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

TÉCNICAS DE TOLERÂNCIA A FALHAS APLICADAS A REDES INTRA-CHIP

VINICIUS MORAIS FOCHI

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
2015

Dados Internacionais de Catalogação na Publicação (CIP)

F652t Fochi, Vinicius Morais

Técnicas de tolerância a falhas aplicadas a redes intra-chip / Vinicius Morais Fochi. – Porto Alegre, 2015.
70 p.

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

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

CDD 004.35

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 "Técnicas de Tolerância a Falhas Aplicadas a Redes Intra-Chip" apresentada por Vinícius Morais Fochi como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 13/03/2015 pela Comissão Examinadora:

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

Prof. Dr. Alexandre de Morais Amory– PPGCC/PUCRS

Profa. Dra. Fernanda Gusmão de Lima Kastensmidt– UFRGS

Homologada em 21/05/2015, conforme Ata No. 008 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
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 as pessoas que realmente tiveram influência sobre o desenvolvimento desta dissertação e do meu Mestrado. Gostaria de dedicar esta conquista aos meus pais, José Jorge Fochi que me apoiou a continuar estudando, Rosi Morais Fochi e a minha irmã Cristina.

Ao Professor Fernando Moraes, por aceitar o imenso desafio de me orientar. Muito obrigado por manter reuniões semanais, me obrigando assim a trabalhar intensamente para não perder o foco no desenvolvimento do trabalho. Obrigado por todo o conhecimento transmitido e por sempre acreditar no nosso trabalho. Por ter compartilhado seu tempo e conhecimento para que a realização deste trabalho. Sou muito grato por me aceitar como aluno de mestrado.

Agradeço a todos os professores do PPGCC pela transmissão de seus conhecimentos. Aos empregados da PPGCC pela dedicação em nos manter sempre informados dos prazos e burocracias. Um agradecimento especial ao Professor Alexandre de Morais Amory, que enriqueceu esta dissertação em suas avaliações. Aos colegas que me ajudaram durante a jornada do mestrado: Samir, Guedes, Madalozzo, Castilhos, Ruaro, L.Heck, Matheus, Wachter, Augusto e a todos os demais: obrigado pela ajuda! E a Dell por ter possibilitado e financiado esta pesquisa.

TÉCNICAS DE TOLERÂNCIA A FALHAS APLICADAS A REDES INTRA-CHIP

RESUMO

O contínuo desenvolvimento na tecnologia de transistores possibilitou que centenas de processadores trabalhassem interconectados por NoCs (network-on-chip). A nanotecnologia permitiu o desenvolvimento de complexos sistemas, porém a vulnerabilidade a falhas também aumentou. A literatura apresenta soluções parciais para o tema de tolerância a falhas, tendo como alvo partes do sistema. Uma importante lacuna na literatura é um método integrado para detecção de falhas do nível do roteador até a correta execução das aplicações em MPSoC reais. O objetivo principal desta dissertação é apresentar um método com tolerância a falhas da camada física até a camada de transporte. O MPSoC é modelado em nível de RTL, usando VHDL.

O presente trabalho propõe técnicas de tolerância a falhas aplicadas a redes intra-chip. São estudadas técnicas de tolerância a falhas em nível sistêmico, nível do roteador, nível de enlace e algoritmos de roteamento tolerante a falhas. Este trabalho apresenta a pesquisa e o desenvolvimento de duas técnicas: (*i*) protocolos para permitir a correta transmissão dos dados com degradação parcial do enlace, de forma a permitir que o roteador opere mesmo com canais físicos falhos; (*ii*) método de teste e recuperação do roteador. O modelo de falhas utilizado nesta Dissertação é de falhas permanentes e transientes.

Para avaliar as técnicas propostas, foi utilizada a plataforma HeMPS, juntamente com uma campanha de injeção de falhas onde até cinco falhas aleatórias foram injetadas nos canais de comunicação entre os roteadores simultaneamente em cada cenário. Foram utilizadas duas aplicações para avaliar as técnicas: codificador MPEG e uma aplicação sintética, com um total de 2,000 cenários simulados. Os resultados demonstram a efetividade da proposta, com a maioria dos cenários executando corretamente com roteadores operando em modo degradado, com um impacto no tempo de execução abaixo de 1% e um aumente do área de 30% no roteador.

Palavras-Chave: MPSoC, NoC, HeMPS, detecção de falhas, degradação parcial do enlace, teste e recuperação de falhas.

FAULT TOLERANCE TECHNIQUES APPLIED TO NETWORKS ON CHIP

ABSTRACT

The continuous development of the transistor technology has enabled hundreds of processors to work interconnected by a NoC (network-on-chip). Nanotechnology has enabled the development of complex systems, however, fault vulnerability also increased. The literature presents partial solutions for fault tolerance issues, targeting parts of the system. An important gap in the literature is an integrated method from the router-level fault detection to the correct execution of applications in the MPSoC. The main goal of this dissertation is to present a fault-tolerant method from the physical layer to the transport layer. The MPSoC is modeled at the RTL level using VHDL.

This work proposes fault tolerance techniques applied to intra-chip networks. Related work on fault tolerance at a systemic level, router level, link level and routing algorithms are studied. This work presents the research and development of two techniques: (*i*) protocols to enable the correct communication between task with partial degradation of the link enabling the router to operate even with faulted physical channels; (*ii*) test recovery method and of the router. This Dissertation considers permanent and transient faults.

The HeMPS platform is the reference platform to evaluate the proposed techniques, together with a fault injection campaign where up to five random failures were injected simultaneously at each simulated scenario. Two applications were used to evaluate the proposed techniques, MPEG encoder and a synthetic application, resulting in 2,000 simulated scenarios. The results demonstrated the effectiveness of the proposal, with most scenarios running correctly with routers operating in degraded mode, with an impact on the execution time below 1%, with a router area overhead around 30%.

Keywords: MPSoC, NoC, HeMPS, fault detection, partial degradation of the link, testing and fault recovery.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BIST	Built-in self-test
CRC	Cyclic redundancy check
ECC	Error Control Coding
FIFO	First-In First-Out
FPGA	Field Programmable Gate Arrays
FT	Fault Tolerant
HeMPS	Hermes Multiprocessor System
MIPS	Microprocessor without Interlocked Pipeline Stages
MPEG	Moving Picture Experts Group
MPI	Message Passing Interface
MPSoC	Multi-Processor System-on-Chip
NI	Network Interface
NoC	Network-on-Chip
PE	Processing Element
SEU	Single Event Upset
SoC	System-on-Chip
TMR	Triple Module Redundancy
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very-Large-Scale integration

SUMARIO

INTRODUÇÃO

O contínuo desenvolvimento na tecnologia da fabricação de transistores gera circuitos integrados (CIs) cada vez menores. Para a efetiva utilização da quantidade de transistores que podem ser fabricados em um mesmo CI, a utilização de módulos de propriedade intelectual (IP) tem-se tornado um método padrão no projeto destes circuitos. A solução mais comumente utilizada para interconectar os diversos módulos IP são as arquiteturas de barramento, tanto centralizada como hierárquica.

Esta arquitetura de interconexão, barramento, apresenta limitações de escalabilidade, desempenho e consumo de energia em sistemas computacionais integrados em um único CI, como MPSoCs (*multiprocessor system on chip*). Dentre as soluções propostas para a interconexão de IPs, destaca-se a utilização de redes intra-chip (*network-on-chip* - NoC) [BEN02]. NoCs têm por característica a escalabilidade, o que permite que o constante crescimento do atraso nas comunicações. Outra característica é que a dissipação de potência é amenizada devido a suas propriedades físicas (fios mais curtos que os barramentos). A estrutura física de NoCs compreende segmentos de fios (enlaces), interconectados por blocos de roteamento (denominados na literatura por roteadores ou chaves), de forma a prover certo nível de qualidade de serviço (QoS) [BRA90]. Qualidade de serviço em NoC inclui parâmetros de desempenho a serem respeitados, como largura de banda disponível para a comunicação entre IPs, dissipação de potência e confiabilidade.

A redução no tamanho físico dos transistores os torna mais rápidos e com menor consumo de potência. Apesar desta redução parecerem benéficas, a probabilidade de falhas tende a crescer. As interconexões entre os componentes do CI tornam-se relativamente mais longas e mais estreitas, podendo sofrer efeitos como eletro-migração, curto-circuito ou fios aberto. Os transistores por possuírem canal mais curto, também ficaram mais suscetíveis a falhas transientes. Técnicas de projeto visando o baixo consumo podem gerar consequências negativas no correto funcionamento dos CIs. CIs operando com baixa tensão de alimentação são mais vulneráveis a falhas, fazendo com que o circuito sofra perturbações causadas pelas condições de ambiente (e.g. ruídos, SEUs, dentre outros).

Assim, projeto tolerante a falhas assume uma dimensão cada vez maior nas atuais tecnologias nanométricas. Em particular, o presente trabalho tem por *objetivo* estudar técnicas para tolerância a falhas em NoCs.

1.1 Definições básicas

Confiabilidade é definida de acordo com a IEEE [IEE90] como a habilidade do sistema ou do componente de realizar determinada função sob uma determinada condição e por um período específico de tempo. Uma falha no sistema ocorre ou está presente quando o serviço que provê o sistema difere do serviço especificado ou do serviço que deveria ser oferecido.

Existem três tipos de falhas: permanentes, transientes e intermitentes [GRE07].

Falhas permanentes são falhas constantes, como falha no enlace, no roteador ou em um elemento de processamento. Falhas transientes são falhas esporádicas e podem ser geradas por fatores externos como interferência eletromagnética ou por algum ruído/erro do próprio circuito integrado, como *crosstalk*. Falhas intermitentes são ocasionadas por erros em rajada, geralmente se repetem em um mesmo local esporadicamente.

Falhas permanentes são causadas por uma irreversível mudança física no CI. A origem mais comum para esse tipo de falha é um defeito gerado no processo de fabricação. Falhas permanentes podem ocorrer durante toda a vida do circuito, particularmente quando o circuito começa a entrar na zona de envelhecimento. Característica comum a todas as falhas permanentes é que uma vez que a falha foi detectada, ela não irá desaparecer.

Falhas transientes geram um estado temporário de mau funcionamento causado por algum fator externo do ambiente, como radiação, ou um fator interno como ruído proveniente de outra parte do circuito. Falhas transientes não deixam nenhum resíduo no circuito, pois uma vez que o erro termina, o circuito volta a operar normalmente. Uma manifestação comum para erros transientes é a mudança do valor binário em um sinal. A ocorrência de uma falha transiente é tipicamente aleatória e difícil de ser detectada.

Falhas intermitentes são ocasionadas por erros em rajada. Estas falhas geralmente se repetem em um mesmo local, porém não são contínuas como as falhas permanentes. Falhas intermitentes são tipicamente associadas a um circuito instável ou velho, e são ativadas por um fator relacionado ao ambiente como mudança de temperatura ou mudança de voltagem. Elas frequentemente precedem falhas permanentes. Além disso, falhas intermitentes são difíceis de detectar, pois elas podem ocorrer apenas sob uma determinada condição do ambiente ou através de uma combinação especifica de entradas.

Para representar a quantidade de falhas de um sistema ou componente em relação ao tempo é muito utilizada a curva da banheira, conforme ilustrado na Figura 1 [JHA03]. Esta curva contém três zonas distintas. Durante o início da operação de um dado sistema, a curva começa com uma taxa decrescente de falhas, e os circuitos falhos nesta zona devem-se à “mortalidade infantil”. Tipicamente na zona da mortalidade infantil as falhas são geradas devido a defeitos de fabricação, como problemas na oxidação ou nas máscaras utilizadas para a litografia. Na zona central da curva temos o tempo de vida útil do circuito. Falhas transientes podem ocorrer aleatoriamente, devido a diversos fatores como mencionado acima. Na parte final da curva temos a zona de envelhecimento. Nesta região a curva de falhas tem um crescimento devido ao final da vida útil do circuito. Falhas nesta região ocorrem devido à própria degradação dos componentes do circuito.

Podemos dizer que um defeito em um sistema eletrônico é diferença entre o hardware implementado e o projeto pretendido. Alguns exemplos típicos de defeitos em circuitos VLSI são:

- Defeitos no processo de fabricação: capacitâncias parasitas, rompimento das linhas de metal, variabilidade na dopagem do silício, impurezas na superfície;

- Defeitos relacionados ao tempo: eletromigração, rompimento de dielétrico.

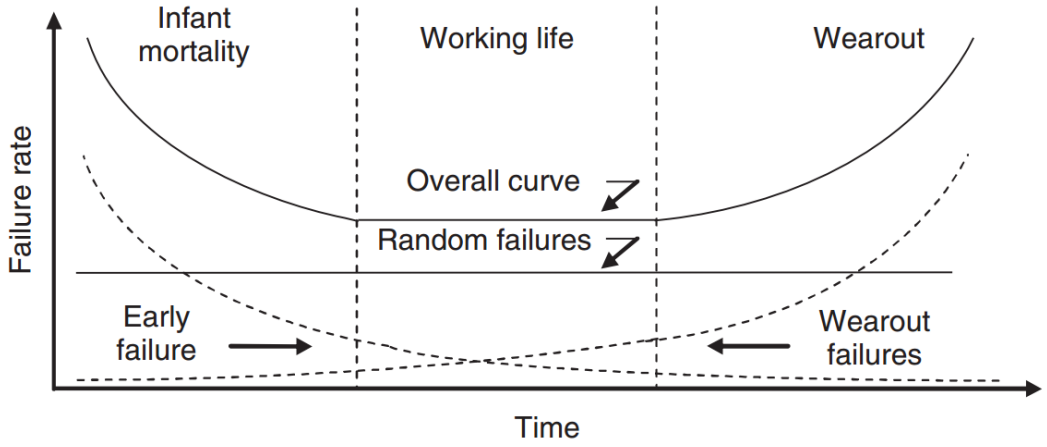

Figura 1 – Curva de envelhecimento do circuito [JHA03].

Tipicamente são utilizadas diversas métricas para detecção e estimativa das falhas no funcionamento de sistemas eletrônicos e componentes. Dentre as técnicas para detecção de falhas, a técnica BIST [JHA03] *(Built-In-Self-Test*) é frequentemente utilizada. BIST é uma técnica utilizada em tempo de projeto, utilizando hardware adicional no circuito com o objetivo de realizar testes sem a dependência de equipamentos externos.

Para realizar a estimativa de falhas tipicamente são utilizados as seguintes técnicas: MTBF (*Mean-Time-Between-Failures)* ou MTTF (*Mean-Time-To-Failure*) e MTTR (*Mean-Time-To-Repair*). MTBF é o parâmetro utilizado para representar o tempo médio para a próxima falha ocorrer, sendo calculado da seguinte forma:

MTBF = Total do tempo de operação / Número de falhas encontradas

Outra métrica bastante utilizada para a validação em sistemas tolerantes a falhas com capacidade de se reparar/recuperar é o MTTR, definido como:

MTTR = Tempo gasto para realizar o reparo / Número de reparos

Com base nas métricas MTBF e MTTR é possível calcular a disponibilidade do sistema e podendo assim mensurar o impacto de falhas em aplicações:

Disponibilidade = (MTBF / (MTBF + MTTR)) * 100%

Tradicionalmente são utilizadas técnicas de detecção e correção de erros para proteger os sistemas contra efeitos transientes de mau funcionamento. Porém, os projetistas precisam avaliar o custo de hardware. Mecanismos como redundância e detecção e correção de erros (EDC - *error detection and correction*) podem exigir um custo adicional de energia, dissipação de potência e aumento de área, podendo inclusive afetar o desempenho do circuito.

1.2 Objetivos do trabalho

A presente Dissertação possui os seguintes objetivos estratégicos e específicos.

Objetivos estratégicos:

- Domínio da tecnologia de redes intra-chip (NoC);
- Domínio de técnicas de tolerância a falhas em NoC;
- Implementação de técnicas de tolerância a falhas em NoC permitindo a operação da mesma em presença de falhas.

Objetivos específicos:

- Modificação da rede MazeNoC [WAC12] para inclusão de decodificadores e codificadores CRC em alguns pontos do roteador de forma a prover mecanismos de detecção de falhas;
- Integração da MazeNoC-CRC (rede MazeNoC com mecanismos de CRC) com a plataforma HeMPS;
- Domínio de técnicas de injeção de falhas;
- Desenvolver um método de controle para permitir a operação da NoC com a degradação parcial do sistema após detecção de falhas;
- Propor um protocolo de comunicação tolerante a falhas;
- Desenvolver um método de recuperação do sistema após detecção de falhas.

1.3 Contribuições da Dissertação

A presente Dissertação tem por objetivo contribuir no desenvolvimento de duas técnicas de tolerância a falhas para NoCs em projetos de MPSoCs: (*i*) operação do roteador em modo degradado em caso de detecção de falhas [FOC15]; (*ii*) recuperação do canal após ser detectado um falha. Ambas as técnicas são transparentes à aplicação final. A técnica de operação do roteador em modo degradado consiste em garantir que o enlace do roteador continue operando mesmo na presença de falha em um dos canais físicos. A técnica de regeneração do canal consiste no envio de um pacote de teste através do canal falho para determinar se o canal sofreu uma falha transiente ou permanente. Caso o pacote de teste seja recebido corretamente o enlace volta a operar normalmente.

1.4 Estrutura do documento

Este documento está organizado como segue. O Capítulo 2 revisa o estado da arte de técnicas de tolerância a falhas abordando técnicas de nível sistêmico, nível de roteador, nível de enlace e algoritmos de roteamento tolerante a falhas. O Capítulo 3 apresenta a plataforma HeMPS, usada como base desse trabalho. O Capítulo 4 apresenta a técnica de detecção de falhas no roteador. O Capítulo 5 apresenta protocolo de recuperação de falhas. O Capítulo 6 apresenta o método de recuperação do roteador. O Capítulo 7 apresenta a avaliação dos resultados. O Capítulo 8 apresenta os trabalhos futuros, sendo seguidas pelas referências bibliográficas.

ESTADO-DA-ARTE

A estrutura deste Capítulo está dividida conforme os temas de pesquisa relacionados ao trabalho. Na primeira sessão apresenta-se o estado da arte para tolerância a falhas em nível sistêmico. Na segunda sessão têm-se as pesquisas relacionadas à tolerância a falhas em nível de roteador. Na terceira sessão apresenta-se o estado da arte para tolerância a falhas em nível de enlace. Na quarta sessão apresentam-se os trabalhos relacionados a algoritmos de roteamento tolerante a falhas. O Capítulo encerra com uma avaliação crítica sobre os trabalhos revisados.

A Tabela 1 resume os trabalhos analisados nas seções subsequentes.

Tabela 1 - Tabela comparativa entre as técnicas estudadas.

Técnica	Resumo
Algoritmo de remapeamento em tempo de execução.	A proposta envolve diferentes camadas de software e hardware para lidar com tolerância a falhas [MEL12].
Lógica no roteador	Propõe uma NoC que suporta falhas em muitos componentes de rede, e ainda assim continuar operando corretamente [FIC09].
	Sugere um método de tolerância a falhas que utiliza buffers compartilhados entre os enlaces adjacentes quando uma falha no buffer é detectada [CON09].
	Propõe um método tolerante a falhas com uma degradação proporcional a quantidade de falhas detectadas em um mesmo enlace. [VIT12]
Lógica no roteador aplicada a enlaces	Técnicas de paridade e de CRC, com a retransmissão de dados em caso de detecção de uma falha [VEI10].
	Técnicas de CRC e Hamming para detecção de falhas, com a retransmissão de dados em caso de detecção de uma falha [SIL10].
Algoritmo de roteamento	Técnica de roteamento com foco em falhas permanentes, detectadas durante a execução do sistema [WAC13].

1.5 Técnicas de tolerância a falhas em nível sistêmico

O objetivo de técnicas de tolerância a falhas em nível sistêmico é prover, tanto em nível de hardware quanto de software, mecanismos que permitam às aplicações recuperarem-se e executarem corretamente em caso de falhas.

1.5.1 System Adaptivity and Fault-tolerance in NoC-based MPSoCs: the MADNESS Project Approach

Meloni et al. [MEL12] propõem um sistema adaptativo e tolerante a falhas para MPSoCs baseados em NoC, utilizando métodos aplicados no Projeto MADNESS. O trabalho busca reduzir a perda de desempenho ao utilizar técnicas de remapeamento

dinâmico (migração de tarefas) devido à detecção de falhas em tempo de execução.

Sistemas embarcados atuais necessitam de técnicas adaptativas tolerantes a falhas, pois a carga de trabalho que o sistema necessita tratar não pode ser prevista em tempo de projeto. Novas aplicações podem ser executadas ou necessitam de um tempo de resposta com recursos de energia e processamento limitados. Para isso é necessário realizar um balanceamento da carga de trabalho nos PEs do sistema. O projeto MADNESS pode lidar com essas situações, com técnicas que permitem mudar o mapeamento de uma aplicação em tempo de execução.

A arquitetura do MPSoC considera uma memória distribuída e um modelo de *tile* interconectados a uma NoC. A arquitetura é um modelo personalizado para um número de processadores e uma topologia de rede. Em nível de software foi elaborado uma camada que permite a execução de aplicações utilizando um modelo chamado *Polyhedral Process Network* (PPN). Esse modelo foi escolhido pois permite gerenciar o estado das tarefas das aplicações em tempo de execução. ?

As técnicas de tolerância a falhas foram introduzidas em nível de hardware e software, explorando a técnica de migração de tarefas nos PEs. Quando uma tarefa está mapeada em um PE falho é necessário transferir a tarefa para um PE sem falha, para que a aplicação continue a execução sem interrupção. Primeiramente é detectado o PE falho para que o processo de migração comece. O PE falho é excluído do processo de remapeamento e um hardware no *tile* falho garante o processo de migração. E por fim, a decisão do remapeamento é tomada levando em conta a menor perda de desempenho possível.

A Figura 2 ilustra a arquitetura proposta no projeto. A NI (*Network Interface*) é conectada com um gerenciador de mensagens com capacidade de DMA. Processadores RISC ou ASIP podem ser integrados utilizando a interface do barramento. A comunicação e a sincronização são gerenciadas através da memória mapeada nos registradores da interface de rede.

