UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA, INSTITUTO DE FÍSICA, INSTITUTO DE QUÍMICA, ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

MARCOS BARCELLOS HERVÉ

Métodos de Teste de Redes-em-Chip (NoCs)

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Microeletrônica

Prof. Dr. Marcelo Soares Lubaszewski
Orientador

Porto Alegre, agosto de 2009.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Hervé, Marcos Barcellos

Métodos de Teste de Redes-em-Chip (NoCs) / Marcos Barcellos Hervé – Porto Alegre: Programa de Pós-Graduação em Microeletrônica, 2009.

119 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica. Porto Alegre, BR – RS, 2009. Orientador: Marcelo Soares Lubaszewski.

1.Redes-em-Chip. 2.Teste. 3.Localização de Falhas. I. Lubaszewski, Marcelo Soares. II. Título.

AGRADECIMENTOS

Gostaria de agradecer aos meus pais, Ana Maria Barcellos Hervé e Hegel Mesquita Hervé, pelo incentivo, carinho e por terem me dado todas as condições de alcançar meus objetivos, tanto os acadêmicos como os pessoais. Também gostaria de agradecer aos meus irmãos, Bruno e Cássio, pelo apoio e à minha namorada, Monique, pelo amor e por todos os momentos juntos.

Gostaria, também, de agradecer a todos os meus professores, especialmente, ao meu orientador de mestrado professor Marcelo Lubaszewski pelos ensinamentos e pela amizade, e às professoras Érika Cota e Fernanda Kastensmidt pela ajuda nas atividades acadêmicas. Agradeço ainda a secretária do PGMicro, Zíngara Lubaszewski, pelo auxílio na obtenção de toda a documentação necessária durante o curso.

Também agradeço aos colegas da universidade e a todos os meus amigos pelo auxílio durante o curso e pelos momentos de descontração.

Por fim, gostaria de agradecer à UFRGS, com seus professores e funcionários, e à comunidade pela oportunidade de receber uma educação pública de qualidade.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 9

LISTA DE FIGURAS 11

LISTA DE TABELAS 13

RESUMO 15

ABSTRACT 17

INTRODUÇÃO 19

1 NOÇÕES DE REDES-EM-CHIP 21

1.1 Redes-em-Chip 21

1.2 Propriedades das Redes-em-Chip 22
1.2.1 Topologia 22
1.2.2 Roteamento 24
1.2.3 Chaveamento 25
1.2.4 Controle de Fluxo 26
1.2.5 Arbitragem 26
1.2.6 Memorização 27

1.3 Relação das Redes-em-Chip com as Camadas OSI-ISO 27

1.4 Interface Padrão Para Interconexão de Núcleos 28

1.5 Estudo de Caso: SoCIN 28

RESUMO DO CAPÍTULO 1 30

2 TESTE DE SOCS BASEADOS EM NOCS 31

2.1 Conceitos de Teste 31
2.1.1 Verificação Funcional x Teste 31
2.1.2 Falhas e Defeitos 32
2.1.3 DfT – *Design-for-Testability* 33

2.1.4 IEEE std. 1149.1 – *Boundary Scan* 33
2.1.5 IEEE std. 1500 – Teste de SoCs 36
2.1.6 Mecanismos de Acesso ao Teste (TAM – T*est Access Mechanism*) 37

2.2 Teste de Redes-em-Chip 38
2.2.1 Teste dos Roteadores 38
2.2.2 Teste das Interconexões 46
2.2.3 Comparação dos Métodos 50

RESUMO DO CAPÍTULO 2 51

3 MÉTODO PARA O TESTE FUNCIONAL DAS INTERCONEXÕES DE REDES-EM-CHIP 53

3.1 Teste Funcional das Interconexões da NoC – Sinais de Dados 53
3.1.1 NoC Utilizada – Estudo de Caso 53
3.1.2 Modelo de Falhas 53
3.1.3 Ambiente de Simulação 54
3.1.4 Método de Envio de Mensagem na Rede Para o Teste das Interconexões 55
3.1.5 Seqüência de Teste 57
3.1.6 Composição do Pacote da Mensagem 57
3.1.7 Escalabilidade do Método de Teste 60
3.1.8 Simulações 61
3.1.9 Resultados 63

3.2 Teste Funcional das Interconexões da NoC – Inclusão dos Sinais de Controle 63
3.2.1 NoC Utilizada – Estudo de Caso 64
3.2.2 Modelo de Falhas 64
3.2.3 Ambiente de Simulação 64
3.2.4 Método do Envio de Mensagem na Rede Para o Teste das Interconexões 64
3.2.5 Composição do Pacote da Mensagem 65
3.2.6 Escalabilidade do método de teste 67
3.2.7 Resultados 67

RESUMO DO CAPÍTULO 3 70

4 ESTRUTURAS DE TESTE (CIRCUITOS DE BIST) 71

4.1 TDG – Características 71

4.2 TDG – Configuração 72

4.3 TRA – Características 75

4.4 TRA – Configurações 77

4.5 Configuração das Estruturas de Teste 78

4.6 Configuração das Estruturas de Teste Através de Cadeias *Scan* 78

4.7 Configuração das Estruturas de Teste Através de uma Envoltória de Teste 79

4.8 Teste dos Circuitos de BIST 82

4.9 Experimentos 82

RESUMO DO CAPÍTULO 4 86

5 CAPACIDADE DE LOCALIZAÇÃO DE FALHAS 87

5.1 Modelo de Falhas 87

5.2 NoC Utilizada – Estudo de Caso 89

5.3 Método de Localização de Falhas 89

5.4 Avaliação da Capacidade de Localização de Falhas do Método de Teste das Interconexões Apresentado na Seção 3.1 89

5.5 Método de Teste Modificado Visando a Localização de Falhas 91

5.6 Extensão do Método de Localização de Falhas para Casos Não Resolvidos 95

5.7 Ambiente de Simulação 98

5.8 Resultados 98

RESUMO DO CAPÍTULO 5 100

6 INCLUSÃO DO TESTE DOS ROTEADORES 101

6.1 NoC Utilizada – Estudo de Caso 101

6.2 Modelo de Falhas 101

6.3 Ambiente de Simulação 102

6.4 Teste dos Roteadores da Rede 102

6.5 Teste dos Roteadores da Rede – FIFOs 103

6.6 Teste dos Roteadores da Rede – Lógica de Roteamento 104

6.7 Teste dos Roteadores da Rede – Lógica de Arbitragem 106

6.8 Teste dos Roteadores da Rede – Interconexões 109

6.9 Resultados 111

RESUMO DO CAPÍTULO 6 114

7 CONCLUSÕES 115

REFERÊNCIAS 117

LISTA DE ABREVIATURAS E SIGLAS

IP	Intellectual Property
SoC	System-on-Chip
NoC	Network-on-Chip
VCI	Virtual Component Interface
OCP	Open Core Protocol
VSIA	Virtual Socket Initiative Alliance
FIFO	First-in-First-out
SoCIN	System on Chip Interconnection Network
RASoC	Router Architecture for System on Chip
BIST	Built in Self Test
TAM	Test Access Mechanism
ATE	Automatic Test Equipment
MAF	Maximum Aggressor Fault
FCFS	First Come Fist Served
LRS	Least Recently Served
OSI	Open Systems Interconnection
IEEE	Institute of Electrical and Electronics Engineers
RAM	Random Access Memory
TAS	Test Access Switch
GTC	Global Test Controller

LISTA DE FIGURAS

Figura 1. 1: Redes diretas: (a) grelha 2-D; (b) toróide 2-D; (c) hipercubo 3-D (ZEFFERINO, 2003). ...23
Figura 1. 2: Redes indiretas: (a) núcleo de chaveamento 4x4; (b) multiestágio 8x8 bidirecional (ZEFFERINO, 2003). ...24
Figura 1. 3: Representação do roteador RASoC com 5 portas de comunicação (ZEFERINO, 2003).....29
Figura 2 1: Célula Boundary-Scan (IEEE, 2001). ...34
Figura 2 2: Máquina de estados do controlador TAP do padrão boundary Scan (IEEE, 2001). ...35
Figura 2 3: Envoltória de teste do padrão IEEE std 1500 (IEEE, 2005). ...37
Figura 2 4: Metodologia de acesso ao teste proposto (AMORY, 2005). ...39
Figura 2 5: Envoltória de teste proposta (AMORY, 2005). ...40
Figura 2 6: Uso de células Boundary scan para teste dos roteadores de uma NoC (AKTOUF, 2002). ...40
Figura 2 7: Estrutura de teste baseada em boundary scan para teste dos roteadores da rede com uso de comparadores internos para análise de resultados (AKTOUF, 2002). ...41
Figura 2 8: Circuitos de BIST compartilhados para teste de memória nos roteadores da NoC (GRECU, 2005). ...42
Figura 2 9: Representação do esquema de teste para comunicação multicast baseado no reuso da NoC proposto. "N" representa operação normal e "T" representa modo de teste (GRECU, 2005). ...44
Figura 2 10: Representação do esquema de teste para comunicação unicast baseado no reuso da NoC proposto. "N" representa operação normal e "T" representa modo de teste (GRECU, 2005). ...44
Figura 2 11: Representação do TAS (Test Acess Switch) proposto em (HOSSEINABADY, 2006). ...45
Figura 2 12: Envoltória de teste do esquema proposto em (HOSSEINABADY, 2006). ...45
Figura 2 13: Wrapper de teste do esquema de teste proposto em (LIU, 2006). ...46
Figura 2 14: Falhas representadas pelo modelo MAF (GRECU, 2006_a). ...47
Figura 2 15: Vetores de teste para detecção das falhas do modelo MAF (GRECU, 2006_a). A figura representa um conjunto de 8 vetores de teste a serem aplicados para cada fio vítima (i), sendo que os demais fios recebem os valores atribuídos aos fios agressores. ...48
Figura 2 16: Utilização de pares TDG/TED para teste das interconexões da NoC (GRECU, 2006_a). ...48
Figura 2 17: Representação do teste utilizando apenas um TDG e um GTC com comunicação unicast (GRECU, 2006_a), onde os canais em negrito representam a transmissão dos dados. ...49
Figura 2 18: Representação do teste utilizando apenas um TDG e um GTC com comunicação multicast (GRECU, 2006_a), onde os canais negrito representam a transmissão dos dados. ...49
Figura 3. 1: Representação do ambiente de simulação utilizado. ...55
Figura 3. 2: Representação do pacote enviado na rede. ...56
Figura 3. 3: Representação do circuito sob teste. Os nós representam os circuitos ligados aos roteadores, responsáveis pelo envio, recebimento da mensagem, além da sinalização de erro. Estes circuitos possuem uma interface de rede (network interfaces) representada externamente aos nós. Os elementos centrais representam os roteadores (routers). ...57
Figura 3. 4: Composição do pacote de teste enviado na rede. ...58
Figura 3. 5: Representação da forma de envio dos pacotes pelos nós em função do tempo. A figura mostra que os quatro pacotes são enviados ao mesmo tempo em um instante inicial. O pacote enviado por cada nó é composto de tal forma que os vetores de teste presentes entre os flits de payload acabam defasados. ...58
Figura 3. 6: Rodadas de teste para uma NoC de tamanho 5x5. (a) mostra a primeira rodada de teste; (b) mostra a segunda rodada de teste; (c) mostra a terceira rodada de teste; (d) mostra a quarta rodada de teste. As áreas em cinza representam as configurações de teste que podem rodar paralelamente numa mesma rodada de teste. ...60
Figura 3. 7: Forma de onda com o envio dos pacotes de teste. ...62

Figura 3. 8: Falha na informação de roteamento da mensagem. 62
Figura 3. 9: Falha no vetor de teste enviado na rede. O valor lógico que deveria ser 1 em um determinado fio está em nível lógico 0. 62
Figura 3. 10: Representação do envio de pacotes do método apresentado em 3.1. Na figura pode ser visto que os bits de bop (begin of packet) e eop (end of packet) possuem o mesmo valor lógico num mesmo instante de tempo, impossibilitando, assim, a detecção das falhas. 65
Figura 3. 11: Representação da forma de envio dos pacotes pelos nós em função do tempo. A figura mostra que os quatro pacotes são enviados de forma defasada. 66
Figura 3. 12: Composição do pacote de teste enviado na rede. 67
Figura 3. 13: A figura mostra a representação do controle de fluxo do roteador RASoC e as formas de onda dos sinais de controle durante uma operação: (a) sem falhas; (b) na presença de uma falha do tipo wired-AND entre in_val e in_ack; (c) na presença de falhas do tipo wired-OR entre in_val e in_ack. 69
Figura 4. 1: Rodadas de teste para uma NoC de tamanho 4x4. 73
Figura 4. 2: Envio dos pacotes na rede em função do tempo. (a) Método de teste sem inclusão dos sinais de controle (3.1); (b) Método de teste incluindo os sinais de controle (3.2). 75
Figura 4. 3: Envoltórias de teste proposto para facilitar o acesso ao teste. 80
Figura 4. 4: Resultados de tempo total de teste para as diferentes implementações. (a) cadeia scan única; (b) uso de envoltória de teste e cadeia única de registradores; (c) uso de envoltória de teste e cadeia dupla de registradores (uma cadeia para os TDGs e outra cadeia para os TRAs). 84
Figura 4. 5: Comparação do tempo de teste entre os diferentes métodos propostos em função da largura do canal para uma NoC de tamanho 4x4. 85
Figura 5. 1: Vizinhança de teste do método apresentado em 3.1. 87
Figura 5. 2: Exemplo de erro em um bit do canal de comunicação. A falha pode ter ocorrido nos caminhos “a”, “b” ou “c”, porém não é possível localizar o exato local da falha. 89
Figura 5. 3: Ciclos de envio de mensagem para o método de teste modificado visando localização de falhas. (a) Ciclo 1; (b) Ciclo 2. 92
Figura 5. 4: Ciclos de teste adicionais visando o aumento da capacidade de localização de falhas.(a) Ciclo 3; (b) Ciclo 4; (c) Ciclo 5. 97
Figura 6. 1: Teste das interconexões de uma NoC 4x4. 103
Figura 6. 2: Rodada de teste adicionais para detecção de falhas na lógica de roteamento. 106
Figura 6. 3: Configurações de teste adicionais para detecção de falhas na lógica de arbitragem. 108
Figura 6. 4: (a) Vizinhança de teste proposta no método de teste das interconexões; (b) e (c) vizinhança de teste estendida devido às rodadas extra para o teste dos roteadores. 110

LISTA DE TABELAS

Tabela 1. 1: Tabela com as características da rede SoCIN. *28*
Tabela 2. 1 Quadro comparativo dos métodos apresentados nas seções 2.2.1 e 2.2.2. ..... *50*
Tabela 3. 1: Resultados da simulação do método apresentado. *63*
Tabela 3. 2: Resultados da simulação do teste incluindo os sinais de controle na análise. *67*
Tabela 4. 1 Máquina de estados do TDG. *71*
Tabela 4. 2: Máquina de estados do TRA. *75*
Tabela 4. 3: Resultados de área das estruturas de teste. *82*
Tabela 5. 1: Combinação de sinalização de erros para o método descrito na seção 3.1. ..... *90*
Tabela 5. 2: Lista dos canais suspeitos de apresentarem falhas (falhas de curto circuito entre fio i do canal u e fio j do canal v para o exemplo citado no texto). *90*
Tabela 5. 3: Classificação dos resultados de localização de falhas para o método de teste descrito na seção 3.1. ..... *91*
Tabela 5. 4: Canais suspeitos de falha para o exemplo apresentado. *93*
Tabela 5. 5: Possíveis combinações de sinalização de erro por parte das estruturas de teste após os dois ciclos de envio de mensagem de teste na rede (R = Resolvido, N = Não resolvido, I = Impossíveis de ocorrer para o modelo de falhas adotado). *94*
Tabela 5. 6: Resultados das simulações para validação do método de localização de falhas. *99*
Tabela 6. 1: Resultado do teste das interconexões de uma NoC 2x2 (conforme apresentado em 3.2). *103*
Tabela 6. 2: Resultado parcial de cobertura de falhas para o teste das FIFOs de um roteador central de uma NoC 3x3. ..... *104*
Tabela 6. 3: Resultados parciais do método de teste visando as falhas na lógica de roteamento de um roteador central em uma NoC 3x3. ..... *106*
Tabela 6. 4: Representação dos flits de payload do pacote enviado para deteção de falhas na lógica de arbitragem. *108*
Tabela 6. 5: Acréscimo na cobertura de falhas devido ao método de teste que visa as falhas na lógica de arbitragem de um roteador central de uma NoC 3x3. ..... *109*
Tabela 6. 6: Resultados incrementais de cobertura de falhas nas interconexões da vizinhança estendida da NoC sob teste. *110*
Tabela 6. 7: Resultados dos experimentos para validação da metodologia proposta. *112*

RESUMO

Este trabalho tem como objetivo estudar e propor métodos de teste funcional visando a detecção e localização de falhas na infra-estrutura das redes-em-chip. Para isso, o trabalho apresenta, inicialmente, uma descrição das principais características das redes-em-chip, explicando o que elas são e para que elas servem. Em seguida são apresentados conceitos de teste de circuitos integrados, bem como trabalhos relacionados ao teste das redes-em-chip.

Um método de teste visando a detecção de falhas nas interconexões de dados de uma NoC é apresentado no trabalho, sendo este método posteriormente estendido para incluir as interconexões de controle. Os circuitos de teste necessários para implementar a estratégia de teste proposta também são descritos.

A partir do método de teste apresentado, é feito um estudo sobre sua capacidade de localização de falhas, onde alterações visando o aumento dessa capacidade de localização de falhas são propostas. Por fim o método de teste é estendido para detecção de falhas nos roteadores da rede.

Palavras-Chave: Redes-em-Chip, teste, localização de falhas, roteadores, interconexões

ABSTRACT

The purpose of this work is to study and propose functional test methods that aim the detection and location of faults in the NoC's infrastructure. In order to do so, this work presents, initially, a description of the main characteristics of networks-on-chip, explaining what are NoCs and what is their purpose. Fallowing this description, some concepts related to the test of integrated circuits are presented as well as related works on NoC testing.

A method aiming the detection of data interconnect faults in a NoC is presented in this work. This method is later extended to include faults in the control interconnections as well. The circuits used to implement the proposed strategy are also described here.

Based on the proposed test strategy, the method's capability to locate faults is studied. Changes are proposed to the test method in order to increase this fault location capability. Finally, the test method is extended to include faults inside the router's logic.

Keywords: Networks-on-Chip, test, fault location, routers, interconnections.

