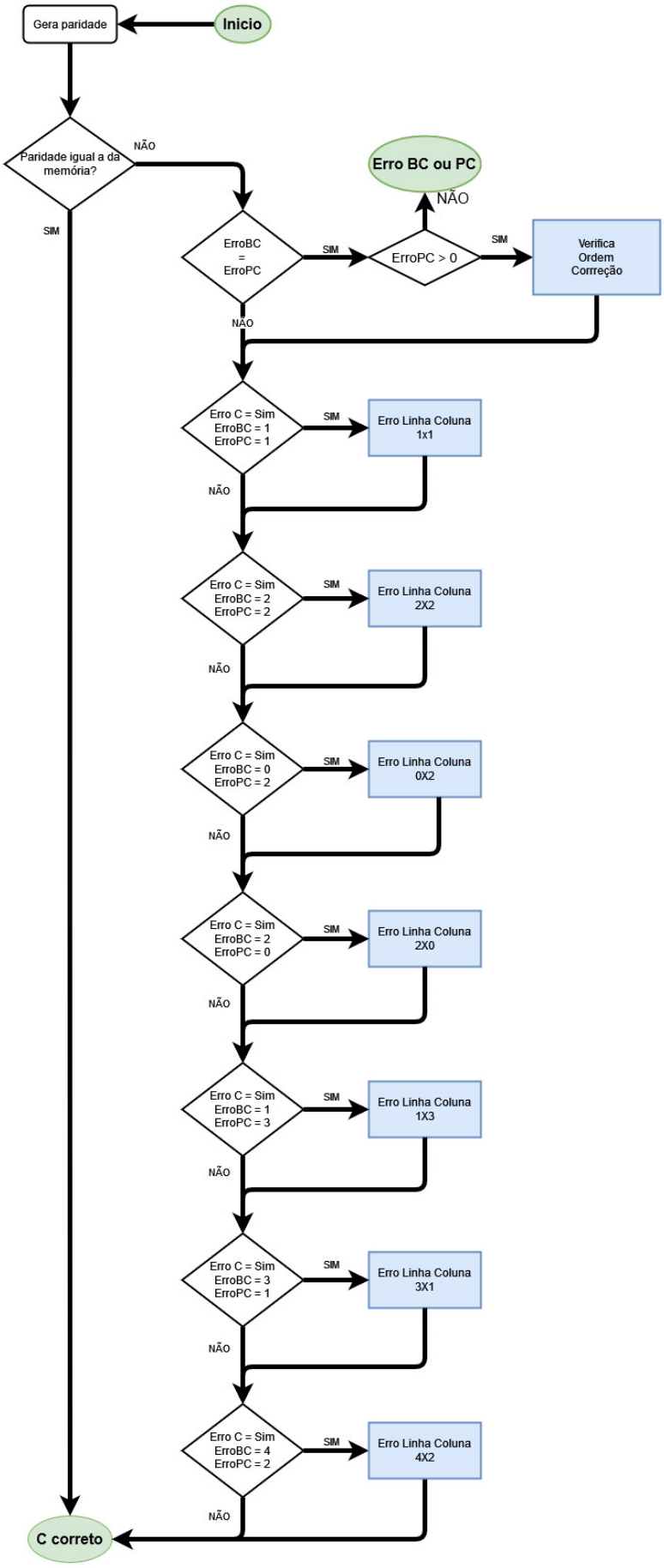
Fonte: Autoria própria

Caso seja necessária a correção do **C**, conta-se quantos **PCs**, entendendo que nesta seção serão referenciados somente as posições do **PC** de 11 a 14, e **BCs** que apresentam diferenças (ErroPC e ErroBC). A combinação destes valores informará o tipo de erro existente em **C** e qual o procedimento para corrigi-los, que consiste na alteração dos bits de **C**, geração de novos **CC** e **BP** e sua comparação com os originais da memória. Esse processo encontra-se descrito na Figura 3.4, onde:

- ErroBC = ErroPC e diferente de 0 : Erro Simples;
- ErroBC = 1 e ErroPC = 1: 3 Erros no **C**;
- ErroBC = 2 e ErroPC = 2: 4 Erros no **C**;
- ErroBC = 0 e ErroPC = 2: 2 Erros no **C**;
- ErroBC = 2 e ErroPC = 0: 2 Erros no **C**;

- ErroBC = 1 e ErroPC = 3: 3 Erros no **C**;
- ErroBC = 3 e ErroPC = 1: 3 Erros no **C**;
- ErroBC = 2 e ErroPC = 4: 4 Erros no **C**;

Figura 3.4 – Fluxograma erro no C

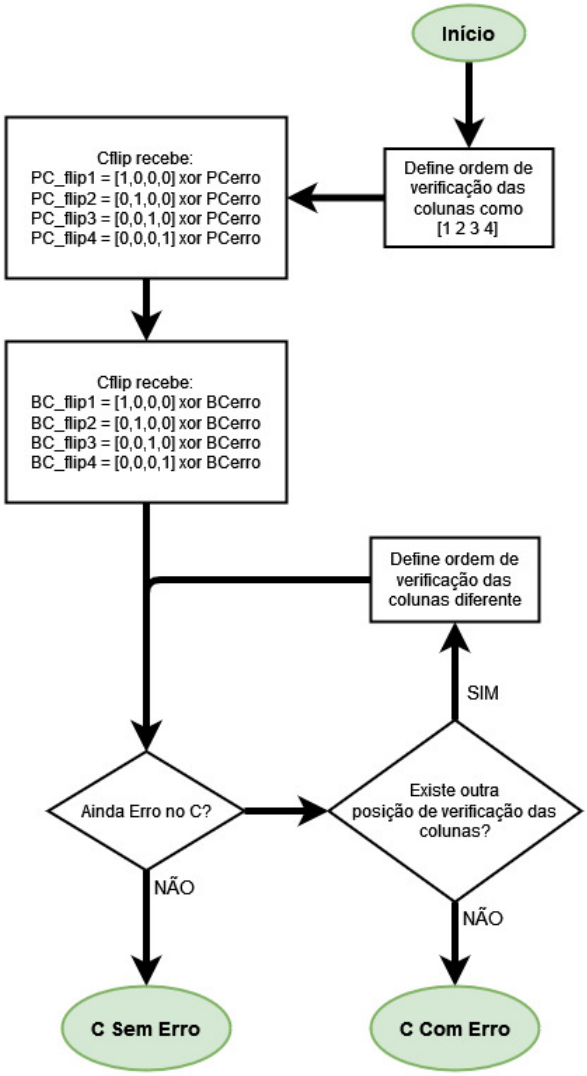


Fonte: Autoria própria

3.1.2.1 Erro: ErroBC = ErroPC

Quando a quantidade de erros em **BC** é a mesma de **PC**, pode-se interpretar a existência de erros simples (Tabela 3.1), onde cada erro encontra-se em uma linha e coluna única. Para corrigi-lo tenta-se cada combinação possível de erros, até que sejam corrigidos ou esgotem-se as possibilidades (Figura 3.5). Neste caso usa-se os procedimentos descritos nas subseções a seguir.

Figura 3.5 – Fluxograma erro simples



Fonte: Autoria própria

Na Tabela 3.1 tem-se um exemplo com 3 erros simples (marcados em laranja) e as posições diferentes, que indicam as linhas e colunas do erro (marcados em azul). Inicialmente define-se as posições C5, C14 e C19, verifica-se se ao alterar esses *bits* de **C** ocorre a correção do mesmo. Caso não o corrija tenta-se a próxima possibilidade de combinação C13, C6 e C19, até que a solução C17, C14 e C7 seja encontrada.

Tabela 3.1 – Exemplo de erro para ErroBC = 3 e ErroPC = 3

C0	C1	C2	C3	BC0
C4	C5	C6	C7	BC1
C8	C9	C10	C11	BC2
C12	C13	C14	C15	BC3
C16	C17	C18	C19	BC4
C20	C21	C22	C23	BC5
PC11	PC12	PC13	PC14	

Fonte: Autoria própria

3.1.2.2 ErroBC = 1 e ErroPC = 1

Em caso de que o **C** não seja corrigido pela permutação de erro simples, pode não ser 1 erro simples, mas 3 erros que estão dispostos de forma que anulam a paridade simples de **BC** e **PC** (Tabela 3.2).

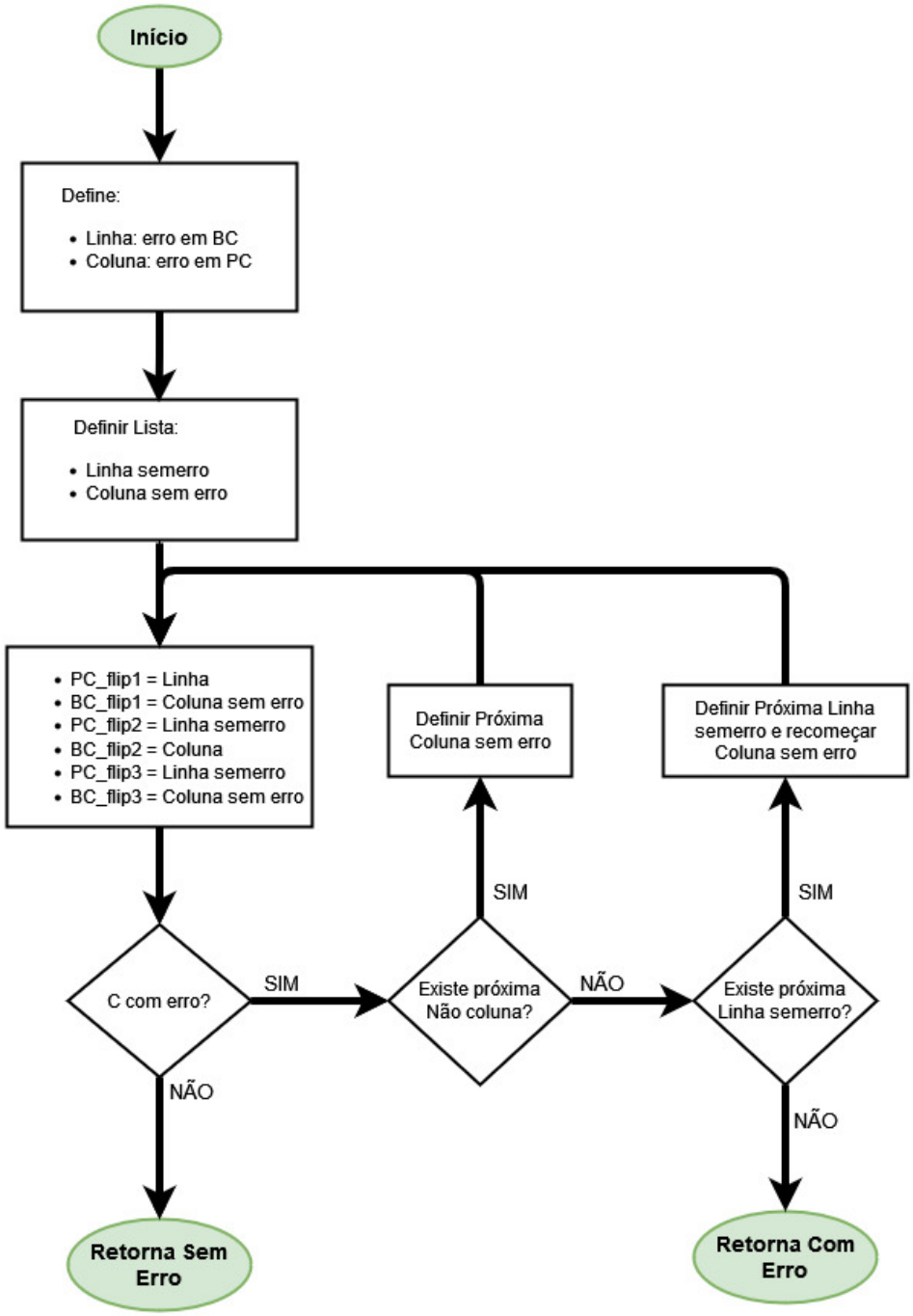
Tabela 3.2 – Exemplo de erro para ErroBC = 1 e ErroPC = 1

C0	C1	C2	C3	BC0
C4	C5	C6	C7	BC1
C8	C9	C10	C11	BC2
C12	C13	C14	C15	BC3
C16	C17	C18	C19	BC4
C20	C21	C22	C23	BC5
PC11	PC12	PC13	PC14	

Fonte: Autoria própria

Para corrigir esse padrão de erro: a) Define-se uma linha sem erro e uma coluna sem erro, verifica-se se a alteração de valor dos pares (linha sem erro, coluna sem erro), (linha sem erro, coluna com erro) e (linha com erro, coluna sem erro) corrigem o problema. b) Caso não seja corrigido escolhe-se a próxima coluna disponível, até que não existam possibilidades disponíveis. c) Se nenhuma solução for encontrada escolhe-se a próxima linha e repete-se o processo *b*, como indicado no fluxograma da Figura 3.6.

