

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE COMPUTAÇÃO**

Liz Cristine Moreira Coutinho

**AVALIAÇÃO DO USO DE UMA REDE *INTRA-CHIP*
PARA COMUNICAÇÃO ENTRE PROCESSADORES DE
NANOSATÉLITES**

Araranguá

2019

Liz Cristine Moreira Coutinho

**AVALIAÇÃO DO USO DE UMA REDE *INTRA-CHIP*
PARA COMUNICAÇÃO ENTRE PROCESSADORES DE
NANOSATÉLITES**

Trabalho de Conclusão de Curso
submetido à Universidade Federal
de Santa Catarina, como parte dos
requisitos necessários para a obten-
ção do Grau de Bacharel em Enge-
nharia de Computação.

Orientador: Prof. Marcelo Daniel
Berejuck, Dr.

Araranguá

2019

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Coutinho, Liz Cristine Moreira
Avaliação do Uso de uma Rede Intra-Chip para
Comunicação entre Processadores de Nanossatélites /
Liz Cristine Moreira Coutinho ; orientador, Marcelo
Daniel Berejuck, 2019.
127 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Campus
Araranguá, Graduação em Engenharia de Computação,
Araranguá, 2019.

Inclui referências.

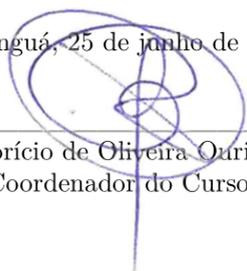
1. Engenharia de Computação. 2. Rede intra-chip.
3. Nanossatélite. 4. Código de Hamming. 5. Evento de
efeito único. I. Berejuck, Marcelo Daniel. II.
Universidade Federal de Santa Catarina. Graduação em
Engenharia de Computação. III. Título.

Liz Cristine Moreira Coutinho

**AVALIAÇÃO DO USO DE UMA REDE *INTRA-CHIP*
PARA COMUNICAÇÃO ENTRE PROCESSADORES DE
NANOSSATÉLITES**

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de “Bacharel em Engenharia de Computação” e aprovado em sua forma final pela Universidade Federal de Santa Catarina.

Araranguá, 25 de junho de 2019.



Prof. Fabrício de Oliveira Ourique, Dr.
Coordenador do Curso

Banca Examinadora:

Marcelo Daniel Berekuck  Assinado de forma digital
por Marcelo Daniel Berekuck
Dados: 2019.07.10 12:35:19
-03'00'

Prof. Marcelo Daniel Berekuck, Dr.
Universidade Federal de Santa Catarina



Prof. Tiago Oliveira Weber, Dr.
Universidade Federal de Santa Catarina



Prof. Julián Jair López Salamanca, Me.
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus pais,
Roberto e Janice.

AGRADECIMENTOS

Agradeço, primeiramente, aos meus pais Roberto Murilo Coutinho e Janice Maria Moreira Coutinho pelo vasto suporte em toda a minha trajetória. Possuo profunda gratidão a ambos, pois acreditaram no meu potencial e sempre priorizaram e incentivaram meus estudos.

Gostaria também de agradecer à minha irmã, Anne Elise Moreira Coutinho, pelos ensinamentos ao longo de minha vida e ao meu namorado, Nuno Francisco Simão, que, com companheirismo, amor e compreensão nos meus tempos de ausência e estresse, permanentemente, manteve-me motivada. Assim como, aos demais familiares por me estimularem em todas as etapas, desde a mudança de cidade até a conclusão do curso.

Devo também demonstrar minha gratidão ao meu amigo Felipe Canever e às meninas do curso de Engenharia de Computação que sempre juntaram seus esforços aos meus para atingirmos com excelência nossos objetivos e à minha amiga Adrieny Nunes, pelo apoio recíproco e incansável desde que nos conhecemos. Ainda, aos colegas da Seção de Arquitetura e Tecnologia da WEG S.A., os quais me proporcionaram uma enorme bagagem de conhecimento pessoal e profissional.

Por fim, meu eterno agradecimento à Universidade Federal de Santa Catarina pela oportunidade de realizar esse curso; aos servidores do Setor de T.I., Paulo Goulart e Willian Rochadel, por estimularem meu crescimento durante os anos de estágio; a todo o corpo docente por proverem aulas inspiradoras e motivadoras; e, especialmente, ao meu orientador, Prof. Dr. Marcelo Daniel Berejuck, pelas instruções, paciência e dedicação a fim de enriquecer este trabalho.

“Deixe o mundo um pouco melhor do que encontrou.”

(Robert Baden-Powell)

RESUMO

Nanossatélites fazem parte de uma categoria de satélites artificiais com duas características importantes: tamanho reduzido e baixo custo de matéria prima para sua construção. Eles, normalmente, possuem entre quatro e cinco placas com eletrônica embarcada, as quais controlam todas as suas funções. Uma alternativa para minimizar ainda mais seus custos é o desenvolvimento de um sistema *intra-chip* (também conhecido como *System-on-Chip* ou SoC), que pode utilizar um dispositivo lógico programável, tipo FPGA, de uso comercial (ou COTS, acrônimo do inglês *Commercial Off-The-Shelf component*). Para utilizar este tipo de dispositivo, é necessário acrescentar mecanismos que possam evitar problemas decorrentes da radiação cósmica que atinge os componentes eletrônicos, o que é comum no ambiente espacial por onde circulam os nanossatélites. Este trabalho apresenta a proposta de uma rede *intra-chip* que conecta até quatro processadores do tipo *soft-core*, formando um SoC. A rede, amparada pela técnica dos códigos de Hamming, oferece mecanismos que permitem a ela seu restabelecimento de uma falha temporária dos tipos *Single Event Upset* e *Single Event Transient*. Esta proteção foi confirmada com simulações realizadas com um *software* que possibilita a injeção de falhas, denominado ModelSim. Resultados de menor consumo de silício, concomitante a economias no projeto e redução da sensibilidade à ionização cósmica, podem ser observados após os experimentos elaborados.

Palavras-chave: Rede *intra-chip*; Nanossatélite; Código de Hamming; Evento de efeito único; Único evento alterado; Único evento transiente.

ABSTRACT

Nanosatellites are part of a category of artificial satellites with two essential characteristics: reduced size and low cost of raw material for their construction. They usually have between four and five boards with embedded electronics, which control all their functions. An alternative to further minimize its costs is the development of an intra-chip system (also known as System-on-Chip or SoC), which may use a commercially available programmable logic device (or COTS – Commercial Off-The-Shelf component), as an FPGA. To use this type of device is necessary to add mechanisms that can avoid problems arising from the interception of cosmic radiation in its electronic components, characteristic in the nanosatellites' space environment. This work presents the proposal of a Network-on-Chip (called NoC) that connects up to four soft-core processors, forming an SoC. The Network-on-Chip, supported by the Hamming codes technique, provides mechanisms that allow it to re-establish a temporary failure of the types Single Event Upset and Single Event Transient. This protection was confirmed by simulations carried out with a software that allows the injection of faults, named ModelSim. Results of lower silicon consumption, concomitant to economies in design and reduction of sensitivity to cosmic ionization, can be observed after the elaborated experiments.

Keywords: Network-on-Chip; Nanosatellite; Hamming Code; Single Event Effect; Single Event Upset; Single Event Transient.

LISTA DE FIGURAS

Figura 1	Exemplos de estruturas de comunicação em SoCs	21
Figura 2	Modelos de <i>CubeSats</i>	33
Figura 3	Camadas do Modelo OSI	36
Figura 4	Conectividade direta	36
Figura 5	Topologias de rede	38
Figura 6	Conectividade em rede anelar	38
Figura 7	Relação entre falha, erro e defeito	42
Figura 8	Partícula energética atingindo um dispositivo eletrônico	43
Figura 9	SEU em uma memória SRAM	44
Figura 10	Evento de SET em diferentes cenários de tempo	45
Figura 11	Eventos de SET e SEU em uma célula lógica de FPGA	46
Figura 12	Modelo estrutural do NanosatC-Br1	53
Figura 13	Modelo estrutural do NanosatC-Br2	54
Figura 14	Curva de envelhecimento do circuito	58
Figura 15	Foto ilustrativa do FloripaSat	62
Figura 16	Rede anelar simplificada com processadores e roteadores	63
Figura 17	Especificação dos <i>bits</i> dos <i>flits</i>	64
Figura 18	Simplificação da rede desenvolvida	67
Figura 19	Visão interna dos roteadores da rede (simplificada) . .	68
Figura 20	Multiplexador e demultiplexador	69
Figura 21	Fluxograma do Roteador 0	70
Figura 22	Diagrama de blocos de transmissão de dados	72
Figura 23	Posições de x e m	75
Figura 24	Sinais locais e de rede (Origem: 3 e Destino: 1)	78
Figura 25	Envio de dado do Roteador 0 para o Roteador 1	82
Figura 26	Envio de dado do Roteador 0 para o Roteador 2	82
Figura 27	Envio de dado do Roteador 0 para o Roteador 3	82
Figura 28	Envio de dado do Roteador 1 para o Roteador 0	83
Figura 29	Envio de dado do Roteador 1 para o Roteador 2	83
Figura 30	Envio de dado do Roteador 1 para o Roteador 3	83

Figura 31	Envio de dado do Roteador 2 para o Roteador 0.....	84
Figura 32	Envio de dado do Roteador 2 para o Roteador 1.....	84
Figura 33	Envio de dado do Roteador 2 para o Roteador 3.....	84
Figura 34	Envio de dado do Roteador 3 para o Roteador 0.....	85
Figura 35	Envio de dado do Roteador 3 para o Roteador 1.....	85
Figura 36	Envio de dado do Roteador 3 para o Roteador 2.....	85
Figura 37	Envio de 2 pacotes: R1-R0 e R3-R1	86
Figura 38	Envio de 4 pacotes: R0-R1, R1-R2, R2-R3 e R3-R0..	86
Figura 39	Envio de 4 pacotes: R0-R2, R2-R1, R1-R0 e R3-R1..	86
Figura 40	Envio de 1 pacote: R0-R2 (8 <i>flits</i>).....	88
Figura 41	Envio de 2 pacotes: R0-R2 (8 <i>flits</i>) e R2-R0 (6 <i>flits</i>)..	88
Figura 42	Circuito “votador” na implementação de Tripla Redundância de <i>Hardware</i>	89
Figura 43	Circuito “votador” – Implementação interna.....	90
Figura 44	Rede <i>intra-chip</i> com a técnica de TMR.....	90
Figura 45	Rede <i>intra-chip</i> com codificação de Hamming.....	91
Figura 46	Diagrama ilustrando um Elemento Lógico da família Cyclone IV (Fabricante: Intel)	92
Figura 47	Comparação do consumo de silício entre as redes analisadas	93
Figura 48	Comparação do consumo de silício dos componentes internos da rede com Hamming	93
Figura 49	Injeção de falha na saída do Roteador 1	95
Figura 50	Injeção de falha na saída do Roteador 0	95
Figura 51	Injeção de falha na saída do Roteador 2	96
Figura 52	Injeção de falha na saída do Roteador 3	97
Figura 53	Rede implementada	113
Figura 54	Interior do Roteador 0 da rede implementada.....	114
Figura 55	Estados e suas transições.....	115

LISTA DE TABELAS

Tabela 1	Categorização satelital por massa	31
Tabela 2	Conceitos abordados	59
Tabela 3	Especificação dos <i>bits</i> mais significativos	64
Tabela 4	Exemplo de envio de <i>flits</i>	65
Tabela 5	Entradas e saídas do Roteador 0	66
Tabela 6	Potências de 2 com resultado no intervalo de 1 a 32 .	74
Tabela 7	Definição de m em relação a x	75
Tabela 8	Definição de x em relação a m	76
Tabela 9	Lógica programável <i>XOR</i>	77
Tabela 10	Definição de P em relação a x e m	77
Tabela 11	Pacote da Tabela 4 – Decodificado e Codificado	79
Tabela 12	Pacotes das Figuras 25, 26 e 27 (hexadecimal e binário)	82
Tabela 13	Pacotes das Figuras 28, 29 e 30 (hexadecimal e binário)	83
Tabela 14	Pacotes das Figuras 31, 32 e 33 (hexadecimal e binário)	84
Tabela 15	Pacotes das Figuras 34, 35 e 36 (hexadecimal e binário)	85
Tabela 16	Consumo de silício para os três modelos de rede <i>intra-chip</i> : normal, com TMR e com Hamming	92
Tabela 17	Consumo de silício para os componentes internos da rede com Hamming	93
Tabela 18	Injeção de falha no 8º <i>bit</i>	94
Tabela 19	Injeção de falha no 10º <i>bit</i>	95
Tabela 20	Injeção de falha no 19º <i>bit</i>	97
Tabela 21	Injeção de falha no 38º <i>bit</i>	97
Tabela 22	Condições para mudança de estado no Roteador 0 . . .	116
Tabela 23	Condições para mudança de estado no Roteador 1 . . .	119
Tabela 24	Condições para mudança de estado no Roteador 2 . . .	122
Tabela 25	Condições para mudança de estado no Roteador 3 . . .	125

LISTA DE ABREVIATURAS E SIGLAS

ALU	<i>Arithmetic Logic Unit</i>
ASIC	<i>Application Specific Integrated Circuit</i>
BER	<i>Backward Error Recovery</i>
BIST	<i>Built-In Self-Test</i>
BRAM	<i>Block Random Access Memory</i>
CalPoly	<i>California Polytechnic State University</i>
CI	<i>Circuito Integrado</i>
COTS	<i>Commercial Off-the-shelf Components</i>
CPU	<i>Central Process Unit</i>
CRAM	<i>Configuration Random Access Memory</i>
CR	<i>Checkpoint and Recovery</i>
CRC	<i>Cyclic Redundancy Check</i>
DDS	<i>Differential Delay Sensor</i>
Demux	<i>Demultiplexer</i>
DPR	<i>Dynamic Partially Reconfigurable</i>
ECC	<i>Error Correction Code</i>
EDC	<i>Error Detection and Correction</i>
EPS	<i>Electric Power System</i>
ESA	<i>European Space Agency</i>
FDIR	<i>Fault Detection, Fault Isolation and Recovery</i>
FER	<i>Forward Error Recovery</i>
Flit	<i>Flow Control Unit</i>
FPGA	<i>Field Programmable Gate Array</i>
GALS	<i>Globally Asynchronous Locally Synchronous</i>
HDL	<i>Hardware Description Language</i>
INPE	<i>Instituto Nacional de Pesquisas Espaciais</i>
ISO	<i>International Organization for Standardization</i>
LE	<i>Logic Elements</i>
LISHA	<i>Software/Hardware Integration Lab</i>
LUT	<i>Look Up Table</i>
MBU	<i>Multiple Bit Upset</i>
Mux	<i>Multiplexer</i>

NASA	<i>National Aeronautics and Space Administration</i>
NI	<i>Network Interface</i>
NoC	<i>Network-on-Chip</i>
OBC	<i>On-Board Computers</i>
OBDH	<i>On-Board Data Handling</i>
OSI	<i>Open System Interconnection</i>
PC	<i>Program Counter</i>
P&D	<i>Pesquisa e Desenvolvimento</i>
RP	<i>Reconfigurable Partition</i>
RTL	<i>Register Transfer Level</i>
SBU	<i>Single Bit Upset</i>
SEE	<i>Single Event Effect</i>
SET	<i>Single Event Transient</i>
SEU	<i>Single Event Upset</i>
SM	<i>State Machine</i>
SoC	<i>System-on-Chip</i>
SpaceLab	<i>Lab. de Sistemas Embarcados para Aplicações Espaciais</i>
SRAM	<i>Static Random Access Memory</i>
TID	<i>Total Ionizing Dose</i>
TMR	<i>Triple Modular Redundancy</i>
TR	<i>Time Redundant</i>
TT&C	<i>Telemetry, Tracking and Command</i>
UFSC	<i>Universidade Federal de Santa Catarina</i>
UFMS	<i>Universidade Federal de Santa Maria</i>
VHDL	<i>VHSIC Hardware Description Language</i>
VHSIC	<i>Very High Speed Integrated Circuits</i>

SUMÁRIO

1	INTRODUÇÃO	21
1.1	PROBLEMÁTICA	23
1.2	OBJETIVOS	25
1.2.1	Objetivo Geral	25
1.2.2	Objetivos Específicos	25
1.3	JUSTIFICATIVA E MOTIVAÇÃO	26
1.4	PROCEDIMENTOS METODOLÓGICOS	27
1.5	ORGANIZAÇÃO DO TRABALHO	28
2	FUNDAMENTAÇÃO TEÓRICA	31
2.1	PEQUENOS SATÉLITES	31
2.1.1	Nanossatélites	32
2.2	REDES <i>INTRA-CHIP</i>	34
2.2.1	Comunicação em camadas	35
2.2.2	Topologias de rede	37
2.2.3	Modos de chaveamento	39
2.2.4	Roteamento dos dados	40
2.2.5	Vantagens técnicas	40
2.3	FALHAS EM DISPOSITIVOS ELETRÔNICOS	41
2.3.1	Efeitos da radiação em dispositivos eletrônicos	42
2.3.2	<i>Single Event Upset (SEU)</i>	44
2.3.3	<i>Single Event Transient (SET)</i>	44
2.3.4	Erros “leves” (<i>soft errors</i>) em FPGAs	45
2.4	TÉCNICAS DE TOLERÂNCIA A FALHAS DEVIDO À RADIAÇÃO	47
2.4.1	Detecção e Correção	48
2.4.1.1	Redundância de <i>Hardware</i>	48
2.4.1.2	Redundância de <i>Software</i>	49
2.4.1.3	Redundância de Informação	49
2.4.1.4	Redundância Temporal	50
2.4.2	Recuperação	50
2.4.3	Diagnósticos e Reparo	51
3	TRABALHOS RELACIONADOS	53
4	PROJETO DA REDE	61
4.1	PROJETO LÓGICO	62
4.1.1	Topologia	62
4.1.2	Formato do pacote	64
4.1.3	Canais físicos	65

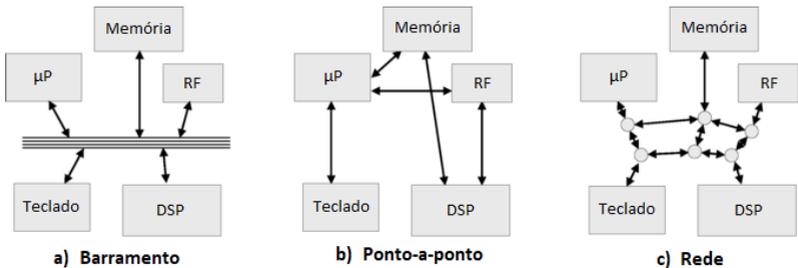
4.1.4	Roteador	67
4.1.5	Módulo Hamming	72
5	RESULTADOS EXPERIMENTAIS	81
5.1	AVALIAÇÃO DE TRÁFEGO E LATÊNCIA	87
5.2	CONSUMO DE SILÍCIO	88
5.2.1	Implementação com TMR	89
5.2.2	Implementação com código de Hamming	91
5.2.3	Consumo de silício das duas soluções	91
5.3	SIMULAÇÃO DE FALHAS COM MODELSIM	94
6	CONSIDERAÇÕES FINAIS	99
6.1	TRABALHOS FUTUROS	101
	REFERÊNCIAS	103
	APÊNDICE A – Rede e Roteadores – Visualização completa	113

1 INTRODUÇÃO

Nos anos 90, quando se introduziu o conceito de SoC, sigla em inglês de “sistemas *intra-chip*”, e até hoje em alguns *chips*, as estruturas de comunicação têm sido, usualmente, caracterizadas por projetos *ad hoc* com misturas de barramentos e *links* ponto-a-ponto (LAHIRI; RAGHUNATHAN; DEY, 2001). Como à medida que surgiram novos avanços tecnológicos, permitiu-se a inserção de um grande número de transistores em apenas uma única pastilha de silício, tem-se a necessidade de repensar novos métodos de projetos para conexões mais eficientes.

Desta forma, com o advento de CPUs (*Central Process Unit*) *multi-core* e a tendência previsível para arquiteturas de múltiplos núcleos massivamente integrados, como em grandes SoCs que interconectam suas dezenas de núcleos de processamento, surgiu uma nova abordagem para superar as limitações das presentes estruturas: implementar uma *Network-on-Chip* (RADETZKI et al., 2013; ZIMMER; JANTSCH, 2003). Esta terminologia foi apresentada por Hemani et al. (2000) e, em português, pode ser proferida como Rede-em-Chip, rede *intra-chip* ou NoC. Ela é vista como um conceito unificador e não como uma nova alternativa explícita e possui a vantagem de maior desempenho agregado, explorando a operação paralela (BJERREGAARD; MAHADEVAN, 2006).

Figura 1 – Exemplos de estruturas de comunicação em SoCs



Fonte: Adaptado de Bjerregaard e Mahadevan (2006)

A Figura 1 ilustra alguns tipos – mencionados – de estruturas de comunicação básicas em um SoC. Os barramentos são fáceis de modelar, pois fundamentam-se em conceitos satisfatoriamente compreendidos e servem como uma solução intermediária. Não obstante, em sistemas com múltiplos núcleos altamente interconectados, fácil e rapidamente, eles são capazes de se tornar um gargalo na comunicação, já que são

limitadamente escaláveis (BJERREGAARD; MAHADEVAN, 2006). Além disso, o uso de linhas globais em um circuito integrado de ordem submicrônica estabelece restrições críticas ao desempenho do sistema, devido às altas capacitâncias e resistências parasitas, características dos fios longos.

Os *links* ponto-a-ponto dedicados são ideais quando se pode projetá-los de acordo com a disponibilidade de largura de banda, latência e uso de energia. No entanto, o número de *links* necessários, isto é, fios em torno do núcleo, aumenta exponencialmente ao passo que o número de núcleos e a complexidade do sistema crescem.

Para máxima flexibilidade e escalabilidade, deve-se implementar uma estrutura de comunicação global compartilhada e segmentada. Ou seja, uma NoC, a qual contém uma rede de roteamento de dados composta por *links* de comunicação e nós de roteamento (roteadores), implementados no *chip*. Seus dados são transmitidos via pacotes divididos em unidades atômicas, intituladas de *flits* (*Flow Control Unit*). Em comparação com as demais, essa se adapta bem ao tamanho e à complexidade do circuito (BJERREGAARD; MAHADEVAN, 2006).

As NoCs, utilizadas atualmente em sistemas embarcados, além de possuírem uma arquitetura reutilizável e escalável, podem prover o desempenho conjecturado, concomitante a um baixo consumo de energia (WACHTER et al., 2013). Cada roteador possui um conjunto de portas usadas para conexão com seus roteadores vizinhos e uma porta de comunicação com o núcleo de processamento, através de uma interface de rede (*Network Interface* – NI). Os pacotes são enviados e recebidos pelos roteadores, conforme as NoCs utilizam o modelo de comunicação de transmissão de mensagens, até atingir o seu destino (BENINI; MICHELI, 2002).

Publicaram-se pesquisas e livros didáticos sobre o tema, por exemplo Ogras, Hu e Marculescu (2005), Bjerregaard e Mahadevan (2006), Micheli e Benini (2006). Mesmo assim, ainda há diversas questões de pesquisa em aberto sobre os tópicos mais avançados de NoC, como visto em Owens et al. (2007) e Marculescu et al. (2009). As tendências de pesquisa mais recentes abrangem, principalmente, o *design* de NoCs tolerantes a falhas, visto que é um assunto relevante e inevitável, segundo tendências tecnológicas e artigos publicados.

A manipulação de grandes taxas de dados entre núcleos demanda alta atividade do circuito. O calor estimula os processos de envelhecimento e desgaste dos componentes, ainda mais se tratando de circuitos miniaturizados. Ademais, comunicações em SoC podem sofrer os efeitos

de *crosstalk*¹ e radiação, tornando-se suscetíveis a variações causadas por influências de produção ou ambientais, uma vez que a diminuição contínua no tamanho dos componentes e o aumento da frequência os torna mais vulneráveis (RADETZKI et al., 2013).

O fortalecimento do baixo nível, por si só, não será eficaz na solução desses problemas. É preciso aprender a construir sistemas confiáveis a partir de componentes não confiáveis, sem introduzir sobrecarga excessiva (BORKAR, 2005). Dado que NoCs propiciam caminhos de comunicação inerentemente redundantes, são uma ótima escolha contra falhas parciais. Para isto, deve-se investigar todas as camadas de rede e escolher modelos adequados para detecção de erros, procedimentos de diagnóstico, tolerância a falhas e reconfiguração (RADETZKI et al., 2013).

Resumindo, portanto, uma NoC tolerante a falhas deve ser capaz de detectar a ocorrência de uma falha e evitar que o erro resultante cause um mau funcionamento na aplicação. Todavia, esta confiabilidade necessária afeta o desempenho e resulta em aumento nos custos com silício (BERTOZZI, 2006).

1.1 PROBLEMÁTICA

Comunicações via satélite são o resultado de pesquisas na área de comunicações e tecnologias espaciais, cujo objetivo é atingir cada vez maiores faixas orbitais e capacidades de processamento com os menores custos possíveis (MARAL; BOUSQUET, 2011). No entanto, satélites grandes e pesados requerem foguetes maiores e altas quantias de dinheiro para o seu lançamento. Em razão disso, o desenvolvimento na área espacial sempre foi restrito a poucos países e sua exploração com objetivos comerciais ainda é um privilégio ao alcance desta minoria (COSTA, 2004).

Neste contexto, surgiram os nanossatélites, que vêm alcançando grandes mercados nos últimos anos. Com lançamento consideravelmente barato e complexidade reduzida, eles são uma ótima ferramenta para aprendizado (CASARIL; SOUZA; BRITO, 2018). Ainda, podem ser utilizados em constelações envolvendo comunicação com baixo fluxo de dados, formações para colher dados de múltiplos pontos, missões que visam inspeção orbital de outros satélites e pesquisas universitárias, como se vê no trabalho de Carvalho et al. (2012).

¹Interferência indesejada que um canal de transmissão causa em outro circuito ou canal.

Os dispositivos *off-the-shelf* (*Commercial Off-The-Shelf components - COTS*), ou “prontos para uso” em tradução livre, são uma alternativa ao desenvolvimento completo de nanossatélites. Contudo, estes aparelhos disponíveis comercialmente são produzidos em série, para pronta entrega, e normalmente deixam a desejar em quesitos de tolerância a falhas, pois não são preparados para situações de radiação cósmica ou ambientes agressivos em termos de radiação, aos quais serão submetidos. Desta forma, vale um estudo mais profundo em tópicos relacionados a nanossatélites em P&D (pesquisa e desenvolvimento), a fim de se conseguir criar e implementar soluções mais seguras e válidas.

Para que um nanossatélite seja considerado confiável, suas estruturas devem aplicar técnicas de tolerância a falhas e manter o sistema funcional na ocorrência de erros. Em sistemas *multi-core*, efetua-se a confiabilidade pelos elementos de processamento e pela arquitetura de interconexão. À vista disso, estudam-se as falhas comumente associadas ao sistema em questão e, após implementar as técnicas de tolerância apropriadas, ele passa por uma bateria de testes com o propósito de validá-lo em situações adversas.

Falhas permanentes ocorrem uma vez e persistem a partir desse momento, geralmente por algum transtorno na fabricação de um componente ou dano sofrido durante a instalação ou conservação do equipamento que o contém. Apesar de esta falha poder permanecer no sistema durante toda sua vida útil sem nunca se manifestar, quando ela o leva a um estado errôneo, pode causar seriíssima desordem. Manifestam-se como um erro repetido, a menos que o elemento anômalo seja reparado.

Como Weber (2003) menciona, há facilidade de reconhecimento de erros na fase de teste do sistema e boa tolerância a falhas deste tipo. Entretanto, concomitantemente, há um grande custo de desenvolvimento e manutenção de sistemas que conseguem se recuperar dessas falhas, visto que existe elevada complexidade para conseguir se reestruturar enquanto analisa os componentes defeituosos.

Igualmente, é importante frisar a ocorrência das falhas temporárias, também conhecidas como transitórias ou transientes, as quais ocorrem uma vez (devido a condições ambientais temporárias) e não persistem. Suas causas frequentemente notáveis são a radiação de partículas produzidas quando os raios cósmicos afetam a atmosfera e as partículas alfa, produzidas pelo decaimento natural dos isótopos radioativos. Além de este tipo de falha ser mais fácil de se detectar em execução, seu custo de reparo é mais barato em comparação com as falhas permanentes.

