PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# IMPLEMENTAÇÃO E AVALIAÇÃO DE MÉTODOS PARA CONFIABILIDADE DE REDES INTRA-CHIP

ALZEMIRO HENRIQUE LUCAS DA SILVA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre, Brasil
2010

**Dados Internacionais de Catalogação na Publicação (CIP)**

S586i Silva, Alzemiro Henrique Lucas da
Implementação e avaliação de métodos para confiabilidade de redes intra-chip / Alzemiro Henrique Lucas da Silva. – Porto Alegre, 2010.
89 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Informática. 2. Redes de Computadores. 3. Arquitetura de Redes. 4. Tolerância a Falhas (Computação). I. Moraes, Fernando Gehm. II. Título.

CDD 004.6

**Ficha Catalográfica elaborada pelo**
**Setor de Tratamento da Informação da BC-PUCRS**

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Implementação e Avaliação de Métodos para Confiabilidade de Redes Intra-Chip", apresentada por Alzemiro Henrique Lucas da Silva, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais , aprovada em 27/01/10 pela Comissão Examinadora:

Prof. Dr. Fernando Gehm Moraes – Orientador — PPGCC/PUCRS

Prof. Dr. Eduardo Augusto Bezerra – PPGCC/PUCRS

Prof. Dr. Marcelo Soares Lubaszewski – UFRGS

Dr. Alexandre de Morais Amory – Bolsista PNPD FACIN/PUCRS

Homologada em..27../.04../.10....., conforme Ata No.007..... pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
**Campus Central**
Av. Ipiranga, 6681 - P32- sala 507 - CEP: 90619-900
Fone: (51) 3320-3611 - Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

# AGRADECIMENTOS

Gostaria de agradecer a todos que de uma forma ou de outra contribuíram para realização deste trabalho de pesquisa de dois anos e na consequente escrita desta dissertação.

Agradeço a minha família, especialmente aos meus pais, Marina Leivas Lucas e Azemiro Pereira da Silva, que sempre prezaram pela qualidade da minha educação e me incentivaram a estudar e escolher meus próprios caminhos, e a minha irmã, Gabriela Lucas da Silva, que sempre esteve próxima a mim dando bons (e alguns maus) exemplos.

Agradeço também ao meu orientador, Fernando Moraes, pelo incentivo à pesquisa e pela dedicação à orientação, que mesmo com filho pequeno e como coordenador do programa de pós-graduação tomando boa parte do seu tempo, sempre me ajudou quando eu precisei. Sem ele, essa dissertação com certeza não aconteceria.

Agradeço ao Prof. Eduardo Bezerra, pelo acompanhamento dos trabalhos através das bancas de avaliação e pelos conselhos que ajudaram este trabalho a crescer, ao Dr. Alexandre Amory pela contribuição em uma de minhas publicações e ao Leandro Möller que serve de exemplo como pesquisador e também me levou a dar os primeiros passos na pesquisa de NoCs.

Manifesto meu agradecimento ao bolsista Thiago Gouvea, que me ajudou muito na realização deste trabalho, automatizando a geração das redes, facilitando o estudo e análise das arquiteturas desenvolvidas.

Agradeço pela oportunidade de fazer parte dos projetos TETHA e X10GIGA, onde pude ter um contato com pesquisa e ao mesmo tempo com o lado empresarial do desenvolvimento de hardware antes de começar o mestrado, sendo que muitas das idéias desta dissertação partiram de atividades realizadas nestes projetos. Agradeço também a todos os colegas que fizeram parte destes projetos enquanto eu estava por lá.

Agradeço aos meus colegas de graduação que seguiram o mesmo rumo que eu no mestrado e juntos sempre nos divertimos e motivamos um ao outro para a realização das atividades. São eles: Luciano Azevedo, Samuel Marczak e Taciano Rodolfo. Sem esquecer nosso colega Carlos Reif, que embora tenha escolhido outro caminho após iniciar o mestrado, continuou motivando os que ficaram.

Por fim, agradeço aos demais colegas de grupo e de pós-graduação, pela parceria e pelo exemplo: Guindani, Ost, Edson, Rafael, Leonel, Carara, Alexandra e Shonarth.

# IMPLEMENTAÇÃO E AVALIAÇÃO DE MÉTODOS PARA CONFIABILIDADE DE REDES INTRA-CHIP

## RESUMO

As inovações na fabricação de circuitos integrados têm reduzido continuamente o tamanho dos componentes, permitindo um aumento na densidade lógica de sistemas eletrônicos complexos, denominados SoCs (*Systems-on-a-Chip*), mas afetando também a confiabilidade destes componentes. Barramentos globais utilizados para interconexão de componentes em um chip estão cada vez mais sujeitos aos efeitos de *crosstalk*, que podem causar atrasos e picos nos sinais. Este trabalho apresenta e avalia diferentes técnicas para tolerância a falhas em redes intra-chip, nos quais a rede é capaz de manter o mesmo desempenho da rede original mesmo na ocorrência de falhas. Quatro técnicas são apresentadas e avaliadas em termos de consumo adicional de área, latência dos pacotes, consumo de potência e análise de defeitos residuais. Os resultados demonstram que o uso de codificação CRC nos enlaces é vantajoso quando o mínimo acréscimo de área e consumo de potência é o principal objetivo. Entretanto, cada um dos métodos apresentados neste trabalho tem as suas próprias vantagens e podem ser utilizados dependendo da aplicação alvo.

**Palavras Chave:**tolerância a falhas; confiabilidade; redes intra-chip (NoCs); detecção e correção de erros.

# IMPLEMENTATION AND EVALUATION OF RELIABILITY METHODS FOR INTRA-CHIP NETWORKS


## ABSTRACT

The innovations on integrated circuit fabrics are continuously reducing components size, which increases the logic density of systems-on-chip (SoC), but also affect the reliability of these components. Chip-level global buses are especially subject to crosstalk faults, which can lead to increased delay and glitches. This work evaluates different fault tolerant approaches for Networks-on-chip (NoCs) such that the network can maintain the original network performance even in the presence of faults. Four different approaches are presented and evaluated in terms of area overhead, packet latency, power consumption, and residual fault coverage. Results demonstrate that the use of CRC coding at each link is preferred when minimal area and power overhead are the main goals. However, each one of the methods presented here has its own advantages and can be applied depending on the target application.



**Keywords:**fault tolerance; reliability; Networks-on-Chip (NoCs); error correction and detection.

# LISTA DE FIGURAS


FIGURA 1. ARQUITETURA DE NOC COM TOPOLOGIA MALHA. ..... 19
FIGURA 2. TRANSIÇÕES NECESSÁRIAS NO MAF MODEL. ..... 33
FIGURA 3. DIAGRAMA DE INTERFACES DO MÓDULO SABOTEUR. ..... 34
FIGURA 4. EXEMPLO DE OPERAÇÃO DO MÓDULO *SABOTEUR*. ..... 35
FIGURA 5. ARQUITETURA DO ROTEADOR DA REDE XPIPES, INCLUINDO LÓGICA DE CONTROLE DE ERROS [BER04]. ..... 39
FIGURA 6. ESQUEMAS DE RECUPERAÇÃO DE ERROS FIM-A-FIM (A) E ROTEADOR-ROTEADOR (B) [MUR05]. ..... 41
FIGURA 7. LATÊNCIAS OBSERVADAS NOS DIFERENTES ESQUEMAS DE PROTEÇÃO DE ERROS. [MUR05] ..... 43
FIGURA 8. CONSUMO DE ENERGIA DOS DIFERENTES ESQUEMAS DE PROTEÇÃO DE ERROS. [MUR05] ..... 43
FIGURA 9. RELAÇÃO ENTRE O CONSUMO DE POTÊNCIA DOS DIFERENTES ESQUEMAS DE PROTEÇÃO DE ERROS COM A TAXA DE ERROS RESIDUAL DESEJADA. ..... 44
FIGURA 10. IMPLEMENTAÇÃO DO PROTOCOLO STALL/GO [PUL05]. ..... 46
FIGURA 11. IMPLEMENTAÇÃO DO PROTOCOLO T-ERROR [PUL05]. ..... 47
FIGURA 12. IMPLEMENTAÇÃO DO PROTOCOLO ACK/NACK [PUL05]. ..... 48
FIGURA 13. EXEMPLO DE EXECUÇÃO DO ALGORITMO *N RANDOM WALK* COM N=1 [ZHU07]. ..... 50
FIGURA 14. ILUSTRAÇÃO DO ALGORITMO *NEGATIVE FIRST* NA PRESENÇA DE ROTEADORES COM FALHAS PERMANENTES [ZHU07]. ..... 51
FIGURA 15. RELAÇÃO ENTRE A TAXA DE INJEÇÃO E A VAZÃO OBSERVADA NOS DIFERENTES ALGORITMOS DE ROTEAMENTO [ZHU07]. ..... 51
FIGURA 16. ENTREGAS DE MENSAGENS COM SUCESSO NOS DIFERENTES ALGORITMOS DE ROTEAMENTO [ZHU07]. ..... 52
FIGURA 17. CONSUMO DE ENERGIA DOS DIFERENTES ALGORITMO DE ROTEAMENTO. ..... 53
FIGURA 18. ARQUITETURA DO ROTEADOR COM VERIFICAÇÃO DE CRC [KOH09]. ..... 54
FIGURA 19. LATÊNCIA OBSERVADA COM DIFERENTES MÉTODOS DE RECUPERAÇÃO DE ERROS [GRE07]. ..... 55
FIGURA 20. LATÊNCIA PARA RECUPERAÇÃO DE ERROS [GRE07]. ..... 56
FIGURA 21. LATÊNCIA PARA RECUPERAÇÃO DE ERROS COM PROBABILIDADE DE ENTREGA COM SUCESSO VARIÁVEL [GRE07]. ..... 56
FIGURA 22. CODIFICADOR CRC PARA O POLINÔMIO G = 1+X+X4. ..... 61
FIGURA 23. CODIFICADOR CRC PARALELO. ..... 61
FIGURA 24. INTERFACE ENTRE UMA PORTA DE SAÍDA E UMA PORTA DE ENTRADA DOS ROTEADORES, COM ADIÇÃO DOS SINAIS *CRC_IN-CRC_OUT* E *ERROR_IN-ERROR_OUT*. ..... 65
FIGURA 25. DIAGRAMA DO CONSUMO DE DADOS PELO BUFFER DESTINO. ..... 66
FIGURA 26. ILUSTRAÇÃO DA RECUPERAÇÃO ATRAVÉS DA BUSCA NO BUFFER DE ORIGEM. ..... 67
FIGURA 27. REPRESENTAÇÃO DE ENLACES DA REDE COM CRC NA ORIGEM. ..... 68
FIGURA 28. INTERFACE ENTRE DOIS ROTEADORES DE UMA REDE COM HAMMING NOS ENLACES. ..... 69
FIGURA 29. DIAGRAMA SIMPLIFICADO DE UMA REDE COM *HAMMING* NA ORIGEM. ..... 69
FIGURA 30. INTERFACE DA PLATAFORMA ATLAS PARA A GERAÇÃO DAS REDES COM TOLERÂNCIA A FALHAS. ..... 71
FIGURA 31. REENVIO DE UM DADO CORROMPIDO UTILIZANDO-SE CODIFICAÇÃO CRC. ..... 72
FIGURA 32. PROCESSO DE BUSCA DO DADO CORROMPIDO NO BUFFER DE ORIGEM. ..... 73
FIGURA 33. CORREÇÃO DE UM DADO CORROMPIDO ATRAVÉS DE CÓDIGO DE *HAMMING*. ..... 73
FIGURA 34. CORREÇÃO DE UMA FALHA NO BUFFER DE ENTRADA COM *HAMMING* NA ORIGEM. ..... 74
FIGURA 35. DISTRIBUIÇÃO DE LATÊNCIAS NO PRIMEIRO CENÁRIO DE TESTE. ..... 79
FIGURA 36. DISTRIBUIÇÃO DE LATÊNCIAS NO SEGUNDO CENÁRIO DE TESTE. ..... 80
FIGURA 37. DISTRIBUIÇÃO DE LATÊNCIAS NO TERCEIRO CENÁRIO DE TESTE. ..... 81
FIGURA 38. DISTRIBUIÇÃO DE LATÊNCIAS NO QUARTO CENÁRIO DE TESTE. ..... 82

# LISTA DE TABELAS


TABELA 1. TIPOS DE FALHAS E MODELOS DE FALHAS CONSIDERADOS EM REDES DE COMUNICAÇÃO.....27
TABELA 2. RELAÇÃO ENTRE O SENTIDO DE TRANSIÇÃO E DO FATOR DE ATRASO G NA VÍTIMA K. OS SÍMBOLOS ↑, ↓ E – REPRESENTAM TRANSIÇÃO POSITIVA, TRANSIÇÃO NEGATIVA E SEM TRANSIÇÃO, RESPECTIVAMENTE. .....32
TABELA 3. COMPARAÇÃO ENTRE OS PROTOCOLOS DE CONTROLE DE FLUXO. (*N* REFERE-SE AO NÚMERO DE ESTÁGIOS ENTRE REGISTRADORES E *M* CORRESPONDE À *N*/2) .....46
TABELA 4. CONSUMO DE ÁREA DOS ALGORITMOS DE ROTEAMENTO. .....53
TABELA 5. TABELA DE RESUMO DOS TRABALHOS RELACIONADOS A TOLERÂNCIA A FALHAS EM NOCS. .....57
TABELA 6. CONSUMO DE ÁREA DA REDE HERMES 8X8 SEM ALTERAÇÕES EM FPGA. .....75
TABELA 7. CONSUMO DE ÁREA E FREQUÊNCIA DE OPERAÇÃO DASREDES 8X8 COM CODIFICAÇÃO CRC EM FPGA. .....75
TABELA 8. CONSUMO DE ÁREA E FREQÜÊNCIA DE OPERAÇÃO DAS REDES 8X8 COM CÓDIGOS DE *HAMMIG* EM FPGA. 76
TABELA 9. CONSUMO DE ÁREA DA REDE ORIGINAL E DAS REDES COM CRC DE TAMANHO 8X8 MAPEADAS PARA *STANDARD CELLS*. .....76
TABELA 10. CONSUMO DE ÁREA DA REDE ORIGINAL E DAS REDES COM CÓDIGOS DE *HAMMING* DE TAMANHO 8X8 MAPEADAS PARA *STANDARD CELLS*. .....77
TABELA 11. LATÊNCIAS OBSERVADAS NO PRIMEIRO CENÁRIO DE TESTE (32 *FLITS*). .....78
TABELA 12. LATÊNCIAS OBSERVADAS NO SEGUNDO CENÁRIO DE TESTE (48 *FLITS*). .....79
TABELA 13. LATÊNCIAS OBSERVADAS NO TERCEIRO CENÁRIO DE TESTE. .....81
TABELA 14. LATÊNCIAS OBSERVADAS NO QUARTO CENÁRIO DE TESTE. .....82
TABELA 15. ANÁLISE DE DEFEITOS RESIDUAIS. .....83
TABELA 16. AUMENTO NO CONSUMO DE POTÊNCIA DAS ARQUITETURAS COM CRC. .....84
TABELA 17. AUMENTO NO CONSUMO DE POTÊNCIA DAS ARQUITETURAS COM CÓDIGO DE *HAMMING*. .....84

# LISTA DE SIGLAS

ASIC *Application Specific Integrated Circuit*

BE *Best Effort*

BER *Bit Error Rate*

CAD *Computer Aided Design*

CRC *Cyclic Redundancy Check*

DVD *Digital Video Disk*

FEC *Forward Error Correction*

FF *Flip-Flop*

FIT *Failiure unIT*

FPGA *Field Programmable Gate Array*

GALS *Global Asynchronous Local Synchronous*

GT *Guaranteed Throughput*

IP *Intellectual Property*

LUT *Look Up Table*

MAF *Maximum Aggressor Fault*

MP3 MPEG *Layer* 3

MPEG *Moving Picture Expertise Group*

MTBF *Mean Time Between Failure*

NI *Network Interface*

NoC *Network-on-Chip*

QoS *Quality of Service*

RTL *Register Transfer Level*

SER *Soft Error Rate*

SEU *Single Event Upset*

SoC *System-on-Chip*

TMR *Triple Modular Redundancy*

VCD *Value Change Dump*

VHDL VHSIC *Hardware Description Language*

VHSIC *Very High Speed Integrated Circuits*

# SUMÁRIO


**1 INTRODUÇÃO ..... 15**
1.1 Motivações do Trabalho ..... 17
1.2 Objetivos do Trabalho ..... 17
1.3 Organização do Documento ..... 17

**2 RELAÇÃO ENTRE CARACTERÍSTICAS DE NOCS E TOLERÂNCIA A FALHAS ..... 19**
2.1 Comunicação em Camadas ..... 20
2.2 Algoritmos de Roteamento ..... 21
2.3 Métodos de Chaveamento ..... 21
2.4 Qualidade de Serviço ..... 22
2.5 Considerações Finais ..... 23

**3 FUNDAMENTOS DE CROSSTALK E MODELOS DE FALHAS ..... 25**
3.1 Falha, Erro e Defeito ..... 27
3.2 Proposta de Modelo de Falhas para NoCs ..... 28
3.3 Modelagem de Crosstalk ..... 30
3.3.1 Definição de Erros de *Crosstalk* ..... 31
3.3.2 Impacto do Crosstalk no Desempenho ..... 31
3.3.3 Modelo MAF (Maximal Aggressor Fault) ..... 32
3.4 Módulo Saboteur ..... 34

**4 ESTADO DA ARTE ..... 36**
4.1 Recuperação de Erros por Comunicação Estocástica ..... 36
4.2 Recuperação de Erros por Correção e por Detecção+Reenvio ..... 38
4.2.1 Recuperação Fim-a-Fim versus Roteador-a-Roteador ..... 39
4.2.2 Controle de Fluxo para Controle de Erro ..... 45
4.3 Algoritmos de Roteamento Adaptativos Visando Tolerância a Falhas ..... 48
4.4 Métricas Utilizadas Para Avaliação ..... 55
4.5 Resumo dos Trabalhos Relacionados ..... 57

**5 ARQUITETURAS DE NOCS CONFIÁVEIS ..... 59**
5.1 HERMES ..... 59
5.2 Codificação ..... 60
5.2.1 Circuito CRC ..... 60
5.2.2 Hamming ..... 62
5.3 NoC com CRC nos Enlaces ..... 64
5.4 NoC com CRC na Origem ..... 67
5.5 NoC com Hamming nos Enlaces ..... 68
5.6 NoC com Hamming na Origem ..... 69
5.7 Plataforma ATLAS ..... 70

**6 RESULTADOS ..... 72**
6.1 Processo de Recuperação de Erros ..... 72
6.2 Ocupação de Área e Frequência ..... 74
6.3 Impacto de Latência ..... 77

6.4 Análise de Defeitos Residuais ..... 83
6.5 Consumo de Potência ..... 83
**7 CONCLUSÕES E TRABALHOS FUTUROS ..... 85**
7.1 Trabalhos Futuros ..... 86
**REFERÊNCIAS BIBLIOGRÁFICAS ..... 88**

# 1 INTRODUÇÃO

A evolução contínua na tecnologia de produção de circuitos integrados permite que os dispositivos agreguem um número cada vez maior de transistores, pois seu tamanho está sendo continuamente reduzido. Isto leva à criação de dispositivos cada vez mais complexos, que agregam de dezenas a centenas de módulos se comunicando internamente em um único chip, conduzindo ao conceito de System-on-a-Chip (SoC).

O paradigma de comunicação entre os diversos módulos de um SoC vem mudando nos últimos anos. A comunicação através de barramento não satisfaz mais os requisitos de desempenho e escalabilidade para os dispositivos compostos por diversos núcleos. O conceito de redes Intra-Chip (*Networks on Chip - NoC*) [BEN02] vem sendo cada vez mais utilizado para suprir a demanda de alto desempenho dos SoCs atuais. Uma NoC pode ser vista como um conjunto de roteadores capazes de transportar dados em forma de pacotes, possibilitando um certo grau de paralelismo e aumentando o desempenho da comunicação entre os núcleos de um SoC.

A redução no tamanho dos transistores leva à utilização de uma tensão de operação reduzida e freqüências de relógio mais elevadas. Tecnologias atuais, como 65nm e 45nm, utilizam voltagens de operação da ordem de 1 Volt e freqüências na ordem de gigahertz. Apesar de aumentar a capacidade de integração de um chip, consumindo uma quantidade de energia aceitável em dispositivos móveis, as tecnologias submicrônicas são mais suscetíveis a ruídos que podem introduzir diversos tipos de falhas em um circuito. Ruídos na fonte de alimentação são muito mais críticos em circuitos que operam em baixas tensões, pois a tolerância entre os valores lógicos nestes circuitos é muito menor. Além disso, devido às altas freqüências de operação, problemas de atrasos e picos devido ao chaveamento em sentidos opostos de fios adjacentes, conhecidos como *crosstalk*, podem acontecer mais freqüentemente, prejudicando a confiabilidade na transmissão de informações dentro do chip. Outro problema observado nas tecnologias submicrônicas é chamado de SEU (*Single Event Upset*), onde o valor lógico contido em um elemento de memória é alterado através de radiações. Este tipo de interferência, que no passado prejudicava a confiabilidade em sistemas espaciais, hoje pode ser observado em dispositivos operando no nível do mar devido à redução no tamanho dos componentes. O crescimento do uso de circuitos assíncronos, motivado pela redução de potência na distribuição da árvore de clock em SoCs com muitos núcleos, introduz erros de sincronismo na comunicação entre módulos operando em freqüências diferentes, sendo outro fator que pode limitar a confiabilidade dos projetos de circuitos eletrônicos atuais.

Conforme descrito em [BEN02] uma NoC pode ser organizada através de uma pilha de protocolos explorando o conceito geral de redes de comunicação. A garantia de comunicação confiável não pode ser transferida para a camada física devido a todos os problemas de confiabilidade citados acima. Para garantir o comportamento ideal nos fios que interconectam os roteadores de uma NoC, alterações teriam de ser feitas nas regras de layout do circuito. Além de exigir conhecimentos técnicos mais apurados, essa técnica não possui o melhor custo de desempenho em relação ao aumento de área e consumo de potência. Dessa maneira, a tarefa de garantir transferência de dados confiável através de ligações físicas não confiáveis deve ser realizada pela camada de enlace [BER06].

Prover confiabilidade na comunicação intra-chip deixou de ser uma preocupação apenas para os engenheiros de teste. Também é necessário prover confiabilidade adicionando mecanismos de tolerância a falhas em tempo de projeto para garantir um nível mínimo de falhas aceitável para cada aplicação em tempo de execução. Dessa maneira, fica a cargo do projetista de hardware prover confiabilidade em conexões com probabilidade não nula de falhas. No contexto de NoCs, muitos trabalhos estão sendo realizados para garantir comunicação segura entre os diversos dispositivos de um SoC. As técnicas mais utilizadas são baseadas em códigos de controle de erros, que adicionam informações redundantes nos pacotes transmitidos e permitem detectar a presença de erros ou corrigi-los utilizando técnicas de FEC (*Forward Error Correction*). Existem também técnicas baseadas em algoritmos de roteamento adaptativos para o caso de falhas permanentes em um circuito. Embora já existam alguns trabalhos publicados nesta área, este é um assunto ainda pouco explorado e aberto para experimentações.

A preocupação com o consumo de potência é evidente em todos os métodos propostos para controle de erros em redes intra-chip. A codificação ideal deve adicionar o máximo de confiabilidade ao mesmo tempo em que adiciona o mínimo de potência consumida. Códigos detectores de erros requerem reenvio no caso de detecção, aumentando o consumo de acordo com o número de retransmissões. Além disso, devem se adicionar novos buffers para armazenar os pacotes caso estes tenham de ser retransmitidos, sendo que as operações de escrita e leitura em buffer são o maior gargalo no consumo de potência em NoCs. Métodos capazes de corrigir erros podem evitar o reenvio de pacotes, porém os codificadores e os decodificadores para correção de erros são mais complexos e consomem mais potência. Neste caso a escolha do método mais adequado deve levar em conta não apenas o requisito de potência, mas também a confiabilidade requerida pela aplicação, uma vez que podem acontecer erros de correção caso ocorram mais erros

que o código é capaz de corrigir. Neste caso, em aplicações críticas em confiabilidade, podem ser utilizados mecanismos de correção simples juntamente com detecção múltipla, garantindo a confiabilidade e evitando alguns processos de retransmissão, porém adicionando maior custo de área ao circuito. O trabalho proposto em [ZIM03] apresenta uma implementação onde a mesma rede pode ter diversos esquemas de recuperação de erros, sendo que a escolha do método pode ser feita em tempo de execução baseado na confiabilidade exigida pelos diferentes fluxos de dados que circulam na rede.

## 1.1 Motivações do Trabalho

Sistemas embarcados são sistemas eletrônicos utilizados cotidianamente em telefones celulares, setup-box, DVDs, tocadores de MP3, eletrônica automotiva, entre outros. A pesquisa e o desenvolvimento de técnicas de projeto para sistemas embarcados é estratégica para as indústrias que tenham em sua cadeia produtiva o desenvolvimento de sistemas computacionais, como telecomunicações, automação de processos e linha automotiva.

A motivação do presente projeto reside na importância das arquiteturas de SoCs para os atuais sistemas embarcados e na necessidade de desenvolver meios de comunicação confiáveis, eficientes e de baixo custo entre seus componentes. Aplicações típicas de SoCs possuem requisitos rígidos em termos de desempenho, consumo de energia e gasto em área. Estes requisitos são facilmente compreendidos, visto que os exemplos mencionados acima requerem alta duração de bateria (baixo consumo), transmissão de áudio e vídeo (desempenho) e tamanho reduzido (área). A utilização de tolerância a falhas nestas arquiteturas é uma área forte de pesquisa, pois é vista como uma solução aos problemas de confiabilidade introduzidos pelas tecnologias submicrônicas mencionados anteriormente.

## 1.2 Objetivos do Trabalho

O presente trabalho tem como enfoque o desenvolvimento e avaliação de técnicas de detecção e correção de erros em dados que trafegam em NoCs, considerando que esta estrutura de comunicação seja sensível a ruídos. Os cenários experimentais utilizam a NoC desenvolvida na PUCRS-PPGCC (HERMES), permitindo assim validar os métodos propostos em uma infraestrutura de NoC operacional.

## 1.3 Organização do Documento

O restante deste documento é organizado como segue. O capítulo 2 aborda conceitos de NoCs, necessários para a compreensão das técnicas de implementação de tolerância a falhas nas

mesmas. O Capítulo 3 apresenta noções de modelagem de falhas, bem como um método proposto para modelagem de falhas em redes intra-chip e também explora conceitos de *crosstalk* e o modelo MAF para modelagem de *crosstalk*. No Capítulo 4 é feito um estudo do estado da arte em redes intra-chip tolerantes a falhas, através de um levantamento dos trabalhos publicados na área. O Capítulo 5 apresenta as arquiteturas desenvolvidas no contexto deste trabalho, correspondendo à principal *contribuição* desta Dissertação. Por fim, o Capítulo 6 apresenta os resultados e avaliações das arquiteturas desenvolvidas, levando em conta parâmetros como consumo de área, potência, latência adicional e taxa residual de defeitos. O capítulo 7 encerra este trabalho com as conclusões e direções para trabalhos futuros.

# 2 RELAÇÃO ENTRE CARACTERÍSTICAS DE NOCS E TOLERÂNCIA A FALHAS

Este Capítulo tem por objetivo estudar as arquiteturas e os algoritmos mais utilizados no projeto de redes intra-chip e fazer uma relação com os métodos de tolerância a falhas existentes, indicando onde estes métodos podem ser integrados para proteger a rede de possíveis falhas que possam ocorrer na comunicação de dados.

No geral, uma NoC é uma infra-estrutura de comunicação proposta para substituir barramentos em SoCs que possuam muitos módulos de hardware se comunicando de maneira estocástica. A Figura 1 representa uma arquitetura de NoC muito comum e largamente utilizada [MOR04].