INTRODUÇÃO

Com a crescente quantidade de módulos de propriedade intelectual (IP) sendo integrados dentro de um único chip, os circuitos integrados têm agregado um número cada vez maior de funcionalidades. Os sistemas-em-chip (SoCs) são exemplos de circuitos que fazem uso dessa capacidade de integração, contendo vários módulos com funcionalidades distintas que se comunicam formando um sistema.

Do ponto de vista de desempenho, uma parte crítica do projeto de SoCs está ligada à comunicação entre os diferentes módulos IP que compõem o sistema. Soluções baseadas em barramentos são predominantes na indústria, sendo citadas como exemplo: ARM AMBA, IBM CoreCoennect, Silicore Corp. WISHBONE e Motorola IP Interface (ZEFFERINO, 2003). Contudo, este tipo de solução apresenta limitações de desempenho conforme mais módulos IP são inseridos num mesmo chip. Estas limitações ocorrem pelo fato da largura de banda de um barramento ser compartilhada entre os dispositivos conectados a ele (GUERRIER, 2000). Uma solução para as limitações impostas pelo uso de barramentos na comunicação intra-chip é a utilização de redes de chaveamento de pacotes. Esta solução apresenta ganhos em desempenho, devido ao paralelismo da rede, podendo, também, apresentar uma redução no consumo de potência (BENINI, 2002).

As redes de interconexão baseadas em chaveamento de pacote usadas para comunicação intra-chip também são chamadas de Redes-em-Chip ou NoCs (*Networks-on-Chip*). A grande vantagem na utilização dessas estruturas está ligada às suas características de paralelismo, escalabilidade e reusabilidade. A testabilidade das redes-em-chip e de sistemas com comunicação baseada em NoCs é outra característica a ser avaliada.

Neste trabalho, as NoCs são abordadas sob o ponto de vista da etapa de teste de circuitos integrados. Esta etapa é responsável por averiguar se o circuito implementado possui algum defeito decorrente do processo de fabricação. Em grandes sistemas-em-chip, esta etapa de projeto torna-se bastante dispendiosa devido à baixa observabilidade e controlabilidade dos sinais internos a estes sistemas (ZORIAN, 1999). Dessa forma, a metodologia de teste deve ser pensada já na etapa de projeto do circuito, fazendo uso de estruturas específicas que visam o teste do mesmo.

O teste dos sistemas-em-chip baseados em NoCs tem sido objeto de pesquisa do meio acadêmico e industrial, visto a grande quantidade de publicações recentes na área. Conforme mencionado, a etapa de teste dos circuitos integrados pode ser bastante dispendiosa (principalmente em grandes SoCs) podendo compor, assim, uma grande parcela do custo total do chip. Dessa forma, a motivação para o estudo de métodos eficientes de teste está ligada, em última instância, ao custo final do chip, além de sua qualidade. O desafio no teste de SoCs baseados em NoCs está ligado à necessidade de

testar os núcleos do sistema, bem como sua rede de interconexão (NoC) aproveitando os benefícios trazidos por este novo paradigma de comunicação intra-chip. Levando-se em conta que a própria rede de interconexão pode ser reutilizada para carregar os vetores de teste aos núcleos do sistema (COTA, 2004), o teste da rede em si se torna muito importante, motivando, assim, o presente trabalho.

Este trabalho tem como objetivo estudar e propor métodos de teste funcional visando a detecção e localização de falhas na infra-estrutura das redes-em-chip. O trabalho está dividido da seguinte forma: o capítulo 1 faz uma descrição das redes-em-chip, explicando suas principais características. No capítulo 2 são apresentados conceitos de teste de circuitos integrados, bem como trabalhos relacionados ao teste das redes-em-chip. O capítulo 3 apresenta um método de teste visando a detecção de falhas nas interconexões de dados de uma NoC, sendo este método posteriormente estendido para incluir as interconexões de controle. Os circuitos de teste necessários para implementar a estratégia de teste proposta são apresentados no capítulo 4. O capítulo 5 apresenta um estudo sobre a capacidade de localização de falhas do método de teste proposto, onde alterações visando o aumento dessa capacidade de localização de falhas são propostas. Por fim o capítulo 6 estende o método de teste para detecção de falhas nos roteadores da rede e o capítulo 7 conclui o trabalho.

1 NOÇÕES DE REDES-EM-CHIP

As redes de interconexão baseadas em chaveamento de pacote usadas para comunicação intra-chip também são chamadas de Redes-em-Chip ou NoCs (*Networks-on-Chip*). A grande vantagem na utilização dessas estruturas está ligada às suas características de paralelismo, escalabilidade e reusabilidade. A rede utiliza circuitos responsáveis por fazer o roteamento da mensagem de um núcleo (módulo IP) origem a um núcleo destinatário.

Neste capítulo, são apresentadas as principais características de uma NoC.

1.1 Redes-em-Chip

As Redes-em-Chip (NoCs) são estruturas de interconexão que tem como objetivo fazer a ligação de diversos núcleos dentro de um sistema em chip. A idéia por trás dessas redes baseia-se nos conceitos de redes de computadores (BENINI, 2002) e são alternativas aos barramentos ou ligações ponto-a-ponto entre os núcleos do sistema (que podem ser processadores, memórias ou até mesmo um computador completo, com memória local) (ZEFFERINO, 2003).

A rede responsável pelas ligações, também chamada de rede de interconexões, é formada por roteadores e interconexões. Essas estruturas são descritas a seguir.

- Roteadores: São responsáveis por encaminhar a mensagem ao seu destino. São formados por:
    - Núcleo de chaveamento (*crossbar*);
    - Lógica de controle para roteamento e arbitragem;
    - Portas de comunicação para outros roteadores e para o núcleo local (canais de entrada e saída). Nessas portas ainda podem existir estruturas de controle, para o protocolo de comunicação e buffers, para memorização.
- Interconexões: São responsáveis pela ligação dos roteadores entre si e com os núcleos. Formados por canais unidirecionais com sentidos opostos, estabelecendo uma comunicação do tipo *full duplex* (onde a transmissão pode ser feita nos dois sentidos simultaneamente), ou por canais bidirecionais.

A transmissão de mensagens na rede é feita do núcleo fonte ao núcleo destino. A mensagem pode, ou não, ser dividida em pacotes de tamanhos menores. Em geral, os pacotes possuem sua estrutura separada em campos que representam as diferentes informações necessárias para a transmissão da mensagem, como por exemplo: um campo de cabeçalho (onde é colocada a informação de roteamento), um campo de carga

útil (onde é colocada a informação a ser transmitida) e um campo terminador (ou *trailer*, com informação de final de pacote). O pacote pode ainda ser dividido em *flits*, que são a menor unidade de dados sobre a qual é realizado controle de fluxo. O tamanho da mensagem é limitado pelo tamanho físico do canal. Essa largura física de um canal do roteador é denominada *phit* (ZEFFERINO, 2003).

1.2 Propriedades das Redes-em-Chip

As redes-em-chip apresentam as seguintes propriedades (ZEFFERINO, 2003):

- Topologia: Define o arranjo dos roteadores e das interconexões sob a forma de um grafo.
- Roteamento: Determina como uma mensagem escolhe o caminho dentro da rede.
- Chaveamento: Define como e quando um canal de entrada é conectado a um canal de saída selecionado pelo caminho de roteamento.
- Controle de fluxo: Lida com a alocação de canais e *buffers* para uma mensagem que atravessa a rede.
- Arbitragem: Determina qual canal de entrada pode utilizar um determinado canal de saída do roteador. O roteamento define a saída e o árbitro define a entrada.
- Memorização: Define como e onde são armazenadas mensagens bloqueadas de um roteador.

Essas características são detalhadas nos itens listados a seguir.

1.2.1 Topologia

As redes podem ser classificadas em dois tipos: as redes diretas e as redes indiretas.

As redes diretas apresentam os roteadores ligados aos núcleos formando uma estrutura única, denominada nodo. Para uma mensagem ser transmitida de um nodo para outro não adjacente (não vizinho) na rede, é necessária a passagem por nodos intermediários. Nesses nodos a mensagem é apenas passada adiante pelo roteador associado ao nodo. (ZEFFERINO, 2003).

Uma rede com conectividade ideal apresenta os nodos completamente conectados entre si, ou seja, cada nodo é ligado a todos os outros da rede (conexões ponto a ponto entre todos os nodos da rede). O custo dessa implementação, contudo, passa a ser proibitivo a medida que aumenta o número de nodos da rede, já que, para esta implementação, cada nodo teria que ter N-1 canais de entrada e saída (onde N é o número total de nodos da rede).

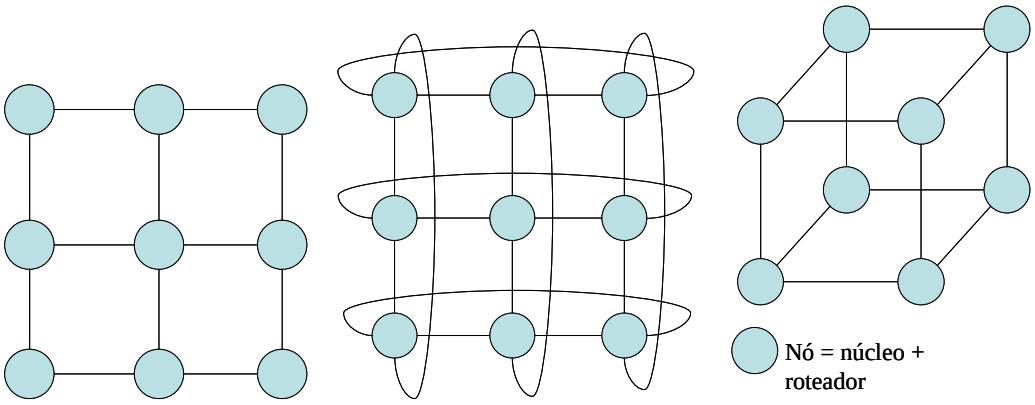
Uma solução para este problema é o uso de redes ortogonais. Uma rede apresenta topologia ortogonal se seus nodos podem ser arranjados em um espaço n-dimensional e cada link entre nós vizinhos produz um deslocamento em apenas uma dimensão. Dentre as redes diretas ortogonais as topologias mais utilizadas são a grelha n-dimensional, o toróide e o hipercubo (ZEFERINO, 2003).

Já as redes indiretas apresentam roteadores independentes dos núcleos, de forma que os núcleos devem apresentar uma interface compatível com a rede de roteadores. Dentre as topologias de redes indiretas, destacam-se o crossbar e as redes multiestágio.

A topologia crossbar caracteriza-se por apresentar um único roteador com uma chave NxN (onde N é o número de núcleos ligados a rede). Em termos de conectividade esta é uma rede indireta ideal, por fazer a conexão entre todos os núcleos ligados à rede passando por apenas um elemento de roteamento.

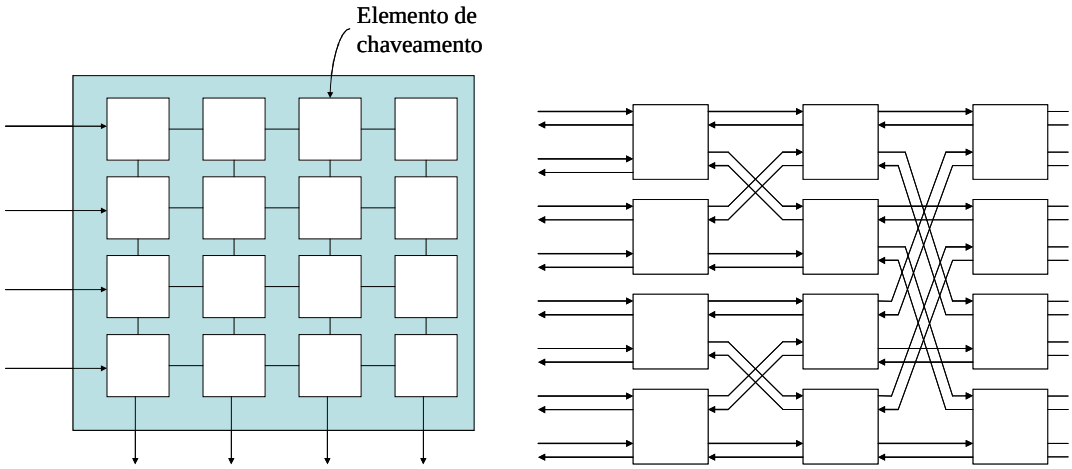
Já a topologia multiestágio faz uso de uma série de roteadores ligados entre si, de forma que a mensagem deve passar por elementos intermediários de roteamento. Nessa topologia, os roteadores são arranjados em estágios, sendo que o primeiro e o ultimo são estágios de entrada e saída, ligados aos núcleos. Os estágios intermediários são ligados entre si através de padrões regulares de conexão (ZEFERINO, 2003).

Exemplos de topologias de rede direta são apresentados na figura 1.1. Exemplos de topologias de redes indiretas são apresentados na figura 1.2.



(a) (b) (c)

Figura 1. 1: Redes diretas: (a) grelha 2-D; (b) toróide 2-D; (c) hipercubo 3-D (ZEFFERINO, 2003).



(a) (b)

Figura 1. 2: Redes indiretas: (a) núcleo de chaveamento 4x4; (b) multiestágio 8x8 bidirecional (ZEFFERINO, 2003).

1.2.2 Roteamento

O roteamento é o método que determina como uma mensagem escolhe um caminho a ser percorrido dentro da rede para atingir seu destino. O algoritmo de roteamento utilizado tem forte impacto no desempenho da comunicação. Dessa forma, algumas características devem ser observadas como: a capacidade de rotear mensagens de qualquer fonte para qualquer destino (conectividade); a capacidade de garantir que nenhuma mensagem ficará bloqueada ou circulando na rede sem atingir o destino; a capacidade de fazer o roteamento através de caminhos alternativos, no caso de congestionamento ou falhas (adaptatividade); a capacidade de rotear a mensagem corretamente na presença de falhas (tolerância a falhas).

Existem na literatura diversos algoritmos de roteamento, sendo que sua classificação pode ser feita através das seguintes características:

- Momento de realização do roteamento: Se o algoritmo de roteamento é executado em tempo de execução da aplicação ele é dito dinâmico. Caso o algoritmo de roteamento seja realizado no tempo de compilação da aplicação ele é dito estático.
- Número de destinos: Pode haver um único destino para as mensagens (*unicast*), ou múltiplos destinos (*multicast*).
- Lugar onde as decisões de roteamento são tomadas: A definição do caminho a ser seguido pela mensagem pode ser feita por um controlador central (centralizado), pelo nodo emissor da mensagem (fonte), ou ainda, pelos roteadores enquanto as mensagens atravessam a rede (distribuído).
- Implementação: Pode ser feita baseada em tabela, por meio de leitura na memória, ou baseado em máquina de estados.
- Adaptatividade: Pode ser classificado em determinístico ou adaptativo. No caso determinístico, é feito sempre o mesmo caminho pela mensagem entre fonte e

destino. No caso adaptativo é possível fazer caminhos alternativos entre fonte e destino. Neste caso, pode-se classificar ainda em termos de progressividade, minimalidade e número de caminhos. A progressividade define se o cabeçalho avança pela rede reservando um canal a cada passo de roteamento (progressivo), ou se o cabeçalho retorna na rede liberando canais previamente reservados (regressivo). Quanto à minimalidade, o caminho pode ser mínimo, sempre levando a caminhos mais próximos do destino, ou não mínimo, podendo selecionar canais que levem a mensagem a se afastar do destino. Por fim, o número de caminhos define se o algoritmo de roteamento pode utilizar todos os caminhos disponíveis (completo), ou se apenas um subconjunto desses caminhos pode ser utilizado (parcial).

1.2.3 Chaveamento

O chaveamento define a forma pela qual os dados são transferidos do canal de entrada para um canal de saída. Alguns exemplos de técnica de chaveamento são: chaveamento por circuito, chaveamento por pacote *store-and-forward*, chaveamento por pacote virtual *cut-through* e chaveamento por pacote *worm hole* (ZEFERINO, 2003).

1. Chaveamento por circuito:

   No chaveamento por circuito é estabelecido um caminho físico entre fonte e destino para a transmissão da mensagem. Uma vez estabelecido o caminho, este será mantido até o término da transmissão da mensagem, de forma que se alguma outra mensagem necessitar a utilização dos mesmos canais de roteamento, esta mensagem não será transmitida. Este tipo de roteamento é justificado para mensagens longas e pouco freqüentes.

2. Chaveamento por pacote *store-and-forward*:

   No chaveamento por pacotes *store-and-forward*, a mensagem é dividida em pacotes de tamanho fixo, contendo cabeçalho, a carga útil e um terminador. Estes pacotes são armazenados em elementos de memória (*buffers*) com capacidade de armazenar um pacote inteiro, presentes em cada roteador do caminho da mensagem. Uma vez armazenado no buffer, o pacote é identificado pelo roteador que o direciona ao canal de saída adequado.

3. Chaveamento por pacote virtual *cut-through*:

   Esta técnica de chaveamento é similar à anterior (por pacote *store-and-forward*), contudo o pacote não é armazenado inteiramente no *buffer* do roteador quando o canal de saída requisitado pela mensagem está disponível. Nesse caso a mensagem é repassada diretamente. No pior caso, o chaveamento por pacote virtual *cut-through* se comporta como o chaveamento por pacote *store-and-forward*.

4. Chaveamento por pacote *worm hole*:

   Esta técnica de chaveamento é uma variação do chaveamento por pacote virtual *cut-through*. O pacote da mensagem é dividido em *flits* que avançam pela rede em um modo *pipeline*. Com isso, os elementos de memória presentes nos roteadores que fazem parte do caminho da mensagem podem ser menores, já que estes devem armazenar apenas alguns *flits* ao invés do pacote inteiro.

Na medida que os canais de saída dos roteadores, requisitados pela mensagem, tornam-se disponíveis, os *flits* são repassados à rede. Dessa forma, se um canal de saída requisitado pela mensagem não se encontra disponível, os *flits* da mensagem ficarão armazenados nos buffers dos roteadores.

1.2.4 Controle de Fluxo

O controle de fluxo é a política utilizada para decidir o que deve ser feito com um determinado pacote no caso de um recurso requisitado, por exemplo, um determinado canal de saída, já estar sendo utilizado (colisão de recurso).

Um exemplo simples de controle de fluxo é o protocolo de aperto de mão (*handshake*). Este protocolo utiliza sinais que avisam a intenção do emissor enviar dados na rede e sinalizam a disponibilidade de recursos na rede para a recepção e transmissão desses dados. Outros exemplos de controle de fluxo são: controle de fluxo baseado em *slack-buffer*, controle de fluxo baseado em canais virtuais e controle de fluxo baseado em créditos.