Figura 3.6 – Erro Linha x Coluna 1x1



Fonte: Autoria própria

3.1.2.3 ErroBC = 2 e ErroPC = 2

No entanto se o **C** não foi corrigido pela permutação de erro simples, então podem não ser 2 erros simples, mas 3 erros (C5, C7 e C17 da Tabela 3.3) e 1 erro simples (C20

no exemplo da Tabela 3.3) que estão dispostos de forma que anulam a paridade simples de **BC** e **PC**, como mostrado na seção 3.1.2.2.

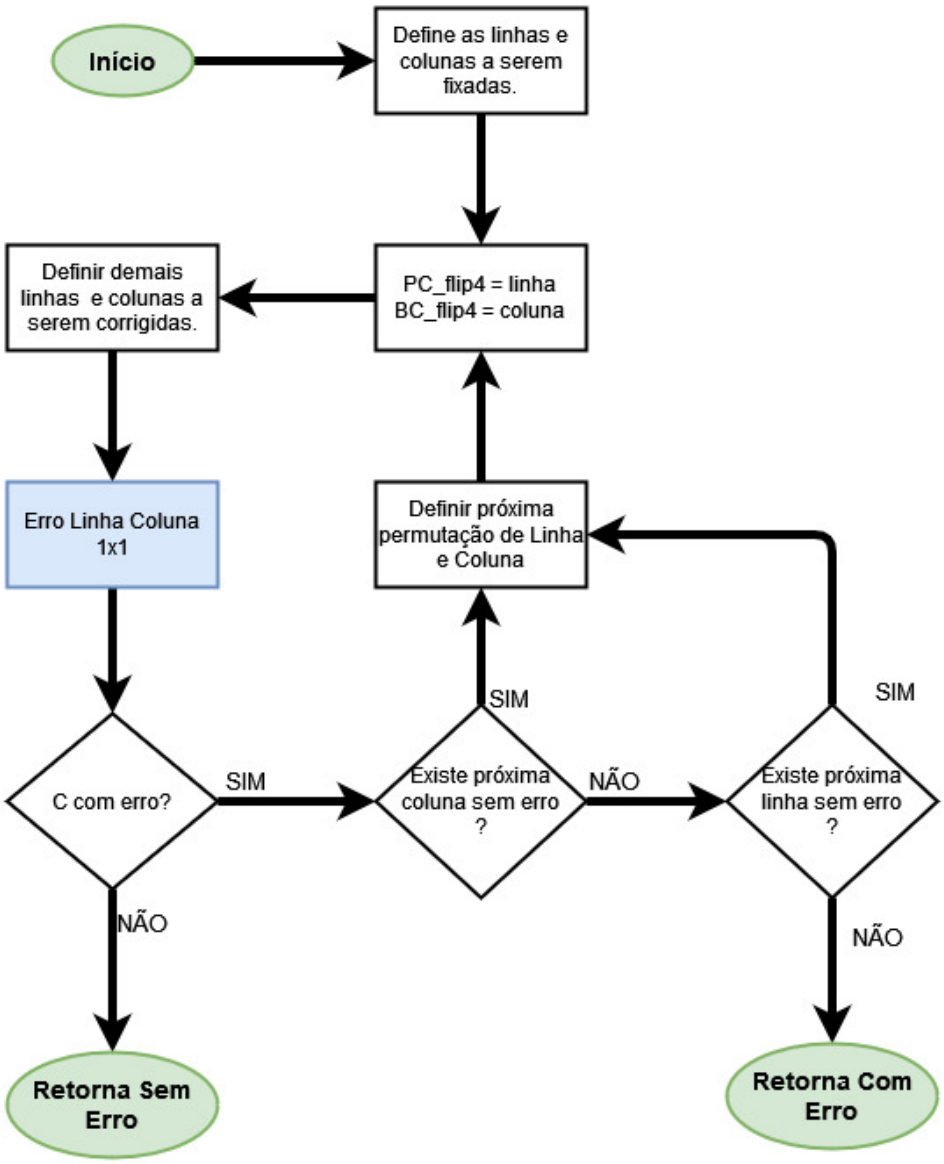
Tabela 3.3 – Exemplo de erro para ErroBC = 2 e ErroPC = 2

C0	C1	C2	C3	BC0
C4	C5	C6	C7	BC1
C8	C9	C10	C11	BC2
C12	C13	C14	C15	BC3
C16	C17	C18	C19	BC4
C20	C21	C22	C23	BC5
PC11	PC12	PC13	PC14	

Fonte: Autoria própria

Para corrigi-lo define-se qual posição pode conter o erro simples e com as demais realiza-se o procedimento apresentado na Seção 3.1.2.2. Caso não encontre-se solução define-se o erro simples como a próxima opção possível e repete-se o procedimento, repetindo-o até que as 4 combinações de erro simples tenham sido testadas ou que a correção seja encontrada (Figura 3.7).

Figura 3.7 – Erro Linha x Coluna 2x2

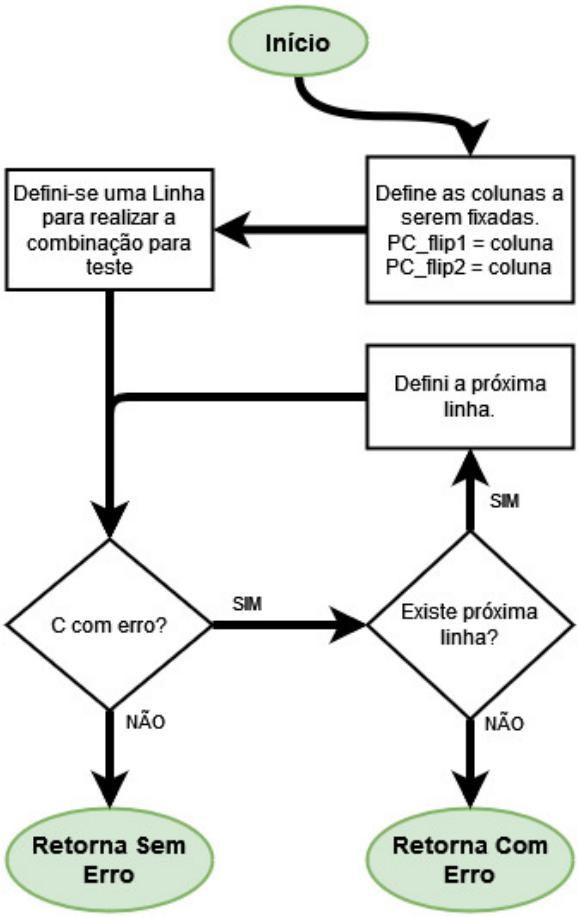


Fonte: Autoria própria

3.1.2.4 ErroBC = 0 e ErroPC = 2

Este padrão de erro é formado com 2 erros na mesma linha, anulando-se assim a paridade simples em **BC**, porém sabe-se em quais colunas contém erro. Para corrigir este padrão de erro fixam-se as colunas de **PC** e tenta-se corrigir uma linha por vez até que encontre-se a solução (Figura 3.8).

Figura 3.8 – Erro Linha x Coluna 0x2



Fonte: Autoria própria

Um exemplo deste erro pode ser observado na Tabela 3.4. Inicia-se fixando PC12 e PC14, em seguida define-se BC0 como a linha a ter os bits invertidos, verifica-se e caso não seja a solução tenta-se a linha seguinte (BC1), até que ao testar com a BC4 obtém-se a correção do **C**.

Tabela 3.4 – Exemplo de erro para ErroBC = 0 e ErroPC = 2

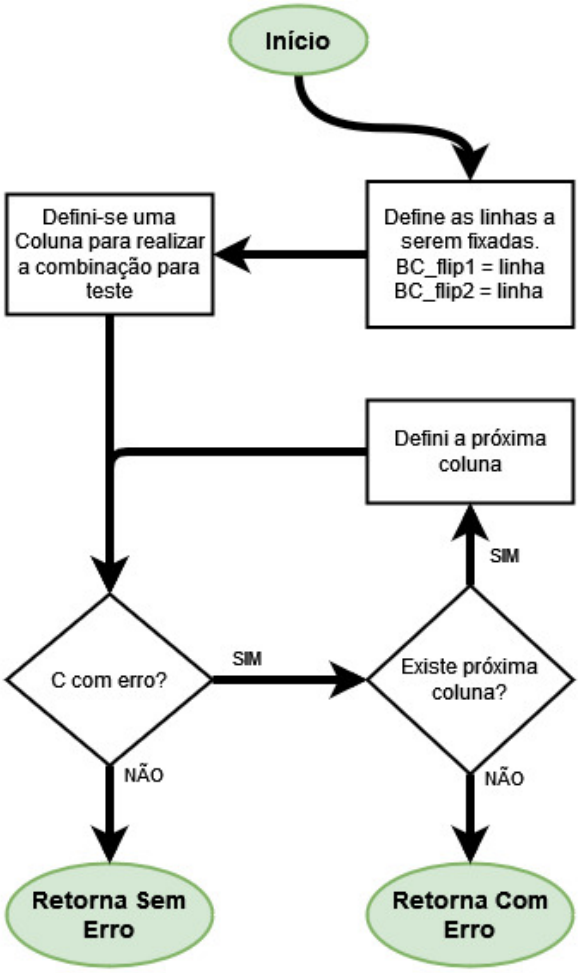
C0	C1	C2	C3	BC0
C4	C5	C6	C7	BC1
C8	C9	C10	C11	BC2
C12	C13	C14	C15	BC3
C16	C17	C18	C19	BC4
C20	C21	C22	C23	BC5
PC11	PC12	PC13	PC14	

Fonte: Autoria própria

3.1.2.5 ErroBC = 2 e ErroPC = 0

Este padrão de erro é formado com 2 erros na mesma coluna, anulando-se assim a paridade simples em **PC**, porém sabe-se em quais colunas contém erro. Para corrigir este padrão de erro fixam-se as linhas de **BC** e tenta-se corrigir uma coluna por vez até que encontre-se a solução (Figura 3.9).

Figura 3.9 – Erro Linha x Coluna 2x0



Fonte: Autoria própria

Um exemplo deste erro pode ser observado na Tabela 3.5. Inicia-se fixando BC1 e BC3, em seguida define-se PC11 como a linha a ter os bits invertidos, verifica-se e caso não seja a solução tenta-se a linha seguinte (PC12), até que ao testar com a PC13 obtém-se a correção do **C**.

Tabela 3.5 – Exemplo de erro para ErroBC = 2 e ErroPC = 0

C0	C1	C2	C3	BC0
C4	C5	C6	C7	BC1
C8	C9	C10	C11	BC2
C12	C13	C14	C15	BC3
C16	C17	C18	C19	BC4
C20	C21	C22	C23	BC5
PC11	PC12	PC13	PC14	

Fonte: Autoria própria

3.1.2.6 ErroBC = 1 e ErroPC = 3

Este padrão é formado por 3 erros em colunas distintas, porém 2 erros estão na mesma linha e 1 em uma linha distinta, sendo a combinação de um erro simples (Seção 3.1.2.1) e dois erros na linha (Seção 3.1.2.4). A Tabela 3.6 apresenta um exemplo da distribuição deste erro, onde o erro 1 é o erro simples e os erros 2 e 3 são os erros na linha.

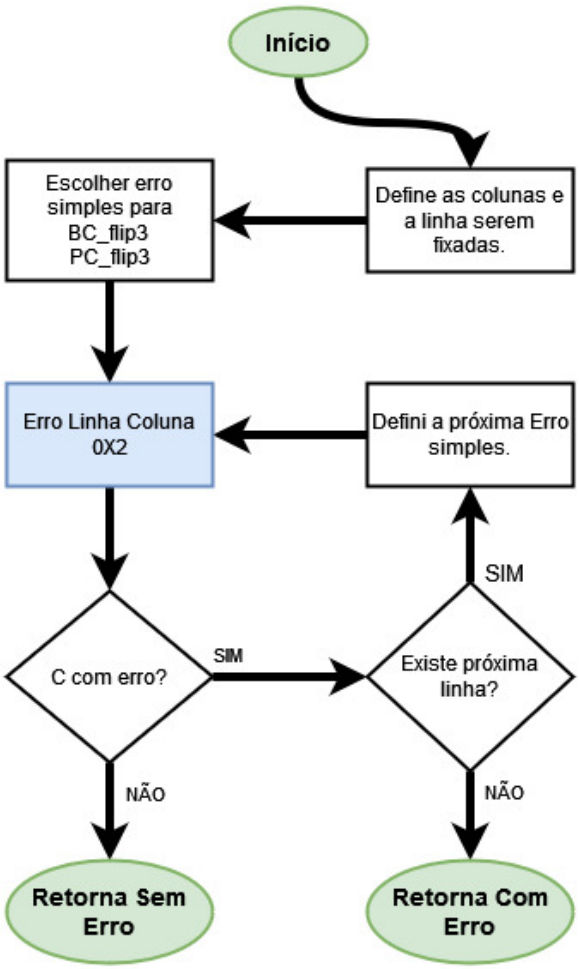
Tabela 3.6 – Exemplo de erro para ErroBC = 1 e ErroPC = 3

C0	C1	C2	C3	BC0
C4	C5	C6	C7	BC1
C8	C9	C10	C11	BC2
C12	C13	C14	C15	BC3
C16	C17	C18	C19	BC4
C20	C21	C22	C23	BC5
PC11	PC12	PC13	PC14	

Fonte: Autoria própria

Para corrigi-los, escolhe-se uma das colunas com erro e a linha com erro para definir como erro simples, então dá-se início ao processo de correção dos dois erros na mesma linha (Seção 3.1.2.4). Caso não seja corrigido, define-se o próximo erro simples possível e faz-se uma nova correção utilizando o processo de dois erros na mesma linha. Este ciclo é feito até que seja encontrado o **C** correto ou cessem-se as possibilidades.