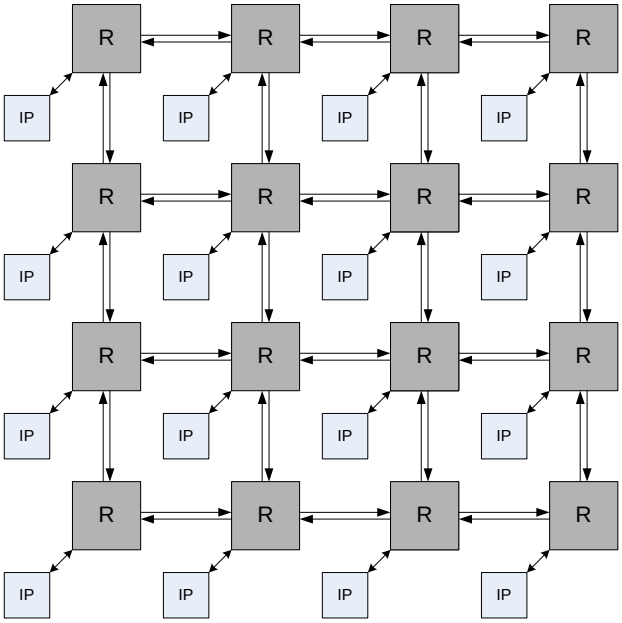


**Figura 1. Arquitetura de NoC com topologia malha.**

As estruturas mais escuras (R) representam os roteadores, enquanto as claras (IP) representam os módulos de hardware conectados à rede. Os módulos de hardware podem se comunicar entre si enviando mensagens para a rede na forma de pacotes. Cada roteador é conectado aos roteadores vizinhos e ao módulo de hardware através de enlaces de entrada e enlaces de saída independentes. Para que um módulo origem se comunique com um módulo destino ele deve enviar um pacote para o roteador ao qual está conectado, que por sua vez repassa para um de seus roteadores vizinhos seguindo um algoritmo de roteamento. Os demais roteadores realizam a mesma operação até que a mensagem chegue ao roteador conectado com o módulo destino, que encaminha a mensagem para este módulo.

A disposição dos roteadores da rede ilustrada na Figura 1 caracteriza a arquitetura malha bidimensional, que apesar de ser uma estrutura simples de interconexão, possui diversas vantagens que faz com que esta arquitetura seja a mais utilizada na prática. Esta disposição de roteadores garante a escalabilidade da rede e o uso de fios mais curtos interligando os roteadores, diminuindo o atraso de propagação dos sinais e possibilitando freqüências de operação mais altas, aumentando a banda disponível na rede. Além disso, essa topologia facilita o roteamento dos pacotes, diminuindo a área dos roteadores, e a disposição plana dos roteadores facilita o posicionamento dos mesmos em um circuito integrado.

## 2.1 Comunicação em Camadas

O artigo [BEN02] propõe o uso de uma pilha de protocolos similar à pilha utilizada em redes de comunicação para o projeto de redes intra-chip. A comunicação em camadas é utilizada para gerenciar a complexidade dos diversos requisitos e funcionalidades necessárias para a implementação de uma rede intra-chip separando claramente os aspectos de implementação de cada uma delas. Os projetistas da rede devem ser capazes de desenvolver as camadas de transporte, de rede e de enlace. A camada física é de responsabilidade dos projetistas de circuitos integrados e normalmente é gerada automaticamente pelas ferramentas de CAD, logo, dificilmente passa por refinamentos para atender a algum requisito específico de desempenho da rede. As camadas superiores ficam por conta dos projetistas dos módulos que serão conectados à rede.

A camada física é relacionada com as características físicas do meio utilizado para conectar os roteadores e os demais recursos de hardware que fazem parte do sistema. No contexto de SoCs, está relacionada com os níveis de tensão utilizados pelo circuito, com o comprimento e largura dos fios, com as características de temporização, entre outras.

A camada de enlace é a camada mais próxima do nível físico, e sua função é garantir a integridade dos dados que trafegam na rede. Os protocolos da camada de enlace devem aumentar a confiabilidade de um canal a um nível aceitável para cada aplicação, assumindo que a camada física não é capaz de prover a confiabilidade desejada. *Nesta camada são inseridos os métodos de detecção e correção de erros, sendo esta o principal foco deste trabalho*.

A camada de transporte é responsável por decompor as mensagens em pacotes na origem e remontar os pacotes da mensagem no destino. Esta tarefa é realizada na interface de rede (NI - *Netowrk interface*). A granularidade do empacotamento é uma decisão crítica no projeto e pode influenciar diretamente na implementação dos métodos de detecção e correção de erros.

A camada de rede é responsável pela implementação da comunicação fim-a-fim entre os módulos de hardware. Esta camada compreende o algoritmo de roteamento e o método de chaveamento utilizado para o tráfego de pacotes entre origem e destino.

## 2.2 Algoritmos de Roteamento

Os algoritmos de roteamento podem ser divididos em algoritmos determinísticos e adaptativos. Os algoritmos determinísticos provêem sempre o mesmo caminho entre uma mesma origem e um mesmo destino. Já em algoritmos adaptativos, um pacote pode fazer diversos caminhos de uma determinada origem até o seu destino, sendo que a escolha do caminho é feita normalmente com a finalidade de evitar os caminhos mais congestionados ou aqueles com maior probabilidade de falhas, no caso de redes tolerantes a falhas. Os algoritmos adaptativos possuem maior complexidade de implementação, porém são capazes de evitar contenção e proteger a rede de caminhos ou roteadores que apresentam falhas permanentes ou onde a probabilidade de falhas é grande. Maiores detalhes sobre os algoritmos de roteamento tolerantes a falhas serão abordados na sessão 4.3.

## 2.3 Métodos de Chaveamento

Os métodos de chaveamento mais utilizados compreendem o chaveamento por circuito e o chaveamento por pacotes. No chaveamento por circuito, uma conexão é estabelecida entre a origem e o destino e o caminho é reservado antes da transmissão dos dados. Este método de chaveamento garante que os dados serão transmitidos na taxa máxima suportada pela rede, porém não é vantajoso para a transferência de uma pequena quantidade de dados devido ao tempo para o estabelecimento do circuito. O chaveamento por circuito também pode causar contenção na rede devido à reserva de recursos. No chaveamento por pacotes, um fluxo é quebrado em diversos pacotes para a transmissão através da rede. Cada pacote é individualmente roteado da origem até o destino, e carrega um cabeçalho que contém informações de roteamento e de controle.

O chaveamento por pacotes requer o uso de uma política de repasse de pacotes, o qual define como os pacotes se movem através dos roteadores. As três políticas mais utilizadas são *store-and-forward*, *virtual-cut-through* e *wormhole*. Os métodos *store-and-forward* e *virtual-cut-through* necessitam de um buffer capaz de armazenar pelo menos um pacote inteiro, enquanto no método *wormhole* os pacotes são divididos em unidades menores, chamadas de *flits*, que são transmitidas um a um aos roteadores intermediários até o destino. Um *flit* pode ser definido como a unidade de transferência entre dois roteadores. O método *wormhole* não é o mais adequado para

recuperação de erros fim-a-fim, pois uma vez que um erro for detectado, o pacote deve ser retransmitido por completo. No método *wormhole* o pacote estará distribuído na rede, com *flits* em diversos roteadores intermediários, de maneira que o descarte de um pacote no caso de um erro se torna muito custoso. Dessa maneira os métodos *store-and-forward* e *virtual-cut-through* podem ser facilmente utilizados para recuperação fim-a-fim, enquanto o método *wormhole* pode ser utilizado para recuperação de dados roteador-a-roteador.

Os dois métodos de chaveamento mencionados podem suportar detecção e correção de erros. Porém o chaveamento por circuito suporta apenas o reenvio de dados fim-a-fim, mesmo que seja possível realizar o controle em todos os roteadores para localizar o problema. Utilizando-se o chaveamento por pacotes é possível realizar também o reenvio roteador-a-roteador, onde cada roteador verifica se o dado recebido possui erros antes de repassá-lo. No controle fim-a-fim dados de paridade normalmente são adicionado no final do pacote, de maneira que o destino deve receber um pacote por completo para verificar a existência de erros ou corrigi-los. Utilizando-se o controle roteador-a-roteador é possível utilizar alguns bits de paridade para cada *flit*, possibilitando a verificação de integridade dos dados sem a necessidade do recebimento completo de um pacote.

## 2.4 Qualidade de Serviço

Qualidade de serviço pode ser aplicada ao conceito de redes intra-chip para garantir que fluxos prioritários ou de tempo real, atendam os requisitos de latência e vazão necessários para determinadas aplicações. O esquema mais utilizado de qualidade de serviço em redes intra-chip separa os fluxos em dois níveis: vazão garantida (GT) e melhor esforço (BE); onde a qualidade de serviço é garantida apenas para os fluxos que utilizem GT, sendo os fluxos de melhor esforço enviados à medida que os recursos estiverem disponíveis. Este esquema de qualidade de serviço não está diretamente relacionado com o conceito de tolerância a falhas em NoCs, porém em [VEL04] é possível encontrar a implementação de uma rede com suporte a técnicas de controle de erro e a qualidade de serviço em GT e BE.

Outro trabalho que contempla QoS e controle de erro pode ser encontrado em [ZIM03]. Neste trabalho a rede proposta possui uma ligação mais direta dos esquemas de controle de erro com os níveis de qualidade de serviço suportados pela rede. O autor define quatro tipos de fluxos de dados que a rede deve suportar: máxima vazão, integridade garantida, latência mínima e alta confiabilidade.

O esquema de máxima vazão contempla a transferência de dados que possam sofrer pequenas alterações em seu conteúdo, mas que necessitam de taxas de transferência mais altas,

como um fluxo de vídeo por exemplo. Neste caso nenhum esquema de correção é aplicado e a carga útil dos pacotes é maior, aceitando-se que um pequeno número de *flits* sejam entregues com seu valor alterado.

No esquema de integridade garantida utilizam-se apenas métodos de detecção de erros. Correção de erros não é utilizada em fluxos que necessitam de integridade garantida porque, caso ocorram mais erros que o método utilizado pode corrigir, podem ocorrer erros de correção. No caso de uma detecção de erro, o pacote pode ser descartado ou pode ser solicitado reenvio através da rede.

Quando um fluxo necessita de latência mínima os pacotes são sempre repassados em direção ao seu destino (algoritmo de roteamento mínimo) e métodos de correção de erros são utilizados para evitar retransmissões. Este método deve ser utilizado para fluxos que podem tolerar falhas raras, que podem acontecer por erros de correção, mas que possuem fortes restrições de temporização para funcionar, como aplicações de tempo real.

Fluxos que necessitam de alta confiabilidade devem ser protegidos por métodos de correção e detecção de erros ao mesmo tempo, de maneira que seja possível detectar erros de correção. Dessa maneira é possível evitar vários processos de retransmissão e garantir que nenhum dado recebido esteja corrompido. Este método de transmissão garante a confiabilidade dos dados, porém diminui a latência e a taxa de transferência do fluxo de dados, sendo o método mais adequado para transferir um código objeto em uma memória de programa, por exemplo.

## 2.5 Considerações Finais

Este Capítulo apresentou conceitos gerais de redes intra-chip e os relacionou com os métodos de tolerância a falhas para proteger a rede dos efeitos negativos ocasionados pela redução contínua no tamanho dos transistores. Como foi apresentado, a garantia de transmissão confiável dos dados não é mais garantida pelo nível físico, de forma que o projeto da camada de enlace de uma NoC deva prover mecanismos de detecção e correção de erros para aumentar a confiabilidade na transmissão de dados. Foi visto também que para aumentar ainda mais a confiabilidade na implementação de uma rede intra-chip, também é possível utilizar algoritmos de roteamento adaptativos, adicionando mecanismos de tratamento de falhas na camada de rede.

Em relação aos métodos de chaveamento, foi visto que para suportar controle de erro no nível de pacote é necessário utilizar políticas como *store-and-forward* ou *virtual-cut-trough*, que adicionam mais latência no envio dos pacotes, porém utilizando-se controle de erro no nível de *flit*

é possível utilizar *wormhole* e poupar recursos de armazenamento provendo menor latência. Por fim, este capítulo estudou técnicas de QoS empregadas no contexto de NoCs com tolerância a falhas, onde os fluxos são separados de acordo com a vazão e confiabilidade exigida.

# 3 FUNDAMENTOS DE CROSSTALK E MODELOS DE FALHAS

Uma preocupação constante quando se fala em confiabilidade é o modelo de falhas utilizado para simular o comportamento de um sistema exposto aos diversos tipos de interferências e defeitos que podem ocorrer durante seu funcionamento. Técnicas de tolerância a falhas em NoCs devem lidar com três questões: como a comunicação através dos fios vai falhar, quão freqüente isso vai acontecer e como recuperar a informação corrompida. O modelo de falhas busca lidar com a primeira questão, descrevendo o comportamento da falha, abstraindo os detalhes físicos do acontecimento da mesma.

Falhas podem ser classificadas como permanentes ou transientes. Falhas permanentes são aquelas que estão presentes constantemente no sistema. Falhas transientes ocorrem randomicamente durante o funcionamento do sistema e são usualmente denominadas falhas transientes. Falhas transientes representam 80% do total de falhas que ocorrem em um sistema [MAH04] e ainda podem ser classificadas em: falhas de temporização e erros leves.

Erros leves são também chamados de SEU e são associados a dados armazenados em memória, enquanto as falhas de temporização estão associadas com a amostragem incorreta de um sinal quando seu atraso de propagação se torna maior que o período de amostragem. Um SEU pode ser ocasionado por partículas energéticas nucleares ou por outros dispositivos elétricos emissores de energia eletromagnética. No caso de partículas energéticas nucleares, a perturbação pode vir a partir de raios cósmicos espaciais, que bombardeiam a terra constantemente, ou de átomos radioativos, que existem em vestígios em todos os materiais devido ao decaimento atômico [MAH04]. Um SEU é uma conseqüência de um SET (*Single Event Transient*). Quando um SET ocorre em uma célula de memória e inverte o valor lógico contido nela, é transformado em um SEU. Já quando um SET ocorre em um fio, ele se transformará em um SEU se o seu valor lógico for armazenado invertido no registrador subjacente. No passado, a probabilidade de ocorrência de um SEU associado à lógica era muito menor que em memória [BER06]. À medida que as freqüências de relógio aumentam e a voltagem de operação diminui, aumenta a probabilidade que um distúrbio de natureza iônica presente em um determinando fio seja captado por um registrador causando erros na saída do circuito, de maneira que a probabilidade de falhas lógicas acaba se tornando similar à probabilidade de um SEU em memória.

Falhas de temporização são normalmente ocasionadas por problemas na integridade do sinal devido ao efeito de *crosstalk* e *ground bounce*. *Crosstalk* é o nome que se dá ao efeito de atraso gerado pelo chaveamento de fios adjacentes em sentidos opostos, enquanto que *groud bounce*

está associado com a perda de integridade do sinal de terra em transistores mais afastados da fonte de alimentação no *layout* de um circuito integrado. Falhas da temporização também podem ser conseqüência de problemas de integridade de sinal devido a variações no processo de fabricação. Variações no processo de fabricação podem aumentar o atraso de sinais em um caminho crítico e também ocasionar falhas transientes. Testar um circuito para verificar a existência de falhas de temporização após sua fabricação é inevitável, porém, testar um dispositivo complexo de maneira a excitar todos os seus caminhos críticos, é um processo muito complexo computacionalmente. Dessa maneira, um circuito com falhas de temporização pode ir para o mercado mesmo após testes exaustivos de pós-fabricação.

A Tabela 1 [BER06] apresenta os tipos de falha, o modelo de falha associado e a probabilidade típica de ocorrência destas falhas para redes de comunicação em geral. Algumas falhas, como o ruído Gaussiano e o choque de partículas alfa, causam falhas transientes que resultam em um ou mais bits em erro, mas não impedem o funcionamento do sistema. Outras falhas, como a eletromigração de um condutor, podem causar uma falha permanente em algum módulo. As chamadas falhas de colagem (*stuck-at*) acontecem quando o valor lógico de um nodo fica constantemente com seu valor lógico inalterado. Na presença de falhas de travamento, alguns componentes param de funcionar e podem avisar os nodos adjacentes. Existem também as falhas bizantinas, onde ao invés de parar de responder, o dispositivo continua respondendo propositalmente de maneira errônea na tentativa de causar falhas nos dispositivos adjacentes. Falhas de colagem e falhas de travamento são exemplos de falhas permanentes, e são normalmente medidas através do tempo médio entre defeitos (*Mean-time between faliure* - MTBF). Um tempo aceitável para a maioria das aplicações e comumente observado na prática chega à ordem de $10^9$ h, que representa um FIT na Tabela 1. BER (*Bit Error Rate*) indica a taxa de erro medida em erros por segundo, enquanto SER (*Soft Error Rate*) representa a taxa de erros leves observada. Estas probabilidades apresentadas na Tabela 1 são observadas sob as novas tecnologias submicrônicas em condições normais de operação, tais como temperatura ambiente e nível do mar.

As falhas apresentadas na Tabela 1, que antes eram esperadas apenas em redes fora de um chip, podem muito bem ilustrar o comportamento de uma rede intra-chip implantada sob as novas tecnologias submicrônicas, uma vez que não se pode mais afirmar que a probabilidade de falhas dentro de um chip é nula, pois as interconexões estão cada vez mais longas, o nível de integração

de um chip continua crescendo a as altas freqüências e baixas voltagens de operação aumentam a sensibilidade dos circuitos a interferências eletromagnéticas.

**Tabela 1. Tipos de falhas e modelos de falhas considerados em redes de comunicação.**

| Tipo de falha | Modelo de Falha | Probabilidade | Unidade |
|---|---|---|---|
| Ruído Gaussiano em um canal | Falha transiente | $10^{-20}$ | BER |
| Choque de partículas alfa em memória (chip) | Erro leve | $10^{-9}$ | SER |
| Choque de particulas alfa em lógica (chip) | Falha transiente | $10^{-10}$ | BER |
| Eletromigração de condutor | Falha de colagem | 1 ($10^9$ h) | MTBF (FITs) |
| Alteração na tensão de *threshold* | Falha de colagem | 1 ($10^9$ h) | MTBF (FITs) |
| Conector aberto por corrosão | Falha de colagem | 10 ($10^9$ h) | MTBF (FITs) |
| Falha na fonte de alimentação | Falha de travamento | $10^4$ ($10^9$ h) | MTBF (FITs) |
| Falha de software | Falha de travamento ou Bizantina | $10^4$ ($10^9$ h) | MTBF (FITs) |

Além das falhas apresentadas na Tabela 1, existem outros fatores que podem influenciar no comportamento da comunicação intra-chip. Uma delas é a temperatura de operação. Variações na temperatura podem alterar o tempo de atraso dos fios e portas lógicas de um circuito, prejudicando a previsibilidade nas respostas e causando outros efeitos, como quedas de tensão e aumento no consumo de potência. Outro fator que pode acarretar em falhas é a divisão da árvore de relógio para a construção de sistemas localmente síncronos e globalmente assíncronos (GALS). Esta divisão é motivada com o intuito de diminuir o consumo de potência na transmissão do sinal de relógio em circuitos com muitos núcleos e evitar escorregamento de clock em árvores muito grandes. Erros de sincronização podem ser evitados por diversas técnicas de projeto de circuitos assíncronos de maneira a se tornarem praticamente inexistentes, mas não nulos.

## 3.1 Falha, Erro e Defeito

As definições de falha, erro e defeito aqui descritas podem der encontradas com mais detalhes em [LAP04]. É importante observar que existem diversos trabalhos propondo definições de falhas, erros e defeitos que podem divergir destas aqui apresentadas.

Um **defeito** (*failure*) é um evento que ocorre quando o serviço disponibilizado desvia do serviço correto. Um defeito pode ocorrer tanto porque o serviço entregue desvia do serviço especificado, quanto quando a especificação não descreve adequadamente o serviço e ser

entregue. Um defeito é uma transição de um serviço correto para um serviço incorreto. A transição de um serviço incorreto para um serviço correto é chamado de restauração de serviço.

Um **erro** (*error*) é uma parte do estado total do sistema que pode levar a um defeito de serviço. É importante observar que muitos erros não atingem o estado externo do sistema e conseqüentemente não geram defeitos.

Uma **falha** (*fault*) é a causa hipotética de um erro. Uma vez que uma falha seja ativada ela passa a causar um erro, caso contrário ela permanece dormente. Na maioria dos casos, uma falha causa um erro em um estado interno de um componente que representa uma parte interna do sistema e o estado externo não é imediatamente afetado, neste caso, permitindo o uso de técnicas de tolerância a falhas ou recuperação de erros, sendo o objetivo deste trabalho.

## 3.2 Proposta de Modelo de Falhas para NoCs

Em sistemas expostos aos efeitos de *crosstalk*, falhas em fios adjacentes de um barramento não podem ser consideradas independentemente como no modelo de falhas tradicional citado na Tabela 1. Para criar um cenário mais realista de modelo de falhas, especialmente para redes intra-chip, uma nova notação foi proposta em [ZIM03] e é largamente aceita pelos pesquisadores da área. Esta proposta contempla falhas em múltiplos fios e com duração maior que um ciclo de relógio.

Todas as falhas de um barramento que são causadas pelo mesmo efeito físico pertencem ao mesmo tipo de falha $f_i$. Por exemplo, falhas causadas por *crosstalk* e falhas causadas por choque de partículas alfa representam dois tipos de falha. Falhas de tipos diferentes são independentes.

O modelo de falha FM($f_i$) descreve falhas do tipo $f_i$ numa determinada arquitetura de barramento. Este modelo é baseado na probabilidade de ocorrência de falhas ($a_i$), nas suas características descritas pela matriz $P_i$ e em uma métrica de distância $DM_i$, baseada na arquitetura do barramento. Dessa maneira um modelo de falhas representando um determinado efeito físico pode ser descrita da seguinte maneira:

$$FM : f_i \rightarrow (a_i, P_i, DM_i)$$

A notação $a_i$ representa a probabilidade relacionada à ocorrência de um determinado tipo de falha. Um valor normalmente observado de $a_i$ na prática é da ordem de $10^{-9}$. A característica da falha, denotada pela matriz $P_i$, indica de que maneira que a falha com probabilidade $a_i$ irá afetar o circuito. Um tipo de falha $f_i$ ocorre com uma probabilidade $p_i(w,d,e)$, afetando $w$ fios durante $d$ ciclos de relógio e causa o efeito $e$. Alguns exemplos do efeito $e$ são: inverter o valor lógico do fio,

forçar o valor do fio ao valor lógico 0 ou 1 e atrasar a propagação do sinal de maneira que ele permaneça com o valor anterior.

Todas as possíveis combinações de *pi*(*w*,*d*,*e*) são representadas na matriz $P_i$, com dimensões $w_{max}$x ($d_{max}$+1) x $e_{max}$. $w_{max}$ e $d_{max}$ representam os valores máximos de fios afetados e duração da falha respectivamente, e podem variar dependendo do tipo de falha $f_i$. Enquanto $d_{max}$ pode assumir qualquer valor positivo, $w_{max}$ é limitado pela largura do barramento.

A matriz abaixo ilustra um exemplo genérico para esta notação de modelo de falhas levando em consideração apenas o efeito de inversão, denotado por *inv*. É importante observar que com apenas um efeito, a matriz $P_i$ deixa de ser tridimensional passando a ter apenas o eixo da duração das falhas e o número de fios afetados.

$$P_i = \begin{bmatrix} p_i(1,0,inv) & \cdots & p_i(1,d_{\max},inv) \\ p_i(2,0,inv) & \cdots & p_i(2,d_{\max},inv) \\ \cdots & \cdots & \cdots \\ p_i(w_{\max},0,inv) & \cdots & p_i(w_{\max},d_{\max},inv) \end{bmatrix}$$

Uma falha transiente ocorre durante pelo menos um ciclo de clock ($d \geq 1$) e em pelo menos um fio ($w \geq 1$). Os elementos de $P_i$ com $d=0$ representam a probabilidade de falhas permanentes. Uma vez que os elementos da matriz definem as probabilidades de diferentes características de uma determinada ocorrência de falha, eles devem satisfazer a seguinte condição:

$$\sum_{a=1}^{w_{\max}} \sum_{b=0}^{d_{\max}} \sum_{c=1}^{e_{\max}} p_i(a,b,c) = 1$$

A probabilidade de ocorrência de uma falha do tipo $f_i$ afetando *w* fios durante *d* ciclos de clock pode ser obtida a partir do valor normalizado encontrado na matriz $P_i$, $p_i$(*w*,*d*,*e*), multiplicado pela probabilidade de ocorrência $a_i$. A matriz $P_1$, apresentada abaixo, apresenta um exemplo numérico desta notação de falhas e ajuda a entender melhor este conceito. Neste exemplo o efeito é fixo e indica inversão de um bit (*e=inv*).

$$P_1 = \begin{bmatrix} 0 & 0.65 & 0.1 \\ 0 & 0.2 & 0.05 \end{bmatrix}$$

A partir da interpretação desta matriz podemos observar que se a falha $f_1$ ocorrer, ela irá afetar um fio durante um ciclo de clock em 65% dos casos. A probabilidade que ela aconteça em dois fios durante um ciclo é de 20%. Em 10% das ocorrências da falha $f_1$, um fio será perturbado durante dois ciclos. Os 5% restantes correspondem a ocorrência da falha $f_1$ em dois fios durante dois ciclos de clock.

A métrica de distância define quais os fios que uma determinada falha vai atingir caso ela ocorra em mais de um fio. Esta métrica depende da arquitetura do barramento e diz que a falha que atinge um determinado fio irá atingir os fios mais próximos a ele. Caso múltiplos fios tenham a mesma distância, a escolha entre eles pode ser randômica em cenários de simulação.

Esta notação permite realizar simulações mais precisas do comportamento de falhas em uma rede intra-chip para avaliar o desempenho das técnicas de tolerância a falhas. Estes modelos, apesar de abstraírem os detalhes físicos da ocorrência das falhas, permitem realizar testes probabilísticos em um alto nível de abstração, gerando resultados rápidos com boa proximidade dos resultados observados na prática.

## 3.3 Modelagem de Crosstalk

O efeito de *crosstalk* pode ser caracterizado pela interferência visualizada em um sinal como resultado da atividade de chaveamento em linhas vizinhas. A perturbação resultante age como um ruído e pode causar falhas transientes difíceis de serem reproduzidas e identificadas. É importante observar que apesar do efeito de *crosstalk* apresentar natureza transiente, sendo um pulso de curta duração, o seu efeito é considerado permanente, pois se as mesmas condições que levam um circuito a uma falha de *crosstalk* se repetirem em diferentes instantes de tempo, a falha também se repete. Estas condições compreendem temperatura, frequência de operação e mesmo conjunto de transições em linhas adjacentes do circuito.

O acoplamento entre fios próximos pode ter natureza capacitiva e indutiva. O acoplamento capacitivo é dominante principalmente em linhas paralelas mais longas, enquanto a preocupação com o acoplamento indutivo é maior em circuitos de entrada e saída de projetos mistos (digital-analógico), não representando ameaças no projeto digital [RAB03].

O efeito de *crosstalk* pode causar diversos efeitos indesejáveis como atrasos excessivos em sinais, picos de tensão, positivos e negativos, e até uma redução no atraso de uma rede, o que também pode causar falhas de operação do circuito.

O aumento da capacitância de acoplamento entre uma rede de fios pode ser causada pela redução do espaçamento entre condutores, o aumento da distância que os fios percorrem paralelamente e o aumento da densidade de fios devido ao aumento das camadas de metal disponíveis para o desenvolvimento dos circuitos integrados.

O grau de *crosstalk* em uma rede está associado à capacidade de corrente do sinal, ao tamanho da linha, à freqüência do relógio, ao escorregamento do sinal de relógio e à sua

impedância. Variações no processo de fabricação podem aumentar excessivamente as capacitâncias de acoplamento em redes de um circuito introduzindo falhas de temporização em circuitos que originalmente trabalhariam sem problemas em uma determinada freqüência. Dessa forma, técnicas de recuperação de erros em linhas longas de comunicação podem aumentar o rendimento na produção de circuitos integrados.