Ainda, existem as falhas intermitentes, que ocorrem repetida-

mente, mas não continuamente. Tais falhas mencionadas acontecem por causas diferentes, mas podem ter abordagens de detecção e correção semelhantes, com técnicas e complexidades distintas.

O foco dessa pesquisa é em redes *intra-chip* tolerantes a falhas temporárias para uso em nanossatélites. Por conseguinte, este Trabalho de Conclusão de Curso visa responder as seguintes questões de investigação:

- É possível estabelecer uma NoC com uma estrutura simples e que possa ser tolerante a falhas temporárias em ambiente espacial?
- É possível aplicar esta solução em dispositivos *off-the-shelf* para a construção de nanossatélites?
- Qual o consumo de silício para uma solução com proteção a falhas temporárias em nanossatélites com a arquitetura de NoC proposta?
- Qual a sua eficiência quando comparada com outras soluções da literatura?

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O principal objetivo deste trabalho é desenvolver uma rede *intra-chip* para interconectar até quatro processadores *soft-core* que operarão no controle de nanossatélites. Esta rede deverá apresentar mecanismos que permitam a ela seu restabelecimento de uma falha temporária causada por radiação.

1.2.2 Objetivos Específicos

- Avaliar o uso de rede *intra-chip* para interconexão de núcleos em um SoC.
- Identificar os mecanismos de tolerância a falhas disponíveis para sistemas que operam em ambiente com níveis críticos de radiação.
- Elaborar e implementar uma rede *intra-chip* com baixo consumo de silício, em relação a projetos de semelhante propósito.

- Garantir que o sistema funcione com baixa sensibilidade à ionização cósmica.
- Avaliar e discutir os resultados obtidos a partir do modelo proposto e comparar com outras soluções da literatura.

1.3 JUSTIFICATIVA E MOTIVAÇÃO

Em aplicações aeroespaciais, percebe-se que existe um empecilho quando se trata da troca de dados entre computadores de bordo (OBCs – *On-Board Computers*) e estações terrestres, pois há uma pequena janela de comunicação disponível. A redução no tamanho físico dos transistores os tornou mais rápidos e com menor consumo de potência. Conseqüentemente, com o avanço da tecnologia, foi possível aumentar a capacidade de processamento disponível em OBCs, uma vez que se valorizou a integração de seus componentes.

Suas arquiteturas de comunicação contêm cada vez mais núcleos de processamento, o que leva a sempre ter de aprimorar seus canais para melhor adequação. No entanto, alguns trabalhos encontrados na literatura não satisfazem os requisitos de comunicação de OBCs com múltiplos núcleos idealizados para o futuro (WALTERS et al., 2011).

A utilização de NoCs, paulatinamente, vem substituindo os barramentos de interconexão no contexto *multi-core*. Apesar do considerável custo e necessidade de técnicas de tolerância a falhas para comprovar a confiabilidade, é uma solução eficiente para oferecer desempenho de SoCs, diminuindo o nível de ruído na comunicação e aumentando o desempenho em termos de capacidade de processamento.

Os subsistemas de barramentos oferecem sobrecarga de *hardware* reduzida e bom rendimento de dados. Em comparação, NoCs constituem, de longe, o maior subsistema coeso em arquiteturas com muitos núcleos (ANGIOLINI et al., 2006; LEE et al., 2007). Ao ajustar os parâmetros, estas podem ser adaptadas de maneira flexível aos requisitos de aplicação. Contudo, tamanho, complexidade e densidade de integração farão com que as futuras NoCs sejam cada vez mais vulneráveis (RADETZKI et al., 2013).

A presente proposta é motivada pela necessidade de desenvolver uma arquitetura de comunicação confiável para a integração de núcleos em um único *chip* perante sistemas de nanossatélite. Justifica-se sua pesquisa por abordar aspectos de confiabilidade em sistemas aeroespaciais, assunto de interesse atual na comunidade científica, e sua implementação pela carência de técnicas e tecnologias de projeto e avaliação

de circuitos nesse âmbito.

Esta proposta vai ao encontro de alguns trabalhos já desenvolvidos e também em execução no Projeto FloripaSat², da Universidade Federal de Santa Catarina (UFSC). Portanto, o projeto é uma colaboração entre o Laboratório de Integração de *Software* e *Hardware* (LISHA), o Laboratório de Sistemas Embarcados para Aplicações Espaciais (SpaceLab), ambos da UFSC, e a Agência Espacial Europeia (*European Space Agency* – ESA).

1.4 PROCEDIMENTOS METODOLÓGICOS

Define-se este trabalho como uma pesquisa aplicada, uma vez que, segundo Matias-Pereira (2012), “os conhecimentos adquiridos são utilizados para aplicação prática e voltados para a solução de problemas concretos da vida moderna”. Como se faz necessário o levantamento do estado da arte do tema, fundamentação teórica e definição da contribuição do trabalho, seu objeto é uma pesquisa bibliográfica. E, por fim, enquadra-se na modalidade “Pesquisa Tecnológica”, pois será criado um equipamento tecnológico, no qual o foco da pesquisa incumbirá o roteamento de dados em um *chip multi-core*.

Para atingir os objetivos apresentados na Seção 1.2, a metodologia de desenvolvimento desta Pesquisa Bibliográfica e Tecnológica Aplicada está dividida em 9 etapas, cronologicamente, conforme segue:

- Etapa 1: Análise e definição do escopo do trabalho.
- Etapa 2: Levantamento bibliográfico sobre NoCs, SoCs e suas tolerâncias a falhas.
- Etapa 3: Elaboração e desenvolvimento de um projeto de rede *intra-chip* tolerante a falhas, que roteie dados enviados por processadores fictícios.
- Etapa 4: Implementação da rede em um protótipo com sistema funcional e *soft-core*.
- Etapa 5: Criação de um cenário de testes para avaliar o protótipo.
- Etapa 6: Avaliação e discussão dos resultados obtidos no cenário proposto.
- Etapa 7: Com o sistema finalizado, validado e avaliado, propõe-se uma discussão e comparação dos resultados com projetos que utilizam topologias diferentes.
- Etapa 8: Escrita de um artigo científico, apresentando resultados

²Disponível em: <https://floripasat.ufsc.br/>.

do trabalho.

- Etapa 9: Elaboração, escrita e apresentação do Trabalho de Conclusão de Curso (TCC).

1.5 ORGANIZAÇÃO DO TRABALHO

O presente trabalho é dividido em 6 capítulos. O **Capítulo 1** expõe uma introdução do estado da arte das áreas abrangidas, assim como a problemática, os objetivos gerais e específicos, a justificativa e os procedimentos metodológicos do trabalho.

O **Capítulo 2** é uma revisão de conceitos abordados no decorrer do trabalho. Explicações sobre pequenos satélites, redes *intra-chip*, falhas em dispositivos eletrônicos e técnicas de tolerância a falhas devido à radiação são as principais questões exploradas. Não se pretende cobrir a tolerância a falhas em geral, foca-se especificamente em NoCs, mais precisamente eletrônicas e de fabricação em 2D.

O **Capítulo 3** aborda outros trabalhos que vão ao encontro do tema deste, visando complementar os estudos presentes. Alguns trabalhos discorrem sobre projetos de nanossatélites, outros sobre falhas causadas por radiação e outros sobre redes *intra-chip*. Não há uma composição que unifique todas as questões englobadas no desenvolvimento deste trabalho de conclusão de curso, portanto, entende-se que é um assunto contemporâneo e sujeito a melhorias.

O **Capítulo 4** detalha o sistema proposto, exibindo uma perspectiva lógica e uma física, de modo que se consiga compreender como os componentes se comportam e se comunicam nas suas interações necessárias. Além disto, há a explicação do formato padronizado para os pacotes que transitam na rede e dos códigos de detecção e correção de erros.

O **Capítulo 5** ilustra o cenário criado e os resultados obtidos a partir da aplicação da proposta no sistema, inclusive com injeção de falhas nas simulações da rede. Avaliam-se questões de tráfego e latência da rede desenvolvida, do consumo de silício em comparação com uma rede sem técnicas de detecção e correção e com o uso de outra técnica conhecida na literatura. Figuras explicitam os testes realizados e comprovam a eficácia do método proposto.

O **Capítulo 6** expõe as conclusões obtidas a partir dos resultados, em relação com o esperado no início do trabalho e em comparação com os resultados dos trabalhos de mesmo propósito examinados. Ademais, inclui considerações finais e perspectivas para trabalhos futuros.

Além disso, no fim da composição, há o **Apêndice – A**, com figuras e tabelas elaboradas pela autora, para auxiliar na compreensão da rede proposta. Elas contêm todos os elementos e ligações, a nível de rede e roteador, bem como a ilustração da Máquina de Estados e as condições para mudança de estado em cada roteador.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 PEQUENOS SATÉLITES

Os pequenos satélites foram desenvolvidos inicialmente há mais de 50 anos, mas suas classificações consoantes aos tamanhos se deram a partir de 1992 (HELVAJIAN; JANSON, 2008). A Tabela 1 indica as categorias principais dos pequenos satélites, associados essencialmente às suas massas. Os autores dela, afirmam que “Com avanços em células solares, baterias, micro e nano eletrônica, e sensores miniaturizados, os picossatélites de hoje conseguem superar a performance dos microssatélites dos anos 1960”.

Tabela 1 – Categorização satelital por massa

Categoria	Massa
Minissatélite	100 a 500 <i>kg</i>
Microssatélite	10 a 100 <i>kg</i>
Nanossatélite	1 a 10 <i>kg</i>
Picossatélite	0,1 a 1 <i>kg</i>
Femtossatélite	0,01 a 0,1 <i>kg</i>

Fonte: Helvajian e Janson (2008)

Dados disponibilizados por Filho (2015) informam que, entre os anos de 1995 e 2014, lançaram-se 2.366 satélites, dos quais 863 eram satélites de categorias inferiores à 500 *kg*. Até janeiro de 2015, 1.266 satélites estavam funcionando em órbita da Terra, desses, 363, isto é, 28%, eram também de pequeno porte. Estes 363 pequenos satélites estavam distribuídos da seguinte maneira: 150 minissatélites, 110 microssatélites, 78 nanossatélites e 25 picossatélites (nesta última categoria, o autor desse diagnóstico engloba todos os satélites menores ou iguais a 1 *kg*).

Ainda, Filho (2015) expõe que o número de satélites orbitando no dia 31 de agosto de 2015 saltou para 1.305. Nacionalidades com predomínio em suas possessões são Estados Unidos – 549, China – 142 e Rússia – 131. Os demais 483 pertenciam a outros países e seus respectivos programas espaciais. A partir deste estudo, foi possível perceber que o monopólio da realização de atividades espaciais por nações desenvolvidas e vistas como “únicas detentoras do conhecimento tecnológico” vem diminuindo nas últimas décadas e dando espaço a outras

nações em desenvolvimento, conforme Costa (2004) também revela em seu trabalho.

Para Antunes (2017), com a progressiva dependência humana em relação à tecnologia espacial, surgiram novas alternativas para facilitar a exploração do espaço. Estas visavam evitar o congestionamento de órbitas mais baixas e proporcionar gastos e tempo de desenvolvimento inferiores – comparados aos seus irmãos de maiores proporções, dado que os orçamentos estavam em declínio no pós Guerra Fria. Shiroma et al. (2011) afirmam que muitas entidades acadêmicas, governamentais e comerciais estão realizando projetos de pequenos satélites em todo o mundo.

Efetivamente, percebe-se que a produção de pequenos satélites, com peças de menor complexidade e maior acesso (COTS¹), já ocorre há algumas décadas em nações como Estados Unidos, Reino Unido, Itália, Espanha e Canadá (ANTUNES, 2017). Outros países, como o Brasil, perceberam a oportunidade de ultrapassar limitações técnicas e financeiras nesse cenário e recentemente vêm entrando neste mercado. Além desta vantagem, pode-se acrescentar outros fatores para o aumento no interesse de *small satellites*, tais como miniaturização de tecnologias facilitadoras, necessidade de ‘plataformas de resposta rápida’² para atenuação de desastres naturais e gestão de crises e conflitos e o fascínio de lançar um satélite pessoal, como citam Shiroma et al. (2011).

2.1.1 Nanosatélites

Santos et al. (2018) expressam que nanosatélites são satélites artificiais, com dimensões, massa e tamanho reduzidos para utilização em missões específicas. Estas, utilizadas atualmente, sobretudo, para fins de pesquisa e aplicações técnico-científicas, podem ser relacionadas a comunicações, validações de tecnologias, observações do planeta Terra, sensoriamento remoto, aplicações militares e outras situações tocantes à obtenção de dados científicos, captura de imagens e realiza-

¹Componentes de *hardware* e *software* comerciais, conhecidos como componentes de “prateleira”, são padronizados de fábrica e facilmente encontrados no mercado, a qualquer tempo. Por esta razão, projetos que contam com sua implementação conseguem reduzir custos iniciais e simplificar sua estrutura (CUNHA, 2005).

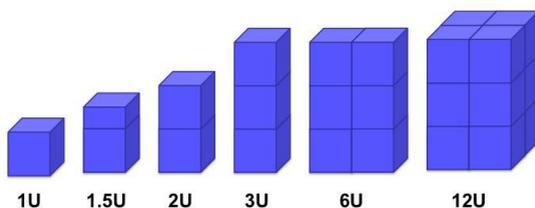
²Por exemplo, em 2007, os Estados Unidos criaram o *Operationally Responsive Space Office*, com a finalidade de garantir o desenvolvimento de novas capacidades espaciais militares que pudessem ser ligeiramente colocadas em operação. Esse tipo de serviço impõe uma nova lógica propícia à aceitação de riscos e à confiabilidade de missões espaciais (VILLELA; BRANDÃO; LEONARDI, 2016).

ção de experimentos, como é possível observar no trabalho de Villela, Brandão e Leonardi (2016).

Eles são compostos por peças miniaturizadas, semelhantes às de porte maior aplicadas em satélites. Seu ciclo de desenvolvimento mais curto oportuniza a inserção de novas tecnologias de carga útil (SHIROMA et al., 2011). Ademais, uma constelação, com vários pequenos satélites, é potencialmente mais flexível do que o uso de um satélite grande e clássico, visto que é possível sua reconfiguração de acordo com as necessidades de cada missão, distribuindo tarefas, e a redundância presente nessa contribui com a atenuação de suscetibilidade à falha catastrófica de ponto único. Ainda, em um cenário de ataque inimigo, este terá muito mais dificuldade em imobilizar todos os nanosatélites em uma rede com vários nós, comparado a um único típico satélite.

O modelo cúbico de produção satelital, *CubeSat*, estimulou o uso desses satélites pequenos em grandes missões. No Brasil, este é o modelo mais adotado nos últimos tempos em nanosatélites (ANTUNES, 2017). Conforme um de seus criadores, o professor Jordi Puig-Suari, da CalPoly (*California Polytechnic State University*), os satélites educacionais eram muito grandes e complexos (STEPHENSON, 2010). Para alterar isso, com o auxílio do professor Robert Twiggs, da *Stanford University*, desenvolveu-se seu primeiro exemplar, um satélite miniaturizado com área de 10 cm^3 (também denominado³ “1U”, Figura 2) e peso de até 1 kg.

Figura 2 – Modelos de *CubeSats*



Fonte: NASA (2015)

Shiroma et al. (2011) revelam que os primeiros *CubeSats* consistiam em cobaias, a fim de provar que se poderia utilizar esta plataforma

³No vocabulário de Engenharia Espacial, medidas cúbicas (altura, largura e comprimento) recebem a indicação “U”, de unidade. Assim, no caso deste satélite com 10 cm em cada dimensão, pode ser designado como um *CubeSat* 1U. À medida em que se aumentam o número de cubos, a nomenclatura muda para 2U, 3U, 4U e assim sucessivamente (ANTUNES, 2017).

tão pequena, com curtos prazos de desenvolvimento, orçamentos baixos e equipes pequenas, para respeitáveis projetos científicos. A partir destas vantagens, conseguiu-se ampliar seu desenvolvimento dentro de universidades como uma ferramenta prática de tecnologia espacial para os alunos (COSTA; DURÃO; CRS, 2011). Com avanços acadêmicos e sucessos em suas demonstrações, entidades comerciais e governamentais – antes céticas – despertaram interesses e iniciativas.

O Instituto Nacional de Pesquisas Espaciais (INPE) em cooperação com a Universidade Federal de Santa Maria (UFSM) lançou a primeira missão espacial brasileira empregando *CubeSats* em 19 de junho de 2014. Nela, o NanosatC-Br1 foi desenvolvido com o objetivo de que as informações capturadas através dele pudessem ser utilizadas em pesquisas sobre clima espacial e fenômenos que impactam a Terra, como a Anomalia Magnética do Atlântico Sul (INPE, 2018).

Alguns elementos deste *CubeSat*, como a plataforma e o magnetômetro utilizado na carga útil, foram adquiridas de empresas estrangeiras. No entanto, atribui-se a instituições brasileiras o desenvolvimento de componentes como os circuitos integrados para testes de radiação no ambiente espacial, o dispositivo para acionamento remoto de cargas úteis e o *software* que administra os problemas ocasionados por efeitos da radiação ionizante em um FPGA (*Field Programmable Gate Array*) (VILLELA; BRANDÃO; LEONARDI, 2016).

2.2 REDES INTRA-CHIP

Segundo Sorin (2009), desde antigamente, o propósito imprescindível dos arquitetos de computadores estava relacionado ao desempenho. Com transistores menores e cada vez mais rápidos, proporcionados pela ‘Lei de Moore’⁴, alcançaram-se melhores performances. Este grande avanço nas tecnologias de fabricação de circuitos integrados (CIs) permitiu a implementação de sistemas complexos, com milhões de transistores em uma única pastilha de silício, os *Systems-on-Chip* (SoCs). A partir disto, o atraso em fios globais rapidamente se tornou

⁴“A complexidade para componentes com custos mínimos tem aumentado em uma taxa de aproximadamente um fator de dois por ano [...] Certamente, em um curto prazo, pode-se esperar que esta taxa se mantenha, se não aumentar. A longo prazo, a taxa de aumento é um pouco mais incerta, embora não haja razão para acreditar que ela não se manterá quase constante por pelo menos 10 anos. Isso significa que, em torno de 1975, o número de componentes por circuito integrado para um custo mínimo será 65.000. Eu acredito que circuitos grandes como este poderão ser construídos em um único componente (pastilha)” (MOORE, 2006, tradução autora).

maior que o atraso em portas lógicas.

Sistemas que utilizam CIs distintos para a troca de informações entre o *hardware* e o *software* contam com redução em seus desempenhos. Uma vez que os componentes de *hardware* e *software* se localizam integrados em um mesmo CI, o desempenho global do sistema tende a ser maior e o consumo de potência menor.

A concepção de uma *Network-on-Chip* (NoC) visa desassociar a estrutura de comunicação e a aplicação no plano físico. Na perspectiva de sistema, cada recurso implementado é um sistema embarcado independente e possui uma interface padronizada consoante à estrutura de comunicação (SOININEN; HEUSALA, 2003). Dessa forma, um sistema baseado em NoC é basicamente um sistema distribuído, com recursos que podem empregar distintos controles de relógio (*clock*) e se comunicar entre si síncrona ou assincronamente, conforme o paradigma GALS⁵ (*Globally Asynchronous Locally Synchronous*) (PALMA, 2007).

De acordo com Benini e Micheli (2002), as NoCs podem ser estruturadas em uma pilha de protocolos, similar à pilha de conceito geral de redes de comunicação. As camadas encapsulam funções equivalentes às definidas para os níveis hierárquicos do modelo internacional de referência OSI (*Open System Interconnection*), apresentado pela ISO (*International Organization for Standardization*) com Zimmermann (1980). Este modelo possui sete níveis de protocolos, os quais estão especificados na Figura 3. A finalidade de estruturar protocolos em camadas é a delimitação e o isolamento de funções de comunicação para cada nível. Deste modo, é possível separar evidentes aspectos de implementação de cada uma, administrando a complexidade dos requisitos e funcionalidades para a rede *intra-chip*.

2.2.1 Comunicação em camadas

Como mencionado, em uma rede de interconexão, a conectividade entre um conjunto de nós ocorre em diferentes níveis. Projetistas da rede desenvolvem as camadas de transporte, de rede e de enlace, de forma que a camada física seja de responsabilidade dos projetistas de circuitos integrados. Já as camadas superiores (5 a 7, da Figura 3) são elaboradas pelos projetistas dos módulos que se conectarão à rede (SILVA et al., 2010).

⁵GALS é um modelo de computação que permite projetar sistemas computacionais compostos de ilhas síncronas interagindo com outras ilhas de comunicação assíncronas.

Figura 3 – Camadas do Modelo OSI



Fonte: Mello, Möller e Heleno (2003)

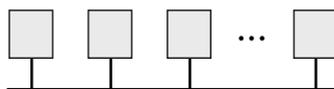
A camada física realiza a transferência de dados em nível de *bits* através de um canal. Ela possui relação direta com as características físicas do meio utilizado para conectar os recursos de *hardware* disponíveis. Em SoCs, associa os níveis de tensão do circuito, ao comprimento e à largura dos fios, conectores, taxas de dados e outros recursos.

A camada de enlace, ou canal, é a mais próxima ao nível físico, e realiza o acesso aos meios de comunicação. Pode conectar diretamente dois nós ou mais (PETERSON; DAVIE, 2003). A Figura 4 demonstra canais físicos a) limitados a um par de nós e b) compartilhando o mesmo canal. Em contrapartida, a conexão direta não é a única, um conjunto de nós que cooperam entre si representam a conectividade indireta. Nesta, os nós que se conectam a pelo menos dois canais devem ser capazes de transmitir os dados recebidos de um canal para outro. Um exemplo é a Topologia em Estrela Estendida, na Figura 5, onde cada nó possui um ou mais canais ponto-a-ponto.

Figura 4 – Conectividade direta



a) Ponto-a-ponto



b) Barramento

Fonte: Adaptado de Mello (2007)

Esta camada faz a comunicação em nível de quadros, isto é, grupos de *bits* e se preocupa com a integridade dos dados que trafegam na rede, através de tratamento de erros e controle do fluxo de transferência dos quadros.

A camada de rede é responsável pela comunicação ponto-a-ponto entre os módulos de *hardware*, ou seja, a comunicação em nível de pacotes (grupos de quadros). Nela, encontram-se os algoritmos de empacotamento das mensagens, de roteamento dos pacotes e de controle do congestionamento e contabilização de pacotes transferidos corretamente. Portanto, a partir dela, encontram-se os endereços e a escolha do melhor caminho.

A última das camadas antes das superiores é a de transporte. Ela utiliza interfaces de rede (NI – *Network Interface*), as quais têm como principais objetivos inserir os dados em diversos pacotes e converter o protocolo de comunicação ponto-a-ponto para o utilizado na rede. No caso de um chaveamento por pacotes, explicado na Seção 2.2.3, cada pacote pode ser quebrado em *flits*. Neste nível, constrói-se também o cabeçalho do pacote – indicando informações de origem e destino – para, ao fim, em seu destino correto juntar os pacotes, transformando-os em mensagens novamente.

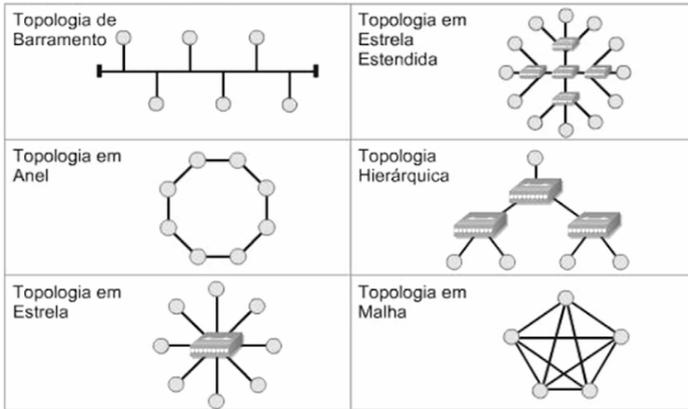
2.2.2 Topologias de rede

A forma como os componentes de uma rede ficam dispostos em seu projeto geral é extremamente importante e influencia diretamente no desempenho e na capacidade da mesma. Em vista disso, idealizaram-se algumas topologias físicas de rede – como são denominadas – para analisar a melhor maneira de interligar determinados componentes em diferentes ambientes e situações, além de como a informação propagar-se-á. A Figura 5 retrata algumas topologias de rede conhecidas da literatura e possíveis em NoCs.

A topologia de barramento, também conhecida como *bus*, é uma topologia de redes de difusão e possui suas estações conectadas a um cabo serial, por onde trafegam todos os dados da rede. As demais topologias ilustradas na Figura 5 pertencem ao conjunto das topologias de redes ponto-a-ponto, cuja a rede dispõe de linhas de comunicação concatenadas a duas estações por vez.

A topologia em anel, ou anelar, caracteriza-se por um anel fechado, estruturado através de nós e *links*. Suas estações se conectam cada uma a um nó respectivo e as mensagens passam de um dispositivo

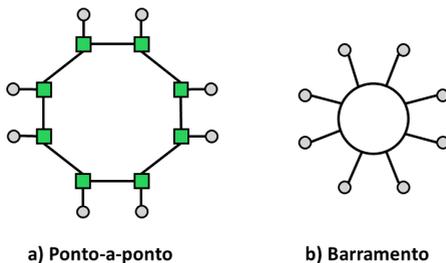
Figura 5 – Topologias de rede



Fonte: Ross (2008)

a outro, em qualquer direção – apesar de geralmente ser utilizada a configuração unidirecional, a fim de se estabelecer um protocolo de comunicação menos sofisticado e com maior garantia de entregas corretas e em sequência (ROSS, 2008). Na Figura 6.a) os quadrados representam os roteadores e os círculos as estações que necessitam se comunicar. Há, ainda, a conectividade anelar em que um conjunto de nós compartilha o mesmo canal (barramento), conforme a Figura 6.b).

Figura 6 – Conectividade em rede anelar



Fonte: Autora

A topologia em malha, ou *mesh*, também tem o objetivo de criar uma rede descentralizada, sem dependência de um ponto central, e ainda se autoconfigurar conforme a necessidade. Já a topologia em estrela contém todas suas estações interligadas a um nó central, o qual é responsável pelo controle da comunicação. Neste tipo, segundo Kurose

e Ross (2006), a confiabilidade é um problema, visto que pode haver comunicação simultânea. Inclusive, caso haja falha no nó central, os componentes ligados a ele perdem suas conexões.

Seguindo o raciocínio, a topologia em estrela estendida é uma variação da topologia em estrela, pois conecta suas estações a nós interligados ao nó central. Por fim, a topologia hierárquica, ou em árvore, possui uma hierarquia – como seu próprio nome sugere – para a conexão das estações.

2.2.3 Modos de chaveamento

Para que seja possível a transferência de dados de um roteador para outro, emprega-se um modo de chaveamento. Segundo Mohapatra (1998), os mais utilizados estão descritos abaixo:

- **Chaveamento por circuito:**

A conexão é estabelecida antes do início da transmissão dos dados e, para que ela ocorra, os recursos solicitados pela aplicação necessitam estar acessíveis por todos os roteadores no caminho da conexão (KUMAR; MANJUNATH; KURI, 2004). Uma degradação no desempenho do sistema como um todo pode ocorrer ao se utilizar este método, em consequência de os recursos permanecerem alocados até mesmo nos períodos em que a aplicação não envia dados.

- **Chaveamento por pacotes:** O fluxo de mensagens é dividido em pedaços menores – pacotes – para sua transmissão. De acordo com Duato, Yalamanchili e Ni (2003), cada pacote é individualmente roteado da origem até o destino correto e precisa conter um cabeçalho com informações de roteamento e controle. Como, geralmente, a taxa máxima total que as origens podem transmitir excede a capacidade do canal, esses dados excedentes são armazenados em um *buffer*, provocando um aumento da latência.