Figura 3.10 – Erro Linha x Coluna 1x3



Fonte: Autoria própria

3.1.2.7 ErroBC = 3 e ErroPC = 1

Este padrão é formado por 3 erros em linhas distintas, porém 2 erros estão na mesma coluna e 1 em uma coluna distinta, sendo a combinação de 1 erro simples (Seção 3.1.2.1) e dois erros na coluna (Seção 3.1.2.5). A Tabela 3.7 apresenta um exemplo da distribuição deste erro, onde o C10 é o erro simples e C5 e C13 são os erros na coluna.

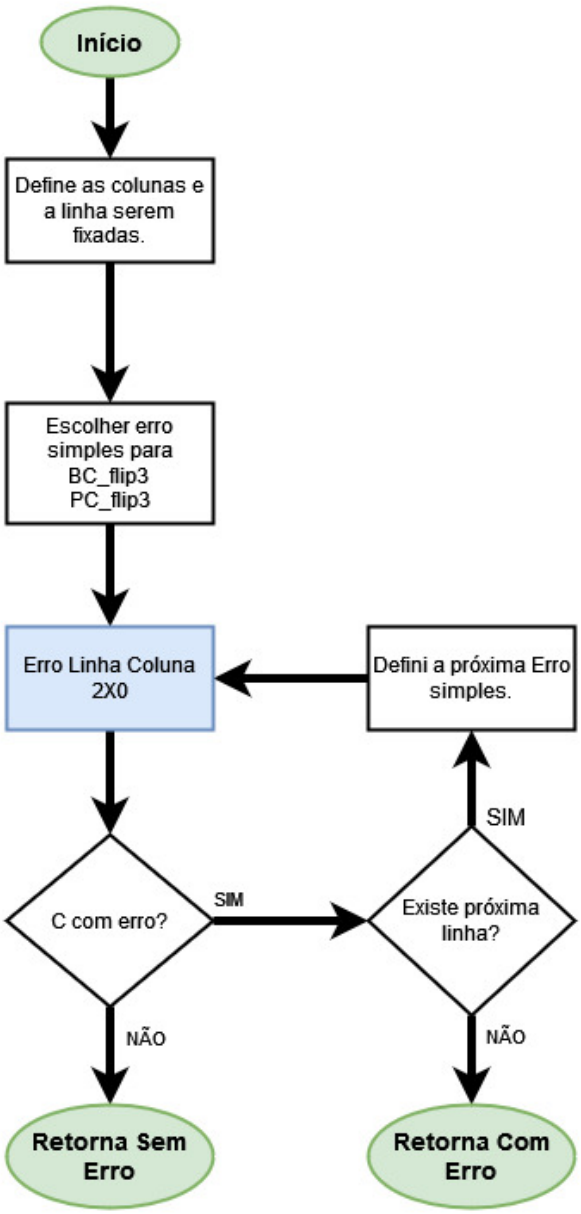
Tabela 3.7 – Exemplo de erro para ErroBC = 3 e ErroPC = 1

C0	C1	C2	C3	BC0
C4	C5	C6	C7	BC1
C8	C9	C10	C11	BC2
C12	C13	C14	C15	BC3
C16	C17	C18	C19	BC4
C20	C21	C22	C23	BC5
PC11	PC12	PC13	PC14	

Fonte: Autoria própria

Para corrigi-los escolhe-se uma das linhas com erro e a coluna com erro para definir como erro simples, dá-se início ao processo de correção dos dois erros na mesma coluna (Seção 3.1.2.5). Caso não seja corrigido, define-se o próximo erro simples possível e faz-se uma nova correção utilizando o processo de dois erros na mesma coluna. Este ciclo é feito até que seja encontrado o **C** correto ou cessem-se as possibilidades.

Figura 3.11 – Erro Linha x Coluna 3x1



Fonte: Autoria própria

3.1.2.8 ErroBC = 4 e ErroPC = 2

Este padrão de erros é a combinação de dois erros simples e dois erros na coluna, como mostrado no exemplo na Tabela 3.8 representados por C10 e C19, e C5 e C13, respectivamente.

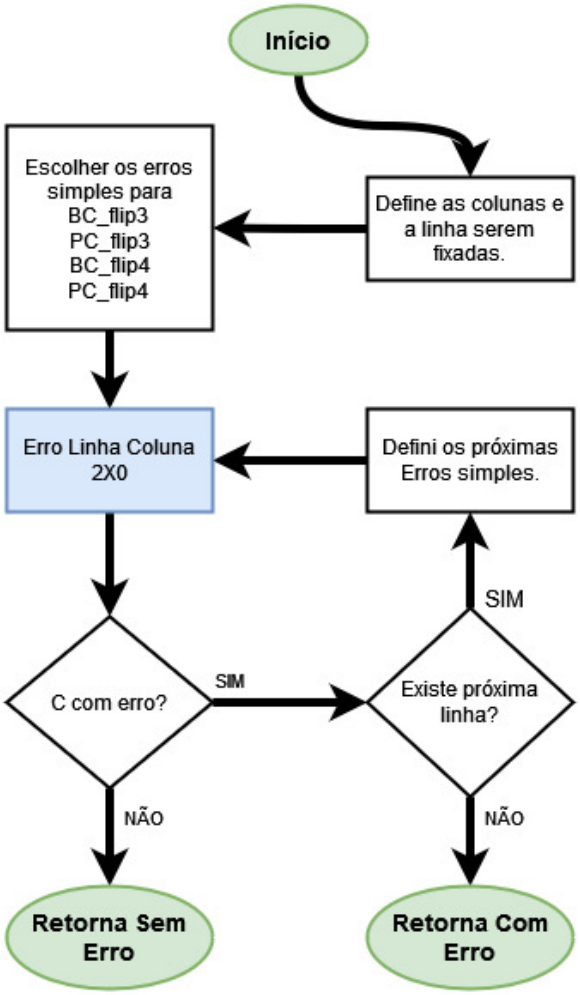
Tabela 3.8 – Exemplo de erro para ErroBC = 4 e ErroPC = 2

C0	C1	C2	C3	BC0
C4	C5	C6	C7	BC1
C8	C9	C10	C11	BC2
C12	C13	C14	C15	BC3
C16	C17	C18	C19	BC4
C20	C21	C22	C23	BC5
PC11	PC12	PC13	PC14	

Fonte: Autoria própria

Para a correção deste padrão de erro define-se 2 erros simples e realiza-se a correção dos demais erros utilizando o processo de correção de dois erros na coluna (Seção 3.1.2.5). Caso não encontre solução, definem-se outros 2 erros simples até que cessem-se mais possibilidades.

Figura 3.12 – Erro Linha x Coluna 4x2



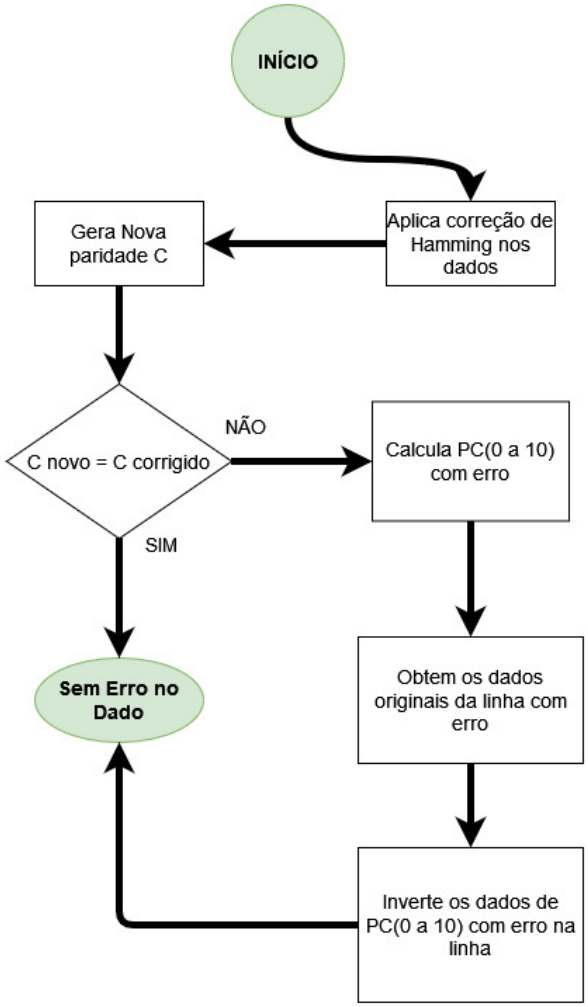
Fonte: Autoria própria

Para corrigir o exemplo da Tabela 3.8 inicialmente define-se as coordenadas dos 2 erros simples como C6 e C11, e o erro da coluna será nas linhas BC2 e BC4. Esta combinação não é a solução, então define-se outro par de erros simples até que seja encontrada a solução. Para este exemplo tem-se 12 combinações de pares simples, sendo elas [C6, C11], [C6, C15], [C6, C19], [C10, C7], [C10, C15], [C10, C19], [C14, C7], [C14, C11], [C14, C19], [C18, C7], [C18, C11] e [C18, C15].

3.1.3 Correção do Dado

Após a correção de **C** e **PC**, faz-se a correção do dado utilizando a técnica de Hamming SEC-DED utilizando o **C** e o **PD**. Contudo, a presença de 3 erros na mesma linha de dados acarreta em uma falsa correção do Hamming. Para mitigar este *evento* gera-se a paridade C com os dados corrigidos, caso alguma linha tenha valor diferente, usa-se o **PC**(0 a 10) e mudam-se os *bits* da linha com erro e das colunas com erro (Figura 3.13).

Figura 3.13 – Fluxograma correção do dado



Fonte: Autoria própria

A seguir tem-se um exemplo da correção de 3 erros em uma mesma linha de dado (Tabela 3.9). Nele o Hamming Estendido é incapaz de corrigir o erro, então restaura-se a linha do dado para o valor original e utilizam-se os **PCs** diferentes como índices das posições a serem invertidas na linha com erro.