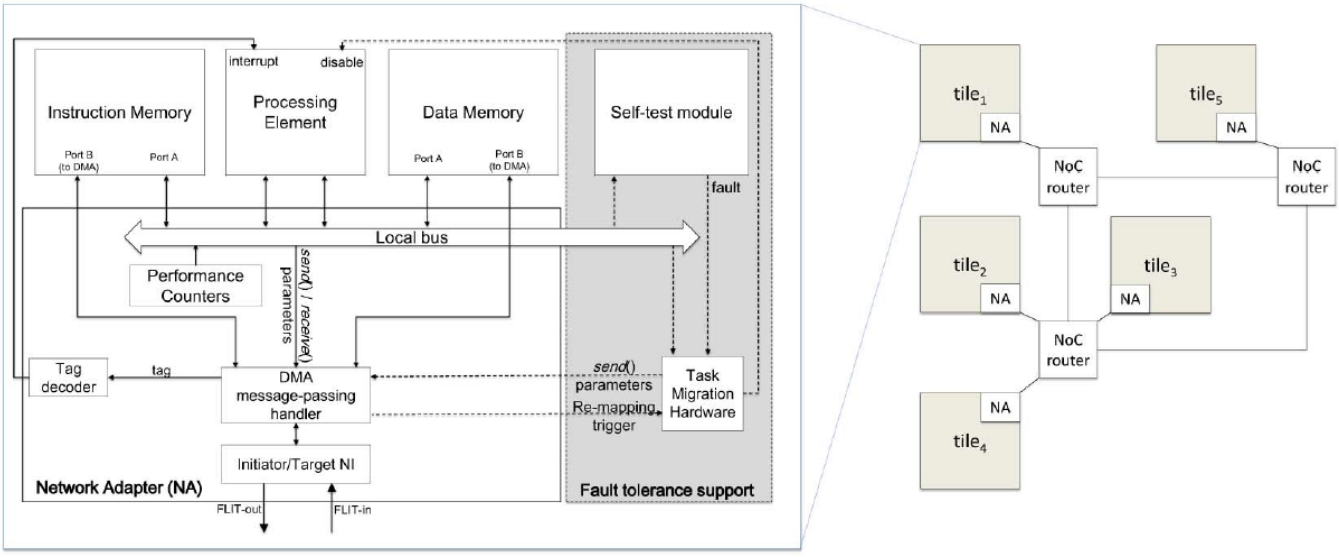

Figura 2 – Modelo da arquitetura proposta no Projeto MADNESS [MEL12].

A parte em cinza da Figura 2 representa o circuito adicional que fornece suporte à tolerância a falhas. Em casos em que a aplicação não é critica e um número limitado de erros é aceitável, uma rotina de teste é executada periodicamente em todos os PEs para detectar as falhas permanentes. O módulo de tolerância a falhas verifica o resultado das execuções de uma rotina em software e compara com os resultados previamente calculados. Caso haja alguma inconsistência, um sinal de erro é enviado ao módulo de migração de tarefas. Em casos de aplicações críticas, são utilizadas técnicas de testes em nível de rede. Cada tarefa é replicada em diferentes PE e depois do resultado ser processado, o mesmo é analisado para verificar a integridade do valor. Caso seja detectada uma falha, o PE falho é isolado e a aplicação é migrada para outro PE.

O módulo de hardware de migração de tarefas (TMH – *task migration hardware*) é usado caso um PE falhe. Ele é responsável por extrair os dados do *tile* falho. Ele recebe o sinal de falha do módulo de teste e inicia o procedimento para a migração. Uma condição fundamental para a migração de tarefas é que o PE falho não auxilie na migração. Para resolver esse problema o TMH é quem isola o PE falho para não participar da migração.

O projeto MADNESS define as estratégias de remapeamento em tempo de execução em duas classes. Na primeira, analisa-se exaustivamente todos os possíveis cenários de falhas em tempo de projeto, avaliando qual o melhor remapeamento para cada cenário. Com isso, quando uma falha ocorre não é necessário realizar um cálculo para detectar qual o melhor cenário. Contudo, essa abordagem requer o uso de memórias para armazenar tais cenários. A segunda técnica realiza o remapeamento em tempo de execução. O projeto MADNESS utiliza preferencialmente a segunda opção, pois esta requer uma quantidade menor de memória, não requerendo um estudo maior em tempo de projeto, e pode ser utilizado em aplicações que não são conhecidas *a priori*. Entretanto, a técnica executada em tempo de execução pode reduzir o desempenho das aplicações comparado à técnica realizada em tempo de projeto. Os autores relatam uma degradação do sistema em torno de 3% em casos onde foi necessário uma migração de tarefas.

1.6 Técnicas de tolerância a falhas no nível do roteador

O objetivo de técnicas de tolerância a falhas no nível do roteador é prover, em nível de hardware, mecanismos adicionais de recuperação de falhas que permitam ao roteador recuperar-se e executar corretamente em caso de falhas.

1.6.1 A Fault-Tolerant NoC Scheme Using Bidirectional Channel

Tsai et al. [TSA11] propõem uma NoC tolerante a falhas com enlaces bidirecionais. A arquitetura é capaz de suportar falhas transientes e permanentes nos enlaces do roteador. Em uma arquitetura tradicional de uma NoC com tolerância a falhas, quando um enlace falha o pacote é forçado a ser desviado por um caminho válido, fazendo com que tenha um maior custo na transmissão. Contudo, esse caminho válido pode não existir e com a mudança da rota original, pode ser gerado um *deadlock.* Para se encontrar um caminho alternativo algumas regras relacionadas com o algoritmo de roteamento podem ser violadas para encontrar o caminho válido. A proposta da técnica é utilizar uma rede

com canais bidirecionais reconfiguráveis, gerando uma maior flexibilidade para falhas transientes e permanentes.

Algumas técnicas utilizam roteadores e canais adicionais para garantir a confiabilidade da rede, porém isso causa um inevitável aumento no custo da NoC. Outras técnicas utilizadas para evitar *deadlocks* desenvolvem estratégias de roteamento através de desvios de pacote. Porém, esta última técnica requer uma reconfiguração das tabelas de roteamento causando um aumento da latência na transmissão dos pacotes. Essas técnicas são ineficientes para falhas transientes.

NoCs com canais bidirecionais e com largura de banda adaptativa foram propostas recentemente [LAN11] e [CHO09]. Canais bidirecionais suportam reconfiguração em tempo de execução para transmissões que demandam um maior desempenho. Contudo, o potencial de NoCs com canais bidirecionais em relação à confiabilidade não foi completamente explorado e estudado.

Em NoCs tradicionais, os roteadores vizinhos se comunicam com um par de canais unidirecionais, um para transmissão (TX) e outro para recepção (RX). NoCs com canais bidirecionais podem alternar a direção dos canais de transmissão RX e TX utilizando um mecanismo que suporte a mudança conforme a demanda do canal.

Na Figura 3 podemos observar os dois canais bidirecionais entre os roteadores vizinhos. Também é possível observar o canal adicional de controle para realizar dinamicamente a mudança da direção dos canais bidirecionais sempre que uma falha é detectada. Durante a execução normal dos roteadores, um canal é usado para transmissão e outro para recepção. Quando um canal falha, o canal operacional é reconfigurado para trabalhar de forma bidirecional, fazendo com que o canal opere tanto para recepção quanto para transmissão. No caso de falha em ambos os canais bidirecionais de uma mesma porta (e.g. porta norte), o controle do roteador informa a falha para os roteadores vizinhos para que o algoritmo de roteamento procure por rotas alternativas para aquele destino. Em casos onde a falha é transiente e o canal volte a funcionar ele reconfigura novamente os canais para operar separadamente. O método de recuperação do canal é fora do escopo deste trabalho. A proposta utiliza dois algoritmos de roteamento, ODD-EVEN [CHI00] e o XY, trabalhando em conjunto com os canais bidirecionais.

Os resultados encontrados demonstram que NoCs com canais bidirecionais de tamanho 8x8 apresentam confiabilidade de 90,80% caso tenham sete canais com falha. Combinando a técnica desenvolvida com os algoritmos de roteamento ODD-EVEN e XY foi obtida uma confiabilidade de 99,94% e 99,58% respectivamente. O custo em área adicional foi de 4,07% em relação a uma NoC unidirecional e de 1,86% de consumo de energia adicional.

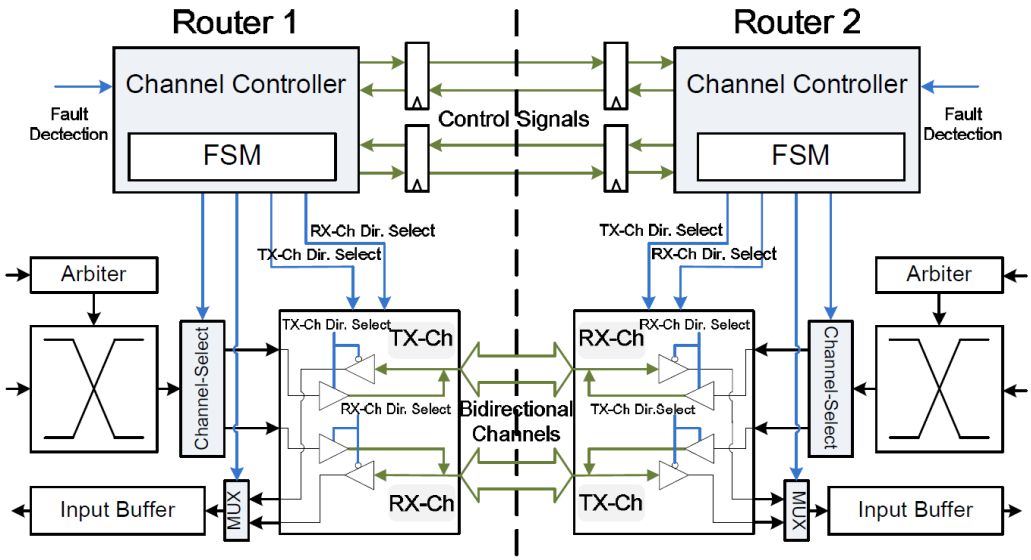

Figura 3 – Arquitetura dos roteadores e suas interconexões incluindo os sinais de controle e os canais bidirecionais [TSA11].

1.6.2 Vicis: A Reliable Network for Unreliable Silicon

Fick et. al. [FIC09] propõem uma NoC que suporta falhas em muitos componentes de rede e ainda assim continua funcionando corretamente. A rede Vicis utiliza como método a redundância em nível de rede e em nível de roteador para manter o funcionamento do sistema. A detecção da falha de um componente da rede é realizada em tempo de execução utilizando BIST para detectar exatamente qual componente apresenta falha. O trabalho descreve o funcionamento do método em quatro passos: 1) detecção do erro; 2) diagnóstico do erro; 3) reconfiguração do sistema; 4) recuperação do sistema.

Os mecanismos de detecção de erro informam que uma nova falha foi encontrada. Após é realizado um diagnóstico para determinar onde a falha foi detectada. A reconfiguração do sistema desabilita o componente com falha, e determina como será a nova configuração dos componentes funcionais. Por fim, o sistema de recuperação, caso detecte que um componente saiu do estado de falha, para um estado funcional restaura a configuração original do sistema.

A NoC Vicis trabalha com enlaces bidirecionais, com dois canais físicos por enlace. Cada enlace necessita de duas portas de entrada e duas portas de saída para que o enlace opere corretamente. Caso uma dessas portas falhe, um canal do enlace torna-se indisponível. Neste caso três portas ainda estarão funcionais, possibilitando que elas sejam utilizadas em outro enlace.

Conforme ilustrado na Figura 4, as portas de entrada do roteador são compostas por FIFOs (*First-in First-out*), e um decodificador. Um *port swapper* é posicionado em frente às FIFOs com o objetivo de alternar qual enlace físico está conectado a cada porta de entrada no roteador. As portas de saída não possuem o *port swapper* com o objetivo de economizar área. Os Autores relatam que como as FIFOS estão ligadas ao crossbar, não há necessidade de incluir um *port swapper* adicional.

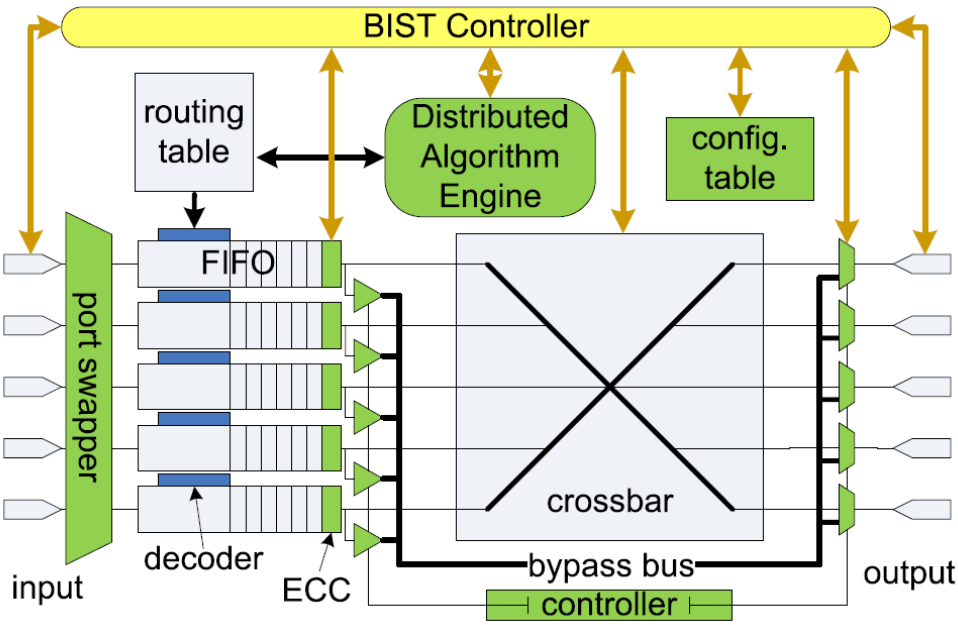

Figura 4 – Arquitetura do roteador da Vicis [FIC09].

A Figura 5 ilustra um exemplo de uso do *port swapping*. Na Figura à esquerda apresentam falha a porta sul de entrada do roteador central, e a porta oeste de saída do roteador à direita do roteador central. Nesse estado, apenas as portas oeste e norte do roteador central podem operar corretamente. Na Figura 5, à direita, ilustra-se a configuração após o *port swapping*. Como resultado, três portas estão em funcionamento e apenas à porta leste do roteador central está em estado de falha. O *port swapper* não necessita ser conectado com cada porta de entrada para que seja conectado com cada enlace físico. A configuração da rede Vicis define que a porta local está habilitada para conectar-se a três portas diferentes e os demais enlaces estão habilitados a conectarem-se em duas portas de entrada. Isto resulta em uma maior prioridade para a porta local, garantindo assim o funcionamento desta porta sempre que possível.

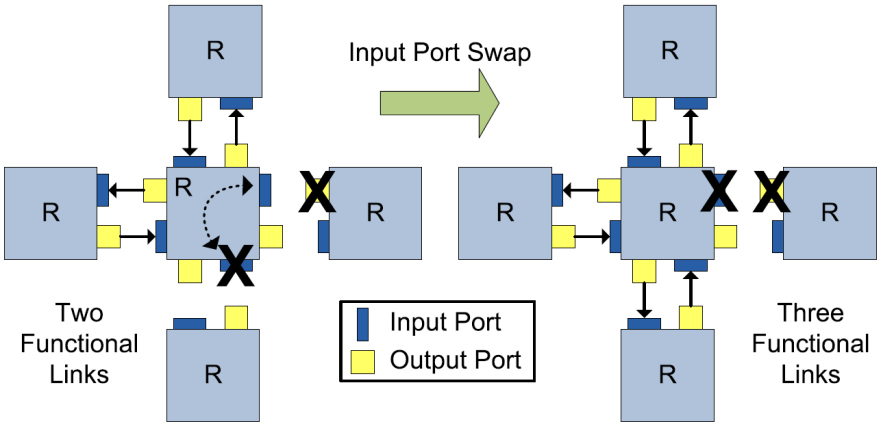

Figura 5 – Exemplo de *port swapping* [FIC09].

Após o algoritmo de *port swapping* terminar o processo, cada roteador e o algoritmo de roteamento são informados sobre quais enlaces são funcionais. O algoritmo de roteamento então realiza a reconfiguração da rede encerrando o processo. Para realizar o roteamento da rede a Vicis utiliza o algoritmo de roteamento proposto por Fick et al. [FIC09a], pois não utiliza canais virtuais e por ter baixo custo de implementação.

Conforme ilustrado na Figura 4, o barramento de *by-pass* em paralelo ao crossbar permite que os *flits* contornem o *crossbar* caso ele apresente em falha. O controlador do *crossbar* é configurado através do BIST para direcionar o tráfego através do *crossbar* ou o contornar utilizando o barramento de *by-pass* caso seja detectado uma falha no *crossbar*. Caso muitos *flits* necessitem usar o barramento de by-pass simultaneamente, então é escolhido um *flit* para ser enviado e os outros aguardam até o próximo ciclo. Os caminhos alternativos permitem que a NoC Vicis mantenha a rede operacional mesmo que múltiplas falhas ocorram no *crossbar*, perdendo desempenho, porém operando corretamente.

Os resultados relatados foram que a NoC Vicis teve um custo de área em torno de 42% superior à rede de referência, sendo considerado pelos Autores um custo aceitável comparado com a NoC NMR (*N-modular-redundancy*), que teve um aumento superior a 100% em área. Quanto à confiabilidade da rede, a NoC Vicis garante 100% de confiabilidade mesmo que apenas 50% dos roteadores continuem funcionando. A NoC Vicis obteve um bom desempenho em deixar os IP's conectados a pelo menos outro IP da rede. Os autores relatam que proposta se mostrou altamente tolerante a falhas de hardware. Cada roteador utiliza BIST para diagnosticar as falhas, e os algoritmos de reconfiguração mantém o roteador operando.

1.6.3 Fault Tolerant Mechanism to Improve Yield in NoCs Using a Reconfigurable Router

Concatto et al. [CON09] propõe um método de tolerância a falhas que utiliza buffers compartilhados entre os enlaces adjacentes quando uma falha em algum buffer é detectada. A técnica utiliza códigos de Hamming e TMR (*Triple Modular Redundancy*) para proteger os dados. O objetivo da proposta é diminuir a latência do sistema e aumentar o tempo de vida do sistema, juntamente com um baixo consumo de energia.

A técnica utiliza um roteador baseado no roteador RASOC, utilizado na rede SOCIN [ZEF03]. SOCIN é uma NoC com topologia malha com roteadores parametrizáveis, que utiliza o algoritmo XY para realizar o roteamento. O RASOC é um roteador com cinco portas bidirecionais (Local, Norte, Sul, Leste e Oeste). O roteador RASOC foi escolhido devido à possibilidade da reconfiguração dos buffers utilizados em cada porta de entrada. Uma porta pode emprestar uma parte ou o buffer inteiro para as portas adjacentes. Para reduzir as interconexões apenas as portas vizinhas de uma dada porta compartilham os buffers. Com isso cada porta tem a disponibilidade de compartilhar três buffers caso seja necessário. Em casos em que o buffer de uma porta esteja cheio, ele pode solicitar o uso dos buffers adjacentes. O método pressupõe que a NoC passe por um teste *off-line* para detectar e identificar os buffers falhos. Cada roteador tem um flip-flop extra por cada porta de entrada, para informar qual buffer apresenta falha.

A Figura 6 ilustra a porta Sul do roteador. Foram utilizados multiplexadores extras para permitir a reconfiguração dos buffers. Quando apenas uma unidade do buffer falha, o buffer inteiro não é descartado, apenas a unidade falha é isolada e a próxima unidade do buffer é usada no lugar da unidade defeituosa. Cada porta recebe três entradas de dados. Considerando a porta Sul como exemplo: sua própria entrada (*din_S*) é conectada com a entrada da porta à direta (*din_E*) e com a entrada da porta à esquerda (*din_W*), como

ilustrado na Figura 6. Todas as portas de comunicação tem a mesma arquitetura com exceção da porta local que não compartilha seu buffer.

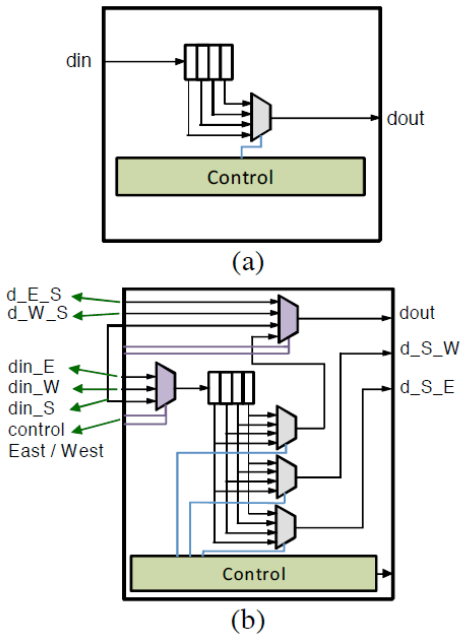

Figura 6 – FIFO de entrada (a) original (b) modelo proposto [CON09].

O trabalho proposto tem por objetivo combinar três técnicas de tolerância a falhas permanentes. Cada técnica abrange uma parte do circuito para resultar no menor custo adicional de área e desempenho. Foi utilizado o roteador reconfigurável como solução para os buffers com falha. A técnica de códigos da Hamming protege os enlaces caso ocorra alguma falha que inverta um bit. Conforme ilustrado na Figura 7, o processo de codificação e decodificação ocorre antes do dado entrar no buffer. Foi necessário incluir seis bits adicionais no enlace para o código de Hamming, somado aos trinta e quatro bits (trinta e dois de dados e dois de controle) totalizando quarenta bits. A terceira técnica utilizada para controle da FIFO foi TMR, visto que a FSM que controla a FIFO é muito pequena e tem apenas três estados. A TMR triplica a FSM dando assim uma maior tolerância a falhas para o controle da FIFO.

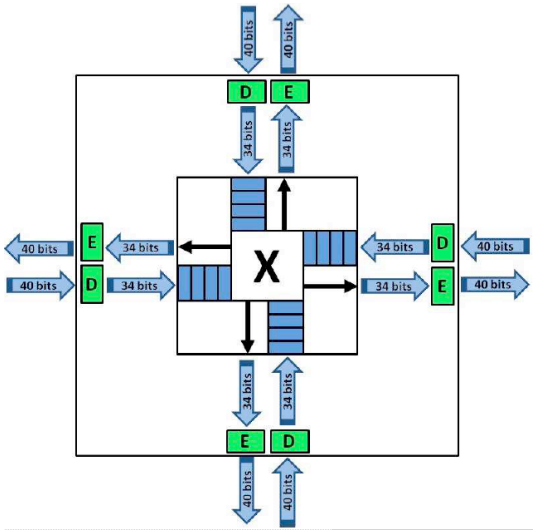

Figura 7 – Códigos de Hamming nos enlaces da NoC. E – Codificação, D – Decodificação [CON09].