### 3.3.1 Definição de Erros de *Crosstalk*

A maneira pela qual as falhas de *crosstalk* podem ocasionar erros podem ser quantificadas formalmente pela especificação de alguns parâmetros de sinais definidos abaixo [CUV99].

- **$Vp_{th}$** - Tensão de *threshold* na qual um pico de tensão positivo pode causar um erro.
- **$Vn_{th}$** - Tensão de *threshold* na qual um pico de tensão negativo pode causar um erro.
- **$Vr_{th}$** - Tensão na qual um atraso de subida pode causar m erro.
- **$Vf_{th}$** - Tensão na qual um atraso de descida pode causar um erro.
- **$\Delta t_g$** - Tempo mínimo de duração que um pico de tensão deve permanecer acima de sua tensão de *threshold* para gerar um erro.
- **$\Delta t_d$** - Tempo mínimo entre o disparo de um sinal em um barramento até sua amostragem no qual um atraso gera um erro.

Através destes parâmetros, é possível formalizar a definição de erros da seguinte maneira:

- **$g_p$** - Erro de pico positivo: pico positivo em uma linha vítima que passa de $Vp_{th}$ por uma duração maior ou igual a $\Delta t_g$.
- **$g_n$** - Erro de pico negativo: pico negativo em uma linha vítima que fica abaixo de $Vn_{th}$ por uma duração maior ou igual e $\Delta t_g$.
- **$d_r$** - Erro de atraso de subida: um atraso maior ou igual a $\Delta t_d$ faz com que uma vítima não atinja $Vr_{th}$ durante uma transição de subida.
- **$d_f$** - Erro de atraso de descida: Um atraso maior ou igual a $\Delta t_d$ faz com que uma vítima não atinja $Vf_{th}$ durante uma transação de descida.

### 3.3.2 Impacto do Crosstalk no Desempenho

Para ilustrar os efeitos do *crosstalk*, considera-se um barramento de *N* bits, onde fios de tamanho *L* são roteados em paralelo, igualmente espaçados, e acionados independentemente.

Assume-se que todas as entradas *N* transacionam ao mesmo tempo. Devido à capacitância existente entre os fios, o atraso do fio *k* é uma função das transições de seus vizinhos, *k*-1 e *k*+1. Baseando-se no modelo de Elmore, é possível obter-se uma aproximação deste atraso através da seguinte equação [RAB03]:

$$t_{p,k} = g.C_w\,(0.38.R_w + 0.69.R_D)$$

Onde: $C_W=c_wL$ e $R_W=r_w$. As variáveis $c_w$ e $r_w$ representam a capacitância entre o fio e o terra, e a resistência do fio por unidade de comprimento, respectivamente, enquanto $R_D$ é equivalente à resistência de ativação do sinal. O fator de correção *g* introduz o efeito de *crosstalk* e é uma função da razão $r=c_i/c_w$, e das atividades nos fios. $c_i$ representa a capacitância entre dois fios por unidade de comprimento. O valor *g* para um conjunto representativo de cenários é dado na Tabela 2. Quando os três fios (*k*-1,*k*,*k*+1) transacionam na mesma direção, a capacitância entre os fios não influencia no atraso e *g*=1. O pior caso ocorre quando os dois vizinhos transacionam em direções opostas ao fio *k*, onde *g*=1+4*r*. Para um caso comum onde $c_i=c_w$, *g*=5. Dessa maneira, o atraso de um fio pode variar até 500% entre o pior e o melhor caso, puramente através da direção da transição dos fios.

**Tabela 2. Relação entre o sentido de transição e do fator de atraso g na vítima k. Os símbolos ↑, ↓ e – representam transição positiva, transição negativa e sem transição, respectivamente.**

| bit *k*-1 | bit *k* | bit *k*+1 | Fator *g* |
|---|---|---|---|
| ↑ | ↑ | ↑ | 1 |
| ↑ | ↑ | – | 1+*r* |
| ↑ | ↑ | ↓ | 1+2*r* |
| – | ↑ | – | 1+2*r* |
| – | ↑ | ↓ | 1+3*r* |
| ↓ | ↑ | ↓ | 1+4*r* |

### 3.3.3 Modelo MAF (Maximal Aggressor Fault)

O modelo MAF, proposto em [CUV99], foi introduzido com o objetivo de facilitar a criação de padrões de teste para exercitar a ocorrência de *crosstalk* em um circuito integrado. Este modelo reduz o conjunto de falhas considerando as combinações de pior caso das capacitâncias de acoplamento entre todos os possíveis agressores. No caso de acoplamento resistivo/capacitivo, é possível observar três efeitos quando todas as linhas agressoras transacionam simultaneamente na mesma direção.

1) O efeito de acoplamento de toda capacitância entre a vítima e os agressores são somados construtivamente quando a vítima transaciona em sentido oposto às agressoras ou permanece no mesmo estado.

2) O efeito da capacitância entre agressoras é reduzido, uma vez que a diferença de tensão entre essas linhas é, idealmente, zero para transições no mesmo sentido.
3) A impedância total dos agressores em relação à fonte de corrente da vítima é mínima, uma vez que as impedâncias das agressoras são somadas em paralelo, facilitando a atividade agressora e aumentando a interferência causada na linha vítima.

Portanto, o modelo MAF considera todos os N-1 agressores em um barramento transacionando na mesma direção como uma falha. Este modelo considera apenas uma falha para cada erro em uma linha vítima $Y_i$, e somente um conjunto de transições pode excitar esta falha. A Figura 2 mostra as transições necessárias na linha vítima e nas linhas agressoras que podem excitar quatro tipos diferente de falhas em uma vítima $Y_i$ segundo o modelo MAF, conforme apresentado na Seção 3.3.1

É importante observar que quando o atraso de propagação de uma interconexão longa é projetado para respeitar tempos de *setup* e *hold*, a diminuição no atraso de propagação também pode gerar condições de erro, porém o modelo MAF não leva em conta estas condições, abordando apenas erros de pico e de atraso.

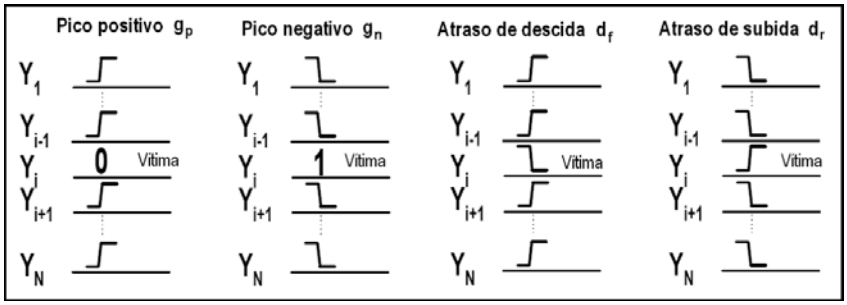


**Figura 2. Transições necessárias no MAF model.**

Para um conjunto de N interconexões, o modelo de falhas possui 4N falhas, e necessita de 4N padrões de teste. Quando o defeito é capacitivo ou resistivo, o número de falhas cobertas pelo modelo MAF é suficiente, cobrindo um número suficiente de falhas de pico e de atraso, facilitando a análise de um barramento propenso a *crosstalk*.

Dentro do escopo deste trabalho o modelo MAF é utilizado para acionar a injeção de falhas nas interconexões da rede intra-chip, com o objetivo de avaliar os métodos de recuperação de erros.

## 3.4 Módulo Saboteur

Em [GRA01] são apresentadas diversas técnicas de injeção de falhas em projetos no nível RTL, entre elas é dado o conceito de sabotadores. Baseado neste conceito foi desenvolvido no contexto deste trabalho o módulo denominado *saboteur* para controlar a injeção de falhas durante o processo de simulação. Este módulo é responsável por monitorar os dados transmitidos em todos os canais de comunicação de rede, e de acordo com os padrões encontrados, mudar o valor de alguns bits para simular o efeito de *crosstalk*. O modelo MAF é utilizado como referência para a implementação do módulo *saboteur*.

Além da injeção de falhas, cada módulo *saboteur* é responsável pela contagem de *flits* transmitidos durante uma simulação completa da rede, bem como a contagem de falhas inseridas no canal, possibilitando a geração de estatísticas relacionadas à injeção de falhas em cada canal e conseqüentemente em toda a rede.

A Figura 3 apresenta um diagrama que ilustra o funcionamento básico do módulo *saboteur*.

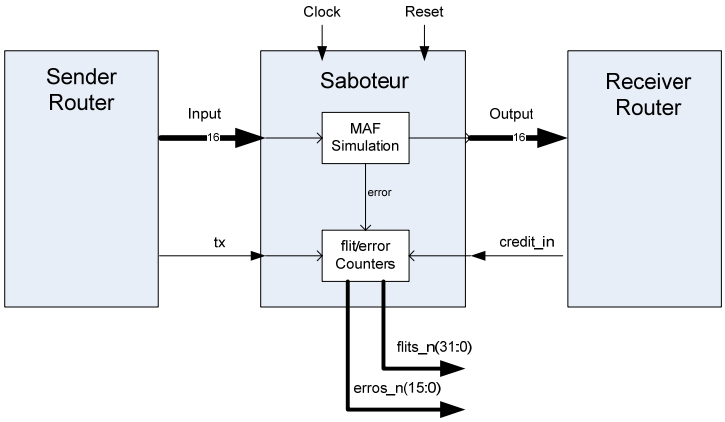


**Figura 3. Diagrama de interfaces do módulo saboteur.**

Como podemos observar o módulo *saboteur* é inserido nos canais da rede entre um roteador origem e um roteador destino, e possui internamente dois sub-módulos, sendo um responsável pela simulação do modelo MAF e outro responsável pela contagem de *flits* transmitidos e erros inseridos no canal.

O sub-módulo responsável pela contagem de *flits* transmitidos e de falhas inseridas recebe o sinal *TX* do roteador que está enviando os dados e o sinal *credit_in* do roteador que está recebendo, para verificar se o roteador destino está efetivamente consumindo os dados para contar o número total de *flits* transmitidos. A partir do sinal de erro, recebido do simulador do modelo MAF, é feita a contagem de falhas injetadas. Esses dados são enviados para um controlador central em forma de barramento de dados, sendo representados pelas portas *erros_n* e *flits_n*.

O sub-módulo responsável pela simulação do modelo MAF, recebe o sinal *input* e o armazena em um registrador interno para comparar a entrada atual com a anterior. A partir desta comparação é feita a verificação das condições do modelo MAF. A verificação das condições de atraso de subida e descida e também de pico positivo e negativo, apresentadas na Seção 3.3.3, são observadas a cada 5 bits do barramento, ou seja, para cada vítima são observados dois bits a sua esquerda e dois bits a sua direita. Teoricamente, os demais bits do barramento também possuem influência sobre a vítima, porém enquanto mais próximos os fios, maior é o acoplamento, por isso optou-se por utilizar duas linhas agressoras de cada lado da vítima. A Figura 4 apresenta uma condição onde o módulo *saboteur* insere uma falha.


**Figura 4. Exemplo de operação do módulo *saboteur*.**

Como podemos observar, o sinal *input* registra uma transição de 110110 para 001000 em seus 6 últimos bits, considerando que o módulo observa quatro linhas agressoras para uma vítima, esta transição gera a injeção de uma falha no barramento. Sendo assim, a saída do *saboteur*, representada pelo sinal *output*, permanece durante um ciclo de relógio com o valor diferente da entrada, para que a condição de atraso seja observada pelo receptor da informação.

# 4 ESTADO DA ARTE

Este capítulo tem por objetivo revisar o estado da arte em tolerância a falhas para redes intra-chip, tanto em questões de implementação quanto em requisitos de desempenho e métricas propostas para avaliar o desempenho de redes tolerantes a falhas. Como poderá ser observado, a preocupação com tolerância a falhas em redes intra-chip é algo bastante recente, sendo que as primeiras publicações na área datam de 2003.

## 4.1 Recuperação de Erros por Comunicação Estocástica

Um dos primeiros trabalhos publicados com o objetivo de explorar o problema de confiabilidade em redes intra-chip, levando em conta a maior probabilidade de falhas introduzidas pelas novas tecnologias submicrônicas, propõe um novo paradigma de repasse de mensagens baseado num algoritmo de *broadcast* probabilístico [MAR03].

Em [MAR03] o autor defende sua idéia com o argumento que o sistema tradicional de detecção e reenvio pode ser muito custoso em termos de latência e pode causar *deadlocks*, principalmente a uma taxa de erros mais elevada. Também argumenta que a largura de banda nas interconexões intra-chip é elevada e pode suportar diversas cópias de uma mesma mensagem circulando na rede em troca de uma latência menor no caso uma falha. Entretanto, neste esquema muito mais pacotes são trocados em relação aos esquemas tradicionais apenas para melhorar a latência em caso de falhas e facilitar o algoritmo de recuperação de falhas. Isto pode ocasionar maior consumo de potência e congestionamento na rede à medida que a taxa de injeção aumenta.

O comportamento deste algoritmo é similar à passagem de uma informação entre um grupo de amigos. Assumindo que inicialmente apenas uma pessoa no grupo possui a informação, esta pessoa pode passar a informação para outra pessoa aleatoriamente. Em um próximo momento, qualquer um dos dois que possuam a informação pode passá-la adiante independentemente para outras duas pessoas. O processo continua da mesma maneira até que o grupo inteiro tenha recebido a informação.

A analogia deste processo para redes intra-chip é feita de maneira que cada roteador da NoC representa uma pessoa, e os pacotes transmitidos representam as informações. Admitindo-se uma arquitetura em malha, cada roteador é conectado com no máximo quatro roteadores e deve escolher um dos roteadores aos quais está conectado aleatoriamente para o envio da mensagem, criando assim o conceito de comunicação estocástica, ou de broadcast probabilístico. Se um roteador detectar um erro na mensagem ele pode simplesmente descartá-la, pois certamente após

mais alguns ciclos este roteador receberá uma nova cópia da mensagem e poderá repassá-la ou consumi-la.

O desempenho da comunicação estocástica pode ser controlado utilizando-se alguns parâmetros. A mensagem pode chegar ao seu destino antes que o broadcast seja finalizado, de maneira que o espalhamento da mensagem pode ser interrompido antes do término do broadcast para reduzir o número de mensagens transmitidas pela rede, economizando banda e energia. Este controle pode ser realizado atribuindo-se um tempo de vida (*ttl – time to live*) a cada mensagem.Cada roteador que encaminhar uma mensagem decrementa este tempo de vida até que atinja o valor zero, podendo então descartar a mensagem quando este valor for atingido.

Outro parâmetro importante que ajuda a controlar o desempenho do algoritmo é a probabilidade (*p*) que uma mensagem será transmitida através de um determinado canal. Este parâmetro pode diminuir o número de mensagens transmitidas e o número de ciclos que levará para chegar a seu destino, visto que será encaminhada com uma probabilidade maior para um caminho que mais se aproxime ao seu destino.

Variando-se os parâmetros *p* e *ttl*, este algoritmo pode apresentar um bom desempenho sem aumentar muito o número de mensagens que trafegam na rede. A partir deste algoritmo também é possível que uma mensagem seja roteada de uma origem a um determinado destino através de vários caminhos, de maneira que os caminhos com falhas permanentes ou com alta probabilidade de falhas se mantenham transparentes ao algoritmo de roteamento. Portanto, este é um algoritmo bastante simples de se implementar, que pode atribuir diversos caminhos para que uma mensagem seja transmitida entre um par de roteadores, garantindo tolerância a falhas, com o custo de algumas transmissões desnecessárias.

Em [MAR03] o autor apresenta resultados de simulação do seu algoritmo variando a taxa de erros observada no sistema e os parâmetros *p* e *ttl*, mostrando os ganhos obtidos ao variar estes parâmetros, porém não compara a sua proposta com implementações tradicionais de roteamentos adaptativos e de recuperação de erros em redes intra-chip. O artigo [ZHU07] faz uma comparação mais crítica entre os roteamentos adaptativos propostos para tolerância a falhas em redes intra-chip e demonstra o baixo desempenho do algoritmo de broadcast probabilístico. Alguns pontos apontados são:

- O algoritmo de broadcast probabilístico causa uma sobrecarga de comunicação significativa, que limita o seu uso apenas a aplicações com baixa taxa de injeção de pacotes na rede.

- Este algoritmo tende a ser mais resistente a falhas. Apenas com taxas de erro acima de 10% que os pacotes começam a ser perdidos. Mas na prática, esta é uma probabilidade de erro muito grande.

Por fim, apesar dos ganhos introduzidos pelos ajustes da probabilidade do caminho em que as mensagens serão transmitidas e da diminuição do overhead de comunicação através da inserção de tempo de vida nas mensagens, a eficiência do algoritmo de broadcast probabilístico é baixa se comparada com algoritmos adaptativos mais recentes para tolerância a falhas em NoCs [MUR07].

## 4.2 Recuperação de Erros por Correção e por Detecção+Reenvio

A partir do aumento da preocupação com a tolerância a falhas em redes intra-chip, principalmente em cenários onde é necessário alto desempenho, com freqüências de operação na ordem de gigahertz e com altos requisitos de confiabilidade, tanto pesquisadores quanto os demais projetistas de NoC passaram a utilizar por padrão codificação de controle de erro nos pacotes que circulam na rede.

A arquitetura apresentada em [BER04], denominada *Xpipes*, é um exemplo da utilização intensa de controle de erros em redes intra-chip para garantir o desempenho da rede mesmo na ocorrência de falhas transientes. *Xpipes* é descrita como sendo uma rede baseada em chaveamento por pacotes de alta velocidade, projetada com o objetivo de suportar altas freqüências de operação para atender a demanda das novas aplicações de alto desempenho que compõem os SoCs atuais. A partir da ferramenta denominada *XpipesCompiler* é possível criar redes parametrizáveis com topologias arbitrárias para serem utilizadas como meio de interconexão de sistemas heterogêneos. O grau de parametrização da arquitetura *Xpipes* permite tanto modificações em cada bloco interno dos roteadores quanto parâmetros globais da rede. Os parâmetros que podem ser alterados incluem: o tamanho do *flit*, o grau de redundância associado à lógica de controle de erro, a localização dos núcleos, os métodos de controle de fluxo e também o número de registradores que serão inseridos em um enlace entre dois roteadores, com o objetivo de garantir que a rede opere como um todo a uma determinada freqüência definida a priori.

A Figura 5 apresenta a estrutura genérica de uma parte do roteador da arquitetura *Xpipes* com o sistema de detecção de erros empregado nesta rede.

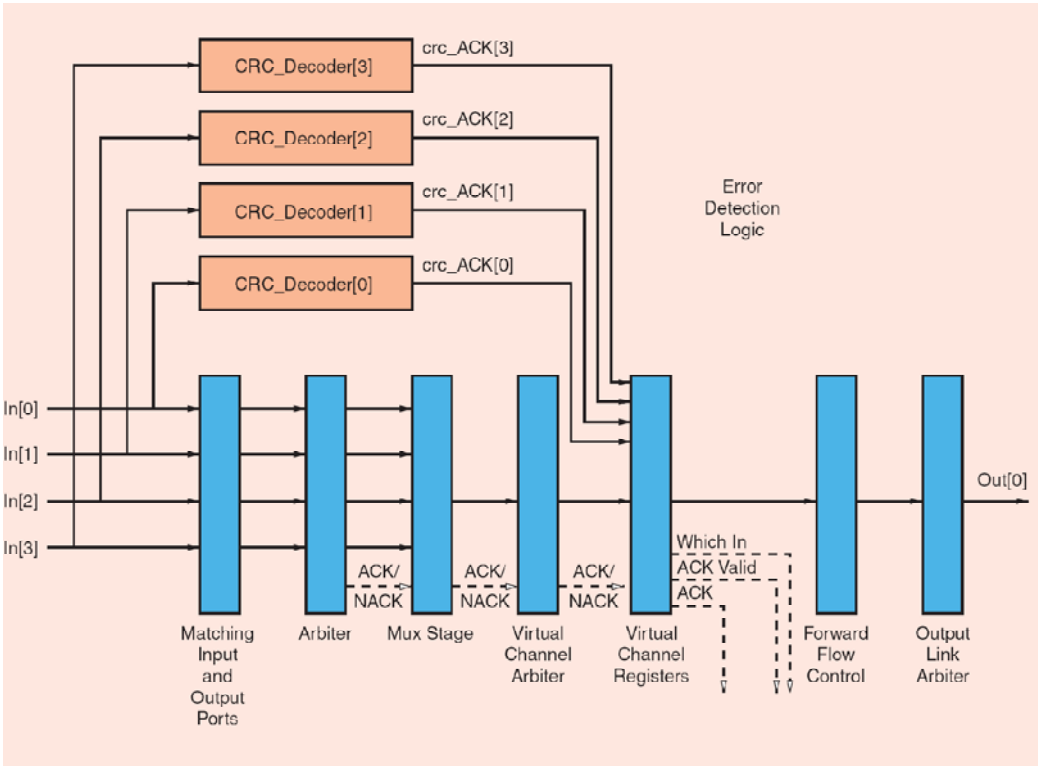


**Figura 5. Arquitetura do roteador da rede Xpipes, incluindo lógica de controle de erros [BER04].**

Cada canal de entrada do roteador possui um decodificador CRC dedicado para a detecção de falhas nos *flits* recebidos. A interface de rede (NI) do sistema deve realizar o cálculo dos bits de *overhead* de maneira transparente como parte da construção do pacote. Na Figura 5 é possível observar a estrutura do *pipeline* de sete estágios do roteador, onde cada *flit* passa pelos estágios de arbitragem, multiplexação, lógica de canais virtuais e registradores de saída. Esta estrutura de *pipeline* é utilizada para maximizar a freqüência de operação do roteador. Os decodificadores CRC trabalham em paralelo à estrutura de *pipeline* do roteador, de maneira que ao detectar um erro é gerado um sinal de NACK. Existe ainda um sistema de chaveamento centralizado responsável por encaminhar o sinal de NACK ao roteador que originou a mensagem corrompida. Ao receber este sinal o roteador originador da mensagem deve providenciar o seu reenvio.

### 4.2.1 Recuperação Fim-a-Fim versus Roteador-a-Roteador

Como foi visto na seção anterior, a rede *Xpipes* emprega um controle de falha roteador-a-roteador no nível de *flit* para que a recuperação de uma falha não acarrete em um grande aumento de latência. Entretanto, a recuperação de falha fim-a-fim também é uma alternativa viável.

A escolha do local onde a proteção de erro será implementada é uma decisão crítica no projeto de NoCs tolerantes a falha e poucos trabalhos publicados exploram métricas suficientes para avaliar o desempenho destas aplicações. Na literatura pesquisada, o artigo [MUR05]

apresenta comparações entre as diferentes opções de posicionamento do controle de erro ao longo de uma rede intra-chip, porém levando em conta apenas simulações em alto nível.

Três esquemas de recuperação de erros foram analisados em [MUR05]: fim-a-fim, roteador-a-roteador e híbrido, com os três esquemas presentes na mesma rede. Para avaliar o aumento de área inserido pelos codificadores e decodificadores, a dissipação de potência e o desempenho das aplicações executando sobre este sistema, os autores definem o modelo de NoC que é utilizada como referência para implementação destes esquemas de recuperação de erro da seguinte maneira:

- Roteadores com filas de entrada e fluxo de controle baseado em crédito;
- Roteamento estático na origem.

Para garantir a máxima vazão nesta rede, cada porta de entrada de um roteador deve ter um buffer capaz de armazenar pelo menos $2N_L+1$ *flits*, onde $N_L$ é o número de ciclos necessários para atravessar um canal entre dois roteadores[1]. A razão deste valor é devido ao controle de fluxo baseado em crédito, onde $N_L$ ciclos são necessários para o sinal de crédito chegar ao roteador origem e mais $N_L$ ciclos para que o *flit* chegue ao roteador destino. A seguir serão apresentadas as características dos diferentes tipos de implementação apresentados em [MUR05].

- **Proteção de erro fim-a-fim:** No esquema de proteção de erros fim-a-fim (*ee*), bits de paridade (*ee-par*) ou de CRC (*ee-crc*) são adicionados aos pacotes. O codificador de paridade ou CRC é adicionado à interface de rede (NI) do roteador que envia a mensagem, e o decodificador à NI que recebe a mensagem. A NI origem possui um ou mais buffers de pacotes onde são armazenados os pacotes que foram transmitidos. A NI receptora envia um sinal de *ack/nack* de volta ao roteador origem informando se o pacote recebido possui erros ou não. Numa transição com resposta, como um pedido de leitura, o sinal de *ack/nack* pode ser enviado juntamente com o pacote de resposta. Para contabilizar os erros nos pacotes de *ack/nack*, um mecanismo de *timeout* para retransmissão no NI origem foi implementado. Para detectar o recebimento de pacotes duplicados, foram utilizados identificadores de seqüência nos pacotes. Como o *flit* de cabeçalho possui informações críticas, como informações de roteamento, ele deve ser protegido também por paridade ou CRC, de maneira que cada vez que ele atravesse um roteador, sua integridade deve ser verificada. Se um roteador detectar um erro no *flit* de cabeçalho, ele pode descartar o pacote. Um diagrama desta arquitetura pode ser

[1] Na rede Xpipes os canais são bufferizados

observado na Figura 6(a).

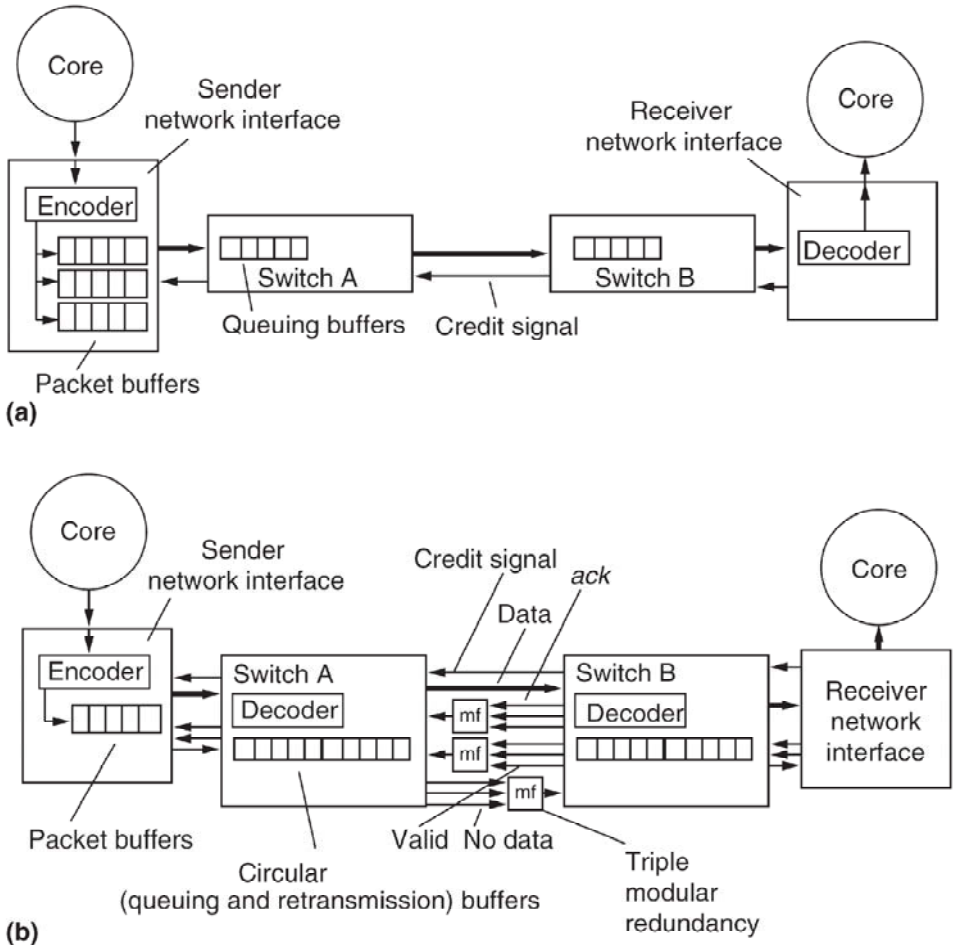


**Figura 6. Esquemas de recuperação de erros fim-a-fim (a) e roteador-roteador (b) [MUR05].**

- **Proteção de erro roteador-a-roteador:** O esquema de proteção roteador-a-roteador adiciona hardware de detecção na porta de entrada de cada roteador e retransmite os dados entre roteadores adjacentes. Foram implementados dois tipo de proteção roteador-a-roteador: paridade ou CRC no nível de *flit* e no nível de pacote. A arquitetura do roteador desta implementação é apresentado na Figura 6(b). Os buffers adicionados a cada porta de entrada dos roteadores armazenam os pacotes até o recebimento do sinal de *ack/nack* do próximo roteador. O número de buffers necessários para suportar retransmissões entre roteadores depende se a proteção é realizada no nível de *flit* ou no nível de pacote.