Esta categoria de chaveamento requer uma política de repasse de pacotes, sendo que as três mais utilizadas são *store-and-forward*, *virtual-cut-through* e *wormhole* (MOHAPATRA, 1998). Nesta última, um pacote é transmitido em unidades chamadas *flits*, que segundo Dally e Towles (2004), é a menor unidade de informação sobre a qual se realiza o controle de fluxo.

2.2.4 Roteamento dos dados

O caminho feito pelo pacote de dados entre uma origem e um destino é denominado roteamento. Como ele intervém demasiadamente na média da taxa de envelhecimento dos pacotes que percorrem a rede, é visto por diversos projetistas como o desafio primordial no projeto da camada de rede em NoCs. Assim, escolher a topologia da rede, o método de chaveamento e o algoritmo de roteamento é essencial, após analisar os requisitos do projeto, visando evitar atraso na entrega dos pacotes, congestionamento da rede e reduzir o número de transições desde a fonte até o destino.

Zhang et al. (2009) classificam os roteamentos conforme o modo de transmissão dos dados. Em um roteamento determinístico, o caminho é estabelecido unicamente pelos endereços do roteador atual e do roteador destino. Neste método sempre se provém o mesmo caminho entre um par origem-destino. Por outro lado, um algoritmo de roteamento adaptativo decide qual o caminho a ser tomado em função dos dados de endereço e das condições de tráfego da rede. Uma de suas vantagens é a maior disponibilidade de caminhos para o pacote chegar ao destino, porém, com isso, pode ocorrer entrega de pacotes fora de ordem, dado que mensagens com mesmo par origem-destino podem vir a percorrer caminhos distintos.

Há também outra classificação de roteamentos, de acordo com o número de destinos (DUATO; YALAMANCHILI; NI, 2003). Quando as mensagens possuem um único destino, o algoritmo de roteamento é conhecido como *unicast*. Para mais de um destino, denomina-se *multicast*. No caso particular onde todos os possíveis destinos são os destinos reais das mensagens, ainda, intitula-se como *broadcast*.

2.2.5 Vantagens técnicas

Uma rede é projetada de acordo com suas aplicações, demandando modelagem e simulação correta do tráfego para obter o desempenho desejado pelas aplicações alvo. Com as simulações, pode-se redimensionar a rede para se encaixar nos requisitos que ela deve suportar. Isto posto, o uso de redes *intra-chip* possibilita a observação de diversas vantagens técnicas em relação a outros tipos de estruturas utilizadas (AMORY, 2007). As principais estão descritas abaixo:

- Redução das limitações referentes a fios longos.

- Reuso de núcleos de *hardware*.
- Escalabilidade flexível dos nós.
- Desacoplamento da comunicação e do processamento, dividindo o sistema em subtarefas independentes.
- Frequência menor, ocasionando menor consumo de energia.
- Facilidade de compreensão do sistema.
- Facilidade de implementação de serviços de comunicação.
- Baixo custo de área de integração da NoC com os processadores.
- Possibilidade de integrar mais lógica em um *chip*.

2.3 FALHAS EM DISPOSITIVOS ELETRÔNICOS

As informações transmitidas via meios eletrônicos podem ser atingidas por fatores externos e internos aos seus sistemas. Tais interferências conseguem alterar parâmetros físicos das operações de envio e recepção do sinal, além do próprio conteúdo das informações, deixando-as incorretas ou incompletas.

Não obstante, existem sistemas de comunicação que exigem informações exatas, como em satélites, uma vez que o conteúdo das mensagens transmitidas não pode ser retransmitido ou, se possível, depende alto custo. Neste caso, se houver informações inexatas, o sistema pode vir a se tornar defeituoso e proceder desastrosamente. Não é como em um cenário de ligação via celular, em que interferências nos sinais enviados podem causar baixa qualidade de voz e até mesmo perda de algumas palavras, mas o receptor da mensagem consegue entendê-la apesar da breve dificuldade.

À vista disso, um computador tolerante a falhas fornece segurança, sobretudo quando ligado, apesar da possibilidade de falhas. Para este, nunca se produzirá um resultado incorreto visível ao usuário, ocultando-lhe seus efeitos caso ocorram. Sorin (2009) afirma que é preferível que um computador pare de fazer algo a produzir respostas errôneas. Avizienis et al. (2004) explicam a relação entre falha, erro e defeito, conforme descrito abaixo.

Quando uma falha produz um erro, ela está ativa, isto é, o serviço entregue se desvia do serviço correto, seja por não conformidade com a especificação funcional ou por esta não expor adequadamente a função

do sistema. A falha se refere à transição do serviço correto para o incorreto. Quando se entrega este último, tem-se uma interrupção. O processo de transicionar inversamente é conhecido como restauração do serviço.

O desvio é chamado de erro e a sua causa (vista ou pressuposta) é a falha. Um erro pode ser disseminado e gerar outros erros. No entanto, quando a especificação funcional inclui um conjunto de funções, o erro de um ou mais serviços que implementam estas é o defeito, que pode degradar o sistema, o qual ainda oferece serviços essenciais ao usuário. Ou seja, o defeito acontece quando um erro é propagado para a interface de serviço, ocasionando irregularidades na sua forma de execução.

A Figura 7 elucida a representação da relação entre falha, erro e defeito, em seus universos de atuação. Sorin (2009) exemplifica falha como um fio quebrado ou um transistor com um óxido de porta quebrado; um erro como um *bit* que é zero em vez de um; e defeito como cálculos incorretos e interrupções do sistema. Weber (2003) ilustra isto através de um exemplo dos três acontecimentos relacionados: Um *chip* de memória que apresenta uma falha do tipo “grudado-em-zero” (*stuck-at-zero*) em um de seus *bits* pode ocasionar uma análise errada da informação armazenada em uma estrutura de dados e resultar em uma não autorização de embarque a todos os passageiros de um voo.

Figura 7 – Relação entre falha, erro e defeito



Fonte: Adaptado de Weber (2003)

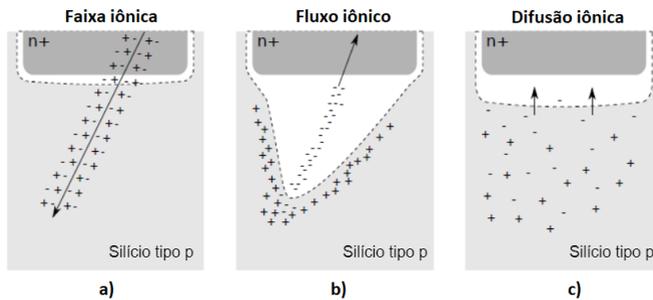
2.3.1 Efeitos da radiação em dispositivos eletrônicos

Ao operar em ambientes severos de radiação, os dispositivos eletrônicos podem ser atingidos diretamente por tipos diferentes de partículas (por exemplo, fótons, elétrons, prótons, nêutrons ou partículas mais pesadas), alterando suas propriedades elétricas e possivelmente

levando a um erro do dispositivo. A situação se agrava ainda mais em dispositivos mais novos, pois a miniaturização e as tensões operacionais mais baixas reduzem a energia necessária a um evento de radiação, induzindo a um erro. Portanto, ataques de partículas de energia mais baixos, que eram inofensivos em gerações anteriores de dispositivos eletrônicos, podem agora ser considerados uma ameaça.

A Figura 8 ilustra a sequência de eventos que pode ocorrer uma vez que uma partícula energética atinge o substrato de um dispositivo de silício. No início, o evento de radiação cria uma faixa de ionização com cargas positivas e negativas (*eletron/holes*) em pares (a). Então, quando a trilha de ionização cruza a região de depleção, o campo elétrico rapidamente coleta cargas elétricas em forma de funil, criando um fluxo de corrente/tensão (b). Em outra fase, a difusão domina o processo de coleta, até que todas as cargas excedentes sejam coletadas, recombinadas ou difundidas, longe da área de junção (c). No caso da carga total coletada (Q_{coll}) exceder a um valor de carga crítica (Q_{crit}) (que por sua vez depende das características da junção do silício, por exemplo, capacitância e tensão de operação), o evento pode produzir um erro (BAUMANN, 2005a).

Figura 8 – Partícula energética atingindo um dispositivo eletrônico



Fonte: Adaptado de Baumann (2005a)

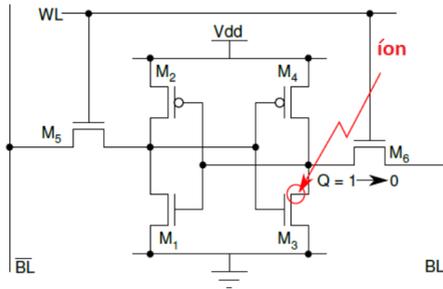
O termo usado para se referir a eventos causados por uma única partícula ao atingir o dispositivo eletrônico é *Single Event Effect* (SEE). Um SEE pode ser destrutivo e gerar erros graves, quando resulta em danos permanentes ao dispositivo (por exemplo, ruptura de porta e *bits* congelados) ou não destrutivo (erros leves), quando o evento gera apenas uma corrupção temporária de dados (por exemplo, inversão de *bits* indesejados em células de memória e registradores, que podem ser resolvidos sobrescrevendo o *bit* afetado). A rede proposta neste trabalho foi projetada para atuar na ocorrência de SEE leves.

2.3.2 *Single Event Upset (SEU)*

Um evento denominado *Single Event Upset (SEU)* acontece toda vez que uma partícula energeticamente carregada atinge de modo direto um componente de armazenamento (como célula de memória, registrador ou *flip-flop*), causando distúrbios de carga suficientes para modificar o valor armazenado (BAUMANN, 2005a). No cenário mais comum, o evento de radiação afeta apenas um único *bit*, chamado de *Single Bit Upset (SBU)*. Eventos de radiação com energias mais altas podem causar modificação de múltiplos *bits*, isto é, *Multiple Bit Upset (MBU)*.

A Figura 9 descreve como um evento de radiação pode induzir um SEU em uma célula de memória de acesso aleatório estático (SRAM – *Static Random Access Memory*) de seis transistores. Primeiramente, antes do evento de radiação, a célula está armazenando o valor “1”. Quando o íon atinge a célula de memória, no dreno do transistor NMOS M3, ocorre uma mudança transitória na tensão de saída do inversor direito, que está diretamente conectada à entrada do inversor esquerdo. Caso a tensão na entrada do inversor esquerdo fique abaixo do limite de comutação, o valor armazenado na célula é alterado e passa a ser “0”, resultando, assim, em um SEU.

Figura 9 – SEU em uma memória SRAM



Fonte: Adaptado de Sajid et al. (2017)

2.3.3 *Single Event Transient (SET)*

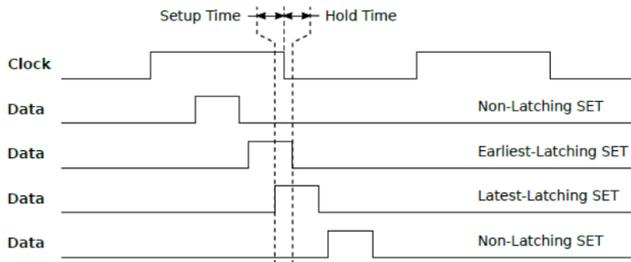
Quando uma partícula energética atinge um circuito lógico combinacional, em vez de um elemento de um circuito de armazenamento, a radiação gerará um pulso transitório (ou seja, uma falha) na saída, co-

nhecido como *Single Event Transient (SET)* (BAUMANN, 2005b). Neste caso, o pulso transiente é propagado e armazenado em um elemento de memória, gerando um erro leve.

Em contraste com os SEUs, que têm uma taxa de erro independente da frequência do *clock* do circuito, um SET só pode produzir um erro leve se o pulso transiente chegar na entrada do elemento de memória durante a borda de transição do *clock*. Devido a este comportamento, a probabilidade de travar um pulso transitório aumenta com frequências de *clock* mais altas ou quanto maior for a largura do pulso (GADLAGE et al., 2004).

A Figura 10 mostra pulsos de SET chegando à entrada de um elemento de memória em diferentes instantes de tempo. Observe que, no exemplo considerado, aciona-se por borda *falling edge*, exigindo um pulso SET cronometrado com a borda descendente do *clock* para induzir um erro. Nos sinais “Data” superior e inferior (*Non-Latching SET*), o pulso acontece ou muito cedo ou muito tarde e, portanto, são SETs sem travamento. Por outro lado, nos dois sinais de “Data” intermediários, os pulsos de SET são alinhados com o *clock* e, desta forma, travados no elemento de memória. Ademais, note que, se o pulso for longo o suficiente (ou seja, pelo menos um ciclo de *clock*), o sinal sempre será travado.

Figura 10 – Evento de SET em diferentes cenários de tempo



Fonte: Adaptado de Benedetto et al. (2006)

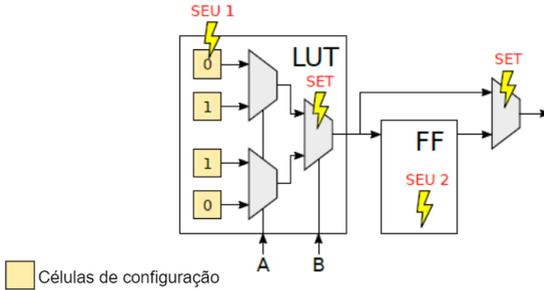
2.3.4 Erros “leves” (*soft errors*) em FPGAs

A fim de propor uma arquitetura de rede eficiente de tolerância a falhas visando a erros leves (*soft*) em FPGA, é importante entender as particularidades dos dispositivos FPGAs em relação ao tipo de erro. Mais especificamente, saber como SETs e SEUs podem perturbá-los.

No caso de SETs, eles correspondem a apenas uma pequena fração da taxa global de falhas em FPGAs, sendo muito menos frequente do que em dispositivos ASIC (*Application Specific Integrated Circuit*) (HUSSEIN; SWIFT, 2012; LESEA et al., 2005). Deve-se, principalmente, esta alta tolerância ao grande carregamento capacitivo presente nos caminhos de sinal dentro da estrutura do FPGA. Conseqüentemente, aumenta-se a carga crítica, exigindo partículas de alta energia para induzir um erro. Além disso, as frequências de operação mais altas encontradas em FPGAs são tipicamente muito menores do que o necessário para que os SETs sejam um elemento significativo na contribuição à taxa de erros *soft*.

Do ponto de vista do SEU⁶, há duas maneiras pelas quais o FPGA pode ser afetado: por perturbações na memória de configuração (por exemplo, CRAM – *Configuration Random Access Memory*), bem como por transtornos em *bits* que não são usados para configuração do FPGA (como registradores e memória incorporada BRAM – *Block Random Access Memory*). A Figura 11 ilustra os dois casos, como SEU1 e SEU2, respectivamente.

Figura 11 – Eventos de SET e SEU em uma célula lógica de FPGA



Fonte: Travessini (2018)

Os FPGAs dependem de sua memória de configuração para armazenar o arquivo de configuração, conhecido como “*bitstream*”. Este é responsável por especificar o comportamento de cada elemento lógico dentro do dispositivo, juntamente com a rede de interconexão interna do FPGA. Por conseguinte, a inversão de *bits* (ou *bitflips*) na memória de configuração pode deteriorar a funcionalidade do circuito original, permanecendo errônea até que uma nova configuração seja baixada no dispositivo FPGA.

⁶Disponível em: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01206-introduction-single-event-upsets.pdf>.

A probabilidade de um SEU acontecer na memória de configuração é altamente dependente da tecnologia FPGA. Aqueles baseados em SRAM, atualmente dominantes no mercado, são, sobretudo, sensíveis a esse tipo de erro (WANG et al., 1999). Nessa tecnologia, a memória de configuração é implementada com “interruptores” em SRAM, que não retêm a configuração quando a energia é removida. Alternativas não voláteis, como baseadas em memória *flash* e FPGAs de anti-fusíveis, proporcionam uma melhor tolerância à radiação (POIVEY; GRANDJEAN; GUERRE, 2011). Particularmente, as células anti-fusíveis são imunes a SEUs, porém, em desvantagem, não são reprogramáveis.

Para ser usado em ambientes hostis de radiação, os dispositivos FPGA que não têm células de configuração imunes ao SEU (por exemplo, baseadas em SRAM), devem fazer uso de estratégias específicas de tolerância a falhas atenuando os efeitos de falha nesta memória específica. A técnica mais frequentemente empregada é chamada de “deapuração de configuração” (*configuration scrubbing*). Esta abordagem envolve uma atualização periódica da memória de configuração do FPGA enquanto ele está operando. Este procedimento de re-escrita periódica evita o acúmulo de múltiplas falhas de configuração e reduz o tempo em que um circuito danificado com *bitflip* pode operar (KASTENSMIDT et al., 2005).

Quando se trata de SEUs em *bits* não relacionados à configuração, eles têm efeitos similares aos observados nos ASICs. Portanto, técnicas tradicionais, como as baseadas em redundância, podem ser aplicadas.

2.4 TÉCNICAS DE TOLERÂNCIA A FALHAS DEVIDO À RADIAÇÃO

A fim de se evitar defeitos em sistemas atingidos por radiação, pesquisadores experimentaram técnicas de tolerância a estes tipos de falhas, algumas descritas no Capítulo 3. A maioria dos trabalhos utiliza uma das duas principais abordagens consoantes ao tema: proteger os componentes, endurecendo-os pelo SEU, através de modificações no processo de fabricação ou transistores customizados; e, a redundância, por meio de códigos detectores e códigos corretores de erros.

No âmbito de FPGAs, as técnicas baseadas em redundância são as preferidas, uma vez que podem ser utilizadas concomitantemente aos COTS, sem a necessidade de projetar e desenvolver dispositivos customizados. Desta maneira, mesmo que um satélite passe por inter-

ferências e falhas de SEU e SET, a informação será reparada e passada ao receptor em sua forma correta, com menor custo e tempo para comercialização (KASTENSMIDT; CARRO; REIS, 2006).

2.4.1 Detecção e Correção

As redundâncias estão tão intimamente relacionadas a tolerância a falhas que, na indústria nacional, o termo “sistema redundante” é utilizado para caracterizar um sistema tolerante a falhas. Weber (2003) afirma que todas as formas de redundância impactam de alguma forma no sistema, seja no custo, no desempenho, na área (peso e tamanho) ou na potência consumida. Logo, não obstante possuam técnicas que certamente auxiliem na tolerância a falhas, cada projeto deve sempre ponderar o uso e o perfil da redundância.

As implementações podem ser feitas em diferentes níveis do projeto de circuito, bem como com distintos graus de proteção. Esta grande flexibilidade gera ampla variedade de técnicas redundantes, das quais a maioria, segundo Goloubeva et al. (2006), se enquadra nas categorias especificadas abaixo.

2.4.1.1 Redundância de *Hardware*

Também conhecida como Redundância Espacial, baseia-se na replicação do destino protegido e fornece tolerância a falhas direta e efetiva no custo do *hardware*. Villa (2018) afirma que a maioria das técnicas utilizadas para mitigação de SEUs em FPGAs são baseadas neste tipo de redundância.

Classifica-se em estática, dinâmica e híbrida e consiste na replicação de componentes e caminhos (*links*, roteadores, estruturas internas, tabelas de roteamento adaptativo) e inclusão de verificadores e testadores de *hardware*. Em geral, é a mais adequada para lidar com falhas permanentes, pois permite a continuação dos processos, mesmo na presença de partes avariadas sobre o sistema.

Segundo Weber (2003), a TMR (*Triple Modular Redundancy*), a qual triplica um componente pré-definido, é uma das técnicas mais conhecidas desta redundância. Da literatura, já se sabe que a TMR é ideal para períodos não muito longos de missão, visto que, com o tempo, aumenta-se a probabilidade de componentes falharem, ou seja, ela apresentará confiabilidade pior do que um sistema não redundante.

Como nanosatélites possuem pequenos tempos de vida, a aplicação da TMR é uma ótima opção para eles. À vista disso, em um dos testes deste trabalho, no Tópico 5.2.1, implementou-se a TMR na rede proposta para comparação de resultados.

2.4.1.2 Redundância de *Software*

Ajuda com todas as classes de falhas e pode conter dois ou mais fragmentos de código em execução para evitá-las. É possível dispor de variações em todos os níveis, como dados, controle de fluxo de programa e combinações híbridas. Como componentes idênticos de *software* apresentarão erros idênticos, não basta copiar um programa e executá-lo em paralelo ou executar o mesmo programa duas vezes.

Existem técnicas como os blocos de recuperação, a verificação de consistência e a diversidade (ou programação n-versões) (WEBER, 2003). Nesta última, a redundância pode ser classificada em duas classes: a versão única, em que um único módulo de *software* é modificado para incluir mecanismos de detecção, contenção e recuperação de falhas; e, múltiplas versões, com várias versões do mesmo módulo de *software* desenvolvidas, geralmente, por equipes diferentes, objetivando a redução da probabilidade de uma falha comum (GOLOUBEVA et al., 2006).

2.4.1.3 Redundância de Informação

Como usa *bits* adicionais para detecção e correção de erros (EDC e ECC), e estes *bits* não expandem a capacidade de representação de dados do código, pode-se perceber a razão da codificação também ser considerada uma forma de redundância. Adicionando-se informações de interesse, consegue-se identificar, mascarar e tolerar determinados erros.

Segundo Kim e Somani (2001), ela é utilizada principalmente para proteger matrizes de memória e comumente considerada de difícil aplicação em funções lógicas computacionais ou de controle. Grecu et al. (2006) utilizam a técnica de paridade, Kohler, Schley e Radetzki (2010) a verificação de redundância cíclica (CRC - *Cyclic Redundancy Check*) e Pereira et al. (2017) a replicação do *flit* de cabeçalho. Label e Gates (1996) ainda citam o Código de Hamming com notável robustez neste tipo de redundância.

No Tópico 5.2.2, é explicitado o teste realizado para colher os resultados com a implementação da técnica de detecção e correção de erros via Hamming. Com ela, utilizaram-se *bits* de paridade em conjunto com a adição de dois novos componentes físicos, o decodificador e o codificador. Isto é, além de conter redundância de informação, o projeto interno do roteador foi ampliado – o que fez com que fosse necessário obter também os resultados de consumo de silício desta implementação em relação à rede sem redundância.

2.4.1.4 Redundância Temporal

Ao lidar com falhas transitórias e intermitentes, consiste na reexecução de uma operação para posterior comparação e validação. Ela é implementada através da geração de múltiplas amostragens de uma mensagem, porquanto, na ocorrência destas duas falhas, haverá resultados diferentes para a mesma mensagem. No caso de falhas permanentes, repete-se a computação com dados codificados e o resultado é decodificado antes da comparação com o dado anterior.

Caso haja incompatibilidade dos resultados, pode-se executar a verificação novamente, a fim de apurar a ainda presença do erro ou seu desaparecimento. Sua ideia é semelhante à redundância de *hardware*, não obstante envolva repetição de tarefas de maneira sequencial, ao invés da utilização de *hardware* extra para executar as operações em paralelo.

Com ela, evita-se o custo de *hardware* adicional, porém aumenta-se o tempo necessário para realizar uma operação computacional. Por isto, geralmente é utilizada em sistemas onde o tempo não é crítico ou onde o processador trabalha com ociosidade. Ainda, há estratégias que agregam estas redundâncias, buscando reduzir influências tanto na área, como no desempenho.

2.4.2 Recuperação

Após corrigir uma falha, o sistema deve ser recuperado, ou seja, deve-se realizar a troca do estado atual incorreto para um estado livre de falhas. A recuperação é simples para sistemas com um único processo, porém complexa em processamento distribuído, e, neste último, pode ocasionar um efeito dominó. Assim, há duas técnicas de recuperação, conforme segue:

- ***Forward Error Recovery (FER)***

A recuperação antecipada de erros corrige o erro sem retornar a um estado anterior. Pode ser implementado utilizando redundância de *hardware*, *software*, informação ou temporal, porém é específica para cada sistema. Ela é eficiente, mas se deve prever cuidadosamente a ocorrência de danos.

- ***Backward Error Recovery (BER)***

Neste caso, é possível uma aplicação genérica, pois se restaura o estado do sistema para um estado saudável, conhecido como ponto de recuperação (ou linha de recuperação para um sistema com múltiplos núcleos), isto é, antes de ter acontecido o erro. Mas, como saber qual estado salvar para o ponto de recuperação? Ou qual algoritmo deve ser usado para salvá-lo? São algumas perguntas a se responder no momento de sua implementação.

É importante saber que, inicialmente, deve-se aguardar para enviar dados ao ‘mundo exterior’⁷ até que os mecanismos de detecção de erros tenham concluído a verificação de todas as operações antes do envio dos dados, evitando o problema de confirmação de saída. Sendo assim, o BER requer mais elementos de memória, tornando o sistema mais suscetível a falhas de SEU, bem como com um custo mais elevado.

2.4.3 Diagnósticos e Reparo

Se o erro está relacionado a uma falha permanente, as etapas de detecção e recuperação podem não ser suficientes. Portanto, torna-se necessário testar e verificar os componentes. Destarte, após o diagnóstico de um elemento defeituoso, deve-se desativá-lo e reconfigurar o sistema para reparar o feito – manualmente, por um operador local ou remoto; ou automaticamente, pelos componentes livres de falha.

Não obstante, Sorin (2009) afirma que para erros transitórios a execução pode ser retomada sem problemas após a etapa de recuperação, visto que, neste momento, o erro temporário não estará mais presente. Assim sendo, como o foco desta pesquisa é em falhas temporárias, esta etapa final não será verificada em seu desenvolvimento.

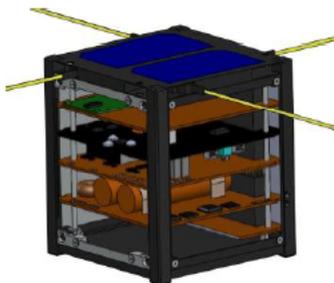
⁷“Tudo aquilo que não pode ser recuperado com o esquema BER” (SORIN, 2009).

3 TRABALHOS RELACIONADOS

Dispostos a atenuar falhas induzidas por radiação em dispositivos eletrônicos, pesquisadores propuseram e experimentaram diversas técnicas de tolerância a falhas. Abordagens como componentes fortalecidos para falhas do tipo SEU ou proteção do dispositivo através de redundância são as principais utilizadas.

O primeiro *CubeSat* nacional e segundo nanossatélite brasileiro, (MÂNICA, 2018), foi uma parceria bem-sucedida entre INPE, UFSM e AEB (INPE, 2018). O NanosatC-Br1, lançado em 2014, foi desenvolvido a partir de módulos interligados por barramentos no padrão PC-104¹, ou seja, não foi utilizada uma rede *intra-chip*. A Figura 12 ilustra o modelo estrutural deste nanossatélite.

Figura 12 – Modelo estrutural do NanosatC-Br1



Fonte: Costa, Durão e CRS (2011)

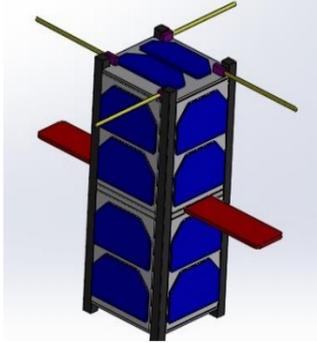
Franke et al. (2014) afirmam que, do ponto de vista térmico, o satélite deste projeto voará seguramente, mesmo em casos extremos. O que se deve atentar são as falhas causadas por radiação e ionização cósmica. Neste caso, há funções de “comando e controle”, as quais possibilitam a operação remota de seus subsistemas, tanto os que compõem a plataforma, quanto os da carga útil no decorrer de sua vida. Via telemetria altamente confiável e em tempo real, consegue-se corrigir uma falha em termos de parâmetros críticos à sobrevivência da missão e operar mecanismos conforme estabelecido (COSTA; DURÃO; CRS, 2011).

O NanosatC-Br2, que veio na sequência, é um *CubeSat* 2U, idêntico à ilustração da Figura 13. Devido a semelhança com o NanosatC-Br1 e a ausência de uma NoC, não se aprofundou o estudo de suas

¹Disponível em <https://pc104.org/>.

estruturas, técnicas e questões sobre tolerância a falhas, para dar prioridade a outros trabalhos com conceitos mais próximos.