Tabela 3.9 – Exemplo 3 erros em uma linha do dado

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	C0	C1	C2	C3	PD0
D11	D12	D13	D14	D15	D16	D17	D18	D19	D20	D21	C4	C5	C6	C7	PD1
D22	D23	D24	D25	D26	D27	D28	D29	D30	D31	D32	C8	C9	C10	C11	PD2
D33	D34	D35	D36	D37	D38	D39	D40	D41	D42	D43	C12	C13	C14	C15	PD3
D44	D45	D46	D47	D48	D49	D50	D51	D52	D53	D54	C16	C17	C18	C19	PD4
D55	D56	D57	D58	D59	D60	D61	D62	D63	-	-	C20	C21	C22	C23	PD5
PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC4	-

Fonte: Autoria própria

3.2 IMPLEMENTAÇÃO EM HDL

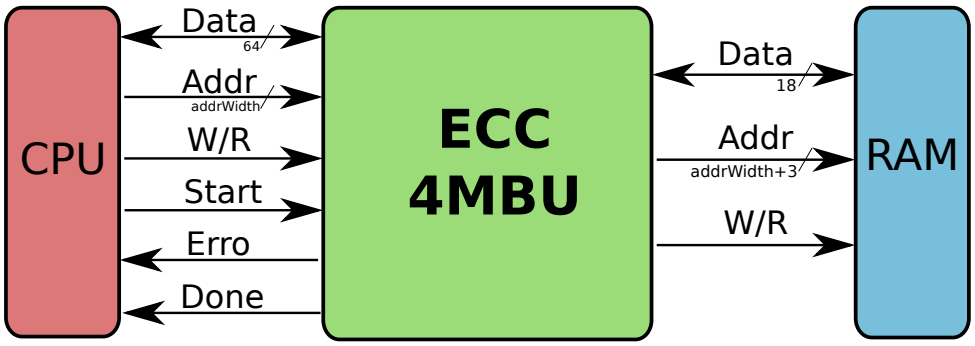
Para a implementação do algoritmo em *hardware* fez-se necessária a criação do circuito através da linguagem de descrição de *hardware* VHDL. Para tal considera-se uma topologia (Figura 3.14) onde a comunicação entre a CPU e o ECC faz-se com:

- **Data**: barramento de 64 *bits* de dado;
- **Addr**: barramento de endereçamento de tamanho genérico;
- **W/R**: sinal para comando de leitura ou escrita do dado da memória;
- **Erro**: sinal que indica para a CPU que ocorreu um erro na memória;
- **Start**: sinal de comando para o início da operação;
- **Done**: sinal que indica o fim da operação;

e a topologia de comunicação entre o ECC e a memória faz-se com:

- **Data**: barramento de 18 *bits* de dado;
- **Addr**: barramento de endereçamento de tamanho genérico + 3 *bits*;
- **W/R**: sinal para comando de leitura ou escrita do dado da memória;

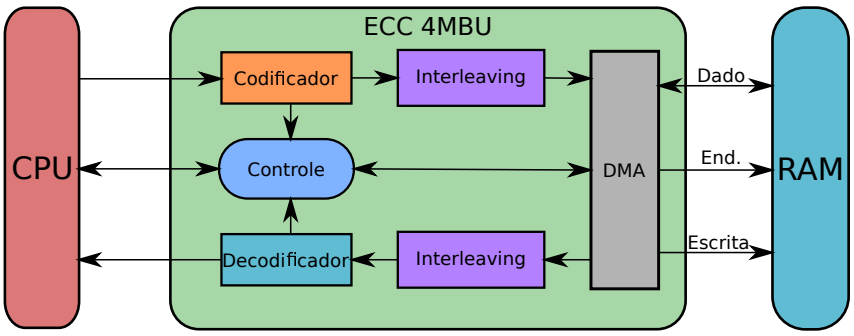
Figura 3.14 – Interface de comunicação do circuito.



Fonte: Autoria própria

Dentro do bloco ECC-4MBU encontram-se as seguintes estruturas: controle do ECC, *Interleaving*, codificador, DMA (*Direct Memory Access*) e decodificador (Figura 3.15).

Figura 3.15 – Topologia funcional em blocos do circuito.



Fonte: Autoria própria

3.2.1 Controle do ECC

O controle do ECC tem por objetivo realizar o direcionamento do dado, para isto utiliza-se de uma FSM (*Finite-State Machine*) de 4 estados, sendo estes o de espera, o de leitura, o de gravação e o de decodificação. Quando encontra-se no estado de espera, significa que o ECC está esperando por algum comando do processador, seja de leitura ou de escrita na memória.

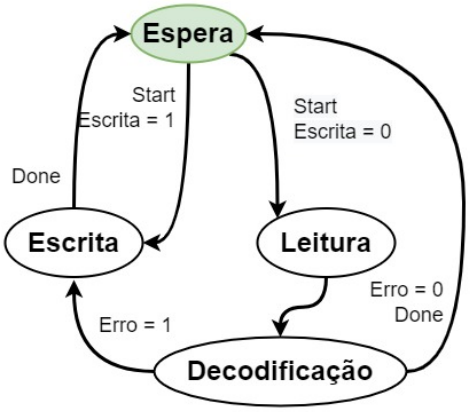
No momento em que o ECC recebe o comando de gravação, ele envia os sinais de controle para o DMA realizar a gravação dos dados na memória e retornar ao estado de espera.

Ao receber o comando de leitura o ECC comanda o DMA para a obtenção dos dados da memória, então permanece no estado de decodificação onde aguarda o decodificador corrigir o dado, caso necessário.

Após a decodificação, caso tenha sido identificado erro no dado ou na paridade é realizada a operação de escrita, onde o dado correto gera uma nova paridade que é salva na memória através do DMA, e então fornece o dado ao processador seguindo, deste modo, novamente para o estado de espera.

Já em caso de não possuir erros o dado é fornecido para o processador e volta para o estado de espera.

Figura 3.16 – Máquina de estados do ECC



Fonte: Autoria própria

3.2.2 Interleaving

O *Interleaving* está presente na entrada e saída do DMA, onde é responsável por embaralhar os dados que serão enviados para a memória e por desembaralhar aqueles provenientes da memória, sua estrutura é considerada utilizando o mapeamento direto, onde o bit da posição 0 da entrada é a posição 64 da saída, da posição 1 é a saída 19 e assim sucessivamente, como descrito em (ROHDE, 2020).

3.2.3 Codificador

O codificador consiste na geração da paridade a partir dos dados, para isto utiliza de blocos de circuito dispostos de forma que gerem a paridade, sendo estes: codificadores simples, geradores de Hamming e Hamming Estendido e somadores (Figura 3.1).

3.2.4 DMA

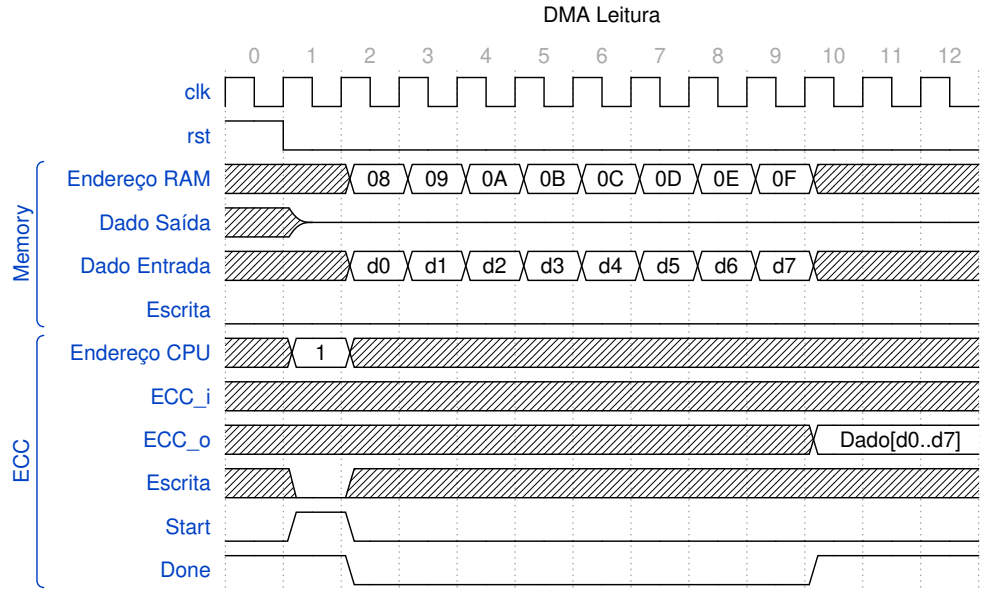
O DMA é responsável pela obtenção e armazenamento do dado na memória, pois cada endereço fornecido pelo processador é convertido em 8 posições consecutivas na memória. Para realizar a comunicação com os circuitos externos utiliza as seguintes portas:

- **Endereço CPU**: Endereço fornecido pela CPU;

- **Endereço RAM**: Endereço da memória;
- **Dado Entrada**: Dado obtido da memória SRAM;
- **Dado Saída**: Dado a ser escrito na memória SRAM;
- **Escrita_i**: Sinal de escrita proveniente da CPU e do bloco decodificador caso tenha ocorrido algum bit flip na memória SRAM;
- **Escrita_o**: Sinal de escrita enviado à memória SRAM;
- **ECC_i**: Dodo + Paridade proveniente da CPU após a realização do interleaving.
- **ECC_o**: Dado + Paridade provindos da memória SRAM com interleaving;
- **Start**: Sinal de comando do começo da operação de leitura ou escrita;
- **Done**: Sinal que indica que o DMA concluiu a operação e está a espera do próximo comando.

Para o processo de leitura da memória ocorrer é necessário que seja informado o **endereço CPU** a ser lido, o comando de **escrita_i** ser "0"e o sinal ***Start*** seja habilitado. Então o DMA instanciará em cada ciclo de sinal de relógio um dos 8 **endereços RAM** em que o dado referente ao endereço fornecido está armazenado. Ao final dos ciclos o sinal ***Done*** será "1"e o dado lido estará em ***ECC_o*** (Figura 3.17).

Figura 3.17 – Formas de onda da leitura da memória

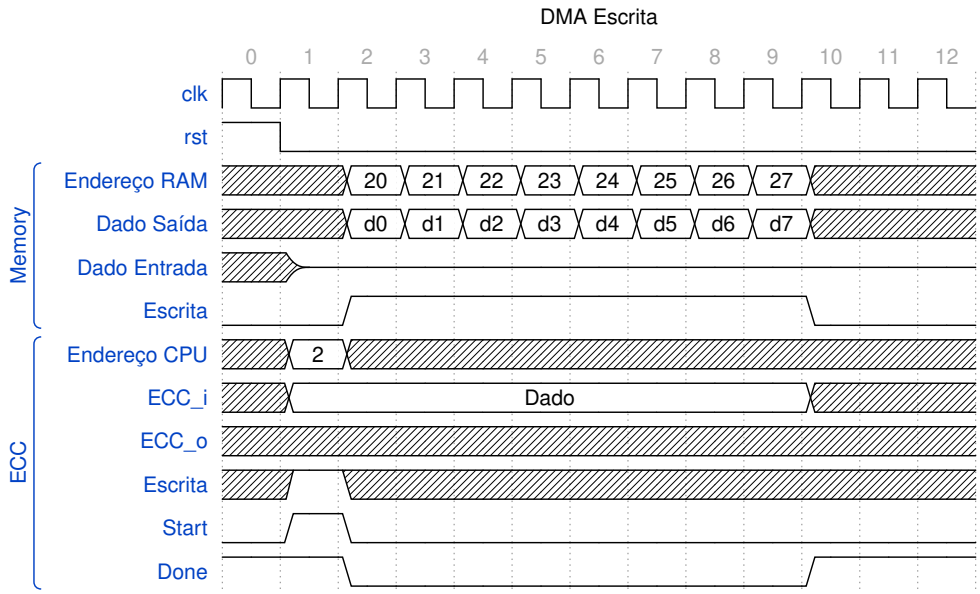


Fonte: Autoria própria

Já para o processo de escrita da memória ocorrer é necessário que seja informado o **endereço CPU** a ser armazenado, o dado a ser armazenado (**ECC_i**), o comando de **escrita_i** ser "1"e o sinal ***Start*** ser habilitado. Deste modo, o DMA instanciará em cada ciclo de sinal de relógio um dos 8 **endereços RAM** em que o dado fornecido será armazenado.

Ao final dos ciclos o sinal ***Done*** será "1"e o dado estará armazenado na memória (Figura 3.18).

Figura 3.18 – Formas de onda da escrita na memória



Fonte: Autoria própria

3.2.5 Decodificador

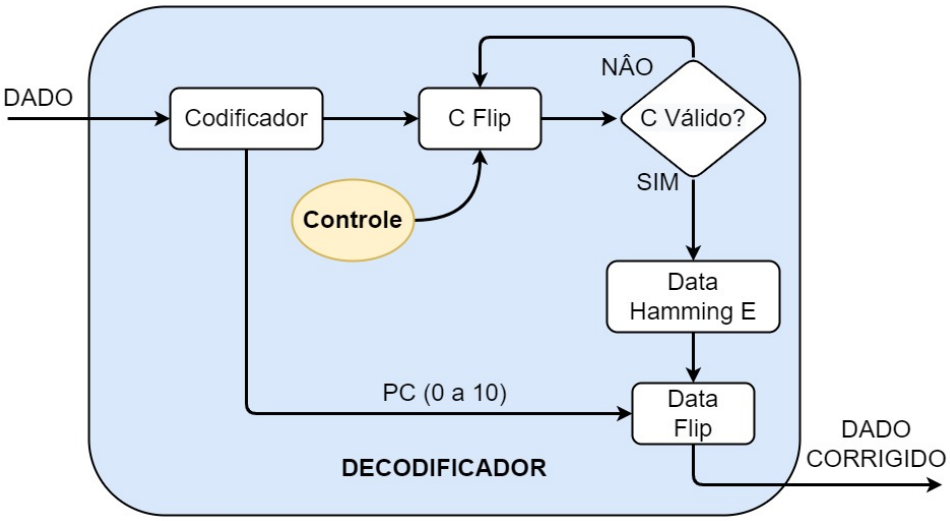
Após o dado sair do DMA, passa pelo circuito que desfaz o embaralhamento e chegando ao decodificador, onde o dado recebido será verificado e corrigido caso seja detectado algum erro.