  No esquema de proteção roteador-a-roteador no nível de *flit* (*ssf*), A NI ou o roteador originador da mensagem adicionam os bits de paridade ou CRC a cada *flit*. A cada porta de entrada de um roteador existem dois conjuntos de buffers: buffers para controle de fluxo baseado em crédito, como na arquitetura básica do roteador, e buffers

de retransmissão para suportar o mecanismo de retransmissão roteador-a-roteador. Assim como os buffers de entrada, os buffers de retransmissão devem ter uma capacidade de $2N_L+1$ *flits* para garantir a operação da rede em latência máxima. Como pode ser observado na Figura 6(b), é utilizado um mecanismo de redundância (TMR) nas linhas de controle, tais como *ack*.

No esquema de proteção roteador-a-roteador no nível de pacote (*ssp*), os bits de paridade ou CRC são adicionados ao último *flit* do pacote. Como a detecção de erro acontece apenas quando o último *flit* atinge o próximo roteador, o número de buffers de retransmissão necessários a cada porta de entrada é $2N_L+f$, onde $f$ é o número de *flits* do pacote. O esquema *ssp* também necessita de proteção no *flit* de cabeçalho, assim como o esquema *ee*.

- **Esquema de recuperação híbrido:** No esquema de proteção híbrido foram implementados mecanismos de correção simples e detecção múltipla de erros (*ec+ed*), onde qualquer receptor pode corrigir um erro simples em um *flit*, mas para múltiplos erros é necessário o reenvio fim-a-fim do pacote pelo NI origem.

A eficiência destes diferentes métodos de proteção de erros foram investigados utilizando-se como referência uma rede 4 × 4 do tipo malha com 16 núcleos e 16 roteadores. Os pacotes utilizados contém 4 *flits* de 64 bits cada. Foram executadas diversas simulações com tráfego uniforme variando-se a taxa de erros e a taxa de injeção dos pacotes na rede. A Figura 7 apresenta resultados destas simulações levando em conta a latência observada no sistema com os diferentes esquemas de proteção de erros.

Como mostra a Figura 7, com baixa taxa de erro e baixa taxa de injeção, os vários esquemas de proteção de erro apresentam praticamente as mesmas latências. Entretanto, à medida que a taxa de erros e injeção crescem, o esquema de recuperação fim-a-fim (*ee*) demonstra maior aumento de latência que os outros esquemas. O esquema de recuperação roteador-a-roteador no nível de pacote (*ssp*) apresenta um pequeno aumento de latência em relação ao esquema roteador-a-roteador no nível de *flit* (*ssf*), pois no esquema *ssf* os erros são detectados mais cedo. Como esperado, o esquema híbrido apresenta as menores latências.

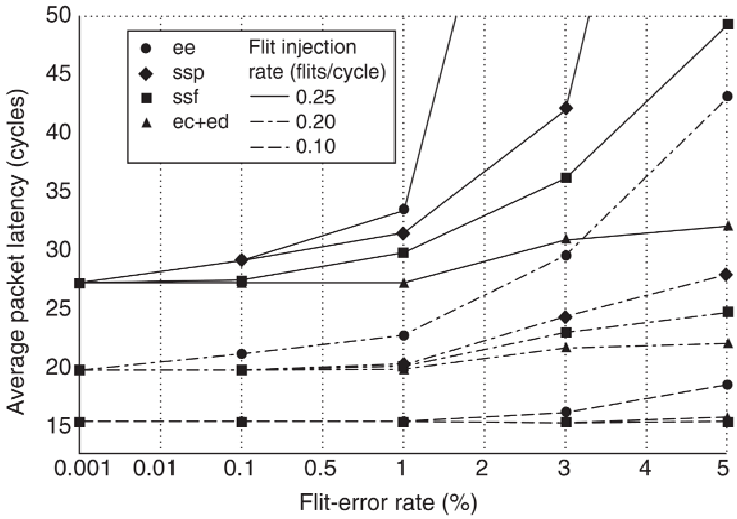


**Figura 7. Latências observadas nos diferentes esquemas de proteção de erros. [MUR05]**

A Figura 8 apresenta um gráfico com o consumo de energia dos esquemas de recuperação de erro levando em conta apenas a taxa de erro. As simulações realizadas para estimar o consumo de potência das diferentes implementações foram obtidas através da modelagem dos componentes na tecnologia 70*nm* utilizando o modelo descrito em [http://www-device.eecs.berkley.edu/~ptm/](http://www-device.eecs.berkley.edu/~ptm/). A tensão de operação foi fixada em 0.85v. Para o hardware de detecção e correção de erro foram utilizadas regras de *layout* para torná-los mais tolerantes a falhas transientes, levando a um consumo de potência adicional destes componentes de 8% a 10%, que também foi levado em conta nestas simulações. A distância entre dois roteadores foi definida em 2 ciclos, ou seja, para um *flit* atravessar um canal ele deve passar por dois registradores intermediários. A taxa de injeção foi mantida constante em 0.1 *flits* por ciclo (taxa de injeção igual a 10%).

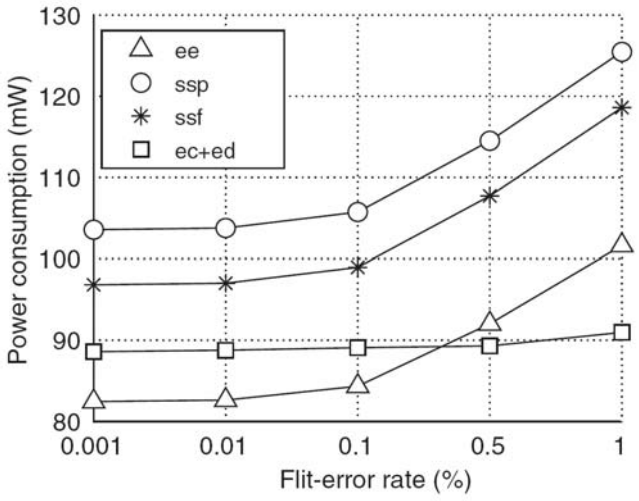


**Figura 8. Consumo de energia dos diferentes esquemas de proteção de erros. [MUR05]**

Os resultados mostram que o consumo dos esquemas de proteção roteador-a-roteador (*ssf* e *ssp*) são maiores que o consumo dos esquemas fim-a-fim (*ee* e *ssp*). Existem duas razões para justificar estes resultados. Em primeiro lugar, o número de buffers necessários para a retransmissão nos esquemas *ssf* e *ssp* são maiores que nos esquemas *ee* e *ec + ed*. Em segundo lugar, devido ao tráfego uniforme, o tráfego entre os roteadores é maior, uma vez que o número de roteadores que um pacote poderá atravessar é maior, aumentando o overhead na transmissão nos esquemas *ssf* e *ssp*. O esquema fim-a-fim puro consome menor potência, principalmente com baixas taxas de erro, que normalmente acontecem na prática, porém adiciona maior latência quando um erro é detectado.

Outra comparação importante apresentada em [MUR05] mostra o consumo de potência variando-se a tensão de operação para obter a taxa de erro residual desejada. A Figura 9 mostra os resultados desta comparação.

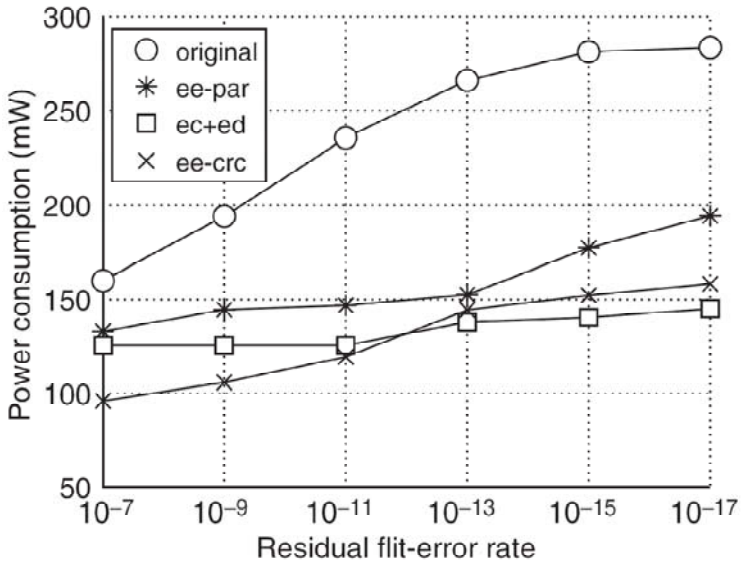


**Figura 9. Relação entre o consumo de potência dos diferentes esquemas de proteção de erros com a taxa de erros residual desejada.**

Como podemos observar, uma rede sem nenhum esquema de proteção de erros consome muito mais potência para manter a taxa de erros baixa, mesmo que o overhead de área na implementação do controle de erros não exista, pois ela deve operar a uma voltagem muito superior para garantir resistência às diversas fontes que podem interferir no circuito. O esquema *ee-par* consome mais potência que os esquemas *ee-crc* e *ec+ed* porque a capacidade de detecção de erros do código de paridade é menor, portanto necessita de uma maior tensão de operação para garantir baixa taxa residual de erros. O esquema híbrido *ec+ed* consome menos potência quando se deseja taxa de erros residual mais baixa, enquanto o esquema *ee-crc* consome menos

potência em taxas de erros residuais superiores a $10^{-12}$. A razão disso é que com altas taxas de erro, a retransmissão dos pacotes causa mais tráfego na rede. Para baixas taxas de erro residual, o mecanismo de correção empregado no esquema *ec+ed* consome menos potência do que o mecanismo de retransmissão do esquema *ee-crc*. Quando se permite uma taxa de erro residual mais elevada, o esquema *ec+ed* acaba gastando mais potência porque necessita de mais bits nos pacotes para poder corrigir e detectar erros.

Apesar dos resultados apresentados serem bastante interessantes, e ressaltarem a importância da inserção de métodos de controle de erros em redes intra-chip, eles ainda podem ser consolidados para padrões de tráfego mais realistas e com estimativas de potência mais precisas. Utilizando-se um tamanho de pacote de apenas 4 *flits*, como proposto no trabalho, a latência resultante da quebra das mensagens em muitos pacotes pequenos pode prejudicar o desempenho do sistema. Isso acontece devido ao acréscimo na latência necessária para rotear estes pacotes e da menor carga útil que será transportada. Portanto outros tamanhos de pacote e de *flit* devem ser avaliados para se obter resultados mais úteis para diversas aplicações.

### 4.2.2 Controle de Fluxo para Controle de Erro

O controle de fluxo determina como os recursos da rede são alocados para os pacotes atravessarem a rede. Podemos vê-lo também como um mecanismo para resolver os problemas de contenção na rede. A utilização de mecanismos para prover recuperação de erros depende do suporte que o mecanismo de controle de fluxo fornece. Por exemplo, para retransmitir um dado corrompido é necessário parar o fluxo de pacotes do roteador origem, sinalizando o pedido de retransmissão e realocando os buffers e a banda para a transmissão da mensagem original. Infelizmente, nem todos os métodos de controle de fluxo para NoCs são capazes de realocar os recursos após um erro de transmissão, podendo apenas gerenciar o congestionamento dos canais, como é o caso do controle de fluxo baseado em crédito. Neste caso, a confiabilidade na comunicação só pode ser garantida através de correção de erro, ou deve ser transferida para as camadas superiores à camada de enlace.

O artigo [PUL05] apresenta características e comparações de três mecanismos de controle de fluxo: STALL/GP, T-Error e ACK/NACK. As comparações foram realizadas tendo como base uma rede *Xpipes*.

Cada protocolo de controle de fluxo oferece diferentes opções de tolerância a falhas, assim como diferentes características de desempenho, potência e área, como mostra a Tabela 3.

**Tabela 3. Comparação entre os protocolos de controle de fluxo. (*N* refere-se ao número de estágios entre registradores e *M* corresponde à *N*/2)**

| | **STALL/GO** | **T-Error** | **ACK/NACK** |
|---|---|---|---|
| **Número de buffers** | $2N+2$ | $>3M+2$ | $3N+k$ |
| **Área de lógica** | Baixa | Alta | Média |
| **Desempenho** | Bom | Bom | Depende |
| **Potência (est.)** | Baixa | Média/Alta | Alta |
| **Tolerância a falhas** | Não suportada | Parcial | Suportada |

O protocolo STALL/GO é um método de controle de fluxo muito simples e requer apenas dois sinais de controle: um sinal no sentido origem-destino, indicando que existem dados disponíveis, e um sinal no sentido destino-origem, que indica se o buffer de recepção está cheio (STALL) ou se o buffer está livre (GO). STALL/GO pode ser implementado com buffers distribuídos ao longo do enlace, onde cada repetidor pode ser projetado como uma simples fila de dois estágios. A origem só necessita de dois buffers para lidar com o estado de STALL do primeiro repetidor do enlace, resultando na necessidade de $2N$ +2 registradores, com muito pouca lógica de controle. A grande desvantagem do método STALL/GO é que ele não oferece nenhum método de tratamento de falhas. Caso alguma falha ocorra, deve-se realizar o tratamento em camadas superiores. A figura abaixo mostra um diagrama de implementação do protocolo STALL/GO.

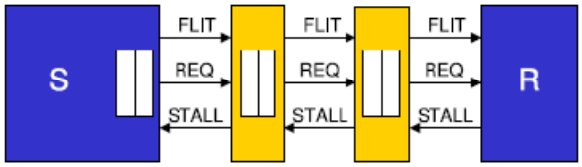


**Figura 10. Implementação do protocolo STALL/GO [PUL05].**

O protocolo T-Error é utilizado para lidar com o aumento do comprimento de um canal físico ou com o aumento da freqüência de operação. Como resultado, as falhas de temporização acontecem com mais freqüência no canal. As falhas são gerenciadas por uma arquitetura de repetidor capaz de amostrar o dado com um sinal de clock atrasado, além do clock original do sistema, para detectar possíveis inconsistências e sinalizar o repetidor vizinho. No caso de um dado recebido consistentemente o repetidor envia um sinal *valid*. Se a lógica nas extremidades do enlace permanecer inalterada, é necessário adicionar um estágio de ressincronização antes que o dado chegue no destino. Este estágio é responsável por realinhar o sinal de clock atrasado o dado a ser enviado ao destino, gerando um sinal de *valid*. Este procedimento consome mais um ciclo de clock no *pipeline* do canal.

O uso do protocolo T-Error pode ser usado para aumentar a confiabilidade de um canal, configurando o espaçamento ente os estágios e a freqüência de operação, conservando as demais camadas da rede. Porém, T-Error não possui nenhum método que possibilite um verdadeiro controle de erros, pois se os erros de temporização não forem detectados, nenhuma falha poderá ser recuperada no nível da rede, somente no nível da aplicação.

Os requisitos de área do protocolo T-Error incluem três buffers em cada repetidor e dois na origem. Além disso, a lógica de controle de recepção adiciona um pouco de área ao sistema. Uma estimativa da quantidade de buffers necessários para implementação do protocolo T-error é de 3*M*+2, sendo *M* 50% menor que *N* se este método for utilizado para aumentar o espaçamento do canal. Retransmissões desnecessárias de *flits* são evitadas com o protocolo T-Error, mas o overhead de potência ainda existe devido à lógica de controle. A Figura 11 apresenta um diagrama da implementação deste protocolo.

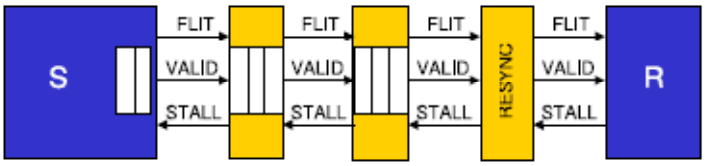


**Figura 11. Implementação do protocolo T-Error [PUL05].**

A principal idéia do protocolo ACK/NACK é que os erros de transmissão podem acontecer durante uma transição. Por esse motivo, enquanto os *flits* são enviados em um enlace, um cópia é mantida localmente em um buffer na origem. Quando os *flits* são recebidos, um sinal de ACK ou NACK é enviado de volta. No caso do recebimento de um ACK, a origem deleta a sua cópia local do *flit*. No caso de um NACK, a origem re-insere em sua saída o *flit* armazenado no buffer e sinaliza dado válido para o próximo estágio. Isto significa que qualquer *flit* que esteja a caminho durante a detecção de um erro será descartado e enviado novamente após o reenvio do dado corrompido. Outros esquemas de retransmissão também são possíveis, mas apresentam maior complexidade lógica. ACK/NACK pode ser implementado tanto fim-a-fim como roteador a roteador. Os métodos de proteção de erro são suportados pelo protocolo, mas os codificadores e decodificadores devem ser implementados juntamente com a fonte e o destino, respectivamente.

No protocolo ACK/NACK, uma vazão de um *flit* por ciclo pode ser obtida utilizando-se a quantidade certa de buffers. Os repetidores podem ser registradores simples, de maneira que com *N* repetidores, 2*N*+*k* buffers são necessários na origem para garantir vazão máxima, uma vez que o sinal ACK/NACK alimenta a origem apenas após atravessar todos os estágios do canal. O valor de *k* depende da latência da lógica na recepção e envio dos roteadores. Sobretudo, o número mínimo

de buffers necessários para que não ocorram penalidades na taxa de transmissão em um cenário onde não ocorrem NACKs é de 3*N*+*k* buffers.

O controle de fluxo baseado em ACK/NACK provê vazão máxima enquanto nenhum NACK for gerado. Se NACKs forem gerados somente por erros esporádicos, o impacto no desempenho não deve ser afetado. Entretanto, se NACKs tiverem de ser transmitidos devido a congestionamentos, a latência necessária para o sinal atravessar os estágios do canal pode trazer um grande impacto no desempenho, principalmente em canais com muitos estágios. Além do mais, muitas retransmissões podem causar desperdício de potência. A figura a seguir apresenta o diagrama de implementação do protocolo ACK/NACK.

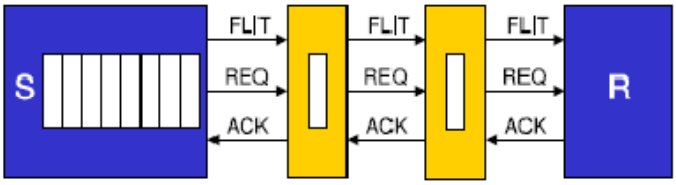


**Figura 12. Implementação do protocolo ACK/NACK [PUL05].**

## 4.3 Algoritmos de Roteamento Adaptativos Visando Tolerância a Falhas

O tipo de roteamento utilizado em NoCs pode ser definido na origem ou ser distribuído. No roteamento definido na origem, um ou mais caminhos são selecionados para os fluxos da rede em tempo de projeto. No caso de roteamento distribuído, os caminhos são selecionados baseado nas características de tráfego da rede em tempo de execução. Devido à simplicidade, e do fato que o tráfego das aplicações podem ser facilmente caracterizados, o roteamento na origem é largamente utilizado em NoCs. Quando comparado com roteamento na origem, o roteamento distribuído melhora a diversidade dos caminhos que as mensagens podem percorrer, minimizando o congestionamento na rede e os gargalos no tráfego da rede.

A maioria das arquiteturas de NoCs atuais tanto com roteamento definido na origem quanto distribuído, utilizam algoritmos determinísticos, ou seja, um único caminho entre origem-destino, como o algoritmo XY. Uma das razões deste fato é que com múltiplos caminhos permitidos, os pacotes podem chegar ao seu destino desordenados devido a diferenças no comprimento dos caminhos e nos níveis de congestionamento de cada caminho. Para muitas aplicações, a entrega fora de ordem não é aceitável. O artigo [MUR07] apresenta um método para roteamento de pacotes com múltiplos caminhos, capaz de realizar a entrega dos pacotes ordenadamente com garantias de tolerância a falhas.

A estratégia proposta em [MUR07] para a proteção contra erros transientes é o envio de cópias de um mesmo pacote através de diversos caminhos. Na interface de rede do destino, o circuito de detecção de erros verifica a consistência dos pacotes e aceita um pacote sem erros. Para evitar replicação desnecessária de caminhos, o autor utiliza um modelo de falhas para garantir que um tempo médio entre falhas residuais de alguns anos, utilizando-se o mínimo de caminhos para um mesmo pacote. Esta estratégia garante latência mínima na entrega dos pacotes mesmo em caso de falhas transientes, pois não haverá necessidade de retransmissão quando um erro for detectado. Para muitas aplicações um pedido de retransmissão pode prejudicar o desempenho do sistema, principalmente se o fluxo necessitar da vazão máxima oferecida pela rede

Para garantir a proteção contra erros permanentes, os pacotes devem ser enviados por múltiplos caminhos que não possuam intersecção. A não intersecção dos caminhos garante que se existir uma falha em um canal ou em um roteador ela não afetará os pacotes transmitidos pelos demais caminhos. No caso da localização de uma falha permanente, a origem é informada da localização desta falha e ela não considerará mais o caminho falho para rotear os pacotes.

O artigo [ZHU07] apresenta dois algoritmos adaptativos para se obter tolerância a falha em NoCs e estabelece critérios para comparação destes algoritmos. O primeiro algoritmo apresentado se chama *N random walk*, sendo um aprimoramento do algoritmo descrito na seção 4.1 (Broadcast probabilístico). O segundo algoritmo denominado *negative first*[GLA92] é um algoritmo parcialmente adaptativo bastante simples para implementação em redes do tipo malha.

O algoritmo *N random walk* permite a injeção de um número fixo de cópias (N) de uma mensagem na rede. Usando o algoritmo de caminhamento randômico, cada roteador passa adiante as cópias das mensagens para um de seus canais de saída, fazendo com que as mensagens percorram caminhos não determinísticos até o destino. A escolha do canal de saída é determinada por um conjunto de probabilidades randômicas. As probabilidades são calculadas levando-se em conta a distância *manhattan* entre o destino e o nodo atual e também entre os nodos vizinhos e o destino. A distância *Manhattan*, $D_c$, pode ser calculada a partir da equação abaixo:

$$D_C = \left|X_{destino} - X_{atual}\right| + \left|Y_{destino} - Y_{atual}\right|$$

Um fator de multiplicação *Mx* é definido em 1 para qualquer direção onde *Dx* (*x* denota o nodo atual ou qualquer um de seus vizinhos) é maior que *Dc*. Para os nodos restantes (onde *Dx≤Dc*), ou seja, aqueles que estão mais próximos do destino, o fator *Mx* é igual ao mínimo entre *Mx* e 4. Os fatores de multiplicação são normalizados para se obter as probabilidades $P_N$, $P_S$, $P_L$ e $P_O$, indicando a probabilidade que as mensagens serão enviadas para norte, sul, leste e oeste

respectivamente. Logo após, um número randômico é gerado para determinar qual será a porta de saída da mensagem, baseando-se nas probabilidades geradas no passo anterior. Através do procedimento descrito acima, as mensagens seguirão com grande probabilidade em direção ao destino, mas poderão tomar caminhos diferentes. A Figura 13 mostra um exemplo de execução do algoritmo *N random walk* com apenas uma cópia da mensagem sendo enviada na rede (N=1).

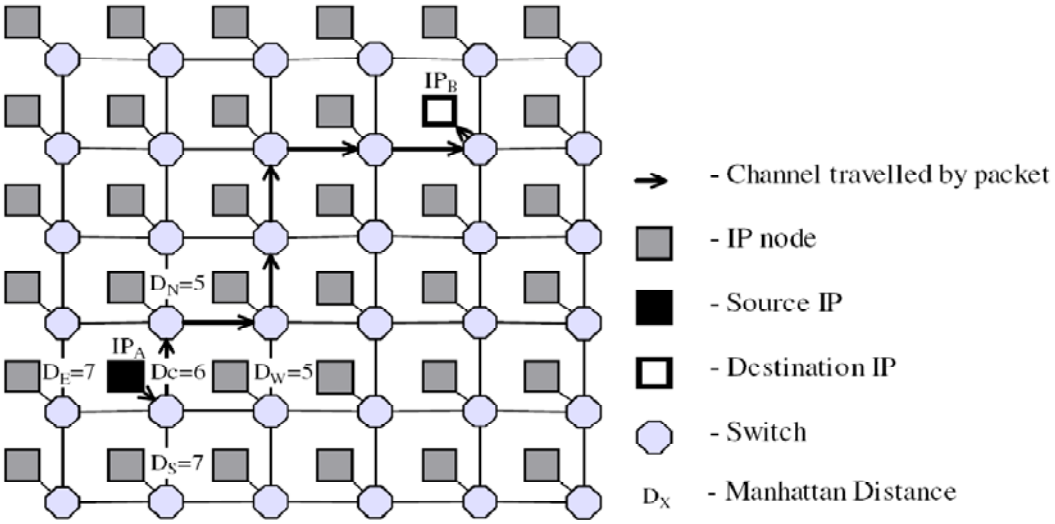


**Figura 13. Exemplo de execução do algoritmo *N random walk* com N=1 [ZHU07].**

Enquanto o algoritmo *N random walk* pode sustentar um alto nível de transferências de dados com sucesso, ele dissipa uma quantidade significativa de energia devido às mensagem redundantes que circulam na rede.

O algoritmo *negative first* pode ser usado para se obter adaptabilidade parcial no roteamento de pacotes em redes com topologia malha. Neste algoritmo algumas curvas no caminhamento dos pacotes são proibidas, de maneira que *deadlocks* são evitados. Este modelo pode ser adaptado para lidar com falhas em canais e em roteadores de uma NoC. Neste algoritmo os pacotes podem ser roteados na direção negativa em cada sentido na primeira fase, e depois ser roteados na direção positiva na segunda fase. Especificamente, a mensagem é enviada inicialmente para oeste ou sul até que ela chegue à coluna ou linha especificada, depois se move para leste ou norte. A versão tolerante a falhas deste algoritmo, roteia adaptativamente na direção negativa, mesmo depois que a mensagem tenha atingido a coluna ou a linha desejada. A efetividade deste algoritmo na presença de uma falha é apresentada na Figura 14.

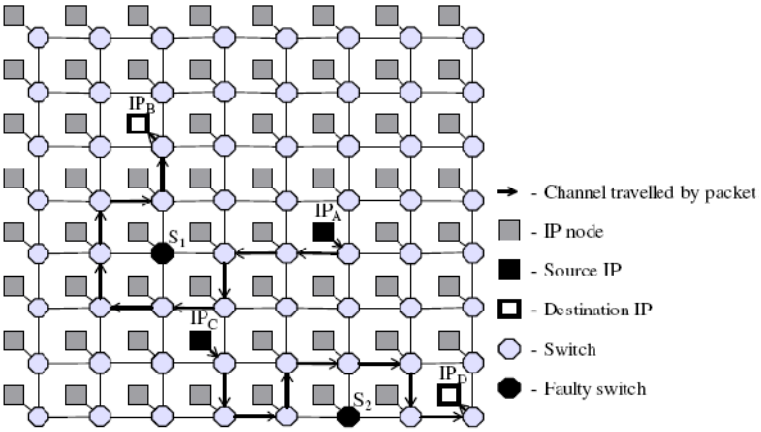


**Figura 14. Ilustração do algoritmo *negative first* na presença de roteadores com falhas permanentes [ZHU07].**

Se o algoritmo *negative first* for adaptado, como indicado na Figura 14, o pacote pode ser roteado mais ao sul e à oeste para evitar o roteador falho, e depois voltar para leste para atingir o destino. A exceção ocorre quando o pacote está sendo roteado na borda inferior da rede e encontra um roteador com falha. Como se pode observar na Figura 14, supondo que o $IP_C$ esteja tentando se comunicar com o $IP_D$ e o roteador $S_2$ esteja com uma falha permanente. Neste caso o pacote deve obrigatoriamente ser roteado ao único roteador perpendicular encontrado e depois dois roteadores em direção ao destino, para logo em seguida voltar para a borda. A principal vantagem do algoritmo *negative first* é que ele permite que os pacotes sejam roteados para oeste e sul mesmo após a chegada na linha ou coluna desejada, de maneira que mais caminhos são possíveis, e a probabilidade de que um pacote possa ser roteado com sucesso, mesmo na presença de falhas permanentes, é aumentada.