1. Controle de fluxo baseado em *Slack-buffers*: É baseado em um sinal de controle de nível do *buffer*, que avisa quando este se encontra cheio ou vazio. Esta sinalização é utilizada para avisar ao emissor a possibilidade de enviar dados na rede ou não.
2. Controle de fluxo baseado em canais virtuais: Neste tipo de controle de fluxo, os buffers de entrada associados aos canais físicos dos roteadores são chaveados formando filas de profundidade menor alocadas independentemente umas das outras. Estas filas são chamadas canais virtuais e servem para resolver problemas de colisão de recursos no caso do chaveamento por pacote *worm hole*.
3. Controle de fluxo baseado em créditos: Este tipo de controle de fluxo é feito da seguinte forma: 1) o receptor envia informação de créditos relativos ao espaço para recepção de dados disponível no buffer; 2) o emissor envia toda a informação possível, dentro dos limites impostos pela informação de créditos obtida; 3) a informação de crédito é decrementada com a recepção da mensagem.

1.2.5 Arbitragem

A arbitragem é o mecanismo que define qual porta de entrada poderá utilizar uma determinada porta de saída. Se por um lado o roteamento é o mecanismo responsável pela seleção da saída, a arbitragem é o mecanismo responsável pela seleção da entrada.

O mecanismo de arbitragem pode ser caracterizado como centralizado, quando o roteamento e a arbitragem são feitos num mesmo módulo, ou distribuído, quando o roteamento e a arbitragem são realizados de forma independente um do outro. Quanto aos critérios de arbitragem podem ser mencionados:

- O uso de prioridades estáticas: as prioridades estáticas são atribuídas de forma fixa e, uma vez definidas, não são alteradas pelo mecanismo de arbitragem.
- O uso de prioridades dinâmicas: as prioridades dinâmicas não recebem um valor fixo, seu valor pode ser alterado pelo mecanismo de arbitragem.
- Política de FCFS (*First-Come-First-Served*): os canais de entrada são selecionados por ordem de chegada. O primeiro canal a fazer a requisição é o primeiro a ser atendido.

- Política de LRS (*Least-Recently-Served*): o canal de entrada que menos vezes tenha sido selecionado recentemente é o canal a ser selecionado.
- Método *Round-Robin*: Este método baseia-se no uso de fatias de tempo (*quantum*) para seleção dos canais de entrada, de forma que cada canal de entrada é selecionado por uma determinada fatia de tempo. Caso o envio da mensagem não seja concluído ao término da fatia de tempo, a próxima porta de entrada é selecionada. A porta bloqueada (com a mensagem armazenada na memória) espera até ser selecionada novamente pelo mecanismo de arbitragem. Isso é repetido até que toda a mensagem de cada canal tenha sido transmitida.

1.2.6 Memorização

A parte de memorização é a responsável pela definição de como e onde são armazenadas as mensagens bloqueadas em um roteador. No caso de uma colisão de recursos, os pacotes devem ser guardados em algum elemento de memória até que os recursos requisitados estejam disponíveis e a mensagem possa continuar trafegando na rede até o seu destino. No caso de não haver elementos de memória, no caso de uma colisão de recursos, a mensagem não seria armazenada (seria perdida).

A memorização pode ser feita de forma centralizada e compartilhada, na entrada ou na saída. No caso da memorização centralizada e compartilhada, um buffer central é utilizado para armazenar os pacotes bloqueados de todas as portas de entrada do roteador. Já no caso de memorização na entrada, são utilizados buffers independentes entre si nas portas de entrada do roteador. Por fim, no caso de memorização na saída, os espaços de memorização são particionados entre as portas de saída.

Existem várias possibilidades de implementar tais elementos de memória, dentre estas possibilidades destacam-se os buffers FIFO, onde as informações são armazenadas em posições de memória e lidas por ordem de chegada.

1.3 Relação das Redes-em-Chip com as Camadas OSI-ISO

O modelo OSI consiste em uma descrição de sistemas de comunicação baseada em sete camadas hierárquicas, são elas: camada física, camada de enlace de dados, camada de rede, camada de transporte, camada de seção, camada de apresentação e camada de aplicação. Levando em consideração o contexto das redes-em-chip as camadas do modelo OSI se relacionam da seguinte forma (OST, 2004):

- Camada física: Define os parâmetros elétricos dos sinais, direção dos sinais e largura dos canais.
- Camada de enlace: Diminui a falta de confiabilidade na transferência de dados sobre o meio físico. Define o protocolo de comunicação, como, por exemplo, *handshake*.
- Camada de rede: Define os algoritmos de chaveamento e roteamento. Determina a conexão entre origem e destino.
- Camada de transporte: Estabelece o controle de fluxo, define o algoritmo de empacotamento e desempacotamento das mensagens. Garante a recepção ordenada dos pacotes.

- Camada de seção, apresentação e aplicação: Estas três camadas se misturam no gerenciamento e sincronização das mensagens e conversão do formato da mensagem pelo receptor.

As três primeiras camadas dizem respeito ao roteador da rede, enquanto as últimas camadas estão relacionadas aos núcleos da rede (*cores*).

1.4 Interface Padrão Para Interconexão de Núcleos

A fim de garantir a reusabilidade e escalabilidade das redes-em-chip, faz-se necessária a utilização de uma interface padrão da rede com os núcleos interligados através dela. Algumas das interfaces padrão propostas são: VCI (V*irtual Component Interface*), OCP (*Open Core Protocol*) e VSIA (*Virtual Socket Iniciative Alliance*). Os dois primeiros são descritos brevemente a seguir.

- Padrão VCI: É formado por dois canais unidirecionais, um para o iniciador, que envia as requisições, e um para o alvo, que envia respostas. As informações são transferidas sob a forma de pacotes de requisição e resposta. Um protocolo de *handshake* é utilizado para garantir o controle de fluxo de dados nos canais. O padrão VCI possui diferentes níveis de complexidade visando aplicações com diferentes requisitos, são eles: VCI básico, VCI periférico e VCI avançado (BIRNBAUM apud OST, 2004).
- Padrão OCP: A comunicação consiste, basicamente, em comandos de leitura e escrita. A comunicação com o padrão OCP é feita entre um núcleo atuando mestre e um núcleo atuando como escravo. O padrão define ainda sinais (todos unidirecionais) classificados em três grupos: sinais de fluxo de dados, sinais opcionais de controle e sinais de teste (utilizados para testar o núcleo) (OST, 2004).

1.5 Estudo de Caso: SoCIN

A rede SoCIN (ZEFFERINO, 2003) é um exemplo de NoC acadêmica apresentada na literatura. Suas características são mostradas na tabela 1.

Tabela 1. 1: Tabela com as características da rede SoCIN.

Topologia	Rede direta: grelha 2D
Roteamento	Determinístico (roteamento XY); Baseado na fonte
Chaveamento	Chaveamento por pacote *Worm hole*
Controle de Fluxo	*Handshake*
Arbitragem	Árbitro *Round-Robin* com codificador de prioridade programável.
Memorização	Memorização na entrada (*buffers* FIFO)
Interface Padrão	VCI

Esta rede é parametrizável, sendo possível selecionar o tamanho do canal de dados. O roteador utilizado na SoCIN é chamado RASoC (*Router Architecture for System-on-Chip*) e é constituído por até 5 portas de comunicação (conforme mostrado na figura 1.3), sendo elas: porta leste, porta oeste, porta norte, porta sul e porta local (ligada ao núcleo). Cada porta de comunicação possui dois canais unidirecionais: canal de entrada e canal de saída.

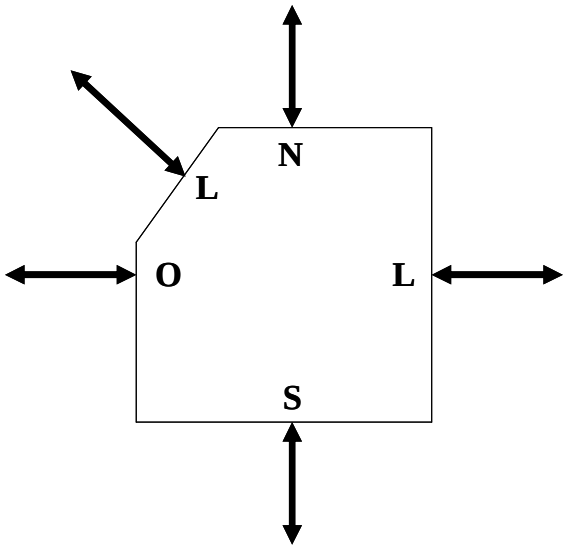


Figura 1. 3: Representação do roteador RASoC com 5 portas de comunicação (ZEFERINO, 2003).

A rede SoCIN será utilizada como estudo de caso dos métodos apresentados no Capítulo 3.

RESUMO DO CAPÍTULO 1

O Capítulo 1 apresentou uma descrição das Redes-em-Chip (NoCs). As NoCs são apresentadas como uma solução para a comunicação intra-chip. As principais vantagens apresentadas pela comunicação baseada no chaveamento de pacotes, no qual as redes-em-chip se baseiam, são devidas à escalabilidade, à reusabilidade e ao paralelismo. Já as soluções de comunicação baseadas em barramentos apresentam limitações de desempenho com o aumento do número de módulos IP sendo conectados a um barramento.

Neste capítulo, foram apresentadas algumas propriedades das NoCs, tais como:

- Topologia.
- Roteamento.
- Chaveamento.
- Controle de fluxo.
- Arbitragem.
- Memorização.

A relação das NoCs com as camadas OSI-ISO também foi apresentada.

A NoC utilizada como estudo de caso no Capítulo 3 é caracterizada neste capítulo levando em consideração as propriedades descritas anteriormente. Para outros tipos de NoCs, algumas características podem ser diferentes.

2 TESTE DE SOCS BASEADOS EM NOCS

A etapa de teste dos circuitos integrados é responsável por averiguar se o circuito implementado possui algum defeito decorrente do processo de fabricação. Em grandes sistemas-em-chip, esta etapa torna-se bastante dispendiosa devido à baixa observabilidade e controlabilidade dos sinais internos a estes sistemas (ZORIAN, 1999). Dessa forma, a metodologia de teste deve ser pensada já na etapa de projeto do circuito, fazendo uso de estruturas específicas que visam o teste do mesmo.

O custo da etapa de teste dos circuitos integrados pode compor uma grande parcela do custo total do chip. Assim, a motivação no estudo de métodos eficientes de teste está ligada à redução do custo total do chip.

Este capítulo apresenta conceitos básicos na área de teste de circuitos integrados, além de métodos de teste de redes-em-chip propostos recentemente na literatura. Não é abordado com detalhes o teste dos núcleos do sistema, atendo-se apenas ao teste das

2.1 Conceitos de Teste

2.1.1 Verificação Funcional x Teste

Inicialmente, recordemos alguns conceitos básicos da área de teste de circuitos integrados.

Teste: A etapa de teste visa averiguar o chip resultante do processo de fabricação, indicando se o circuito fabricado possui algum defeito decorrente desse processo (AXEL, 2003).

Verificação: A etapa de verificação busca averiguar o correto funcionamento do ponto de vista do projeto do circuito, indicando se o circuito projetado encontra-se de acordo com a especificação do projeto (AXEL, 2003).

Tomando como base o conceito de teste descrito acima, nota-se que o teste tem impacto direto na qualidade do circuito vendido (circuito final), impactando diretamente, também, no custo final do chip. Quantitativamente, definindo-se o rendimento (*yield*) Y e a qualidade do teste T, é possível calcular a fração de *chips* ruins que passam no teste e são repassados para os clientes. Esta fração é chamada de *defect level* e é dada por:

$$DL = 1 - Y^{1-T}$$ (1) (Willians apud AXEL, 2003).

Para minimizar a fração de chips defeituosos que passam no teste, deve-se maximizar o termo Y e maximizar o termo T. O termo Y depende do processo de fabricação e das ferramentas utilizadas ao longo do fluxo de concepção do circuito

integrado. Já o termo T depende da qualidade do teste proposto e dos vetores de teste empregados.

Os chips defeituosos que passam na etapa de teste podem ser utilizados em suas respectivas aplicações, resultando em produtos defeituosos. O custo para a reparação desses produtos tende a aumentar consideravelmente dependendo de quão tarde as falhas são detectadas.

O teste de circuitos integrados é feito através de equipamentos específicos para tal fim, chamados ATEs (*Automatic Test Equipaments* – Equipamento Automático de Teste). O custo desses equipamentos é, em geral, muito elevado, sendo que o tempo de teste (tempo que o chip passa no testador) torna-se um fator determinante no custo do teste. Dessa forma, estratégias de teste que visam reduzir o tempo gasto no testador são fundamentais para a diminuição do custo desta etapa.

2.1.2 Falhas e Defeitos

A fim de quantificar a qualidade do teste, deve-se avaliar a cobertura de defeitos do teste. Contudo, os defeitos são os erros físicos do processo de fabricação sendo muito difíceis de ser identificados corretamente (podem ser curtos-circuitos, circuitos abertos, ou até mesmo a ausência de transistores) (AXEL, 2003). Dessa forma, os testes trabalham com a manifestação desses defeitos, que recebem a denominação de falhas. A quantificação da qualidade do teste pode ser feita, portanto, pela cobertura de falhas do teste. Esta cobertura de falhas é dada pelo número de falhas detectadas pelo teste dividido pelo universo de falhas consideradas no teste.

O universo de falhas consideradas em um determinado método de teste constitui o modelo de falhas desse método. Alguns modelos de falhas são explicados a seguir:

- Colagem (*stuck-at*):

  Um exemplo de modelo de falhas comum é o modelo de colagem, que representa falhas onde o nível lógico de um sinal mantém o valor fixo em 1 ou 0. A falha pode ser representada por "s-a-*v*", onde *v* representa o nível lógico fixo: colagem em 0 (*stuck-at-0*) ou colagem em 1 (*stuck-at-1*). Esta falha modela o comportamento faltoso decorrente de um defeito, por exemplo,do tipo curto-circuito com a linha de alimentação (ABRAMOVICI, 1990).

- Curto-circuito (*bridging fault*):

  É a denominação do modelo de falhas decorrente de um defeito do tipo curto-circuito entre dois fios de forma que uma nova função lógica é introduzida nestes sinais. A falha recebe a denominação de curto-circuito tipo *AND*, ou *wired-AND*, no caso de introduzir uma função AND nos dois fios afetados e a denominação curto-circuito tipo *OR*, ou *wired-OR*, no caso de introduzir uma função OR nos dois fios afetados (ABRAMOVICI, 1990).

Os modelos de falhas descritos acima representam apenas falhas permanentes, sendo que estes modelos são utilizados nos métodos de teste apresentados no capítulo 3. Modelos de falhas transientes e intermitentes não serão detalhados neste trabalho.

Por fim, os erros representam uma operação incorreta do sistema e podem estar relacionados a uma falha (e, em última instância, a um defeito).

2.1.3 DfT – *Design-for-Testability*

O conceito de *Design-for-Testability* representa os esforços feitos durante a etapa de projeto do circuito visando exclusivamente tornar o teste deste dispositivo menos dispendioso, ou seja, visando aumentar a testabilidade do circuito (ABRAMOVICI, 1990).

Alguns exemplos de técnicas DfT são:

- Inserção de cadeias *scan*:

  Um exemplo de técnica de DfT é a inserção de cadeias *scan* no circuito, uma vez que isto é feito na etapa de projeto do *chip* e que busca aumentar a controlabilidade e observabilidade dos sinais internos, aumentando, assim, a testabilidade do circuito. Nesta técnica de DfT, os *flip-flops* que fazem parte da cadeia *scan* são substituídos por *scan-flip-flops* e são conectados em cadeia formando uma espécie de registrador de deslocamento. Estes *scan-flip-flops* apresentam um multiplexador que seleciona os dados a serem armazenados: dados funcionais ou vetores de teste.

  Assim, através da cadeia *scan*, os vetores de teste são inseridos serialmente no circuito sob teste pelo ATE. Uma vez que estes vetores se encontram armazenados nos *scan-flip-flops*, eles são aplicados à lógica do circuito sob teste e capturados pelos *flip-flops* pertencentes à cadeia *scan* no ciclo de relógio seguinte. Os valores capturados pela cadeia *scan* podem, então, ser extraídos pelo ATE.

- BIST (*Built-In Self-Test*)

  Técnicas de auto-teste (BIST) também fazem parte da estratégia de DfT, uma vez que são inseridos na etapa de projeto do circuito e visam facilitar o teste.

  O conceito de *Built-In Self-Test* está relacionado à capacidade do circuito de auto testar-se. O auto-teste pode ser feito de duas formas: *on-line* e *off-line*. O auto teste *on-line* ocorre quando o teste é feito dentro do modo funcional do circuito e pode ser concorrente ou não-concorrente. O teste *on-line* concorrente ocorre simultaneamente com a operação normal do circuito (modo funcional), enquanto o teste *on-line* não-concorrente ocorre quando o circuito, apesar de estar em modo funcional, encontra-se em um estado de espera. Já o teste *off-line* ocorre com o circuito operando em modo de teste, não em modo funcional.

  Este tipo de teste geralmente envolve um circuito gerador de vetores de teste, circuitos analisadores de resposta e rotinas de diagnóstico (ABRAMOVICI, 1990). Os circuitos utilizados para realizar o auto-teste compõe a estrutura de BIST (circuitos de BIST, ou *hardcore*).

- Padrões de Teste:

  Alguns padrões de teste foram desenvolvidos visando uma maior testabilidade do circuito. Neste trabalho são citados dois padrões de teste: IEEE std. 1149.1 (IEEE, 2001) e IEEE std. 1500 (IEEE, 2005). Estes padrões de teste são apresentados brevemente a seguir.

2.1.4 IEEE std. 1149.1 – *Boundary Scan*

O padrão IEEE std. 1149.1, ou simplesmente *Boundary Scan*, foi proposto inicialmente com o objetivo de facilitar o teste das interconexões de placas de circuito

impresso. Neste trabalho são apresentados apenas os componentes desse padrão de teste, que é constituído das seguintes partes:

- Sinais de entrada e saída:
    - TDI (*Test Data Input* – Entrada de Dados de Teste) – Sinal de entrada de dados.
    - TMS (*Test Mode Signal* – Sinal de Modo de Teste) – Sinal de modo de teste.
    - TCK (*Test Clock* – Relógio de Teste) – Sinal de clock de teste.
    - TDO (*Test Data Output* – Saída de Dados de Teste) – Sinal de saída de dados.
- Células *boundary scan*: São células especiais posicionadas na periferia do *chip* com o objetivo de aumentar a controlabilidade e observabilidade dos sinais que entram e saem do mesmo. Sua estrutura é apresentada na figura 2.1.