Figura 13 – Modelo estrutural do NanosatC-Br2



Fonte: Mantovani et al. (2017)

Em seu trabalho, Travessini (2018) buscou a melhoria da confiabilidade em processadores *soft*. Para isso, propôs uma estratégia de tolerância a falhas de baixo custo, em que somente as partes mais vulneráveis do processador sejam protegidas. Primeiramente, fez-se um estudo sobre o processador-alvo LEON3, passo fundamental para as demais etapas. E, visando mitigar erros de *software* em processadores flexíveis baseados em um *chip* reprogramável a nível de portas lógicas (FPGA), o autor realizou uma extensa campanha de injeção de falhas.

As primeiras injeções foram aplicadas sem quaisquer técnicas de tolerância a falhas, visto que, a partir disto, conseguir-se-iam estatísticas relevantes, como caminhos de propagação de falhas, tempo de exposição a falhas e, principalmente, os componentes mais vulneráveis do LEON3. Em seguida, com os resultados obtidos, implementou-se a técnica de redundância parcial, a qual protegia exclusivamente os componentes mais vulneráveis. Ao fim, avaliou-se de forma a quantificar a melhoria na tolerância a falhas por meio das injeções e a determinar o custo de área e desempenho adquirido com o projeto.

O LEON3, processador utilizado, possui um *pipeline* de sete estágios, com *caches* de instrução e dados separados e vários módulos operacionais. Travessini (2018) investigou o *Processor Core*, onde há a maior parte da sua lógica de controle. E, como o LEON3 é capaz de ser amplamente configurado, pôde-se ativar seus módulos de interface para monitorar caminhos de propagação de falhas.

Para definir o melhor método de injeção de falhas para este tra-

balho, revisou-se propriedades como acessibilidade, repetibilidade, controlabilidade, reprodutibilidade, não-intrusividade, medição de tempo e eficácia. Assim, o método que melhor se ajustou aos requisitos foi a injeção de falhas baseadas em simulação, com a seleção do simulador de linguagem de descrição de *hardware* (HDL – *Hardware Description Language*) ModelSim, da Mentor Graphics. A principal limitação encontrada foi o tempo de simulação.

Em cada experimento, uma única falha foi injetada, invertendo o valor lógico do sinal alvo. Inicialmente, sem técnicas de tolerância, o comportamento errôneo por conta das falhas estava limitado a um breve conjunto de registradores, entre eles o contador de programa (PC – *Program Counter*) e os operandos da unidade lógica e aritmética (ALU – *Arithmetic Logic Unit*). Seus resultados corresponderam a falhas de resultado incorreto, tempo limite esgotado e exceção. Em relação a propagação de falhas, somente um terço das falhas injetadas levaram a um distúrbio nas interfaces e, como esperado, todas as falhas com efeitos prejudiciais se propagaram para pelo menos uma interface.

Com a técnica de redundância parcial, obteve-se que protegendo somente 30 registradores de 362 já se poderia reduzir em seis vezes o número de falhas que levaram a efeitos danosos. Ademais, os custos de utilização de recursos do FPGA foram bem inferiores aos usados com Tripla Redundância Modular. Notou-se também que o mecanismo de exceção do processador detectou apenas pouco mais da metade das falhas que levaram a efeitos prejudiciais, fato que mostra a necessidade de incluir outros mecanismos de detecção no caso de falhas críticas. Frisou-se, ainda, que a técnica proposta não é ideal para todas as aplicações.

Os fabricantes de FPGA desenvolveram tecnologias para tornar alguns dispositivos menos sensíveis à radiação cósmica, o que fez com que tais produtos se tornassem mais caros. Percebe-se, portanto, que em um trabalho de nanossatélite, isto vira um empecilho, visto que se desejam equipamentos de baixo custo. Desta forma, os FPGAs de baixo custo necessitam de uma estratégia em caso de falha no dispositivo, causada por radiação. Martins et al. (2018) abordaram o tratamento para reduzir falhas permanentes em FPGAs de baixo custo.

Introduziu-se, em seu trabalho, a metodologia de reconfiguração parcialmente dinâmica (DPR – *Dynamic Partially Reconfigurable*), cuja técnica é permitir a escrita de uma parte específica do FPGA, enquanto o restante do dispositivo ainda está em funcionamento. Utilizaram-se as fases de Detecção de Falhas, Isolamento de Falhas e Recuperação (FDIR – *Fault Detection, Fault Isolation and Recovery*), com recupe-

ração parcial. A fase de detecção contou com duas técnicas, uma que detecta falhas devido ao efeito impresso da dose total de ionização (TID – *Total Ionizing Dose*), através da replicação temporal, e a outra que se baseia no sensor de envelhecimento chamado *Differential Delay Sensor* (DDS), projetado para resolver as sequelas do desgaste. Portanto, quando alguma dessas técnicas detecta uma falha permanente, isola-se a região defeituosa e se coloca o módulo reconfigurável em uma nova partição.

A tecnologia FPGA adotada para estudo e experimentos foi a Xilinx Virtex-6, a qual possui partições reconfiguráveis (RP – *Reconfigurable Partition*), capazes de permitir um limite para a posição de um módulo ou, inclusive, configurar suas propriedades dinamicamente. Para tudo funcionar, regras relativas à interface de cada RP e ao roteamento nas respectivas bordas foram implementadas. Permitiu-se a troca de módulos entre os RPs, para aumentar a confiabilidade, e se introduziu um novo medidor de atraso, a fim de auxiliar na decisão de quando realizar a modificação.

Escolheu-se um esquema baseado em redundância temporal para melhorar a confiabilidade do sistema, o qual usa apenas um processador para executar duas vezes o mesmo cálculo, acompanhado da comparação dos resultados. Considerando que ao longo do tempo ocorre um processo natural de degradação física do transistor, os autores propuseram um novo medidor de atraso, o qual utiliza recursos que estão em uso quando um módulo é alocado em uma RP específica.

Com este trabalho, conseguiu-se assegurar que dentro de uma RP existam apenas sinais relacionados a ela; redução no uso de silício; desenvolvimento e implementação do DDS. Com um pequeno aumento na memória, o mecanismo proposto consegue tolerar falhas permanentes, substituindo módulos de *hardware* em múltiplas partições, sem a necessidade de múltiplos fluxos de *bits* parciais. No entanto, o *software* de eleição, que analisa os resultados e seleciona uma RP para trocar no caso de defeito, é executado na Unidade de Processamento Central (CPU), tornando-se um ponto único de falha.

O projeto apresentado por Villa (2018) trata, também com redundância temporal, falhas transitórias do tipo SEU, de forma que tenta restaurar o correto funcionamento do sistema ao custo do tempo de processamento em vez de se replicar o *hardware*. Com base em FPGAs, ele propõe uma arquitetura voltada para sistemas embarcados de aplicações espaciais.

Ao incluir pontos de verificação e recuperação (CR – *Checkpoint and Recovery*), o autor consegue mostrar que esta é uma alternativa vá-

lida para Redundância Modular Tripla e Dupla (TMR e DMR, respectivamente). Para isto, salvam-se pontos considerados seguros ao longo da execução do processador e, caso um erro seja detectado, reverte-se a este estado protegido. Novos elementos de memória são necessários para armazenar esta informação, mas não há triplicação dos seus constituintes para comparação.

Em se tratando de aplicações espaciais, com área lógica limitada e pequeno consumo de energia, sistemas com menos itens e mais confiabilidade se destacam. Nesta circunstância, economizam-se componentes, comparado ao TMR, por exemplo, mas deve-se atentar à frequência dos *checkpoints*, pois a reexecução do processador trará mais custos de consumo. Para atestar sua confiabilidade, a técnica de CR detectou os erros repetindo a realização de cada pedaço de instrução e a executou pela terceira vez para corrigir o erro, demandando mais energia.

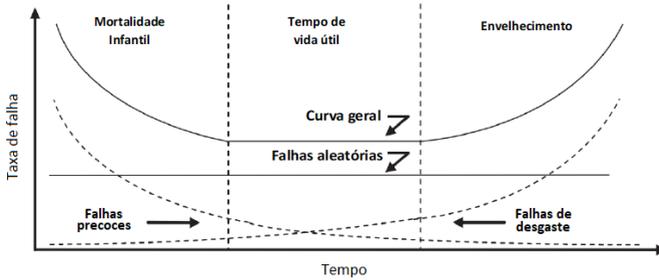
O trabalho de Villa (2018) também simulou o processador, visto no trabalho de Travessini (2018), LEON3 na ferramenta ModelSim, com uma campanha de injeção de falhas. Comparou seu modelo proposto com outras três arquiteturas: TMR, DMR e TR (*Time Redundant*) e obteve valores assertivos mesmo em casos de área lógica limitada e economias no consumo de potência – âmbitos de interesse para pesquisadores e desenvolvedores de aplicações espaciais.

Por fim, a pesquisa de Fochi (2015) foca em técnicas de tolerância a falhas aplicadas em redes *intra-chip*. Ele mostra a representação da quantidade de falhas em um sistema ou componente em relação ao tempo, conforme a Figura 14, também conhecida como “curva da banheira”. A fase de “mortalidade infantil” geralmente é relativa a defeitos de fabricação; na fase de “tempo de vida útil” acontecem falhas aleatoriamente por conta de diversos fatores (taxa constante de falhas); e, na fase de “envelhecimento”, os componentes já estão no fim de sua vida útil, resultando em falhas de desgaste – adiantadas em elementos cada vez menores, principalmente por aquecimentos (taxa crescente de falhas).

Além disso, Fochi (2015) cita algumas técnicas de estimativa de falhas. Ele introduziu técnicas de tolerância a falhas em nível de *hardware* e *software*, explorando a técnica de migração de tarefas em tempo de execução. E, para a detecção de falhas, adicionou módulos de CRC como lógica de BIST² (*Built-In Self-Test*) e fez testes com um *software* injetor de falhas.

²Técnica utilizada em tempo de projeto, adicionando *hardware* no circuito com a intenção de realizar testes sem a dependência de equipamentos externos (FOCHI, 2015).

Figura 14 – Curva de envelhecimento do circuito



Fonte: Adaptado de Fochi (2015)

No nível sistêmico, a decisão do remapeamento foi tomada de acordo com o preceito de se ter a menor perda de desempenho possível. Falhas no nível do roteador necessitaram de mecanismos adicionais de recuperação. E, no nível de enlace, adicionaram-se módulos de isolamento de portas de entrada com falha, trocando o canal.

Ao se tratar de falhas permanentes, em aplicações não críticas e com número limitado de erros aceitáveis, implementou-se uma rotina de testes para serem executados periodicamente. Se a aplicação fosse crítica, o autor utilizou técnicas de testes em nível de rede, replicando cada tarefa para comparação após o processamento do resultado.

Ainda, ele trata falhas transientes. Em ambos os casos, desativa-se o canal defeituoso e retransmite-se a mensagem. Mas, para a possibilidade de recuperar o canal, também se implementou outros dois módulos em *hardware*, o de teste e o de recuperação. Ao fim da leitura deste trabalho, pode-se inferir que o autor focou fortemente na resolução de falhas em redes *intra-chip*, adicionando componentes e novos acessos, sem se preocupar com o consumo de silício. Para realizar os testes, injetou-se falhas através do comando *force* no simulador ModelSim.

Na Tabela 2, pode-se visualizar um resumo dos principais conceitos abordados em cada trabalho examinado. Após esta análise, infere-se que são poucos trabalhos relacionados ao tema de tolerância a falhas em redes *intra-chip* referentes a nanossatélites. Além de escassas, as pesquisas contém somente alguns conceitos vinculados e são extremamente recentes, com um vasto campo a ser estudado. No Capítulo 6, há o conjunto de conceitos abordados na proposta desse trabalho de conclusão de curso, somado a comparação de melhorias englobadas em relação aos trabalhos descritos nesse capítulo.

Tabela 2 – Conceitos abordados

	(COSTA; DURÃO; CRS, 2011)	(TRAVESSINI, 2018)	(MARTINS et al., 2018)	(VILLA, 2018)	(FOCHI, 2015)
Nanossatélite	X		X		
Rede <i>intra-chip</i>					X
Baixo custo de silício		X	X	X	
Monitoramento em tempo real	X		X	X	X
Alvo de falhas	Plataforma e carga útil	Elementos sequenciais	Módulos ionizados e componentes envelhecidos	Registradores	Nível sistêmico, de roteador e de enlace
Método para detecção	Telemetria	Injeção de falhas em simulação	Efeito Impresso ao TID e DDS	Injeção de falhas em simulação	CRC e injeção de falhas em simulação
Técnica utilizada	Comando e controle remoto	Redundância Parcial	DPR e Redundância Temporal	CR e Redundância Temporal	Protocolos; Teste e recuperação do roteador
Falhas permanentes			X		X
Falhas transitórias	X	X		X	X
Falhas causadas por radiação	X	X	X	X	

Fonte: Elaborada pela autora

4 PROJETO DA REDE

A percepção de se implementar uma *Network-On-Chip* em um nanossatélite vai além do fato de que todos os componentes soldados em uma única placa é mais protegido do que a utilização de conectores, que podem soltar ou gerar mau contato. Deve-se atentar também que, na realidade, em um ambiente espacial há muita radiação e ionização cósmica, as quais podem causar interferência no sistema em execução, de acordo com a quantidade de silício incorporado na placa.

A rede *intra-chip* proposta neste documento foi projetada para atender à comunicação entre processadores *soft-core* em um *System-on-Chip* para nanossatélite. O desenvolvimento da rede levou em consideração as seguintes premissas:

- 1) A rede deve suportar até quatro processadores do tipo *soft-core*;
- 2) Os processadores *soft-core* utilizados devem possuir mecanismos internos para tratar falhas devido à radiação;
- 3) A memória de configuração do FPGA deve ser atualizada por um mecanismo de proteção externo ao FPGA; e
- 4) O consumo de silício da rede deve ser minimizado, consequentemente, reduzindo a probabilidade da ocorrência de falhas.

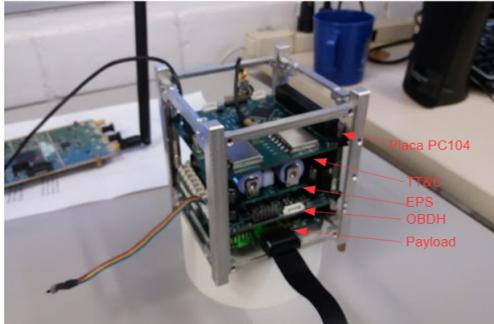
A primeira premissa se deve ao fato de o projeto do SoC substituir as quatro placas usadas em um nanossatélite de referência, o FloripaSat¹. Ele é um *CubeSat* 1U de 10 cm³ e seu principal objetivo é envolver os acadêmicos em uma missão espacial completa, desde o desenvolvimento dos módulos de um nanossatélite até a sua estação terrestre.

Cada placa do FloripaSat tem um microcontrolador: OBDH (*On-Board Data Handling*), TT&C (*Telemetry, Tracking and Command*), EPS (*Electric Power System*) e *Payload*. Estas placas estão ilustradas na Figura 15 e são interligadas por uma placa com conectores no padrão industrial PC-104, de forma semelhante ao que foi visto na Figura 12.

A placa OBDH atua no satélite como uma Unidade Central de Processamento. A placa TT&C possui os módulos de radiofrequência utilizados para comunicação com a estação na Terra. A placa EPS

¹Disponível em <https://floripasat.ufsc.br>.

Figura 15 – Foto ilustrativa do FloripaSat



Fonte: Disponível em <https://floripasat.ufsc.br>

controla a carga das baterias recarregáveis e das placas fotovoltaicas que envolvem o satélite. Por fim, a placa *Payload* é responsável por coletar dados de experimentos que estejam em andamento no ambiente espacial.

A segunda e a terceira premissas são importantes para delimitar o contexto desse trabalho. A reconfiguração periódica do FPGA e o uso de processadores *soft-cores* com proteção para ambientes de radiação, como os processadores LEON3², são importantes para que o foco da proteção idealizada nesse trabalho seja apenas na rede *intra-chip* proposta.

Por fim, a quarta premissa segue a lógica de que menor área de silício implica em menor probabilidade de ocorrência de falhas. Devido a isto, há quatro roteadores configurados e dispostos em uma rede anelar. Mais ainda, a comunicação entre eles, isto é, o deslocamento dos pacotes através da rede se dá apenas em um sentido (determinado como o sentido horário).

4.1 PROJETO LÓGICO

4.1.1 Topologia

Das topologias apresentadas na Seção 2.2, há de se concordar que a rede *mesh* dependeria uma maior quantidade de silício, uma vez que todas as estações estariam ligadas entre si. Ao contrário, pensando

²Disponível em <https://www.gaisler.com/index.php/products/processors/leon3>.

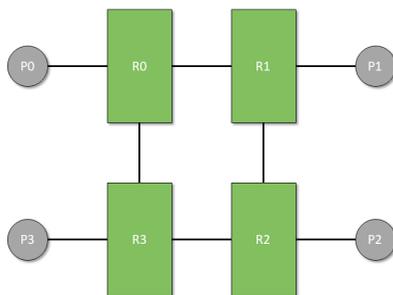
em uma economia de silício, o barramento, em sistemas com múltiplos núcleos altamente interconectados, tornar-se-ia um gargalo na comunicação. Já uma rede em estrela, devido à dependência de um ponto central, poderia conter alto congestionamento de dados e um simples comprometimento neste nó causaria a perda da conexão.

Desta forma, uma rede descentralizada, que não interconecta diretamente todos seus roteadores a todos os demais, adequar-se-ia aos quesitos de menor consumo de silício e em caso de uma interrupção em um dos caminhos, haveria como realizar a passagem do dado por outro lado. Esta rede é a anelar e deveria possuir roteamento bidirecional a fim de se evitar perda de pacotes em caso de falha permanente em um ponto de roteamento. Como neste trabalho o foco é específico em falhas transitórias, em todas as hipóteses levantadas os dados serão danificados temporariamente e o sistema deverá ser capaz de corrigir o erro sem a necessidade de passar os dados corretos por outro caminho. Portanto, uma rede anelar unidirecional é o suficiente para esta pesquisa.

Com base na primeira premissa, a rede proposta deverá conectar quatro núcleos de processamento e como se definiu que a topologia da rede desenvolvida será a anelar direta, cada processador precisará de um roteador para enviar e receber seus dados. Empregou-se o roteamento determinístico, em que o caminho já está definido de acordo com a origem e o destino, e do tipo *unicast* (vide 2.2.4), pois cada mensagem terá único roteador final.

Inicialmente, pode-se simplificar a representação da rede conforme a Figura 16, em que os Ps representam os processadores e os Rs os roteadores. Ao longo do trabalho, cada componente interno aos roteadores será exposto para explanação.

Figura 16 – Rede anelar simplificada com processadores e roteadores



Fonte: Autora

4.1.2 Formato do pacote

Utilizando o chaveamento por pacotes (explicitado em 2.2.3) e denominando um “pacote” como um “conjunto de *flits*”, estipulou-se que cada *flit* a entrar na rede conterà, por padrão, 32 *bits*, dos quais os dois primeiros (os mais significativos) indicam se é início (*header*), meio (*payload*) ou fim (*tail*) do pacote, conforme a Tabela 3, e os terceiro e quarto especificam o roteador em que o dado deve chegar. Os demais 28 *bits* são os dados que de fato se quer enviar, a carga útil para se utilizar nos experimentos. Por conseguinte, conseguiu-se padronizar, de forma simples e instrutiva, a indicação de se o pacote já acabou de ser transferido, seu destino e os *bits* com conteúdo, consoante à Figura 17.

Tabela 3 – Especificação dos *bits* mais significativos

	1 ^o e 2 ^o <i>bits</i> mais significativos	3 ^o e 4 ^o <i>bits</i> mais significativos
00	<i>Payload</i>	Roteador 0
01	<i>Payload</i>	Roteador 1
10	<i>Tail</i>	Roteador 2
11	<i>Header</i>	Roteador 3

Fonte: Elaborada pela autora

Figura 17 – Especificação dos *bits* dos *flits*

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
	Posição no pacote		Destino		Dado																															

Fonte: Autora

A Tabela 4 mostra um exemplo de um pacote completo enviado do Roteador 0, 2 ou 3 para o Roteador 1. Ainda, a implementação da rede não suporta o envio de dados de seu processador para ele mesmo, conhecido como *write back*. Portanto, não é possível enviar um dado do Roteador 1 para o Roteador 1, por exemplo.

Como o *flit* que entrará na rede deve ser codificado para garantir a integridade de seus dados, utilizou-se o Código de Hamming, esclarecido no item 4.1.5. A partir deste código, para um *flit* com 32 *bits*, adicionam-se 6 *bits* de redundância. Desta forma, quando o dado circula pelos canais físicos entre os roteadores, o *flit* contém 38 *bits*. Ao chegar no próximo roteador, ele é decodificado para averiguação de

Tabela 4 – Exemplo de envio de *flits*

Pacote	1 ^o e 2 ^o <i>bits</i> mais significativos (Fração)	3 ^o e 4 ^o <i>bits</i> mais significativos (Destino)
1101 0000 0100 1111 0011 1000 0100 0110	<i>Header</i>	Roteador 1
0001 1000 1000 1111 1101 1011 1101 1000	<i>Payload</i>	Roteador 1
0001 1011 1101 0001 0111 1100 1011 0110	<i>Payload</i>	Roteador 1
1001 0000 0000 0110 0000 0000 1001 0000	<i>Tail</i>	Roteador 1

Fonte: Elaborada pela autora

integridade, retornando a 32 *bits* e comparando com os 32 iniciais, e identificação do destino para confirmar se é o roteador atual. Após isto, é codificado novamente para passar pelos componentes do roteador e direcionado à saída correta, se para a rede: o *flit* segue codificado ao próximo roteador; se local: o *flit* é decodificado novamente e enviado ao processador local.

4.1.3 Canais físicos

Com as características definidas nos tópicos 4.1.1 e 4.1.2, pode-se implementar a rede com 4 roteadores na linguagem VHDL (*VHSIC Hardware Description Language*, onde VHSIC significa *Very High Speed Integrated Circuits*), através do *software* Quartus II 13.1. Cada roteador recebe o *flit* de seu anterior e o envia ao roteador seguinte, em sentido horário, até que se reconheça o destino correto dele. Além disso, é enviado um sinal em sentido anti-horário para indicar se o roteador está ocupado com outro pacote ou se pode receber o dado que o roteador anterior precisa transferir, a fim evitar perda ou subscrição de *flits*.

A inicial do nome dos sinais de entrada nos roteadores foi padronizada com um “*i_*” de *input* e os de saída com um “*o_*” de *output*. A finalização do nome contém “N” de *Network* (ligação com a rede) ou “L + número” de *Local* e o número de seu respectivo roteador (ligação com seu processador) ou nenhum dos dois (caso seja um sinal global). A Tabela 5 contém os sinais das entradas e saídas do Roteador 0, suas aplicabilidades e a quantidade de *bits* que são transferidos em cada li-

gação. Para os demais roteadores, pode-se apenas alterar os números correspondentes nos nomes dos sinais, pois, no projeto, foi desenvolvido um roteador e triplicado-o, alterando somente seu fluxo de sinais, ou seja, internamente os roteadores possuem os mesmos componentes, apesar de ligações diferentes.

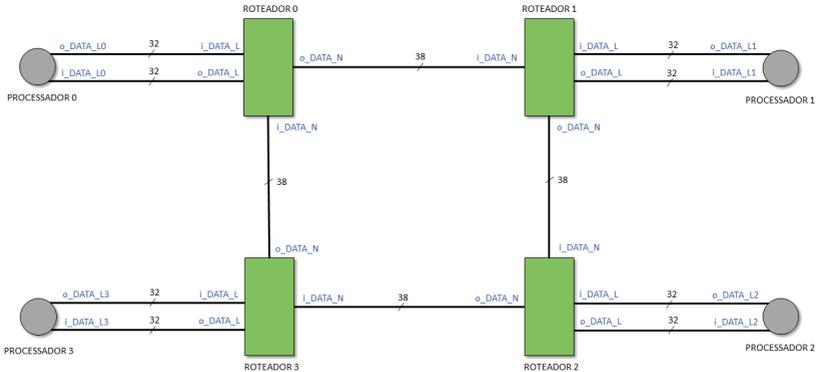
Tabela 5 – Entradas e saídas do Roteador 0

Sinais	Aplicação	Bits
<i>i_CLK</i>	Dar os pulsos do <i>clock</i>	1
<i>i_RST</i>	Reiniciar o roteador (limpeza de dados)	1
<i>i_BUSY_L0</i>	Setar o Roteador 0 como ocupado (recebendo mensagem do Processador 0) ou desocupado	1
<i>i_WR_L0</i>	Indicar se o dado local pode entrar no Roteador 0	1
<i>i_DATA_L0</i>	Enviar ao Roteador 0 o dado do Processador 0	32
<i>i_BUSY_N</i>	Setar o Roteador 0 como ocupado (recebendo mensagem do Roteador 3) ou desocupado	1
<i>i_WR_N</i>	Indicar se o dado vindo do Roteador 3 pode entrar no Roteador 0	1
<i>i_DATA_N</i>	Enviar ao Roteador 0 o dado do Roteador 3	38
<i>o_BUSY_L0</i>	Informar ao Processador 0 se o Roteador 0 está ocupado (recebendo mensagem) ou desocupado	1
<i>o_WR_L0</i>	Indicar se o dado recebido pode ir ao Processador 0	1
<i>o_DATA_L0</i>	Enviar ao Processador 0 o dado do Roteador 0	32
<i>o_BUSY_N</i>	Informar ao Roteador 3 se o Roteador 0 está ocupado (recebendo mensagem) ou desocupado	1
<i>o_WR_N</i>	Indicar se o dado recebido pode ir ao Roteador 1	1
<i>o_DATA_N</i>	Enviar ao Roteador 1 o dado codificado do Roteador 0	38

Fonte: Elaborada pela autora

A Figura 18 ilustra as ligações da rede pelas quais os *flits* são transmitidos. Além disso, também há sinais sendo enviados e recebidos de sua rede local, a qual futuramente conectará o roteador a um processador, através de uma *Network Interface*. Conforme visto na Figura 16, cada processador da rede local se comunica diretamente apenas com seu determinado roteador, o qual estabelecerá a comunicação pela rede global caso a estação local necessite enviar um pacote para outra estação, localizada através dos *bits* mais significativos, como mostrado na Tabela 3.

Figura 18 – Simplificação da rede desenvolvida



Fonte: Autora

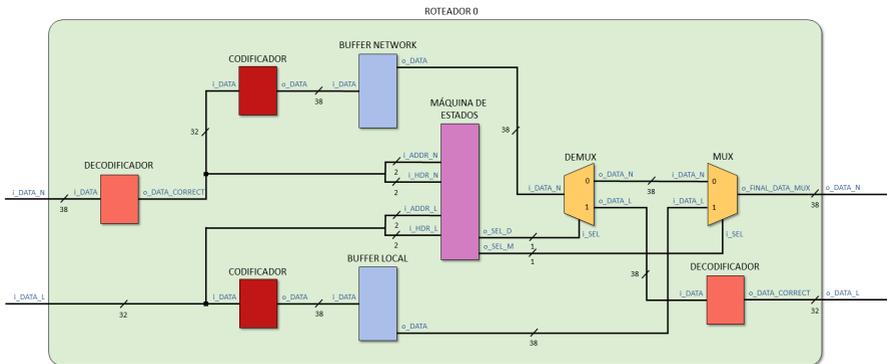
Para visualização de todos os sinais descritos na Tabela 5, verifique o *RTL Viewer* – uma visualização do *Register Transfer Level*, onde o comportamento do circuito é descrito em termos do fluxo de sinais, no Apêndice A.1.

4.1.4 Roteador

Para que os pacotes possam transitar pela rede desenvolvida de forma definida e ordenada, foi necessário projetar a arquitetura interna dos roteadores. À vista disso, criou-se um roteador e depois espelhou-se em mais três, alterando somente o fluxo dos sinais. Por consequência, abaixo há somente a explicação do Roteador 0, na Figura 19, que expõe os componentes internos com os sinais de transmissão dos *flits* e os seletores para combinação lógica. No Apêndice A.2, encontra-se a imagem completa da visão interna do Roteador 0.