Os resultados descritos pelos Autores foram que combinando as três técnicas citadas, um total de 63% do circuito do roteador da NoC está protegido (enlaces, buffers e o controle). Contudo não foi possível cobrir todas as falhas permanentes no roteador. O *crossbar* e o algoritmo de roteamento não possuem mecanismo de detecção e correção de falhas. O trabalho relata um aumento da área do circuito (172um² de *overhead*) e uma diminuição da frequência de operação, porém o método de reconfiguração manteve o desempenho e reduziu a energia dissipada.

1.6.4 A Dynamically Adjusting Gracefully Degrading Link-Level Fault-Tolerant Mechanism for NoCs

Vitkovskiy et. al. [VIT12] propõe um método tolerante a falhas com uma degradação proporcional a quantidade de falhas detectadas em um mesmo enlace. O método é chamado de PFLRM ”*Partially-faulty link recovery mechanism*”. O princípio básico do método é uma transmissão multi-ciclo fazendo um deslocamento de flits através dos fios em falha do enlace, conforme um determinado nível de máximo de falhas nos fios. Assim, os fios que estão em falha são ignorados, enquanto o roteador reagrupa os bits deslocados inicialmente no flit. O projeto PFLRM tem um leve aumento de área na arquitetura da NoC e uma pequena mudança no controle de fluxo dos pacotes.

A maior contribuição do PFLRM é lidar com um ambiente com muitas falhas, sem a necessidade de utilizar algoritmos de roteamento adaptativos, visto que o enlace não é completamente desabilitado. Um determinado número de fios falhos pode tornar não mais benéfico continuar utilizando o enlace funcionando parcialmente. Neste ponto o método faz com que o enlace se torne inativo. Em casos onde o enlace é desabilitado a rede pode utilizar um algoritmo de roteamento adaptativo.

O método PFLRM recupera os flits corrompidos atráves de uma PFL (Partially-faulty link) utilizando três etapas:

1) Ocorrência de uma falha e a detecção.

2a) Geração de um vetor de falha.

2b) Calculando a latência para a recuperação.

3) Retransmissão dos flits, utilizando o vetor de falha para enviar os flits, ao chegar no roteador adjacente o flit é recuperado.

Todas as três fases são executadas apenas quando uma falha ocorre. Depois que o vetor de falhas é gerado apenas a última fase é necessária, até que eventualmente seja detectado uma nova falha em um fio diferente.

Os resultados mostraram que o mecanismo PFLRM gerou um aumento de área e potência do roteador de 12,08% e 13,83% respectivamente. Os autores relatam que um aumento de área no chip como um todo seria de apenas 3%. Com isso se justificaria o uso do método PFLRM, para sistemas onde é benéfico que a rede mantenha a conectividade em um ambiente propício a muitas falhas.

1.7 Técnicas de tolerância a falhas no nível de enlace

O objetivo de técnicas de tolerância a falhas no nível de enlace é prover, em nível de hardware, mecanismos que permitam que a rede continue trabalhando corretamente mesmo se um enlace parar de funcionar.

1.7.1 Implementation of Techniques for Fault Tolerance in a Network-on-Chip

Veiga et. al. [VEI10] propõem duas técnicas para tolerância a falhas transitórias de *crosstalk*. As técnicas desenvolvidas foram de CRC [VEI10] e paridade com retransmissão de dados em caso de detecção de uma falha. As técnicas foram implementadas na rede SoCIN, modificando o bloco de controle de fluxo do roteador ParIS. O objetivo foi proteger os enlaces contra falhas de *crosstalk* nos canais de enlace.

A Figura 8 ilustra o diagrama de blocos do roteador ParIS. O roteador é constituído por um conjunto de módulos e internamente cada módulo é formado por um conjunto de blocos. O alvo do trabalho foram os blocos IFC (*Input Flow Controller*) e o OCF (*Output Flow Controller*). Os respectivos blocos são responsáveis pelo fluxo de entrada e saída de dados do roteador.

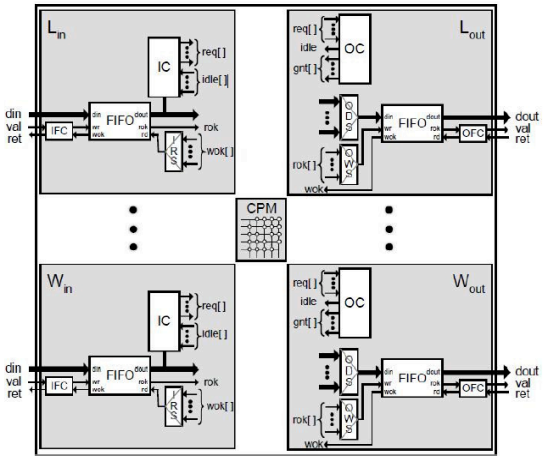

Figura 8 – Diagrama de blocos do roteador ParIS [VEI10].

Falhas causadas por *crosstalk* geram uma inversão de bits no canal de dados. Para simular o erro foi utilizado um módulo de injeção de falhas chamado Sabotador que é inserido no canal de dados. O módulo Sabotador monitora os dados transferidos em cada canal da rede e modifica o valor de alguns bits de dados para gerar o efeito de *crosstalk*.

Para realizar a técnica de paridade, foram realizadas alterações nos módulos de entrada e saída do roteador. Foi adicionado um fio de paridade e um fio de sinalização de erro para cada canal. A técnica proposta utiliza um único bit de paridade, que é calculado na saída do bloco OFC (Figura 9). A cada dado transmitido é acrescentado um bit de modo que o total de bits em ‘1’ seja par. Foi desenvolvido um circuito especifico para o cálculo de paridade, sendo realizado sobre os bits de dados. Quando o IFC recebe os bits de dados e o bit de paridade, é realizado o cálculo de paridade no canal de entrada e comparado com o bit de paridade recebido. Se o valor calculado for o mesmo, o dado é

considerado correto e é enviada uma sinalização de sucesso para o transmissor utilizando o fio *ret*, conforme ilustrado na Figura 9. Caso contrário, o receptor envia uma mensagem de erro através do fio *error*, solicitando uma retransmissão do dado. Caso ocorra *crosstalk* e um número par de bits for invertido, a técnica não consegue identificar o erro.

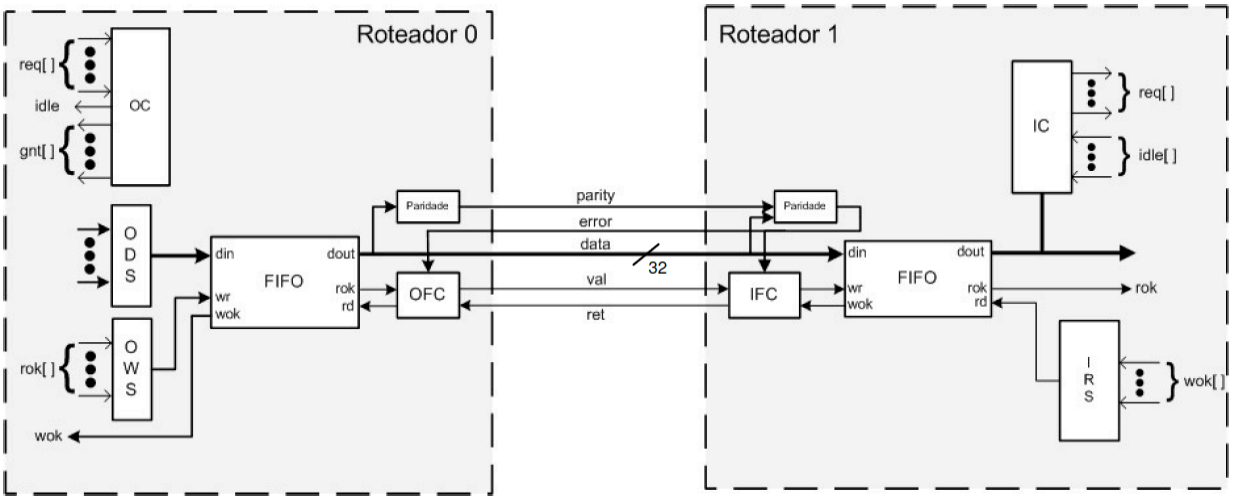

Figura 9 – Enlace da rede SoCIN incluindo os circuitos e os sinais para a técnica de paridade [VEI10].

A Figura 10 ilustra a modificação realizada nos módulos para desenvolver a técnica de CRC. Para desenvolver a técnica de CRC foi necessário modificar os módulos de entrada e saída, de forma semelhante à técnica de paridade. Foi acrescido um fio de CRC para cada quatro fios de dados e um fio de sinalização de erro. Foi necessário criar um componente de codificação do CRC no módulo de saída e um componente de decodificação no módulo de entrada. O módulo de saída é responsável por enviar os dados e gerar o CRC a ser enviado para o roteador destino. O modulo de entrada por sua vez recebe o código gerado e os dados enviados, comparando os bits de CRC recebidos com o CRC gerados localmente. Se os bits forem iguais então o dado é computado. Caso contrário, o receptor envia um sinal de erro ao transmissor para que seja feita a retransmissão do dado. Uma vez detectado o erro, o decodificador CRC desabilita o sinal de escrita na FIFO para que o dado não seja consumido.

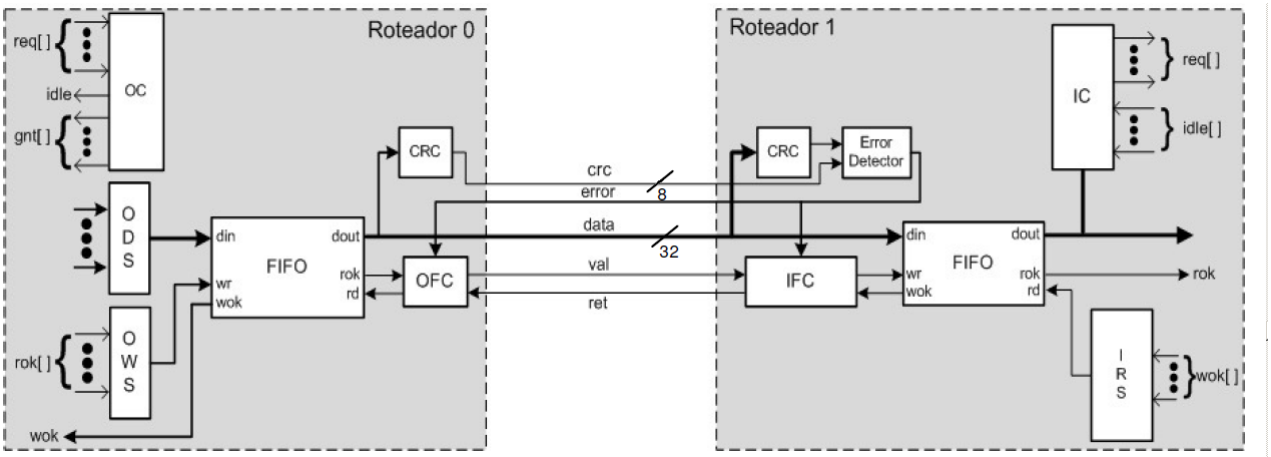

Figura 10 – Enlace da rede SoCIN incluindo os circuitos e sinais para a técnica de CRC [VEI10].

Os Autores relatam que as técnicas de paridade e CRC geram um custo adicional

de fios no enlace. Para ambas técnicas é necessária a inclusão de um fio (*error*) para sinalizar o erro detectado. Na técnica de paridade, além do fio *error* é também necessário um segundo fio para a sinalização de paridade (*parity*), com um custo adicional de área igual a 6,25%. Na técnica de CRC o custo foi de 28,125% pois é necessário incluir um fio de controle a cada quatro fios de dados. As técnicas desenvolvidas têm por objetivo apenas detectar o erro, e solicitar um reenvio dos dados, nenhuma das técnicas tem mecanismos de correção do erro.

1.7.2 Implementação e Avaliação de Métodos para Confiabilidade de Redes Intra-Chip

Silva [SIL10] propõe duas técnicas de tolerância a falhas para detectar e corrigir falhas transientes em sistemas expostos aos efeitos de crosstalk. Crosstalk é caracterizado pela interferência em um sinal resultante da atividade de chaveamento em linhas vizinhas. O Autor desenvolveu as técnicas de CRC/Hamming nos enlaces e CRC/Hamming na origem.

O ambiente de desenvolvimento das técnicas foi a rede Hermes [CAR09]. A rede Hermes é uma rede parametrizável, modelada em VHDL, em nível RTL. O algoritmo de roteamento escolhido foi o XY e o tamanho básico da rede foi de 8x8 com a topologia do tipo malha bidirecional.

A Figura 11 ilustra a interface de comunicação entre dois roteadores. O codificador CRC é adicionado em cada porta de saída do roteador, e o decodificador CRC é inserido em cada porta de entrada. Conforme ilustrado na Figura 11, quando o roteador 1 deseja enviar um dado para o roteador 2, o Codificador CRC gera a codificação em função do dado a ser enviado em data_out. O código gerado é enviado através do sinal crc_out. Ao receber o dado, o Roteador 2 calcula o CRC para o dado recebido e compara com o código CRC recebido em crc_in. Caso o dado tenha alguma inconsistência é enviado um sinal de erro através de error_out solicitando o reenvio.

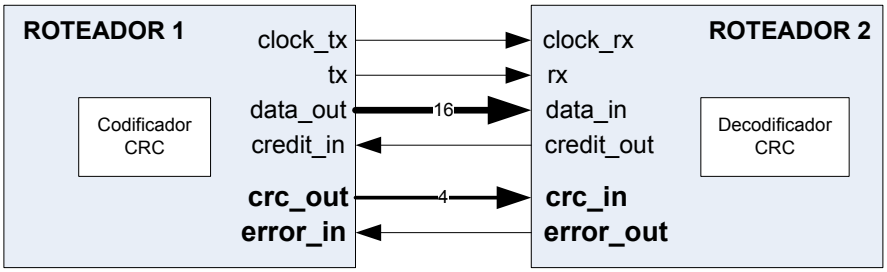

Figura 11 – Interface entre uma porta de saída e uma porta de entrada dos roteadores, com adição dos sinais de crc_in, crc_out, erro_in e erro_out [SIL10].

A Figura 11 ilustra a técnica de CRC roteador a roteador com reenvio em nível de *flit*. As vantagens da técnica são que a detecção do erro ocorre antes da decisão de roteamento. Desta maneira protegendo a rede contra pacotes com erro. O método mais rápido, e requer menos área, se comparado com a retransmissão de pacotes? completos, pois o mesmo deveria ser completamente armazenado no buffer para que o cálculo de CRC fosse executado.

A segunda técnica proposta, *CRC na origem*, tem por objetivo economizar a área

da rede. A técnica consiste em colocar o codificador/decodificador CRC na interface de rede (NI) do módulo conectado à porta local do roteador que origina os dados. Desta maneira não é necessário que cada porta de saída do roteador tenha um codificador CRC. Contudo, os bits de CRC têm que ser incluídos no *flit*, sendo necessário assim um aumento da largura de todos os buffers da rede, o que ocasiona ao final do processo um aumento de área. A verificação do CRC é executada em cada porta de entrada da interface de rede, com isso quatro módulos de CRC são economizados na área do roteador.

A terceira técnica proposta, *utilização de códigos de Hamming nos enlaces*, buscando corrigir 1 erro de bit em caso de *crosstalk*. Códigos de *Hamming* são códigos lineares desenvolvidos para a correção de erros, frequentemente utilizados para o controle de erros em aplicações de comunicação digital e em sistemas de armazenamento de dados. Eles têm a capacidade de corrigir apenas um erro por bloco e tem por vantagem serem simples de decodificar. No caso da utilização de códigos de *Hamming* não foi utilizado nenhum mecanismo de reenvio, pois o código de *Hamming* não provê nenhuma forma de sinalizar se houve erro em mais de um bit. A Figura 12 ilustra a arquitetura interna dos roteadores com a utilização do codificador/decodificador com códigos de *Hamming*.

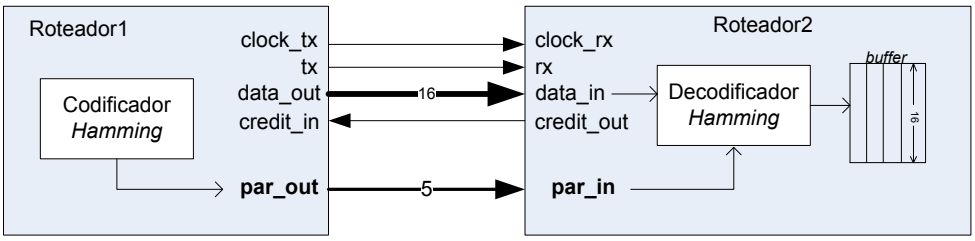

Figura 12 – Interface entre dois roteadores de uma rede com Hamming nos enlaces [SIL10].

O processo de decodificação adotado na arquitetura é transparente para o roteador, pois não consome nenhum ciclo adicional de relógio. Quando o dado chega à porta de entrada ele é repassado ao decodificador *Hamming*. O dado é verificado e o corrige caso um bit tenha sido alterado. O buffer recebe o dado já decodificado fazendo com que a latência da rede não se altere. A cobertura de falhas do código de *Hamming* é muito menor do que a cobertura do CRC, pois se houver mais de um bit com erro ele não consegue detectar, e por consequência o reenvio do pacote não é possível de ser solicitado e o dado é repassado com erro.

A quarta e última técnica desenvolvida foi a de *Hamming na origem*, tendo por objetivo a tolerância de falhas interna ao roteador e a correção de erros simples nos enlaces. Conforme ilustrado na Figura 13, é utilizado código *Hamming* na interface de rede que origina os flits. Na interface de entrada e saída de cada roteador também é utilizado *Hamming*, possibilitando a correção de falhas nos enlaces e de falhas internas nos buffers.

Devido à inclusão dos Codificadores *Hamming* em cada porta de entrada e saída do roteador e na interface local, a rede tem um grande aumento em sua área. Contudo, esta rede oferece a maior cobertura de falhas nos testes relatados no trabalho utilizando

apenas códigos de *Hamming*. A única falha não detectada nesta arquitetura é caso ocorra à inversão de um bit em uma das tabelas de roteamento do *crossbar*. Caso isso ocorra o pacote pode ser encaminhado para um caminho errado, o que pode causar um *deadlock* na rede.

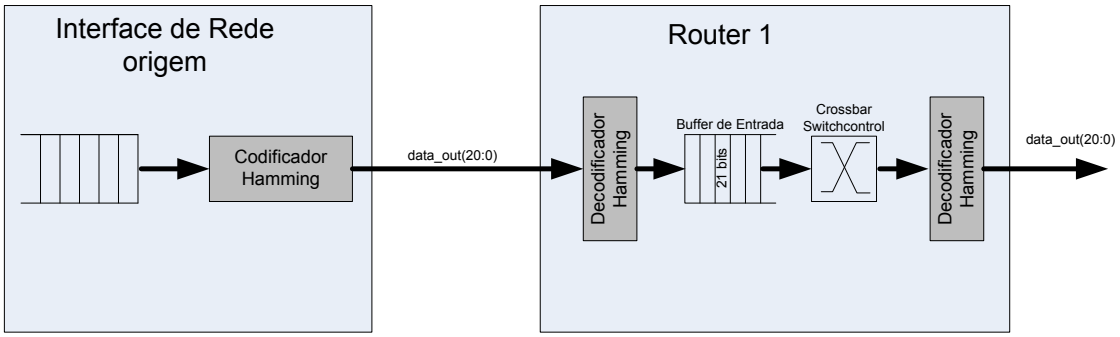

Figura 13 - Diagrama simplificado de uma rede com Hamming na origem [SIL10].

Os resultados encontrados foram que utilizando CRC nos enlaces o aumento de área foi de 12% contra 26% utilizando CRC na origem. A diferença de 14% deve-se ao fato que mesmo não utilizando CRC nas portas de saída o tamanho do buffer foi aumentado gerando assim uma maior área da rede. A utilização de *Hamming* nos enlaces gerou um aumento de 1% na área da rede, a técnica de *Hamming* na origem e nos enlaces gerou um aumento de 49% na rede, devido aos codificadores acrescentados na origem e em cada porta de entrada e saída, porém esta é a arquitetura que apresenta maior cobertura de falhas transientes utilizando apenas códigos de *Hamming*..

1.8 Algoritmos de roteamento tolerante a falhas

O objetivo de técnicas de tolerância a falhas em nível de algoritmo de roteamento é prover, tanto em nível de hardware quanto de software, mecanismos que permitam a NoC encontrar um caminho alternativo na ocorrência de falhas.

1.8.1 Topology-Agnostic Fault-Tolerant NoC Routing Method

Wachter et al. [WAC13] propõem uma técnica de roteamento com foco em falhas permanentes, detectadas durante a execução do sistema. A técnica engloba topologias de rede regulares e irregulares e é independente em relação ao tamanho da NoC. A falha é detectada em tempo de execução utilizando algoritmos de teste, como BIST (fora do escopo deste trabalho).

O algoritmo de roteamento proposto inicia considerando que a NoC está sem falhas, sendo os pacotes enviados utilizando o algoritmo XY de roteamento. Uma vez que o pacote alcança o destino, é enviada uma confirmação para a origem avisando que o pacote foi recebido sem erros. Caso a confirmação não seja recebida, a origem considera que o destino é inalcançável. O algoritmo de roteamento é responsável pela procura de um novo caminho válido, explicado a seguir. Uma vez encontrado o novo caminho ele é registrado na memória do PE. O método é executado apenas uma vez a cada mensagem de confirmação perdida, evitando assim que a cada pacote o método seja reexecutado.

O método proposto inicia ao detectar um destino inalcançável. O método possui um

módulo responsável pela transmissão da requisição, este módulo será melhor explicado nos parágrafos seguintes. O primeiro passo do processo é denominado de *seek step.* Uma requisição é enviada a todos os roteadores vizinhos à origem, buscando alcançar o destino. Em cada passo da busca os seguintes dados são armazenados nos roteadores: S/T/P/# (origem, destino, porta de entrada no roteador e número de saltos). Cada roteador ao receber a requisição verifica na sua tabela de roteamento se é o destino solicitado. Caso não seja, é incrementado o número de saltos e repassada à requisição para todos os roteadores vizinhos, com exceção da porta de origem da requisição. O processo é repetido roteador a roteador, independentemente de se encontrar o destino. O processo é semelhante a uma onda que parte da origem da requisição, visitando todos os roteadores da rede.