Para verificar se houve algum erro enquanto o dado estava armazenado utiliza-se os seguintes blocos:

- **Codificador**: Para gerar uma paridade a partir do dado;
- **C_flip**: responsável por mudar 1 bit no conjunto CC da paridade;
- **Data_flip**: responsável por mudar 1 bit no bloco de dados;
- **C_valido**: Gerador de paridade a partir do conjunto C;
- **Decodificadores de Hamming estendido**;
- **Controle**: responsável por gerar os sinais de controle para os corretores e assim garantir a correta correção do dado.

Estes blocos estão dispostos segundo a Figura 3.19

Figura 3.19 – Estrutura do Decodificador



Fonte: Autoria própria

A seguir encontram-se apresentados os blocos e sua funcionalidade:

3.2.5.1 C flip e Dado flip

O inversor de C e o inversor do dado são responsáveis pela mudança de um ou mais *bits* através do mapeamento de linha e coluna. Ele é instanciado 4 vezes para corrigir o C, sendo que a saída de cada um é ligada a entrada do próximo, assim é possível que se realizem 4 conversões distintas.

Para a correção de múltiplos erros no dado, é implementada a mesma estrutura inversora, porém utiliza-se apenas uma instância, já que após a execução do Hamming Estendido tem-se 2 ou 3 erros em uma linha, para tanto, o inversor necessitará alterar os dados apenas em uma linha. Também poderia ocorrer 2 erros em 2 linhas, que necessitariam de 2 estruturas em série, todavia o *interleaving* impede que tal formação ocorra.

Para os índices dos inversores do dado tem-se as posições diferentes de **PC**(0 a 10) corrigido e do dado atual, e para identificar a linha gera-se novamente os bits de Hamming buscando a linha que não tem os mesmos valores que o C corrigido.

3.2.5.2 C válido

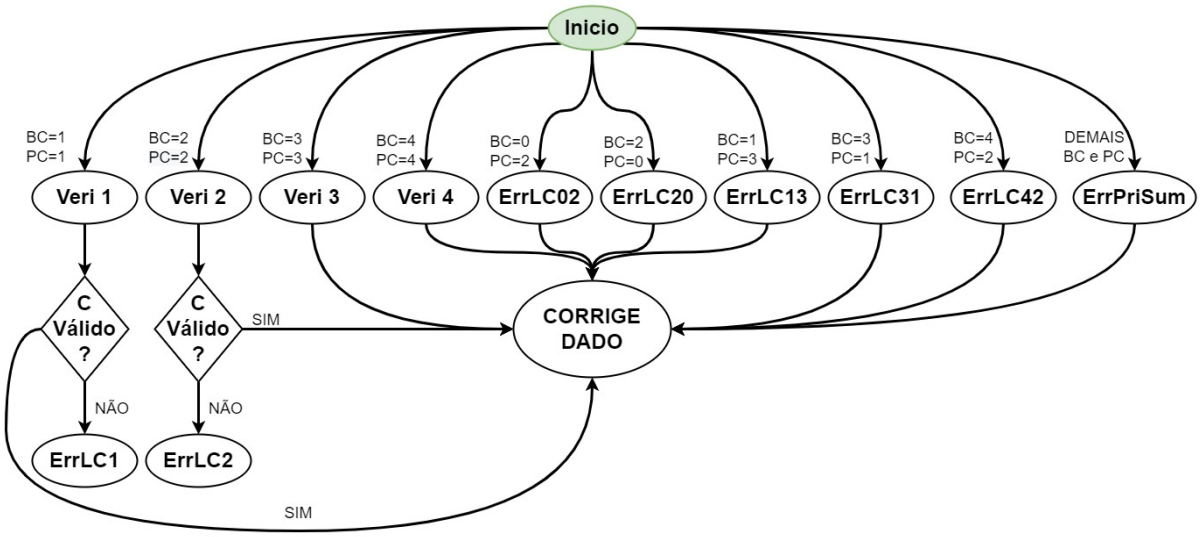
Este bloco é responsável por gerar as paridades **CC** e **BP** provenientes do *C flip*, compara-las com as provenientes da memória e emitir um sinal informando se são iguais

("1") ou diferentes ("0").

3.2.5.3 Controle

Para a correção do **C** são realizados diversas etapas, até que o C correto seja encontrado, para isso utilizam-se máquinas de estados secundárias. Na Figura 3.20 está disposta a estrutura de máquinas de estados. Quem habilita o início de seu funcionamento é o controle do ECC, que detecta o fim da obtenção do dado pelo DMA.

Figura 3.20 – Máquina de estados do decodificador



Fonte: Autoria própria

3.3 VETORES DE TESTES

Durante o processo de desenvolvimento realizam-se testes de funcionamento dos blocos, sendo gerado padrões de dados (vetores) para as entradas dos blocos e então é realizada uma análise da saída. Para o Hamming foram geradas todas as opções de erros corrigíveis e algumas não corrigíveis.

Já com o inversor de dado e inversor do **C** testa-se invertendo uma posição por vez, dois elementos por linha, dois elementos por coluna e algumas combinações de linhas e colunas.

Por sua vez, com o *interleaving* o teste realiza-se conectando 2 blocos idênticos, um embaralhando e o outro desembaralhando. Para testar insere-se todos os dados "0"e o *bit* da posição 0 como "1", analisa-se se após o embaralhamento a posição é a correta e se

após a passagem pelo segundo bloco retorna para a posição original. Ao final, realiza-se o teste para cada uma das 144 posições possíveis.

Em seguida, para o DMA realizam-se testes de gravações e leituras do dado com padrão xadrez e 1 bit por posição.

Por fim, para testar o ECC foram gerados vetores de dados considerando o dado como todos os bits "0" e a posição do erro como "1", para uma mais clara visualização da posição do erro. Os vetores agrupam-se em 4 arquivos, contendo neles os 144 padrões com 1 erro, os 3456 padrões para 2 erros, os 8064 padrões para 3 erros e os 12096 padrões para 4 erros. Estes padrões são obtidos considerando-se que podem ocorrer em uma área de 3x3 células. Sendo os vetores de testes carregados para a memória e então lidos.

Ao final dos testes armazena-se a quantidade de ciclos necessários para a obtenção do dado, que serão analisados no próximo capítulo.

3.4 IMPLEMENTAÇÃO EM FPGA

Para a execução da implementação em FPGA escolheu-se as seguintes linhas da Xilinx: *Spartan 3*, com litografia de 90 nm, *Spartan 6* com litografia de 45 nm e *Artix 7*, com litografia de 28 nm. A escolha das mesmas deu-se devido a diferença entre as tecnologias de fabricação, suas especificações encontram-se nas figuras a seguir (Figura 3.21, 3.22, 3.23).

Figura 3.21 – Folha de dados das Spartan3.

Device	System Gates	Equivalent Logic Cells	CLBs	Slices	Distributed RAM Bits(1)	Block RAM Bits(1)	In-System Flash Bits(2)	Dedicated Multipliers	DSP48As	DCMs	Maximum User I/O
XC3S50A/AN	50K	1,584	176	704	11K	54K	1M	3	-	2	144
XC3S200A/AN	200K	4,032	448	1,792	28K	288K	4M	16	-	4	248(3)
XC3S400A/AN	400K	8,064	896	3,584	56K	360K	4M	20	-	4	311
XC3S700A/AN	700K	13,248	1,472	5,888	92K	360K	8M	20	-	8	372
XC3S1400A/AN	1400K	25,344	2,816	11,264	176K	576K	16M	32	-	8	502
XC3SD1800A	1800K	37,440	4,160	16,640	260K	1,512K	-	-	84	8	519
XC3SD3400A	3400K	53,712	5,968	23,872	373K	2,268K	-	-	126	8	469

Notes:
1. By convention, one Kb is equivalent to 1,024 bits.
2. In-System flash is available in the Spartan-3AN devices only.
3. Maximum user I/O for XC3S200AN is 195.

Fonte: Adaptado de (XILINX, 2011a)

Figura 3.22 – Folha de dados das Spartan6.

Table 1: **Spartan-6 FPGA Feature Summary by Device**

Device	Logic Cells(1)	Configurable Logic Blocks (CLBs)			DSP48A1 Slices(3)	Block RAM Blocks		CMTs(5)	Memory Controller Blocks (Max)(6)	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices(2)	Flip-Flops	Max Distributed RAM (Kb)		18 Kb(4)	Max (Kb)						
XC6SLX4	3,840	600	4,800	75	8	12	216	2	0	0	0	4	132
XC6SLX9	9,152	1,430	11,440	90	16	32	576	2	2	0	0	4	200
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232
XC6SLX25	24,051	3,758	30,064	229	38	52	936	2	2	0	0	4	266
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358
XC6SLX75	74,637	11,662	93,296	692	132	172	3,096	6	4	0	0	6	408
XC6SLX100	101,261	15,822	126,576	976	180	268	4,824	6	4	0	0	6	480
XC6SLX150	147,443	23,038	184,304	1,355	180	268	4,824	6	4	0	0	6	576
XC6SLX25T	24,051	3,758	30,064	229	38	52	936	2	2	1	2	4	250
XC6SLX45T	43,661	6,822	54,576	401	58	116	2,088	4	2	1	4	4	296
XC6SLX75T	74,637	11,662	93,296	692	132	172	3,096	6	4	1	8	6	348
XC6SLX100T	101,261	15,822	126,576	976	180	268	4,824	6	4	1	8	6	498
XC6SLX150T	147,443	23,038	184,304	1,355	180	268	4,824	6	4	1	8	6	540

Notes:

1. Spartan-6 FPGA logic cell ratings reflect the increased logic cell capability offered by the new 6-input LUT architecture.
2. Each Spartan-6 FPGA slice contains four LUTs and eight flip-flops.
3. Each DSP48A1 slice contains an 18 x 18 multiplier, an adder, and an accumulator.
4. Block RAMs are fundamentally 18 Kb in size. Each block can also be used as two independent 9 Kb blocks.
5. Each CMT contains two DCMs and one PLL.

Fonte: Adaptado de (XILINX, 2011b)

Figura 3.23 – Folha de dados das Artix7.

Table 4: **Artix-7 FPGA Feature Summary by Device**

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices(2)	Block RAM Blocks(3)			CMTs(4)	PCIe(5)	GTPs	XADC Blocks	Total I/O Banks(6)	Max User I/O(7)
		Slices(1)	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XC7A12T	12,800	2,000	171	40	40	20	720	3	1	2	1	3	150
XC7A15T	16,640	2,600	200	45	50	25	900	5	1	4	1	5	250
XC7A25T	23,360	3,650	313	80	90	45	1,620	3	1	4	1	3	150
XC7A35T	33,280	5,200	400	90	100	50	1,800	5	1	4	1	5	250
XC7A50T	52,160	8,150	600	120	150	75	2,700	5	1	4	1	5	250
XC7A75T	75,520	11,800	892	180	210	105	3,780	6	1	8	1	6	300
XC7A100T	101,440	15,850	1,188	240	270	135	4,860	6	1	8	1	6	300
XC7A200T	215,360	33,650	2,888	740	730	365	13,140	10	1	16	1	10	500

Notes:

1. Each 7 series FPGA slice contains four LUTs and eight flip-flops; only some slices can use their LUTs as distributed RAM or SRLs.
2. Each DSP slice contains a pre-adder, a 25 x 18 multiplier, an adder, and an accumulator.
3. Block RAMs are fundamentally 36 Kb in size; each block can also be used as two independent 18 Kb blocks.
4. Each CMT contains one MMCM and one PLL.
5. Artix-7 FPGA Interface Blocks for PCI Express support up to x4 Gen 2.
6. Does not include configuration Bank 0.
7. This number does not include GTP transceivers.

Fonte: Adaptado de (XILINX, 2020)

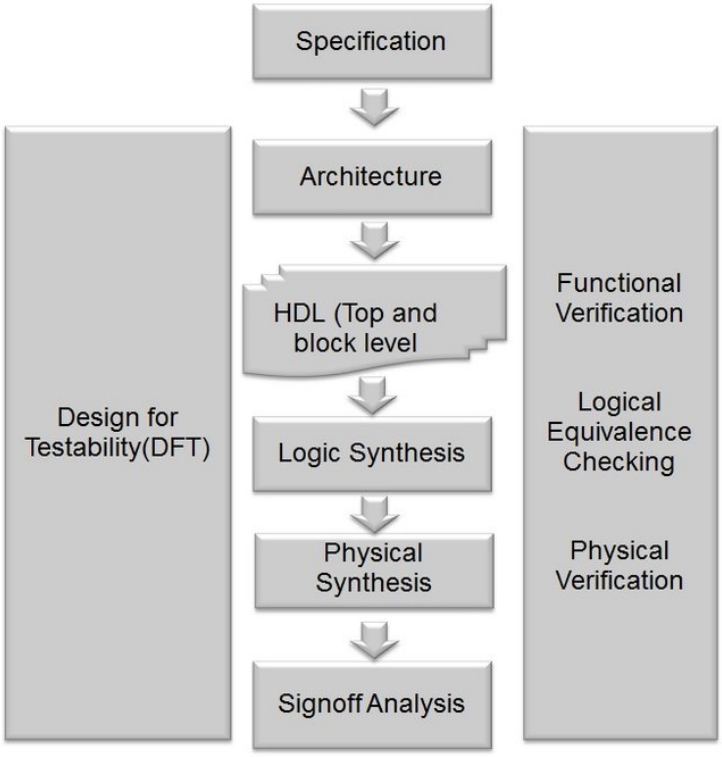
Pode-se observar que o avanço da tecnologia permitiu que mais blocos fossem inseridos nas FPGAs, possibilitando assim a integração de sistemas mais complexos. Todavia, a quantidade de pinos de I/O manteve-se constante, por não ocorrer uma diminuição das dimensões dos dispositivos, sendo limitada a comunicação com os demais componentes a no máximo 500 pinos.