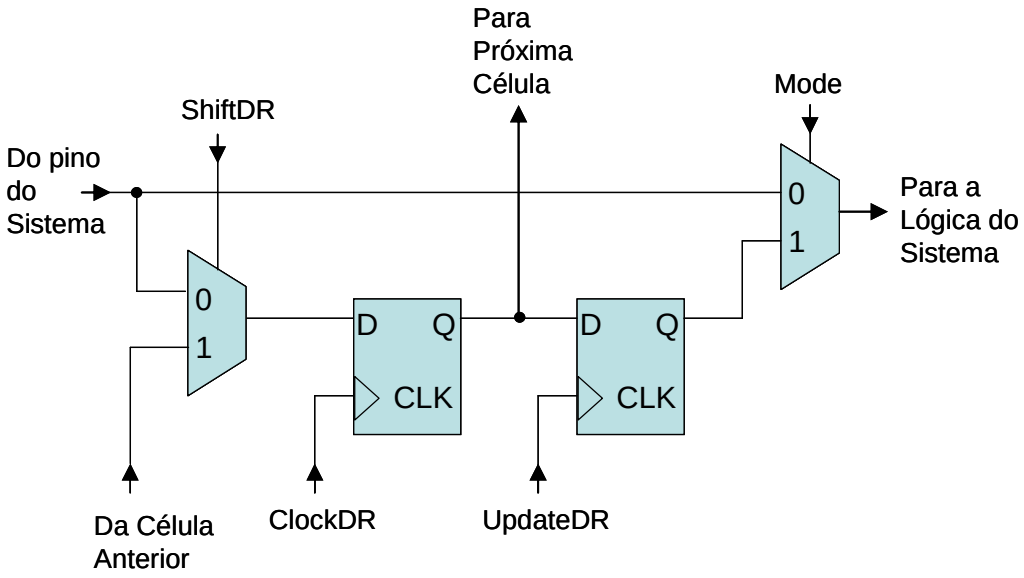


Figura 2 1: Célula Boundary-Scan (IEEE, 2001).

As células *boundary scan* permitem o deslocamento dos vetores de teste através da cadeia a qual estão ligados, a inserção de vetores de teste na lógica/pino do sistema e a captura de valores oriundos da lógica/pino do sistema.

- Controlador TAP: É a máquina de estados que gera os sinais de controle das células *boundary scan*. É controlada pelos sinais TCK (relógio de teste) e TMS (sinal de seleção de modo de teste). A máquina de estados é apresentada na figura 2.2.

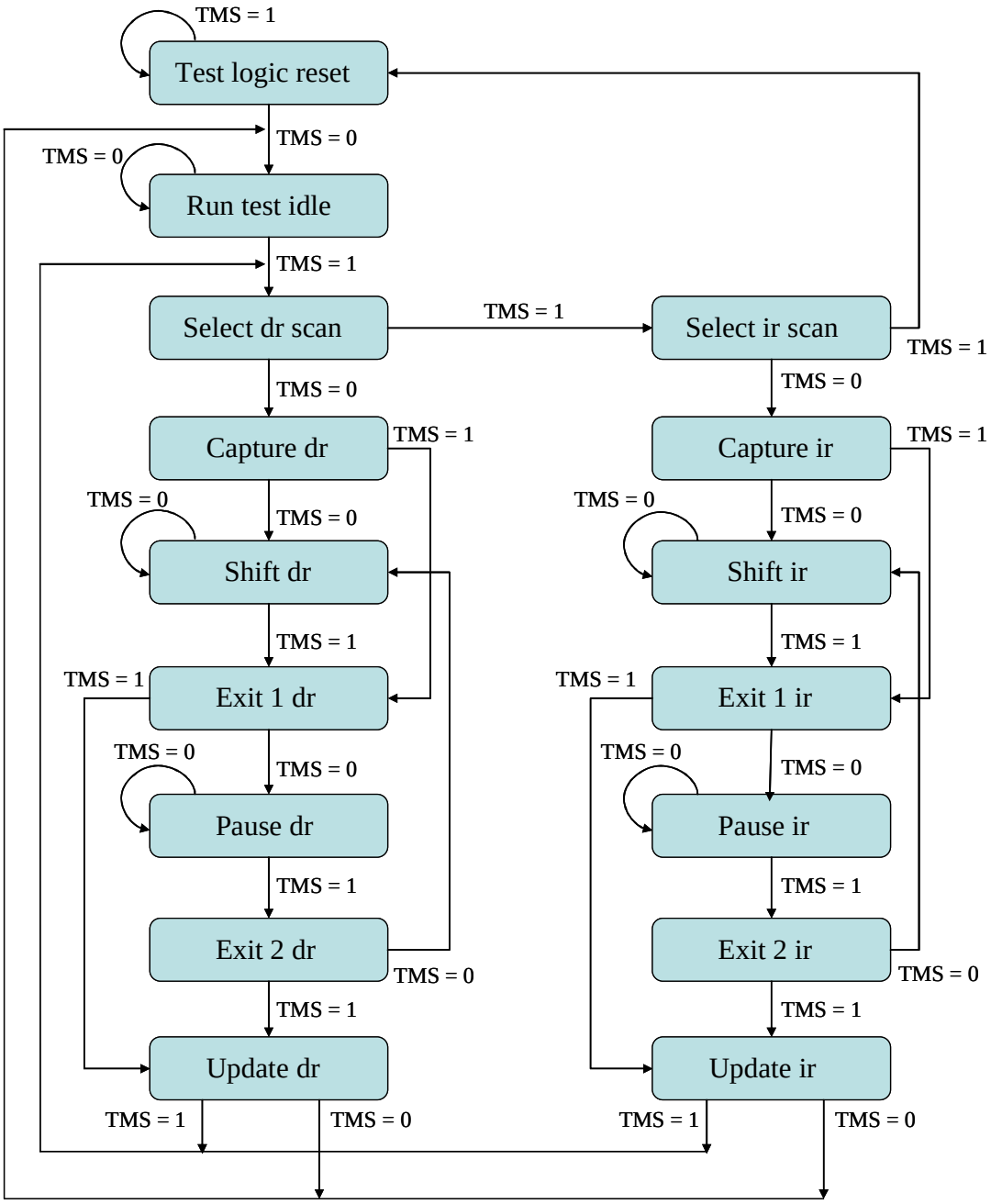


Figura 2 2: Máquina de estados do controlador TAP do padrão *boundary Scan* (IEEE, 2001).

- Registrador de instrução: É o registrador que recebe o código relativo às instruções de teste (*op code*). Pode conter bits extras utilizados para armazenar dados referentes a testes específicos definidos pelo projetista.
- Registrador de *Bypass*: É o registrador utilizado no caso do circuito não fazer parte do teste.
- Demais Registradores (opcionais): São registradores adicionais que podem conter informações como a identificação do componente (registrador de identificação) ou outros dados.

O padrão *boundary scan* possui quatro instruções obrigatórias, são elas: *Bypass* (utiliza o registrador de *bypass* e pode ser utilizada quando o circuito não faz parte do teste), *Sample* ou amostra (amostra os valores oriundos da entrada dos registradores de *boundary scan*), *Preload* ou carga (carrega os valores nos registradores *boundary scan*) e *Extest* (insere os vetores de teste pré-carregados nos registradores de *boundary scan* na saída desses registradores).

2.1.5 IEEE std. 1500 – Teste de SoCs

O padrão IEEE std. 1500 foi desenvolvido com influência do padrão *Boundary Scan*, de forma que os dois padrões possuem objetivos semelhantes, porém para diferentes níveis de integração. Enquanto o *Boundary Scan* propõe uma envoltória de teste e mecanismos de acesso ao teste visando o teste no nível da placa de circuito impresso, o padrão IEEE std. 1500 propõe uma estrutura similar que visa estabelecer uma estratégia de teste estruturada para circuitos integrados contendo múltiplos núcleos (SoCs, por exemplo). O padrão propõe uma arquitetura escalonável e modular cujo objetivo é facilitar o reuso de módulos dentro de um sistema-em-chip, mantendo a testabilidade do mesmo (IEEE, 2005).

Este padrão faz uso de uma envoltória de teste utilizada de forma a promover uma interface padrão de acesso ao teste para os diferentes módulos que compõem o sistema. Esse circuito envoltório pode ser visto na figura 2.3. O padrão também estabelece uma linguagem de teste de núcleos (CTL – *Core Test Language*). Neste trabalho são apresentados apenas os componentes desse padrão de teste.

Similarmente ao *boundary scan*, este padrão de teste propõe uma envoltória de teste (figura 2.3) para padronização dos sinais. Esta envoltória é composta pelas seguintes partes:

- WSP (*Wrapper Serial Port*) – Sinal serial utilizado para carregar dados e instruções para dentro e para fora da envoltória. É constituído de uma porta serial de entrada (WSI – *Wrapper Serial Input*) e uma porta serial de saída (WSO – *Wrapper Serial Output*), além do controlador serial (WSC – *Wrapper Serial Control*) responsável por controlar a operação de todos os registradores da envoltória.
- WPP (*Wrapper Parallel Port*) – Porta paralela da envoltória de teste, constituída por uma porta paralela de saída e uma porta paralela de entrada com interfaces definidas pelo projetista.
- WIR (*Wrapper Instruction Register*) – Registrador de instruções da envoltória.
- WBY (*Wrapper Bypass Register*) – Registrador de *Bypass*.
- WBR (*Wrapper Boundary Register*) – É o registrador pelo qual os estímulos de teste são aplicados ao circuito e os resultados são capturados. Faz um papel similar ao registrador de *boundary scan* do padrão IEEE std. 1149.1.

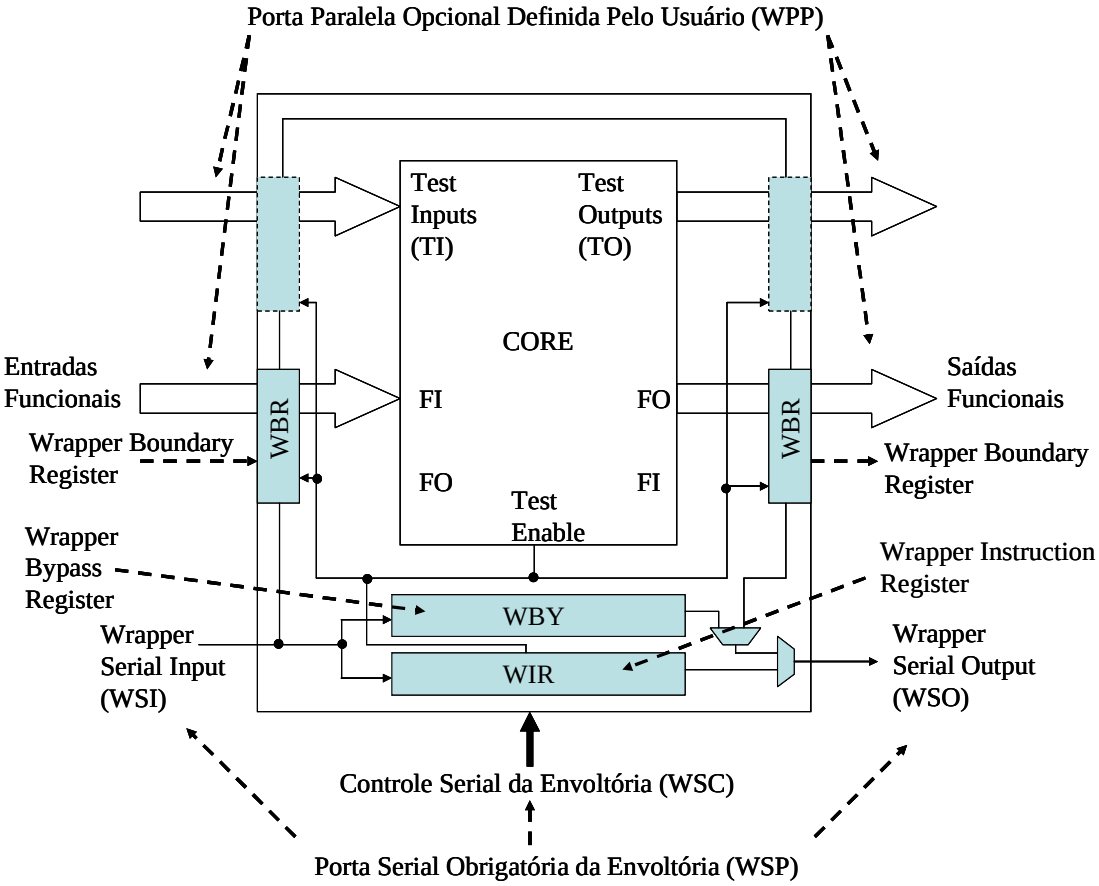


Figura 2 3: Envoltória de teste do padrão IEEE std 1500 (IEEE, 2005).

2.1.6 Mecanismos de Acesso ao Teste (TAM – T*est Access Mechanism*)

Os mecanismos de acesso ao teste (TAMs) são estruturas responsáveis por fornecerem os meios para o transporte dos vetores de teste ao circuito sob teste. Da mesma forma, estas estruturas estabelecem os meios para o transporte dos resultados do teste de volta ao equipamento testador externo (ATE). Exemplos comuns de mecanismos de acesso ao teste são pinos de entrada e saída e conexões dedicadas apenas ao transporte dos vetores de teste.

Com o objetivo de reduzir ou eliminar o custo em área causado por esses mecanismos (no caso de mecanismos dedicados), foram propostos mecanismos de acesso ao teste que fazem uso de estruturas já presentes no circuito e que são utilizadas durante sua operação normal. No caso de sistemas-em-chip que fazem uso de NoC para comunicação entre os núcleos do sistema, a infra-estrutura da NoC pode ser utilizada como mecanismo de acesso ao teste. Dessa forma, os vetores de teste são enviados para os circuitos sob teste através de interconexões já existentes.

Para poder usar a própria NoC para fazer o transporte dos vetores de teste, contudo, tais vetores devem ser enviados na forma de pacotes, respeitando o protocolo de comunicação da rede. Além disso, a envoltória que conecta os núcleos à NoC deve ser adaptada visando a conexão com a interface de teste (como controles de *scan* ou pinos de *scan*) (COTA, 2004).

2.2 Teste de Redes-em-Chip

Por se tratar do mecanismo de comunicação entre os diferentes módulos que compõe o sistema-em-chip, a NoC ocupa uma posição singular dentro do sistema. Do ponto de vista de teste, o desafio é aumentar a testabilidade da rede e do sistema como um todo estabelecendo uma estratégia de teste eficiente que faça uso das características intrínsecas da rede (paralelismo, regularidade, posição dentro do sistema). A seguir, são apresentadas algumas estratégias de DfT, além de métodos de teste aplicáveis a NoCs, levando em consideração as duas partes que compõem a rede-em-chip: os roteadores e as interconexões.

2.2.1 Teste dos Roteadores

Dentro do contexto de sistemas-em-chip (SoCs) as NoCs têm sido abordadas, por alguns autores, como sendo mais um dos núcleos do sistema. Dessa forma, o teste da NoC pode ser feito da mesma forma como são testados os demais núcleos pré-projetados dentro de um SoC, com a singularidade de ser formada geralmente por múltiplos sub-núcleos idênticos (como os roteadores, por exemplo) e por geralmente estar localizada em uma região central do chip devido à necessidade de conexão com os demais “módulos IPs” do sistema (VERMEULEN, 2003).

O teste baseado em núcleos utilizado tradicionalmente em SoCs baseia-se no uso do padrão de teste IEEE std 1500 (IEEE, 2005). Este padrão faz uso de uma envoltória de teste conforme mencionado anteriormente. O IEEE std. 1500 foi inspirado no padrão de teste IEEE 1149.1 (*Boundary scan*), de forma que esses dois padrões possuem objetivos semelhantes, porém para diferentes níveis de integração.

Amory (2005) propõe uma estratégia de teste escalonável e eficiente, do ponto de vista de custo de teste, visando o teste dos roteadores da rede. Esta estratégia faz uso de técnicas de DfT (*Design for Testability*) através de uma envoltória de teste compatível com a proposta no padrão IEEE std. 1500, mecanismos dedicados de acesso ao teste e cadeias de *scan*.

Esse trabalho (AMORY, 2005) parte de uma observação acerca do subrecusto de área e volume de teste decorrente de diferentes estratégias de inserção de cadeias scan para o teste dos roteadores da NoC. Foram implementadas, inicialmente, cadeias *scan* completas (*full scan*) na NoC de duas formas: a NoC sendo vista como um núcleo hierárquico e a NoC sendo vista como um núcleo sem hierarquia. Na forma hierárquica, cada roteador da NoC foi visto como um circuito individual, cada qual com sua respectiva envoltória de teste. Da forma sem hierarquia, a NoC foi encarada como um circuito único, com apenas uma envoltória de teste compreendendo toda a rede.

Os resultados mostraram que a configuração que apresentou menor sobrecusto de área foi a da NoC considerada como um núcleo sem hierarquia, devido ao fato de necessitar de uma quantidade menor de envoltórias de teste. Entretanto foi observado, também, que o volume de teste da NoC vista como um núcleo sem hierarquia é muito maior do que no caso da rede sendo vista como um núcleo hierárquico. Isto ocorre porque quando os roteadores são tratados de forma hierárquica, cada roteador é testado individualmente (com cadeias *scan* separadas), necessitando um conjunto menor de vetores de teste. No caso da rede ser vista como um circuito único, os vetores necessários para o teste de todos os roteadores devem ser inseridos através de uma única cadeia *scan*. Em suma, o circuito sem hierarquia apresenta uma cadeia *scan* maior, o

que implica em mais vetores de teste, apesar de apresentar uma quantidade menor de envoltórias de teste, resultando em um sobrecusto de área menor.

A partir dessas conclusões, foi proposta uma estratégia de teste que considera a NoC um núcleo sem hierarquia, utilizando apenas uma envoltória de teste para toda a rede, além da implementação de uma cadeia *scan* parcial, ao invés de uma cadeia *scan* completa (*full scan*). Com isto, é mantido o menor sobrecusto de área, bem como um volume menor de vetores de teste.

As cadeias *scan* foram definidas dentro de cada roteador da rede compreendendo elementos do *buffer* FIFO (primeira cadeia) e dos *flip-flops* da lógica de controle (segunda cadeia). Em cada *buffer* FIFO, foram compreendidos pela cadeia *scan* apenas a primeira posição da memória, o que permite agregar controlabilidade e observabilidade para o restante do *buffer*.

Explorando a regularidade da NoC (considerando todos os roteadores iguais), pode-se utilizar os mesmos vetores de teste para todos os roteadores da rede. O mecanismo de acesso ao teste proposto aplica simultaneamente os vetores de teste aos roteadores, sendo que as saídas desses são comparadas entre si para detecção de erros. Este esquema é mostrado na figura 2.4.