A primeira comparação apresentada em [ZHU07] leva em conta a vazão em relação à taxa de injeção de dados na rede para os dois algoritmos de roteamento apresentados, e ainda compara com os resultados obtidos com o roteamento determinístico utilizando o algoritmo XY.

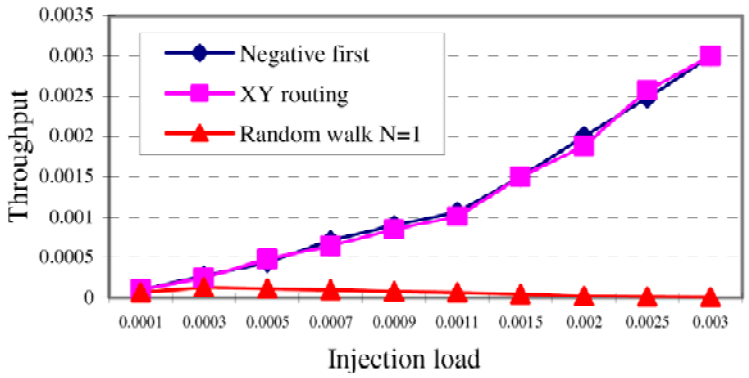


**Figura 15. Relação entre a taxa de injeção e a vazão observada nos diferentes algoritmos de roteamento [ZHU07].**

A conclusão obtida através da análise do gráfico é que apenas em taxas de injeção muito baixas, como 0.05%, o algoritmo *N random walk* tem resultados significativos. Em altas taxas de injeção a rede é rapidamente saturada. Já o algoritmo *negative first* apresenta uma vazão crescente com o aumento da taxa de injeção, praticamente acompanhando a vazão observada na utilização do algoritmo XY. Isto denota uma grande limitação no algoritmo *N random walk*.

Outra comparação apresentada em [ZHU07] diz respeito à probabilidade de chegada das mensagens com sucesso no destino variando-se a taxa de erros induzidos na rede. A Figura 16 apresenta os resultados desta análise.

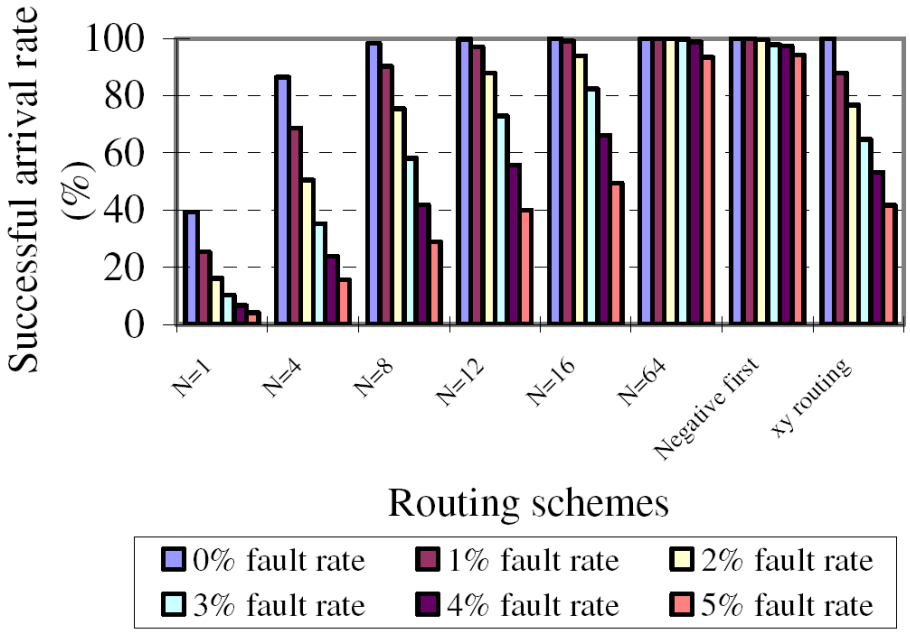


**Figura 16. Entregas de mensagens com sucesso nos diferentes algoritmos de roteamento [ZHU07].**

Como mostra a Figura 16, o algoritmo *negative first* oferece melhor desempenho na presença de falhas se comparado ao algoritmo XY e ao algoritmo *N random walk* com N<64. Ainda possui o desempenho praticamente idêntico ao *N random walk* com N=64, sendo que no último tem-se um alto congestionamento da rede para taxas de injeção mais elevadas.

Outra métrica importante analisada em [ZHU07] diz respeito ao consumo de energia dos algoritmos de roteamento apresentados. Os resultados do consumo de energia dos diferentes algoritmos foram analisados através de simulações do *netlist* gerado pela ferramenta *Synopsys Prime Power*, utilizando-se a tecnologia de 90nm. A Figura 17 apresenta os resultados.

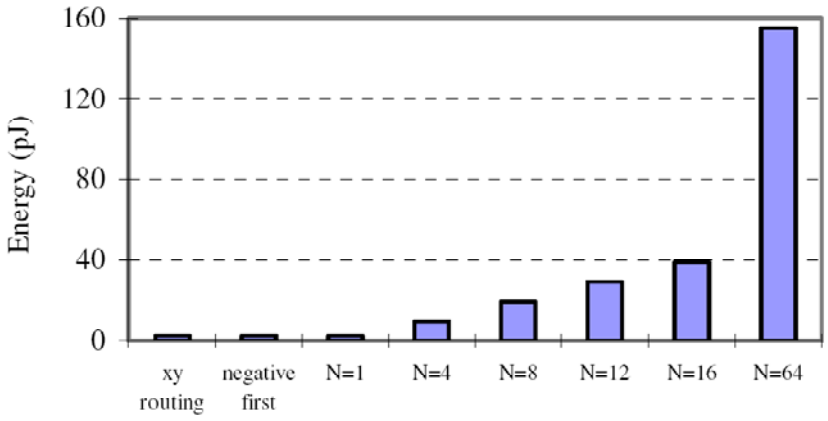


**Figura 17. Consumo de energia dos diferentes algoritmo de roteamento.**

Como pode ser observado os algoritmos *negative first* e *N random walk* com N=1 consomem praticamente a mesma energia do algoritmo XY. O aumento de N gera um crescimento exponencial no consumo de energia do algoritmo *N random walk*. Mesmo sustentando entregas de pacotes com sucesso a altas taxas de erro, o custo de energia deste algoritmo com N=64 praticamente proíbe seu uso na maioria das aplicações.

Por último, o resultado do consumo de área destes algoritmos foi analisado. Para se obter os dados de área, os algoritmos foram desenvolvidos e sintetizados utilizando-se a biblioteca padrão de 90nm da ferramenta *Synopsys Design Compiler*. A Tabela 4 mostra a área ocupada por cada um dos algoritmos.

**Tabela 4. Consumo de área dos algoritmos de roteamento.**

| Algoritmo de roteamento | Área (Nº de portas NAND com 2 entradas) |
|---|---|
| Roteamento XY | 250 |
| N=1 *random walk* | 11160 |
| *Negative First* | 333 |

A Tabela 4 evidencia que o algoritmo *negative first* provê pequeno acréscimo de área se comparado ao algoritmo XY, enquanto que o algoritmo *N random walk* consome muito mais área que os outros 2.

Kohler et al. [KOH09] propõem uma rede com tolerância a falhas transientes e permanentes através de um algoritmo adaptativo com deflexão, utilizando-se informações estatísticas a respeito da ocorrência de erros em enlaces e roteadores. Os pacotes que circulam nesta rede são codificados através de CRC e os roteadores contêm unidades de detecção de erros. Baseado nas informações das unidades detectores de erros é realizado um diagnóstico distribuído do estado de cada roteador e de suas conexões com seus vizinhos. Os resultados deste diagnóstico são armazenados em estruturas intra-chip que contém informações detalhadas sobre o modelo de

falhas encontrado. Desta maneira, utiliza-se um algoritmo adaptativo deflexivo para evitar partes da rede com defeito e ainda assim continuar oferecendo a mesma funcionalidade para o usuário. Entretanto, pode ocorrer uma degradação no desempenho da rede no caso de um grande número de componentes falhos.

O modelo de falhas aplicado diferencia erros que ocorrem durante a transmissão dos pacotes entre roteadores adjacentes e erros que são introduzidos localmente pelo roteador durante o processo de chaveamento de dados. Para determinar a origem dos erros detectados, cada porta de entrada, com exceção da porta local, possui uma unidade de CRC combinacional. Durante a recepção de um pacote, bem como antes de seu envio para o roteador vizinho, é feita a verificação de paridade do dado pelas unidades de CRC responsáveis pela entrada e pela saída respectivamente. Erros produzidos internamente pelo crossbar são detectados quando o CRC na entrada é correspondente ao dado e não correspondente na saída. Um CRC não correspondente na entrada indica um erro de transmissão no enlace roteador-roteador. A Figura 18 apresenta a arquitetura do roteador utilizado nesta rede.

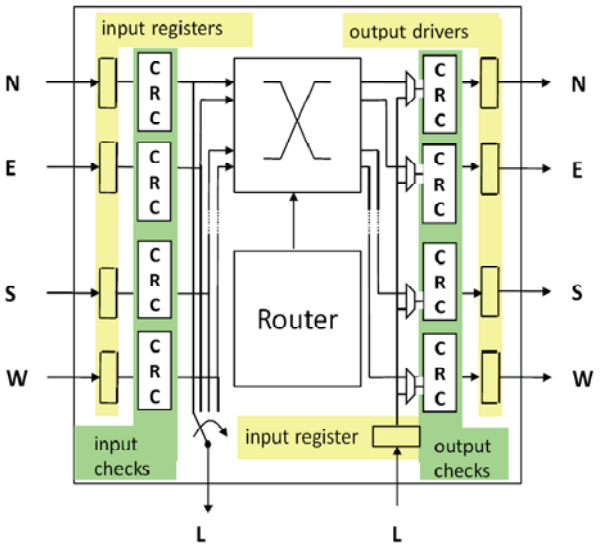


**Figura 18. Arquitetura do roteador com verificação de CRC [KOH09].**

Como resultado, este trabalho obteve uma rede bastante confiável, capaz de operar na presença tanto de falhas transientes como permanentes. Os resultados também mostram que utilizando-se esta técnica não há prejuízos em termos de latência para garantir tolerância a falhas, porém pacotes com erro não são corrigidos no momento da detecção através das unidades de CRC, apenas são utilizados para ativar as estruturas de contagem de falhas em canais ou conexões internas dos roteadores, e por sua vez decidir quais os caminhos ou roteadores que devem ser evitados no roteamento dos pacotes.

## 4.4 Métricas Utilizadas Para Avaliação

Todos os trabalhos estudados apresentam resultados baseados em algum tipo de métrica para avaliar o desempenho de suas implementações. Porém muitas vezes estas métricas divergem de um trabalho para outro. O artigo [GRE07] busca analisar as métricas mais relevantes para avaliar o desempenho de NoCs tolerantes a falha e apresentar resultados baseados nas métricas propostas.

As métricas são analisadas utilizando-se uma NoC 4x4 com topologia malha através de simulações no nível de ciclos de clock. As mensagens são injetadas na rede com a mesma probabilidade em todos os 16 núcleos conectados na rede. O roteamento XY foi utilizado nas simulações e cada mensagem é composta de 16 *flits*. As falhas são injetadas randomicamente e distrubuidas uniformemente entre todos os núcleos da rede. Os experimentos apresentados em [GRE07] pretendem observar principalmente as capacidades de detecção e recuperação de erros, levando em conta recuperação fim-a-fim (*e2e*), roteador-a-roteador (*s2s*) e recuperação híbrida com proteção interna no roteador (*ccd*).

Dois métodos de recuperação foram utilizados. No primeiro método, a prioridade das mensagens retransmitidas é a mesma das demais mensagens, denominando-se *epr* (*equal priority recovery*). O segundo método prioriza as mensagens retransmitidas para garantir baixa latência mesmo na ocorrência de uma falha. Este método é chamado de *pbr* (*priority based recovery*).

O primeiro teste realizado busca avaliar a latência na detecção de um erro para os diferentes esquemas de recuperação e é ilustrado na Figura 19.

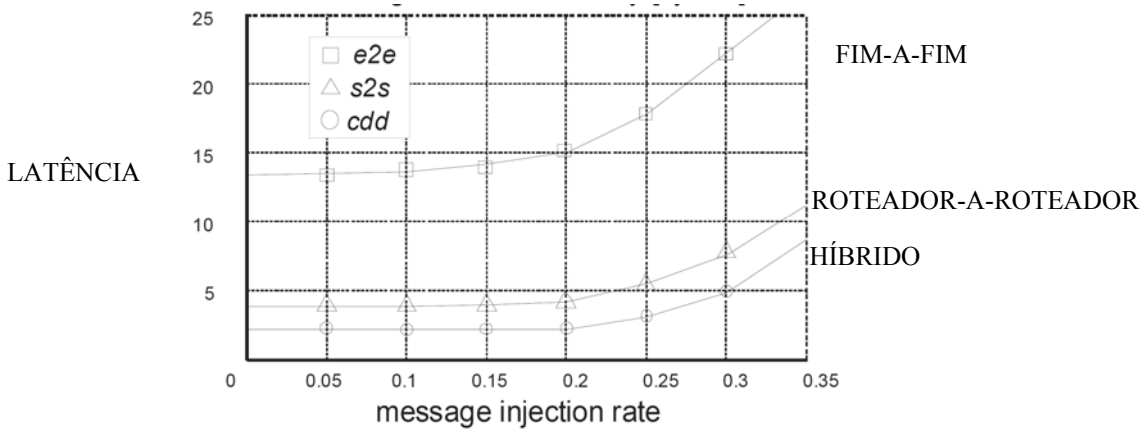


**Figura 19. Latência observada com diferentes métodos de recuperação de erros [GRE07].**

Para esta simulação, a probabilidade de erro utilizada é de $e=10^{-10}$. Como pode-se observar a detecção fim-a-fim possui a pior latência na detecção de um erro, uma vez que as mensagens devem atravessar a rede da origem até o destino para que o erro seja detectado. O mecanismo de

recuperação híbrido possui melhor desempenho porque faz a detecção em cada nodo que os *flits* atravessam.

O segundo experimento buscou medir a latência na recuperação de um erro para os três métodos de recuperação, utilizando retransmissão com e sem prioridade. Os resultados deste experimento são apresentados na Figura 20.

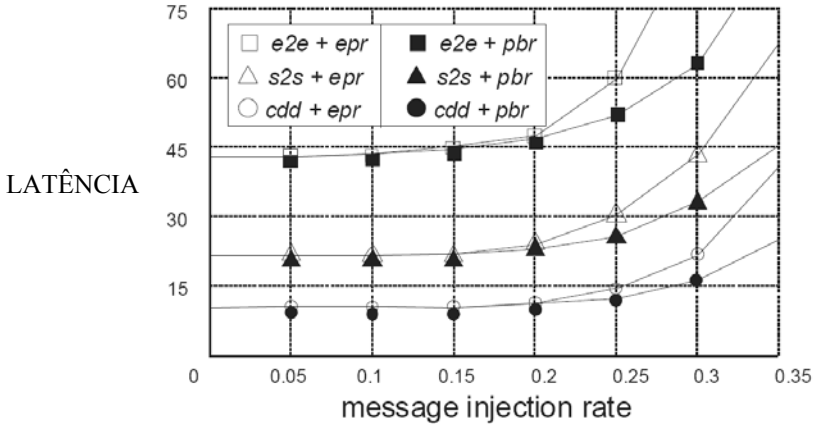


**Figura 20. Latência para recuperação de erros [GRE07].**

A Figura 20 mostra que a retransmissão com prioridade possui menor latência em relação ao método usual de recuperação de erros, principalmente quando a taxa de injeção é maior. Dessa maneira o método de retransmissão com prioridade pode ser mais vantajoso para aplicações que necessitem de qualidade de serviço.

A última métrica apresentada em [GRE07] buscou avaliar a latência de recuperação de erros variando a probabilidade de entrega das mensagens com sucesso.

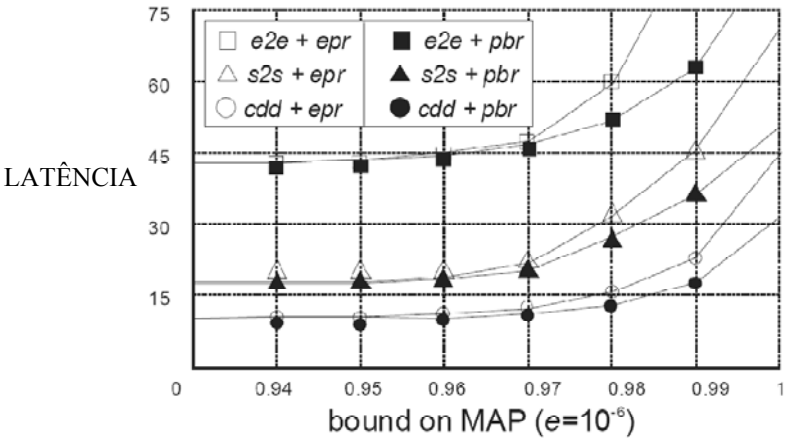


**Figura 21. Latência para recuperação de erros com probabilidade de entrega com sucesso variável [GRE07].**

A Figura 21 apresenta as latências observadas fixando-se a taxa de entrega das mensagens sem erros (MAP) com uma probabilidade de falhas de $10^{-6}$.Podemos observar que à medida que

probabilidade de entrega se aproxima de 1 (eixo X), ou seja, entrega com sucesso garantida, a latência das mensagens (eixo Y) aumenta significativamente, de modo que métodos mais avançados de tolerância a falhas devem ser empregados.

Por fim, as métricas apresentadas em [GRE07] buscam separar as características de desempenho da implementação de uma rede com tolerância a falhas das questões de consumo de potência, vazão e área adicional destes métodos. Porém, a escolha de um método de tolerância a falhas deve levar em conta a aplicação que será executada na rede e em qual contexto esta aplicação está inserida, para decidir qual das métricas mais importantes que devem ser atendidas para viabilizar bom desempenho e tolerância a falhas.

## 4.5 Resumo dos Trabalhos Relacionados

A Tabela 5 resume as principais características dos trabalhos avaliados, posicionando-os em relação ao trabalho desenvolvido na presente Dissertação. O foco do trabalho é em codificação, através de implementação no nível RTL, avaliando-se custo de latência, área, potência consumida e taxa residual de defeitos, e o tipo de tolerância é para falhas transientes.

**Tabela 5. Tabela de resumo dos trabalhos relacionados a tolerância a falhas em NoCs.**

| Autores | Método | Implementação/Avaliação | Métricas | Tipos de falhas toleradas |
|---|---|---|---|---|
| **MAR03** | -Comunicação estocástica | Implementação | -Latência<br>-Potência | Permanentes e transientes |
| **ZIM03** | -CRC/Hamming nos enlaces<br>-Modelo de falhas<br>-QoS | Implementação | -Taxa de de defeitos residuais | Transientes |
| **BER04** | -CRC na origem<br>-Reenvio roteador-a-roteador | Implementação | Não apresenta | Transientes |
| **VEL04** | -Paridade/Hamming nos enlaces<br>-QoS | Implementação | -Latência<br>-Potência | Transientes |
| **MUR05** | -Reenvio fim-a-fim<br>-Reenvio roteador-a-roteador:<br>-Nível de flit<br>-Nível de pacote<br>-Correção + detecção | Avaliação | -Latência<br>-Potência<br>-Taxa de defeitos residuais | Transientes |
| **PUL05** | -Controle de Fluxo | Avaliação | -Suporte a tolerância a falhas<br>-Latência<br>-Potência | Transientes |
| **MUR07** | -Roteamento Adaptativo | Implementação | -Área<br>-Potência<br>-Latência | Permanentes e Transientes |
| **ZHU07** | -Roteamento Adaptativo | Avaliação | -Área<br>-Vazão<br>-Potência | Permanentes |
| **GRE07** | -Reenvio fim-a-fim<br>-Reenvio roteador-a-roteador:<br>-Nível de flit<br>-Nível de pacote<br>-Reenvio com e sem prioridade | Avaliação | -Taxa de entrega de mensagens com sucesso<br>-Tempo médio de detecção<br>-Tempo médio de correção | Transientes |
| **KOH09** | -CRC na origem<br>-Decodificadores na entrada e na saída<br>-Roteamento adaptativo | Implementação | -Taxa de entrega de mensagens com sucesso.<br>-Área | Permanentes e transientes |
| **Trabalho proposto** | - CRC/Hamming nos enlaces<br>- CRC/Hamming na origem<br>- Retransmissão rotador a roteador | Implementação e avaliação | - Latência<br>- Área<br>- Potência<br>- Taxa residual de defeitos | Transiente |

A partir desta tabela, podemos afirmar que o presente trabalho buscou fazer uma exploração de métodos que já se mostravam eficazes quando utilizados para tolerância a falhas em NoCs, implementando-os em uma rede funcional com precisão de ciclo de relógio e avaliando-os sobre as métricas mais relevantes que foram observadas.

A contribuição deste trabalho se encontra na implementação de diversos métodos para a confiabilidade de redes intra-chip, avaliando-os sob diversos aspectos, levando em conta principalmente o custo para a implementação e a análise de trafego e latência sob os métodos implementados, dando boas noções aos projetistas de quais métodos podem melhor atender os requisitos de um determinado projeto.

# 5 ARQUITETURAS DE NOCS CONFIÁVEIS

Baseando-se nos trabalhos revisados, desenvolveu-se 4 métodos de tolerância a falhas tendo por base a rede HERMES [MOR04]. Estes 4 métodos foram escolhidos devido aos bons resultados observados em comparação com outras técnicas e também por nenhum trabalho apresentar uma comparação direta entre eles. O objetivo das arquiteturas desenvolvidas é apresentar baixa sobrecarga de área e potência em relação à rede original, aliado a uma baixa latência de recuperação, de maneira que os circuitos desenvolvidos não interfiram significativamente no desempenho original da rede.

Este Capítulo corresponde à *principal contribuição* da Dissertação, a implementação das arquiteturas tolerantes a falhas. O restante desta Seção apresenta os módulos de hardware desenvolvidos para adicionar controle de erros à arquitetura alvo bem como as estratégias adotadas para a geração de 4 métodos de tolerância a falhas para a rede Hermes.

## 5.1 HERMES

A rede HERMES é uma infra-estrutura parametrizável, especificada em VHDL no nível RTL. Os roteadores possuem lógica de roteamento centralizada e cinco portas de comunicação bidirecionais. Uma porta é utilizada para estabelecer a conexão entre o roteador e seu módulo de processamento local, enquanto as outras portas são conectadas a roteadores vizinhos. Cada porta de entrada armazena os dados recebidos em uma memória organizada em forma de fila. O algoritmo de roteamento escolhido para este trabalho é o XY. Múltiplos pacotes podem chegar simultaneamente em um determinado roteador. Um algoritmo de arbitragem centralizado do tipo round-robin é utilizado para que os pacotes de entrada tenham acesso ao roteamento. Se a requisição de um pacote é atendida, o algoritmo XY é executado e conecta a porta de entrada à porta de saída correspondente. Se o algoritmo informar que a porta de saída está ocupada, o cabeçalho e todos os demais *flits* do pacote ficam bloqueados até a liberação da mesma.

A arquitetura básica da rede utilizada para a implementação das técnicas de tolerância a falhas aqui apresentadas possui 8x8 roteadores em uma rede malha bidimensional, o tamanho do *flit* é de 16 bits de dados, filas de 8 posições e não possui canais virtuais. Conforme visto no Capítulo 2, a arquitetura malha possui diversas vantagens para o projeto de redes intra-chip, tais como área dos roteadores, frequência de operação e também exploração de algoritmos de roteamento. Optou-se por utilizar uma rede relativamente grande, com 64 roteadores, para possibilitar a criação de tráfegos realistas e poder avaliar a rede sobre diversos aspectos através

destes tráfegos. *Flits* de 16 bits são utilizados devido às diversas aplicações que já funcionam sobre a rede Hermes utilizarem esse padrão. Por fim, buffers de 8 posições são utilizados devido ao tempo de processamento dos cabeçalhos nos roteadores da rede Hermes, sendo o tamanho mínimo para garantir vazão máxima se a rede não estiver congestionada.

## 5.2 Codificação

Codificação de dados prove um mecanismo eficiente e independente de tecnologia para aumentar a confiabilidade em aplicações de comunicação em geral. Existem diversos códigos capazes de detectar e corrigir erros e eles são classificados em dois grandes grupos: códigos de bloco e códigos convolucionais.

Em códigos de bloco, um bloco é formado por *k* símbolos de carga útil e *n-k* símbolos de paridade para formar um bloco de código denotado por (*n*,*k*). Os símbolos de paridade são algebricamente relacionados com os *k* símbolos de carga útil.

Já os códigos convolucionais são gerados a partir da convolução discreta no tempo da sequência de dados de entrada através da resposta ao impulso do codificador. Enquanto codificadores de bloco aceitam apenas entradas com *K* símbolos, um codificador convolucional aceita sequências contínuas de entrada.

No contexto de redes intra-chip existe um interesse maior em códigos de bloco lineares devido a certas propriedades oferecidas por este tipo de codificação. Para o controle de erro na camada de enlace de redes intra-chip, o atraso inserido pelos codificadores e decodificadores são somados ao atraso inserido pelos canais que interligam os roteadores, limitando a freqüência máxima de operação da rede. Portanto, os módulos de codificação e de codificação devem ter o menor caminho crítico possível para não degradar o desempenho geral da NoC. Além disso, a preocupação com o consumo de potência no projeto de NoCs conduz à adoção de métodos de controle de erro que consumam o mínimo necessário para atingir a confiabilidade requerida.

No contexto deste trabalho são utilizados dois tipos de códigos de bloco lineares para proteger a rede contra falhas transientes: códigos de *Hamming* e CRC. As sessões a seguir descrevem o funcionamento destes códigos e como eles foram aplicados na NoC de referência.

### 5.2.1 Circuito CRC

CRC (*Cyclic Redundancy Check*) pertence à classe de códigos cíclicos lineares, possuindo a propriedade de que qualquer palavra código que for deslocada de maneira circular também

resultará em uma palavra código. CRC tem a vantagem de consumir poucos recursos de hardware, tanto que o decodificador tem a mesma complexidade do codificador, e possui a limitação de apenas detectar erros e não corrigi-los.

O circuito CRC utilizado neste trabalho codifica palavras de 16 bits a cada ciclo de relógio. Utilizou-se 4 bits de paridade para cada palavra de 16 bits para garantir cobertura de múltiplos bits com erro.

O polinômio gerador utilizado é $g = 1+X+X^4$. A escolha deste polinômio se deve ao fato de ser o polinômio primitivo de grau 4 com menor número de coeficientes não nulos. Polinômios primitivos são necessários para gerar CRCs únicos para determinados conjuntos de valores, e um baixo número de coeficientes não nulos normalmente facilita sua implementação em hardware, consumindo menos recursos. Os 4 bits de paridade de cada palavra são obtidos pelo resto da divisão polinomial entre a palavra e o polinômio gerador. O circuito apresentado na Figura 22 realiza esta operação em 16 ciclos de relógio, pois recebe apenas um bit de entrada a cada ciclo de relógio. Os 4 bits de paridade ficam armazenados nos registradores (FF) no final desta operação.