A Figura 14, ilustra um cenário com quatro roteadores falhos, um roteador origem e um roteador destino. Inicialmente o roteador origem envia uma requisição para encontrar um caminho válido até o destino (Figura 14(b)) para o único roteador vizinho não falho. O processo é repetido (Figura 14(c)), até se alcançar o roteador alvo (Figura 14(d)). Notar que o processo de propagação da "onda" visita todos os roteadores da rede, independentemente se o destino foi ou não alcançado. Este processo é realizado desta forma para se reduzir o controle do processo de propagação.

O segundo passo do método é o processo de regresso. Este processo consiste em retornar do destino à origem, utilizando o caminho livre de falhas encontrado durante o primeiro passo. O pacote de regresso ao alcançar o roteador origem terá a informação das voltas necessárias para alcançar o destino.

Figura 14 – Demonstração do método proposto por [WAC13] em uma NoC 4x4.

O terceiro passo do método é determinar um caminho livre de *deadlocks*. O PE ao receber a informação do caminho, armazena este na tabela de roteamento do PE para futuras transmissões, utilizando roteamento na origem. *Deadlocks* são evitados utilizando dois algoritmos de roteamento, *west-first* e *east-first*, cada um em um canal físico distinto. Em paralelo ao cálculo do caminho livre de *deadlocks*, o PE origem solicita que as tabelas utilizadas na etapa um (*seek*) sejam limpas, para permitir futuras buscas de caminhos.

Ao realizar o broadcast na rede para enviar as requisições (*seeks*) pode-se gerar um congestionamento na rede, causando uma queda de desempenho nas aplicações. Por essa razão, o Autor sugere a criação de um módulo de roteamento específico para quando houver requisições, conforme ilustrado na Figura 15.

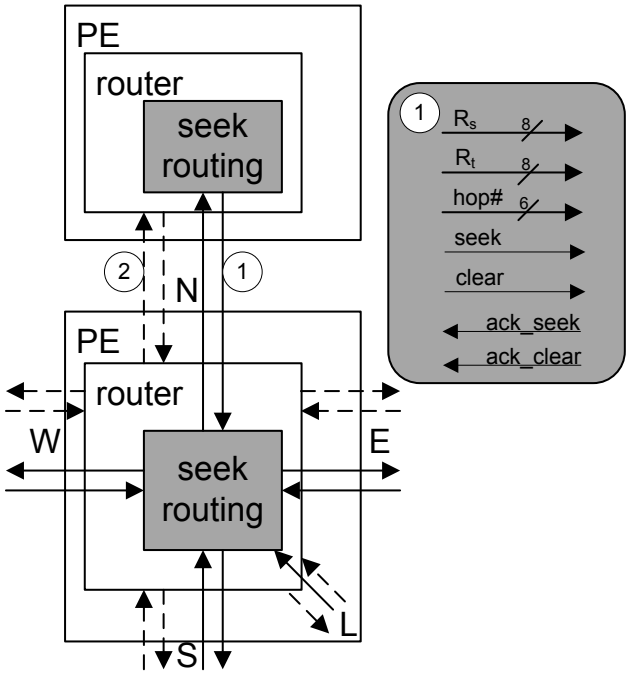

Figura 15 – Conexões entre os roteadores e o modulo de roteamento [WAC13].

A Figura 15 ilustra como o módulo adicional é conectado entre os roteadores vizinhos, criando-se duas redes disjuntas. A rede convencional é representada através das linhas tracejadas, e a rede específica para o módulo de requisição através das linhas contínuas. Os enlaces do módulo de requisição são utilizados apenas quando uma requisição é solicitada ou para fazer a limpeza das tabelas de roteamento. Desta maneira, o congestionamento da NoC é evitado, mantendo o desempenho da rede durante os processos de requisição.

Os resultados encontrados foram que uma área adicional de 10% do PE foi gerada, porém com um grande ganho de confiabilidade em diversos cenários, topologias de rede e diferentes dimensões de NoC. A prioridade do método consiste na busca por um caminho livre de falhas utilizando um módulo ligado a uma rede dedicada para a busca de um caminho livre de falhas. Em termos de desempenho de comunicação a latência teve um acréscimo não significativo, menor que 1%, devido ao uso do módulo de roteamento adicional. A área adicional no roteador para a inclusão do modulo de roteamento foi de aproximadamente 50%.

1.8.2 A Fault-Tolerant and Hierarchical Routing Algorithm for NoC Architectures

Valinataj et al. [VAL11] sugerem um algoritmo hierárquico de roteamento tolerante a falhas. O algoritmo utiliza reconfiguração dinâmica para lidar com falhas permanentes. A cada reconfiguração o algoritmo cria uma nova rota de roteamento para cada pacote sem utilizar canais virtuais. O método seleciona o caminho de cada pacote a cada roteador com base nas informações locais dos enlaces adjacentes que são armazenadas em um registrador local.

O método pressupõe que a NoC passe por um teste *off-line* para detectar e identificar os enlaces falhos. O método possui três níveis de tolerância a falhas: no primeiro, FT_XY, o algoritmo pode lidar com falhas que ocorrem em um enlace; no segundo nível, RDR1, o algoritmo consegue lidar com múltiplos enlaces falhos; no terceiro nível, RDR2, o algoritmo considera falhas nos roteadores.

Os Autores utilizaram como base o algoritmo de roteamento XY para desenvolver o novo método. Quando um enlace falho é detectado, um novo caminho é definido usando a menor distância possível entre os roteadores se nenhuma volta proibida for violada. Caso contrário, o novo caminho poderá ser mais longo, evitando as voltas proibidas. O registrador interno utilizado para armazenar o estado dos enlaces falhos utiliza quatro bits para armazenar esta informação. O algoritmo de roteamento FT_XY é livre de *deadlock*, pois utiliza os mesmos princípios de proibição de voltas utilizados no XY. A diferença entre o XY e o FT_XY é que o FT_XY após os testes realizados para detecção de falhas é feito o cálculo do caminho dos pacotes para os roteadores vizinhos utilizando a política de voltas do algoritmo XY. O Autor utiliza tabelas de caminhos alternativos para contornar um enlace falho.

O método RDR1 estende as características utilizadas no algoritmo FT_XY para tolerar falhas em múltiplos enlaces. Quando dois enlaces falham, coincidindo com o novo caminho do pacote, o algoritmo FT_XY não é capaz de corrigir a falha e o pacote pode ir para um caminho errado. Existem trinta situações diferentes onde duas falhas podem ocorrer em uma NoC de topologia malha. As situações foram agrupadas em oito grupos conforme ilustrado na Figura 16.

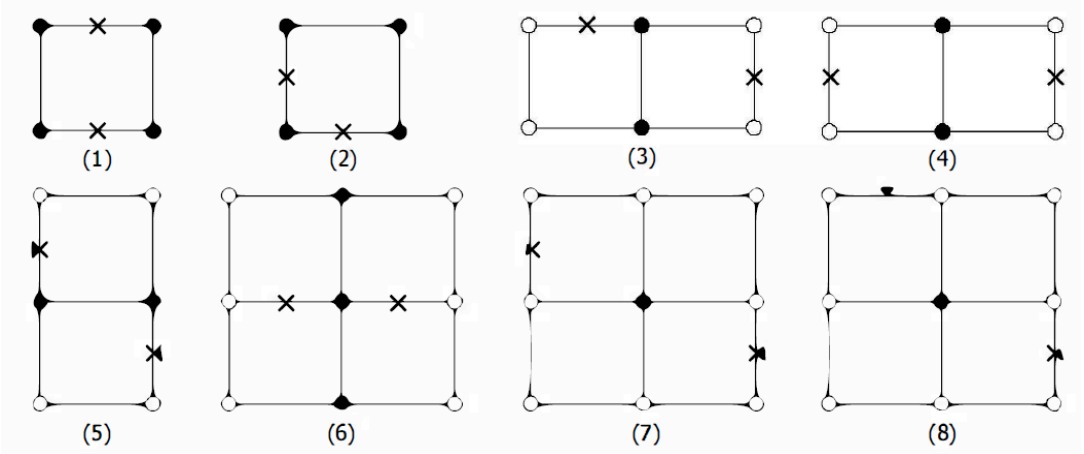

Figura 16 – Oito diferentes grupos de situações onde ocorre duas falhas simultâneas em uma NoC [VAL11].

As falhas em dois enlaces podem ser toleradas sem a utilização de canais virtuais se forem apropriadamente consideradas. Os Autores modificaram o algoritmo FT_XY para que ele conseguisse lidar com um número maior de falhas criando assim o algoritmo RDR1. O método RDR1 pode tolerar quinze situações de falhas e dez situações parcialmente sem o uso de canais virtuais. A diferença entre os dois níveis de tolerância é o tamanho de configuração dos registradores. O registrador que armazena as informações de falha passa a ter oito bits.

A Figura 17 ilustra três caminhos obtidos utilizando o algoritmo em uma rede com muitos enlaces em falha. A origem S1 com destino D1, origem S2 e destino D2 e origem S3 e destino D3. Os roteadores destacados em preto são comuns aos três caminhos, pois são usados para contornar os enlaces em falha.

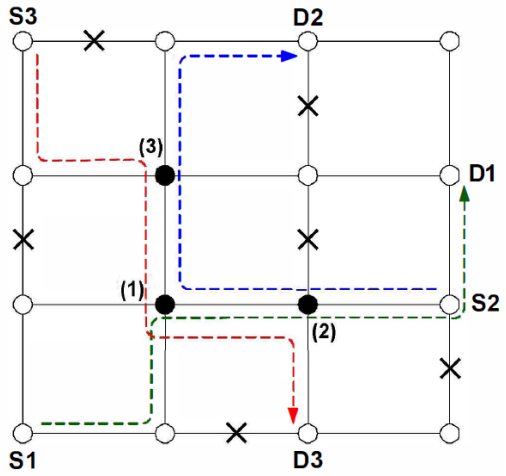

Figura 17 – Caminho obtido através do algoritmo RDR1 em uma rede com 25% de enlaces com falha [VAL11].

O terceiro algoritmo de roteamento RDR2 é similar ao RDR1, com a diferença que pode tolerar roteadores com falhas. Caso ocorra alguma falha permanente no roteador e ele não consiga encaminhar os pacotes corretamente, o algoritmo assume que os quatro enlaces do roteador estão falhos. Os registradores dos roteadores vizinhos têm seus dados atualizados para que esses enlaces não sejam utilizados.

Os resultados reportam um aumento na área do roteador de 16,2% para o algoritmo FT_XY, 31,8% para o RDR1 e de 40% para o RDR2. O método apresentado é adequado para NoCs que necessitam de tolerância a falhas e não podem ter um aumento muito significativo de área.

1.9 Considerações finais

Foram apresentados trabalhos com objetivos distintos, técnicas de tolerância a falhas em diferentes níveis: sistêmico [MEL12], roteador [FIC09], enlace [SIL10] e algoritmos de roteamento [VAL11]. Em [FIC09] ilustra uma modificação no roteador para que a rede continue operando mesmo na presença de muitas falhas. [SIL10] módulos de CRC são uma boa alternativa para detecção de falhas. Cada uma das técnicas busca tratar uma falha específica ou seja tratando apenas uma dimensão do problema. O trabalho de [WAC13] aborda tanto o roteamento de pacotes como o nível sistêmico

(comunicação entre PEs). Os trabalhos [FIC09], [SIL10] e [WAC13] serviram de base para o desenvolvimento das técnicas que serão explicadas nos capítulos seguintes.

Buscando criar uma abordagem completa para técnicas de tolerância a falhas, é proposta neste trabalho uma solução para tolerância a falhas em nível de roteador e enlace. Estas técnicas integram-se com o trabalho proposto por [WAC13] visto que o trabalho de [WAC13] é focado nas camadas de comunicação de enlace e rede e futuramente na camada de aplicação. A presente Dissertação é voltada para a camada física e de enlace, proporcionando assim tolerância a falhas em diversos níveis de comunicação.

A Tabela 2 apresenta os níveis OSI (sem os níveis de apresentação e de sessão), e para cada nível citam-se características importantes da arquitetura adotada no presente trabalho (segunda coluna). A terceira coluna apresenta os recursos adicionados em cada nível para se prover tolerância a falhas. Notar que o nível superior, o nível de aplicação, não requer modificações no código do usuário. O mesmo código projetado para a plataforma original pode ser utilizado na plataforma tolerante a falhas.

Tabela 2 - Vista simplificada do modelo OSI e recursos de tolerância a falhas.

Nível	**Arquitetura**	**Recursos adicionados de tolerância a falhas**
Física	Canais físicos duplicador por enlace (16- bit flit)	CRC adicionado nas portas de entrada do roteador
Enlace	Controle de fluxo baseado em créditos	Test wrappers; Descarte de pacotes
Rede	Roteamento adaptativo	NoC auxiliar; controle para operar em modo degradado; busca por um caminho livre de falhas
Transporte	Troca de mensagens	protocolo de comunicação tolerante falhas; tetransmissão de pacotes
Aplicação	API de comunicação	Não alterada

Cada nível tem as seguintes características:

- Nível Físico: Consiste em dois canais físicos com o objetivo de manter a comunicação em caso de falha em um dos canais. Foram adicionados módulos de CRC para detectar falhas nos canais e no buffer de entrada.
- Nível de Enlace: O controle de fluxo é sincronizado e baseado em crédito. Inclui-se *test wrappers* (células de isolação) para que em caso de falhas o canal seja isolado para que dados corrompidos não sejam propagados. Após a detecção de uma falha é realizado o descarte dos pacotes provenientes do canal ou buffer com falha.
- Nível de Rede: O algoritmo de roteamento padrão na ausência de falhas é o XY. Caso seja detectado uma falha no enlace, o algoritmo de roteamento torna-se *source*

routing (roteamento na origem). Uma rede auxiliar foi incluída para realizar a busca por um caminho livre de falhas. E uma lógica adicional no roteador foi desenvolvida para fazer o roteador operar em modo degradado em caso de falha.

- Nível de Transporte: Responsável por transmitir os pacotes. Foi adicionado um mecanismo de retransmissão em caso de falhas.
- Nível de Aplicação: API de comunicação, tarefas produtoras e consumidoras trocando mensagens. Não foi alterada.

A presente Dissertação contribui nas técnicas de nível físico (inclusão dos módulos CRC aproveitados do trabalho [SIL10]), rede (operação em modo degradado) e transporte (retransmissão de pacotes). Além destas contribuições, desenvolveu-se um método para recuperação do roteador falho em caso de falha transiente.

PLATAFORMA DE REFERÊNCIA – MPSOC HEMPS

Neste Capítulo é apresentada a arquitetura e a estrutura de comunicação entre tarefas do MPSoC homogêneo HeMPS [CAR09]. Na sequência menciona-se as modificações realizadas por [WAC13], relacionadas à implementação do roteamento tolerante a falhas. É esta plataforma, derivada da HeMPS, que será utilizada como referência neste trabalho.

1.10 MPSoC HeMPS

O MPSoC HeMPS [WOS07][CAR09] é composto por um conjunto de PEs (elementos de processamento) interconectados por uma NoC do tipo malha. Cada PE pode executar múltiplas tarefas, e também se comunicar com os vizinhos através da NoC. Na Figura 18 é ilustrada uma instância da HeMPS, utilizando uma NoC Hermes com canais físicos duplicados, com dimensão 2x3, interconectando os Plasmas-IP.

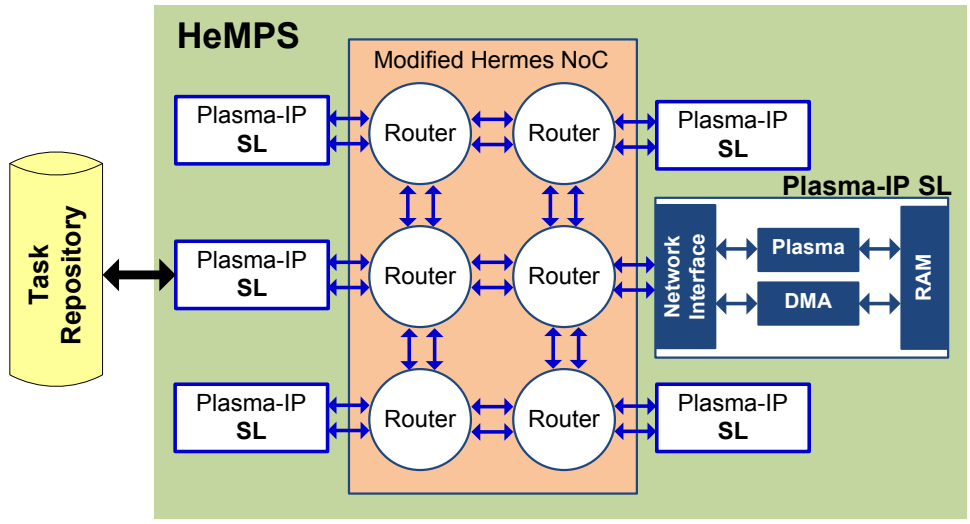

Figura 18 – Instância do MPSoC HeMPS com 9 elementos de processamento Plasma-IP.

1.10.1 Plasma-IP

Os elementos de processamento do MPSoC HeMPS, denominados Plasma-IP, assumem as funções de: mestre, chamado de *Plasma-IP MP*, responsáveis pela gerência dos recursos do sistema; e escravos, chamados de *Plasma-IP SL*, responsáveis por executar tarefas de usuário. O MPSoC HeMPS utilizado na presente Dissertação adota gerência de recursos centralizada, contendo apenas um Plasma-IP MP. Os componentes internos do Plasma-IP compreendem:

- um processador Plasma [OPE10]: processador RISC de 32 bits com subconjunto de instruções da arquitetura MIPS. Diferentemente do MIPS original, o Plasma apresenta uma organização de memória Von Neumann. Além disso, o processador ainda oferece suporte a linguagem C e tratamento de interrupções. O Plasma do MPSoC HeMPS possui algumas modificações em relação ao Plasma original, como por exemplo, exclusão de módulos (como UART), e inclusão de novos registradores mapeados em memória para acesso à NoC e temporizadores utilizados pelo sistema operacional.

- uma memória privada: contém o sistema operacional executado pelo processador Plasma e as tarefas da aplicação do usuário. No caso dos Plasma-IP SL, a memória é dividida em páginas de tamanho fixo onde é feita a alocação de tarefas. É importante mencionar que uma tarefa utiliza apenas uma página de memória.
- uma interface de rede (do inglês, Network Interface – NI) : realiza a interface entre o Plasma e a NoC Hermes. É responsável pelo envio e recebimento de pacotes na rede.
- um módulo de acesso direto à memória (do inglês, Direct Memory Access - DMA): desenvolvido para auxiliar o Plasma na troca de mensagens com a NI, possibilitando ao processador continuar sua execução de tarefas sem controlar diretamente a troca de mensagens com a rede. O DMA permite a recepção e o envio de grandes pacotes, como código-objeto de tarefas ou pacotes oriundos de outras tarefas.

1.10.2 NoC Hermes

A interconexão dos elementos de processamento do MPSoC HeMPS é realizada através da NoC Hermes, versão QoS, com características para atendimento a qualidade de serviço [CAR09]. Esta NoC é parametrizável e possui topologia malha 2D. O mecanismo de comunicação é realizado por chaveamento de pacotes, utilizando modo de roteamento *wormhole*, no qual um pacote é transmitido entre os roteadores através de *flits*.

Conforme a Figura 19 os roteadores da NoC possuem buffers de entrada, uma lógica de controle compartilhada por todas as portas do roteador (*switch control*), um crossbar interno de dimensão 10x10, e até cinco portas bi-direcionais. Estas portas são: East, West, North, South e Local. A porta Local estabelece a comunicação entre o roteador e seu núcleo local, sendo as demais portas utilizadas para ligar o roteador aos roteadores vizinhos. A arbitragem utilizada pelo roteador é *round-robin*. Essa política utiliza um esquema com prioridades dinâmicas, i.e., após uma porta ter sua requisição atendida, esta porta terá prioridade mínima na próxima avaliação de requisições de arbitragem. Este método proporciona um serviço mais justo que a prioridade estática.

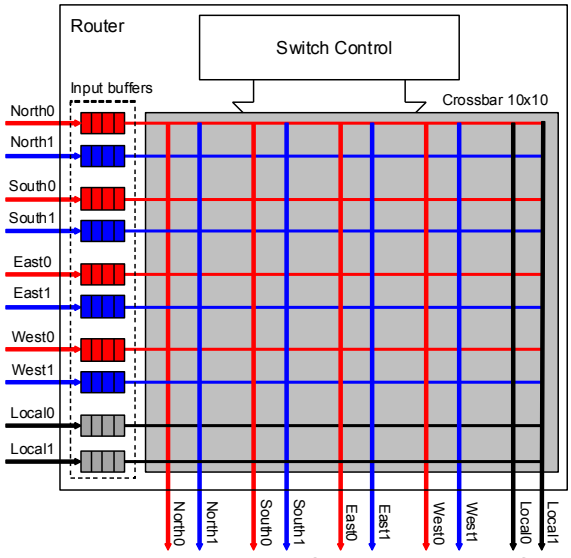

Figura 19 - Roteator da NoC proposta em [CAR09].

Observar que cada porta possui dois canais físicos. Esta é uma importante característica explorada ao longo deste trabalho, pois acrescenta-se redundância física em cada porta. Havendo um canal falho, pode-se continuar a operar em modo degradado, transmitindo-se toda a informação daquela porta pelo mesmo canal físico, o qual está isento de falhas.

1.10.3 Repositório de Tarefas

O repositório de tarefas (tas*k repository*, na Figura 18) é uma memória externa ao MPSoC, contendo o código-objeto de todas as tarefas que executarão no sistema. O Plasma-IP MP é o único PE a ter acesso ao repositório de tarefas e é responsável por alocar as tarefas nos Plasma-IP SL. Todas as tarefas contidas no repositório são armazenadas em tempo de projeto, no momento da geração da plataforma.