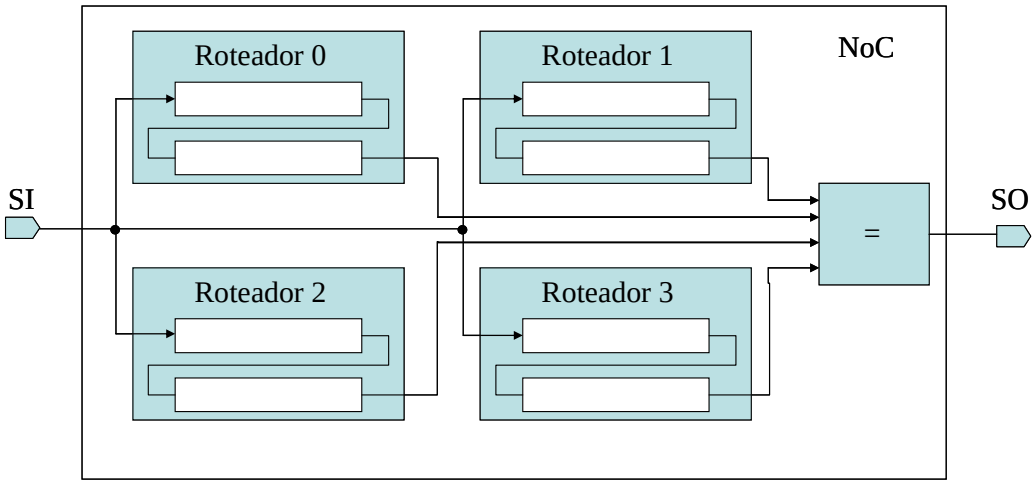


Figura 2 4: Metodologia de acesso ao teste proposto (AMORY, 2005).

Para completar a estratégia de teste, a envoltória de teste compatível com o padrão IEEE std. 1500 foi apresentada. A envoltória de teste permite uma padronização da interface de teste e integra os blocos comparadores propostos para detecção das falhas. A figura 2.5 apresenta tal envoltória. A cobertura de falhas do método proposto foi acima de 98%.

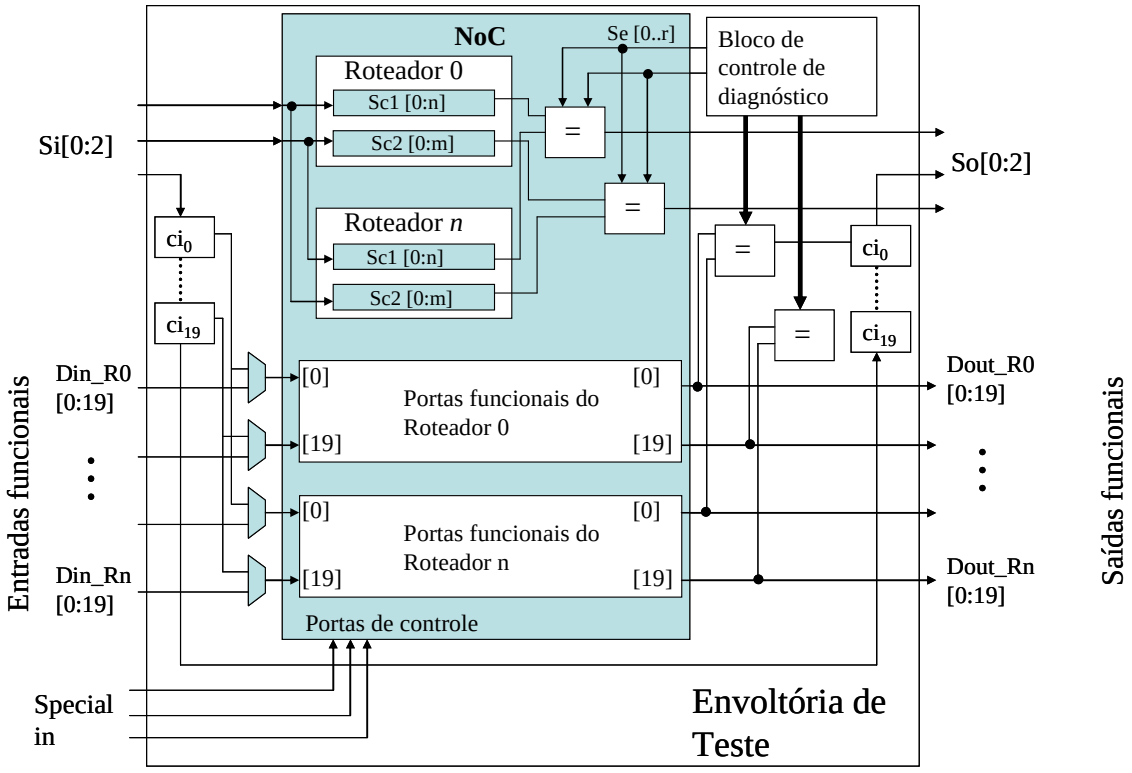


Figura 2 5: Envoltória de teste proposta (AMORY, 2005).

Aktouf (2002) propõe o uso de *boundary scan* para teste dos roteadores da rede. Nesse trabalho, envoltórias de teste *boundary scan* são utilizadas em grupos de células básicas, conforme apresentado na figura 2.6. Assim, para cada conjunto de células, é assumido um controlador TAP, um conjunto de células *boundary scan*, além dos registradores adicionais que fazem parte da envoltória de teste. Assumindo os roteadores como células básicas, é utilizado uma envoltória de teste para cada roteador da rede. Tal estratégia foi implementada por Amory (2005) para comparação com os métodos propostos. O subrecusto de área com o uso da envoltória *boundary scan* se mostrou muito grande em relação as outras metodologias apresentadas (inserção de cadeias *scan*).

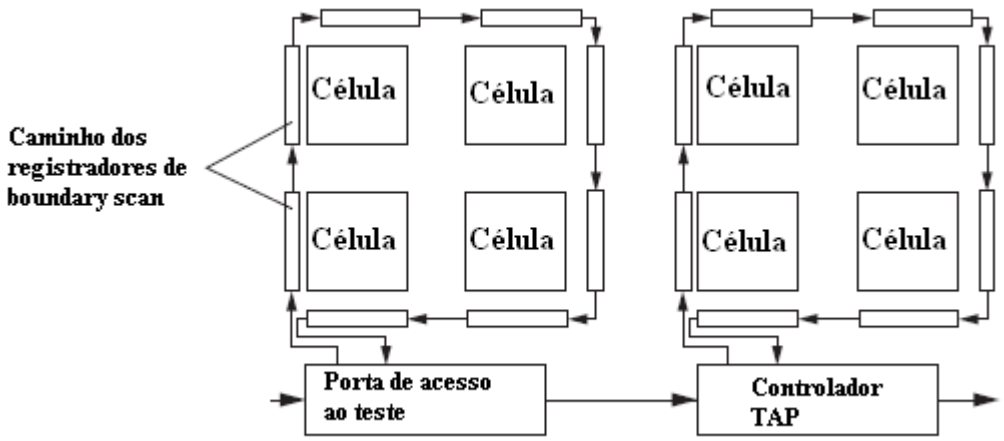


Figura 2 6: Uso de células *Boundary scan* para teste dos roteadores de uma NoC (AKTOUF, 2002).

Aktouf (2002) ainda sugere o uso de comparadores para detecção de falhas dentro da envoltória de teste. A figura 2.7 mostra esta estratégia, o decodificador utilizado tem como objetivo selecionar as saídas a serem comparadas.

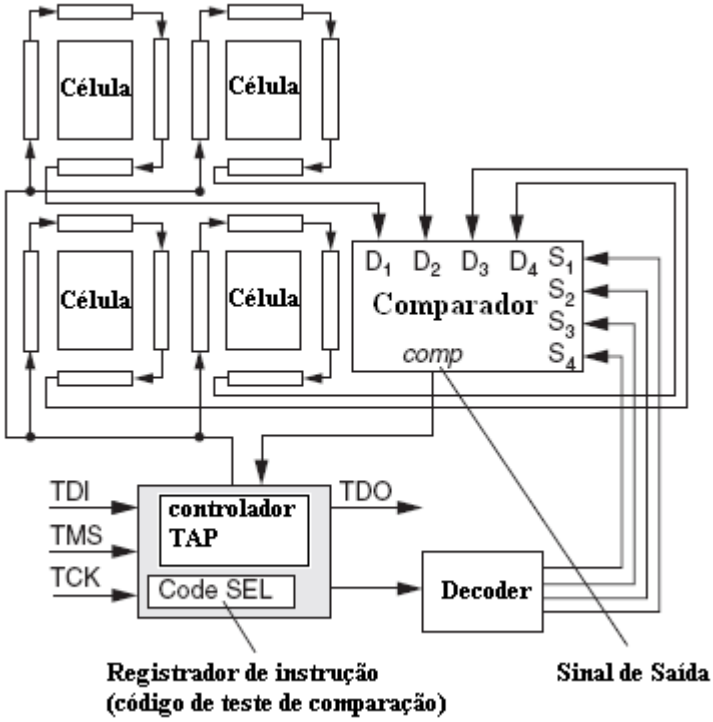


Figura 2 7: Estrutura de teste baseada em *boundary scan* para teste dos roteadores da rede com uso de comparadores internos para análise de resultados (AKTOUF, 2002).

O teste da NoC pode utilizar, também, estratégias de BIST (*Built-In Self-Test*), especialmente para blocos regulares como memórias (VERMEULEN, 2003).

O teste das memórias (*buffers* FIFO) constitui um caso particular dentro do teste dos roteadores. Por se tratar do teste de memórias, o modelo de falhas utilizado difere do modelo de falhas utilizado para o teste de lógica aleatória. Este modelo de falhas pode levar em consideração além de falhas do tipo *stuck-at*, outros tipos de falhas como, por exemplo, falhas de acoplamento. Em falhas do tipo de acoplamento, ao alterar o valor de uma posição da memória o conteúdo de posições vizinhas podem ser alterados de forma involuntária (AKTOUF, 2002).

Estes modelos de falhas considerados no teste de memórias implicam no uso de testes específicos. Os testes tradicionalmente utilizados em memórias baseiam-se no uso de seqüências de leitura e escrita nas posições de memórias. É o que fazem os algoritmos *Marching* (AKTOUF, 2002).

Os *buffers* FIFO que fazem parte dos roteadores da NoC, caracterizam-se por serem vários elementos de memórias de tamanho pequeno distribuídos nos canais de comunicação dos roteadores. Ao mesmo tempo em que o tamanho pequeno dos *buffers* facilita o teste individual de cada um, sua distribuição e quantidade dentro da NoC causam um sobrecusto de área principalmente quando são inseridos circuitos de BIST em cada elemento de memória.

Em Grecu (2005) é proposta uma estratégia de BIST distribuído, onde os mecanismos de controle são compartilhados, conforme mostra a figura 2.8. Na figura, o

circuito de BIST possui um circuito gerador dos sinais de leitura e escrita na FIFO além de um circuito gerador de sinais a serem escritos nas posições da FIFO. Os sinais gerados são enviados simultaneamente para todos os *buffers* do roteador. Isso evita um maior sobrecusto de área. Os analisadores de resposta, contudo, são inseridos localmente, sendo que as respostas são carregadas em um MISR (*Multiple Input Shift Register*). No artigo é descrito, também, o algoritmo de teste dos *buffers*, baseado em técnicas tradicionais de teste de memórias RAM.

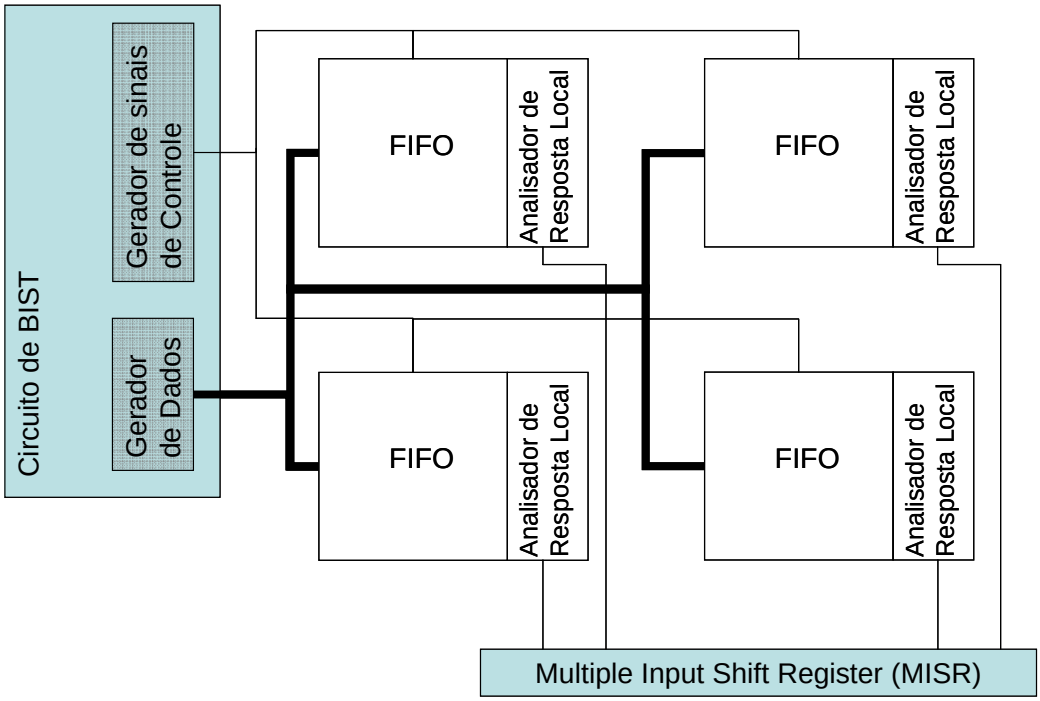


Figura 2 8: Circuitos de BIST compartilhados para teste de memória nos roteadores da NoC (GRECU, 2005).

Aktouf (2002) também debate o uso de técnicas tradicionais de teste de memórias RAM para o teste dos *buffers* dos roteadores. Com o objetivo de reduzir o sobrecusto em área, é defendido, também, o compartilhamento dos circuitos de BIST.

Para transmissão dos vetores de teste e extração dos resultados do teste da NoC foram propostos mecanismos de acesso dedicados (pinos e conexões específicas ao teste) e mecanismos de acesso baseados no reuso das interconexões funcionais. Para o segundo, é necessário um teste prévio das interconexões da NoC que serão usadas como TAMs.

Mecanismos dedicados de acesso ao teste são compostos basicamente por pinos e interconexões que tem como propósito transmitir apenas informações relevantes ao teste do circuito. É possível notar que estas estruturas causam um aumento de área no circuito uma vez que não são utilizadas em operação normal.

Com o objetivo de reduzir o custo em área causado pelos mecanismos de acesso ao teste, foram propostos mecanismos baseados no reuso de interconexões funcionais. Dessa forma, os vetores de teste são enviados para os circuitos sob teste através de interconexões já existentes, utilizadas na operação normal do circuito. Assim, os vetores de teste são enviados através da NoC na forma de pacotes, respeitando as características de comunicação da rede.

Grecu (2005) propõe uma estratégia de teste que faz uso da NoC para testar seus elementos (roteadores) de forma progressiva. O autor divide o teste dos roteadores no teste dos *buffers* e no teste da lógica de controle. O teste da lógica de controle é feito através de cadeias *scan* inseridas em cada roteador. Os dados são inseridos nas cadeias *scan* através das interconexões da NoC. Com isso, não existe a necessidade de ligar a cadeia *scan* externamente através de conexões dedicadas, o que reduz a quantidade de interconexões extras necessárias para o teste e, por conseqüência, o sobrecusto de área. A comparação dos resultados do teste também é feita internamente através de comparadores.

No teste proposto em Grecu (2005), os roteadores possuem dois modos de operação: modo normal e modo de teste. Em modo normal, os roteadores transmitem os dados na rede de forma funcional. Já no modo de teste, os dados que entram no roteador são inseridos em cadeias *scan* para teste dos blocos lógicos de roteamento.

Quanto à estratégia de acesso ao teste, é proposta a utilização progressiva dos recursos da NoC. Isto é feito conectando um roteador diretamente a uma fonte geradora de vetores de teste. Este roteador é testado primeiro, recebendo os vetores de teste diretamente do ATE. O teste é feito da seguinte forma:

- Inicialmente o roteador é colocado em modo de teste, sendo que as informações recebidas pela fonte de vetores de teste são carregadas em sua cadeia *scan*.
- Em seguida, é feita a captura dos resultados do teste e a extração dos resultados.
- O resultado do teste é comparado com os valores esperados que também são enviados pela fonte geradora dos vetores de teste.
- Uma vez testado, este roteador passa de modo teste para modo de operação normal, transportando os dados para o próximo roteador sob teste.

Grecu (2005) considera duas formas de transmissão dos vetores de teste: *unicast* e *multicast*. Na forma *multicast* (figura 2.9), os vetores de teste são enviados simultaneamente a todos os roteadores adjacentes ao último roteador testado. Já na forma *unicast* (figura 2.10) os vetores são enviados apenas para um roteador adjacente ao último roteador testado, sendo necessário um tempo maior de teste.

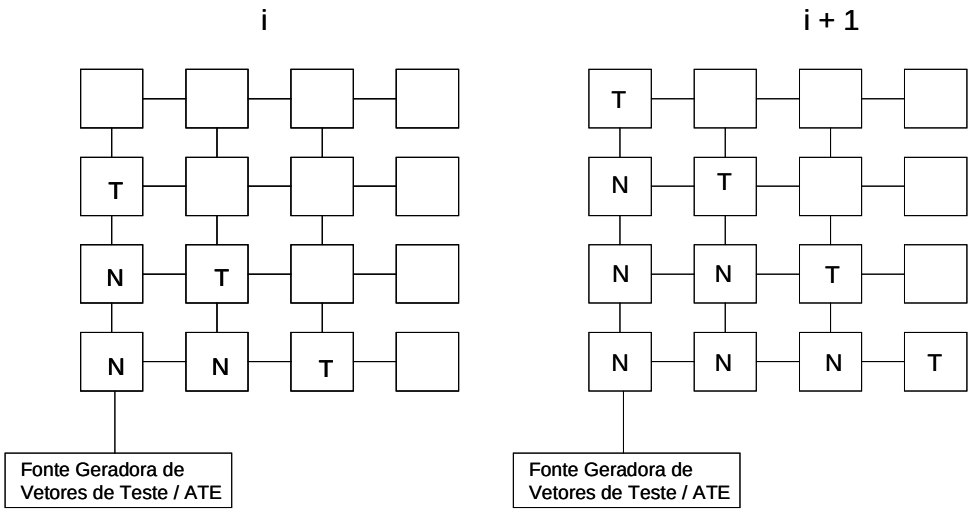


Figura 2 9: Representação do esquema de teste para comunicação *multicast* baseado no reuso da NoC proposto. “N” representa operação normal e “T” representa modo de teste (GRECU, 2005).