Dentro do roteador, cada componente é responsável por executar sua função definida em relação ao destino do pacote e o roteador atual. Inicialmente, o *flit* entra no roteador (se este não estiver ocupado) por uma ligação com o processador local, isto é, o dado ainda contém 32 *bits*, visto que ele só vem codificado pela ligação com a *network*. No entanto, logo na entrada do roteador, é preciso codificá-lo para garantir sua integridade mesmo na presença de um ruído em seu trânsito pelo roteador. Assim, o *flit* agora codificado, com 38 *bits*, é armazenado no *Local Buffer*.

Figura 19 – Visão interna dos roteadores da rede (simplificada)



Fonte: Autora

Com o aval da ‘Máquina de Estados’³ finita (SM – *State Machine*), que analisa os quatro primeiros *bits* do *flit* inicial (32 *bits*), reconhecendo a indicação no pacote e seu destino, e se o próximo roteador está disponível, encaminha-se o *flit* armazenado no *buffer* para um ‘Multiplexador’⁴ (Mux – *Multiplexer*), o qual envia o *flit* codificado para a saída que o direciona ao próximo roteador.

Quando o *flit* chega ao próximo roteador, pela ligação da *network*, ele precisa ser decodificado (retorno a 32 *bits*) para verificação da equidade de seu valor e do real destino. Após, é codificado novamente e armazenado no *Network Buffer*. Com o aval da máquina de estados, ele é enviado para um ‘Demultiplexador’⁵ (Demux – *Demultiplexer*), o qual encaminha o dado para a saída local, precedida por um decodificador (se este for o destino real), ou para o multiplexador enviar à saída de transmissão pela rede. E assim sucessivamente até o direcionamento correto dos *flits* e finalização do pacote.

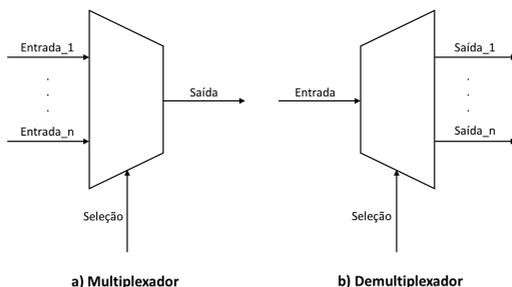
A Figura 21 é um fluxograma do comportamento do Roteador 0 quando há recebimento de dados, sejam vindos local ou globalmente.

³Modelo matemático que representa um circuito lógico. Sua definição é tida como uma máquina abstrata, a qual só pode estar em um estado por vez, o estado atual. Uma transição de estado acontece a partir da execução de uma condição, designada pelo projetista da máquina. Ao atingir um novo estado, ocorre uma ação, isto é, uma atividade pré-definida e que deve ser realizada em um momento estabelecido.

⁴Dispositivo que combina múltiplas entradas em um único terminal de dados. Vide Figura 20.

⁵Dispositivo que distribui o fluxo único de dados para uma das diversas saídas. Vide Figura 20.

Figura 20 – Multiplexador e demultiplexador



Fonte: Autora

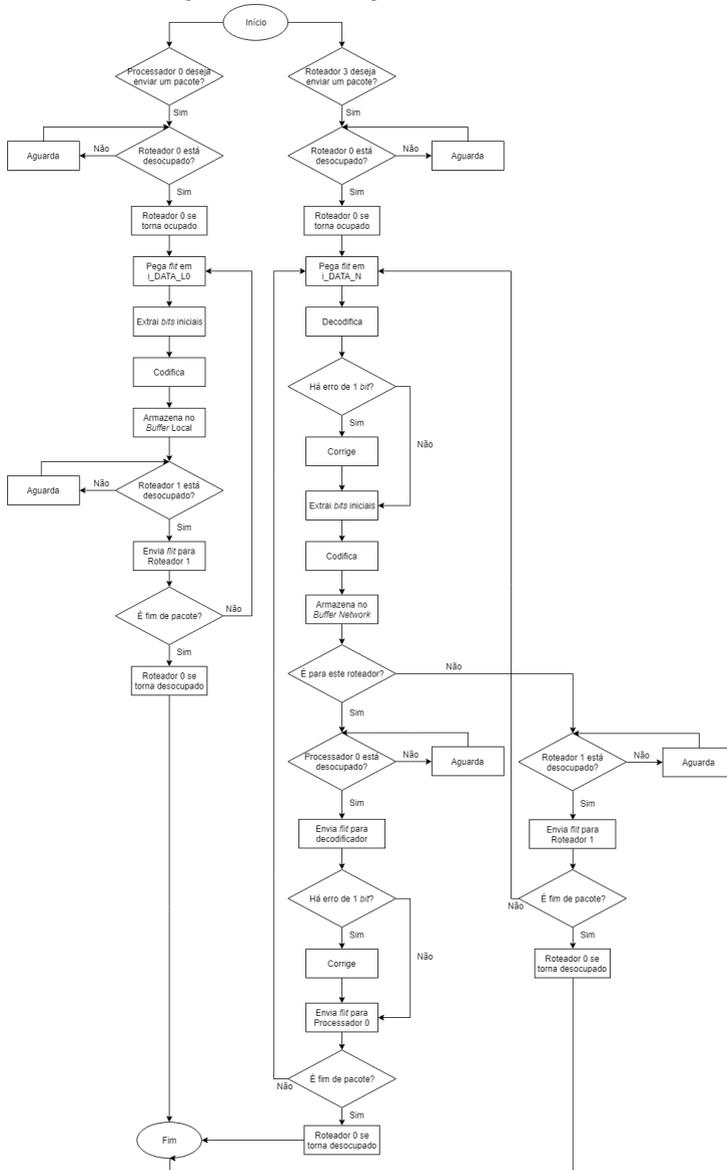
O componente que verifica as condições é a máquina de estados e ela foi desenvolvida neste projeto com 13 estados, nomeados como: *st_IDLE*, *st_WRITE_LN*, *st_WRITE_NN*, *st_WRITE_NL*, *st_DONE_LN*, *st_DONE_NN*, *st_DONE_NL*, *st_WAIT_PKG_NL*, *st_WAIT_PKG_NN*, *st_WAIT_PKG_LN*, *st_LAST_LN*, *st_LAST_NN* e *st_LAST_NL*.

Dentro da máquina de estados, o sinal denominado *i_HDR* (de *header* – cabeçalho) coleta os dois primeiros *bits* da mensagem, interessado em descobrir se o pacote está no início, meio ou fim. Já o sinal *i_ADDR* (de *address* – endereço) verifica o destino através dos 3º e 4º *bits*, conforme especificado na Tabela 3. Em cada roteador ela opera em busca de encaminhar o pacote completo e correto para o seu processador correspondente ou fazê-lo seguir adiante, sempre com base no *flit* decodificado. No Apêndice A.3, há a ilustração da máquina de estados para todos os roteadores da rede e as tabelas de condições para mudança de estado referente a cada um deles.

O estado *st_IDLE* é o estado de identificação inicial. O sistema começa limpando os *buffers* e os sinais de escrita e de seleção do multiplexador e do demultiplexador e aguardando uma entrada. Logo após a chegada de um *flit*, verifica-se qual *buffer* o armazenou, para identificar a origem – *network* ou local. Então, se o *i_HDR* relativo for diferente de “11”, a máquina permanece neste estado, pois ela só desperta seu fluxo a partir do começo de um pacote, nunca em seu meio ou fim. Se realmente for “Header”, ela segue para o próximo estado.

Caso o dado que chegou tenha vindo da rede e precise ser enviado para o processador do referido roteador, o estado atual vira *st_WRITE_NL* (escrever de *network* para local) e seleciona-se 1 na saída do demultiplexador, fazendo com que o dado armazenado no *buf-*

Figura 21 – Fluxograma do Roteador 0



Fonte: Autora

fer da rede já fique disponível na saída correta, esperando apenas o sinal de autorização da escrita. Caso ainda não seja o roteador de destino, o estado se torna *st_WRITE_NN* (escrever de *network* para *network*). Entretanto, caso o dado tenha vindo do processador, sabe-se que ele deve seguir pela rede, pois não pode voltar para ele mesmo, então o estado seguinte é o *st_WRITE_LN* (escrita de local para *network*).

Nos dois últimos casos, o *flit* deve sair do roteador em direção ao próximo. Assim, para que o dado que veio da rede possa sair na rede, a seleção de saída do demultiplexador é setada em 0, o que indica que o dado do *buffer* da rede será enviado à entrada 0 do multiplexador. Já, se o *flit* veio da local, o dado armazenado no *buffer* local fica disponível na entrada 1 do multiplexador.

Nos estados de escrita, se o próximo componente que for receber o *flit* não estiver ocupado, ativa-se o sinal *o_WRITE* relacionado, enviando o dado disponível com base na escolha da máquina de estados, e a envia para o estado que indica que o *flit* foi escrito: *st_DONE_NL*, *st_DONE_NN* ou *st_DONE_LN*. Se o componente estiver ocupado, a máquina permanece no estado de escrita, aguardando sua liberação. Nos estados *st_DONE*, limpa-se o *buffer* que continha os dados, desativa-se o sinal *o_WRITE* e envia a máquina para a espera de um novo *flit* do mesmo pacote.

Desta forma, os próximos estados que a máquina deve assumir são: *st_WAIT_PKG_NL*, *st_WAIT_PKG_NN* ou, então, *st_WAIT_PKG_LN*, em concordância com o caminho que o dado já vinha seguindo. Isto é, a máquina recebe novos dados e verifica: caso não seja fim de pacote, retorna para o *st_WRITE* relativo e refaz o caminho até esse estado novamente; caso a identificação de cabeçalho seja “10” – fim do pacote que iniciou esta transferência de *flits*, a máquina de estados segue para os estados finais.

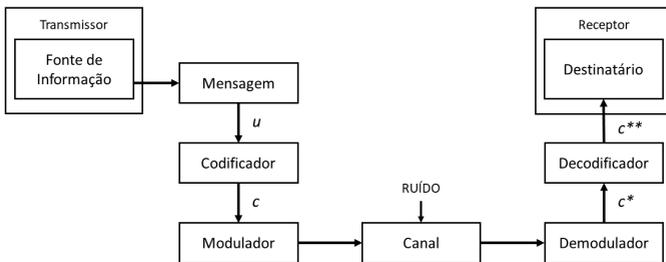
Os últimos estados escrevem a parte terminal do pacote enviado – o último *flit*, limpam seu *buffer* específico e retornam ao estado *st_IDLE* a espera de um novo pacote, seja vindo localmente ou via rede global. Tais estados são denominados como: *st_LAST_NL*, *st_LAST_NN* e *st_LAST_LN*. Note que, enquanto o pacote não é completamente enviado, um *flit* de um novo pacote não começa a sua transição pelo mesmo caminho, isto evita a desorientação da máquina de estados e a entrega desordenada dos pacotes.

4.1.5 Módulo Hamming

Há muitas maneiras de erros atingirem a transmissão de uma mensagem. Eles podem ser detectados e corrigidos por meio de métodos matemáticos, tornando mais confiáveis sistemas que os utilizam. Em casos onde distorções nas informações levam a um descontentamento do usuário, mas não uma danificação do sistema, como em conexões via telefone sem fio, com baixa qualidade da voz e perda de palavras, tais códigos são utilizados para melhorar o desempenho da comunicação. No entanto, sistemas que necessitam de exatidão quanto à informação recebida, como satélites, utilizam-nos a fim de assegurar seu funcionamento em situações insatisfatórias e comumente experienciadas em seus campos de atuação.

Partindo do princípio que uma fonte de informação necessita enviar uma mensagem a um destinatário, através de um canal ruidoso, Belli (2016) considera o seguinte processo pelo qual o dado deve passar: Codifica-se a mensagem e a modula, mediante alguma técnica específica de transmissão dependente do meio que será adotado. Conduz-se via um canal que, ao fim, a direcionará a um demodulador, seguido por um decodificador, com a finalidade de reconhecer a mensagem originalmente enviada e verificar a presença de erros, para, se correta, alcançar o destino. A Figura 22 demonstra a concepção apresentada.

Figura 22 – Diagrama de blocos de transmissão de dados



Fonte: Adaptado de Belli (2016)

Desta forma, uma mensagem (u) – que possui uma sequência de *bits* – entra em um bloco de codificação, responsável por transformá-la em uma nova mensagem (c) – a palavra-código – com a adição de novos *bits*, denominados *bits* de redundância ou de paridade. Se o sinal precisar da conversão analógico/digital ou digital/analógico, blocos de modulação e demodulação são adicionados ao sistema, permitindo que

esta informação consiga ser transmitida. Caso a conversão não seja necessária, o dado passa pelo canal, diretamente, a partir do codificador.

A mensagem que sai do canal (c^*) pode ser, ou não, a mesma que chegou nele (c), visto que ela pode ter sido atingida, ou não, por um ruído. Assim, após sua passagem, o decodificador encaminha ao destinatário a sequência binária decodificada (c^{**}), baseada em c^* . Da mesma maneira, e apesar de essencial a validação, ainda não se pode afirmar a equivalência entre u e c^{**} – no cenário em que codificador e decodificador somente transformam a sequência de *bits*, sem corrigi-la.

Consequentemente, ao utilizar códigos corretores de erro, busca-se que o dado acolhido pelo receptor seja igual ao encaminhado pelo transmissor (sem os *bits* adicionais de paridade). Com os métodos matemáticos disponíveis atualmente, um ou mais erros podem ser detectados e, em alguns casos, indicando até suas posições para correção. Porém, dependendo do algoritmo, somente um destes erros consegue ser corrigido. Para tanto, neste último caso, e com mais de um erro atingindo o sistema, o decodificador entregará uma mensagem corrompida.

O desenvolvimento deste trabalho contou com a escolha do **Código de Hamming**. Ele está qualificado para detectar erros de no máximo dois *bits*, revelando suas posições, embora seja um dos casos que corrige até um *bit*. Logo, se houver mais de uma alteração no mesmo pacote, mas em *flits* diferentes, isto é, erros de no máximo um *bit* por *flit*, poder-se-á recuperar segura e eficientemente a transferência e o armazenamento de dados na rede projetada.

O Código de Hamming é um ‘código de blocos’⁶ lineares fundamentado na adição de *bits* de paridade. Nele, cada *flit* é transformado em uma palavra-código, que tem seu comprimento (total de *bits* do *flit* codificado) estabelecido pela Equação 1, onde m é o comprimento da palavra de dados e x é o número fixo de pontos de controle.

$$N = m + x \tag{1}$$

Sendo assim, como cada *flit* que entra na rede possui 32 *bits*, $m = 32$. Para iniciar a inserção do código detector de erros, é preciso descobrir quantos *bits* de verificação (x) devem ser adicionados. De acordo com o Código de Hamming, estes *bits* ficam na posição em que os índices são potência de 2, ou seja, seu número correspondente em binário tem o *bit* mais significativo igual a 1 e os demais iguais a 0. De

⁶Particiona a mensagem original e armazena estes fragmentos em blocos de mesmo tamanho (m *bits*). Isto posto, há 2^m possibilidades distintas de mensagem.

1 até 32, tem-se 6 valores que se encaixam nestes resultados, designados $x_{\text{número}}$ neste trabalho, conforme a Tabela 6.

Tabela 6 – Potências de 2 com resultado no intervalo de 1 a 32

Potência	Posição do <i>bit</i> de paridade	Valor binário	Designação
2^0	1	00000001	x_1
2^1	2	00000010	x_2
2^2	4	00000100	x_3
2^3	8	00001000	x_4
2^4	16	00010000	x_5
2^5	32	00100000	x_6

Fonte: Elaborada pela autora

Desta forma, somam-se 6 *bits* aos 32 já contabilizados, visto que os de paridade foram fixados nas posições estratégicas explicitadas, mas ainda é preciso ter 32 *bits* de dados. Por conseguinte, tem-se os *bits* de paridade e *bits* de dados, respectivamente, nas posições conforme as Equações 2 e 3 e, enfim, o comprimento da palavra-código com 38 *bits* (Equação 4).

$$bits_{paridade} = \{1, 2, 4, 8, 16, 32\} = \mathbf{6\ bits} \quad (2)$$

$$bits_{dados} = \{3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38\} = \mathbf{32\ bits} \quad (3)$$

$$Total = bits_{paridade} + bits_{dados} = \mathbf{38\ bits} \quad (4)$$

Além disso, nomeou-se as demais posições como $m_{\text{número}}$, de forma que a Figura 23 ilustra as Equações 2 e 3 correlacionando nomenclaturas e posições. Com o propósito de apurar a presença ou ausência de erro, deve-se descobrir os valores de cada x , descritos em função de m . Para isso, deduz-se os m pela soma de suas posições através da soma dos respectivos x . A Tabela 7 expõe o cálculo.

Figura 23 – Posições de x e m

x_1	x_2	m_1	x_3	m_2	m_3	m_4	x_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	x_5	m_{12}	m_{13}	m_{14}
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

m_{15}	m_{16}	m_{17}	m_{18}	m_{19}	m_{20}	m_{21}	m_{22}	m_{23}	m_{24}	m_{25}	m_{26}	x_6	m_{27}	m_{28}	m_{29}	m_{30}	m_{31}	m_{32}
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38

Fonte: Autora

Tabela 7 – Definição de m em relação a x

	Soma	2^0	2^1	2^2	2^3	2^4	2^5
$m_1 = 3$	$1 + 2$	x_1	x_2				
$m_2 = 5$	$4 + 1$	x_1		x_3			
$m_3 = 6$	$4 + 2$		x_2	x_3			
$m_4 = 7$	$4 + 2 + 1$	x_1	x_2	x_3			
$m_5 = 9$	$8 + 1$	x_1			x_4		
$m_6 = 10$	$8 + 2$		x_2		x_4		
$m_7 = 11$	$8 + 2 + 1$	x_1	x_2		x_4		
$m_8 = 12$	$8 + 4$			x_3	x_4		
$m_9 = 13$	$8 + 4 + 1$	x_1		x_3	x_4		
$m_{10} = 14$	$8 + 4 + 2$		x_2	x_3	x_4		
$m_{11} = 15$	$8 + 4 + 2 + 1$	x_1	x_2	x_3	x_4		
$m_{12} = 17$	$16 + 1$	x_1				x_5	
$m_{13} = 18$	$16 + 2$		x_2			x_5	
$m_{14} = 19$	$16 + 2 + 1$	x_1	x_2			x_5	
$m_{15} = 20$	$16 + 4$			x_3		x_5	
$m_{16} = 21$	$16 + 4 + 1$	x_1		x_3		x_5	
$m_{17} = 22$	$16 + 4 + 2$		x_2	x_3		x_5	
$m_{18} = 23$	$16 + 4 + 2 + 1$	x_1	x_2	x_3		x_5	
$m_{19} = 24$	$16 + 8$				x_4	x_5	
$m_{20} = 25$	$16 + 8 + 1$	x_1			x_4	x_5	
$m_{21} = 26$	$16 + 8 + 2$		x_2		x_4	x_5	
$m_{22} = 27$	$16 + 8 + 2 + 1$	x_1	x_2		x_4	x_5	
$m_{23} = 28$	$16 + 8 + 4$			x_3	x_4	x_5	
$m_{24} = 29$	$16 + 8 + 4 + 1$	x_1		x_3	x_4	x_5	
$m_{25} = 30$	$16 + 8 + 4 + 2$		x_2	x_3	x_4	x_5	
$m_{26} = 31$	$16 + 8 + 4 + 2 + 1$	x_1	x_2	x_3	x_4	x_5	
$m_{27} = 33$	$32 + 1$	x_1					x_6
$m_{28} = 34$	$32 + 2$		x_2				x_6

Continuação na próxima página

Tabela 7 – Continuação da página anterior

	Soma	2^0	2^1	2^2	2^3	2^4	2^5
$m_{29} = 35$	$32 + 2 + 1$	x_1	x_2				x_6
$m_{30} = 36$	$32 + 4$			x_3			x_6
$m_{31} = 37$	$32 + 4 + 1$	x_1		x_3			x_6
$m_{32} = 38$	$32 + 4 + 2$		x_2	x_3			x_6

Fonte: Elaborada pela autora

Compreendendo a Tabela 7, é possível inferir os valores de x em função de m conforme a Tabela 8. Nesta, cada x corresponde a associação lógica *XOR* (ou exclusivo) dos m que o contém. Assim, pode-se montar a parcela do Código de Hamming responsável por indicar se há erro e em quais *bits*. A Tabela 9 esclarece o funcionamento da lógica programável *XOR*, a qual recebe duas entradas e produz um valor na saída, em que $C = A \oplus B$.

Tabela 8 – Definição de x em relação a m

x_1	$m_1 \oplus m_2 \oplus m_4 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} \oplus m_{12} \oplus m_{14} \oplus m_{16} \oplus$ $m_{18} \oplus m_{20} \oplus m_{22} \oplus m_{24} \oplus m_{26} \oplus m_{27} \oplus m_{29} \oplus m_{31}$
x_2	$m_1 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11} \oplus m_{13} \oplus m_{14} \oplus m_{17} \oplus$ $m_{18} \oplus m_{21} \oplus m_{22} \oplus m_{25} \oplus m_{26} \oplus m_{28} \oplus m_{29} \oplus m_{32}$
x_3	$m_2 \oplus m_3 \oplus m_4 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{15} \oplus m_{16} \oplus m_{17} \oplus$ $m_{18} \oplus m_{23} \oplus m_{24} \oplus m_{25} \oplus m_{26} \oplus m_{30} \oplus m_{31} \oplus m_{32}$
x_4	$m_5 \oplus m_6 \oplus m_7 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{19} \oplus m_{20} \oplus m_{21} \oplus$ $m_{22} \oplus m_{23} \oplus m_{24} \oplus m_{25} \oplus m_{26}$
x_5	$m_{12} \oplus m_{13} \oplus m_{14} \oplus m_{15} \oplus m_{16} \oplus m_{17} \oplus m_{18} \oplus m_{19} \oplus m_{20} \oplus$ $m_{21} \oplus m_{22} \oplus m_{23} \oplus m_{24} \oplus m_{25} \oplus m_{26}$
x_6	$m_{27} \oplus m_{28} \oplus m_{29} \oplus m_{30} \oplus m_{31} \oplus m_{32}$

Fonte: Elaborada pela autora

Para codificar e decodificar os *flits* enviados, através do Código de Hamming, utilizam-se dois codificadores e dois decodificadores, os quais também estão ilustrados na Figura 19. Sabendo-se que o *flit* possui 32 *bits* quando ele entra no codificador, calculam-se os *bits* de verificação x a partir dos 32 m recebidos, conforme visto na Tabela 8. Após este cálculo, ordenam-se todos os *bits*, x e m , conforme a Figura

Tabela 9 – Lógica programável *XOR*

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Fonte: Elaborada pela autora

23. Desta forma, o novo *flit* possui 38 *bits*, dos quais 6, estrategicamente alocados, são de redundância e 32, nas demais posições, de dados.

Quando se quer decodificar o *flit* de 38 *bits*, deve-se direcioná-lo ao decodificador que, inicialmente, analisará os *bits* em suas posições e, a partir disto, calculará a síndrome (se há erros nos dados recebidos), conforme a Tabela 10. Um *flit* sem erros possui síndrome igual a ‘0’ em todas as verificações dos *bits* P , os quais são um *XOR* de seu x e dos seus respectivos m . Caso algum dos P retorne o valor ‘1’, significa que foi identificada a ocorrência de um erro, vide Equação 5.

Tabela 10 – Definição de P em relação a x e m

P_1	$x_1 \oplus m_1 \oplus m_2 \oplus m_4 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} \oplus m_{12} \oplus m_{14} \oplus m_{16} \oplus$ $m_{18} \oplus m_{20} \oplus m_{22} \oplus m_{24} \oplus m_{26} \oplus m_{27} \oplus m_{29} \oplus m_{31}$
P_2	$x_2 \oplus m_1 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11} \oplus m_{13} \oplus m_{14} \oplus m_{17} \oplus$ $m_{18} \oplus m_{21} \oplus m_{22} \oplus m_{25} \oplus m_{26} \oplus m_{28} \oplus m_{29} \oplus m_{32}$
P_3	$x_3 \oplus m_2 \oplus m_3 \oplus m_4 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{15} \oplus m_{16} \oplus m_{17} \oplus$ $m_{18} \oplus m_{23} \oplus m_{24} \oplus m_{25} \oplus m_{26} \oplus m_{30} \oplus m_{31} \oplus m_{32}$
P_4	$x_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{19} \oplus m_{20} \oplus m_{21} \oplus$ $m_{22} \oplus m_{23} \oplus m_{24} \oplus m_{25} \oplus m_{26}$
P_5	$x_5 \oplus m_{12} \oplus m_{13} \oplus m_{14} \oplus m_{15} \oplus m_{16} \oplus m_{17} \oplus m_{18} \oplus m_{19} \oplus m_{20} \oplus$ $m_{21} \oplus m_{22} \oplus m_{23} \oplus m_{24} \oplus m_{25} \oplus m_{26}$
P_6	$x_6 \oplus m_{27} \oplus m_{28} \oplus m_{29} \oplus m_{30} \oplus m_{31} \oplus m_{32}$

Fonte: Elaborada pela autora

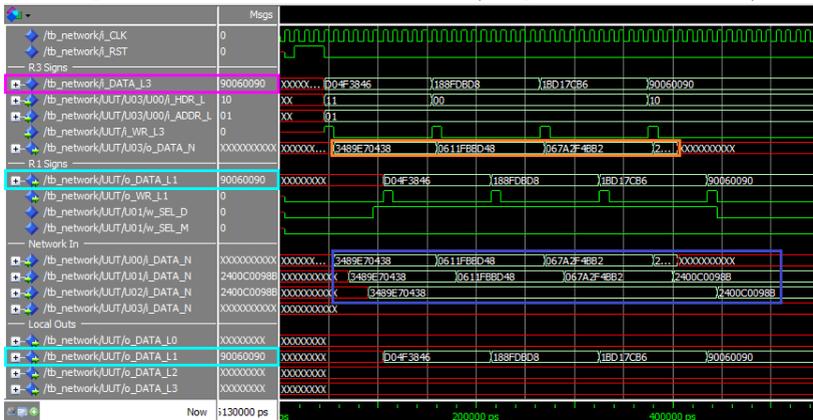
$$houve_erro = P_1 \text{ or } P_2 \text{ or } P_3 \text{ or } P_4 \text{ or } P_5 \text{ or } P_6 \quad (5)$$

É necessário validar todos os *bits* de síndrome para constatar o local do erro! A soma das posições dos *bits* de síndrome com erro identifica o *bit* com problema. Por exemplo, se os *bits* de síndrome nas posições 1, 2 e 8 indicam um erro, então, o *bit* 11 ($1 + 2 + 8$) foi atingido por uma falha. Importante destacar aqui que se apenas um *bit* de síndrome indicar erro, sabe-se que o erro está neste *bit*.

Na Figura 24, tem-se um exemplo de resultado completo da simulação de envio do pacote explicitado na Tabela 4, desde o Roteador 3 até o Roteador 1, sem a interferência de erros, somente testando a codificação e a decodificação de Hamming. Para simulações com erros vide 5.3.

Nessa figura, há a ilustração dos sinais globais de *clock* e *reset*, dos sinais locais de entrada (pacote decodificado, cabeçalho, destino e autorização de escrita) do roteador origem, bem como o resultado da codificação de cada *flit*; dos sinais locais de saída (pacote decodificado e autorização de escrita) e seleção de demux e mux do roteador destino; dos sinais de *network* das entradas em todos os roteadores (pacote codificado); e, por fim, dos sinais locais das saídas de todos os roteadores para seus processadores específicos.

Figura 24 – Sinais locais e de rede (Origem: 3 e Destino: 1)



Fonte: Autora

Pode-se perceber, por consequência, em ‘R3 Signs’ que inicialmente *i_HDR* recebeu o código de início de mensagem – *Header*, depois de *Payload* e, por fim, de *Tail*, de forma que todo o pacote saiu do Processador 3 e entrou localmente no Roteador 3 (ênfase rosa). Assim como, o endereço de destino, indicado em *i_ADDR* foi 01, isto é, o Roteador 1. A saída do Roteador 3, indica que os *flits* saíram para a

rede de forma codificada (laranja). A Tabela 11 representa os *bits* antes e depois da codificação de Hamming, em binário e em hexadecimal.