1.11 Comunicação entre tarefas

Em nível de software, cada processador executa um sistema operacional denominado *microkernel*, responsável pela comunicação entre tarefas, serviços de gerenciamento do processador e execução multitarefa. O método de comunicação é o de troca de mensagens (do tipo MPI), uma vez que é mais adequado para os sistemas distribuídos com memória local. As aplicações são modeladas na forma de um grafo de tarefas A=<T,C> (exemplo na Figura 20), onde T = $\{t_1, t_2, ..., t_m\}$ é o conjunto de tarefas da aplicação, correspondente aos vértices do grafos e C= $\{(t_i, t_j, w_{ij}) \mid (t_i, t_j) \in T \text{ e } w_{ij} \in \mathbb{N}^*\}$ representa as comunicações entre tarefas, o que corresponde às arestas do grafo (o termo w_{ij} representa o volume de comunicação entre as tarefas t_i, e t_j). O MPSoC assume que há uma memória externa, denominada *repositório de tarefas*, com todas as tarefas (conjunto T) que serão inicializados no sistema em tempo de execução.

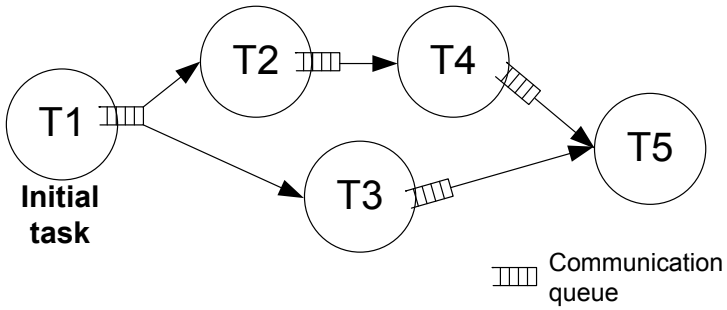

Figura 20 – Aplicação modelada como um grafo de tarefas, com as filas de comunicação.

A comunicação entre tarefas é realizada através de duas primitivas do tipo *MPI*: *Send*() não bloqueante, e *Receive*() bloqueante. A principal vantagem dessa abordagem é que a mensagem somente é enviada para a NoC se o destino requisita o dado, reduzindo assim o congestionamento da rede. Para implementar a primitiva *Send()* não bloqueante é necessário um espaço de memória dedicado no *microkernel*, denominado *pipe,* que armazena cada mensagem escrita pelas tarefas.

A Figura 21 ilustra a comunicação entre duas tarefas, A e B, mapeadas em diferentes PEs. Quando a tarefa A executa o *Send()* (1 na Figura), a mensagem é armazenada no *pipe*, e a computação continua (2). Esta é uma operação não bloqueante

de escrita. Se a tarefa da qual a mensagem foi requisitada está localizada no mesmo processador, a tarefa é lida diretamente do *pipe*. Dado que neste exemplo a tarefa B está mapeada em outro processador, quando a primitiva *Receive()* (3) é executada, o *microkernel* envia a requisição de mensagem através da NoC (4). O escalonador indica para a tarefa B ir para o estado de *wait* (5), caracterizando uma leitura bloqueante. Quando a tarefa A recebe a requisição da mensagem (4), o *microkernel* escreve no slot do pipe que o mesmo está vazio (6) e envia a mensagem através da NoC. A mensagem chega à tarefa B (7), e assim o *microkernel* armazena a mensagem no espaço de memória da tarefa B e retorna para o estado de execução (8).

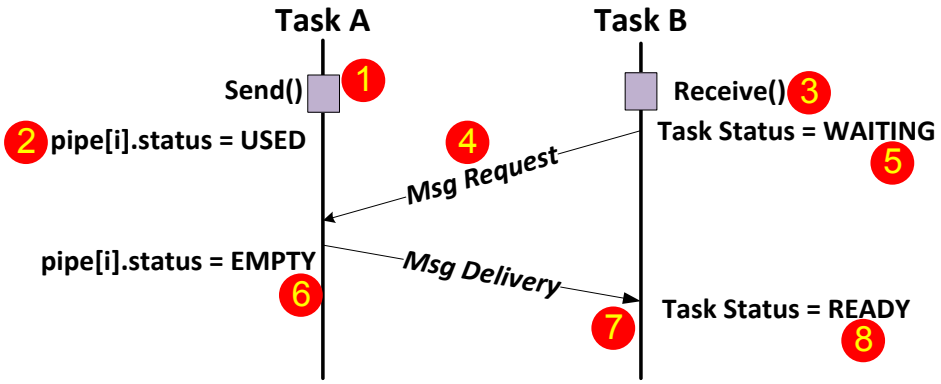

Figura 21 - Protocolo de comunicação adotado na plataforma de referência.

Neste exemplo, podemos observar que o protocolo é suscetível a falhas, em qualquer um dos passos da comunicação. Por exemplo, algumas tarefas podem aguardar indefinidamente por mensagens caso ocorra uma falha na NoC durante o envio da mensagem (etapa 7 da Figura 21). O presente trabalho modifica este protocolo para torná-lo tolerante a falhas.

1.12 Serviços de microkernel

A Figura 21 mostrou que a troca de mensagens entre duas tarefas necessitou de dois pacotes, um do tipo *msg_request* e outro *msg_delivery*. Estes pacotes são gerados pelo microkernel, e sua semântica representa o *serviço* a ser executado. Cada protocolo executado pelo microkernel requer um conjunto de serviços.

Exemplos de protocolos executados pelo microkernel:

- Comunicação entre tarefas: requer os serviços *message_request* e *message_delivery*;
- Mapeamento de tarefas: requer os serviços *task_request*, *task_allocation*, *task_allocated*;
- Gerenciamento de tarefas: requer os serviços *new_task*, *location_request*, *location_answer, task_deallocated*, *task_terminated*.

A Tabela 3 ilustra os principais serviços suportados pela plataforma HeMPS, assim como a descrição dos mesmos.

Tabela 3 – Descrição dos principais serviços suportados pelo *µkernel* da HeMPS.

Serviço	Código	Descrição
REQUEST_MESSAGE	0x00000010	Requisição de uma mensagem
DELIVER_MESSAGE	0x00000020	Entrega de uma mensagem previamente solicitada
NO_MESSAGE	0x00000030	Aviso de que a mensagem solicitada não existe
TASK_ALLOCATION	0x00000040	Alocação de tarefas: uma tarefa deve ser transferida pelo DMA para a memória do processador
ALLOCATED_TASK	0x00000050	Aviso de que uma nova tarefa está alocada no sistema
REQUEST_TASK	0x00000060	Requisição de uma tarefa
TERMINATED_TASK	0x00000070	Aviso de que uma tarefa terminou a sua execução
DEALLOCATED_TASK	0x00000080	Aviso de que uma tarefa terminou a sua execução e pode ser liberada
FINISHED_ALLOCATION	0x00000090	Aviso que o nodo mestre terminou a alocação inicial das tarefas (alocação estática)

Para o atendimento do objetivo específico *protocolo de comunicação tolerante a falhas* é necessário modificar o microkernel original, acrescentando novos serviços, os quais serão detalhados na seção 1.18.

1.13 Considerações sobre o modelo da NoC

Esta plataforma de referência foi modificada por [WAC12] para utilizar dois algoritmos de roteamento parcialmente adaptativos quando ocorrem falhas na NoC: no canal 0 West-First, e no canal 1 East-First. Como resultado, temos duas redes disjuntas, com roteamento totalmente adaptativo. Isto significa que esta rede é capaz de efetuar qualquer curva (*turn*), utilizado em conjunto ambos os algoritmos de roteamento, porém respeitando a utilização dos canais físicos. Desta forma, assegura-se que não haverá a ocorrência de *deadlocks*. É esta característica, canais físicos duplicados, que permite implementar as técnicas propostas neste trabalho, como o suporte à operação do roteador em estado degradado, i.e., falha em um dos canais físicos.

A camada de rede assume que a camada acima (camada de transporte), inicia a busca por um caminho livre de falhas quando uma falha é detectada. Isto pode ser feito com um protocolo de comunicação tolerante a falhas ou com módulos em hardware de detecção de falhas.

Ao iniciar o sistema, a rede assume que nenhuma falha existe e os pacotes são enviados do seu PE de origem até o PE de destino usando o algoritmo de roteamento XY. O método é executado em tempo de execução, conforme ilustrado na Figura 22 (processo inicialmente descrito na sessão 1.8.1): (*i*) busca por um novo caminho; (*ii*) retorno do novo caminho encontrado; (*iii*) limpeza das informações utilizadas nos módulos de busca de um novo caminho. Ao final do processo descrito, o PE de origem recebe um caminho livre de falhas até o PE destino. Este caminho é então armazenado na memória do PE de origem e os próximos pacotes (desta origem para o destino solicitado) utilizaram esse caminho. Portanto, este método é executado apenas para cada pacote perdido, sem a necessidade de fazer o cálculo para pacotes subsequentes.

Quando uma falha é detectada (em [WAC12] a detecção de falhas está fora do escopo de seu trabalho), um módulo denominado **PDN** (*path discover network)* busca um caminho livre de falhas, utilizando os algoritmos de roteamento West-First e East-First.

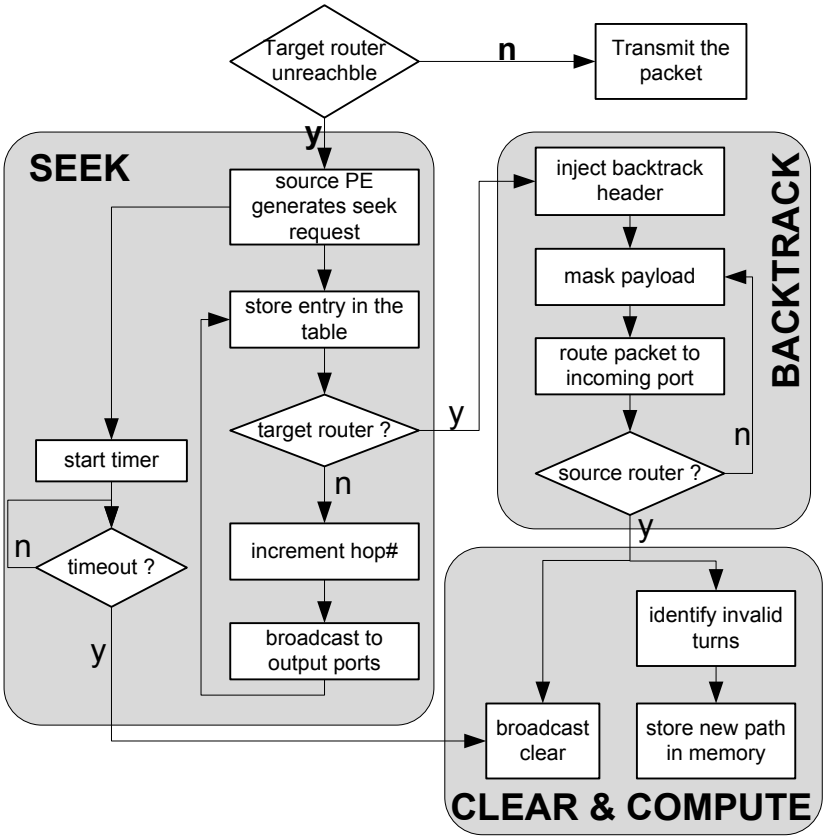

Figura 22 – Operação do método de roteamento proposto em [WAC12].

DETECÇÃO DE FALHAS NO ROTEADOR ATRAVÉS DE CRC

Este Capítulo apresenta a primeira contribuição da Dissertação. O objetivo é agregar detecção de falhas em uma NoC já existente, denominada MazeNoC [WAC13]. Esta NoC caracteriza-se por possuir um módulo denominado *PDN*, responsável principalmente por busca de caminhos alternativos em caso de roteador possuir uma porta de entrada com falha. O enlace da NoC é constituído por dois canais de envio, sendo um canal para mensagens de alta prioridade e outro canal para mensagens de baixa prioridade. A proposta tem por base a inclusão de CRC em cada flit transmitido com a finalidade de verificar a integridade dos dados transmitidos. A utilização de CRC garante cobertura de falhas de 93,75% para enlaces com 16 bits [SIL10].

É importante destacar que o método detecta falhas apenas nos flits transmitidos entre um roteador e outro. O método consegue detectar falhas no buffer de entrada e nos canais de comunicação analisando os dados na saída do buffer. A lógica de controle do roteador não está protegida pela técnica desenvolvida. Mecanismos adicionais de teste devem ser considerados posteriormente, como em [CON09], que utilizou TMR para as máquinas de estado que controlam as FIFOS.

1.14 Adição de módulos de CRC para detecção de falhas

Conforme a Tabela 2, apresentada na Introdução deste manuscrito, o presente Capítulo apresenta o trabalho realizado relacionado à tolerância a falhas no nível físico. A Figura 20 apresenta a proposta conceitual para agregar tolerância a falhas no roteador MazeNoC.

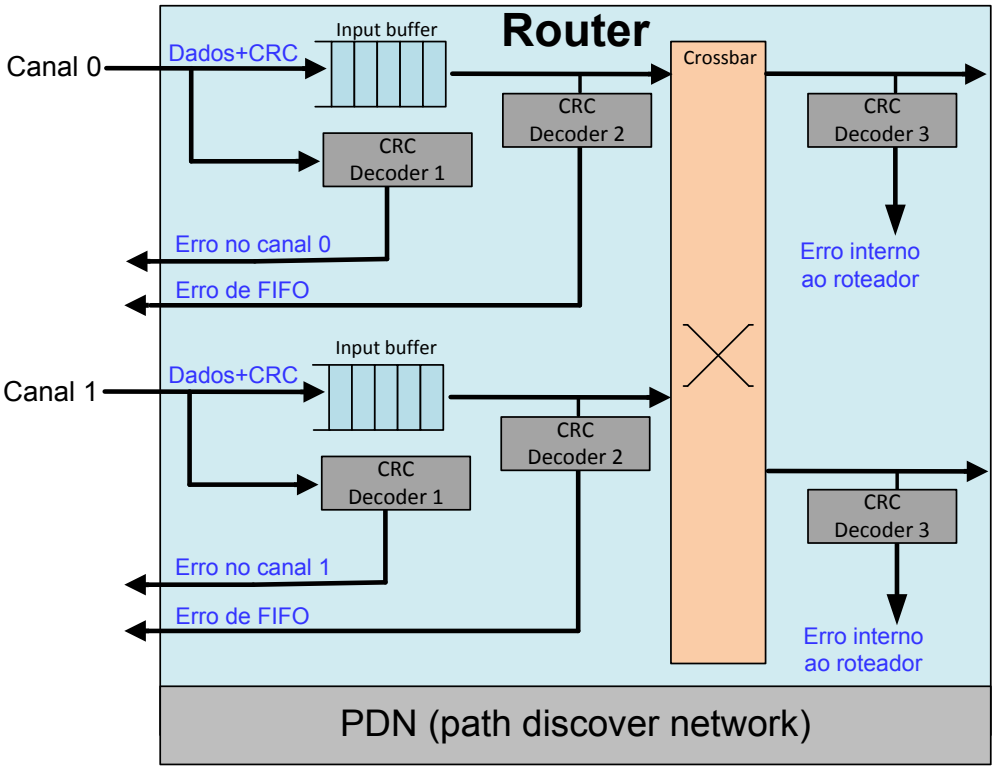

Figura 23 – Proposta de arquitetura para roteador com tolerância a falhas, para uma porta de entrada e uma porta de saída.

Os decodificadores possuem a seguinte função:

- o decodificador de CRC posicionado antes de o flit chegar ao buffer, CRC 1, tem por objetivo a detecção de falhas nos canais, buscando detectar falhas de *crosstalk* ou falhas permanentes nos canais (por exemplo, curto circuito).
- o decodificador de CRC posicionado na saída da FIFO, CRC 2, tem por objetivo detectar falhas nas FIFOS analisando os dados de saída da FIFO, (este trabalho tem por objetivo detectar que uma falha ocorreu e os dados foram corrompidos, o tipo de falha ou a localização exata da falha está fora do escopo do trabalho). O canal pode estar operando corretamente, mas um SEU (*single event upset*) pode ter alterado o estado de um dado bit armazenado nas FIFOS.
- o decodificar de CRC posicionado nas portas de saída, CRC 3, detecta falhas internas ao roteador. Uma vez detectada a falha por este módulo de CRC os roteadores vizinhos são notificados que o roteador está falho. Esta é uma falha crítica, pois pode ter havido falha no crossbar, ou na lógica da porta de saída. Neste caso, o roteador deve ser desabilitado, pois a integridade dos pacotes não poderá ser garantida caso uma falha assim seja detectada.

O procedimento padrão para tratamento de falhas é comum a todos os casos de detecção de falha por CRC:

- todos os *flits* do pacote são **descartados**, sendo sinalizado ao roteador que enviou o *flit* (denominado R_o) que a porta está falha;
- R_o notificará ao PE origem (denominado PE_o), através da rede *PDN*, que o pacote atual não pode ser entregue;
- o PE_O, através da *PDN*, realiza a busca por novo caminho, livre de falhas, e posteriormente reenvia o pacote.

Este procedimento padrão corresponde à primeira etapa da técnica de tolerância a falhas proposta: utilizar os decodificadores de CRC como lógica de BIST.

Para realizar esta primeira etapa do trabalho, alterou-se o tamanho do flit de 16 para 20 bits a fim de incluir os bits de CRC no pacote. O cálculo de CRC utiliza o polinômio $g = 1+X+X^4$, adicionando 4 bits da paridade, calculados segundo as equações abaixo [SIL10] (E indica o índice do bit no flit):

$$S_0 = E_{15} \oplus E_{11} \oplus E_8 \oplus E_7 \oplus E_5 \oplus E_3 \oplus E_2 \oplus E_1 \oplus E_0$$

$$S_1 = E_{12} \oplus E_9 \oplus E_8 \oplus E_6 \oplus E_4 \oplus E_3 \oplus E_2 \oplus E_1$$

$$S_2 = E_{13} \oplus E_{10} \oplus E_9 \oplus E_7 \oplus E_5 \oplus E_4 \oplus E_3 \oplus E_2$$

$$S_3 = E_{15} \oplus E_{14} \oplus E_{10} \oplus E_7 \oplus E_6 \oplus E_4 \oplus E_2 \oplus E_1 \oplus E_0$$

Desta forma, o codificador CRC consiste de 4 portas XOR de 8 ou 9 entradas. O decodificador utiliza o mesmo circuito do codificador, e compara os bits de paridade

recebidos com os bits calculados localmente. Se os valores, recebido e calculado, forem diferentes é sinalizada a presença de um erro.

Especificou-se inicialmente que a codificação CRC seria executada apenas na NI origem. Ao longo do caminho, cada roteador decodificaria cada flit para verificar a integridade do mesmo. A Figura 24 ilustra a arquitetura do roteador, com a inclusão dos codificadores nas portas de saída.

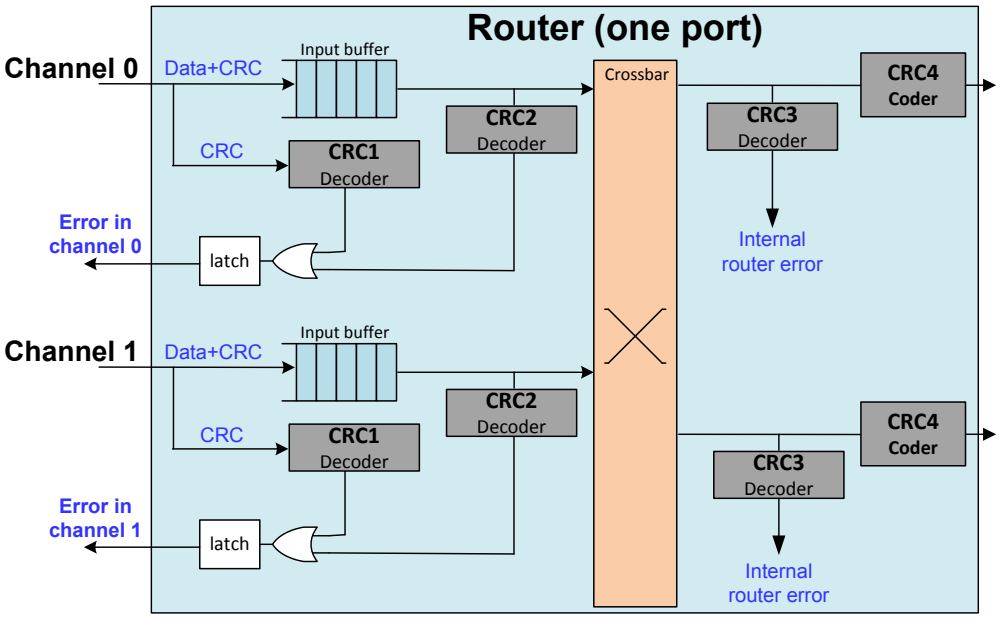

Figura 24 - Arquitetura do roteador, com os módulos de decodificação e codificação de CRC.

A primeira etapa do trabalho consistiu em utilizar CRC apenas como lógica para detecção de falhas. Após a validação da utilização de CRC como lógica para detecção de falhas, passou-se à **segunda etapa** do trabalho, com a operação do roteador em estado *degradado*, i.e., continuar a transmissão de pacotes sem a necessidade de utilizar a rede *PDN*, mesmo que um dos canais físicos apresente falha. Notar que como não há busca por novo caminho, o roteamento continua a ser o XY, havendo apenas troca de canal no(s) roteador(es) com porta falha. Assim, garante-se também ausência de *deadlock*, dado que o algoritmo XY é livre de *deadlocks*. O procedimento neste caso inclui:

- todos os *flits* do pacote são **descartados**, sendo sinalizado ao roteador que enviou o *flit* (R_o) que a porta está falha;
- R_o poderá tomar duas decisões:
  - se há um canal físico operando corretamente, o roteador é reconfigurado para utilizar apenas um canal ao invés de dois, e o roteador solicita ao PE origem (PE_O), através da *PDN* apenas o reenvio do pacote.
  - se ambos os canais físicos estão falhos, procede-se como originalmente definido em [WAC12], ou seja, busca por caminho livre de falhas e utilização de roteamento na origem.

A proposta de utilizar o roteador em modo degradado diminui o número de buscas por caminhos alternativos, assim como o tempo empregado para o cálculo de caminhos válidos. Os resultados são apresentados no Capítulo 0.