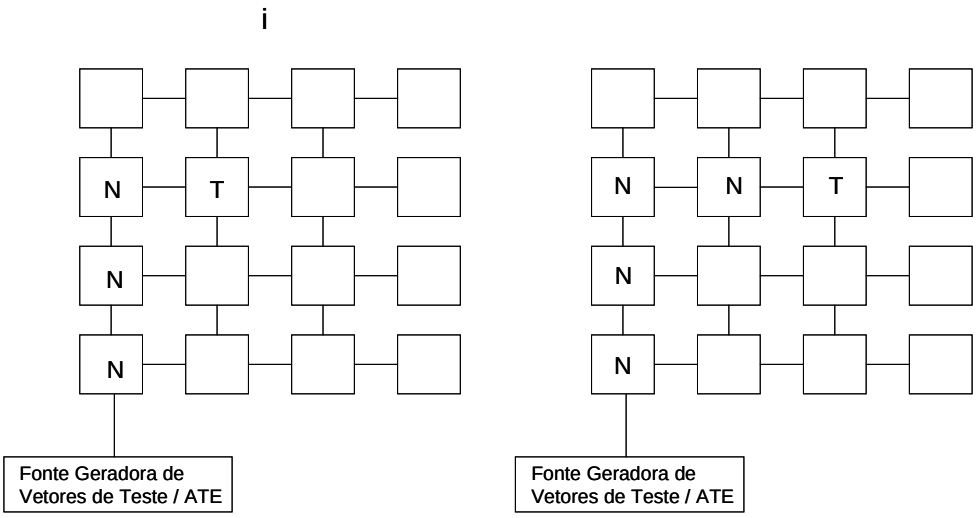


Figura 2 10: Representação do esquema de teste para comunicação *unicast* baseado no reuso da NoC proposto. “N” representa operação normal e “T” representa modo de teste (GRECU, 2005).

Hosseinabady (2006) apresenta uma estratégia de teste dos roteadores muito similar a Grecu (2005) levando em consideração diferentes topologias. Assim como em Grecu (2005), é assumido que um roteador é ligado diretamente ao ATE (gerador de vetores de teste). O roteador ligado diretamente à fonte geradora dos vetores de teste deve ser o roteador que se encontra no centro topológico da rede, de forma a tornar mínima a distância ao roteador mais afastado da fonte de teste. O roteador ligado à fonte geradora de vetores de teste é chamado de TAS (*Test Acess Switch*), conforme mostrado na figura 2.11.

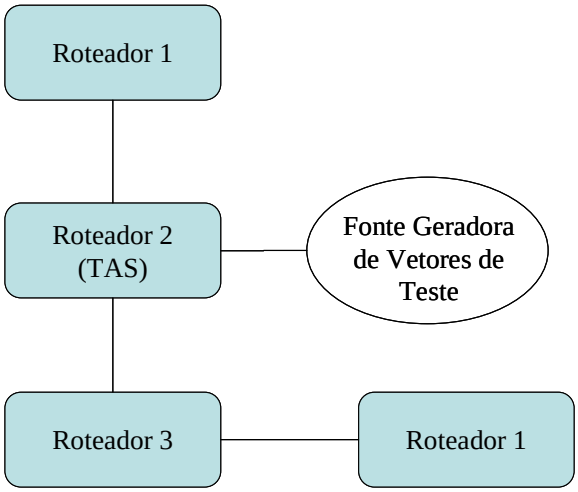


Figura 2 11: Representação do TAS (*Test Acess Switch*) proposto em (HOSSEINABADY, 2006).

No teste, o TAS recebe os vetores de teste e os carrega em sua cadeia *scan*. Ao mesmo tempo, os vetores de teste são enviados para os próximos roteadores da rede. Uma vez que a cadeia *scan* do roteador recebe os vetores de teste, os resultados obtidos da parte combinacional do circuito são capturados novamente na cadeia *scan*. Estes resultados são enviados para os próximos roteadores para comparação (assume-se que todos os roteadores são iguais). No caso de ocorrer alguma discrepância entre os resultados, uma sinalização de erro é enviada ao roteador anterior até chegar ao TAS, onde a sinalização é enviada ao ATE.

Para a realização do teste, contudo, se faz necessário uma envoltória de teste para os roteadores. A envoltória proposta por Hosseinbady (2006) é apresentado na figura 2.12.

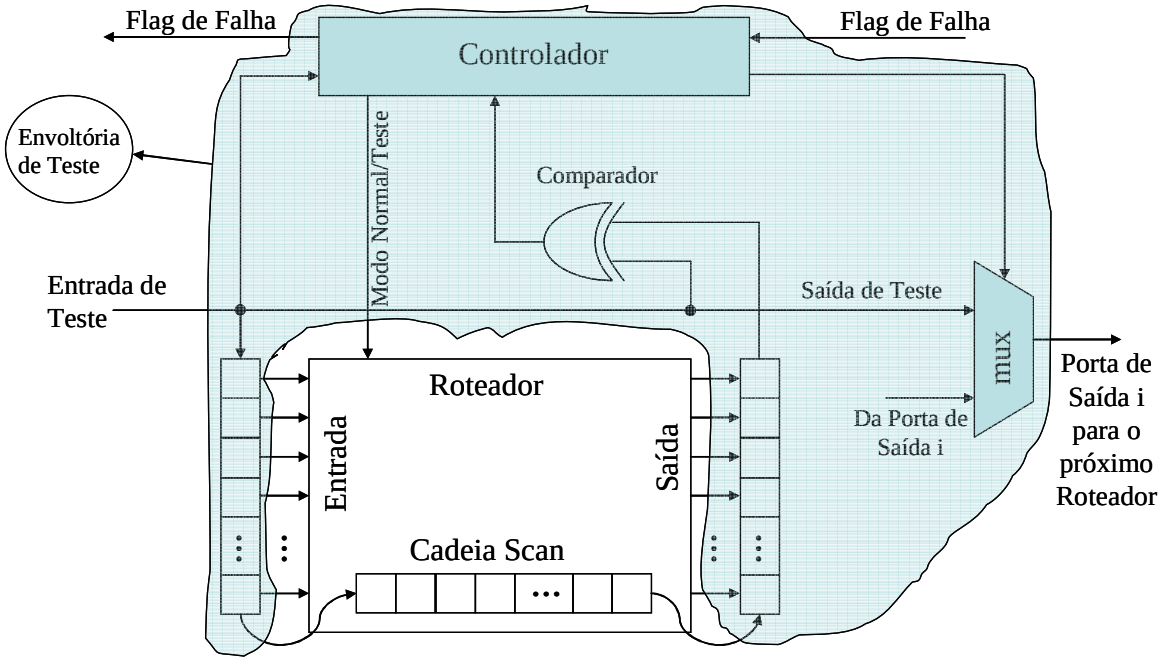


Figura 2 12: Envoltória de teste do esquema proposto em (HOSSEINABADY, 2006).

Como pode ser visto na figura 2.12, a envoltória de teste apresentada por Hosseinadaby (2006) considera o uso de uma cadeia *scan* interna aos roteadores da

rede. A cadeia *scan* é carregada através de dados oriundos das entradas funcionais dos roteadores quando estes se encontram em modo de teste. Uma vez capturados os resultados do teste na cadeia *scan*, estes são extraídos para as saídas primárias do roteador onde são comparados com os valores esperados. Estes valores esperados utilizados na comparação, também são transportados até a envoltória de teste através das interconexões funcionais da NoC.

Outra envoltória de teste é proposta por Liu (2006). Assim como em Grecu (2005), em Liu (2006) é proposta a utilização progressiva dos recursos da NoC durante o teste, onde a rede é utilizada para transportar os vetores de teste. A envoltória de teste proposta no trabalho é compatível com o padrão IEEE std. 1500, conforme mostrado na figura 2.13.

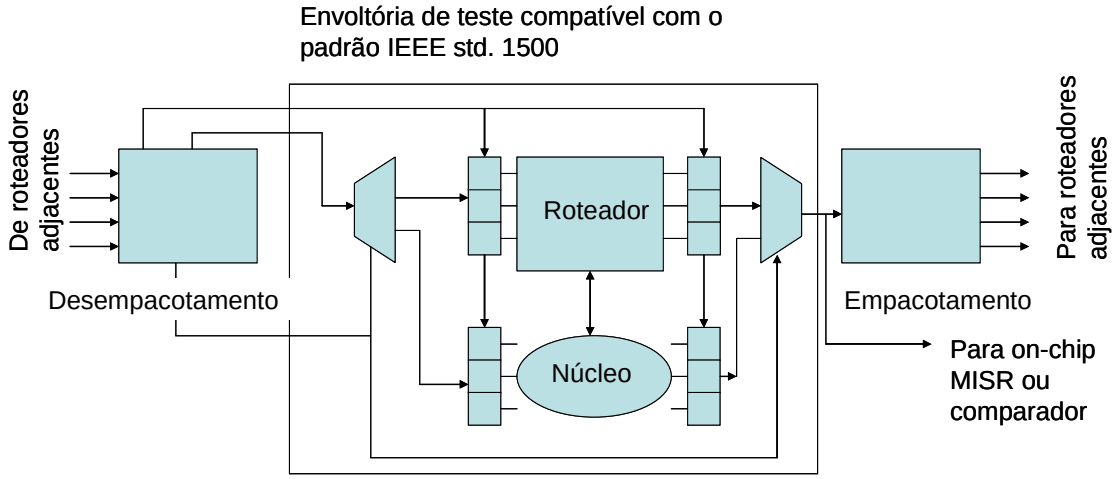


Figura 2 13: Wrapper de teste do esquema de teste proposto em (LIU, 2006).

Liu (2006) complementa os trabalhos mencionados anteriormente levando em consideração o teste dos núcleos do sistema. O método de escalonamento do teste proposto possibilita testar os núcleos paralelamente aos roteadores não testados, uma vez que tenha se estabelecido um caminho testado até o núcleo sob teste. Assim, uma vez que os roteadores que fazem parte do caminho dos vetores de teste até o núcleo sob teste tenham sido testados, este núcleo pode ser testado. Este escalonamento diminui o tempo de teste, já que não é necessário testar todos os roteadores antes de começar o teste dos núcleos.

As técnicas baseadas em reuso das interconexões funcionais para transporte dos vetores de teste, em geral, assumem que as interconexões da rede não possuem falhas ou foram testadas previamente. O teste dessas estruturas (interconexões), entretanto, não é abordado em nenhum dos trabalhos citados até aqui.

Na seção seguinte, são apresentados trabalhos que propõem métodos de teste voltados às interconexões de redes-em-chip.

2.2.2 Teste das Interconexões

O teste das interconexões, em geral, tem sido abordado de forma separada do teste dos outros componentes da NoC, como os roteadores. Contudo, o teste dessas estruturas deve ser visto dentro de um contexto de teste do sistema como um todo. Isto fica evidenciado no caso do reuso da NoC como mecanismo de acesso ao teste visando o

teste dos roteadores e até mesmo dos núcleos, já que para utilizar as interconexões na transmissão dos vetores de teste, estas devem ser devidamente testadas a priori.

Grecu (2006_a) propõe uma estrutura de BIST para teste das interconexões das NoCs. A motivação para a utilização de uma estratégia de BIST está ligada à baixa controlabilidade e observabilidade dos sinais sob teste. O modelo de falhas proposto para utilização no trabalho é o chamado *Maximum Agressor Fault* (MAF) (Cuviello apud Grecu, 2006_a). Este modelo de falhas que representa os defeitos que levam a um dos seis erros de *crosstalk* (atraso de subida/descida, espícula positiva/negativa e aceleração subida/descida), conforme apresentado na figura 2.14.

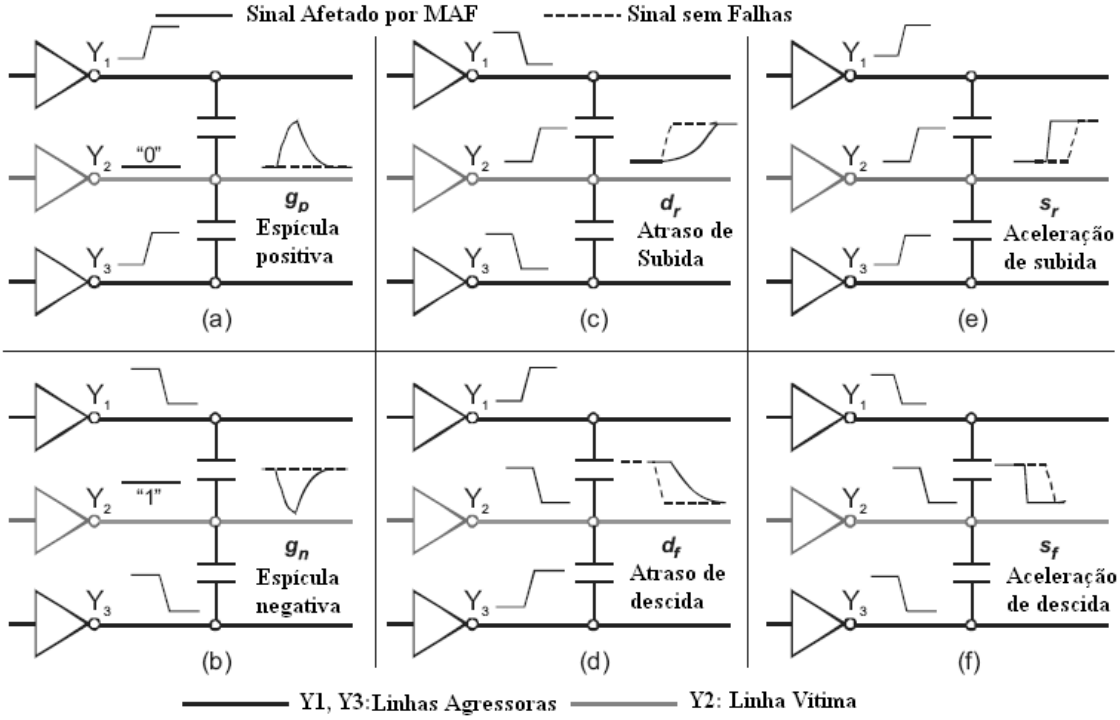


Figura 2 14: Falhas representadas pelo modelo MAF (GRECU, 2006_a).

No MAF, uma linha chamada vítima é afetada por transições em todas as outras interconexões, que são chamadas agressoras, devido ao efeito de *crosstalk.*

Levando em conta este modelo de falhas, são propostas duas estruturas para implementar o teste: TDG (*Test Data Generator*) e o TED (*Test Error Detector*). O TDG possui uma máquina de estados que gera os vetores da seqüência utilizada para a detecção das falhas. Já o TED possui o mesmo circuito gerador da seqüência de teste, porém com uma defasagem de um ciclo de relógio. A comparação dos dados recebidos com os dados gerados é feito com portas XOR. A seqüência de teste gerada pelo TDG é apresentada na figura 2.15.

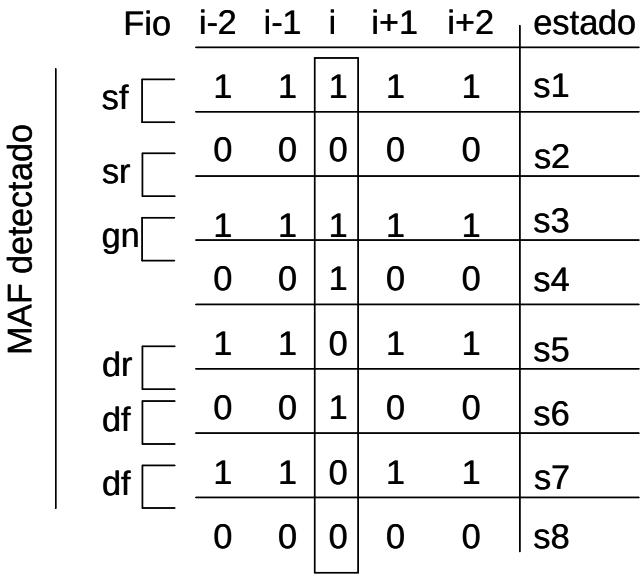


Figura 2 15: Vetores de teste para detecção das falhas do modelo MAF (GRECU, 2006_a). A figura representa um conjunto de 8 vetores de teste a serem aplicados para cada fio vítima (*i*), sendo que os demais fios recebem os valores atribuídos aos fios agressores.

A implementação do teste pode ser feita de três formas. A primeira consiste na utilização de pares TDG/TED nas interconexões entre os roteadores (figura 2.16). Esta implementação acarreta um custo muito alto de área proporcional à quantidade de ligações entre roteadores.

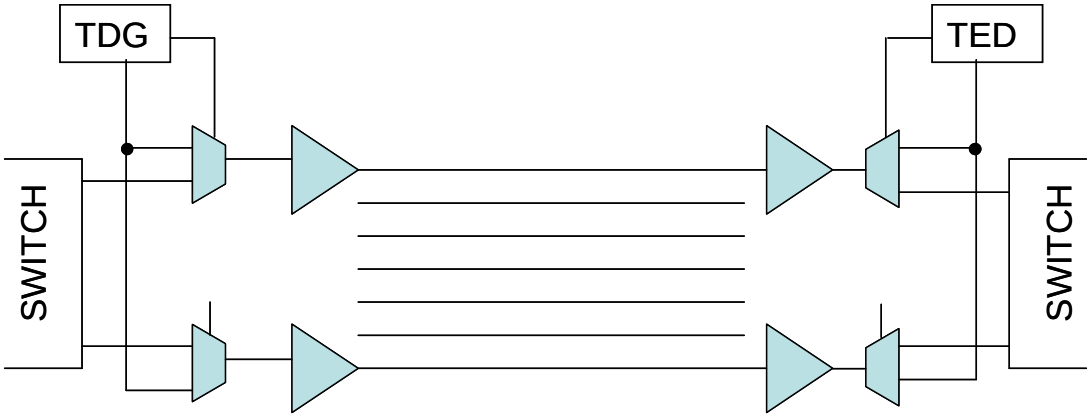


Figura 2 16: Utilização de pares TDG/TED para teste das interconexões da NoC (GRECU, 2006_a).

A segunda forma de teste utiliza apenas um TDG e insere um controlador (GTC – *Global Test Controller*). Isto reduz o custo de área em relação à primeira forma de teste, já que para cada conexão entre roteadores é acrescentado apenas um TED (somente um TDG é utilizado no teste). Neste teste, a informação de teste dos roteadores é intercalada com a informação de teste das interconexões. Esta implementação é mostrada na figura 2.17.