3.5 SÍNTESE

A realização da síntese lógica do ASIC do ECC foi realizada com o *software* da Cadence™ utilizando o *design kit* XH018 da XFab e sua versão modificada com RHBD pela SMDH (VELAZCO; MCMORROW; ESTELA, 2019).

Para o processo de síntese utilizou-se o fluxo de projeto (Figura 3.24) e considerou-se o pior *corner* da tecnologia, sendo baixa tensão de alimentação ($1,6\ Volts$), alta temperatura ($125\ ^{\circ}C$) e pior processo de fabricação (*slow*). Este ponto de operação é conhecido como *worst case* (pior cenário). Assim, quando é realizada a síntese com estes parâmetros garante-se o funcionamento do chip em quaisquer situações para o qual o chip foi projetado.

Figura 3.24 – Fluxo de projeto para um ASIC



Fonte: Adaptado de (PARIS et al., 2015)

Os resultados da síntese bem como os demais resultados obtidos dos processos descritos no presente capítulo serão apresentados e analisados no capítulo seguinte.

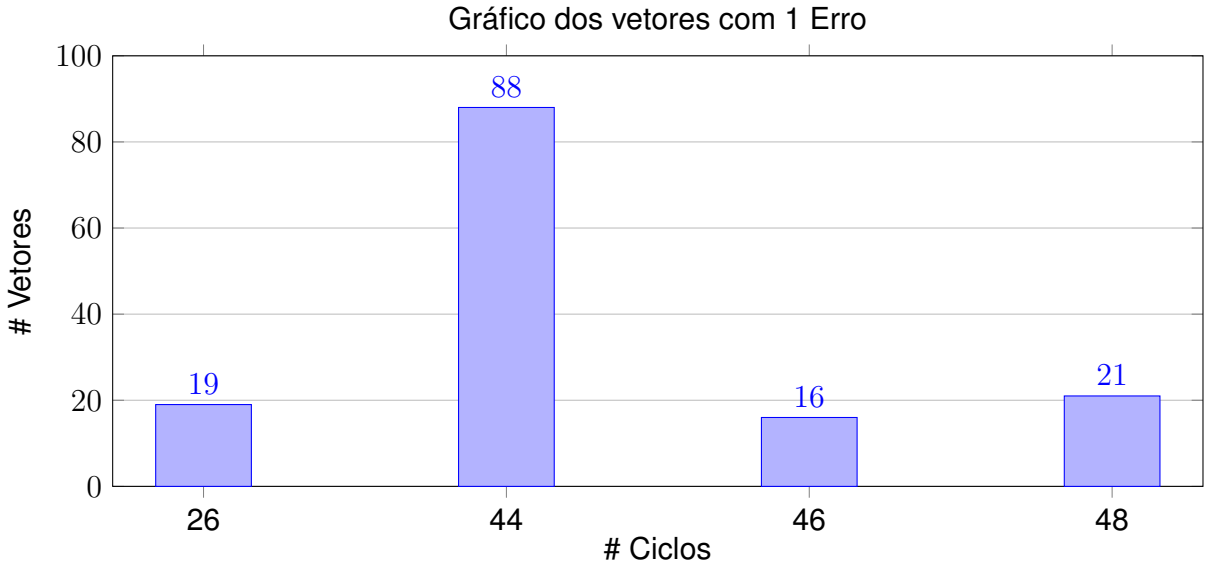
4 RESULTADOS

No presente capítulo serão apresentados os resultados referentes ao vetores de testes e às sínteses em FPGA e ASIC.

4.1 RESULTADO DOS VETORES DE TESTE

Após a implementação do código VHDL foram realizados os testes com os vetores para correção de 1, 2, 3 e 4 erros possíveis em uma área de 3x3 células de memória utilizando o *software* ModelSim. Nas Figuras 4.1, 4.2, 4.3 e 4.4 apresentam-se os gráficos com a quantidade de vetores pela quantidade de ciclos necessários para correção do dado armazenado com erro.

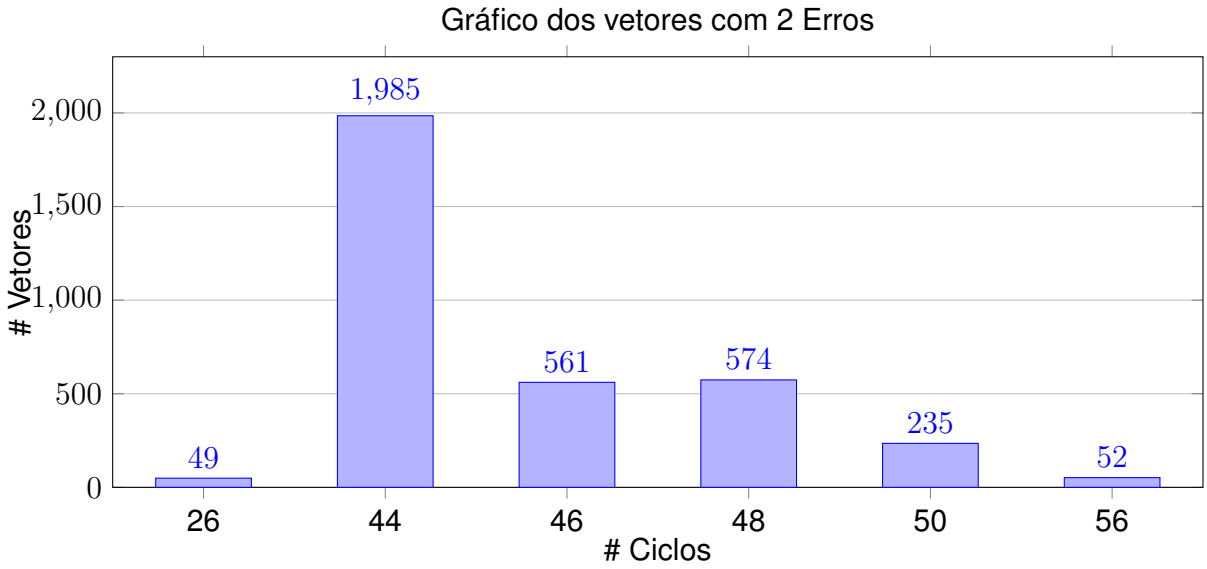
Figura 4.1 – Distribuição dos 144 erros por ciclos para correção



Fonte: Autoria própria

Para 1 erro pode se obter o dado de 26 a 48 ciclos, sendo que 19 padrões de erros levaram 26 ciclos para serem obtidos e 61 % dos vetores levaram 44 ciclos para serem corrigidos e disponibilizados para o processador.

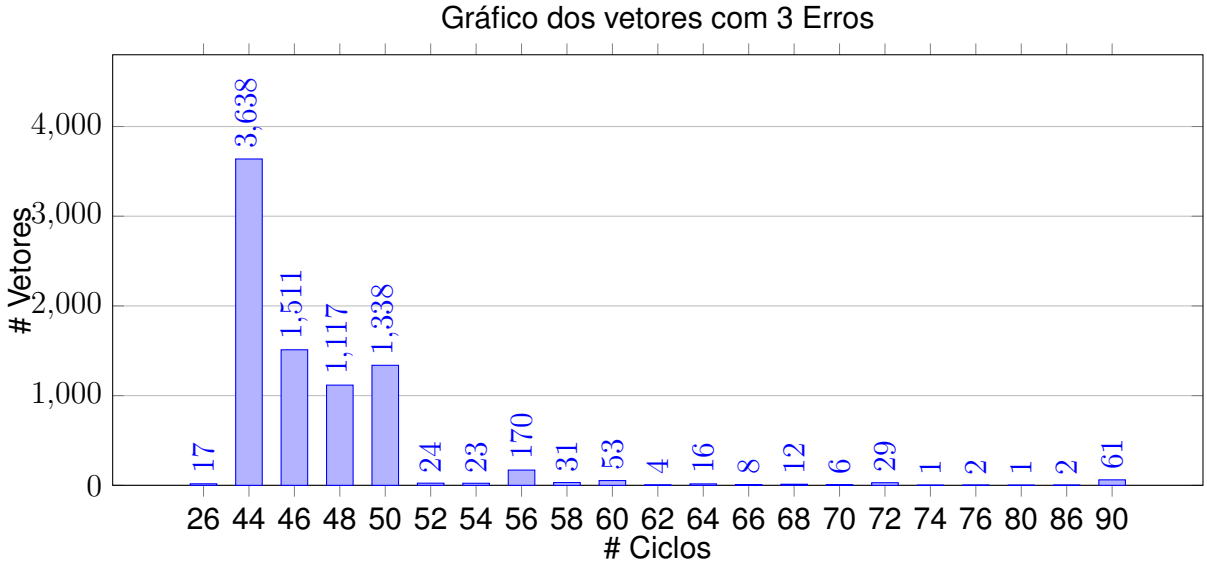
Figura 4.2 – Distribuição dos 3456 erros por ciclos para correção



Fonte: Autoria própria

Para 2 erros pode se obter o dado de 26 a 56 ciclos, sendo que 49 padrões de erros levaram 26 ciclos para serem obtidos e 58 % dos vetores também levaram 44 ciclos para serem corrigidos e disponibilizados para o processador.

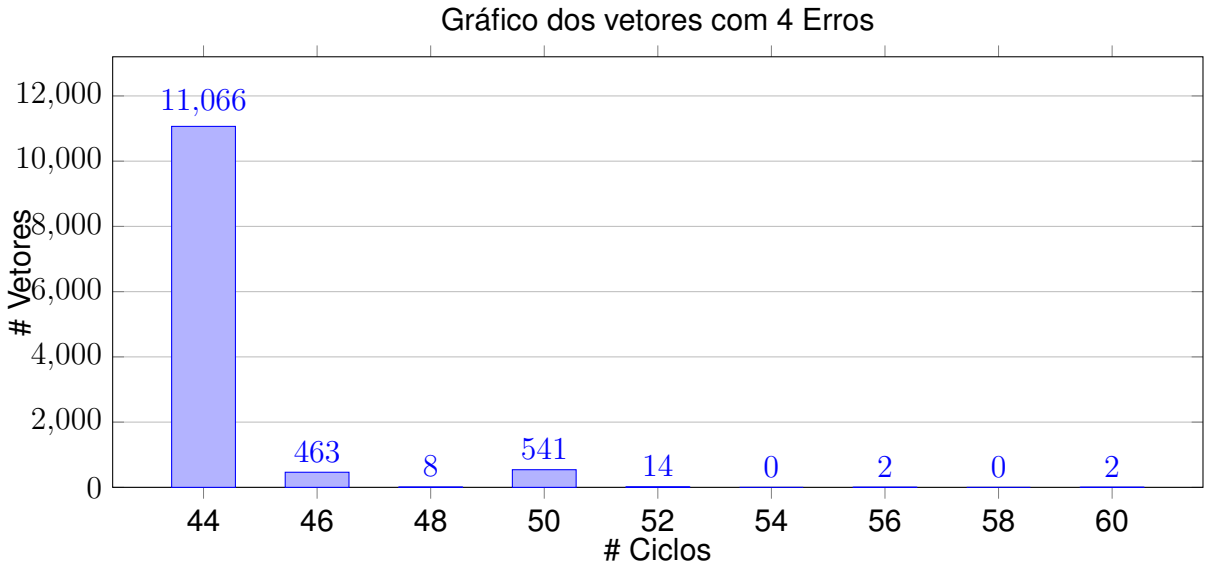
Figura 4.3 – Distribuição dos 8064 erros por ciclos para correção



Fonte: Autoria própria

Para 3 erros pode se obter o dado de 26 a 90 ciclos, sendo que 17 padrões de erros levaram 26 ciclos para serem obtidos, 45 % dos vetores também levaram 44 ciclos e 94 % dos vetores estão entre 44 e 50 ciclos para serem corrigidos e disponibilizados para o processador.