Tabela 11 – Pacote da Tabela 4 – Decodificado e Codificado

	Decodificado	Codificado
Binário	1101 0000 0100 1111 0011 1000 0100 0110	11 0100 1000 1001 1110 0111 0000 0100 0011 1000
	0001 1000 1000 1111 1101 1011 1101 1000	00 0110 0001 0001 1111 1011 1011 1101 0100 1000
	0001 1011 1101 0001 0111 1100 1011 0110	00 0110 0111 1010 0010 1111 0100 1011 1011 0010
	1001 0000 0000 0110 0000 0000 1001 0000	10 0100 0000 0000 1100 0000 0000 1001 1000 1011
Hexadecimal	D04F3846	3489E70438
	188FDBD8	611FBBD48
	1BD17CB6	67A2F4BB2
	90060090	2400C0098B

Fonte: Elaborada pela autora

Em ‘R1 *Signs*’, note que a seleção do demux do Roteador 1 foi 1, conforme convencionado anteriormente para a saída local, e o mux permaneceu no padrão 0, pois ele só é utilizado se o dado tiver de sair para a rede. Ademais, nos sinais da rede entre os roteadores (em ‘*Network In*’), visualiza-se o pacote codificado “caminhando” (destaque em azul escuro) – com a saída somente no roteador designado (visto em ‘*Local Outs*’ e ‘R1 *Signs*’ [azul claro]).

5 RESULTADOS EXPERIMENTAIS

Ao final do desenvolvimento deste trabalho, obteve-se o produto proposto: uma rede anelar, a qual interconecta quatro roteadores entre si e a seus respectivos processadores. Nela, há verificação e tratamento de erros leves, ocasionados por falhas temporárias. A fim de examinar a correta transmissão dos pacotes por toda a rede desenvolvida, executou-se um conjunto de verificações, as quais atestam a válida comunicação desta rede, com as mensagens recebidas exatamente iguais às enviadas. Inicialmente, os testes não englobaram dados interceptados por falhas.

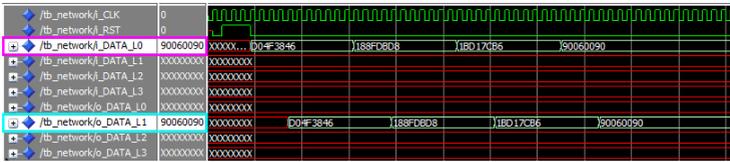
Na síntese da rede foi utilizado o FPGA da família Cyclone IV do fabricante Intel, modelo EP4CGX22CF19C6. Nos testes, estabeleceram-se processadores fictícios por meio de *test benches*, interligados aos roteadores, também implementados em VHDL no *software* Quartus II, e as simulações ocorreram através do *software* ModelSim da Altera, sem testes em um FPGA físico por limitação de tempo e equipamento. Sabe-se que se os *flits* chegaram corretos na saída do roteador final, a codificação e decodificação de Hamming aconteceram com sucesso, visto que logo que o *flit* chega ao primeiro roteador (via conexão local), ele é codificado antes de sair para a rede.

Para as simulações ilustradas, fez-se cada pacote transitar na rede, desde um roteador inicial para todos os demais, em tempo de simulação de 80 μ s. Ordenadamente e com uma visualização simplificada em relação à Figura 24 (mas mesma escala de tempo), percebe-se nas Figuras de 25 a 36 que os *flits* entraram localmente em um roteador, simulando o envio por um processador, e saíram localmente através do roteador escolhido para um *link* que deverá se conectar ao respectivo e real processador de destino futuramente.

Note que a carga útil dos *flits* é a mesma em todas estas simulações, alterando somente os *bits* iniciais que indicam o destino. Os dados estão dispostos no formato hexadecimal, para melhor visualização nos fluxos de sinais, mas podem ser convertidos para binário, como na Tabela 11 – resultando em valores no padrão especificado na Tabela 4. Ainda, estão destacadas em cada figura a entrada local do roteador fonte (rosa) e a saída local do roteador destino (azul claro).

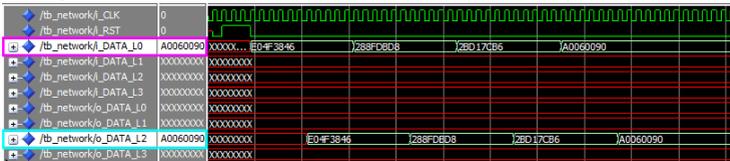
A partir das Figuras 25, 26 e 27, pode-se perceber que o pacote entra localmente no roteador R0, pois os *flits* aparecem na linha de *i_DATA_L0* em cada figura. Para os roteadores encaminharem os dados ao destino correto, os *bits* iniciais (3º e 4º) indicam-no, conforme a Tabela 12. Igualmente, inferem-se as Tabelas 13, 14 e 15.

Figura 25 – Envio de dado do Roteador 0 para o Roteador 1



Fonte: Autora

Figura 26 – Envio de dado do Roteador 0 para o Roteador 2



Fonte: Autora

Figura 27 – Envio de dado do Roteador 0 para o Roteador 3



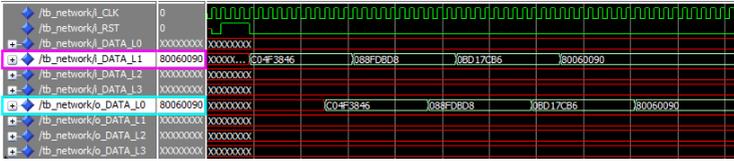
Fonte: Autora

Tabela 12 – Pacotes das Figuras 25, 26 e 27 (hexadecimal e binário)

	Hexadecimal	Binário
Figura 25	D04F3846	1101 0000 0100 1111 0011 1000 0100 0110
	188FD8B8	0001 1000 1000 1111 1101 1011 1101 1000
	1BD17CB6	0001 1011 1101 0001 0111 1100 1011 0110
	90060090	1001 0000 0000 0110 0000 0000 1001 0000
Figura 26	E04F3846	1110 0000 0100 1111 0011 1000 0100 0110
	288FD8B8	0010 1000 1000 1111 1101 1011 1101 1000
	2BD17CB6	0010 1011 1101 0001 0111 1100 1011 0110
	A0060090	1010 0000 0000 0110 0000 0000 1001 0000
Figura 27	F04F3846	1111 0000 0100 1111 0011 1000 0100 0110
	388FD8B8	0011 1000 1000 1111 1101 1011 1101 1000
	3BD17CB6	0011 1011 1101 0001 0111 1100 1011 0110
	B0060090	1011 0000 0000 0110 0000 0000 1001 0000

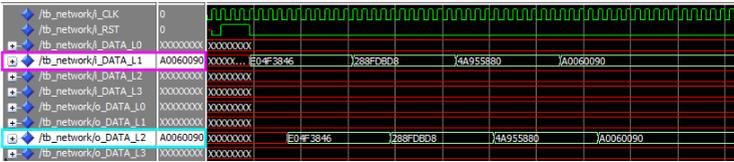
Fonte: Elaborada pela autora

Figura 28 – Envio de dado do Roteador 1 para o Roteador 0



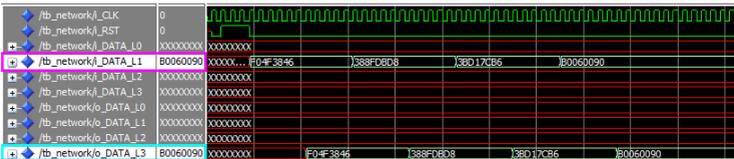
Fonte: Autora

Figura 29 – Envio de dado do Roteador 1 para o Roteador 2



Fonte: Autora

Figura 30 – Envio de dado do Roteador 1 para o Roteador 3



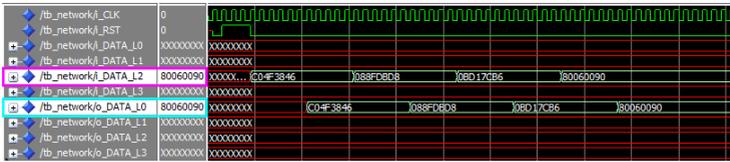
Fonte: Autora

Tabela 13 – Pacotes das Figuras 28, 29 e 30 (hexadecimal e binário)

	Hexadecimal	Binário
Figura 28	C04F3846	1100 0000 0100 1111 0011 1000 0100 0110
	088FDBD8	0000 1000 1000 1111 1101 1011 1101 1000
	0BD17CB6	0000 1011 1101 0001 0111 1100 1011 0110
	80060090	1000 0000 0000 0110 0000 0000 1001 0000
Figura 29	E04F3846	1110 0000 0100 1111 0011 1000 0100 0110
	288FDBD8	0010 1000 1000 1111 1101 1011 1101 1000
	2BD17CB6	0010 1011 1101 0001 0111 1100 1011 0110
	A0060090	1010 0000 0000 0110 0000 0000 1001 0000
Figura 30	F04F3846	1111 0000 0100 1111 0011 1000 0100 0110
	388FDBD8	0011 1000 1000 1111 1101 1011 1101 1000
	3BD17CB6	0011 1011 1101 0001 0111 1100 1011 0110
	B0060090	1011 0000 0000 0110 0000 0000 1001 0000

Fonte: Elaborada pela autora

Figura 31 – Envio de dado do Roteador 2 para o Roteador 0



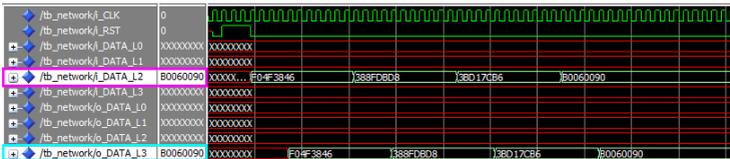
Fonte: Autora

Figura 32 – Envio de dado do Roteador 2 para o Roteador 1



Fonte: Autora

Figura 33 – Envio de dado do Roteador 2 para o Roteador 3



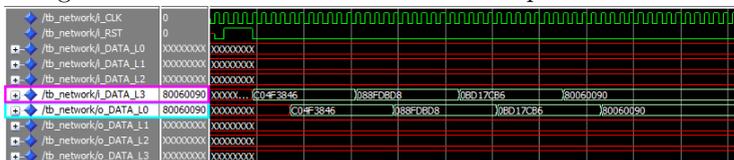
Fonte: Autora

Tabela 14 – Pacotes das Figuras 31, 32 e 33 (hexadecimal e binário)

	Hexadecimal	Binário
Figura 31	C04F3846	1100 0000 0100 1111 0011 1000 0100 0110
	088FD8B8	0000 1000 1000 1111 1101 1011 1101 1000
	0BD17CB6	0000 1011 1101 0001 0111 1100 1011 0110
	80060090	1000 0000 0000 0110 0000 0000 1001 0000
Figura 32	D04F3846	1101 0000 0100 1111 0011 1000 0100 0110
	188FD8B8	0001 1000 1000 1111 1101 1011 1101 1000
	1BD17CB6	0001 1011 1101 0001 0111 1100 1011 0110
	90060090	1001 0000 0000 0110 0000 0000 1001 0000
Figura 33	F04F3846	1111 0000 0100 1111 0011 1000 0100 0110
	388FD8B8	0011 1000 1000 1111 1101 1011 1101 1000
	3BD17CB6	0011 1011 1101 0001 0111 1100 1011 0110
	B0060090	1011 0000 0000 0110 0000 0000 1001 0000

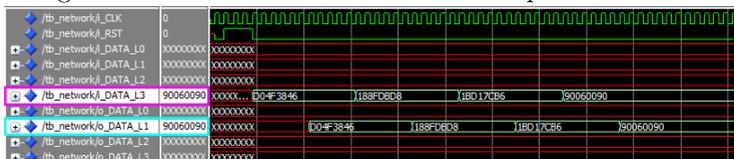
Fonte: Elaborada pela autora

Figura 34 – Envio de dado do Roteador 3 para o Roteador 0



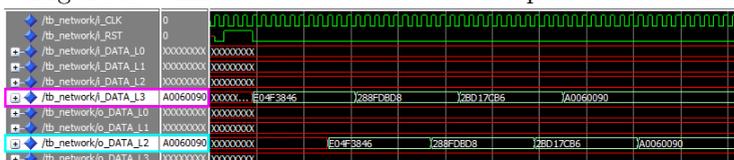
Fonte: Autora

Figura 35 – Envio de dado do Roteador 3 para o Roteador 1



Fonte: Autora

Figura 36 – Envio de dado do Roteador 3 para o Roteador 2



Fonte: Autora

Tabela 15 – Pacotes das Figuras 34, 35 e 36 (hexadecimal e binário)

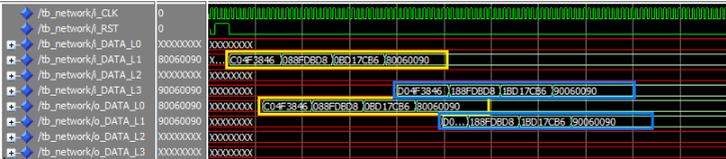
	Hexadecimal	Binário
Figura 34	C04F3846	1100 0000 0100 1111 0011 1000 0100 0110
	088FDBD8	0000 1000 1000 1111 1101 1011 1101 1000
	0BD17CB6	0000 1011 1101 0001 0111 1100 1011 0110
	80060090	1000 0000 0000 0110 0000 0000 1001 0000
Figura 35	D04F3846	1101 0000 0100 1111 0011 1000 0100 0110
	188FDBD8	0001 1000 1000 1111 1101 1011 1101 1000
	1BD17CB6	0001 1011 1101 0001 0111 1100 1011 0110
	90060090	1001 0000 0000 0110 0000 0000 1001 0000
Figura 36	E04F3846	1110 0000 0100 1111 0011 1000 0100 0110
	288FDBD8	0010 1000 1000 1111 1101 1011 1101 1000
	2BD17CB6	0010 1011 1101 0001 0111 1100 1011 0110
	A0060090	1010 0000 0000 0110 0000 0000 1001 0000

Fonte: Elaborada pela autora

Como se validou o envio e recebimento de dados por todos os caminhos possíveis individualmente, testou-se também o envio de pacotes saindo de processadores diferentes. É importante lembrar que é uma rede anelar simples, e, portanto, os pacotes não podem percorrer o mesmo caminho ao mesmo tempo, visto que só existe um canal de passagem dos dados entre cada roteador.

A Figura 37 ilustra um teste realizado enviando dois pacotes. O primeiro – com destaque amarelo – partindo de R1 em direção a R0 e o segundo – em azul, liberado após 400 ns – de R3 até R1. De forma similar, um teste com quatro pacotes é exposto na Figura 38. Cada um destes pacotes, destacados em cores diferentes, é enviado ao seu roteador contíguo a direita, de forma que os *flits* não precisam transitar por toda a rede até seu destino.

Figura 37 – Envio de 2 pacotes: R1-R0 e R3-R1



Fonte: Autora

Figura 38 – Envio de 4 pacotes: R0-R1, R1-R2, R2-R3 e R3-R0



Fonte: Autora

Por fim, a Figura 39 revela que, independente da ordem de envio dos pacotes e da distância dos roteadores, os *flits* chegam ordenadamente. É possível perceber que alguns pacotes, naturalmente, demoram mais do que outros para chegar ao destino. Esta questão está descrita na próxima seção.

Figura 39 – Envio de 4 pacotes: R0-R2, R2-R1, R1-R0 e R3-R1



Fonte: Autora

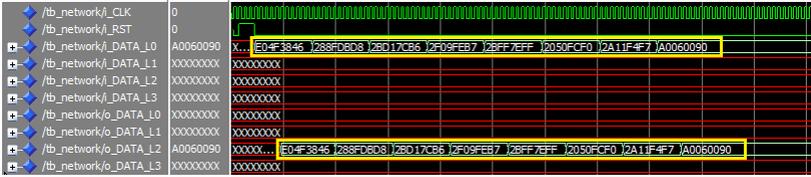
5.1 AVALIAÇÃO DE TRÁFEGO E LATÊNCIA

Em um ambiente onde a frequência de *clock* é de 10 MHz, com a simulação de 80 μs e um intervalo de 100 *ms* entre o envio de cada *flit* do mesmo pacote, a partir das Figuras de 25 a 36, pode-se observar a latência entre os roteadores. Caso o roteador destino seja contíguo à direita do roteador origem, cada *flit* demora 4 pulsos de *clock* para seu envio completo (seguindo o sentido padronizado para a transmissão do dado: sentido horário). Caso haja um roteador entre a origem e o destino, os mesmos *flits* levam 6 pulsos de *clock*. E, por último, caso o roteador destino seja contíguo à esquerda do roteador origem, isto é, o último roteador pelo qual o dado ainda não passou, levam-se 8 pulsos de *clock* para o fim da propagação de cada *flit*.

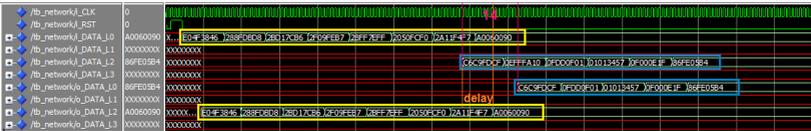
Quando há mais de um pacote circulando pela rede, pode ser que a latência seja maior em algum momento, pois, se o roteador estiver ocupado com uma mensagem, a que chegar depois deverá aguardar a liberação do acesso. Assim, nas Figuras 37 e 38, como nenhum pacote chega antes da finalização do seu anterior, o tempo resultante é semelhante ao descrito acima. Isto indica que os quatro roteadores implementados estão funcionando de forma compatível, independente da quantidade de *flits* que cada pacote contém e das distâncias a serem percorridas.

Ainda, é imprescindível notar, na Figura 39, que o segundo pacote (em azul) é enviado ao roteador R2 antes de o último *flit* do primeiro pacote (em amarelo) chegar ao seu destino. Perceba a leve precipitação da entrada do *flit* “D04F3846” em relação ao *flit* “A0060090”. Neste caso, mesmo que R1 seja contíguo à esquerda de R2, isto é, necessite de 8 ciclos de *clock* para transferir completamente o pacote, há de se acrescentar o tempo de espera. Por conseguinte, a transferência do segundo pacote demorou 10 pulsos de *clock*. Os demais pacotes permaneceram em seu tempo padrão, em virtude do ausente período de atraso.

Outrossim, realizaram-se testes com pacotes maiores, pois até então cada pacote possuía somente quatro *flits*. Desta forma, a partir da Figura 40, que envia apenas um pacote dividido em oito *flits*, do roteador R0 para o R2, conseguiu-se validar o padrão de latência averiguado anteriormente, independente da quantidade de *flits* por pacote. A Figura 41 contém este mesmo envio, com a adição de um novo pacote seccionado em seis *flits*, com origem no roteador R2 e destino em R0 (ambos em uso na primeira mensagem), para teste com mais de um pacote circulando pela rede.

Figura 40 – Envio de 1 pacote: R0-R2 (8 *flits*)

Fonte: Autora

Figura 41 – Envio de 2 pacotes: R0-R2 (8 *flits*) e R2-R0 (6 *flits*)

Fonte: Autora

Note, na Figura 41, que o segundo pacote (azul) foi enviado antes do final do primeiro (amarelo), ocasionando um tempo de espera. Quando se conseguiu enviar o primeiro *flit* do segundo pacote, percebeu-se que o tráfego do *flit* em questão, que deveria ocorrer em 8 ciclos de *clock*, ocorreu em 14, devido ao tempo adicional. Além disso, dado que o segundo *flit* do segundo pacote (0EFFFA10) fora enviado antes de o primeiro deixar o roteador R2, ele foi perdido (sobrescrito).

Nesta rede não há qualquer tipo de reenvio dos *flits* de pacotes por causa de tempo esgotado, pois os roteadores estão programados para armazenar cada *flit* somente após a liberação que indica a finalização da verificação do anterior. Assim, após a transmissão do *flit* “C6C9FDCF”, o novo *flit* disponível na entrada de R2 era “0FDD0F01” – *flit* seguinte ao perdido, ocasionando a ausência do *flit* “0EFFFA10” na saída de R0 (fato que não impossibilitou o término da transmissão do pacote).

5.2 CONSUMO DE SILÍCIO

Para levantar o consumo de silício, utilizaram-se os relatórios de consumo disponibilizados pela ferramenta Quartus II, a qual sintetiza códigos VHDL para FPGA do fabricante Intel. Nestes relatórios, o consumo é apresentado em termos de elementos lógicos (LE – *Logic Elements*) e registradores (*Registers*).

Em um total de 109.424 LE disponíveis no FPGA utilizado para síntese, descrito no início deste Capítulo, o consumo de Elementos Ló-

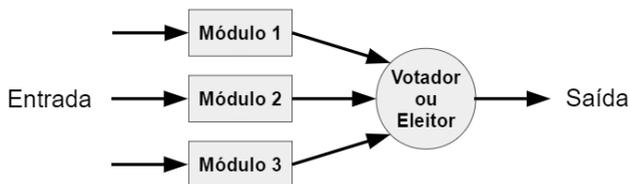
gicos somente da rede, sem qualquer técnica de tolerância a falhas, foi de 1.032 LE, o que se refere a menos de 1% do valor acessível. Para comparação, foram implementadas duas redes, uma delas baseada na Tripla Redundância de *Hardware* (TMR) e a outra usando o Código de Hamming.

5.2.1 Implementação com TMR

A TMR é um dos exemplos mais comuns de redundância de *hardware* passiva. Nela, os elementos redundantes são usados para mascarar falhas, triplicando um ou mais componentes. Assim, todos os elementos executam a mesma tarefa e o resultado é determinado por votação – por maioria (2 em 1) ou por seleção do valor médio (LYONS; VANDERKULK, 1962).

Para esta implementação, os circuitos foram triplicados e a saída de cada um deles, todas conectadas em série com o “votador” ou “eleitor”, foi definida baseada em uma simples função de maioria. Isto significa que, se um dos circuitos (ou módulos, como mostrados na Figura 42) sofrer uma SEU, os outros dois provavelmente terão os resultados corretos e o votador irá escolher a saída comum à preponderância dos três módulos redundantes.

Figura 42 – Circuito “votador” na implementação de Tripla Redundância de *Hardware*

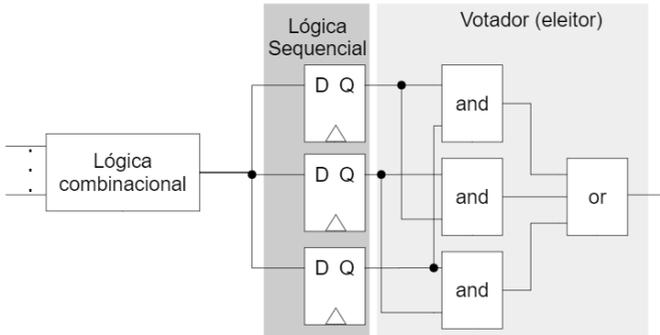


Fonte: Adaptado de Goloubeva et al. (2006)

Deve ser observado que a redundância aumenta o número de componentes do sistema e, quanto maior o número de componentes, maior a possibilidade de falha (WEBER, 2003). A Figura 43 ilustra a implementação de um votador utilizando registradores e portas lógicas. Nesta figura, assume-se que a lógica combinacional está triplicada e cada uma de suas saídas é conectada a um registrador (*flip-flop*), elementos vulneráveis a SEU, do votador.

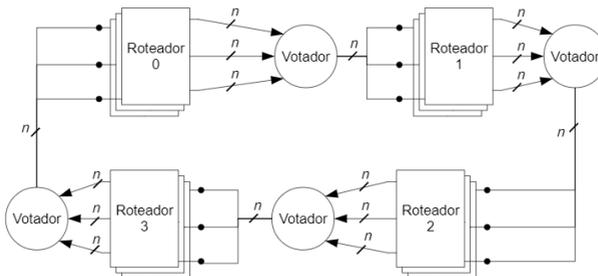
A implementação da TMR poderia ter sido realizada de duas

Figura 43 – Circuito “votador” – Implementação interna



Fonte: Adaptado de Kastensmidt, Carro e Reis (2006)

maneiras distintas. A primeira delas é triplicando cada componente interno de cada roteador da rede. A segunda seria triplicando os roteadores, de modo que se algum SEE acontecer em um componente interno de um determinado roteador, os outros dois fornecerão o valor correto. No primeiro caso, os componentes internos tornariam a implementação pesada e, provavelmente, ter-se-ia maior gasto de silício. Portanto, foi adotada a segunda opção, conforme a Figura 44 expõe. Note que são três *links* que enviam os barramentos de dados paralelos aos votadores (indicados pela letra *n*).

Figura 44 – Rede *intra-chip* com a técnica de TMR

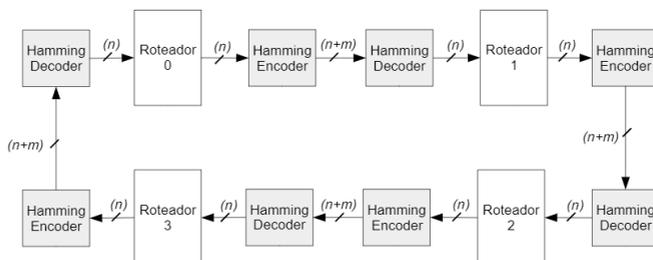
Fonte: Autora

5.2.2 Implementação com código de Hamming

Nesta abordagem, a rede é construída com apenas quatro roteadores, diferentemente da implementação com TMR – onde foram implementados doze roteadores, além dos circuitos de votação.

Aqui, implementou-se o barramento de saída de cada roteador passando por um codificador com código de Hamming (extraído do roteador, a fim de se analisar atentamente os componentes internos *Encoder* e *Decoder*). O *link* é conectado na entrada do roteador subsequente, passando por um decodificador (também externo) com código de Hamming. A topologia desta implementação é mostrada na Figura 45 com os blocos decodificador (*Hamming Decoder*) e codificador (*Hamming Encoder*) na cor cinza. A quantidade de sinais em cada barramento é dada nesta figura pela expressão $(n + m)$, onde $n = 32$ é a quantidade de *bits* de dados e $m = 6$ dos *bits* relativos a codificação de Hamming.

Figura 45 – Rede *intra-chip* com codificação de Hamming



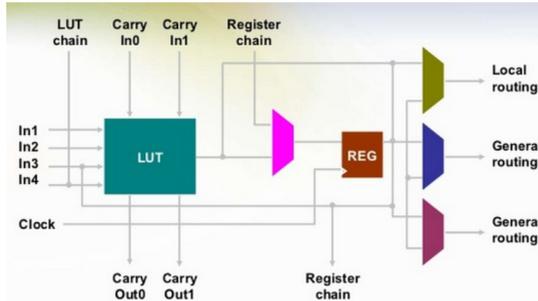
Fonte: Autora

5.2.3 Consumo de silício das duas soluções

Um elemento lógico é constituído essencialmente por três tipos de circuitos eletrônicos: memória tipo LUT (acrônimo de “*Look Up Table*”), *flip-flops* e multiplexadores, além de alguma lógica combinacional. A Figura 46 mostra um diagrama em blocos de um Elemento Lógico da família Cyclone IV da Intel, a nível de exemplo. Perceba a presença dos elementos citados acima.

A Tabela 16 mostra o consumo de silício para três versões da rede *intra-chip*: uma rede simples com quatro roteadores sem tolerância a falhas, outra rede com tripla redundância de *hardware* e uma terceira

Figura 46 – Diagrama ilustrando um Elemento Lógico da família Cyclone IV (Fabricante: Intel)



Fonte: Disponível em: <https://www.intel.com/content/www/us/en/products/programmable/fpga/cyclone-iv.html>

contendo codificação e decodificação de Hamming. Note que o consumo da solução com Hamming praticamente dobra (99%) o consumo em relação a configuração original e simples. Ainda assim, ele é bem menor do que a solução com TMR (233%). Este resultado era esperado, uma vez que a implementação da NoC TMR triplica os roteadores, consequentemente, triplicando seus elementos internos. A Figura 47 ilustra graficamente o consumo dado pela Tabela 16.