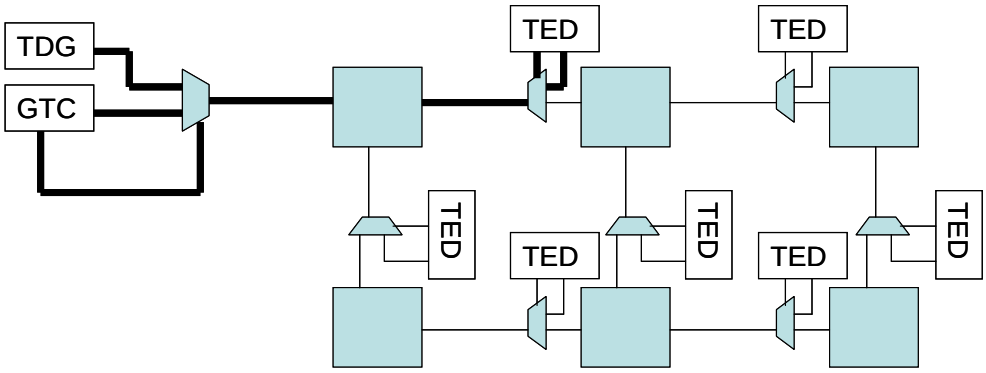


Figura 2 17: Representação do teste utilizando apenas um TDG e um GTC com comunicação *unicast* (GRECU, 2006_a), onde os canais em negrito representam a transmissão dos dados.

Já a terceira forma proposta para a implementação do teste é muito similar à segunda, porém se vale do paralelismo da comunicação *multicast*. A figura 2.18 representa a configuração proposta.

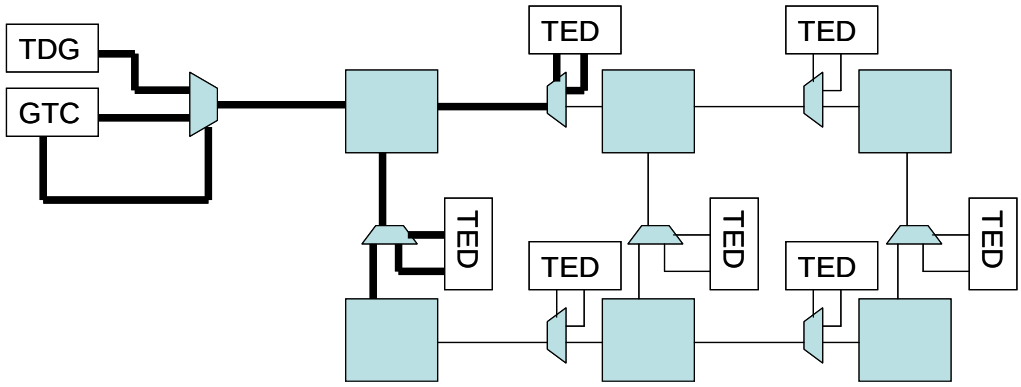


Figura 2 18: Representação do teste utilizando apenas um TDG e um GTC com comunicação *multicast* (GRECU, 2006_a), onde os canais negrito representam a transmissão dos dados.

Jutman (2005) propõe um método de teste de interconexões baseado no uso de circuitos de BIST. O modelo de falhas considerado são falhas do tipo *stuck-at* e falhas de *bridging*. No trabalho são apresentadas diversas seqüências de teste aplicáveis ao teste de interconexões sendo que é proposto o uso da seqüência de teste chamada de ITCC (*Interleaved True/Complement Code*). Também são propostos como circuitos de BIST, circuitos geradores de vetores de teste e analisadores de resposta.

Outra forma de teste em interconexões é apresentado por Cota (2007). A proposta utiliza modelo de falhas de curto-circuito (*bridging*) do tipo *wire-and*, *wired-or* e *strong-driver*. As falhas consideradas podem ocorrer entre diferentes canais de comunicação devido à irregularidade que a NoC pode apresentar dentro de um SoC.

A metodologia propõe o envio de seqüências de teste pelos núcleos ligados aos roteadores, percorrendo um caminho de teste determinado. A mensagem é, então, recebida pelo núcleo destino e comparada a fim de detectar erros. Uma descrição mais detalhada deste método de teste é apresentada em 3.1.

Outro método de teste funcional foi proposto por Raik (2006) visando os roteadores da NoC. Neste trabalho, é realizado o tráfego de mensagens na rede ao mesmo tempo em que são injetadas falhas. O trabalho apresenta uma detecção de falhas acima de 80%.

2.2.3 Comparação dos Métodos

A tabela a seguir (Tabela 2.1) apresenta um quadro comparativo de alguns dos métodos citados nas seções anteriores.

Tabela 2. 1 Quadro comparativo dos métodos apresentados nas seções 2.2.1 e 2.2.2.

Referência	**Elemento da NoC testado**	**Modelo de falhas**	**Inserção de cadeias scan**	**BIST**	**Envoltória de teste**	**Mecanismo de acesso ao teste**
Amory (2005)	Roteadores	*Stuck-at*	Cadeias *scan* parciais.	Comparador interno (na envoltória de teste)	Envoltória baseada no padrão IEEE std. 1500.	TAM dedicada
Aktouf (2002)	Roteadores	*Stuck-at*	-	Comparador Interno (na envoltória de teste) BIST para teste de memória.	Envoltória *boundary scan*	TAM dedicada
Grecu (2005), Hosseinadaby (2006), Liu (2006)	Roteadores	*Stuck-at*	Cadeias *scan* na lógica interna do roteador.	Circuitos de BIST distribuídos para teste de memória.	Envoltória baseada no padrão IEEE std. 1500.	Reuso progressivo da NoC.
Grecu (2005)	Interconexões	MAF	-	Circuitos de BIST: Geradores de vetores de teste e analisadores de resposta.	-	TAM dedicada + utilização das interconexões da NoC.
Cota (2007)	Interconexões	Falhas de curto circuito do tipo *AND* lógico e *OR* lógico	-	Circuitos de BIST: Geradores de vetores de teste e analisadores de resposta.	-	-

RESUMO DO CAPÍTULO 2

O capítulo 2 apresentou alguns conceitos básicos da área de teste de circuitos integrados, tais como: defeito, falha, modelos de falhas e erro. Dentre os modelos de falhas foram apresentados os modelos do tipo *stuck-at* e *bridging*, já que estes modelos são utilizados nos métodos de teste propostos no capítulo 3. Também foram apresentados os conceitos de DfT, *Built-In Self-Test* (BIST) e cadeias *scan*. Dois padrões de teste foram descritos brevemente: o padrão IEEE std. 1149.1 (*Boundary scan*) e o padrão IEEE std. 1500 (utilizado no teste SoCs).

Após serem apresentados os conceitos da área de teste de circuitos integrados, foram apresentados métodos de teste de roteadores e métodos de teste das interconexões das NoCs. Estes métodos de teste foram recentemente propostos na literatura.

3 MÉTODO PARA O TESTE FUNCIONAL DAS INTERCONEXÕES DE REDES-EM-CHIP

A estratégia de teste estudada neste trabalho baseia-se no uso da NoC em modo de operação normal afim de transportar os vetores de teste através da rede. Este método é descrito nas seções que seguem.

3.1 Teste Funcional das Interconexões da NoC – Sinais de Dados

O primeiro método de teste funcional descrito nesse trabalho refere-se ao método apresentado por Cota (2007) e busca detectar falhas nas interconexões de uma NoC. O método baseia-se no envio de mensagens através de uma rede em chip operando em modo funcional. O método é descrito para NoCs de topologia grelha de tamanho 2x2 (quatro roteadores ligados entre si aos pares e ligados aos seus respectivos módulos IP), porém pode ser expandido para NoCs maiores, conforme é apresentado a seguir.

3.1.1 NoC Utilizada – Estudo de Caso

A rede utilizada como estudo de caso foi a NoC denominada SoCIN. A descrição mais detalhada desta rede é feita em capítulos anteriores (capítulo 1.5). Para os experimentos realizados, a rede foi configurada com 8 bits de dados, logo, cada canal de comunicação foi parametrizado para ter 8 interconexões de dados. Além das interconexões de dados, existem, também, interconexões com os sinais de controle de começo e fim de pacote (*bop* e *eop*, respectivamente) e sinais de controle de handshake (*ack* e *val*).

A topologia de rede utilizada foi do tipo direta, grelha 2D de tamanho 2x2. A análise foi estendida para redes de mesma topologia, porém com tamanhos maiores.

3.1.2 Modelo de Falhas

O modelo de falhas utilizado na metodologia de teste apresentada por Cota (2007) compreende falhas do tipo curto-circuito nas interconexões da NoC. Dentre as possíveis falhas de curto-circuito, apenas são consideradas, nesse modelo de falhas, curtos-circuitos que causem, nos fios afetados, um comportamento do tipo AND lógico ou OR lógico. A essas falhas se dá o nome de falhas de curto-circuito do tipo *wired-and* e *wired-or*, respectivamente (conforme descrito no capítulo 2). Dois fios que apresentem falhas de curto-circuito do tipo *wired-and* terão seu nível lógico dependente um do outro obedecendo à tabela verdade de uma porta lógica AND, sendo que o valor resultante da operação lógica é atribuído para as duas interconexões. De forma similar isto ocorre para as falhas de curto-circuito do tipo *wired-or*, sendo que a operação lógica realizada é do tipo OR.

Em um primeiro momento apenas são consideradas como sujeitas a falhas as interconexões de dados da NoC de estudo, desconsiderando-se da análise os fios responsáveis por sinais de controle como os sinais *ack* e v*al*, por exemplo, do protocolo de *handshake* da NoC estudada. Também são retirados da análise os sinais de controle responsáveis pelos *bits* de indicação de começo e fim do pacote (*bop* e *eop*). Isto é feito com o objetivo de facilitar a análise inicial.

O número total de interconexões analisadas na metodologia de teste, portanto, restringe-se ao número de fios de dados da NoC utilizada como estudo de caso, que é parametrizável e será representado por *w*. No estudo de caso, como dito anteriormente, a NoC foi configurada com 8 *bits* de dados, logo *w* = 8 (8 interconexões de dados por canal de comunicação).

A vizinhança na qual estas falhas são analisadas é restrita a uma NoC de tamanho 2x2, ou seja, o conjunto de interconexões que podem apresentar falhas de curto-circuito entre si pertence a esta vizinhança. Para NoCs maiores, um conjunto de testes em NoCs de tamanho 2x2 deverá ser estabelecido, conforme apresentado nas próximas seções. Levando-se em consideração essa restrição, existem 16 * *w* interconexões envolvidas na análise (onde 16 representa o número de canais de comunicação em uma NoC de tamanho 2x2, conforme representado na figura 3.1). Para *w* = 8, o número total de interconexões consideradas é de 16 * 8 = 128.

Dessa forma, a quantidade total de falhas contidas no modelo é definida na Equação 1, na forma genérica, e na Equação 2, particularizando para o estudo de caso, e representa a combinação de todas as interconexões de dados agrupadas duas a duas, já que as falhas ocorrem aos pares. No estudo de caso são analisadas 8128 falhas. A justificativa desse modelo de falhas se deve a forma como as interconexões são distribuídas dentro do circuito integrado.

$$C_2^w = \frac{(16 \cdot w)!}{2! \cdot ((16 \cdot w) - 2)!} \qquad (1)$$

$$C_2^{128} = \frac{128!}{2! \cdot (128 - 2)!} = 8128 \qquad (2)$$

3.1.3 Ambiente de Simulação

Para simular o teste proposto e validar a metodologia apresentada, foi montado um ambiente de simulação (circuito de *testbench*) utilizando a ferramenta ModelSim 6.0. Os circuitos de teste utilizados no ambiente de simulação foram descritos em linguagem de descrição de hardware VHDL em um nível de abstração comportamental, sem preocupação inicial com a capacidade de síntese do circuito descrito. A NoC utilizada como estudo de caso também foi representada em VHDL, porém em nível RTL (ZEFFERINO, 2003).

Através de comandos específicos da ferramenta de simulação utilizada (ModelSim 6.0), como o comando *signal_force*, os sinais das interconexões sob teste puderam ser acessados e manipulados. Assim, foi possível, dentro do ambiente de simulação, injetar as falhas desejadas nas interconexões de interesse. A seguir são mostrados os comandos utilizados para injetar falhas nas interconexões desejadas:

signal_force (list_signals(j), value, open, freeze,open,1);

Dessa forma, o ambiente de simulação foi composto de uma descrição da NoC, de topologia direta grelha 2D de tamanho 2x2, com quatro roteadores, além de quatro circuitos descritos em VHDL comportamental responsáveis por enviar os pacotes para a rede e receber os pacotes da rede, respeitando o protocolo de comunicação (chamados de "nós" da NoC). Esses mesmos circuitos foram responsáveis pela comparação dos pacotes recebidos com os valores esperados e pela sinalização de erro no caso de alguma discrepância. Além disso, um mecanismo de injeção de falhas foi criado dentro desse ambiente de simulação utilizando os comandos descritos anteriormente e proporcionando um controle das falhas injetadas. Este ambiente de simulação é representado na figura 3.1.

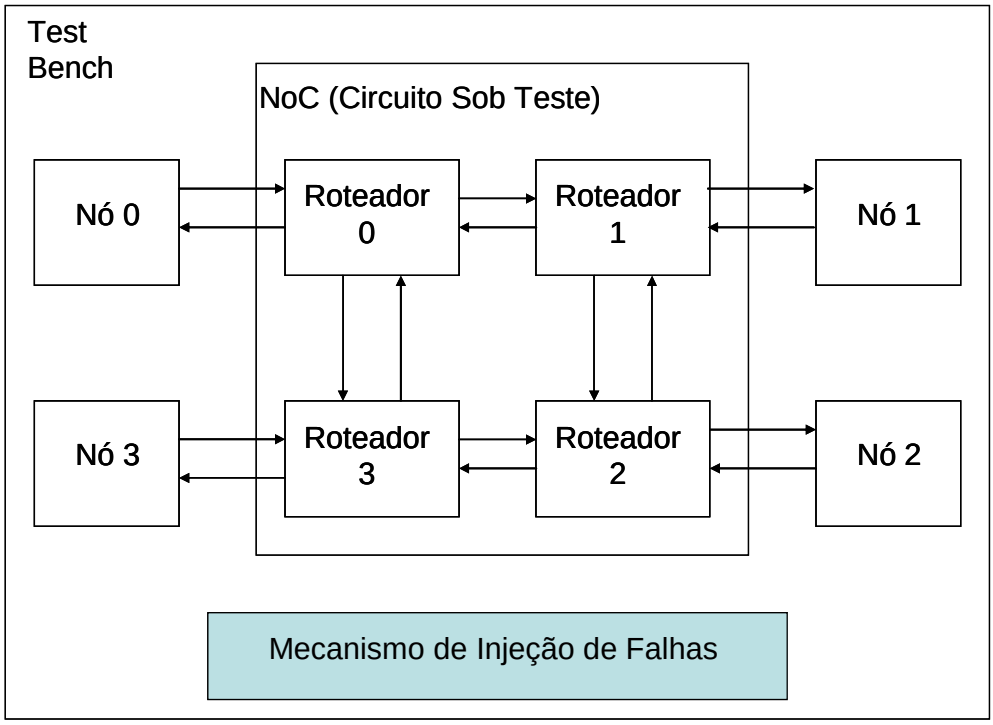


Figura 3. 1: Representação do ambiente de simulação utilizado.

3.1.4 Método de Envio de Mensagem na Rede Para o Teste das Interconexões

O método consiste no envio de pacotes através da rede, com a rede operando em modo funcional. A forma como é feito o envio dos pacotes na rede busca minimizar o tempo de teste além de obter 100% de detecção das falhas contidas no modelo de falhas. Para isso assume-se o seguinte:

- Consideração 1: Conforme mencionado, assume-se os roteadores em modo funcional, com o protocolo de comunicação respeitado (pacotes com cabeçalho/*header*, *payload* e terminador/*tail*, conforme mostrado na figura 3.2) para aplicar os vetores de teste nas interconexões da NoC. A seqüência de teste é transmitida através do *payload* dos pacotes enviados na rede. O conteúdo do cabeçalho do pacote depende da NoC e do algoritmo de roteamento (no estudo de caso trata-se de um roteamento XY).

Figura 3. 2: Representação do pacote enviado na rede.

- Consideração 2: Os pacotes devem ser enviados de forma que todos os canais de comunicação tenham suas interconexões de dados preenchidas com vetores de teste ao mesmo tempo, permitindo controlar essas interconexões na NoC considerada.
- Consideração 3: Assume-se que circuitos para geração de pacotes contendo os vetores de teste e analisadores de resposta são conectados nas interfaces da rede para enviar e receber os pacotes de teste. Estes circuitos podem representar estruturas de auto-teste (BIST) ou até mesmo módulos IP (núcleos) programáveis. A implementação desses circuitos é explorada nos capítulos seguintes.
- Consideração 4: Como todos os pacotes são enviados com cabeçalho (*header*) e terminador (*tail*), falhas de curto-circuito podem afetar esses *flits*, alterando, assim, o roteamento do pacote. Quando isso ocorre, o pacote pode ser roteado erroneamente ou um sinal de final de pacote pode não ser recebido, causando problemas na comunicação da rede e/ou fazendo com que o circuito destinatário não receba o pacote. Nos dois casos, se o circuito destinatário não receber o pacote ou o sinal de final de pacote em um certo intervalo de tempo predeterminado, uma falha de *timeout* será sinalizada.

  Se a falha de curto-circuito não causa perda de pacote, a informação útil do pacote (o *payload*) é recebida pelo circuito destinatário e conferida. No caso de alguma discrepância no valor das mensagens, um erro é sinalizado. Como a rede funciona em modo de operação normal durante o teste, é possível considerar, também, a possibilidade de outros tipos de erro, como *deadlock*. Nota-se, porém que devido ao esquema de roteamento da NoC utilizada como estudo de caso, este tipo de erro não ocorre.

A figura 3.3 ilustra a aplicação da seqüência de teste na NoC estudo de caso. O método proposto exercita todas as interconexões utilizando os roteadores da rede em modo funcional. Considerando-se a estratégia de roteamento XY, isso significa que quatro pacotes podem ser enviados através da rede paralelamente. Os caminhos traçados por estes pacotes são mostrados na figura 3.3 por quatro linhas diferentes, uma linha sólida, uma tracejada, uma linha com pontos e segmentos de reta e uma com linhas triplas. Cada linha representa um canal de comunicação contendo *w* interconexões de dados cada. O conjunto de teste composto pelo circuito ligado ao roteador *i*, $0 \leq i \leq 3$, é chamado de nó *i*. Cada tipo de linha representa um caminho de roteamento.