Notar que seria possível compartilhar o buffer livre de falhas caso a falha tenha ocorrido no enlace. Este compartilhamento implicaria na utilização de multiplexadores adicionais na entrada de cada buffer, permitindo que cada buffer receba *flits* de ambos os canais físicos. Estes multiplexadores representariam um considerável consumo de área (seriam necessários 8 multiplexadores 2x1 com largura de 20 bits), acrescentariam um atraso adicional para o armazenamento dos dados de entrada, além da lógica de controle para os mesmos. Assim, por estas razões, o presente trabalho descarta este compartilhamento de buffers.

1.15 Adição de módulos de isolação de portas de entrada com falha

Conforme a Tabela 2, apresentada na Introdução deste manuscrito, a presente Seção apresenta o trabalho realizado relacionado à tolerância a falhas no nível de enlace.

Em MPSoCs, é necessário a utilização de *test wrappers* para que se possa isolar um determinado componente quando necessário. Conforme apresentado na Figura 25, temos os sinais de controle (test, normal_mode, fault), cada um controlando um dos modos de operação da célula de isolação:

- *Normal_mode* : a célula de isolação é transparente para o PE.
- *Test*: usado durante o teste do PE.
- Fault: Neste modo a célula de isolação não permite que nenhum, sinal seja transmitido.

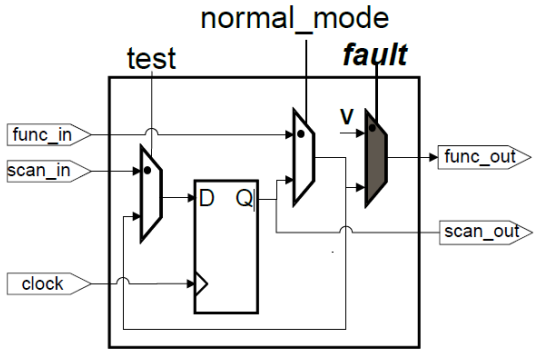

Figura 25 - Arquitetura da célula de *test wrapper* [WAC12].

Para o propósito deste trabalho, a célula de isolação é usada apenas nos modos: *normal_mode* (sem a presença de falhas) e *fault* (quando uma falha é detectada).

A Figura 26 ilustra os módulos responsáveis por isolar um determinado canal após a detecção de uma falha (por simplicidade apenas o CRC1 é apresentando na Figura). O decodificador CRC detecta a falha e a sinaliza para o módulo TW (test wrapper) e também para o roteador adjacente. O módulo TW bloqueia a chegada de novos dados com falha para o roteador fazendo com que o erro não se propague para outros roteadores.

O módulo TW é também responsável por forçar o sinal de crédito em nível lógico ‘1’ (sinal que sinaliza que dados podem ser enviados) ao roteador vizinho. Assim, em caso

de falha o roteador vizinho ao roteador com a porta de entrada com falha pode continuar a enviar os flits do pacote, esvaziando o buffer. É este mecanismo simples que possibilita o descarte de pacotes, evitando-se que flits fiquem bloqueados nos buffers quando uma falha ocorre.

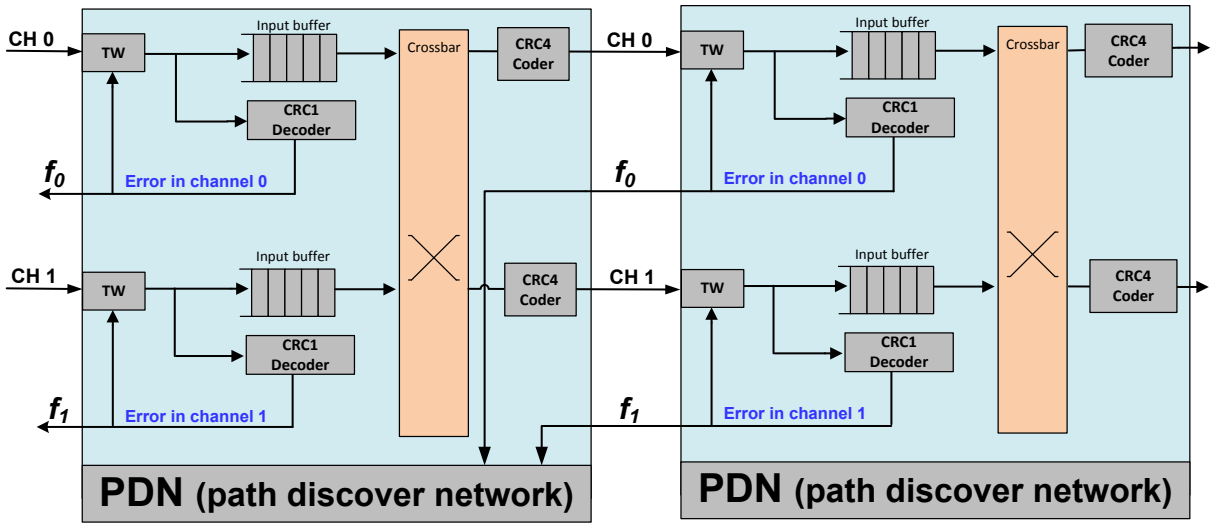

Figura 26 – Células de isolação adicionadas às portas de entrada do roteador para isolar as portas falhas.

1.16 Arquitetura desenvolvida para prova de conceito

Para atingir os objetivos estratégicos (modo degradado do roteador, recuperação de falhas) foi necessário implementar apenas o módulo CRC1 e CRC4. A Figura 27 ilustra a arquitetura do roteador para a prova de conceito. Ela consiste em dois decodificadores CRC nas portas de entrada do roteador para detectar eventuais falhas no canal. O roteador possui um codificador CRC para cada porta de saída

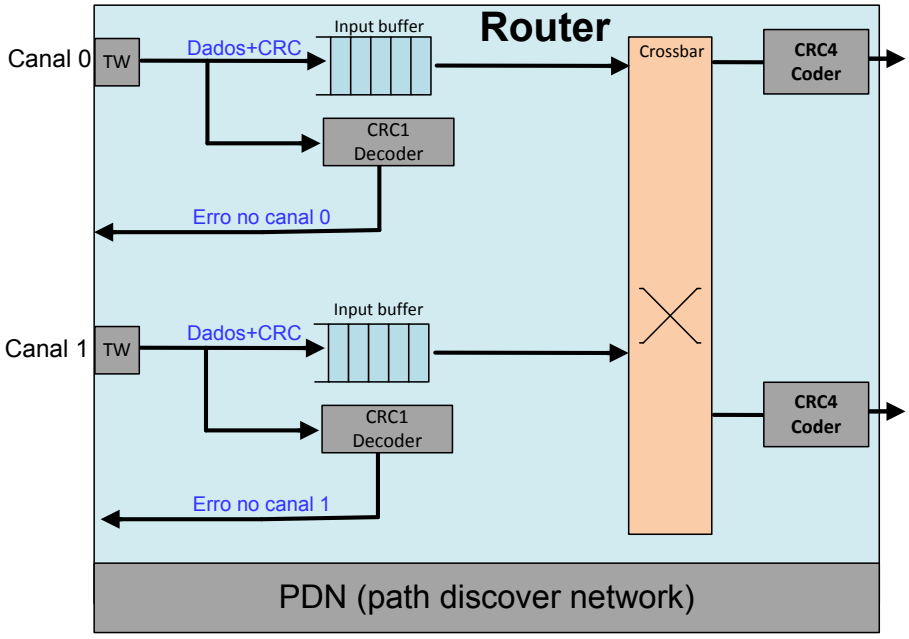

Figura 27 - Arquitetura do roteador sob avaliação, com decodificares nas portas de entrada.

A camada de rede, responsável pelo algoritmo de roteamento, foi previamente desenvolvida por [WAC13], através da PDN e descrito na seção 1.8.1.

PROTOCOLO DE RECUPERAÇÃO DE FALHAS

O Capítulo anterior descreveu a camada de hardware responsável por detectar falhas (camadas física, enlace e rede). O presente Capítulo apresenta a camada de software e hardware responsável pela correta comunicação entre PEs (camada de transporte), mesmo em caso de falha na NoC.

A Figura 28 apresenta um cenário que ilustra a comunicação entre dois PEs (*source* e *target*), com a presença de falha no caminho. A Figura 28(a) apresenta uma comunicação livre de falhas entre os dois PE (o caminho tomado pelos pacotes é representado por quadrados amarelos). Na Figura 28(b), o quadrado vermelho representa uma falha na comunicação entre dois roteadores, R1 e R2. O roteador R2 detecta a falha pelo módulo CRC, notificando o roteador R1. O roteador R1 gera uma requisição de retransmissão (pacote com serviço *seek resend*) para a origem, para que o pacote seja reenviado utilizando o mesmo caminho, conforme Figura 28(c). Na Figura 28(d) temos a representação do reenvio utilizando o mesmo enlace de comunicação onde aconteceu a falha, porém através do canal sem falha. Este processo para a troca de canal é detalhado nas subseções subsequentes.

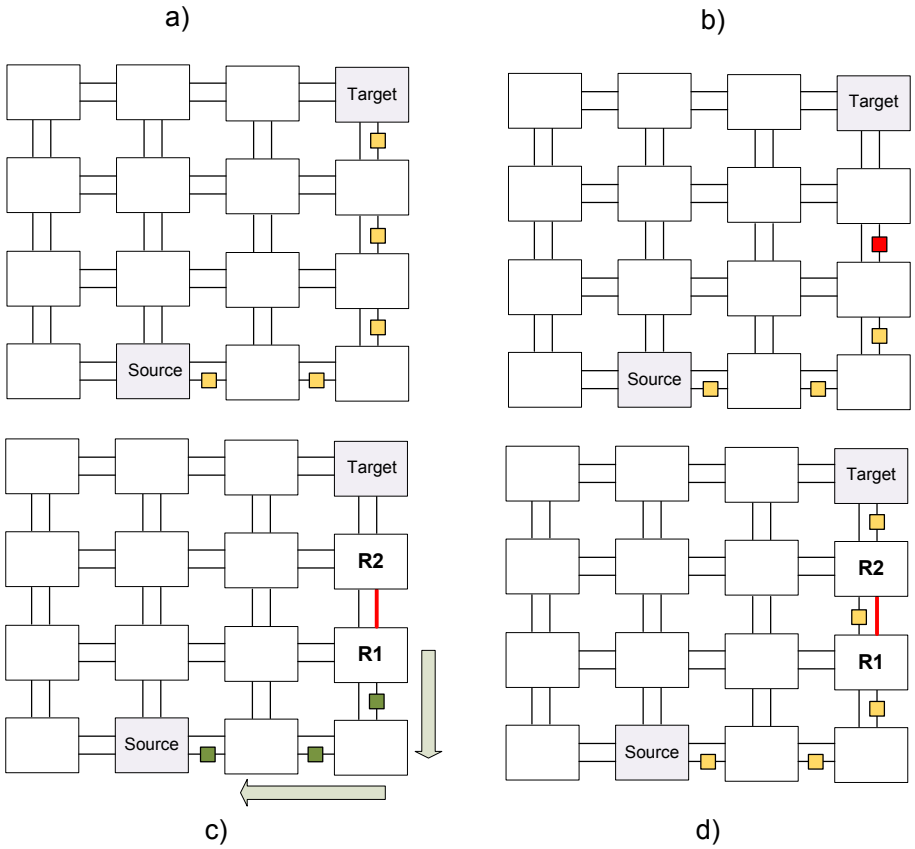

Figura 28 - Procedimento de detecção e reenvio de pacote em caso de falha em um canal enlace.

Observar que apenas no enlace com falha há a troca de canal, evitando o canal falho. No salto seguinte a comunicação volta para o canal de origem. O mecanismo só é viável em redes com canais duplicados. Este exemplo demonstra o benefício de utilizar canais duplicados, pois evitam que uma simples falha isole a comunicação entre dois PEs.

O presente trabalho fez com que a rede ganhe desempenho, pois a busca por novos caminhos livres de falhas só é necessária se ocorrer uma falha nos dois canais do enlace. Com isso a rede se torna mais robusta, pois consegue suportar uma maior quantidade de falhas e ainda assim continuar operacional.

1.17 Controle para degradação parcial do enlace

Como já mencionado, na proposta de [WAC13] quando uma falha ocorre, o canal deve ser desativado, sendo necessária a busca por um novo caminho na rede. Visto que o enlace é composto por dois canais, o roteador pode continuar a operar utilizando o canal sem falhas, porém com uma degradação de desempenho.

Quando uma falha é detectada, o roteador vizinho ao roteador que detectou a falha é avisado que o canal está falho (observar na Figura 26 os sinais *f0* e *f1*). Cada roteador possui uma tabela que informa a disponibilidade dos canais. Quando um pacote é transmitido, o roteador verifica se o canal no qual o pacote deve ser transmitido está com falha ou não. Estando o canal com falha, o pacote é enviado através do canal sem falhas. O roteador que recebe o pacote verifica o canal que deve ser utilizado (através do campo de prioridade do pacote), voltando assim para o canal original do pacote.

O roteador que recebe o sinal de falha, informa o módulo PDN que uma falha foi detectada. O módulo PDN contém registradores com o estado do enlace. Caso o enlace tenha apenas uma falha, um pacote contendo o serviço de SEEK_RESEND é gerado, e enviado à origem do pacote. Se o enlace contiver duas falhas então um pacote contendo o serviço de SEEK_UNREACHABLE é solicitado. O serviço SEEK_RESEND apenas solicita à origem do pacote a retransmissão dos dados, mantendo-se o algoritmo de roteamento XY. Já o serviço SEEK_UNREACHABLE implica no procedimento de busca de novo caminho e retransmissão dos dados, utilizando roteamento na origem. Estes serviços são tratados em nível de software, no *microkernel* do PE.

Para implementar o controle de degradação parcial do enlace e o serviço de retransmissão foi necessário realizar diversas alterações no nível de hardware:

- Na rede PDN foi necessário modificar a máquina de controle incluindo o tratamento do serviço SEEK_RESEND. O serviço SEEK_UNREACHABLE já era suportado.

- No roteador, no módulo responsável por realizar o roteamento do pacote (*switch_control*), foi necessário incluir uma sinalização de que um dado canal está falho para que o roteamento do pacote seja temporariamente feito através do canal não falho, porém sem alterar a direção do pacote.

- No PE, o módulo de *Fail Detect* sinaliza para a rede PDN do roteador: (*i*) serviço que está sendo solicitado; (*ii*) origem; (*iii*) destino do pacote. Este módulo foi modificado de forma que a origem e o destino são capturados em caso de falha. Quando ocorre uma falha utilizando roteamento na origem não é possível identificar o roteador destino, devido ao fato que no cabeçalho do pacote há apenas a informação das curvas (*turns*) necessárias para se alcançar o destino, e não o endereço do roteador destino, como no roteamento XY. Com isso, quando uma falha é detectada em uma comunicação

com roteamento na origem, o campo de destino recebe o mesmo valor que o roteador de origem. Desta forma é possível identificar no roteador origem quais mensagens necessitam de reenvio, conforme apresentado no pseudocódigo da próxima sessão.

- No PE, o módulo *Seek Manager* identifica o tipo de serviço a solicitar em caso de uma falha ser detectada pelos módulos de CRC. Foi necessário incluir a sinalização de falha dos decodificadores CRC. Desta maneira o módulo é capaz de identificar se é necessário solicitar um serviço de SEEK_RESEND (caso tenha apenas um canal em falha) ou um serviço de SEEK_UNREACHABLE (caso seja detectado duas falhas no mesmo enlace).

1.18 Protocolo em nível de software

Como já apresentado na Figura 21, no microkernel de cada PE existe um protocolo responsável por controlar a comunicação entre dois PEs. Este protocolo originalmente não é tolerante a falhas, pois não há a confirmação da correta recepção dos pacotes.

O protocolo de comunicação foi modificado de tal forma a prover uma confirmação de recepção, porém de forma indireta. Quando o microkernel recebe uma requisição de envio de dados, a posição do pipe ao invés de ser liberada (estado EMPTY) passa para o estado WAITING_ACK. Quando uma nova requisição de envio de dados é recebida, isto significa que o dado anterior foi corretamente enviado. Assim, a posição anterior do pipe passa de WAITING_ACK para EMPTY, e a posição atual do pipe para WAITING_ACK. Assim sempre teremos uma posição do pipe em WAITING_ACK. Esta última posição do pipe é liberada quando a tarefa terminar sua execução.

Para implementar o serviço de retransmissão foi necessário incluir um novo serviço no microkernel. Quando a NI recebe um pacote com o serviço SEEK_RESEND, é gerada uma interrupção para o processador. O processador então executa o pseudocódigo apresentado na Figura 29. Essa função verifica qual a mensagem que está aguardando o sinal de confirmação (WAITING_ACK), e se é o destino informado pelo pacote de SEEK_RESEND é o mesmo armazenado no pipe. Sendo esta verificação verdadeira, o pacote é reenviado.

```
função reenvia_mensagem_XY(endereço_destino)
{
        inteiro i;
        para (i=0; i<PIPE_SIZE; i++)
          se (  pipe[i].status = aguardando_confirmação &&
                pipe[i].endereço_destino = endereço_destino)
                {
                      reenvia_pacote(pipe[i]);
                }
}
```

Figura 29 - Pseudocódigo da função para reenviar pacotes usando XY.

Em casos onde é solicitado um SEEK_RESEND para um pacote com roteamento na origem (SR, source routing), a função verifica se existe alguma mensagem esperando o sinal de confirmação, e se o endereço destino é igual ao endereço local (Figura 30). Devido ao fato que os pacotes SR contém em seu cabeçalho apenas as direções para chegar ao destino, não é possível identificar qual é a mensagem a ser retransmitida. Logo, todas as mensagens que estão aguardando o sinal de confirmação são retransmitidas.

```
função reenvia_mensagem_SR(endereço_destino)
{
        inteiro i;
        para (i=0; i<PIPE_SIZE; i++)
          se ( pipe[i].status = aguardando_confirmação &&
                pipe[i].endereço_local = endereço_destino
                  {
                        reenvia_pacote(pipe[i]);
                  }
}
```

Figura 30 - Pseudocódigo da função para reenviar pacotes usando roteamento na origem.

No caso de uma solicitação de SEEK_RESEND todos os pacotes serão retransmitidos mesmo em casos onde a retransmissão não foi solicitada. Porém, isso não irá gerar nenhuma falha na comunicação desses pacotes pois os pacotes contém um campo que identifica a ordem dos pacotes (i.e., número de sequência). Caso um dos PE receber algum pacote fora de ordem ele será descartado e a comunicação irá continuar normalmente.

A Figura 31 apresenta diferentes cenários de operação do protocolo de comunicação tolerante a falhas, com roteador operando em modo degradado e computação de um novo caminho [FOC15]. Conforme ilustrado na Figura 31(a) a falha ocorre em um dos canais da porta sul do roteador 6 (R6) sendo detectada pelo roteador 3 (R3). A canal falho em R6 é desativado, e R3 envia uma mensagem de seek_resend para o roteador 1(R1). O PE conectado a R1 reenvia a mensagem para o roteador 9 (R9), é usado o canal ainda ativo entre R3 e R6 (veja na Figura 31 (b) que o canal desabilitado entre R3 e R6 está com desenhado com uma linha vermelha).

Na Figura 31 (b) uma segunda falha ocorre no canal ainda ativo na porta sul de R6. Neste caso, a PDN é utilizada para estabelecer um novo caminho entre R1 e R9 (R1-R2-R5-R8-R9). Quando o PE1 (conectado ao R1) recebe o pacote de retorno (*backtrack packet*) com o novo caminho, é verificado se os canais físicos utilizados no novo caminho são livres de deadlocks, o novo caminho é então armazenado na NI. Após este processo, a mensagem é retransmitida. Notar que o algoritmo de roteamento padrão é o XY. Apenas quando é solicitada a busca por um novo caminho livre de falhas que o algoritmo de roteamento será com cálculo na origem (*Source Routing*).

O exemplo Figura 31(c) é similar ao da Figura 31(a), com a diferença que agora está sendo utilizado o algoritmo de roteamento com cálculo na origem, ao invés do XY. O

pacote XY contém no seu cabeçalho o endereço de destino. Contudo o pacote do algoritmo com cálculo na origem contém apenas as voltas(direções) que o pacote deve tomar para chegar ao roteador de destino. Está diferença faz com que se tenha um comportamento diferente ao retransmitir mensagens com cálculo na origem. Na Figura 31(a), PE1 retira a informação de destino do pacote de *seek resend*, para efetuar a retransmissão. Porém, na Figura 31 (c) não é possível determinar qual mensagem estava bloqueada é a mensagem com falha, pois a mensagem de *seek resend* não contém o endereço de destino. Neste caso, todas as mensagens que estão no estado “waiting ack” são retransmitidas. Esta questão não representa necessariamente um problema desde que o sistema operacional descarte as mensagens recebidas com o mesmo número de sequência.