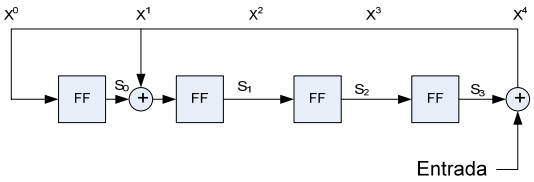


**Figura 22. Codificador CRC para o polinômio g = 1+X+X4.**

Este circuito consome poucos recursos de hardware, porém não é capaz de realizar a codificação na velocidade requerida para a transmissão de um *flit* codificado a cada ciclo de relógio. Um método bastante simples de paralelização deste circuito é a utilização do circuito o apresentado na Figura 23 [SPR01]. Este circuito consiste na ligação de 16 blocos combinacionais com 4 entradas e 4 saídas para gerar os 4 bits de paridade paralelamente.

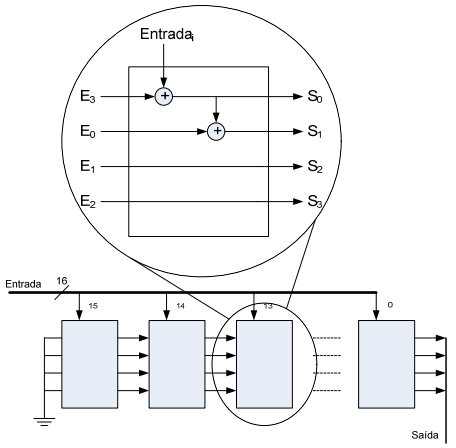


**Figura 23. Codificador CRC paralelo.**

Este circuito gera os 4 bits de paridade de maneira puramente combinacional, realizando a mesma operação do circuito apresentado na Figura 22, sem a necessidade de processamento em 16 ciclos de relógio. Porém, ao conectar 16 destes blocos em série, existe um grande caminho crítico que pode levar o circuito a não operar na freqüência de relógio desejada. A partir da simplificação da equação combinacional obtida através da ligação de 16 blocos, como apresentado na Figura 23, é possível obter uma equação lógica simples para cada bit de paridade, onde o atraso de propagação é mínimo. As equações encontradas para o polinômio g = 1+X+$X^4$ e palavras de 16 bits estão apresentadas a seguir:

$$S_0 = E_{15} \oplus E_{11} \oplus E_8 \oplus E_7 \oplus E_5 \oplus E_3 \oplus E_2 \oplus E_1 \oplus E_0$$
$$S_1 = E_{12} \oplus E_9 \oplus E_8 \oplus E_6 \oplus E_4 \oplus E_3 \oplus E_2 \oplus E_1$$
$$S_2 = E_{13} \oplus E_{10} \oplus E_9 \oplus E_7 \oplus E_5 \oplus E_4 \oplus E_3 \oplus E_2$$
$$S_3 = E_{15} \oplus E_{14} \oplus E_{10} \oplus E_7 \oplus E_6 \oplus E_4 \oplus E_2 \oplus E_1 \oplus E_0$$

Desta maneira, o codificador CRC utilizado neste trabalho consiste de 4 portas XOR de 8 ou 9 entradas. O decodificador utiliza o circuito do codificador, e compara os bits de paridade recebidos com os bits calculados localmente, se os valores forem diferentes ele sinaliza a presença de um erro.

Utilizando-se a estratégia de 4 bits de paridade para 16 de informação útil, garante-se uma cobertura de 93.75% dos possíveis padrões de erro que podem ser inseridos em cada bloco de 20 bits, cobrindo todos os padrões de erro com inversão de até 3 bits e também de erros em bits sequenciais.

### 5.2.2 Hamming

Códigos de *Hamming* foram a primeira classe de códigos lineares desenvolvidos para correção de erros e são largamente utilizados para controle de erros em aplicações de comunicação digital e em sistemas de armazenamento de dados. Tem a vantagem de ser muito simples de decodificar, porém são capazes de corrigir apenas um erro por bloco.

Para o desenvolvimento do codificador e do decodificador de *Hamming* utilizou-se a função *hammgen()* do sotware Matlab [MAT09]. É importante observar que pela teoria do código de *Hamming*, para proteger uma palavra de 16 bits, é necessário no mínimo 5 bits de paridade, pois as seguintes fórmulas são utilizadas para estruturar um código:

- $n = 2^h - 1$
- $k = n - h$

Onde *h* é o número de bits de paridade do código, *n* é o tamanho total do código e *k* é o número de bits de informação útil que o código possui. Desta maneira, com 4 bits de paridade

(*h*=4) é possível de se obter um código de tamanho máximo 15 (*n*=15) e apenas 11 bits de informação (*k*=11), sendo este código denotado por (15,11). Com 5 bits de paridade se obtém o código (31,26), ou seja, 26 bits de informação para 5 bits de paridade. Como a rede que estamos trabalhando possui *flits* de 16 bits, utilizamos o código (31,26) simplesmente descartando-se os 10 bits adicionais de informação, ou seja, considerando para fins de cálculo e verificação de paridade que estes bits permanecem com valor lógico zero.

A partir da aplicação da função *hammgen()* do Matlab, obtemos as duas matrizes utilizadas no processamento do código de *Hamming*. A matriz geradora do código, utilizada no processo de codificação é chamada de matriz *G*, enquanto a matriz de verificação, utilizada no processo de decodificação é chamada de matriz *H*. Em um processo de codificação, multiplica-se a palavra que se deseja codificar pela matriz *G*:

- $c = aG$

Onde *c* é a palavra codificada e *a* é a palavra que se deseja codificar. No processo de decodificação inicialmente calcula-se a síndrome através da multiplicação da palavra recebida pela matriz *H* transposta. Logo após realiza-se a relação da síndrome com o possível padrão de erro inserido na palavra recebida. Se a palavra recebida não tiver erros a síndrome calculada é igual a zero. Logo:

- $s = rH^T$
- $c' = r + padrão(s)$

Onde *r* é a palavra recebida, *s* a síndrome e *c'* é igual à palavra enviada se *r* tiver apenas um ou nenhum erro.

A seguir é apresentada a matriz utilizada para a geração dos 5 bits de paridade a partir dos 16 bits de informação fornecidos:

$$G = I_{16x16}\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Para fins de simplificação não representamos a matriz identidade que também faz parte da matriz geradora. Porém, apenas as últimas 5 colunas da matriz geradora interessam para o desenvolvimento do hardware que realiza a codificação das palavras. Por exemplo, o primeiro bit de paridade deste código é a soma de todos os bits da palavra em que a primeira coluna apresenta o valor 1, sendo a soma representada pela operação XOR, ou seja:

$$p_0 = r_{15} \oplus r_{12} \oplus r_{10} \oplus r_9 \oplus r_6 \oplus r_5 \oplus r_4 \oplus r_3 \oplus r_2$$
$$p_1 = r_{14} \oplus r_{11} \oplus r_9 \oplus r_8 \oplus r_5 \oplus r_4 \oplus r_3 \oplus r_2 \oplus r_1$$
$$p_2 = r_{15} \oplus r_{13} \oplus r_{12} \oplus r_9 \oplus r_8 \oplus r_7 \oplus r_6 \oplus r_5 \oplus r_1 \oplus r_0$$
$$p_3 = r_{14} \oplus r_{12} \oplus r_{11} \oplus r_8 \oplus r_7 \oplus r_6 \oplus r_5 \oplus r_4 \oplus r_0$$
$$p_4 = r_{13} \oplus r_{11} \oplus r_{10} \oplus r_7 \oplus r_6 \oplus r_5 \oplus r_4 \oplus r_3$$

Como podemos observar o codificador de *Hamming* é bastante similar ao codificador CRC em termos de hardware, porém suas características matemáticas são bastante distintas.

Para a etapa de decodificação utilizou-se a matriz *H* transposta apresentada a seguir:

$$H^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Multiplicando-se a palavra recebida pela matriz $H^T$, através da mesma operação apresentada na codificação, se obtém os 5 bits de síndrome da palavra recebida. A partir desta síndrome, é possível identificar qual bit deve ser alterado na palavra recebida para a correção de um erro, sendo que este bit corresponde à linha da matriz $H^T$ onde a síndrome se encontra. Por exemplo, se a síndrome encontrada for '10100', podemos observar que este valor se encontra na sexta linha da matriz $H^T$, de maneira que para corrigir o dado recebido deve-se inverter o sexto bit do mesmo.

## 5.3 NoC com CRC nos Enlaces

A primeira rede com tolerância a falhas desenvolvida no contexto deste trabalho possui codificação CRC apenas nos enlaces, de maneira que o tamanho dos buffers e a largura dos

barramentos internos ao roteador não é alterada, apenas sua interface externa recebe novos sinais para a recuperação de erros. Nesta rede, o codificador CRC é adicionado em cada porta de saída dos roteadores, e o decodificador é inserido em cada porta de entrada. Esta estratégia provê reenvio roteador-a-roteador em nível de *flit* em caso de falhas possuindo as seguintes vantagens:

- Não é necessário armazenar pacotes completos em cada roteador, possibilitando o uso de chaveamento por pacotes (*Wormhole*), reduzindo área, potência e latência se comparado a métodos como *store-and-forward* e *virtual cut-through*.
- Este método é mais rápido se comparado a retransmissão de pacotes completos, pois uma vez que o erro é detectado o *flit* com erro pode ser retransmitido no próximo ciclo de clock.
- A detecção de erro ocorre antes da decisão de roteamento, protegendo a rede contra erros de roteamento devido a erros em cabeçalhos.
- *Flits* a serem retransmitidos estão disponíveis nas filas de entrada dos roteadores, gerando baixo acréscimo de área.

A Figura 24 ilustra a interface entre uma porta de entrada com uma porta de saída de dois roteadores, ilustrando os sinais adicionados para a implementação desta técnica de correção de erros.

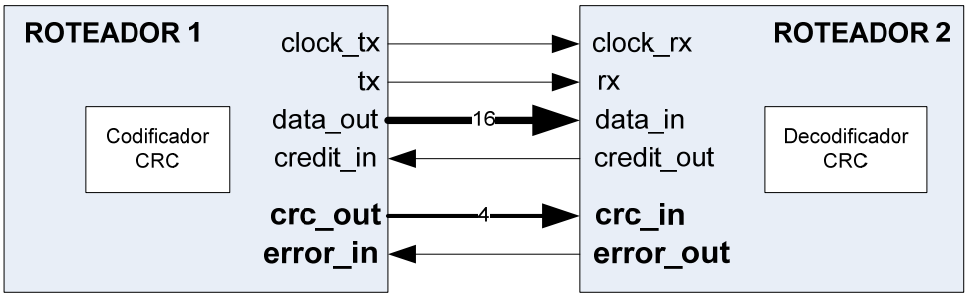


**Figura 24. Interface entre uma porta de saída e uma porta de entrada dos roteadores, com adição dos sinais *crc_in-crc_out* e *error_in-error_out*.**

É importante observar que as portas dos roteadores são bidirecionais, ou seja, esta interface se repete no sentido inverso para cada porta de um roteador. O sinal *clock_tx* indica a taxa de amostragem do dado contido em *data_out* e o sinal *tx* indica a sua validade. O roteador 2 informa que possui espaço em seu buffer para receber novos dados a partir do sinal *credit_out*. O sinal *crc_out* é calculado no roteador 1 de acordo com o dado contido em *data_out*. Ao receber o dado pela porta *data_in* o roteador 2 calcula o CRC para este dado e compara com o sinal recebido em *crc_in*. Caso o valor do CRC calculado seja diferente do valor recebido o roteador 2 sinaliza a

presença de um erro através do sinal *error_out*. É importante observar que os sinais *crc_out* e *error_in* não foram adicionados na interface entre o roteador e a porta local, pois estas conexões são curtas, não havendo efeito significativo de *crosstalk* nestas linhas.

Para evitar que um dado seja escrito com erro no buffer de entrada do roteador de destino, foi adicionada a porta *error* no buffer. Esta porta é ligada diretamente na saída *error* do decoder. Quando este sinal indicar um erro no dado recebido, o buffer não consome o dado recebido em *data_in*, garantindo a integridade dos dados escritos no buffer. O mesmo sinal de erro é amostrado por um registrador e enviado para o roteador origem para requisitar o reenvio do dado. A Figura 25 mostra um diagrama desta operação.

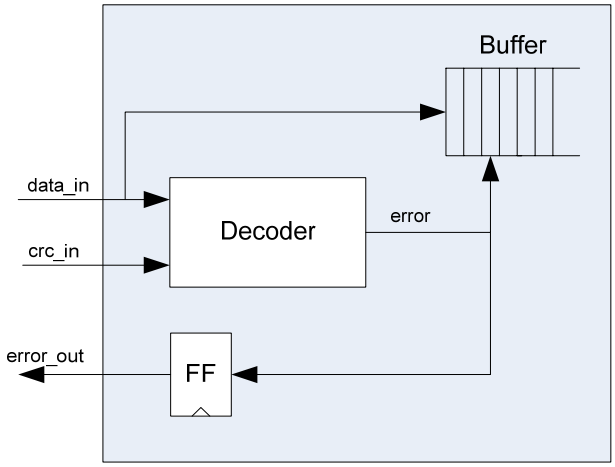


**Figura 25. Diagrama do consumo de dados pelo buffer destino.**

Este circuito garante que um dado com erro seja recuperado em um ciclo de relógio, pois um ciclo após a detecção do erro, o sinal *error_out* passa a ser ‘1’. O roteador que originou o dado corrompido ao receber este sinal disponibiliza novamente este dado em sua porta de saída, e ele é recebido mais uma vez pelo sinal *data_in* no roteador destino. Uma vez que o sinal de *error* tenha seu valor lógico ‘0’ o *buffer* pode consumir este dado.

O roteador que recebe um sinal de erro pela porta leste, por exemplo, como solicitação de reenvio de um *flit*, deve encaminhar este sinal de erro para o buffer que está enviando dados para esta porta. Para isso, este sinal de erro deve ser roteado através do crossbar interno do roteador até o buffer de entrada que está enviando dados pela leste. A Figura 26 ilustra esta operação. Supondo que o buffer que esteja enviando dados para a porta leste esteja na porta norte, o sinal *error_out(NORTH)* passa a receber o sinal *error_in* da porta leste. Dessa maneira o dado com erro é retirado do próprio buffer de origem, sem a necessidade de buffers de saída.

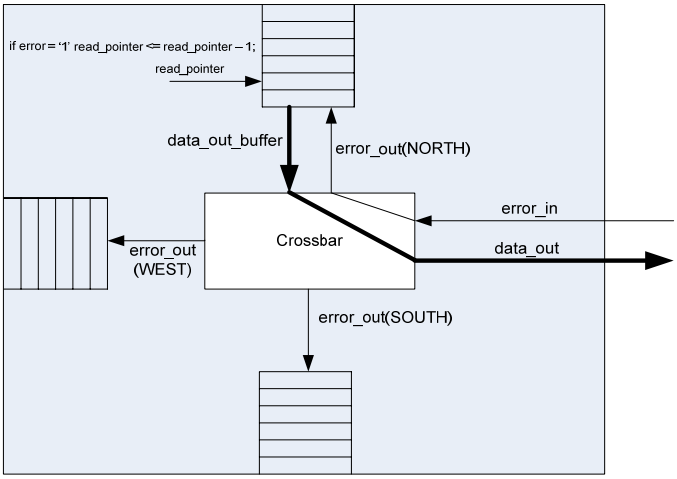


**Figura 26. Ilustração da recuperação através da busca no buffer de origem.**

Durante o recebimento do sinal de erro, um buffer continua recebendo dados de sua origem enquanto não estiver cheio, porém um pedido de reenvio atrasa em um ciclo o consumo de dados neste buffer, podendo facilitar o esgotamento do buffer. Caso isso aconteça, este buffer avisa sua origem que está cheio através do sinal de crédito, fazendo com que a origem pare de enviar dados até que algum dado seja consumido.

## 5.4 NoC com CRC na Origem

Baseando-se na rede apresentada anteriormente, com CRC nos enlaces, partiu-se para a implementação de uma rede com CRC na origem, com o objetivo de economizar área de codificadores ao longo da rede. Nesta abordagem o CRC é calculado na interface de rede (NI) do módulo conectado a porta local do roteador que origina os dados. Desta maneira os bits de CRC passam a fazer parte do *flit*, sendo necessário um aumento da largura de todos os buffers da rede, bem como dos barramentos internos dos roteadores, para que essa informação trafegue pela rede como parte do pacote. Sendo assim, a verificação de CRC é executada em cada porta de entrada conforme na rede anterior, porém o processo de codificação não se torna mais necessário nas portas de saída. Esta rede tem a vantagem de possuir 4 módulos de CRC a menos em cada roteador, representando redução de lógica combinacional, porém o aumento dos *buffers* acaba gerando um aumento da área sequencial. Este compromisso de aumento da área dos *buffers* para o corte de lógica combinacional é analisado no Capítulo 6.

A Figura 27 apresenta um esquemático do funcionamento geral desta rede.

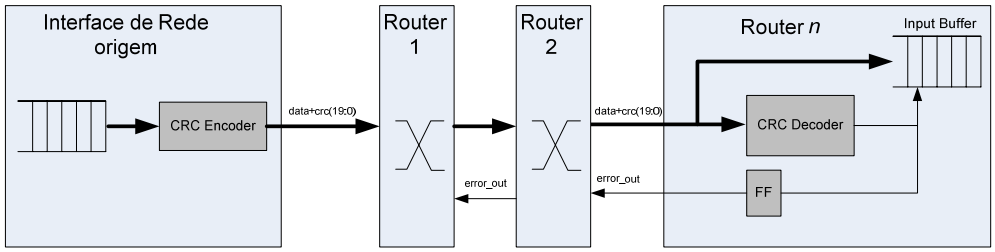


**Figura 27. Representação de enlaces da rede com CRC na origem.**

Como podemos observar o sinal *data_out* possui agora 20 bits ao invés de 16 bits como mostrado na rede anterior, pois nesta rede este sinal contem o sinal *crc_out* apresentado anteriormente.

É importante observar que caso um erro seja detectado pelas unidades de CRC, o mesmo mecanismo de escrita no buffer local, bem como o mesmo mecanismo busca no *buffer* do roteador origem, apresentado para a rede anterior é utilizado, garantindo o mesmo desempenho da rede anterior no caso de falhas.

Outro aspecto importante desta rede está na possibilidade de se usar as informações de CRC contidas no *flit* para diagnosticar falhas internas ao roteador. Dessa maneira, esta arquitetura fornece algum tipo de proteção interna nos roteadores, pois falhas transientes internas ao roteador são recuperadas da mesma maneira que falhas nos enlaces, pois a informação redundante também está presente dentro dos roteadores. Falhas de controle não são detectadas nesta rede e falhas de memória podem ser detectadas, mas não corrigidas, de maneira que outras técnicas de tolerância a falhas devem ser adotadas para a solução deste tipo de problema.

## 5.5 NoC com Hamming nos Enlaces

A terceira rede desenvolvida no contexto deste trabalho consiste na proteção dos canais com códigos de *Hamming*, possibilitando a correção de *flits* que possuam no máximo um bit com erro. Neste caso, não foi utilizado nenhum mecanismo de reenvio, pois o código de *Hamming* não fornece nenhuma forma de identificar se houve erro em mais de um bit. Algumas implementações utilizam um bit adicional de paridade para reenvio em caso de erro em mais de um bit, porém mesmo utilizando-se esta estratégia, a cobertura de falhas permanece baixa.

A estrutura simplificada de um enlace da rede com *Hamming* nos enlaces é apresentada na Figura 28, lembrando que nesta arquitetura nenhuma das estruturas internas dos roteadores Foi alterada, ao contrário das arquiteturas anteriores.

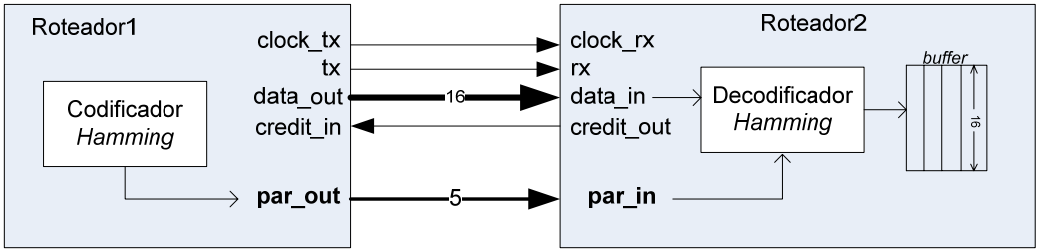


**Figura 28. Interface entre dois roteadores de uma rede com Hamming nos enlaces.**

O processo de decodificação adotado nesta arquitetura é transparente para o roteador, pois não consome nenhum ciclo de relógio. Uma vez que o dado chega pela porta de entrada, ele é imediatamente repassado para o *buffer* já decodificado. Dessa maneira o circuito responsável pela decodificação, não aumenta a latência em relação à rede original. Como a cobertura de falhas do código de *Hamming* é muito menor que a cobertura do CRC, a principal métrica para se avaliar esta arquitetura é a observação da taxa de defeitos residuais encontrada variando-se a taxa de injeção erros, comparando-se com a taxa obtida pela rede que utiliza CRC. Esta análise é apresentada na Seção 6.4.

## 5.6 NoC com Hamming na Origem

O objetivo da implementação de uma rede com *Hamming* na origem é a tolerância de falhas internas no roteador, bem como tolerância a SEUs nos buffers e correção de erros simples nos enlaces. Nesta arquitetura, os bits de paridade são calculados na interface de rede que origina os *flits*, e os mesmos são decodificados tanto na entrada quanto na saída de cada roteador, possibilitando a correção de falhas nos enlaces, e também de falhas internas e nos buffers no caso de correção na saída. A Figura 29 mostra um diagrama simplificado do funcionamento desta rede.

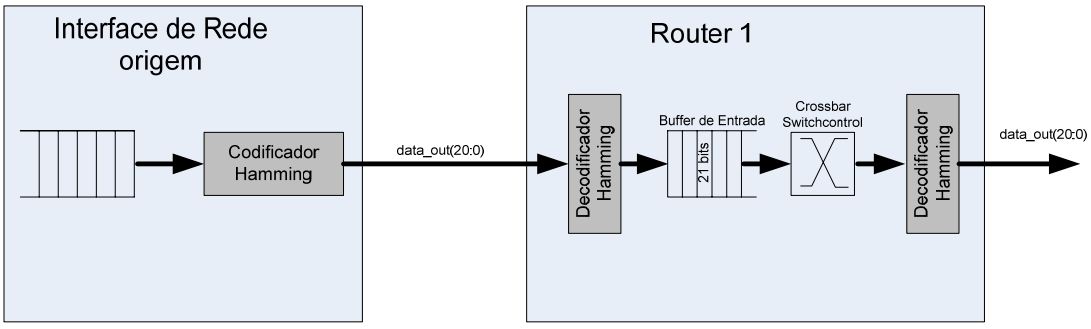


**Figura 29. Diagrama simplificado de uma rede com *Hamming* na origem.**

Como podemos observar, esta rede possui um custo de área muito grande, pois os buffers tiveram seu tamanho aumentado para 21 bits, e um roteador central contendo todas as portas, possui 8 decodificadores de *Hamming* no total, 4 nas portas de entrada e 4 nas portas de saída.

Entretanto, esta é a rede que oferece maior cobertura de falhas como um todo entre as apresentadas neste trabalho.

O único tipo de falha que ainda não é coberto nesta arquitetura são falhas de controle. Caso ocorra uma inversão de bit em uma das tabelas de roteamento da crossbar por exemplo, um pacote pode ser roteado para o caminho errado, o que pode causar um travamento na rede e a perda do serviço. As técnicas mais utilizadas para tratar esse tipo de problema, consistem em redundância completa de componentes, tais como TMR. Porém, estas técnicas estão além do escopo deste trabalho, cujo objetivo é tornar a rede mais confiável com o mínimo de área e potência adicional.

## 5.7 Plataforma ATLAS

ATLAS é uma plataforma desenvolvida originalmente para geração automática da rede Hermes através de parâmetros definidos pelo usuário, tais como dimensões da rede, tamanho dos buffers, largura dos *flits*, algoritmo de roteamento, número de canais virtuais entre outros. Uma das atividades desenvolvidas neste trabalho foi a adição das arquiteturas desenvolvidas na plataforma Atlas, para que estas arquiteturas também possam ser geradas automaticamente. Esta atividade facilitou a etapa de testes e geração de resultados.

A Figura 30 apresenta a interface da plataforma ATLAS para a geração das redes com tolerância a falhas. Como podemos observar, alguns parâmetros foram deixados fixos, como por exemplo a largura dos *flits*, pois a matemática envolvida nos codificadores e decodificadores é alterada com larguras diferentes de *flits*. As parametrizações possíveis para estas redes compreendem as dimensões da rede, o tamanho dos buffers, o tipo de rede (CRC nos enlaces, CRC na origem e *Hamming*) e também a parametrização do módulo *saboteur*, como podemos observar no canto inferior esquerdo da figura. O módulo *saboteur* é utilizado essencialmente para simular a rede e gerar estatísticas de falhas das simulações. Caso o objetivo final do usuário seja a síntese final da rede o módulo *saboteur* pode ser excluído. Também é possível parametrizar o módulo *saboteur* de acordo com as condições do modelo MAF, para controlar o número de falhas injetadas. Quatro tipos de injeção de falhas são permitidas: *Dr*, atraso de subida; *Df,* atraso de descida; *Gn*, *glitch* negativo; *Gp, glitch* positivo (ver seção 3.3.3).

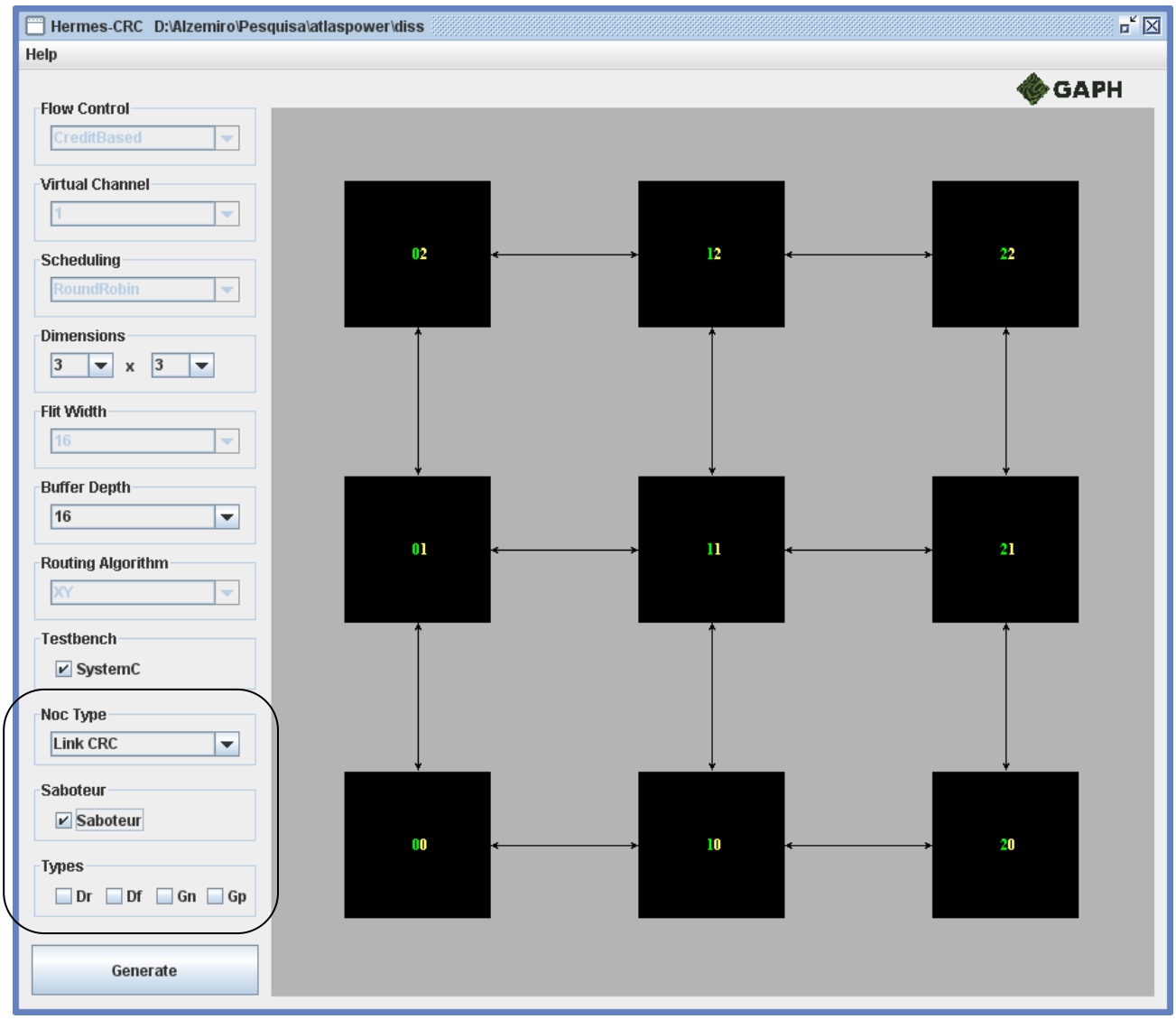


**Figura 30. Interface da plataforma ATLAS para a geração das redes com tolerância a falhas.**

# 6 RESULTADOS

Esta Seção apresenta os resultados deste trabalho, levando em conta as arquiteturas desenvolvidas. Observa-se principalmente o processo de recuperação de erros, ilustrado através de formas de onda do circuito desenvolvido, o acréscimo de área e impacto na frequência de operação, latência utilizando-se injeção automática de falhas, bem como potência consumida e análise de defeitos residuais.