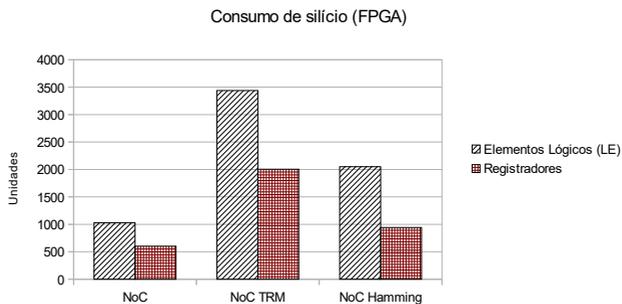
Tabela 16 – Consumo de silício para os três modelos de rede *intra-chip*: normal, com TMR e com Hamming

Rede	Elementos Lógicos (LE)	Registradores	% (LE)
NoC	1.032	602	•
NoC TMR	3.437	2.005	233
NoC Hamming	2.050	942	99

Fonte: Autora

Uma outra avaliação de consumo de silício pode ser feita considerando os componentes internos de um roteador, incluindo o codificador e o decodificador de Hamming. Através da Tabela 17 e, consequentemente, da Figura 48, observe que o decodificador consome mais elementos lógicos do que o próprio roteador em si, o que era esperado por se tratar de um componente com circuitos combinacionais em sua maioria. O *encoder* de Hamming é puramente combinacional, por isso o valor zero para a quantidade de registradores consumida na síntese.

Figura 47 – Comparação do consumo de silício entre as redes analisadas



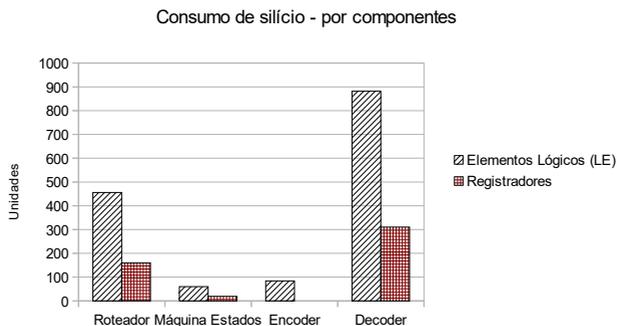
Fonte: Autora

Tabela 17 – Consumo de silício para os componentes internos da rede com Hamming

Componentes	Elementos Lógicos (LE)	Registradores
Roteador	456	160
Máquina Estados	60	19
Codificador	84	0
Decodificador	882	310

Fonte: Autora

Figura 48 – Comparação do consumo de silício dos componentes internos da rede com Hamming



Fonte: Autora

5.3 SIMULAÇÃO DE FALHAS COM MODELSIM

Intencionando-se perante a garantia de estabilidade da rede proposta em um cenário de radiações, foram realizados diversos testes com injeções de falhas, semelhantemente ao exposto no trabalho de Travesini (2018). Por meio do auxílio de um *script* também estruturado em VHDL, pode-se incorporar um ruído arbitrário à simulação da rede no *software* ModelSim. Para isto, deixou-se a rede sempre enviando dados. Inicialmente, os *flits* eram enviados dos roteadores 1 e 3 para, respectivamente, os roteadores 2 e 0, e, após 400 ns, os roteadores 2 e 0 enviavam um novo pacote aos roteadores 3 e 1, nesta ordem.

Como este sistema só consegue corrigir falhas de no máximo 1 *bit* errôneo por *flit*, em cada teste, injetou-se uma única falha temporária (invertendo o valor lógico do sinal alvo) em uma posição aleatória de um roteador aleatório, dos quatro que a rede dispõe, e em um tempo também aleatório, dentro do período de transmissão dos pacotes. Seus resultados deveriam corresponder a falhas de dado incorreto na saída, caso o teste fosse com a rede simples, o que não aconteceu devido à detecção e à correção implementadas.

Nas figuras referentes às verificações, estão destacados: a entrada e a saída local do dado decodificado (em amarelo), o dado codificado com a presença do erro em um *bit* (em rosa) e qual *bit*, dos 38 disponíveis, foi atingido (em laranja). O tempo do ruído também varia de acordo com o teste, alguns duram somente o tempo de um *flit* e outros atingem a rede por um período maior, danificando outros *flits*, ainda que na mesma posição.

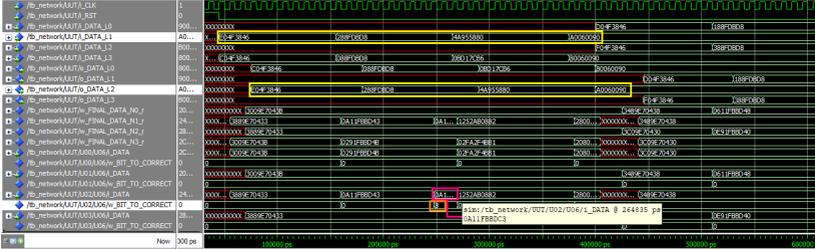
O primeiro teste, ilustrado na Figura 49, teve seu ruído adicionado aos 247,494 ps no *flit* codificado que estava saindo do Roteador 1. O *bit* que corretamente era 0, tornou-se 1. Perceba que apesar de o ruído acontecer e modificar o valor do oitavo *bit*, transformando o *flit* “0A11FBBD43” em “0A11FBBDC3” (vide Tabela 18), a rede conseguiu corrigi-lo e entregar no destino o mesmo pacote que foi enviado na origem.

Tabela 18 – Injeção de falha no 8º *bit*

Pacote em Hexadecimal	Pacote em Binário
0A11FBBD43	00 1010 0001 0001 1111 1011 1011 1101 0100 0011
0A11FBBDC3	00 1010 0001 0001 1111 1011 1011 1101 1100 0011

Fonte: Elaborada pela Autora

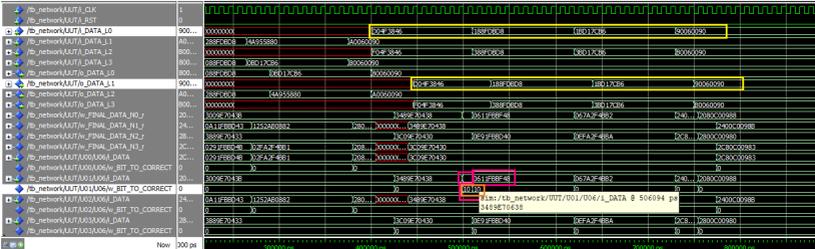
Figura 49 – Injeção de falha na saída do Roteador 1



Fonte: Autora

Um exemplo de falha que permaneceu por mais tempo, danificando dois *flits* está na Figura 50. Aos 498,498 ps, o dado que estava na saída do Roteador 0 teve o 10º bit alterado de 0 para 1. O *flit* que era “3489E70438” virou “3489E70638” e o segundo: “0611FBBD48”, tornou-se “0611FBBF48” (vide Tabela 19). Outros exemplos seguem nas Figuras 51 e 52 e suas respectivas tabelas (20 e 21).

Figura 50 – Injeção de falha na saída do Roteador 0



Fonte: Autora

Tabela 19 – Injeção de falha no 10º bit

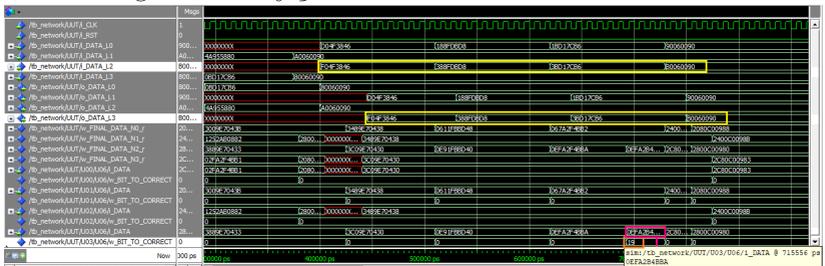
Pacote em Hexadecimal	Pacote em Binário
3489E70438	00 0010 0100 0100 11110 0111 0000 0100 0011 1000
3489E70638	00 0010 0100 0100 1111 0011 1000 00110 0011 1000
0611FBBD48	00 0110 0001 0001 1111 1011 1011 1101 0100 1000
0611FBBF48	00 0110 0001 0001 1111 1011 1011 1111 0100 1000

Fonte: Elaborada pela Autora

Assim, tomando como base a Figura 19, se uma SEU atinge um dos componentes de armazenamento do roteador (*Buffer Network* ou *Buffer Local*), vê-se que o endereço de destino *i_ADDR* e o indicador de início/fim da mensagem (*i_HDR*) serão enviados corretamente para a máquina de estados, uma vez que eles não vêm diretamente dos *buffers*. Estes repassarão o *flit* codificado, porém modificado pelo erro, ou seja, incorreto, para o demux ou o mux, conforme deve seguir seu caminho. Caso o destino não seja o roteador atual, o dado sairá pela rede e, no momento em que ele chegar ao decodificador do próximo roteador, verificar-se-á a presença de erro e se corrigirá. Caso o pacote seja para o processador referente ao roteador em questão, o *flit* com *bit* errôneo será corrigido no decodificador, o qual envia o dado para a saída local, sendo, portanto, também enviado sem erros ao processador correto.

Na hipótese do ruído afetar o dado presente em um circuito lógico combinacional, isto é, um SET, será mais difícil gerar um erro leve, pois para isto será necessário que o pulso transiente chegue na entrada do elemento de memória durante a borda de transição do *clock*. Mesmo assim, ao passar pelo decodificador (seja na saída local ou na entrada global), o erro será percebido e corrigido. Ademais, como a probabilidade de uma falha temporária acontecer duas vezes no mesmo *flit* é excessivamente baixa, somente com a detecção e a correção elaboradas, a rede conseguirá se restabelecer com equilíbrio aceitável entre as variáveis de tempo, custo e confiabilidade.

Figura 51 – Injeção de falha na saída do Roteador 2



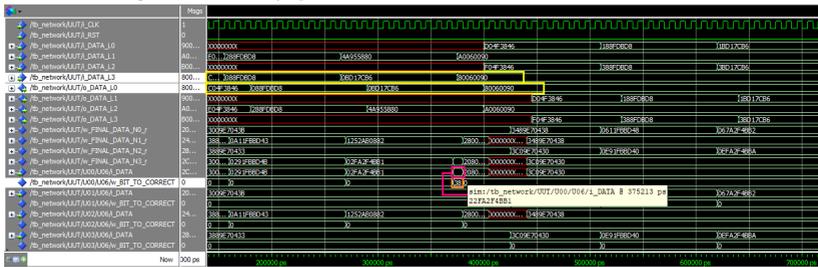
Fonte: Autora

Tabela 20 – Injeção de falha no 19º bit

Pacote em Hexadecimal	Pacote em Binário
0EFA2F4BBA	00 1110 1111 1010 0010 1111 0100 1011 1011 1010
0EFA2B4BBA	00 1110 1111 1010 0010 1011 0100 1011 1011 1010

Fonte: Elaborada pela Autora

Figura 52 – Injeção de falha na saída do Roteador 3



Fonte: Autora

Tabela 21 – Injeção de falha no 38º bit

Pacote em Hexadecimal	Pacote em Binário
02FA2F4BB1	00 0010 1111 1010 0010 1111 0100 1011 1011 0001
22FA2F4BB1	10 0010 1111 1010 0010 1111 0100 1011 1011 0001

Fonte: Elaborada pela Autora

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo o desenvolvimento de uma rede *intra-chip* com até quatro processadores *soft-core*, tolerante a falhas temporárias causadas por radiações encontradas no contexto de nanosatélites.

Para atingir o objetivo geral, fizeram-se análises e estudos de conteúdos como sistemas e redes *intra-chip* e suas tolerâncias a falhas, aferindo a viabilidade da rede idealizada. Com esta confirmação, definiu-se o escopo do projeto, bem como questões de topologia, modos de chaveamento e roteamento de dados que seriam utilizados.

A construção da rede se deu via um *software* que auxilia na implementação de programações para baixo nível. Definiu-se a arquitetura interna dos roteadores da rede, criou-se cada componente pertencente a eles e se implementou suas funcionalidades, seguindo padrões estabelecidos para formato dos pacotes de dados e recebimento e envio de outros sinais.

Sabe-se que em ambientes de radiação, dispositivos eletrônicos podem ser impactados por diferentes tipos de partículas, causando-lhes leves ou graves consequências no armazenamento e/ou transmissão de mensagens, dependendo de local e tempo da corrupção do dado. Quando somente uma partícula atinge um componente do dispositivo, nomeia-se como *Single Event Effect* e pode ser dividido em *Single Event Upset* (componente de armazenamento) e *Single Event Transient* (circuito lógico). Ambos casos foram considerados neste trabalho.

O estudo revelou que em dispositivos mais novos, geralmente miniaturizados e com tensões operacionais mais baixas, as ocorrências se acentuam. Compreendendo-se que sistemas de nanosatélites necessitam de exatidão quanto a informação recebida, pois a retransmissão de pacotes neste cenário é muito dificultosa, recorreu-se à utilização de métodos matemáticos para assegurar a conformidade das mensagens. Assim, com a intenção de tratar falhas temporárias, aperfeiçoou-se a rede, implementando os códigos corretores e receptores de erro, mediante técnica de Hamming.

A fim de se avaliar a rede desenvolvida, elaborou-se um cenário de testes em situações variadas, repercutindo em aceitáveis níveis de tráfego e latência. Além disto, os resultados de consumo de silício em termos de elementos lógicos e registradores foram comparados aos resultados de outra solução para detecção e correção de erros, obtendo melhores resultados.

Sabendo-se que o trabalho em questão trata de tolerância a falhas transitórias, causadas por radiação, em redes *intra-chip* de nanossatélites, especificaram-se as melhorias realizadas em relação à Tabela 2, condizente com o seguinte:

- O desenvolvimento do NanosatC-Br1 (COSTA; DURÃO; CRS, 2011), trata de tolerância a falhas em nanossatélites, porém com módulos interligados sem o auxílio de SoCs. Além disso, consegue corrigir falhas causadas por radiação, via comando e controle remoto, o que não é o foco deste trabalho. Desta forma, utilizar a rede *intra-chip* desenvolvida dentro do nanossatélite ocupará menos espaço, será mais confiável no sentido de os componentes estarem soldados (difícil desmembração ou mau contato em caso de colisão) e não será necessário controle via estações terrestres, pois ela se reestabelecerá automaticamente.
- O trabalho de Martins et al. (2018) tem foco em falhas permanentes em nanossatélites. Para reduzir os custos de silício e processamento de dados, a autora se concentrou na tolerância a falhas transitórias.
- Os trabalhos de Travessini (2018) e Villa (2018) não tratam de nanossatélites, especificamente, mas visam diminuir o custo de silício em sistemas que interligam processadores e podem vir a ser atingidos por feixes espaciais temporários. À vista disso e pensando na linha de Villa (2018), no sentido de redundância temporal, consente-se que a utilização de processadores como o LEON3, utilizado em ambos trabalhos, é uma solução conveniente e, para ampliar a eficiência do sistema, conectá-los através de uma NoC teria uma ótima conclusão.
- E, comparando-se à pesquisa de Fochi (2015), buscou-se focar em apenas uma técnica de tolerância, reduzindo os custos e consumos de silício, uma vez que ele estuda e experimenta diversas estratégias, sem enfoque na causa da falha. Assim, a autora adentra seu empenho em aplicações relativas a NoCs lesionadas por radiações, no contexto de nanossatélites.

Ao final, percebeu-se que é possível estabelecer uma NoC com estrutura simples e tolerante a falhas temporárias em ambiente espacial. Bem como, esta solução pode ser aplicada em dispositivos *off-the-shelf* para a construção de um nanossatélite, visto que, comparada a outras soluções da literatura, sua arquitetura respondeu satisfatoriamente aos experimentos e seu consumo de silício é reduzido.

6.1 TRABALHOS FUTUROS

Com o propósito de complementar e dar continuidade a este estudo, sugerem-se as seguintes propostas para trabalhos futuros:

- Aplicar a rede desenvolvida no nanossatélite FloripaSat.
- Implementar a adição de caminho no sentido anti-horário, mantendo a topologia anelar, para o dado percorrer na hipótese de sobrecarga de acesso.
- Implementar confirmação de recebimento das mensagens e reenvio em caso de *time out*.
- Estudar e incorporar técnicas de tolerância a falhas intermitentes e permanentes com aplicação nesta rede para se adequar a outros cenários.

REFERÊNCIAS

AMORY, A. d. M. **Lógica e escalonamento de teste para sistemas com redes intra-chip baseadas em topologia de malha**. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul (UFRGS), 2007.

ANGIOLINI, F. et al. Contrasting a NoC and a traditional interconnect fabric with layout awareness. In: **Proceedings of the Design Automation Test in Europe Conference**. Munich, DEU: IEEE, 2006. v. 1, p. 1–6. ISSN 1530-1591.

ANTUNES, E. V. **Ações satelitais brasileiras: um estudo sobre o emprego dos nanossatélites no Brasil**. 2017. Dissertação (Mestrado) — Universidade Estadual de Campinas (UNICAMP).

AVIŽIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, Jan 2004. ISSN 1545-5971.

BAUMANN, R. C. Radiation-induced soft errors in advanced semiconductor technologies. **IEEE Transactions on Device and Materials Reliability**, v. 5, n. 3, p. 305–316, Sep. 2005. ISSN 1530-4388.

BAUMANN, R. C. Soft errors in advanced computer systems. **IEEE Design Test of Computers**, v. 22, n. 3, p. 258–266, May. 2005. ISSN 0740-7475.

BELLI, L. S. **Implementação em hardware de um código de Hamming (72,64)**. 2016. Trabalho de Conclusão de Curso (Graduação) — Universidade Federal do Paraná (UFPR).

BENEDETTO, J. M. et al. Digital single event transient trends with technology node scaling. **IEEE Transactions on Nuclear Science**, v. 53, n. 6, p. 3462–3465, Dec. 2006. ISSN 0018-9499.

BENINI, L.; MICHELI, G. D. Networks on chip: a new paradigm for systems on chip design. In: **Proceedings Design, Automation and Test in Europe Conference and Exhibition**. Paris, FRA: IEEE, 2002. p. 418–419. ISBN 0-7695-1471-5. ISSN 1530-1591.

BERTOZZI, D. The data-link layer in noc design. In: **Networks on Chips: Technology and Tools**. New York: Morgan Kaufmann Publishers, 2006. cap. 4. Disponível em: <<http://books.google.com.br/books?id=IHHTmSBcoGICpg=PA75lpg>>.

BJERREGAARD, T.; MAHADEVAN, S. A survey of research and practices of network-on-chip. **ACM Computing Surveys (CSUR)**, v. 38, n. 1, p. 1, 2006.

BORKAR, S. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. **IEEE Micro**, v. 25, n. 6, p. 10–16, Nov. 2005. ISSN 0272-1732.

CARVALHO, M. J. M. de et al. Estudo de estratégias de mitigação de detritos espaciais para uma constelação de nano satélites de coleta de dados ambientais. In: **Proceedings of 16th ISU Anual International Symposium**. Strasbourg, FRA: INPE, 2012.

CASARIL, L.; SOUZA, B. L. de; BRITO, A. G. Simulação orbital de nanosatélites. **Proceeding Series of the Brazilian Society of Computational and Applied Mathematics**, v. 6, n. 1, 2018.

COSTA, F. W. **O programa espacial brasileiro: o desenvolvimento de foguetes lançadores de satélites e sub-orbitais em um ambiente internacional adverso**. 2004. Trabalho de Conclusão de Curso (Graduação) — Centro Universitário de Brasília (UniCEUB).

COSTA, L. L.; DURÃO, O. S. C.; CRS. Projeto de um aplicativo de bordo para missão NanosatC-Br. Relatório final de projeto de iniciação científica PIBIC/INPE-CNPq/MCT, Santa Maria, 2011.

CUNHA, B. Os componentes “COTS” nos sistemas digitais operativos. *Revista Passadiço*, p. 60–61, 2005. Disponível em: <<http://livrozilla.com/doc/293594>>.

DALLY, W. J.; TOWLES, B. P. **Principles and practices of interconnection networks**. San Francisco, USA: Elsevier, 2004.

DUATO, J.; YALAMANCHILI, S.; NI, L. **Interconnection networks: an engineering approach**. Inc., USA: Morgan Kaufmann, 2003.

FILHO, J. B. P. Análises e alternativas para o programa espacial brasileiro. São José dos Campos, BRA, 2015. Disponível em: <<http://www.sindct.org.br/files/Bezerra.pdf>>.

FOCHI, V. M. **Técnicas de tolerância a falhas aplicadas a redes intra-chip**. 2015. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS).

FRANKE, L. L. C. et al. Simulação térmica do primeiro Cubesat brasileiro: NanosatC-Br1. **XXI Congresso Nacional de Estudantes de Engenharia Mecânica**, 2014.

GADLAGE, M. J. et al. Single event transient pulse widths in digital microcircuits. **IEEE Transactions on Nuclear Science**, v. 51, n. 6, Dec. 2004. ISSN 0018-9499.

GOLOUBEVA, O. et al. **Software-implemented hardware fault tolerance**. Torino, ITA: Springer Science & Business Media, 2006.

GRECU, C. et al. On-line fault detection and location for NoC interconnects. In: **Proceedings of the 12th International On-Line Testing Symposium (IOLTS'06)**. Lake Como, ITA: IEEE, 2006. p. 145–150. ISBN 0-7695-2620-9. ISSN 1942-9398.

HELVAJIAN, H.; JANSON, S. W. **Small satellites: past, present, and future**. USA: Aerospace Press, 2008. 876 p. ISBN 1884989225, 9781884989223.

HEMANI, A. et al. Network on a chip: An architecture for billion transistor era. In: **Proceedings of the 18th NorChip Conference**. Turku, FIN: IEEE, 2000. v. 31, p. 11.

HUSSEIN, J.; SWIFT, G. Mitigating Single-Event Upsets. *CiteSeerX*, 2012.

INPE. Primeiro cubesat brasileiro completa 4 anos em operação. 2018. Disponível em: <http://www.inpe.br/crs/nanosat/noticias-/noticia.php?Cod_Noticia=4801>.

KASTENSMIDT, F. L.; CARRO, L.; REIS, R. A. da L. **Fault-tolerance techniques for SRAM-based FPGAs**. Dordrecht, NED: Springer, 2006.

KASTENSMIDT, F. L. et al. On the optimal design of triple modular redundancy logic for SRAM-based FPGAs. In: **Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2**. Washington, USA: IEEE Computer Society, 2005. p. 1290–1295. ISBN 0-7695-2288-2. ISSN 1530-1591.

KIM, S.; SOMANI, A. K. On-line integrity monitoring of microprocessor control logic. In: **Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors (ICCD)**. Austin, USA: IEEE, 2001. p. 314–319. ISSN 1063-6404.

KOHLER, A.; SCHLEY, G.; RADETZKI, M. Fault tolerant network on chip switching with graceful performance degradation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 29, n. 6, p. 883–896, June 2010. ISSN 0278-0070.

KUMAR, A.; MANJUNATH, D.; KURI, J. **Communication networking: an analytical approach**. San Francisco, USA: Elsevier, 2004.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: uma abordagem top-down**. Boston, USA: Person Addison Wesley, 2006.

LABEL, K. A.; GATES, M. Single-event-effect mitigation from a system perspective. **IEEE Transactions on Nuclear Science**, v. 43, n. 2, p. 654–660, Apr. 1996. ISSN 0018-9499.

LAHIRI, K.; RAGHUNATHAN, A.; DEY, S. Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. In: **Proceedings of the 14th International Conference on VLSI Design**. Bangalore, IND: IEEE, 2001. p. 29–35. ISSN 1063-9667.

LEE, H. G. et al. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. In: **Proceedings of the Transactions on Design Automation of Electronic Systems (TODAES)**. New York, USA: ACM, 2007. v. 12, n. 23.

LESEA, A. et al. The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs. **IEEE Transactions on Device and Materials Reliability**, v. 5, n. 3, p. 317–328, Sep. 2005. ISSN 1530-4388.

LYONS, R. E.; VANDERKULK, W. The use of triple-modular redundancy to improve computer reliability. **IBM Journal of Research and Development**, IBM, v. 6, n. 2, p. 200–209, Apr. 1962. ISSN 0018-8646.

MÂNICA, T. R. **Missão NanosatC-Br1 – Análise de telemetria e resultados em órbita**. 2018. Trabalho de Conclusão de Curso (Graduação) — Universidade Federal de Santa Maria.

MANTOVANI, L. Q. et al. NanosatC-Br2, 2 unit CubeSat, power analysis, solar flux prediction, design and 3D. Flight model from the UFSM & INPE's NanosatC-Br, CubeSat development program, 2017.

MARAL, G.; BOUSQUET, M. **Satellite communications systems: Systems, Techniques and Technology**. Hoboken, USA: John Wiley & Sons, 2011.

MARCULESCU, R. et al. Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 28, n. 1, p. 3–21, Jan. 2009. ISSN 0278-0070.

MARTINS, V. M. G. et al. A dynamic partial reconfiguration design flow for permanent faults mitigation in FPGAs. **Microelectronics Reliability**, Elsevier, v. 83, p. 50–63, 2018.

MATIAS-PEREIRA, J. **Manual de metodologia da pesquisa científica**. 3. ed. São Paulo: Atlas, 2012.

MELLO, A. V. d. **Qualidade de serviço em redes intra-chip: implementação e avaliação sobre a rede Hermes**. 2007. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS).

MELLO, A. V. de; MÖLLER; HELENO, L. **Arquitetura multiprocessada em SoCs: estudo de diferentes topologias de conexão**. 2003. Trabalho de Conclusão de Curso (Graduação) — Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS).

MICHELI, G. D.; BENINI, L. **Networks on chips: Technology and Tools**. San Francisco, USA: Academic Press, 2006. ISBN 978-0-12-370521-1. Disponível em: <<http://books.google.com.br/books?id=IHHTmSBcoGICprintsec=frontcoverdq>>.

MOHAPATRA, P. Wormhole routing techniques for directly connected multicomputer systems. **ACM Computing Surveys (CSUR)**, ACM, v. 30, n. 3, p. 374–410, Sept. 1998.

MOORE, G. E. Cramming more components onto integrated circuits – Reprinted from *Electronics*, volume 38, number 8, april 19, 1965, pp.114 ff. **IEEE Solid-State Circuits Society Newsletter**, v. 11, n. 3, p. 33–35, Sept. 2006. ISSN 1098-4232.

NASA. What are SmallSats and CubeSats? 2015. Disponível em: <<http://www.nasa.gov/content/what-are-smallsats-and-cubesats>>.

OGRAS, U.; HU, J.; MARCULESCU, R. Key research problems in NoC design: A holistic perspective. In: **Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis**. New York, USA: ACM, 2005. p. 69–74. ISBN 1-59593-161-9.

OWENS, J. D. et al. Research challenges for on-chip interconnection networks. **IEEE Micro**, v. 27, n. 5, p. 96–108, Sept. 2007. ISSN 0272-1732.

PALMA, J. C. S. **Reduzindo o consumo de potência em redes intra-chip através de esquemas de codificação de dados**. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul (UFRGS), 2007.

PEREIRA, T. F. et al. Mechanisms to provide fault tolerance to a Network-on-Chip. **IEEE Latin America Transactions**, v. 15, n. 6, p. 1034–1042, June 2017. ISSN 1548-0992.

PETERSON, L. L.; DAVIE, B. S. **Computer Networks, A System Approach**. 3. ed. New York, USA: Morgan Kaufmann, 2003.

POIVEY, C.; GRANDJEAN, M.; GUERRE, F. X. Radiation characterization of Microsemi ProASIC3 flash FPGA family. In: **Radiation Effects Data Workshop**. Las Vegas, USA: IEEE, 2011. p. 1–5. ISSN 2154-0535.

RADETZKI, M. et al. Methods for fault tolerance in Networks-on-Chip. **ACM Computing Surveys (CSUR)**, ACM, v. 46, n. 1, p. 1–38, 2013. ISSN 0360-0300.

ROSS, J. **Redes de computadores**. Julio Ross, 2008. ISBN 9788561226046. Disponível em: <<http://books.google.com.br/books?hl=pt-BR&lr=id=mFhCHLDzaPgCoi=fndpg=PA5dqv>>.

SAJID, M. et al. Single Event Upset rate determination for 65nm SRAM bit-cell in LEO radiation environments. **Microelectronics Reliability**, Elsevier, v. 78, p. 11 – 16, 2017. ISSN 0026-2714.