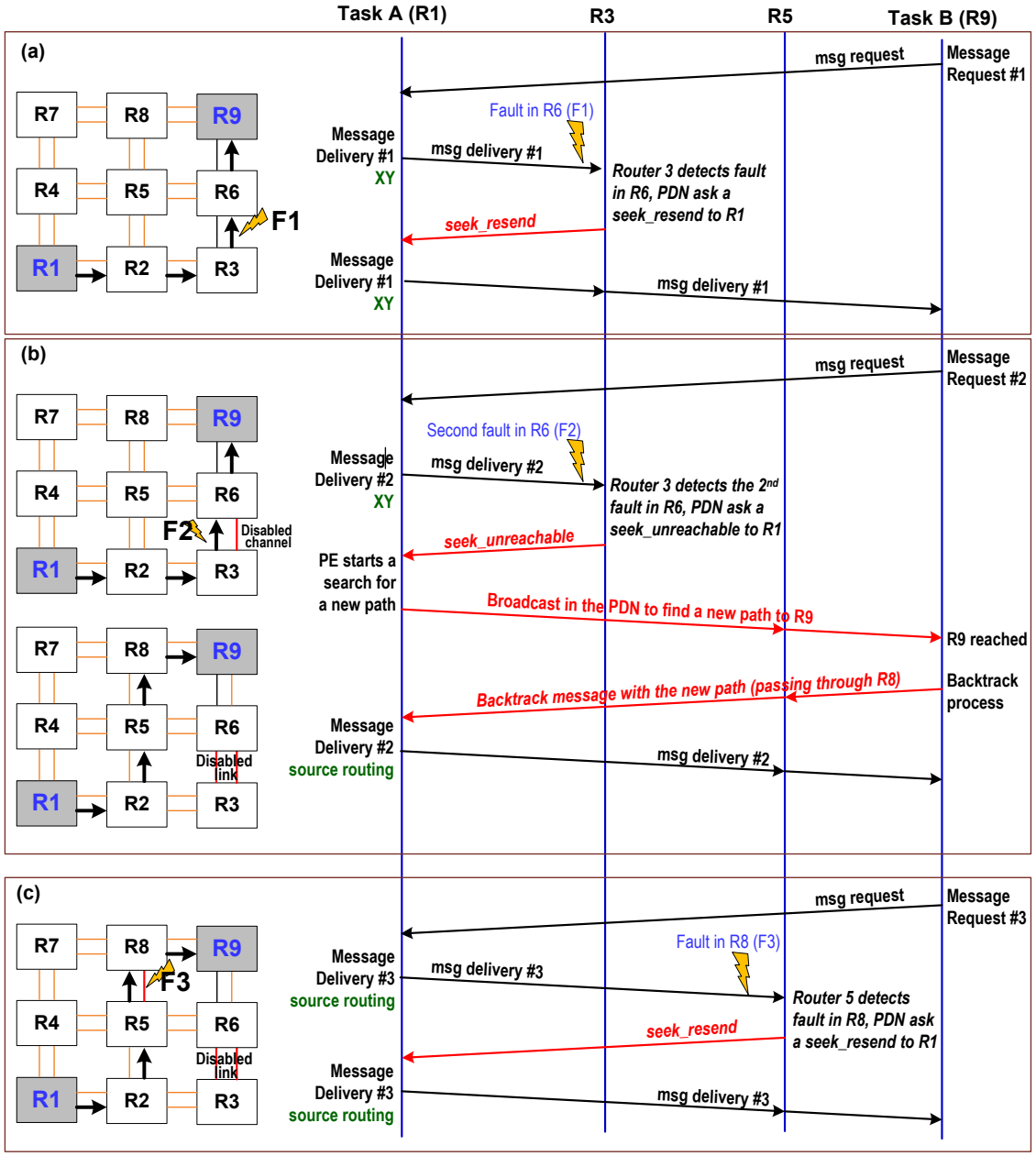

Figura 31 - Protocolo de comunicação tolerante a falhas, com roteador operando em modo degradado e computação de um novo caminho [FOC15].

MÉTODO DE RECUPERAÇÃO DO ROTEADOR

O presente Capítulo apresenta a técnica de recuperação do roteador após a detecção de falhas transientes. A técnica consiste em um teste de recuperação do roteador não intrusivo na presença de falhas transientes. A detecção de falhas pode ser permanente ou transiente. Em ambos os casos, como mostrado na Figura 28 o canal é desativado e a mensagem é retransmitida. Contudo se fez necessário a criação de um mecanismo para verificar se o canal desativado pode ser recuperado.

O método consiste em dois módulos em hardware: módulo de teste (TM) e o módulo de recuperação (TRM). O método é completamente implementado em hardware e transparente para as camadas de software.

Cada roteador possui apenas um TM. O módulo TM é implementado com uma máquina de estados simples, executando as seguintes ações:

- Detecção de uma falha no canal. Os sinais f0 e f1 na Figura 35 notificam o roteador adjacente que a porta de entrada está falha. A falha pode ser no canal do enlace ou no buffer de entrada.
- É iniciado um contador para que o pacote de teste seja inserido no canal falho. A execução do teste ocorre depois do efeito de uma falha transiente.
- O pacote de teste é injetado no canal falho. O pacote de teste pode ser inserido no canal falho com segurança pois a lógica de controle do roteador desabilita o envio de pacotes provenientes do canal falho. O pacote de teste contém uma importante distância de Hamming entre os flits para avaliar o canal e o buffer de entrada, conforme ilustrado na Figura 33.
- Ao final do pacote de teste, se nenhuma falha foi detectada nos módulos decodificadores de CRC então o sinal *fx* indica que o canal pode ser restaurado, (f_x='0') se foi um falha transiente e (f_x='1') caso seja uma falha permanente.

A Figura 32 apresenta a máquina de estados do módulo TM. O módulo de TM injeta um pacote diretamente na porta de saída do roteador utilizando o canal onde a falha foi detectada. Devido a detecção da falha, a porta de saída está desativada, fazendo com que o módulo TM consiga injetar o pacote de teste diretamente, sem concorrência com outros fluxos de dados. O pacote de teste é enviado contendo ao final um identificador de fim de pacote (EoP), esse identificador é importante pois será utilizado no módulo TRM responsável por determinar se o pacote chegou ao final sem nenhuma ocorrência de falha. A função dos estados é a seguinte:

- *idle:* permanece neste estado enquanto não houver a sinalização de nenhuma porta vizinha ao roteador sinalizar falha;
- *timer*: controla um temporizador. Este tempo é necessário que que seja possível descartar o pacote atual, e também cessar o efeito da falha, caso esta seja transiente;
- *selected channel*: seleciona qual porta testar. Se mais de um porta apresentar falha,

um algoritmo *round-robin* é utilizado para a escolha da porta.

- *send packet with EoP*: envio do pacote de teste.

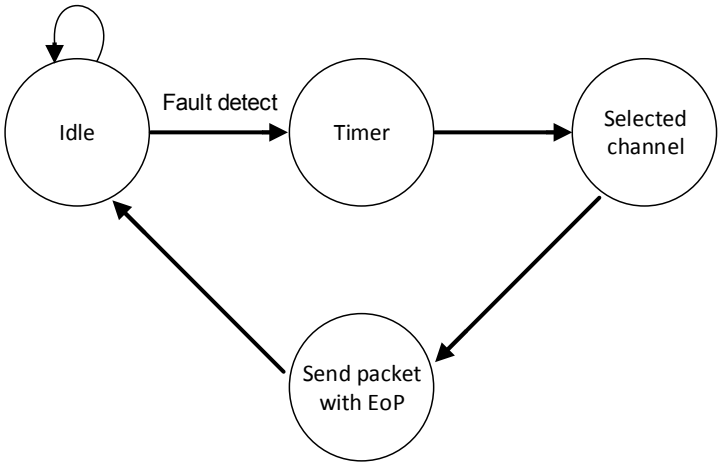

Figura 32 - Máquina de estado do módulo TM.

A Figura 33 apresenta o conteúdo do pacote de teste. O primeiro flit é o header do pacote de teste (0xFEFE). Ele tem por objetivo avisar o módulo TRM que um pacote de teste está sendo injetado no canal. Desta maneira o módulo TRM pode começar a analisar os dados e verificar a presença de falhas. O pacote de teste passa através do canal e do buffer de entrada e então é analisado sua integridade no módulo de CRC. Com isso é possível avaliar tanto o canal quanto o buffer de entrada, módulos onde a maioria das falhas ocorre. O conteúdo do pacote alterna os valores 0xA5A5 e 0x5A5A de forma a ter a maior taxa de chaveamento possível.

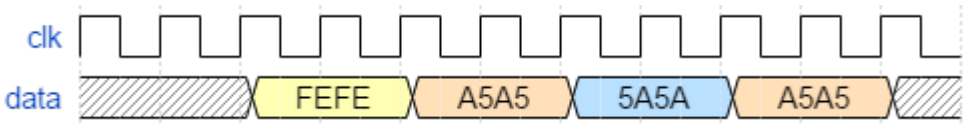

Figura 33 - Pacote de teste utilizado para verificar a integridade do canal.

A Figura 34 apresenta a máquina de estados do módulo TRM. Quando uma falha é detectada o módulo espera o fim do pacote com o identificador EoP. Se uma falha é detectada o módulo retorna para o estado de Idle e considera a falha sendo permanente, então nenhum teste adicional é realizado no canal. Caso o pacote chegue ao final o canal é liberado. Para poder receber o pacote de teste, o módulo TRM ao identificar uma falha no canal faz com que alguns sinais sejam liberados no *Test Wrapper* tais como (rx, tx, eop, crédito e data), para que o pacote consiga ultrapassar a célula de isolação.

Cada roteador possui oito módulos TRM, uma para cada porta de entrada, com exceção das portas locais. A função deste módulo é detectar o inicio do pacote de teste, e ao final do pacote de teste reiniciar o sinal f_x se todos os flits passaram na verificação do módulo CRC e nenhuma falha foi encontrada. O módulo TRM também é responsável por liberar alguns sinais de controle (rx, tx, eop, crédito) para que o pacote de teste consiga passar pela célula de isolação.

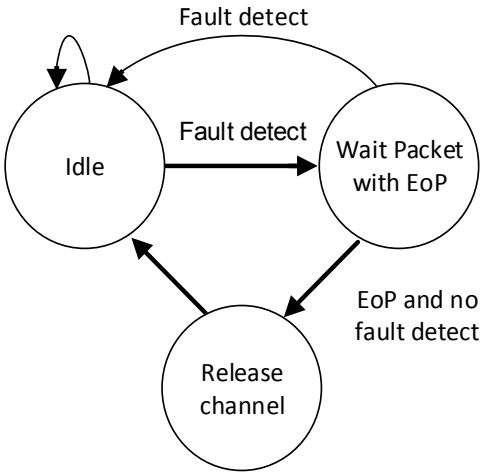

Figura 34 – Maquina de estado do módulo TRM.

A Figura 35 detalha esse processo. O Método de teste contém os seguintes passos:

1. Roteador 2 detecta uma falha e sinaliza para o roteador 1.
2. Roteador 1 envia para o roteador 2 um pacote de teste.
3. Roteador 2 recebe o pacote de teste. Notar que o pacote de teste não é encaminhado para nenhuma porta de saída do roteador 2 pois a porta de entrada está em teste e para a lógica do roteador ela está desabilitada.
4. Roteador 2 pode reiniciar o sinal de falha se todos os flits passarem pela verificação.

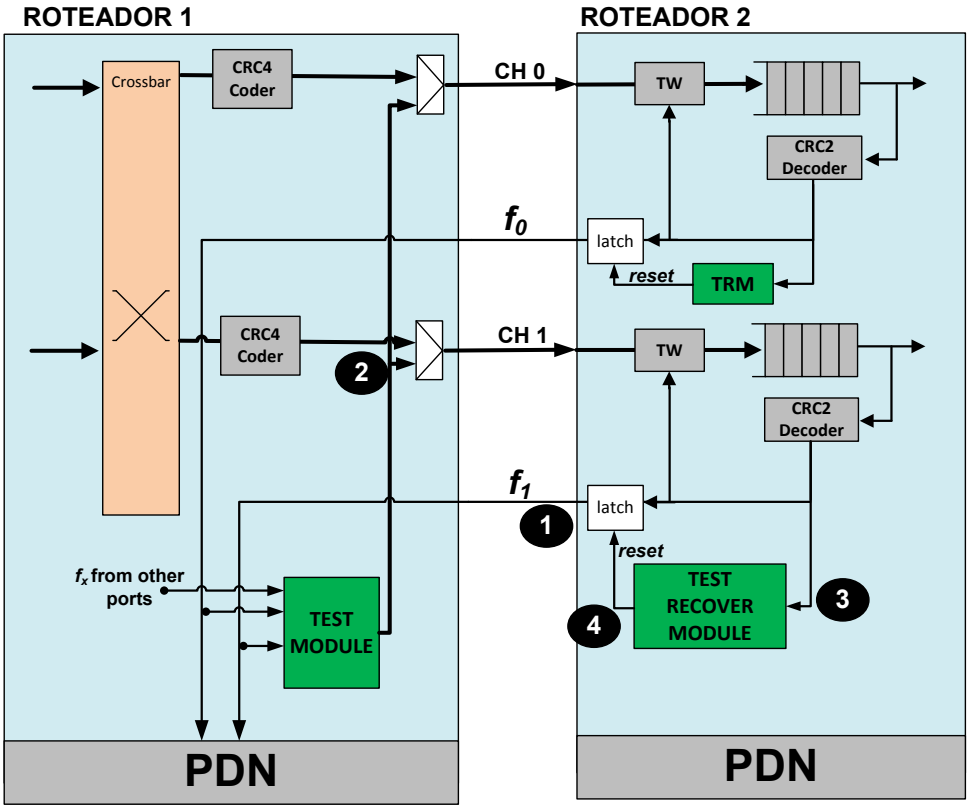

Figura 35 - Módulos de teste e recuperação para detectar falhas transientes. Para simplificar apenas o módulo CRC 2 está presente.

AVALIAÇÃO

Este Capítulo apresenta os resultados da implementação da arquitetura HeMPS com técnicas de tolerância a falhas apresentadas nos Capítulos 4, 5 e 6. São analisados dois cenários de execução: o primeiro onde se apresenta uma execução de uma campanha de injeção de falhas em uma aplicação sintética (6 tarefas) e em uma aplicação MPEG(5 tarefas).

Para efetuar a avaliação do trabalho realizado no decorrer do curso de Mestrado, obteve-se resultados experimentais a partir de uma NoC 4x4 instanciada no MPSoC HeMPS[CAR09], apresentado no Capítulo 3. Na Seção 7.1, apresenta-se o método utilizado para realizar a injeção de falhas. Na Seção 7.2, apresenta-se os resultados iniciais para realizar a prova de conceito. Na Seção 7.3 apresenta-se a avaliação do método de operação do roteador em modo degradado. Na Seção 7.4 apresenta-se a avaliação quantitativa do método de operação em modo degradado, através de uma campanha de injeção de falhas. Na Seção 7.5 apresenta-se os resultados obtidos na campanha a recuperação do roteador após falhas transientes. Na Seção 7.6 apresenta-se os resultados de consumo de área.

1.19 Injeção de falhas

Para realizar a injeção de falhas utilizou-se o comando do simulador Modelsim denominado *force*. Através desse comando é possível escolher o valor lógico que um sinal deve assumir, e em qual instante de tempo ocorre a transição. Desta forma simula-se uma falha *stuck at*, que deve ser detectada pelo módulo de CRC.

A Figura 36 apresenta o arquivo *signals*, o qual indica os sinais que devem ter os valores fixados, assim como o momento para ocorrer esta alteração. Por exemplo, a primeira linha do arquivo indica que o bit 10 do sinal de *data_in* da porta 7 (porta sul do roteador 15) deve assumir o valor lógico ‘1’ no instante 280090104 ps.

```
/test_bench/HeMPS/proc(15)/slav/slave/PE/Router/data_in(7)(10) 1  280090104  ps
/test_bench/HeMPS/proc(15)/slav/slave/PE/Router/data_in(6)(10) 1  352440778  ps
/test_bench/HeMPS/proc(13)/slav/slave/PE/Router/data_in(6)(10) 1  442138242  ps
```

Figura 36 - Arquivo que indica quais sinais onde serão injetadas falhas stuck-at.

A Figura 37 apresenta o script utilizado durante a simulação para ler a lista de sinais a terem os valores fixados, e assim simular a injeção de falhas. Este script lê o arquivo *signals* (primeira linha do arquivo), e para cada sinal é utilizado o comando *force* (*set force_cmd …*).

```
/test_bench/HeMPS/proc(15)/slav/slave/PE/Router/data_in(7)(10) 1  280090104  ps  -cancel 3 ms
```

Figura 37 - Arquivo que indica quais sinais devem ter o valor lógico onde serão injetadas falhas stuck-at, e o tempo em que a falha desaparece.

Este processo permite criar de forma automatizada uma campanha de injeção de falhas, permitindo simular a presença de falhas permanentes ou transientes. O tempo em que as falhas são ativadas é fixo. De forma aleatória cinco falhas são geradas, isso é importante, pois facilita o processo de avaliação, sendo possível criar milhares de cenários de teste.

```
if {[file exists "signals"] == 1} then {

        puts "injecting faults!"
        set fault [open "signals" r]
        set signals [read $fault]
         set lines [split $signals "\n"]
         close $fault

        foreach line $lines {
            set tokens [split $line " "]
            set number [llength $line]
            if { $number == 4 } {
                set force_cmd [format "force %s %s %s %s -cancel 1000 ms"
                        [lindex $tokens 0] [lindex $tokens 1] [lindex $tokens 2] [lindex $tokens 3]]
             }
             if { $number ==6 } {
                set force_cmd [format "force %s %s %s %s -cancel %s %s" [lindex $tokens 0]
                  [lindex $tokens 1] [lindex $tokens 2] [lindex $tokens 3] [lindex $tokens 4] [lindex
$tokens 5]]
                    }
         puts $force_cmd
         eval $force_cmd
         }
} else {
    puts "no fault injection"
}
```

Figura 38 - Script *fault-inject.do* que lê a lista de sinais que devem ter os valores lógicos fixados para simular a injeção de falhas.

1.20 Utilização da lógica de CRC para detecção de falhas

Conforme apresentado anteriormente na Figura 26, foi utilizado um decodificador CRC em paralelo à entrada dos buffers dos roteadores. A simulação foi realizada em uma rede 2x2. A comunicação é realizada do roteador 0 para o roteador 1. Conforme relatado na seção 4.3 foi utilizada esta arquitetura para prova de conceito da técnica. Dado que o objetivo é provar o conceito, esta seção apresenta a operação da arquitetura, apresentando-se posteriormente avaliação quantitativa.

Assim que um *flit* chegar ao *Roteador 1*, o decodificador realiza o teste para determinar se o pacote chegou com falha ao roteador. Caso o pacote apresente uma falha, é gerado um sinal de erro para o *Roteador* 0. O *Roteador 0* é responsável por enviar uma mensagem ao roteador origem do pacote, através da propagação de uma mensagem usando o módulo *PDN* solicitando um novo caminho livre de falhas. O roteador origem do pacote inicia a busca por um novo caminho utilizando o módulo *PDN.*

A Figura 39 ilustra a simulação deste cenário. Os seguintes tempos, em relação ao sinal de relógio, são destacados:

- tempo t1: sinal *data_out(1)* (leste 1) do roteador 0 envia pacote para o roteador 1;
- tempo t2: o roteador 1 inicia a recepção do pacote em *data_in(3)* (oeste 1);
- tempo t4: o valor de CRC do pacote recebido é 'D', entretanto o decodificador calcula 'C', sinalizando erro (*crc_error*);
- tempo t6: o erro é sinalizado pelo sinal *out_failed_port*(3) (oeste 1);
- tempo t7: o sinal de erro é recebido pelo roteador 0 no sinal *in_falied_port(1)* (leste 1).

Este experimento demonstra a correta recepção do erro, assim como a propagação da informação da falha para o roteador vizinho.

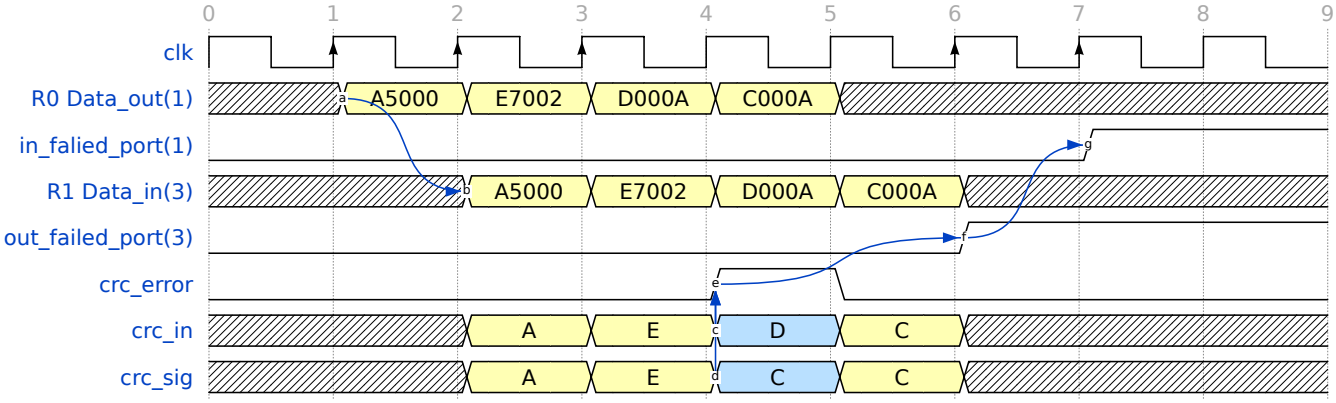

Figura 39 - Detecção de falha em canal com utilização de CRC.

1.21 Avaliação do protocolo de operação em modo degradado

Para a realização dos testes foi utilizado um MPSoC HeMPS de dimensão 4x4, conforme ilustrado na Figura 40. Uma aplicação sintética, produtor-consumidor é utilizada para fins de validação. Nesta aplicação a tarefa *source* envia dados para a tarefa *target*.

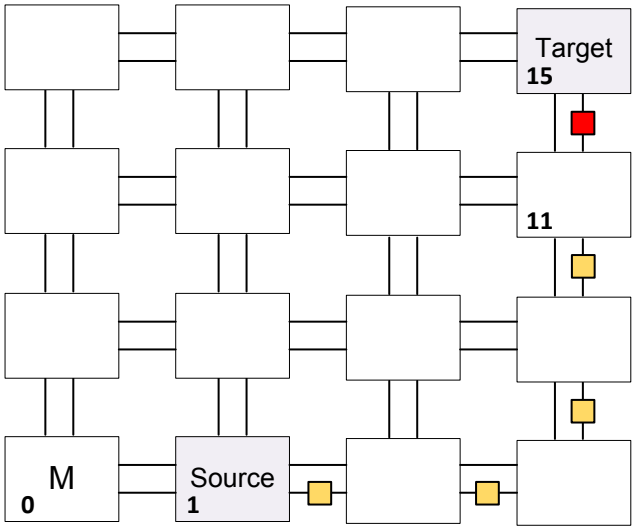

Figura 40 - Cenário de Avaliação. Mestre localizado no PE 0, tarefa *source* mapeada no PE 1, e tarefa *target* mapeada no PE 15.

Quando uma falha é detectada em um canal, é enviada uma requisição de reenvio através da rede dedicada *PDN* até a origem. A origem reenvia o pacote utilizando o mesmo caminho até chegar ao canal em falha. O roteador identifica que o canal

preferencial está falho e envia o pacote através do canal livre de falhas. No próximo salto o pacote volta a utilizar o canal preferencial.

A Figura 41 ilustra este cenário de teste com falha em apenas um canal. Os instantes destacados na Figura 41, em relação ao sinal de relógio, correspondem a:

- t7: momento de injeção do segundo pacote no roteador origem (sinal *Local R1*);
- t8: momento da injeção da falha, observar que o pacote 2 foi parcialmente recebido no destino;
- t13: momento do início do reenvio do pacote;
- t14: momento da recepção do pacote no roteador destino.