## 6.1 Processo de Recuperação de Erros

O primeiro exemplo, apresentado na Figura 31, mostra os sinais da interface externa de dois roteadores envolvidos em um processo de recuperação utilizando CRC. O exemplo ilustra o envio de um pacote do roteador 01 para o roteador 00 com um erro simulado no terceiro *flit* deste pacote. O roteador 01 envia este pacote pela sua porta sul (índice 3) e o roteador 00 recebe este pacote pela porta norte (índice 2).

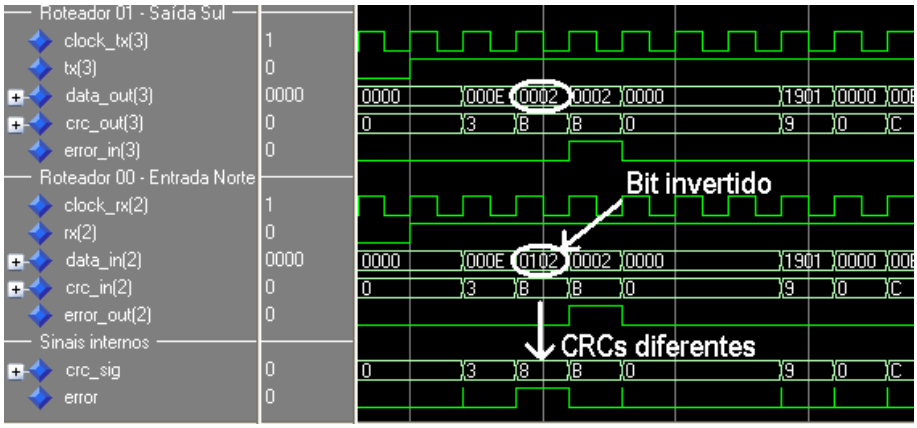


**Figura 31. Reenvio de um dado corrompido utilizando-se codificação CRC.**

Como podemos observar, o terceiro byte enviado pelo roteador 01 teve um bit invertido durante esta simulação. O roteador 00 recebe 0xB como o CRC correspondente ao *flit* recebido, e calcula internamente o CRC 0x8 através do *flit* recebido. Como o CRC recebido é diferente do CRC calculado, o sinal de erro interno fica um ciclo de relógio em '1' indicando que o dado recebido foi corrompido. No ciclo seguinte este sinal é enviado para o roteador 01 através da porta *error_out*. Ao receber o sinal de erro pela porta *error_in*, o roteador 01 envia novamente o dado anterior, neste caso 0x0002, e o mesmo é recebido corretamente pelo roteador 00.

O segundo exemplo, apresentado na Figura 32, apresenta o processo que ocorre no buffer do roteador que recebe uma solicitação de reenvio, neste caso o roteador 01.

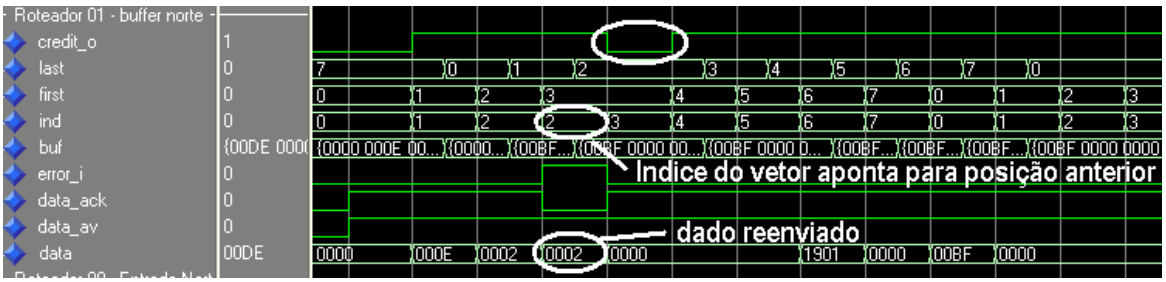


**Figura 32. Processo de busca do dado corrompido no buffer de origem.**

Esta figura mostra que ao receber um sinal de erro, vindo da porta que está enviando os dados para fora do roteador, o buffer retorna uma posição no seu ponteiro de leitura e envia o dado corrompido novamente. O sinal *first* indica a posição do buffer que contém um dado que ainda não foi consumido, sendo originalmente a posição de leitura do buffer, porém o sinal *ind* é utilizado como ponteiro de leitura do buffer e, ao detectar um erro, *ind* aponta para a posição anterior, que contém o dado que foi corrompido durante o processo de envio. Observa-se também que o sinal de crédito (*credit_in*) fica em 0 por algum tempo, indicando que o buffer não pode receber novos dados. Isso acontece porque o buffer, de apenas 8 posições, permanece normalmente cheio durante o processo de envio de um pacote, e o ciclo perdido durante um processo de reenvio faz com que o buffer fique um ciclo sem poder receber dados.

A Figura 33 ilustra o processo de correção de um erro utilizando código de *Hamming*. Este exemplo apresenta o roteador 10 enviando dados através de sua porta oeste para o roteador 00, que recebe estes dados na porta leste.

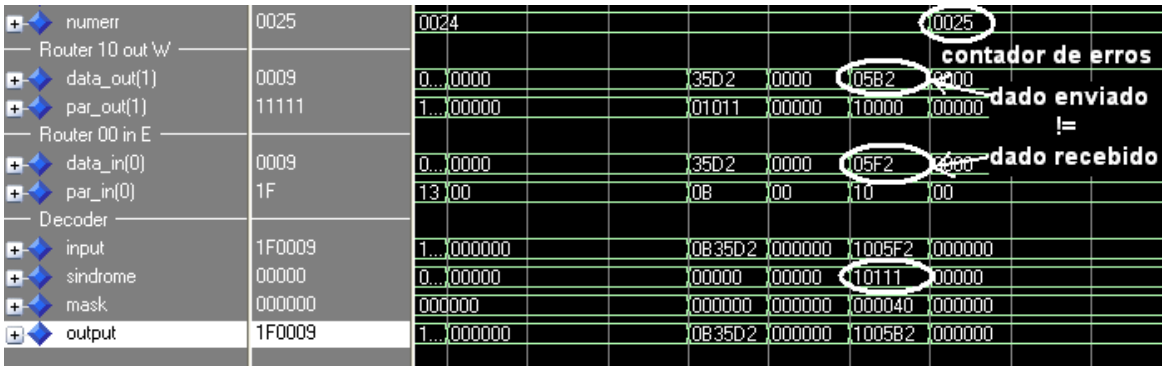


**Figura 33. Correção de um dado corrompido através de código de *Hamming*.**

O primeiro detalhe importante de se observar nesta figura é o contador de erros, inserido no saboteur, que avança de 24 para 25, indicando que no ciclo de clock anterior um erro foi inserido. Portanto podemos observar que o sinal *data_out(1)* do roteador 10 não corresponde ao sinal *data_in(0)* do roteador 00. Neste caso o decodificador recebe como dado de entrada os bits de

paridade, concatenados com o *data_in(0)* recebido, e logo calcula o valor da síndrome, que como podemos observar também em destaque na figura, é diferente de zero, indicando a presença de um erro do dado recebido. De acordo com essa síndrome é gerado uma máscara para corrigir o dado recebido. Na última linha da figura podemos observar a saída do decoder, que é o dado efetivamente consumido pelo buffer de entrada do roteador. Como podemos observar, os 4 digitos finais deste sinal correspondem ao dado enviado pelo roteador 10, indicando uma correção de erro bem sucedida.

O último exemplo, apresentado na Figura 34, busca ilustrar a correção de uma falha no buffer de entrada da rede (proteção interna) através da arquitetura com *Hamming* na origem.

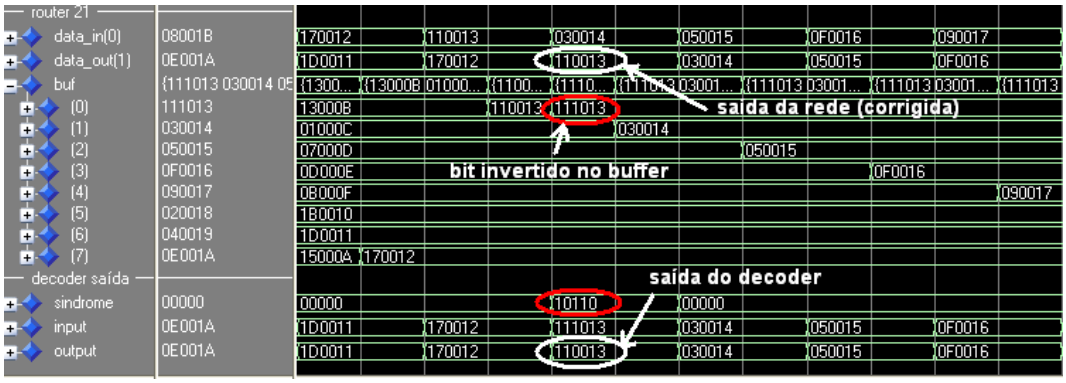


**Figura 34. Correção de uma falha no buffer de entrada com *Hamming* na origem.**

A figura ilustra o roteador 21 recebendo dados pela porta leste, *data_in(0)*, e enviando dados pela porta oeste, *data_out(1)*. Nesta figura é possível ver as 8 posições do buffer de entrada da rede, onde são armazenados todos os *flits* recebidos, juntamente com os bits de paridade do código de *hamming*, que são calculados na origem. A posição 0 deste buffer tem um de seus bits alterados nessa simulação para ilustrar a correção de um erro. No momento que o roteador envia a posição 0 do buffer pela porta leste, podemos observar em destaque que a síndrome calculada é diferente de 0, e a saída do decodificador possui o dado sem o erro que foi inserido, sendo o mesmo enviado corretamente ao roteador vizinho.

## 6.2 Ocupação de Área e Frequência

Para determinar a área adicional utilizada pela rede com tolerância a falhas, inicialmente sintetizamos uma rede sem alterações, com dimensões de 8x8 roteadores com buffers de 8 posições, utilizando o software *Synplify Pro*, tendo como plataforma alvo uma FPGA Xilinx Virtex4 LX100-12, com 98304 blocos lógicos (LUTs). A Tabela 6 mostra os resultados de área e frequência de operação para a rede original, sem tolerância a falhas.

**Tabela 6. Consumo de área da rede Hermes 8x8 sem alterações em FPGA.**

| | Utilizados | Total | Ocupação |
|---|---|---|---|
| **Slices** | 28525 | 49152 | 58% |
| **Flip-Flops** | 14350 | 98304 | 14% |
| **LUTs** | 53926 | 98304 | 55% |
| **Frequência** | 229.3 MHz | | |

Para verificar o impacto dos métodos de tolerância a falhas apresentados no Capítulo 5, sintetizamos estas arquiteturas com as mesmas condições utilizadas na geração da Tabela 6. A Tabela 7 apresenta os resultados de área obtidos para as arquiteturas que utilizam o CRC como método de codificação.

**Tabela 7. Consumo de área e frequência de operação das redes 8x8 com codificação CRC em FPGA.**

| | | CRC nos enlaces | | CRC na origem | |
|---|---|---|---|---|---|
| | Total | Utilizados | % | Utilizados | % |
| **Slices** | 49152 | 31868 | 64 | 36094 | 73 |
| **Flip-Flops** | 98304 | 14939 | 15 | 13132 | 13 |
| **LUTs** | 98304 | 60520 | 61 | 67923 | 69 |
| **Frequência** | | 227.7 MHz | | 154.1 MHz | |

Como podemos observar, a adição de tolerância a falhas através de codificação CRC nos canais com mecanismo de reenvio no caso de erros possui um acréscimo de área moderado em relação à rede original, a ocupação de blocos lógicos em FPGA aumenta cerca de 12% para esta rede.

Para a rede com CRC na origem, o aumento do consumo de área é um pouco mais significativo, ocupando 26% a mais que a rede original e aproximadamente 12% a mais que a rede com CRC nos enlaces. Mesmo que esta rede não possua codificadores em todas as portas de saída dos roteadores, o consumo de área se mostra maior devido ao aumento dos *buffers* da rede de 16 para 20 bits.

Em relação à frequência de operação, podemos observar que a rede com CRC nos enlaces não possui um impacto significativo de redução na frequência de operação em relação à rede original. Porém, a rede com CRC na origem apresentou um grande impacto na redução da freqüência de operação, necessitando de uma freqüência 48% menor que a rede original para continuar operando normalmente.

Os mesmos dados de área foram coletados utilizando-se as arquiteturas que utilizam códigos de *Hamming* como método de codificação. A Tabela 8 apresenta os resultados para estas redes.

**Tabela 8. Consumo de área e freqüência de operação das redes 8x8 com códigos de *Hamming* em FPGA.**

| | | ***Hamming* nos enlaces** | | ***Hamming* na origem** | |
|---|---|---|---|---|---|
| | Total | Utilizados | % | Utilizados | % |
| **Slices** | 49152 | 28815 | 58 | 46621 | 94 |
| **Flip-Flops** | 98304 | 14396 | 14 | 15006 | 15 |
| **LUTs** | 98304 | 54482 | 55 | 80325 | 81 |
| **Frequência** | | 230.5 MHz | | 129.0 MHz | |

A rede codificação *Hamming* nos enlaces demonstra menor acréscimo de área em relação aos demais métodos desenvolvidos considerando uma FPGA como plataforma alvo. O aumento do consumo é de apenas 1% neste caso. Entretanto, a rede com *Hamming* na origem demonstra um aumento de 49% no consumo de recursos da FPGA, porém é a arquitetura que apresenta maior cobertura contra falhas transientes, recuperando falhas não somente nos enlaces, mas também dentro dos roteadores.

A rede com *Hamming* nos enlaces não apresentou impacto na freqüência de operação como esperado. Já a rede com *Hamming* na origem demonstra uma redução de 77% na frequência de operação em relação à rede original. Este resultado está relacionado ao aumento de área, que acaba ocasionando um aumento nos caminhos críticos do circuito.

Logo após as medidas iniciais, tendo como plataforma alvo uma FPGA, considerando que os problemas de confiabilidade deverão ocorrer com mais freqüência em tecnologias nanométricas de circuitos integrados, sintetizamos as arquiteturas desenvolvidas utilizando o software Encounter RTL Compiler da Cadence, com a biblioteca 0.35μm standard cells. A Tabela 9 mostra a ocupação de células da rede original e das arquiteturas desenvolvidas com codificação CRC.

**Tabela 9. Consumo de área da rede original e das redes com CRC de tamanho 8x8 mapeadas para *standard cells*.**

| | **Rede original** | **CRC nos enlaçes** | | **CRC na origem** | |
|---|---|---|---|---|---|
| | # células | # células | % | # células | % |
| **Roteador c/ 5 portas** | 3537 | 4025 | 13.8 | 4145 | 17.2 |
| **- Buffer** | 547 | 590 | 7.8 | 630 | 15.3 |
| **- Lógica de TF** | 0 | 256 | - | 144 | - |
| **Total NoC cells** | 208864 | 236672 | 13.3 | 243968 | 16.8 |

A partir da análise desta tabela podemos observar que o aumento de área observado na rede com CRC nos enlaces, mapeando-se a mesma para a biblioteca de standard cells, é de 13%, muito próximo ao aumento observado utilizando-se FPGA. Entretanto, para rede com CRC na origem, o aumento de área observado utilizando-se standard cells é de 16%, contra 26 % de aumento em

FPGA, demonstrando que o uso da técnica de CRC na origem é uma opção viável caso a plataforma final da rede seja um ASIC. Essa diferença está associada à arquitetura das FPGAs, que não possuem bancos de registradores, e neste caso, para aumentar a largura dos *buffers*, o gasto de LUTs é maior.

A Tabela 10 mostra a comparação da rede original com as redes que utilizam codificação *Hamming* em relação à ocupação de standard cells.

**Tabela 10. Consumo de área da rede original e das redes com códigos de *Hamming* de tamanho 8x8 mapeadas para *standard cells*.**

| | **Rede original** | ***Hamming* nos enlaços** | | ***Hamming* na origem** | |
|---|---|---|---|---|---|
| | # células | # células | % | # células | % |
| **Roteador c/ 5 portas** | 3537 | 4149 | 17.3 | 4864 | 37.5 |
| **- Buffer** | 547 | 547 | 0 | 936 | 71.3 |
| **- Lógica de TF** | 0 | 612 | - | 144 | - |
| **Total NoC cells** | 208864 | 243136 | 16.4 | 281395 | 34.7 |

O acréscimo de área observado na rede com código de *Hamming* nos enlaces mapeada para standard cells é de 16,4%, contra apenas 1% de acréscimo utilizando-se FPGA. Este resultado se deve principalmente ao fato de que um bloco lógico do FPGA pode ser usado para mapear uma equação de várias entradas para um bit de saída, como é o caso dos codificadores e decodificadores do código de *Hamming*, visto que as demais estruturas da rede não são alteradas nesta arquitetura. Já a rede com *Hamming* na origem apresenta um aumento de 34.7% no total de células utilizadas. Isto se deve principalmente ao aumento do número de decodificadores nos roteadores, visto que é adicionado um decodificador a cada porta de saída para corrigir possíveis erros internos aos roteadores, e também ao aumento dos buffers para o transporte dos bits de overhead do código de *Hamming*. Este método aumenta a confiabilidade da rede e pode ser necessário dependendo da aplicação.

## 6.3 Impacto de Latência

As latências são medidas levando em conta 4 cenários, variando-se principalmente a taxa de injeção e o tamanho dos pacotes, com o objetivo de observar o aumento de latência gerado pela recuperação de dados através de reenvio em cenários de alto e baixo congestionamento. Todas as simulações foram realizadas utilizando-se distribuição normal da taxa de injeção, e pacotes com destinos randômicos.

A análise de latência é feita apenas em uma NoC com CRC em relação à NoC de referência, pois as duas versões de redes com CRC apresentam o mesmo acréscimo de latência, devido à

utilização do mesmo mecanismo de retransmissão. Além disso, a rede com correção de erros através de códigos de *Hamming* não apresenta acréscimo de latência para correção de erros, e não possui nenhum mecanismo de retransmissão, de maneira que não é relevante realizar medidas de latência para essa rede.

A variação na taxa de erros é feita alterando-se o número de condições que fazem o modelo MAF gerar uma falha através do módulo *Saboteur*. Com 2 condições se obtêm uma baixa taxa de erros, com 4 condições a taxa de erros observada é alta. Variando-se o tamanho dos pacotes é possível aumentar ou diminuir o overhead de controle da rede. Pacotes maiores geram menos overhead de controle, logo a latência média observada deve ser menor.

O primeiro cenário de teste, apresentado na Tabela 11, foi realizado utilizando-se uma rede 8x8, taxa de injeção de 20% da banda máxima da porta local, sendo injetados 200 pacotes de 32 *flits* em cada porta local, esperando-se um alto grau de congestionamento.

**Tabela 11. Latências observadas no primeiro cenário de teste (32 *flits*).**

| | NoC Original | NoC CRC 2 condições | NoC CRC 4 condições |
|---|---|---|---|
| **Taxa de erros (%)** | 0 | 0,49 | 4,05 |
| **Latência de Rede** | 190,19 | 188,06 | 192,04 |
| **Latência de Aplicação** | 2003,50 | 2055,58 | 2300,96 |

A latência de rede corresponde à média de ciclos de relógio que um pacote leva para chegar ao seu destino após a sua entrada na rede. Em cenários de alto congestionamento, um pacote pode permanecer bloqueado na porta local de um roteador até que a porta de saída correspondente seja liberada. A latência de aplicação busca medir o número médio de ciclos de relógio que os pacotes levam desde o pedido de entrada na rede até o completo recebimento no seu destino. Como podemos observar no primeiro cenário de teste, existe um impacto de aproximadamente 15% de acréscimo na latência de aplicação com uma taxa de injeção de erros de 4%. Porém é importante observar que este cenário de teste possui um grau de congestionamento muito elevado, e uma taxa de erros de 4% dificilmente será observada na prática.

A Figura 35 apresenta uma representação gráfica da distribuição de latências observadas no primeiro cenário de teste. O eixo *X* representa intervalos de latência que variam de 0-50 ciclos de relógio até maior que 3000 ciclos de relógio. O eixo *Y* representa a quantidade de pacotes observados para cada intervalo de latência. As três curvas da figura indicam os resultados para a rede sem injeção de falhas, injeção através de 2 condições e injeção através de 4 condições do modelo MAF.

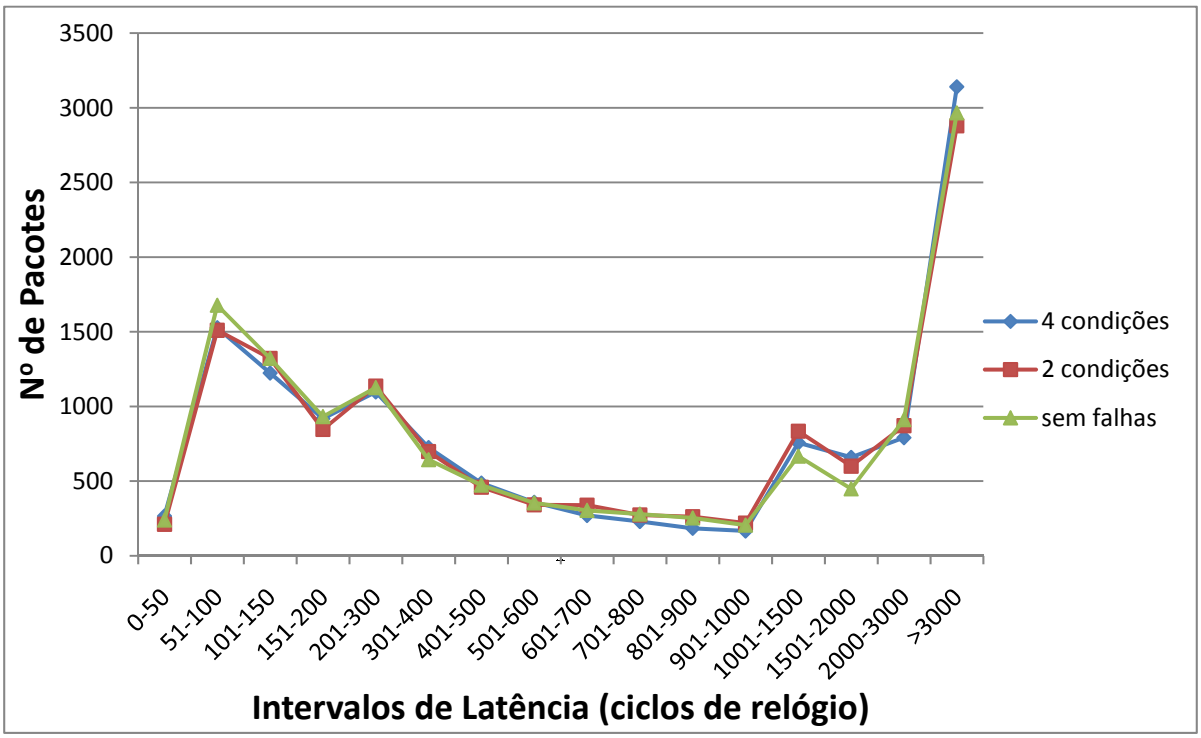


**Figura 35. Distribuição de latências no primeiro cenário de teste.**

Podemos observar que as curvas são muito semelhantes, comprovando o baixo impacto de latência do método de reenvio desenvolvido. As maiores diferenças são observadas próximas ao intervalo de 1501-2000 onde os cenários com injeção de falhas apresentam mais pacotes nesta faixa, enquanto que o cenário sem injeção apresenta mais pacotes no intervalo de 51-100, comprovando que nos cenários com injeção de falhas existem mais pacotes com latências mais altas, devido ao reenvio de *flits* corrompidos.

O segundo cenário de teste, apresentado na tabela Tabela 12, foi realizado utilizando-se uma taxa de injeção de 20% nas portas locais, assim como no primeiro cenário, porém com pacotes de 48 *flits* ao invés de 32*flits* como no cenário anterior, esperando-se uma latência de aplicação ligeiramente menor.

**Tabela 12. Latências observadas no segundo cenário de teste (48 *flits*).**

| | NoC Original | NoC CRC 2 condições | NoC CRC 4 condições |
|---|---|---|---|
| **Taxa de erros** | 0 | 0,0896 | 2,35 |
| **Latência de Rede** | 227,63 | 231,51 | 234,58 |
| **Latência de Aplicação** | 1547,86 | 1866,56 | 1863,17 |

Como esperado as latências de aplicação médias para o segundo cenário de teste são menores do que no primeiro. Entretanto, a latência de rede é maior. Isso acontece porque com pacotes maiores, o tempo de controle gasto em cada roteador no processo de roteamento é dividido entre uma maior carga útil, diminuindo a latência total de aplicação. Já a latência de rede

aumenta porque a latência mínima para o roteamento de um pacote depende do seu tamanho, quando se utiliza o método de chaveamento *wormhole*, sendo basicamente a soma da quantidade de *flits* com o tempo de controle gasto a cada roteador. Entretanto a latência adicional observada nos cenários com injeção de falhas é um pouco maior, representado um aumento de aproximadamente 20% na latência média, considerando as duas taxas de injeção de falhas utilizadas. Novamente, o impacto na latência de rede não é significativo, denotando que uma vez que o pacote entra na rede, o tempo de espera não é alterado significativamente com os processos de reenvio.

A Figura 36 ilustra a distribuição de latências para o segundo cenário de teste.