Figura 4.4 – Distribuição dos 12096 erros por ciclos para correção



Fonte: Autoria própria

Para 4 erros pode se obter o dado de 44 a 60 ciclos, sendo que 91 % dos padrões de erro levaram 44 ciclos para serem obtidos. Esta redução em relação a 3 erros deve-se ao *interleaving* que distribui os erros entre a paridade e o dado ou em diversas linhas da matriz do dados (Figura 2.12), assim a correção de diversos erros é realizada de forma paralela.

Ao realizar uma média do tempo de correção tem-se 42 ciclos para 1 erro, 45 ciclos para 2 erros, 47 ciclos para 3 erros e 44 ciclos para 4 erros. Deve-se considerar que a quantidade de SBU e MBU são dependentes da tecnologia de fabricação, do leiaute da célula de memória, da tensão de alimentação e do ângulo de incidência da partícula radioativa (TONFAT et al., 2017).

4.2 SÍNTESE EM FPGA E ASIC

Posterior a obtenção do número de ciclos para correção por erro, realizou-se a síntese nas FPGAs e em ASIC.

4.2.1 FPGA

Para o presente trabalho optou-se pela utilização de diversas famílias de FPGA da Xilinx, sendo escolhidos os modelos intermediários. Para a Artix 7 escolheu-se um modelo já utilizado na missão NanosatC-BR2 (Tabela 4.1).

Família	Tecnologia	Modelo	*Software*
Spartan3	90 nm	XC3S1400A	ISE 14.7
Spartan6	45 nm	XC6SLX75	ISE 14.7
Artix7	28 nm	XC7A100T	Vivado ML 2021.1

Tabela 4.1 – Tabela de FPGAs selecionadas

Nas Figuras 4.5 e 4.6 apresentam-se os relatórios de ocupação e potência da Spartan 3. Pode-se observar que a potência de *leakage* é de 44 % e ocorreu a utilização de todas as *Slices* contendo somente lógica.

Figura 4.5 – Reporte de área ocupada do *software* ISE 14.7

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	1,461	22,528	6%
Number of 4 input LUTs	6,506	22,528	28%
Number of occupied Slices	4,020	11,264	35%
Number of Slices containing only related logic	4,020	4,020	100%
Number of Slices containing unrelated logic	0	4,020	0%
Total Number of 4 input LUTs	6,538	22,528	29%
Number used as logic	6,506		
Number used as a route-thru	32		
Number of bonded IOBs	124	161	77%
IOB Flip Flops	86		
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	3.22		

Fonte: Autoria própria

Figura 4.6 – Reporte de potência da Spartan 3 utilizada do *software* ISE 14.7

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.024	1	---	---
Logic	0.012	6469	22528	29
Signals	0.028	8038	---	---
IOs	0.007	124	161	77
Leakage	0.056			
Total	0.126			

Fonte: Autoria própria

Para a Spartan 6 tem-se na Figura 4.7 o consumo de potência e no Apêndice A tem-se o relatório da utilização dos CLBs pelo circuito. Pode-se observar que a corrente de *leakage* ainda é alta, sendo responsável por 40 % do consumo da potência.

Figura 4.7 – Reporte de potência da Spartan 6 utilizada do *software* ISE 14.7

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.011	1	---	---
Logic	0.020	4734	46648	10
Signals	0.035	6553	---	---
IOs	0.035	124	328	38
Leakage	0.067			
Total	0.167			

Fonte: Autoria própria

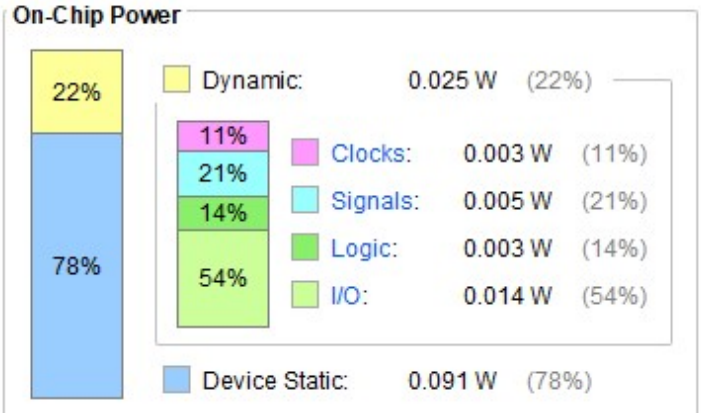
Para a Artix 7 tem-se o relatório de utilização dos blocos internos, potência e tempo nas Figuras 4.8, 4.9 e 4.10, respectivamente. No relatório de potência observa-se um aumento considerável do consumo estático do circuito (78 %) em relação as Spartan 3 e 6 (aproximadamente 45 %).

Figura 4.8 – Reporte de área ocupada do *software* Vivado 2021.1

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
N ECC_top	1960	727	5	591	1960	124	1
Coder (encoder)	109	0	1	45	109	0	0
Deco (decoder)	813	462	0	311	813	0	0
DMA_Block (DMA)	892	175	4	307	892	0	0

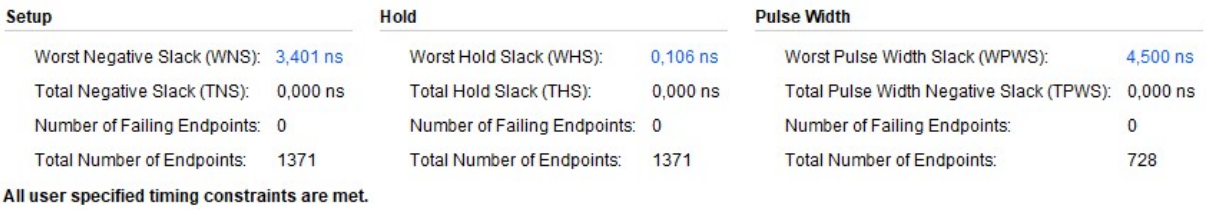
Fonte: Autoria própria

Figura 4.9 – Reporte de consumo de potência do *software* Vivado 2021.1



Fonte: Autoria própria

Figura 4.10 – Reporte de frequência do *software* Vivado 2021.1



Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,401 ns	Worst Hold Slack (WHS):	0,106 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1371	Total Number of Endpoints:	1371	Total Number of Endpoints:	728

All user specified timing constraints are met.

Fonte: Autoria própria

4.2.2 ASIC

Já para ASIC optou-se pela utilização de 3 processos de síntese, o da XFab sem modificações, sendo ela a "XH018 (Rápida)", que consiste na síntese de forma que o circuito opere na maior frequência possível considerando o *worst corner*, tendo no Apêndice C os relatórios de área, tempo e potência gerados pelo *software* da Cadence.

A "XH018 (RH)", que consiste na síntese utilizando a biblioteca de células da SMDH com transistores resistentes a radiação (VELAZCO; MCMORROW; ESTELA, 2019) . Sua frequência é menor por conter células diferentes das convencionais, todavia garante-se que durante o processo de correção e armazenamento do dado não ocorram SEUs no ECC, permitindo uma maior confiabilidade no circuito (Apêndice D).

E a "XH018"é o resultado da síntese utilizando o *design kit* sem modificações da XFab e frequência igual a XH018 (RH). Sendo utilizada para a realização das comparações de área, potência e frequência máxima do circuito resistente e do não resistente (Apêndice B).

4.2.3 Resumo das sínteses

Na Tabela 4.2 tem-se os resultados das implementações em FPGA e ASIC (considerando a probabilidade de chaveamento padrão das portas lógicas de 50 %).

Como resultado para implementação do circuito na Spartan 3 teve-se a ocupação das LUTs em 100 % comprometendo a utilização das mesmas para a implementação de uma CPU na mesma FPGA. Além do mais, a frequência obtida foi inferior a 17 MHz, com um consumo de 126 mW. Sendo assim, conclui-se que a implementação deste circuito compromete a frequência de operação do sistema.

Já, para implementação do circuito na Spartan 6, obteve-se a ocupação em 66 % das *slices* com registrador. Além do mais, a frequência obtida foi inferior a 32 MHz, com um

Tecnologia	Freq [MHz]	Potência [mW]	Área
Spartan3	16,95	126	Figura 4.5
Spartan6	31,25	167	Apêndice A
Artix7	50	116	Figura 4.8
XH018(Rápida)	71,43	66,39	0,336 mm^2
XH018	58,82	24,42	0,241 mm^2
XH018(RH)	58,82	17,90	0,182 mm^2

Tabela 4.2 – Tabela de resultados

consumo de 167 mW. Neste caso, observa-se taxa de ocupação elevada, sendo limitada a implementação em conjunto com CPU na mesma FPGA.

Por fim, como resultado da implementação na Artix 7, teve-se uma taxa de ocupação irrisória, manteve-se uma frequência de operação de 50 MHz e consumo de 90 mW. Sendo assim, dentre os resultados das implementações em FPGA, observa-se a Artix 7 como a melhor opção para a implementação.

Em relação aos resultados da síntese do ASIC tem-se para o XH018 Rápido uma frequência 43 % mais elevada em relação a Artix 7 e um consumo de potência 37 % inferior. Demonstrando que apesar da Artix ser fabricado em 28 nm sua performance é inferior e seu consumo quase o dobro ao ASIC desenvolvido em 180 nm.

Comparando-se os ASICs pode-se observar que a versão com RH tem 80 % da performance em comparação com sua versão mais rápida. Como esperado, as versões RH e normal tiveram o consumo e a área inferiores a versão rápida.

5 CONCLUSÃO

Ao decorrer deste trabalho foram realizados o estudo do algoritmo 4MBU-Rohde, sua implementação em VHDL, a análise da performance e as sínteses para FPGAs da Xilinx e para os *design kit* XFab XH018 com e sem RHBD da SMDH.

Conclui-se que na Spartan 3 é inviável a implementação do ECC e processador devido a ocupação de 100 % das LUTs somente pelo ECC. A Spartan 6 apresentou ocupação superior a 50 %, todavia, entende-se que se realizadas otimizações do código do ECC para redução de utilização de LUTs torna-se viável sua implementação.

Já a Artix 7 apresentou ótimo desempenho por executar a 50 MHz, um baixo consumo de potência (90 mW) e uma ocupação de 3 % das LUTs. Tornando-se uma ótima candidata a realização de testes de exposição a radiação.

As ASICs apresentaram resultados superiores as FPGAs, sendo a versão RH com 80 % da performance da versão Rápida e com um consumo 63 % menor de potência. Isto demonstra que a versão RH aproxima-se a versão Rápida, porém com a vantagem de ser tolerante.

Ao comparar a performance destes ASICs RH com a Artix 7 percebe-se que o primeiro apresenta frequência de execução de 17 % superior, todavia devido a toda complexidade associada ao desenvolvimento de um ASIC compreende-se que a implementação em Artix 7 é viável.

Sendo assim, pode-se concluir que através deste trabalho que o circuito 4MBU-Rohde não é simples, necessitando de diversos ciclos para realizar a aquisição e armazenamento do dado na memória. Também conclui-se que apenas a implementação em FPGA a partir da Artix 7 é viável.

Como trabalhos futuros, sugere-se a realização de uma análise para identificar o motivo pela qual a área e a potência do circuito RH é inferior ao normal, que utiliza o processo convencional da XFab. Além disso, realizar a otimização do código VHDL para redução do número de ciclos usados. E executar teste de radiação com o ECC para sua validação.

REFERÊNCIAS BIBLIOGRÁFICAS

Amagasaki, M. et al. An area compact soft error resident circuit for fpga. In: **2016 International Conference on IC Design and Technology (ICICDT)**. [S.l.: s.n.], 2016. p. 1–4.

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

AZAMBUJA, J.; KASTENSMIDT, F.; BECKER, J. **Hybrid fault tolerance techniques to detect transient faults in embedded processors**. Cham: Springer, 2014. ISBN 9783319063409.

CENTRO DE GESTÃO E ESTUDOS ESTRATÉGICOS. **CubeSats**: Resumo executivo. CGEE, 2018. Produzido. Disponível em: <https://www.cgee.org.br/documents/10195/734063/CGEE_resumoexecutivo_CubeSats_Web.pdf>.

HILLIER, C.; BALYAN, V. Error detection and correction on-board nanosatellites using hamming codes. **Journal of Electrical and Computer Engineering**, Hindawi, v. 2019, p. 3905094, Feb 2019. ISSN 2090-0147. Disponível em: <https://doi.org/10.1155/2019/3905094>.

JOHNSTON, A. Radiation effects in advanced microelectronics technologies. **IEEE Transactions on Nuclear Science**, v. 45, n. 3, p. 1339–1354, 1998.

LAHRICHI, A. **HEAT TRANSFER MODELING AND SIMULATION OF MASAT1**. 2017.

LEE, M. et al. Radiation-tolerance analysis of i-gate n-mosfet according to isolation oxide module in the cmos bulk process. **Microelectronic Engineering**, v. 200, p. 45–50, 2018. ISSN 0167-9317. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167931718304477>.