SANTOS, D. P. et al. Constelação de Nanossatélites para fins de Telecomunicações. **Proceeding Series of the Brazilian Society of Computational and Applied Mathematics**, v. 6, n. 2, 2018.

SHIROMA, W. et al. Cubesats: A bright future for nanosatellites. **Open Engineering**, Versita, v. 1, n. 1, p. 9–15, 2011.

SILVA, A. H. L. d. et al. **Implementação e avaliação de métodos para confiabilidade de redes intra-chip**. 2010. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS).

SOININEN, J.-P.; HEUSALA, H. A design methodology for NOC-based systems. In: **Networks on Chip**. Norwell, USA: Kluwer Academic Publishers, 2003. p. 19–38.

SORIN, D. **Fault Tolerant Computer Architecture**. San Rafael, USA: Morgan & Claypool, 2009. ISBN 9781598299540.

STEPHENSON, H. The Next Big Thing Is Small. Academy of Program/Project & Engineering Leadership – NASA, 2010.

Disponível em:

<<http://appel.nasa.gov/2010/04/12/the-next-big-thing-is-small/>>.

TRAVESSINI, R. **Low overhead soft error reliability improvement for soft processors**. 2018. 90 p. Dissertação (Mestrado) — Universidade Federal de Santa Catarina (UFSC).

VILLA, P. **Reliability enhanced microprocessor architecture for the on-board computer of future satellites**. 2018. Dissertação (Mestrado) — Universidade Federal de Santa Catarina (UFSC).

VILLELA, T.; BRANDÃO, A.; LEONARDI, R. Cubesats e oportunidades para o setor espacial brasileiro. **Parcerias Estratégicas**, CGEE, v. 21, n. 42, p. 91–114, 2016.

WACHTER, E. et al. Topology-agnostic fault-tolerant NoC routing method. **Design, Automation & Test in Europe Conference & Exhibition (DATE)**, p. 1595–1600, March 2013. ISSN 1530-1591.

WALTERS, K. H. et al. Multicore SoC for on-board payload signal processing. In: **Proceedings of the 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)**. San Diego, USA: IEEE, 2011. p. 17–21. ISBN 9781457705984.

WANG, J. J. et al. SRAM based re-programmable FPGA for space applications. **IEEE Transactions on Nuclear Science**, v. 46, n. 6, p. 1728–1735, Dec. 1999. ISSN 0018-9499.

WEBER, T. S. Tolerância a falhas: conceitos e exemplos. Apostila do Programa de Pós-Graduação — Instituto de Informática – UFRGS. Porto Alegre, 2003.

ZHANG, W. et al. Comparison Research between XY and Odd-Even Routing Algorithm of a 2-Dimension 3x3 Mesh Topology Network-on-Chip. In: **Global Congress on Intelligent Systems**. Xiamen, CHN: IEEE, 2009. p. 329–333.

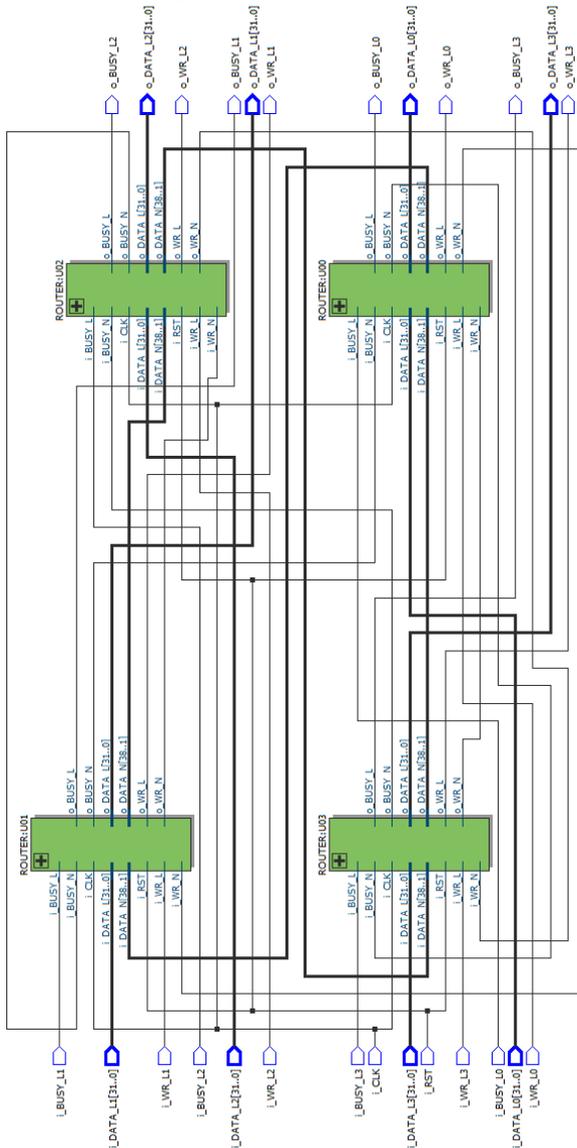
ZIMMER, H.; JANTSCH, A. A fault model notation and error-control scheme for switch-to-switch buses in a Network-on-Chip. In: **First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis**. New York, USA: ACM, 2003. p. 188–193. ISBN 1-58113-742-7.

ZIMMERMANN, H. OSI reference model – The ISO model of architecture for open systems interconnection. **IEEE Transactions on communications**, v. 28, n. 4, p. 425–432, Apr. 1980.

APÊNDICE A – Rede e Roteadores – Visualização completa

A.1 REDE

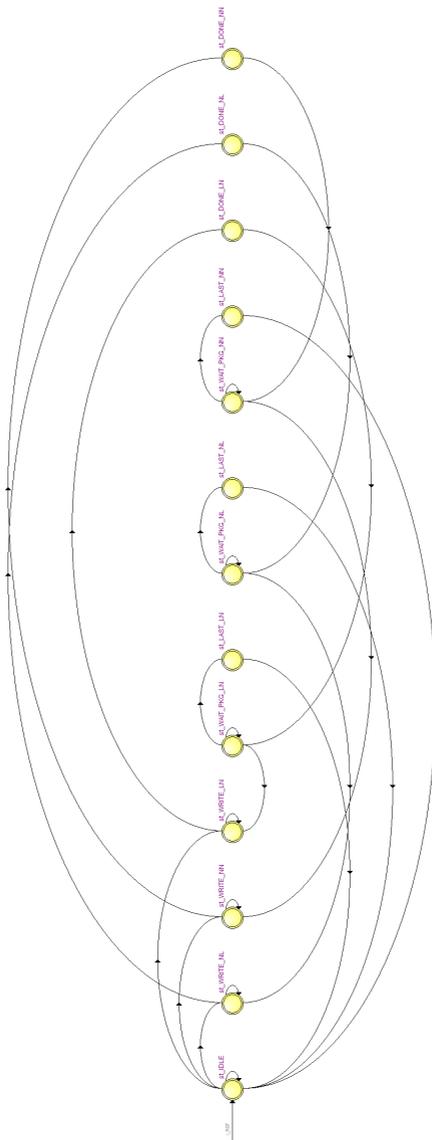
Figura 53 – Rede implementada



Fonte: Autora

A.3 MÁQUINA DE ESTADOS

Figura 55 – Estados e suas transições



Fonte: Autora

Tabela 22 – Condições para mudança de estado no Roteador 0

Estado de Origem	Estado de Destino	Condição
<i>st_IDLE</i>	<i>st_IDLE</i>	<p>(<i>!i_BUSY_BL</i> and <i>!i_BUSY_BN</i>) or (<i>!i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_N[0]</i>) or (<i>!i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_N="10"</i>) or (<i>i_BUSY_BL</i> and <i>!i_BUSY_BN</i> and <i>!i_HDR_L[0]</i>) or (<i>i_BUSY_BL</i> and <i>!i_BUSY_BN</i> and <i>i_HDR_L="10"</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_L[0]</i> and <i>!i_HDR_N[0]</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_L[0]</i> and <i>i_HDR_N="10"</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_L="10"</i> and <i>!i_HDR_N[0]</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_L="10"</i> and <i>i_HDR_N="10"</i>)</p>
<i>st_IDLE</i>	<i>st_WRITE_NL</i>	<p><i>i_BUSY_BN</i> and <i>i_HDR_N="11"</i> and <i>i_ADDR_N="00"</i></p>
<i>st_IDLE</i>	<i>st_WRITE_NN</i>	<p>(<i>i_HDR_N="11"</i> and <i>i_ADDR_N="01"</i> and <i>i_BUSY_BN</i>) or (<i>i_HDR_N="11"</i> and <i>i_ADDR_N[0]</i> and <i>i_BUSY_BN</i>)</p>

Continuação na próxima página

Tabela 22 – Continuação da página anterior

Estado de Origem	Estado de Destino	Condição
<i>st_IDLE</i>	<i>st_WRITE_LN</i>	$(i_HDR_L="11" \text{ and } i_BUSY_BL \text{ and } !i_BUSY_BN)$ or $(i_HDR_L="11" \text{ and } !i_HDR_N[0] \text{ and } i_BUSY_BL \text{ and } i_BUSY_BN)$ or $(i_HDR_L="11" \text{ and } i_HDR_N="10" \text{ and } i_BUSY_BL \text{ and } i_BUSY_BN)$
<i>st_WRITE_NL</i>	<i>st_WRITE_NL</i>	<i>i_BUSY_L</i>
<i>st_WRITE_NL</i>	<i>st_DONE_NL</i>	$!i_BUSY_L$
<i>st_WRITE_NN</i>	<i>st_WRITE_NN</i>	<i>i_BUSY_N</i>
<i>st_WRITE_NN</i>	<i>st_DONE_NN</i>	$!i_BUSY_N$
<i>st_WRITE_LN</i>	<i>st_WRITE_LN</i>	<i>i_BUSY_N</i>
<i>st_WRITE_LN</i>	<i>st_DONE_LN</i>	$!i_BUSY_N$
<i>st_DONE_NL</i>	<i>st_WAIT_PKG_NL</i>	
<i>st_DONE_NN</i>	<i>st_WAIT_PKG_NN</i>	
<i>st_DONE_LN</i>	<i>st_WAIT_PKG_LN</i>	
<i>st_WAIT_PKG_NL</i>	<i>st_WRITE_NL</i>	$(i_HDR_N="00" \text{ and } i_BUSY_BN \text{ and } !i_BUSY_L)$ or $(i_HDR_N[0] \text{ and } i_BUSY_BN \text{ and } !i_BUSY_L)$
<i>st_WAIT_PKG_NL</i>	<i>st_WAIT_PKG_NL</i>	$!i_BUSY_BN$ or $(i_BUSY_BN \text{ and } i_BUSY_L)$
<i>st_WAIT_PKG_NL</i>	<i>st_LAST_NL</i>	<i>i_HDR_N="01" and i_BUSY_BN and !i_BUSY_L</i>
<i>st_WAIT_PKG_NN</i>	<i>st_WRITE_NN</i>	$(i_HDR_N="00" \text{ and } !i_BUSY_N \text{ and } i_BUSY_BN)$ or $(i_HDR_N[0] \text{ and } !i_BUSY_N \text{ and } i_BUSY_BN)$
<i>st_WAIT_PKG_NN</i>	<i>st_WAIT_PKG_NN</i>	$(!i_BUSY_N \text{ and } !i_BUSY_BN)$ or <i>i_BUSY_N</i>

Continuação na próxima página

Tabela 22 – *Continuação da página anterior*

Estado de Origem	Estado de Destino	Condição
<i>st_WAIT_PKG_NN</i>	<i>st_LAST_NN</i>	<i>i_HDR_N="01" and !i_BUSY_N and i_BUSY_BN</i>
<i>st_WAIT_PKG_LN</i>	<i>st_WRITE_LN</i>	<i>(i_HDR_L="00" and !i_BUSY_N and i_BUSY_BL) or (i_HDR_L[0] and !i_BUSY_N and i_BUSY_BL)</i>
<i>st_WAIT_PKG_LN</i>	<i>st_WAIT_PKG_LN</i>	<i>(!i_BUSY_N and !i_BUSY_BL) or i_BUSY_N</i>
<i>st_WAIT_PKG_LN</i>	<i>st_LAST_LN</i>	<i>i_HDR_L="01" and !i_BUSY_N and i_BUSY_BL</i>

Fonte: Elaborada pela autora

Tabela 23 – Condições para mudança de estado no Roteador 1

Estado de Origem	Estado de Destino	Condição
<i>st_IDLE</i>	<i>st_IDLE</i>	<p>(<i>i_BUSY_BL</i> and <i>!i_BUSY_BN</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_N[0]</i>) or (<i>!i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_N="10"</i>) or (<i>i_BUSY_BL</i> and <i>!i_BUSY_BN</i> and <i>!i_HDR_L[0]</i>) or (<i>i_BUSY_BL</i> and <i>!i_BUSY_BN</i> and <i>i_HDR_L="10"</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_L[0]</i> and <i>!i_HDR_N[0]</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_L[0]</i> and <i>i_HDR_N="10"</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_L="10"</i> and <i>!i_HDR_N[0]</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_L="10"</i> and <i>i_HDR_N="10"</i>)</p>
<i>st_IDLE</i>	<i>st_WRITE_NL</i>	<p><i>i_BUSY_BN</i> and <i>i_HDR_N="11"</i> and <i>i_ADDR_N="10"</i></p>
<i>st_IDLE</i>	<i>st_WRITE_NN</i>	<p>(<i>i_HDR_N="11"</i> and <i>i_BUSY_BN</i> and <i>!i_ADDR_N[0]</i>) or (<i>i_HDR_N="11"</i> and <i>i_BUSY_BN</i> and <i>i_ADDR_N="11"</i>)</p>

Continuação na próxima página

Tabela 23 – Continuação da página anterior

Estado de Origem	Estado de Destino	Condição
<i>st_IDLE</i>	<i>st_WRITE_LN</i>	(<i>i_HDR_L</i> ="11" and <i>i_BUSY_BL</i> and ! <i>i_BUSY_BN</i>) or (<i>i_HDR_L</i> ="11" and ! <i>i_HDR_N</i> [0] and <i>i_BUSY_BL</i> and <i>i_BUSY_BN</i>) or (<i>i_HDR_L</i> ="11" and <i>i_HDR_N</i> ="10" and <i>i_BUSY_BL</i> and <i>i_BUSY_BN</i>)
<i>st_WRITE_NL</i>	<i>st_WRITE_NL</i>	<i>i_BUSY_L</i>
<i>st_WRITE_NL</i>	<i>st_DONE_NL</i>	! <i>i_BUSY_L</i>
<i>st_WRITE_NN</i>	<i>st_WRITE_NN</i>	<i>i_BUSY_N</i>
<i>st_WRITE_NN</i>	<i>st_DONE_NN</i>	! <i>i_BUSY_N</i>
<i>st_WRITE_LN</i>	<i>st_WRITE_LN</i>	<i>i_BUSY_N</i>
<i>st_WRITE_LN</i>	<i>st_DONE_LN</i>	! <i>i_BUSY_N</i>
<i>st_DONE_NL</i>	<i>st_WAIT_PKG_NL</i>	
<i>st_DONE_NN</i>	<i>st_WAIT_PKG_NN</i>	
<i>st_DONE_LN</i>	<i>st_WAIT_PKG_LN</i>	
<i>st_WAIT_PKG_NL</i>	<i>st_WRITE_NL</i>	(<i>i_HDR_N</i> ="00" and <i>i_BUSY_BN</i> and ! <i>i_BUSY_L</i>) or (<i>i_HDR_N</i> [0] and <i>i_BUSY_BN</i> and ! <i>i_BUSY_L</i>)
<i>st_WAIT_PKG_NL</i>	<i>st_WAIT_PKG_NL</i>	! <i>i_BUSY_BN</i> or (<i>i_BUSY_BN</i> and <i>i_BUSY_L</i>)
<i>st_WAIT_PKG_NL</i>	<i>st_LAST_NL</i>	<i>i_HDR_N</i> ="01" and <i>i_BUSY_BN</i> and ! <i>i_BUSY_L</i>
<i>st_WAIT_PKG_NN</i>	<i>st_WRITE_NN</i>	(<i>i_HDR_N</i> ="00" and ! <i>i_BUSY_N</i> and <i>i_BUSY_BN</i>) or (<i>i_HDR_N</i> [0] and ! <i>i_BUSY_N</i> and <i>i_BUSY_BN</i>)
<i>st_WAIT_PKG_NN</i>	<i>st_WAIT_PKG_NN</i>	(! <i>i_BUSY_N</i> and ! <i>i_BUSY_BN</i>) or <i>i_BUSY_N</i>

Continuação na próxima página

Tabela 23 – Continuação da página anterior

Estado de Origem	Estado de Destino	Condição
<i>st_WAIT_PKG_NN</i>	<i>st_LAST_NN</i>	<i>i_HDR_N</i> ="01" and ! <i>i_BUSY_N</i> and <i>i_BUSY_BN</i>
<i>st_WAIT_PKG_LN</i>	<i>st_WRITE_LN</i>	(<i>i_HDR_L</i> ="00" and ! <i>i_BUSY_N</i> and <i>i_BUSY_BL</i>) or (<i>i_HDR_L</i> [0] and ! <i>i_BUSY_N</i> and <i>i_BUSY_BL</i>)
<i>st_WAIT_PKG_LN</i>	<i>st_WAIT_PKG_LN</i>	(! <i>i_BUSY_N</i> and ! <i>i_BUSY_BL</i>) or <i>i_BUSY_N</i>
<i>st_WAIT_PKG_LN</i>	<i>st_LAST_LN</i>	<i>i_HDR_L</i> ="01" and ! <i>i_BUSY_N</i> and <i>i_BUSY_BL</i>

Fonte: Elaborada pela autora

Tabela 24 – Condições para mudança de estado no Roteador 2

Estado de Origem	Estado de Destino	Condição
<i>st_IDLE</i>	<i>st_IDLE</i>	<p>(<i>!i_BUSY_BL</i> and <i>!i_BUSY_BN</i>) or (<i>!i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_N[0]</i>) or (<i>!i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_N="10"</i>) or (<i>i_BUSY_BL</i> and <i>!i_BUSY_BN</i> and <i>!i_HDR_L[0]</i>) or (<i>i_BUSY_BL</i> and <i>!i_BUSY_BN</i> and <i>i_HDR_L="10"</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_L[0]</i> and <i>!i_HDR_N[0]</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>!i_HDR_L[0]</i> and <i>i_HDR_N="10"</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_L="10"</i> and <i>!i_HDR_N[0]</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_L="10"</i> and <i>i_HDR_N="10"</i>)</p>
<i>st_IDLE</i>	<i>st_WRITE_NL</i>	<p><i>i_BUSY_BN</i> and <i>i_HDR_N="11"</i> and <i>i_ADDR_N="01"</i></p>
<i>st_IDLE</i>	<i>st_WRITE_NN</i>	<p>(<i>i_HDR_N="11"</i> and <i>i_BUSY_BN</i> and <i>i_ADDR_N="00"</i>) or (<i>i_HDR_N="11"</i> and <i>i_BUSY_BN</i> and <i>i_ADDR_N[0]</i>)</p>

Continuação na próxima página

Tabela 24 – Continuação da página anterior

Estado de Origem	Estado de Destino	Condição
<i>st_IDLE</i>	<i>st_WRITE_LN</i>	$(i_HDR_L="11" \text{ and } i_BUSY_BL \text{ and } !i_BUSY_BN) \text{ or } (i_HDR_L="11" \text{ and } !i_HDR_N[0] \text{ and } i_BUSY_BL \text{ and } i_BUSY_BN) \text{ or } (i_HDR_L="11" \text{ and } i_HDR_N="10" \text{ and } i_BUSY_BL \text{ and } i_BUSY_BN)$
<i>st_WRITE_NL</i>	<i>st_WRITE_NL</i>	<i>i_BUSY_L</i>
<i>st_WRITE_NL</i>	<i>st_DONE_NL</i>	$!i_BUSY_L$
<i>st_WRITE_NN</i>	<i>st_WRITE_NN</i>	<i>i_BUSY_N</i>
<i>st_WRITE_NN</i>	<i>st_DONE_NN</i>	$!i_BUSY_N$
<i>st_WRITE_LN</i>	<i>st_WRITE_LN</i>	<i>i_BUSY_N</i>
<i>st_WRITE_LN</i>	<i>st_DONE_LN</i>	$!i_BUSY_N$
<i>st_DONE_NL</i>	<i>st_WAIT_PKG_NL</i>	
<i>st_DONE_NN</i>	<i>st_WAIT_PKG_NN</i>	
<i>st_DONE_LN</i>	<i>st_WAIT_PKG_LN</i>	
<i>st_WAIT_PKG_NL</i>	<i>st_WRITE_NL</i>	$(i_HDR_N="00" \text{ and } i_BUSY_BN \text{ and } !i_BUSY_L) \text{ or } (i_HDR_N[0] \text{ and } i_BUSY_BN \text{ and } !i_BUSY_L)$
<i>st_WAIT_PKG_NL</i>	<i>st_WAIT_PKG_NL</i>	$!i_BUSY_BN \text{ or } (i_BUSY_BN \text{ and } i_BUSY_L)$
<i>st_WAIT_PKG_NL</i>	<i>st_LAST_NL</i>	$i_HDR_N="01" \text{ and } i_BUSY_BN \text{ and } !i_BUSY_L$
<i>st_WAIT_PKG_NN</i>	<i>st_WRITE_NN</i>	$(i_HDR_N="00" \text{ and } !i_BUSY_N \text{ and } i_BUSY_BN) \text{ or } (i_HDR_N[0] \text{ and } !i_BUSY_N \text{ and } i_BUSY_BN)$
<i>st_WAIT_PKG_NN</i>	<i>st_WAIT_PKG_NN</i>	$(!i_BUSY_N \text{ and } !i_BUSY_BN) \text{ or } i_BUSY_N$

Continuação na próxima página

Tabela 24 – *Continuação da página anterior*

Estado de Origem	Estado de Destino	Condição
<i>st_WAIT_PKG_NN</i>	<i>st_LAST_NN</i>	<i>i_HDR_N="01" and !i_BUSY_N and i_BUSY_BN</i>
<i>st_WAIT_PKG_LN</i>	<i>st_WRITE_LN</i>	<i>(i_HDR_L="00" and !i_BUSY_N and i_BUSY_BL) or (i_HDR_L[0] and !i_BUSY_N and i_BUSY_BL)</i>
<i>st_WAIT_PKG_LN</i>	<i>st_WAIT_PKG_LN</i>	<i>(!i_BUSY_N and !i_BUSY_BL) or i_BUSY_N</i>
<i>st_WAIT_PKG_LN</i>	<i>st_LAST_LN</i>	<i>i_HDR_L="01" and !i_BUSY_N and i_BUSY_BL</i>

Fonte: Elaborada pela autora

Tabela 25 – Condições para mudança de estado no Roteador 3

Estado de Origem	Estado de Destino	Condição
<i>st_IDLE</i>	<i>st_IDLE</i>	<p>(<i>i_BUSY_BL</i> and !<i>i_BUSY_BN</i>) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and !<i>i_HDR_N</i>[0]) or (!<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_N</i>="10") or (<i>i_BUSY_BL</i> and !<i>i_BUSY_BN</i> and !<i>i_HDR_L</i>[0]) or (<i>i_BUSY_BL</i> and !<i>i_BUSY_BN</i> and <i>i_HDR_L</i>="10") or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and !<i>i_HDR_L</i>[0] and !<i>i_HDR_N</i>[0]) or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and !<i>i_HDR_L</i>[0] and <i>i_HDR_N</i>="10") or (<i>i_BUSY_BL</i> and <i>i_BUSY_BN</i> and <i>i_HDR_L</i>="10" and <i>i_HDR_N</i>="10")</p>
<i>st_IDLE</i>	<i>st_WRITE_NL</i>	<p><i>i_BUSY_BN</i> and <i>i_HDR_N</i>="11" and <i>i_ADDR_N</i>="11"</p>
<i>st_IDLE</i>	<i>st_WRITE_NN</i>	<p>(<i>i_HDR_N</i>="11" and <i>i_BUSY_BN</i> and !<i>i_ADDR_N</i>[0]) or (<i>i_HDR_N</i>="11" and <i>i_BUSY_BN</i> and <i>i_ADDR_N</i>="10")</p>

Continuação na próxima página

Tabela 25 – Continuação da página anterior

Estado de Origem	Estado de Destino	Condição
<i>st_IDLE</i>	<i>st_WRITE_LN</i>	(<i>i_HDR_L</i> ="11" and <i>i_BUSY_BL</i> and ! <i>i_BUSY_BN</i>) or (<i>i_HDR_L</i> ="11" and ! <i>i_HDR_N</i> [0] and <i>i_BUSY_BL</i> and <i>i_BUSY_BN</i>) or (<i>i_HDR_L</i> ="11" and <i>i_HDR_N</i> ="10" and <i>i_BUSY_BL</i> and <i>i_BUSY_BN</i>)
<i>st_WRITE_NL</i>	<i>st_WRITE_NL</i>	<i>i_BUSY_L</i>
<i>st_WRITE_NL</i>	<i>st_DONE_NL</i>	! <i>i_BUSY_L</i>
<i>st_WRITE_NN</i>	<i>st_WRITE_NN</i>	<i>i_BUSY_N</i>
<i>st_WRITE_NN</i>	<i>st_DONE_NN</i>	! <i>i_BUSY_N</i>
<i>st_WRITE_LN</i>	<i>st_WRITE_LN</i>	<i>i_BUSY_N</i>
<i>st_WRITE_LN</i>	<i>st_DONE_LN</i>	! <i>i_BUSY_N</i>
<i>st_DONE_NL</i>	<i>st_WAIT_PKG_NL</i>	
<i>st_DONE_NN</i>	<i>st_WAIT_PKG_NN</i>	
<i>st_DONE_LN</i>	<i>st_WAIT_PKG_LN</i>	
<i>st_WAIT_PKG_NL</i>	<i>st_WRITE_NL</i>	(<i>i_HDR_N</i> ="00" and <i>i_BUSY_BN</i> and ! <i>i_BUSY_L</i>) or (<i>i_HDR_N</i> [0] and <i>i_BUSY_BN</i> and ! <i>i_BUSY_L</i>)
<i>st_WAIT_PKG_NL</i>	<i>st_WAIT_PKG_NL</i>	! <i>i_BUSY_BN</i> or (<i>i_BUSY_BN</i> and <i>i_BUSY_L</i>)
<i>st_WAIT_PKG_NL</i>	<i>st_LAST_NL</i>	<i>i_HDR_N</i> ="01" and <i>i_BUSY_BN</i> and ! <i>i_BUSY_L</i>
<i>st_WAIT_PKG_NN</i>	<i>st_WRITE_NN</i>	(<i>i_HDR_N</i> ="00" and ! <i>i_BUSY_N</i> and <i>i_BUSY_BN</i>) or (<i>i_HDR_N</i> [0] and ! <i>i_BUSY_N</i> and <i>i_BUSY_BN</i>)
<i>st_WAIT_PKG_NN</i>	<i>st_WAIT_PKG_NN</i>	(! <i>i_BUSY_N</i> and ! <i>i_BUSY_BN</i>) or <i>i_BUSY_N</i>

Continuação na próxima página

Tabela 25 – Continuação da página anterior

Estado de Origem	Estado de Destino	Condição
<i>st_WAIT_PKG_NN</i>	<i>st_LAST_NN</i>	<i>i_HDR_N</i> ="01" and ! <i>i_BUSY_N</i> and <i>i_BUSY_BN</i>
<i>st_WAIT_PKG_LN</i>	<i>st_WRITE_LN</i>	(<i>i_HDR_L</i> ="00" and ! <i>i_BUSY_N</i> and <i>i_BUSY_BL</i>) or (<i>i_HDR_L</i> [0] and ! <i>i_BUSY_N</i> and <i>i_BUSY_BL</i>)
<i>st_WAIT_PKG_LN</i>	<i>st_WAIT_PKG_LN</i>	(! <i>i_BUSY_N</i> and ! <i>i_BUSY_BL</i>) or <i>i_BUSY_N</i>
<i>st_WAIT_PKG_LN</i>	<i>st_LAST_LN</i>	<i>i_HDR_L</i> ="01" and ! <i>i_BUSY_N</i> and <i>i_BUSY_BL</i>

Fonte: Elaborada pela autora