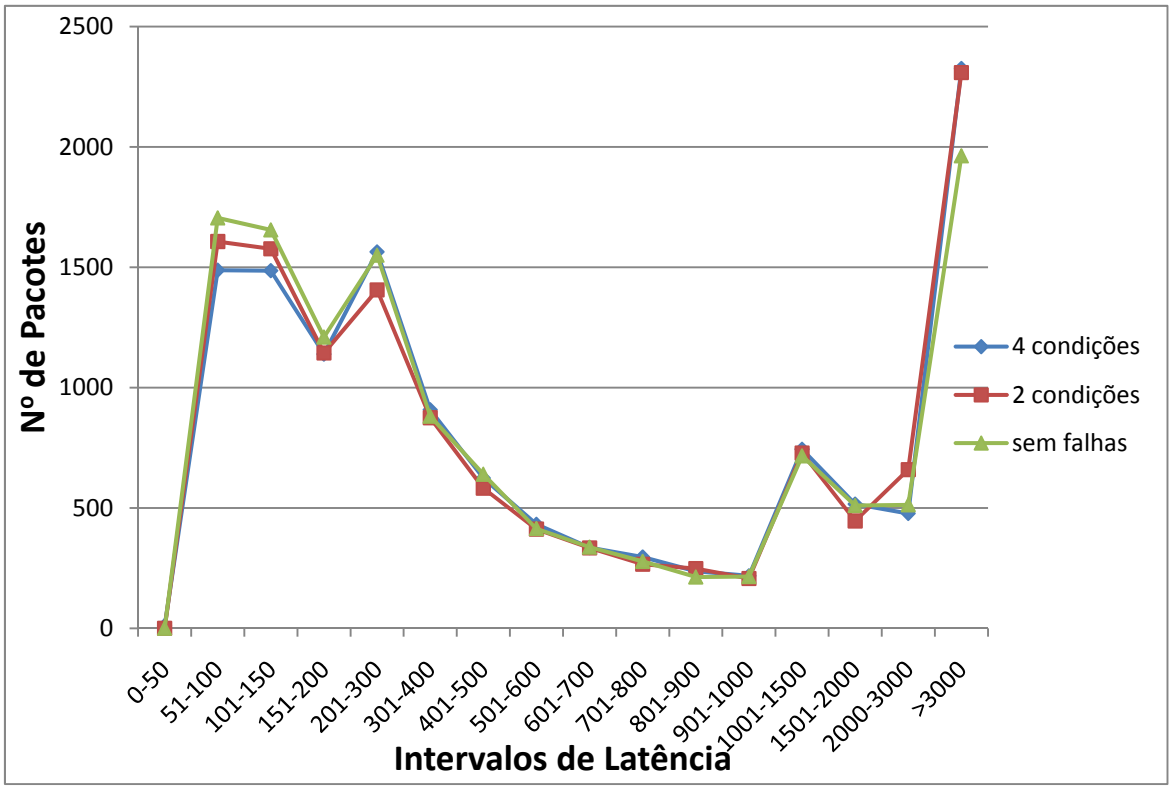


**Figura 36. Distribuição de latências no segundo cenário de teste.**

Novamente podemos observar que as curvas são semelhantes, porém os cenários com injeção de falhas apresentam menos pacotes nos intervalos de 51-100 e 101-150, enquanto apresentam muitos pacotes acima de 3000 ciclos de latência.

Os resultados para o terceiro cenário de teste são apresentados na Tabela 13. Este cenário foi realizado utilizando-se uma taxa de injeção de 10% nas portas locais, com 200 pacotes de 48 *flits* sendo injetados a cada roteador. O objetivo deste experimento é observar o comportamento da rede em um cenário de baixo congestionamento.

**Tabela 13. Latências observadas no terceiro cenário de teste.**

| | NoC Original | NoC CRC 2 condições | NoC CRC 4 condições |
|---|---|---|---|
| **Taxa de erros** | 0 | 0,11 | 2,21 |
| **Latência de Rede** | 101,08 | 101,11 | 101,77 |
| **Latência de Aplicação** | 101,09 | 101,12 | 101,78 |

Como podemos observar, as latências médias observadas neste cenário são muito mais baixas que nos cenários anteriores, devido à menor taxa de injeção nas portas locais. É possível observar também que o impacto do mecanismo de reenvio neste cenário é praticamente negligenciável, sendo menor que 1%, mostrando que em cenários de baixo congestionamento, o processo de recuperação por reenvio não chega a perturbar a latência média dos pacotes na rede.

A Figura 37 apresenta as distribuições de latências por intervalos para o terceiro cenário de teste.

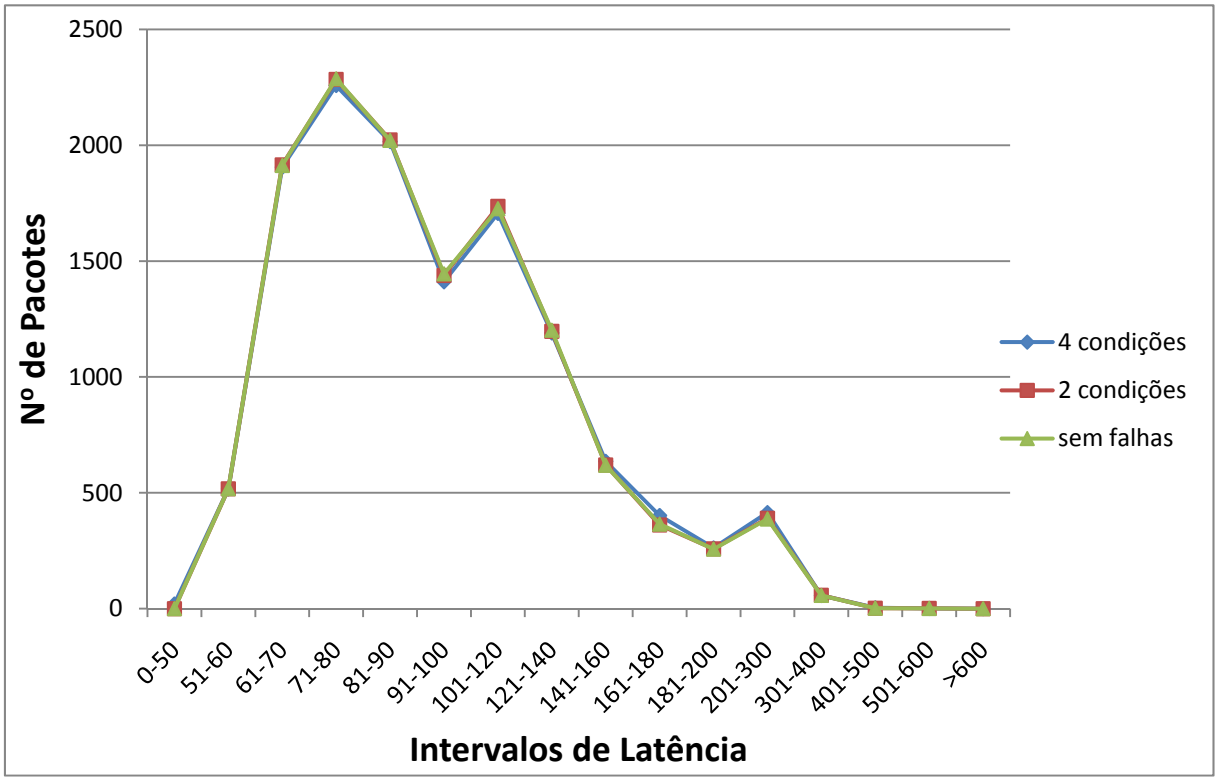


**Figura 37. Distribuição de latências no terceiro cenário de teste.**

Conforme apresentado na figura acima, as linhas encontram-se muito mais próximas que nos cenários anteriores. Isto se deve principalmente ao fato de que a rede pouco congestionada não sofre impacto significativo de latência quando se utiliza recuperação por reenvio conforme o trabalho desenvolvido.

O quarto cenário de teste foi realizado utilizando-se uma taxa de injeção de 15% nas portas locais, também com 200 pacotes de 48 *flits* a cada porta de entrada. A idéia deste cenário é ter um

caso intermediário entre uma taxa de injeção capaz de gerar alto grau de congestionamento e uma taxa que não cause congestionamento. Como podemos ver na Tabela 14, a taxa de injeção de 15% possui ainda um impacto muito menor do que os cenários com 20% nas latências médias da rede, e um impacto ligeiramente superior ao cenário com taxa de 10%.

**Tabela 14. Latências observadas no quarto cenário de teste.**

| | NoC Original | NoC CRC 2 condições | NoC CRC 4 condições |
|---|---|---|---|
| **Taxa de erros** | 0 | 0,054 | 2,34 |
| **Latência de Rede** | 127,74 | 127,84 | 129,28 |
| **Latência de Aplicação** | 129,90 | 130,42 | 131,17 |

Como podemos observar, as latências medidas no quarto cenário são muito menores que as latências observadas no primeiro cenário devido à menor taxa de injeção nas portas locais. É possível observar também que as latências de aplicação são muito próximas às latências de rede, indicando que os pacotes não permanecem bloqueados por muito tempo nas portas locais devido ao baixo congestionamento da rede. Neste cenário o impacto de latência observado utilizando-se o mecanismo de retransmissão foi de apenas 0,9% a uma taxa de injeção de falhas de 2.34%.

A Figura 38 apresenta as distribuições de latências para o quarto cenário de teste.

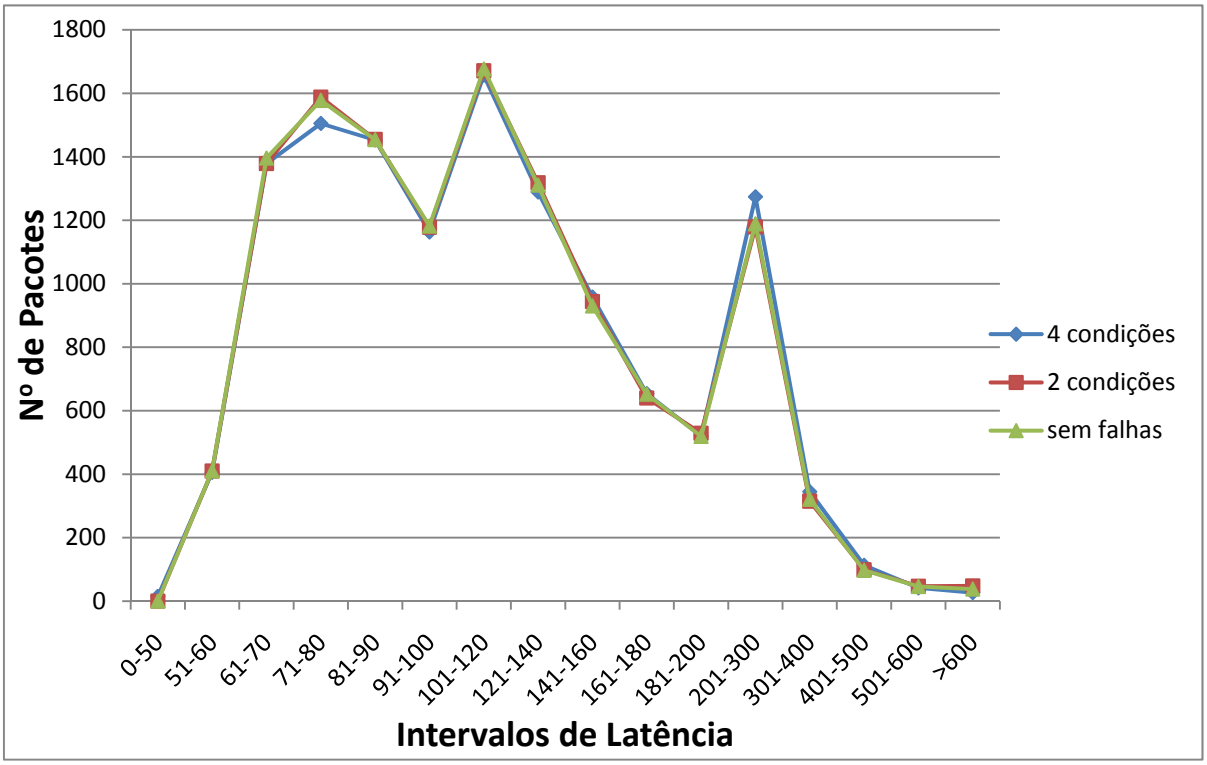


**Figura 38. Distribuição de latências no quarto cenário de teste.**

Novamente as curvas são muito semelhantes. Em apenas dois intervalos podem ser observados uma pequena diferença entre os pontos. No intervalo 71-80 existem menos pacotes no

cenário com 4 condições, enquanto no intervalo 201-300 o cenário com 4 condições possui uma quantidade um pouco maior de pacotes do que com taxas de erros mais baixas.

## 6.4 Análise de Defeitos Residuais

Esta seção analisa a eficácia dos dois métodos implementados para proteger a rede contra falhas transientes. Para analisar a taxa de defeitos residuais utilizando códigos de *Hamming* e CRC, o módulo *Saboteur* foi parametrizado para injetar falhas variando-se o número de condições do modelo MAF para aumentar a taxa de injeção de erro. O cenário simulado considera uma NoC 5x5, com cada módulo enviando 200 pacotes para um destino randômico, usando 15% de taxa de injeção nas portas locais. A Tabela 15 apresenta os resultados desta simulação.

**Tabela 15. Análise de defeitos residuais.**

| Condições MAF | Flits Tansmitidos | Erros Injetados | Defeitos Residuais de CRC | | Defeitos Residuais de Hamming | |
|---|---|---|---|---|---|---|
| $d_r$ | 806.021 | 341 | 0 | 0% | 0 | 0% |
| $d_r$,$d_f$ | 808.716 | 444 | 0 | 0% | 0 | 0% |
| $d_r$,$d_f$, $g_n$ | 794.816 | 896 | 0 | 0% | 0 | 0% |
| $d_r$,$d_f$, $g_n$,$g_p$ | 809.575 | 16.727 | 14 | 0.08% | 389 | 2.32% |

Como esperado, a codificação CRC apresenta uma taxa de defeitos residuais mais baixa que códigos de *Hamming*. Quando todas as condições do modelo MAF são verificadas para a injeção de falhas, a taxa residual de defeitos observada é de 2.32% para a rede protegida com códigos de *Hamming* contra apenas 0.08% da rede protegida com CRC. É possível observar também que com taxas de erro mais baixas, os dois tipos de codificação apresentam desempenho satisfatório para proteger a rede.

## 6.5 Consumo de Potência

O consumo de potência é medido realizando-se a análise do arquivo VCD (atividade de chaveamento) através da ferramenta *Sysnopsys Prime Power*. A rede foi sintetizada com a biblioteca TSMC25 e simulada nesta tecnologia. Para gerar o arquivo VCD utilizou-se uma rede 5x5, com buffers de 8 posições, mapeada em portas lógicas com o mesmo cenário de tráfego utilizado para a análise de defeitos residuais.

A Tabela 16 apresenta o consumo médio de potência da rede original, sem tolerância a falhas, e das arquiteturas que utilizam CRC para proteger a rede de falhas transientes. A tabela analisa a potência dos roteadores em diferentes localidades da rede. Roteadores nas bordas da rede podem possuir 3 ou 4 portas, enquanto os roteadores centrais possuem 5 portas. O consumo de potência total da rede é a soma do consumo de cada roteador.

**Tabela 16. Aumento no consumo de potência das arquiteturas com CRC.**

| | Rede original (mW) | CRC nos enlaces (mW) | % | CRC na origem (mW) | % |
|---|---|---|---|---|---|
| **Roteador 3 portas** | 2.76 | 2.79 | 1.09 | 3.32 | 20.3 |
| **Roteador 4 portas** | 3.62 | 3.66 | 1.10 | 4.38 | 21.0 |
| **Roteador 5 portas** | 4.49 | 4.55 | 1.34 | 5.45 | 21.4 |
| **NoC 5x5** | **94.9** | **96.0** | **1.15** | **115.0** | **21.8** |

Como podemos observar, o aumento no consumo de potência da rede com codificação CRC nos enlaces é pequena (1.15%), comparada com a rede original. Entretanto, o aumento de consumo da rede com CRC na origem é de 21.8%, devido ao aumento de 4 bits nos buffers da rede, confirmando que a maior parte do consumo de potência de NoCs se dá nos mesmos.

A Tabela 17 apresenta o consumo médio de potência das arquiteturas com código de *Hamming*, relacionado os mesmos com o consumo da rede original.

**Tabela 17. Aumento no consumo de potência das arquiteturas com código de *Hamming*.**

| | Rede original (mW) | *Hamming* nos enlaços (mW) | % | *Hamming* na origem (mW) | % |
|---|---|---|---|---|---|
| **Roteador 3 portas** | 2.76 | 2.79 | 1.09 | 3.56 | 27.6 |
| **Roteador 4 portas** | 3.62 | 3.66 | 1.10 | 4.62 | 27.6 |
| **Roteador 5 portas** | 4.49 | 4.57 | 1.78 | 5.83 | 29.8 |
| **NoC 5x5** | **94.9** | **96.2** | **1.37** | **123.0** | **29.6** |

O consumo de potência adicional da arquitetura com *Hamming* nos enlaces não chega a ser significativo, assim como a arquitetura com CRC nos enlaces, consumindo apenas 1.37% a mais que a rede original considerando o mesmo cenário de teste. Já a rede com *Hamming* na origem, apresenta uma sobrecarga no consumo de potência na ordem de 30%, sendo a arquitetura com o maior consumo entre todas as arquiteturas desenvolvidas. Isso se deve principalmente ao aumento dos buffers em 5 bits, para comportar os bits de paridade do código de *Hamming*, ao invés de 4 como na rede com CRC. Este aumento também se deve ao fato de que os decodificadores encontram-se nas portas de entrada e também nas portas de saída para corrigir erros internos ao roteador, sendo que a rede com CRC na origem possui apenas decodificadores nas portas de entrada.

# 7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou diferentes arquiteturas para aumentar a confiabilidade na comunicação de dados dentro de um chip através de NoCs, nas quais a rede é capaz de manter o seu desempenho original mesmo na presença de falhas. As quatro arquiteturas propostas diferem-se na localização do cálculo dos bits de paridade e na codificação utilizada para se obter a redundância dos dados. Uma das contribuições deste trabalho está na disponibilização de uma versão da plataforma ATLAS capaz de gerar as arquiteturas aqui apresentadas, dando aos usuários desta ferramenta a possibilidade de adicionar tolerância a falhas aos seus projetos com redes intra-chip.

A primeira arquitetura apresentada utiliza codificação CRC para proteger os enlaces da rede dos efeitos de *crosstalk*, de maneira que a paridade é calculada toda a vez que um roteador envia um *flit* para um roteador vizinho. Foi observado que esta abordagem é bastante atrativa para se utilizar em projeto de NoCs, pois a área adicional ocupada por este método é bastante baixa, o aumento no consumo de potência é praticamente desprezível, os processos de reenvio não chegam a afetar a latência média da rede quando os fluxos são controlados e a cobertura de falhas é muito alta. A arquitetura com os bits de CRC calculados na origem necessita de um buffer maior para comportar estes bits de CRC, de maneira que ela apresente um consumo de área e potência adicional maior do que a primeira arquitetura. Entretanto, utilizando-se o cálculo de CRC na origem, é possível de se recuperar alguns erros internos ao roteador, caso o componente afetado não seja o buffer, pois como essa rede utiliza reenvio como método de recuperação, a rede pode solicitar reenvio de um dado corrompido no buffer, ou seja, o dado será reenviado com erro. A terceira arquitetura apresentada neste trabalho utiliza código de *Hamming* calculado na saída de cada roteador para a correção de *flits* que possuam no máximo um bit com erro. Esta arquitetura apresentou consumo de área e potência adicional bastante baixo, porém levemente superior à arquitetura com CRC nos enlaces. A grande vantagem desta rede está no fato de corrigir *flits* com erro automaticamente, sem a necessidade de reenvio, de maneira que a mesma não interfira no desempenho da rede mesmo em fluxos mais congestionados. Entretanto, a única desvantagem desta arquitetura em relação a rede com CRC nos enlaces é a menor cobertura de falha, de maneira que sua utilização deve ser estudada para se utilizar ou em arquiteturas que tolerem um pequeno número de falhas de dados, como transmissão de vídeo e música por exemplo, ou se o MTBF do circuito garanta o seu funcionamento com alto grau de confiabilidade. A quarta arquitetura apresentada e avaliada neste trabalho utiliza código de *Hamming* calculado na origem, de maneira

que os buffers da rede também foram aumentados para suportar os bits de paridade juntamente com os bits de dados. Esta arquitetura demonstrou maior aumento de área e consumo de potência entre todas as arquiteturas estudadas, porém é a única capaz de suportar erros em bits de memória nos buffers, que poderão ocorrer com mais frequência em ambientes com mais interferências, como em sistemas espaciais por exemplo. Esta rede também garante cobertura de falhas nos enlaces e a falhas internas ao roteador, além de não influenciar na latência dos pacotes.

Ao final deste trabalho podemos concluir que a implementação de técnicas de tolerância a falhas nos projetos atuais de redes intra-chip é inevitável, uma vez que as aplicações exigem cada vez mais desempenho com frequências de operação elevadas e grande capacidade de integração em um mesmo chip. Neste contexto, não há outro meio de continuar evoluindo a tecnologia sem projetar os componentes pensando em recuperação de erros em tempo de execução, pois como foi estudado não se pode mais confiar totalmente no funcionamento do silício utilizando-se métodos agressivos de voltagem e frequência. Portanto a utilização de métodos de tolerância a falhas pode ser visto como um método eficaz para garantir baixo consumo de potência e alto desempenho para as novas tecnologias de produção de circuitos integrados. Cabe a cada projetista analisar as características e os requisitos de cada projeto para decidir qual dos métodos de tolerância a falhas é o mais adequado para garantir a comunicação intra-chip com o maior grau de confiabilidade possível.

Os resultados deste trabalho foram apresentados como artigo completo na conferência VLSI-SoC 2009 [SIL09], tendo sido o mesmo selecionado para publicação estendida em forma de um capítulo de livro.

## 7.1 Trabalhos Futuros

Dois tipos de falhas não são cobertos por este trabalho e podem ser vistos como trabalhos futuros: (*i*) falhas permanentes e (*ii*) falhas de controle.

Com relação a falhas permanentes a maior parte da literatura utiliza algoritmos de roteamento adaptativo para isolar roteadores ou módulos de hardware com falhas permanentes. O desenvolvimento de uma arquitetura capaz de detectar roteadores com falhas permanentes e a implementação de um algoritmo de roteamento adaptativo para desativar roteadores com falhas permanentes nas arquiteturas atuais é uma das atividades a ser desenvolvida nos próximos trabalhos.

Falhas de controle tais como inversão de um bit nas tabelas de roteamento, podem levar um pacote a ser roteado por um caminho errôneo e trancar a rede após um tempo de funcionamento, levando à perda de serviço. Neste caso a técnica mais utilizada para proteger componentes deste tipo de situação ainda consiste em redundância de componentes. Entretanto, os objetivos deste trabalho compreende a geração de redes mais confiáveis sem o aumento excessivo no consumo de área, de maneira que a replicação completa de componentes é vista como uma técnica inviável. Dessa maneira, um dos trabalhos futuros previstos é o desenvolvimento de técnicas para tolerar falhas de controle com mínima redundância para se utilizar na comunicação intra-chip.

# REFERÊNCIAS BIBLIOGRÁFICAS


[ATL08] "Atlas – An Enviroment for NoC Generation and Evaluation".Capturado em: http://www.inf.pucrs.br/~gaph/AtlasHtml/AtlasIndex_us.html, Agosto 2008.

[BEN02] Benini, L.; Micheli, G. "Networks on Chips: a New SoC paradigm". *IEEE computer*, vol. 35, no. 1, Jan 2002, pp. 70-78.

[BER04] Bertozzi, D.; Benini, L. "Xpipes: A Network-on-chip Architecture for Gigascale Systems-on-Chip". *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, Abr-Jun 2004, pp. 18-31.

[BER06] Bertozzi, D.; "The Data-Link Layer in NoC Design". Em: Micheli, G.; Benini, L. "Networks on Chips: Tecnology and Tools", New York:Morgan Kaufmann Publishers, 2006, 1ª Edição, 408p.

[CUV99] Cuviello, M.; Dey, S.; Bai, X.; Zhao, Y. "Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects".In: IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD'99), 1999, pp. 297-303.

[GLA92] Glass, C.; Ni, L. "The Turn Model for Adaptive Routing", In: 19th Annual International Symposium on Computer Architecture, 1992, pp. 278-287.

[GRA01] Gracia, J.; Baraza, J.C.; Gil, D.; Gil P.J. "Comparison and Application of Different VHDL-Based Fault Injection Techniques".In: IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'01), 2001, pp. 233-241.

[GRE07] Grecu, C.; Anghel, L.; Pande, P. P.; Ivanov, A.; Saleh, R. "Essential Fault-Tolerance Metrics for NoC Infraestructures". In: IEEE International On-Line Testing Symposium (IOLTS 2007), 2007, pp. 37-42.

[HUA08] Huang, P.; Fang, W.; Wang, Y.; Hwang, W."Low Power and Reliable Interconection with Self-Corrected Green Coding Scheme for Network-on-Chip".In: Second International Symposium on Networks-on-Chip (NoCs 2008), 2008, pp. 77-83.

[KOH09] Kohler, A.; Radetzki, M. "Fault-Tolerant Architecture and Deflection Routing for Degradable NoC Switches".In: 3rd ACM/IEEE International Symposium on Networks-on-Chip (NoCs 2009), 2009, pp. 22-31.

[LAP04] Laprie, J. C.; Avižienis, A.; Randell, B.; Landwehr, C. "Basic Concepts and Taxonomy of Dependable and Secure Computing".*IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, Jan-Mar 2004, pp. 11-33.

[MAH04] Maheshwari, A.; Burleson, W.; Tessier, R. "Trading Off Transient Fault Tolerance and Power Consuption in Deep Submicron (DSM) VLSI Circuits".*IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 3, Mar 2004, pp. 299-311.

[MAR03] Marculescu, R. “Networks-on-Chip: The Quest for On-Chip Fault-Tolerant Communication”. In: IEEE Computer Society Annual Symposium on VLSI Design (ISVLSI'03), 2003, pp. 8-12.

[MAT09] The Mathworks Inc. “MATLAB – Documentation”. Capturado em:http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?/, Setembro 2009.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. “HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip”. *Integration, the VLSI Journal*, vol. 38, Out2004, pp. 69-93.

[MUR05] Murali, S.; Benini,L.; Micheli, G.; Theocharides, T.; Vijaykrishnan, N.; Irwin, M.; “Analysis of Error Recovery Schemes for Networks on Chips”. *IEEE Design and Test of Computers*, vol. 22, no.5, Set-Out 2005, pp. 434-442.

[MUR07] Murali, S.; Atienza, D.; Benini, L.; Micheli, G. “A Method for Routing Packets Across Multiple Paths in NoCs with In-Order Delivery and Fault-Tolerance Guarantees”.*VLSI-Design Journal*, vol. 2007, Out 2007, 11p.

[PIR04] Pirretti, M.; Link, G.; Brooks, R.; Vijaykrishnan, N.; Kandemir, M. Irwin, M. “Fault Tolerant Algorithms for Network-on-Chip Interconnect”. In: IEEE Computer Society Annual Symposium on VLSI, 2004, pp.46-51.

[PUL05] Pullini, A.; Angiolini, F.; Bertozzi, D.; Benini, L. “Fault Tolerance Overhead in Network-of-Chip Flow Control Schemes”. In: 18th Symposium on Integrated Circuits and Systems Design (SBCCI’05), 2005, pp. 224-229.

[RAB03] Rabaey, J. “Digital Integrated Circuits: A Design Perspective”. New Jersey: Prentice Hall, 2003, 792 p.

[SIL09] Silva, A.; Moraes, F. “Crosstalk Fault Tolerant NoC: Design and Evaluation”. In: 17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 2009), 2009, 6p.

[SPR01] Sprachmann, M. “Automatic Generation of Parallel CRC Circuits”. *IEEE Design & Test of Computers*, vol. 18, no.3, Jan-Mar 2001, pp. 109-114.

[VEL04] Vellanki, P.; Banerjee, N.; Chatha, K. “Quality-of-Service and Error Control Techniques for Network-on-Chip Architectures”. In: Great Lakes Symposium on VLSI (GLSVLSI’04), 2004, pp. 45-50.

[ZHU07] Zhu, H.; Pande, P.; Grecu, C. “Performance Evaluation of Adaptive Routing for Achieving Fault Tolerance in NoC Fabrics”. In: Application-Specific Systems, Architectures and Processos (IEEE ASAP 2007), 2007, pp. 42-47.

[ZIM03] Zimmer H.; Jantsch A. “A Fault Model Notation and Error-Control Scheme for Switch-to-Switch Buses in a Network-on-Chip”. In: Hardware/Software Codesign and System Synthesis (CODES+ISSS’03), 2003, pp. 188-193.