Neste experimento, o tempo total entre o envio do pacote, detecção da falha e reenvio do pacote (intervalo entre os tempos t7 a t14) totalizou 11.730 ns, o que correspondeu 1.173 ciclos de relógio. A maior parcela deste tempo é consumida pelo processador origem, para executar o reenvio do pacote.

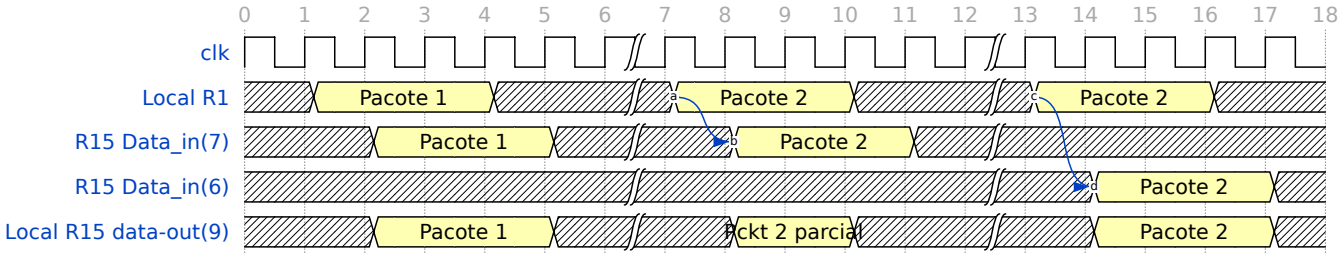

Figura 41 - Cenário de teste com falha em apenas um canal, como roteador operando em modo degradado.

É importante observar na Figura 41:

- sinal router(15)data_in(7) corresponde à porta sul 0 do roteador 15, onde ocorreu a falha. Este pacote é parcialmente consumido, e depois descartado. Da mesma forma a porta local do roteador destino recebe parcialmente o pacote, e o descarta. Com isto, validou-se o processo de descarte de pacotes, tanto pelo roteador vizinho ao roteador falho, assim como no processador destino.

- sinal router(15)data_in(6) corresponde à porta sul 1 do roteador 15, que não está falha. Após o reenvio do pacote (tempo t14), esta porta passa a ser utilizada, encaminhando-se corretamente o pacote para a porta local do roteador 15.

A Figura 42 ilustra este cenário de teste com falha em apenas um canal, porém utilizando a proposta original de [WAC12], onde a falha em um canal desabilita o enlace. Os instantes destacados na Figura 42 correspondem a:

- t7: momento de injeção do segundo pacote no roteador origem (sinal *Local R1*);

- t8: momento da injeção da falha, observar que o pacote 2 foi parcialmente recebido no destino;

- t13: momento do início do reenvio do pacote;
- t14: momento da recepção do pacote no roteador destino.

Neste experimento, o tempo total entre o envio do pacote, detecção da falha e reenvio do pacote (intervalo entre os tempos t7 a t14) totalizou 25.420 ns, o que correspondeu 2.420 ciclos de relógio. Assim como no experimento anterior, a maior parcela deste tempo é consumida pelo processador origem, para executar o reenvio do pacote. Entretanto, neste caso, há o processo de busca de caminho alternativo, o qual é extremamente rápido, por ser implementado em hardware.

É importante observar na Figura 42:

- reenvio do pacote ocorre agora pela porta ***router(15)data_in(2),*** o que corresponde à porta oeste 0 do roteador 15. Esta troca de porta mostra que o roteamento foi alterado, passando-se a utilizar roteamento na origem.
- a recepção no roteador destino passou a ocorrer para porta **local 1**, e não na porta local 0, devido ao roteamento na origem.

A diferença de desempenho entre os 2 métodos correspondeu a 1.173 ciclos de relógio (2.420 - 1.247). Esta diferença deve-se a:

- protocolo de busca de caminho e *backtracking* (em hardware);
- execução do função de cálculo de novo caminho (em software).

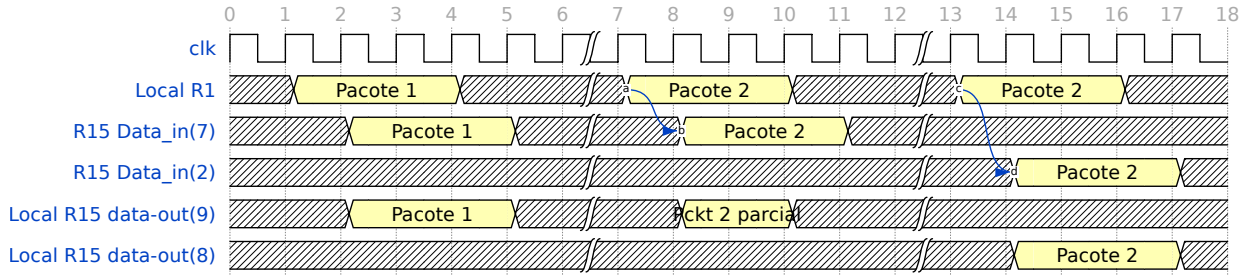

Figura 42 - Cenário de teste com falha no enlace, utilizando a proposta original de [WAC12].

Além do menor tempo necessário para reestabelecer à comunicação em caso de falha, as vantagens em se utilizar o método proposto incluem:

- evita tráfego de pacotes de *backtracking*, o qual circulam pela NoC e não pela rede *PDN*. Estes pacotes podem interferir no desempenho de outras aplicações ou sofrerem atrasos devido a congestionamento na rede, fazendo com que a diferença de tempo observada acima aumenta.
- roteador continua a ser utilizado por mais tempo, mesmo com um canal falho. Na proposta original um canal falho conduzia ao isolamento do enlace. Assim, mesmo com todos os canais 0 falhos (N/S/E/W), o roteador pode continuar operando.
- mantém o roteamento XY, o que reduz a necessidade de tabelas para roteamento na origem.

As desvantagens do método proposto incluem:

- Gasto adicional de área utilizado para incluir a rede PDN.
- Aumento do tamanho do flit de 16 bits para 20 bits para inclusão do valor de CRC em cada pacote.
- Gasto adicional de área na inclusão dos módulos de CRC.

1.22 Avaliação quantitativa do protocolo de operação em modo degradado através de campanha de injeção de falhas

Para realizar a validação das técnicas apresentadas foram realizadas diversas simulações utilizando benchmarks. Em especial duas aplicações foram utilizadas: sintética (6 tarefas) – Figura 42, e MPEG (5 tarefas)- Figura 44. A campanha de injeção de falhas aleatórias foi utilizada para validar diferentes configurações. As falhas foram inseridas nas portas de entrada dos roteadores, com exceção das portas locais (fora do escopo deste trabalho), pois este trabalho assume apenas falhas na NoC. Um fluxo de análise automática foi criado, incluindo a geração do MPSoC, criação dos cenários de teste, simulação de todos os cenários e a avaliação de desempenho.

Para cada aplicação, 1.000 cenários foram simulados (modelos MPSoC em nível RTL em VHDL). Cinco falhas permanentes foram inseridas ao mesmo momento para cada cenário, em portas aleatórias. As simulações demonstraram que o método suporta diversas falhas simultâneas nas portas, levando em conta que o número de falhas não separe a aplicação em grafos disjuntos. Por exemplo as falhas fossem agrupadas de tal maneira que não seria possível encontrar um caminho entre uma origem e destino.

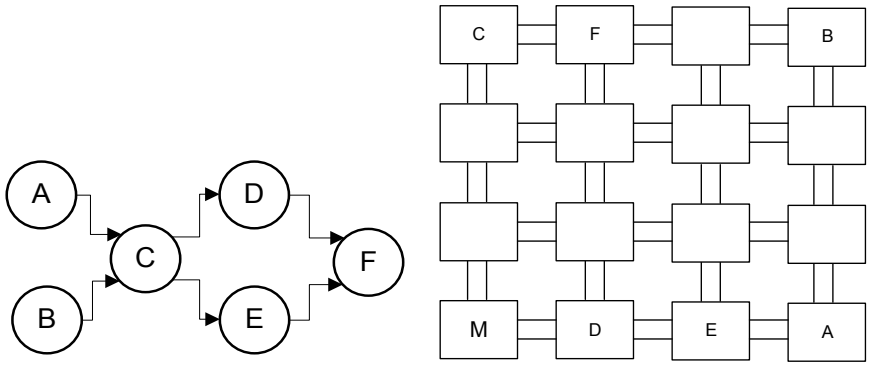

Figura 43 – Grafo da aplicação sintética e mapeamento das tarefas.

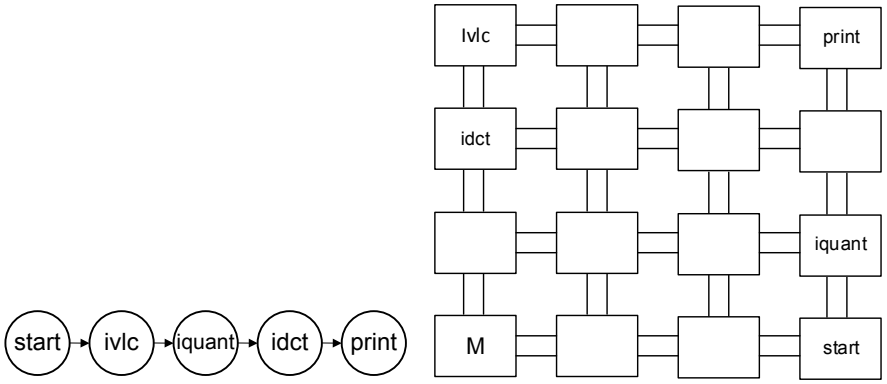

Figura 44 - Grafo da aplicação MPEG e mapeamento das tarefas.

A Tabela 4 resume os resultados relacionados a campanha de injeção de falhas. A primeira linha apresenta o número de cenários não afetados por falhas. A segunda linha contém o número de cenários afetados por falhas, correspondendo a 100% dos cenários com falha. Todos os cenários das aplicações terminaram corretamente sua execução.

A maioria dos cenários com falha necessitou de apenas uma ou duas retransmissões de pacotes (linhas três e quatro da Tabela 4). Para a aplicação sintética 87% dos cenários afetados por falhas necessitou de 1 ou 2 retransmissões apenas. No caso da aplicação MPEG, este valor correspondeu a 92,7%. Poucos cenários necessitaram de três ou quatro retransmissões para executar corretamente as aplicações.

A última linha corresponde aos cenários com falha que foram afetados com duas falhas físicas nos canais do mesmo enlace. Nestes casos foi necessário iniciar a busca por um novo caminho livre de falhas entre as tarefas de origem e destino.

Tais resultados mostram a vantagem de se operar em modo degradado pois o número de buscas por um novo caminho foi drasticamente reduzido. Nos experimentos, a redução chegou a 97,1% e 98.3% para as aplicações sintéticas e MPEG, respectivamente. Outro beneficio de operar em modo degradado é a possibilidade de restaurar o enlace de uma falha transiente sem a necessidade de mudar o modo de operação do roteador.

Tabela 4 - Análise de resultados obtidos através da campanha de injeção de falhas.

		Sintética	MPEG
Número de cenários **não** afetados por falhas		271	420
Número de cenários afetado por falhas		**729** (100%)	**580** (100%)
Número de retransmissões de pacotes **sem** uma realizar uma busca por um novo caminho	1	392 (53.8%)	347 (59.8%)
	2	242 (33.2%)	191 (32.9%)
	3	64 (8.8%)	28 (4.8%)
	4	10 (1.4%)	4 (0.7%)
Número de pacotes retransmitidos com a busca de um novo caminho livre de falhas		21 (2.9%)	10 (1.7%)

O tempo de execução das aplicações não foi na prática afetada pela campanha de injeção de falhas. O tempo médio de execução da aplicação sintética, sem falhas, é de 2.925 ms, enquanto com falhas foi de 2.948 (+ 0.7%) e o pior caso foi de 3.002 ms. O tempo médio de execução da aplicação MPEG, sem falhas foi de 4.591 ms, enquanto que com falhas foi de 4.611 (+0.4%) e o pior caso foi de 4.622 ms.

1.23 Avaliação do protocolo de recuperação

A Figura 45 apresenta o comportamento do método de recuperação de falhas transientes. O cenário é similar ao apresentado na Figura 42, a qual foi assumido que a falha foi permanente. Na Figura 45 a falha transiente desabilita a porta sul 1 (S1), fazendo com que a retransmissão do pacote 2 (P2) através da porta sul 0 (S0). O método de teste injeta o pacote de teste (Ptest) em S1 restaurando o canal. O próximo pacote, P3, utiliza o canal restaurado. A segunda falha transiente é inserida no mesmo canal, repetindo o

comportamento anterior. A Figura 45 mostra que o pacote de dados (P4) pode ser transmitido através do canal sem falha (S0), e simultaneamente com o envio do pacote de teste no canal (S1).

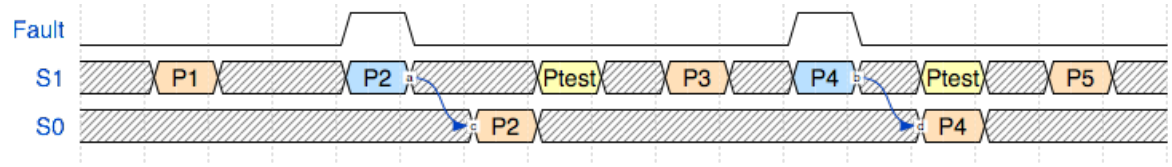

Figura 45 - Recuperação de falhas transientes.

1.24 Avaliação do consumo de área

A implementação contém oito decodificadores e oito codificadores de CRC por roteador. Cada codificador CRC consome 10 LUTs, e cada decodificador CRC consome oito LUTs e um FF. Além destes módulos, há o aumento da largura do flit de 16 para 20 bits, o que impacta no aumento na área do buffer e no crossbar interno. Também foi inserido oito módulos TRM e um módulo TM por roteador. Cada módulo TRM consome 8 LUTs e 4 FF, totalizando 64 LUTs e 32 FF ao total por roteador. O módulo TM consome 124 LUTs e 60 FF.

Dado que os buffers são implementados com LUTRAM, cada posição adicional de buffer impacta em uma LUT adicional. Assim, o impacto de área no roteador foi:

- módulos de CRC: 144 LUTs e 8 FFs;
- buffers: 32 LUTs, configuradas como LUTRAM (8 buffers, 4 posições adicionais por buffer).
- Módulo TM : 124 LUTs e 60 FF.
- Módulo TRM: 64 LUTs e 32 FF.

A Tabela 5 apresenta o consumo de área para o roteador com e sem os módulos de CRC. Observa-se, no roteador, um acréscimo de 181 LUTS e 18 FFs, coerente com o calculado anteriormente. No PE, o acréscimo de área corresponde a 6%. Este aumento de área em relação ao PE original é aceitável, tendo a maior tolerância a falhas na NoC. Também é importante destacar que a busca por um novo caminho só será iniciada quando um enlace inteiro falhar.

Tabela 5 - Comparativo do consumo de área, entre a MazeNoc [WAC13] e a MazeCRC.

Módulo		**MazeNoC**	**MazeCRC**	**Acréscimo**
NI	LUTs	875	818	-6,51 %
	FFs	1047	1022	-2,39%
DMA	LUTs	168	168	0,00%
	FFs	126	126	0,00%
Router	**LUTs**	**2242**	**2951**	**31,62%**
	FFs	**848**	**850**	**0,24%**
PE	LUTs	6298	6758	7,30%
	FFs	2786	2865	2,84%

CONCLUSÃO E TRABALHOS FUTUROS

O presente trabalho apresentou um método integrado para detecção e recuperação de falhas em tempo de execução para NoC em MPSoCs. A proposta adicionou tolerância a falhas na camada física até a camada de transporte, integrando módulos de hardware e funções em software implementadas no microkernel do sistema. Escalabilidade é uma importante característica da proposta, visto que não foi necessário implementar tabelas para armazenar o estado do sistema e o tamanho da NoC auxiliar (PDN) não cresce com o tamanho do sistema.

Uma importante característica para tolerância a falhas em nível de hardware é a adoção de canais físicos duplicados. Duplicando os canais físicos permite ao roteador trabalhar em modo degradado. A maioria dos cenários com simulações de falha necessitou apenas de uma ou duas retransmissões de pacotes, (92,1% e 95,2% para as aplicações da Sintética e MPEG respectivamente) com cinco injeções de falhas simultâneas. O impacto do método nas aplicações gerou um atraso mínimo no tempo de execução (abaixo de 1%).

Como trabalhos futuros cita-se:

1. Quando um PE se torna inalcançável com falhas nos enlaces adjacentes, migrar a tarefa para outro PE.
2. Reduzir o tamanho do hardware na interface de rede. A NI é responsável por armazenar os caminhos quando é utilizado roteamento na origem. Os caminhos podem ser administrados pelo kernel sem afetar o desempenho.
3. Estender o método para lidar com falhas no processador.
4. Proteger as máquinas de controle do roteador contra falha.
5. Método de teste e recuperação pode atuar como um monitor para detectar envelhecimento dos canais. Se um determinado enlace ou buffer de entrada for afetado por um grande número de falhas transientes, este fato pode sinalizar que o componente está desgastado ou no final da vida útil, sendo assim o módulo de teste pode desabilitar a porta permanentemente.

REFERÊNCIAS

[BEN02] Benini, L.; Micheli, G. *Networks on Chips: a New SoC Paradigm*. Computer, v.35(1), 2002, pp. 70-78.

[BRA90] Brander, H.; Howard, J.; Menicou, G.; Plagemann, S. *Quality of service in broadband communications*. In: International Conference on Integrated Broadband Services and Networks, 1990, pp. 166-171.

[CAR09] Carara, E. A.; Oliveira, R. P.; Calazans, N. L. V.; Moraes, F. G. *HeMPS – a framework for NoC-based MPSoC Generation*. In: ISCAS, 2009, pp. 1345-1348.

[CHI00] Chiu, G. *The Odd-Even Turn Model for Adaptive Routing*. IEEE Transactions on Parallel and Distributed Systems, v.11(7), 2000, pp. 729-738.

[CHO09] Cho, M.; Lis, M.; Kinsy, M.; Shim, K.; Wen, T.; Devadas, S. *Oblivious Routing in On-Chip Bandwidth-Adaptive Networks*. In: PACT, 2009, pp. 181-190.

[CON09] Concatto, C.; Matos, D.; Carro, L.; Kastensmidt, F.; Susin, A.; Cota, E.; Kreutz, M. *Fault tolerant mechanism to improve yield in NoCs using a reconfigurable router*. In: SBCCI, 2009, 6p.

[FIC09] Fick, D.; DeOrio, A.; Hu, J.; Bertacco, V.; Blaauw, D.; Sylvester, D. *Vicis: A Reliable Network for Unreliable Silicon*. In: DAC, 2009, pp. 812-817.

[FIC09a] Fick, D.; DeOrio, A.; Chen, G.; Bertacco, V.; Sylvester, D.; Blaauw, D. *A highly resilient routing algorithm for fault-tolerant NoCs*. In: DATE, 2009, pp. 21-26.

[FOC15] Fochi, V.; Wachter, E.; Erichsen, A.; Amory, A.; Moraes. *An Integrated Method for Implementing Online Fault Detection in NoC-based MPSoCs*. In: ISCAS, 2015, 4p.

[GRE07] Grecu, C.; Anghel. L.; Pande, P.; Ivanov, A.; Saleh, R. *Essential Fault-Tolerance Metrics for NoC Infrastructures.* In: IOLTS, 2007, pp. 37-42.

[IEE90] Institute of Electrical and Electronics Engineers (1990). *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York, NY ISBN 1-55937-079-3

[JHA03] Jha, N.; Gupta, S. *Testing of Digital Systems*. Cambridge University Press, 2003, 680p.

[LAN11] Lan Y.; Lo, S.; Lin, Y.; Hu, Y.; Chen, S. *A Bidirectional NoC (BiNoC) Architecture With Dynamic Self-Reconfigurable Channel*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.30(3), 2011, pp. 427-440.

[MEL12] Meloni, P.; Tuveri, G.; Raffo, L.; Cannella, E.; Stefanov, T.; Derin, O.; Fiorin, L.; Sami, M. *System Adaptivity and Fault-tolerance in NoC-based MPSoCs: the MADNESS Project Approach*. In: DSD, 2012, pp. 517-524.

[OPE10] OpenCores. *Plasma-most MIPS I (TM) opcodes: overview*. Capturado em: http://opencores.org/project,plasma, Jul. 2010.

[SIL10] Da Silva, A. *Implementação e Avaliação de Métodos para Confiabilidade de Redes Intra-Chip*. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2010, 89p.

[TSA11] Tsai, W.; Zheng, D.; Chen, S.; Hu, Y. *A Fault-Tolerant NoC Scheme Using Bidirectional Channel*. In: DAC, 2011. pp. 918-923.

[VAL11b] Valinataj, M. *Evaluation of Fault-Tolerant Routing Methods for NoC Architectures*. In: DSD, 2011, pp. 446-449.

[VAL11] Valinataj, M.; Liljeberg, P.; Plosila, J. *A Fault-Tolerant and Hierarchical Routing Algorithm for NoC Architectures*. In: NORCHIP, 2011, 6p.

[VEI10] Veiga, F.; Zeferino, C. *Implementation of Techniques for Fault Tolerance in a Network-on-Chip*. In: WSCAD-SCC, 2010, pp. 80-87.

[VER07] Verdoolaege, S.; Nikolov, H.; Stefanov, T. *PN: a tool for improved derivation of process networks*. EURASIP Journal on Embedded Systems, v.2007(1), 2007, 13p.

[VIT12] Vitkovskiy, A.; Soteriou, V, and Nicopoulos, C, *A Dynamically Adjusting Gracefully Degrading Link-Level Fault-Tolerant Mechanism for NoCs.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.31(8), 2012, pp. 1235-1248.

[WAC13] Wachter, E.; Erichsen, A.; Amory, A.; Moraes, F. *Topology-Agnostic Fault-Tolerant NoC Routing Method*. In: DATE, 2013, pp. 1595-1600.

[WAC12] Wachter, E.; Moraes, F. *MAZENOC: Novel approach for fault-tolerant NOC routing*. In: SOCC, 2012, pp. 364 - 369.

[WOS07] Woszezenki, C. *Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs*. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2007, 121p.

[ZEF03] Zeferino, C.A. *SoCIN: a parametric and scalable network-on-chip*. In: SBCCI, 2003, pp. 169-174.