MILIES, C. P. Breve introdução a teoria dos códigos corretores de erros. **Colóquio de Matemática da Regiao Centro-Oeste**, 2009.

PARIS, L. de et al. **Digital IC Design Flow at IC-Brazil Training Program TC1**. [S.l.], 2015.

PRINZIE, J.; STEYAERT, M.; LEROUX, P. Radiation effects in CMOS technology. In: **Analog Circuits and Signal Processing**. Springer International Publishing, 2018. p. 1–20. Disponível em: <https://doi.org/10.1007/978-3-319-78616-2_1>.

QUISPE, R. D. C. **Single event transient effects in clock distribution networks**. 2014. Tese (Doutorado) — UFRGS, 2014.

Ravi, P. S.; Pargunarajan, K. Adaptive threshold multi bit flipping for low density parity check codes. In: **2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)**. [S.l.: s.n.], 2017. p. 294–298.

ROHDE, T. M. **UM CóDIGO CORRETOR DE ERROS MULTI BIT UPSET COM DUPLA VERIFICAçãO**. 2020. Dissertação (Mestrado) — UFSM, 2020.

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

SILVA, F. G. A. E. Um código extensível para correção de multiple bit upsets em memórias. **DETE - Dissertações defendidas na UFC**, 2018. Disponível em: <http://www.repositorio.ufc.br/handle/riufc/34699>.

SINGH, T.; PASHAIE, F.; KUMAR, R. Redundancy based design and analysis of alu circuit using cmos 180nm process technology for fault tolerant computing architectures. **International Journal of Computing and Digital Systems**, v. 4, p. 53–62, 01 2015.

SLAYMAN, C. Soft error trends and mitigation techniques in memory devices. In: **2011 Proceedings - Annual Reliability and Maintainability Symposium**. [S.l.: s.n.], 2011. p. 1–5. ISSN 0149-144X.

TONFAT, J. et al. Analyzing the influence of the angles of incidence and rotation on mbu events induced by low let heavy ions in a 28-nm sram-based fpga. **IEEE Transactions on Nuclear Science**, v. 64, n. 8, p. 2161–2168, 2017.

TRAVIS, G. **How the Boeing 737 Max Disaster Looks to a Software Developer**. 2019. <https://spectrum.ieee.org/how-the-boeing-737-max-disaster-looks-to-a-software-developer>. Disponível em: <https://spectrum.ieee.org/how-the-boeing-737-max-disaster-looks-to-a-software-developer>.

VAZ, P. I. et al. Improving TID radiation robustness of a CMOS OxRAM-based neuron circuit by using enclosed layout transistors. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, Institute of Electrical and Electronics Engineers (IEEE), v. 29, n. 6, p. 1122–1131, jun. 2021. Disponível em: <https://doi.org/10.1109/tvlsi.2021.3067446>.

VELAZCO, R.; MCMORROW, D.; ESTELA, J. (Ed.). **Radiation Effects on Integrated Circuits and Systems for Space Applications**. Springer International Publishing, 2019. 277–300 p. Disponível em: <https://doi.org/10.1007/978-3-030-04660-6>.

XILINX. **Extended Spartan-3A Family Overview**. [S.l.], 2011. V1.1.

____. **Spartan-6 Family Overview**. [S.l.], 2011. V2.0.

____. **7 Series FPGAs Data Sheet: Overview**. [S.l.], 2020. V2.6.1.

APÊNDICE A – OCUPAÇÃO SPARTAN 6

Device Utilization Summary:

Slice Logic Utilization:				
Number of Slice Registers:	1,737	out of	93,296	1%
Number used as Flip Flops:	1,737			
Number used as Latches:	0			
Number used as Latch-thrus:	0			
Number used as AND/OR logics:	0			
Number of Slice LUTs:	4,723	out of	46,648	10%
Number used as logic:	4,720	out of	46,648	10%
Number using O6 output only:	4,015			
Number using O5 output only:	0			
Number using O5 and O6:	705			
Number used as ROM:	0			
Number used as Memory:	0	out of	11,072	0%
Number used exclusively as route-thrus:	3			
Number with same-slice register load:	3			
Number with same-slice carry load:	0			
Number with other load:	0			
Slice Logic Distribution:				
Number of occupied Slices:	1,682	out of	11,662	14%
Number of MUXCYs used:	892	out of	23,324	3%
Number of LUT Flip Flop pairs used:	5,015			
Number with an unused Flip Flop:	3,329	out of	5,015	66%
Number with an unused LUT:	292	out of	5,015	5%
Number of fully used LUT-FF pairs:	1,394	out of	5,015	27%
Number of slice register sites lost to control set restrictions:	0	out of	93,296	0%

A LUT Flip Flop pair for this architecture represents one LUT paired with one Flip Flop within a slice. A control set is a unique combination of clock, reset, set, and enable signals for a registered element.
The Slice Logic Distribution report is not meaningful if the design is over-mapped for a non-slice resource or if Placement fails.

IO Utilization:				
Number of bonded IOBs:	124	out of	328	37%
IOB Flip Flops:	86			

Specific Feature Utilization:

Number of RAMB16BWERs:	0	out of	172	0%
Number of RAMB8BWERs:	0	out of	344	0%
Number of BUFIO2/BUFIO2_2CLKs:	0	out of	32	0%
Number of BUFIO2FB/BUFIO2FB_2CLKs:	0	out of	32	0%
Number of BUFG/BUFGMUXs:	1	out of	16	6%
Number used as BUFGs:	1			
Number used as BUFGMUX:	0			
Number of DCM/DCM_CLKGENs:	0	out of	12	0%
Number of ILOGIC2/ISERDES2s:	17	out of	442	3%
Number used as ILOGIC2s:	17			
Number used as ISERDES2s:	0			
Number of IODELAY2/IODRP2/IODRP2_MCBs:	0	out of	442	0%
Number of OLOGIC2/OSERDES2s:	69	out of	442	15%
Number used as OLOGIC2s:	69			
Number used as OSERDES2s:	0			
Number of BSCANs:	0	out of	4	0%
Number of BUFHs:	0	out of	384	0%
Number of BUFPLLs:	0	out of	8	0%
Number of BUFPLL_MCBs:	0	out of	4	0%
Number of DSP48A1s:	0	out of	132	0%
Number of ICAPs:	0	out of	1	0%
Number of MCBs:	0	out of	4	0%
Number of PCILOGICSEs:	0	out of	2	0%
Number of PLL_ADVs:	0	out of	6	0%
Number of PMVs:	0	out of	1	0%
Number of STARTUPs:	0	out of	1	0%
Number of SUSPEND_SYNCs:	0	out of	1	0%

APÊNDICE B – XFAB 180 NORMAL

```
============================================================
Generated by:           Encounter(R) RTL Compiler RC14.26
Technology library:     D_CELLS_LPMOS_CPF_slow_1_62V_125C 2.6.0
Operating conditions:   slow_1_62V_125C
Interconnect mode:      global
Area mode:              physical library
============================================================

Timing
------

Clock   Period
--------------
clk    17000.0


  Cost      Critical         Violating
 Group    Path Slack  TNS     Paths
-------------------------------------
C2C             0.8    0         0
C2O        No paths    0
clk        No paths    0
default    No paths    0
I2C             3.4    0         0
I2O        No paths    0
-------------------------------------
Total                  0         0

Instance Count
--------------
Leaf Instance Count              6707
Sequential Instance Count         722
Combinational Instance Count     5985
Hierarchical Instance Count         9

Area
----
Cell Area                          149671.015
Physical Cell Area                 0.000
Total Cell Area (Cell+Physical)    149671.015
Net Area                           91544.274
Total Area (Cell+Physical+Net)     241215.289

Max Fanout                         722 (clk)
```

Average Fanout	2.3
Terms to net ratio	3.3
Terms to instance ratio	3.5

==========

Generated by:	Encounter(R) RTL Compiler RC14.26
Technology library:	D_CELLS_LPMOS_CPF_slow_1_62V_125C 2.6.0
Operating conditions:	slow_1_62V_125C
Interconnect mode:	global
Area mode:	physical library

==========

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
ECC_top	6707	2500.432	24415854.600	24418355.032
Deco	4674	1751.430	15277145.483	15278896.912
Cval_C_par..._OP457_groupi	300	129.440	1752814.479	1752943.919
enc_csa_tr..._OP717_groupi	157	73.975	737406.545	737480.521
Pary_C_csa..._OP457_groupi	129	67.955	599548.433	599616.388
Cval_C_par..._OP412_groupi	27	16.196	197273.148	197289.345
enc_csa_tr..._OP672_groupi	24	15.909	150112.778	150128.687
Pary_C_csa..._OP412_groupi	22	14.844	118950.136	118964.980
Coder_csa_..._OP717_groupi	155	69.965	818488.017	818557.982
Coder_csa_..._OP672_groupi	22	14.447	113401.893	113416.340

APÊNDICE C – XFAB 180 NM RÁPIDO

```
============================================================
Generated by:           Encounter(R) RTL Compiler RC14.26
Module:                 ECC_top
Technology library:     D_CELLS_LPMOS_CPF_slow_1_62V_125C 2.6.0
Operating conditions:   slow_1_62V_125C
Interconnect mode:      global
Area mode:              physical library
============================================================

Timing
------

Clock   Period
-----------------
clk    14000.0


  Cost      Critical         Violating
 Group    Path Slack  TNS     Paths
-------------------------------------
C2C             0.1    0         0
C2O       No paths     0
clk       No paths     0
default   No paths     0
I2C           269.4    0         0
I2O       No paths     0
-------------------------------------
Total                  0         0

Instance Count
--------------
Leaf Instance Count              9833
Sequential Instance Count         710
Combinational Instance Count     9123
Hierarchical Instance Count         2

Area
----
Cell Area                          209354.342
Physical Cell Area                 0.000
Total Cell Area (Cell+Physical)    209354.342
Net Area                           126986.354
Total Area (Cell+Physical+Net)     336340.696
```

Max Fanout	710 (clk)
Average Fanout	2.3
Terms to net ratio	3.3
Terms to instance ratio	3.3

Generated by:	Encounter(R) RTL Compiler RC14.26
Technology library:	D_CELLS_LPMOS_CPF_slow_1_62V_125C 2.6.0
Operating conditions:	slow_1_62V_125C
Interconnect mode:	global
Area mode:	physical library

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
ECC_top	9833	3934.384	66385588.693	66389523.077
Deco	7234	2967.194	50350638.037	50353605.232
Coder_csa_..._OP672_groupi	26	15.840	225930.360	225946.200

APÊNDICE D – XFAB 180 TOLERNTE A RADIAÇÃO

```
============================================================
Generated by:           Encounter(R) RTL Compiler RC14.26
Technology library:     D_CELLS_LPMOS_CPF_slow_1_62V_125C 2.6.0
Operating conditions:   slow_1_62V_125C
Interconnect mode:      global
Area mode:              physical library
============================================================

Timing
------

Clock   Period
--------------
clk    17000.0


  Cost      Critical         Violating
 Group    Path Slack  TNS     Paths
-------------------------------------
C2C             1.5    0         0
C2O        No paths    0
clk        No paths    0
default    No paths    0
I2C            635.9   0         0
I2O        No paths    0
-------------------------------------
Total                  0         0

Instance Count
--------------
Leaf Instance Count              4955
Sequential Instance Count         722
Combinational Instance Count     4233
Hierarchical Instance Count        57

Area
----
Cell Area                      110995.063
Physical Cell Area             0.000
Total Cell Area (Cell+Physical)  110995.063
Net Area                       71349.521
Total Area (Cell+Physical+Net)  182344.584

Max Fanout                     722 (clk)
```

Average Fanout	2.7
Terms to net ratio	3.6
Terms to instance ratio	3.7

Generated by:	Encounter(R) RTL Compiler RC14.26
Technology library:	D_CELLS_LPMOS_CPF_slow_1_62V_125C 2.6.0
Operating conditions:	slow_1_62V_125C
Interconnect mode:	global
Area mode:	physical library

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
ECC_top	4955	1792.791	17895832.329	17897625.120
Deco	3144	1170.407	10505616.520	10506786.927
Cval_C_par..._OP412_groupi	0	0.000	2747.419	2747.419
Cval_C_par..._OP457_groupi	0	0.000	3923.615	3923.615
Pary_C_csa..._OP412_groupi	0	0.000	1973.767	1973.767
Pary_C_csa..._OP457_groupi	0	0.000	2848.567	2848.567
enc_csa_tr..._OP672_groupi	0	0.000	1973.767	1973.767
enc_csa_tr..._OP717_groupi	0	0.000	2597.062	2597.062
Coder_csa_..._OP672_groupi	0	0.000	2632.601	2632.601
Coder_csa_..._OP717_groupi	0	0.000	7744.714	7744.714