- **Caminho 1 (linha sólida)**: um *hop* para leste e um *hop* para sul, do nó 0 para o roteador 00, para o roteador 01, para o roteador 11 e finalmente para o nó 3. Um *hop* representa a passagem da mensagem através de um roteador da rede.
- **Caminho 2 (linhas com pontos e segmentos de reta)**: um *hop* para oeste e um *hop* para sul, do nó 1 para o roteador 01, para o roteador 00, para o roteador 10 e finalmente para o nó 2.

- **Caminho 3 (linhas traçejadas)**: um *hop* para leste e um *hop* para norte, do nó 2 para o roteador 10, para o roteador 11, para o roteador 01 e finalmente para o nó 1.
- **Caminho 4 (linhas triplas)**: um *hop* para oeste e um *hop* para norte, do nó 3 para o roteador 11, para o roteador 10, para o roteador 00 e finalmente para o nó 0.

Os nós ligados aos roteadores têm o papel de enviar, receber, analisar a resposta e sinalizar o erro, caso este ocorra. Dessa forma, consegue-se testar todas as falhas propostas.

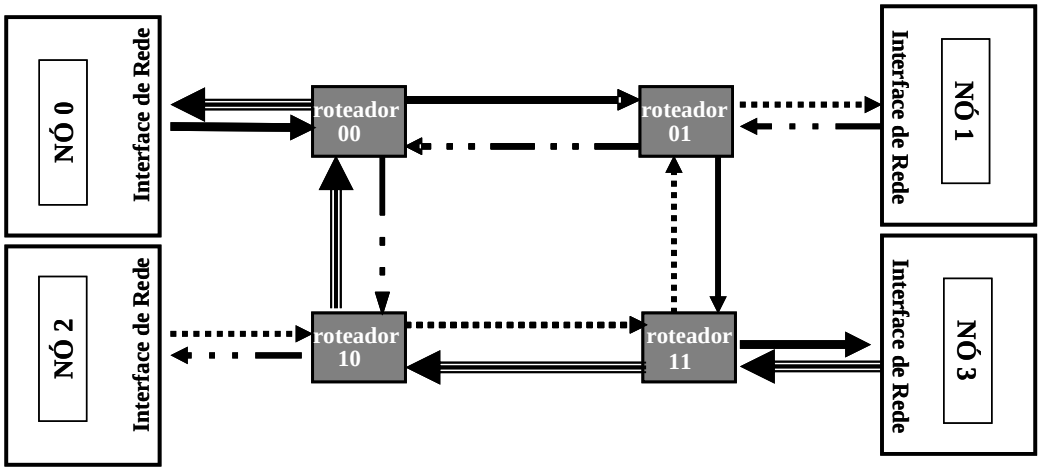


Figura 3. 3: Representação do circuito sob teste. Os nós representam os circuitos ligados aos roteadores, responsáveis pelo envio, recebimento da mensagem, além da sinalização de erro. Estes circuitos possuem uma interface de rede (*network interfaces*) representada externamente aos nós. Os elementos centrais representam os roteadores (*routers*).

3.1.5 Seqüência de Teste

Considerando-se o modelo de falhas adotado (falhas de curto-circuito), uma série de seqüências de teste podem ser utilizadas com a finalidade de detectar todas as falhas com poucos vetores de teste, como, por exemplo, a seqüência utilizada por Grecu (2006_a) para detectar as falhas do modelo MAF (apresentada no capítulo 2). Neste trabalho, porém, a questão da exploração da melhor seqüência visando um menor tempo de teste não é abordada. Simplesmente, com a finalidade de exemplificar o método proposto para uma determinada seqüência de teste, foi utilizada a seqüência *walking one*, que garante 100% de detecção para as falhas consideradas no modelo adotado (CHENG, 1990). Esta seqüência de teste foi incluída na mensagem enviada através da NoC conforme a descrição feita a seguir.

3.1.6 Composição do Pacote da Mensagem

A mensagem a ser enviada na rede é composta por um pacote que inclui em seu *payload* os vetores de teste descritos na seqüência de teste. Estes vetores são incluídos como *flits* de *payload* do pacote, precedidos de um *flit* de cabeçalho e seguido de um *flit* teminador conforme determinado pela especificação da SoCIN. A organização detalhada do pacote usado no método descrito é mostrado na figura 3.4.

A forma como os pacotes são enviados pelos diferentes nós da rede durante o teste, em função do tempo, é mostrado na figura 3.5.

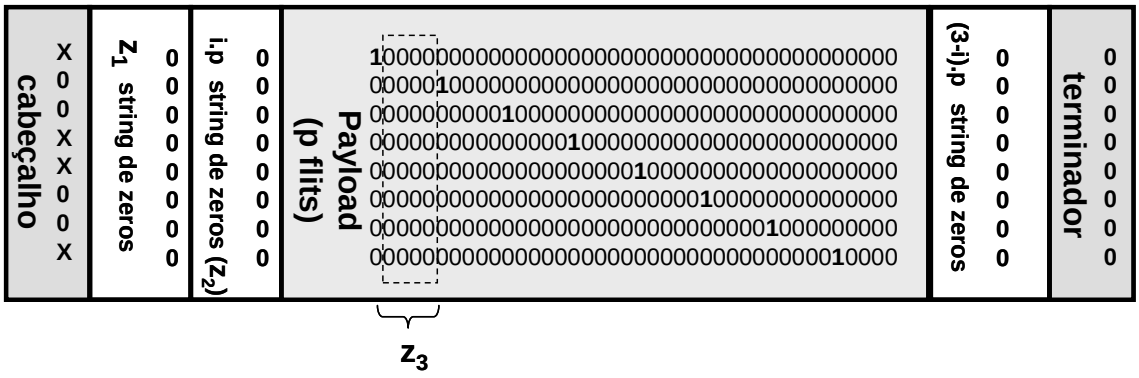


Figura 3. 4: Composição do pacote de teste enviado na rede.

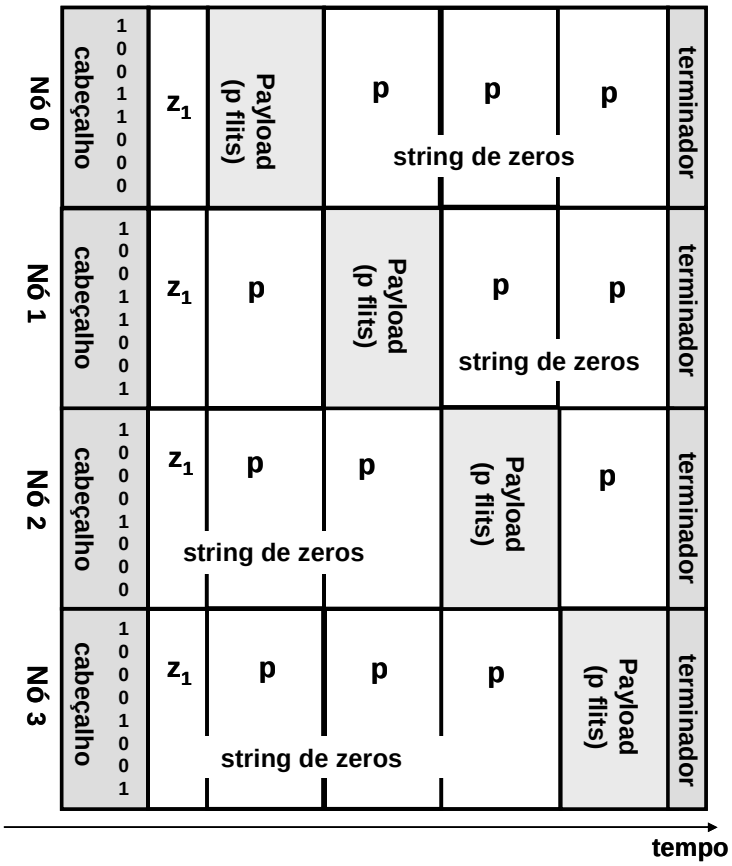


Figura 3. 5: Representação da forma de envio dos pacotes pelos nós em função do tempo. A figura mostra que os quatro pacotes são enviados ao mesmo tempo em um instante inicial. O pacote enviado por cada nó é composto de tal forma que os vetores de teste presentes entre os *flits* de *payload* acabam defasados.

Cada pacote é constituído de 6 partes, construídos conforme a descrição a seguir:

- *Parte 1*: O cabeçalho. É definido de acordo com a estratégia de roteamento da rede, contém a informação de roteamento. Estes *flits* são definidos como h. No caso da rede SoCIN (estudo de caso), o roteamento é XY conforme descrito anteriormente.
- *Parte 2*: Para garantir que apenas um fio esteja em nível lógico “1” em um dado momento, assim que o cabeçalho é enviado, *flits* contendo *strings* de zeros devem ser enviados para resetar todas interconexões (colocando-as em nível

lógico “0”). Chamamos de z_1 o número de flits utilizados para colocar em nível lógio 0 as interconexões após o envio do cabeçalho sendo este igual ao número de ciclos de relógio requerido para transmitir o *flit* de cabeçalho do nó fonte ao nó destino.

- *Parte 3*: Para defasar a aplicação dos vetores de teste, seqüências (*strings*) de zeros adicionais devem ser incluídos. Chamamos de z_2 o número adicional de *flits* em zero (com todos os bits em “0”) nessa parte do pacote. Para o nó 0, nenhum *flit* adicional é necessário. Para outros nós, contudo, z_2 depende do número do nó i ($0 \leq i \leq 3$) e do tamanho do *payload* (p), conforme mostrado nas Equações 3 e 4 (w e z_3 são definidos abaixo).

$$p = w \cdot (1 + z_3) \qquad (3)$$

$$z_2 = i \cdot p \qquad (4)$$

- *Parte 4:* O *payload* contém w vetores de teste, sendo w o número de interconexões de dados no canal de comunicação. Um vetor de teste consiste em um único *flit* contendo um único *bit* em nível lógico “1” (os restantes permanecem em valor lógico zero). Estes vetores de teste constituem a seqüência *walking one*. Contudo, entre cada vetor de teste, faz-se necessário o envio de *strings* de *flits* com todos os *bits* em zero. O tamanho da *string* de zeros entre os vetores de teste é igual ao número de ciclos de relógio necessários para enviar os *flits* de *payload* do nó fonte ao nó destino. Este número é chamado z_3. O número total de *flits* no *payload* (p) é, portanto, w * (1 + z_3), conforme mostrado na Equação 3.
- *Parte 5:* Novamente, *flits* adicionais de zeros devem ser incluídos para garantir que todos os fios estejam preenchidos com zero durante o teste, enquanto um vetor de teste está sendo transmitido através da rede. De acordo com o número (i) e o tamanho do *payload* (p), o número de flits em zero nessa parte do pacote é *(3-i) . p*.
- *Parte 6:* É o terminador (*tail*), definido de acordo com o protocolo da rede. O número de flits do terminador é representado por t e geralmente este valor é 1.

A Equação 5 define o tamanho do pacote de teste de acordo com a descrição acima. O tempo de teste resultante (em número de ciclos de relógio) para a seqüência de teste definida é dado pela Equação 6, onde L é a latência do pacote de teste até chegar ao nó destinatário.

$$S = h + z_1 + 4 \cdot p + t \qquad (5)$$

$$\text{Tempo total de teste} = S + L \qquad (6)$$

Todos os nós da NoC 2x2 enviam os pacotes ao mesmo tempo, contudo com as informações de *payload* defasadas no tempo. Para a rede de estudo de caso, h=1, z_1= 9,

z_3= 4, t=1, w = 8, e L = 11, o que resulta num total de 171 *flits* por pacote de teste e um tempo de teste de 182 ciclos de relógio para uma NoC mesh 2x2.

3.1.7 Escalabilidade do Método de Teste

Como dito anteriormente, o tamanho da NoC 2x2 foi utilizado para definir uma configuração de teste mínima para a detecção de falhas de curto circuito ao pares em fios dentro dessa vizinhança. Para redes maiores, um conjunto de configurações de teste baseadas na configuração mínima deve ser implementado. As configurações de teste que puderem ser feitas em paralelo são agrupadas na mesma rodada de teste, de forma a manter o tempo de aplicação do teste mínimo.

Por exemplo, assumindo uma NoC 5x5 conforme mostrado na Figura 3.6, a primeira rodada de teste (a) é obtida preechendo-se a NoC com configurações básicas de teste (representadas pelas áreas em cinza na figura). As outras rodadas de teste são mostrados em (b), (c), (d), (contendo 4 configurações de teste cada) e são obtidos ao deslocar a primeira rodada de teste para a direita e para baixo.

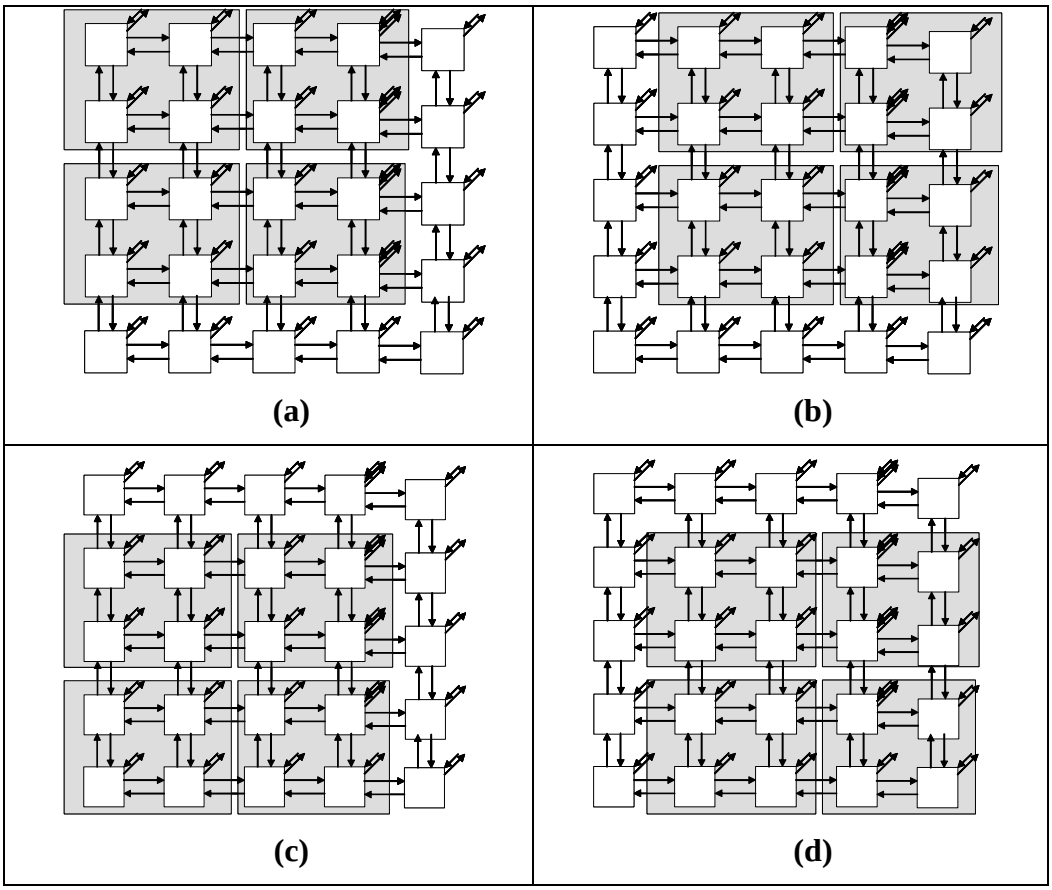


Figura 3. 6: Rodadas de teste para uma NoC de tamanho 5x5. (a) mostra a primeira rodada de teste; (b) mostra a segunda rodada de teste; (c) mostra a terceira rodada de teste; (d) mostra a quarta rodada de teste. As áreas em cinza representam as configurações de teste que podem rodar paralelamente numa mesma rodada de teste.

Estas 4 rodadas de teste não podem cobrir todas as falhas possíveis de curto-circuito na rede 5x5, contudo, podem detectar as falhas realistas de curto circuito em pares de fios de qualquer canal compreendidas na vizinhança 2x2 definida anteriormente.

As rodadas de teste podem ser facilmente obtidas para qualquer NoC *m*x*m*, *m*≥3, através do seguinte procedimento:

- Para obter a rodada de teste 1, preenche-se a NoC, da esquerda para a direita, de cima a baixo, usando tantas configurações de teste quanto possíveis conforme mostrado na figura 3.6(a);
- Desloca-se a rodada de teste obtida no passo 1, uma coluna para a direita para se obter a rodada de teste 2 (figura 3.6 (b));
- Desloca-se a rodada de teste obtida no passo 1, uma coluna para baixo para obter a rodada de teste 3 (figura 3.6 (c));
- para obter-se a rodada de teste 4, simultaneamente desloca-se a rodada de teste 1, uma coluna para a direita e uma coluna para baixo conforme mostrado na figura 3.6(d).

Para o exemplo da figura 3.6, dezesseis configurações de teste são aplicadas nas quatro rodadas de teste. Como mencionado, a janela de possibilidade de falhas é uma NoC 2x2, assim, o conjunto de fios que são suscetíveis a essas falhas são aqueles colocados de maneira próxima em nível de layout.

É assumida que a probabilidade de falhas além desse limite é muito baixa. Nota-se que a vizinhança de teste pode ser alargada com a adição de novas configurações de teste. Por exemplo, na NoC básica 2x2, não é exercitada a possibilidade de roteamento de N↔S e W↔E. Nesse caso, uma vizinhança de 3x3 poderia ser definida de forma que essas possibilidades adicionais de teste fossem contempladas, adaptando-se a seqüência de teste aplicada na rede.

Para uma NoC *m*x*m*, *m*≥3, o número de configurações de teste (tc) é dado por:

$$\boxed{tc = m^2 - 2 \cdot m + 1} \qquad (7)$$

O número de falhas que o método proposto pode detectar é calculado multiplicando-se o número de possíveis falhas de curto circuito em uma vizinhança de 2x2 (dado na Equação 1), pelo número de configurações de teste dados na Equação 7.

Embora tc cresça quadraticamente com m, de acordo com o procedimento mencionado anteriormente nessa seção, o número de rodadas de teste (tr) para qualquer NoC *m*x*m*, *m*≥3, é mantido constante, sendo igual a 4 (tr = 4). Isto ocorre devido ao fato que muitas configurações de teste podem ser acomodadas em uma mesma rodada de teste, podendo, assim, ser executadas em paralelo.

3.1.8 Simulações

As simulações feitas através da ferramenta ModelSim 6.0 resultaram nas formas de onda apresentadas nas figuras a seguir. A figura 3.7 mostra o pacote enviado pela rede. Já a figura 3.8 mostra uma falha de roteamento causada pelo mecanismo de injeção de falhas. Por fim a figura 3.9 mostra a injeção de falhas afetando um dos vetores de teste contidos no pacote de teste enviado na rede.

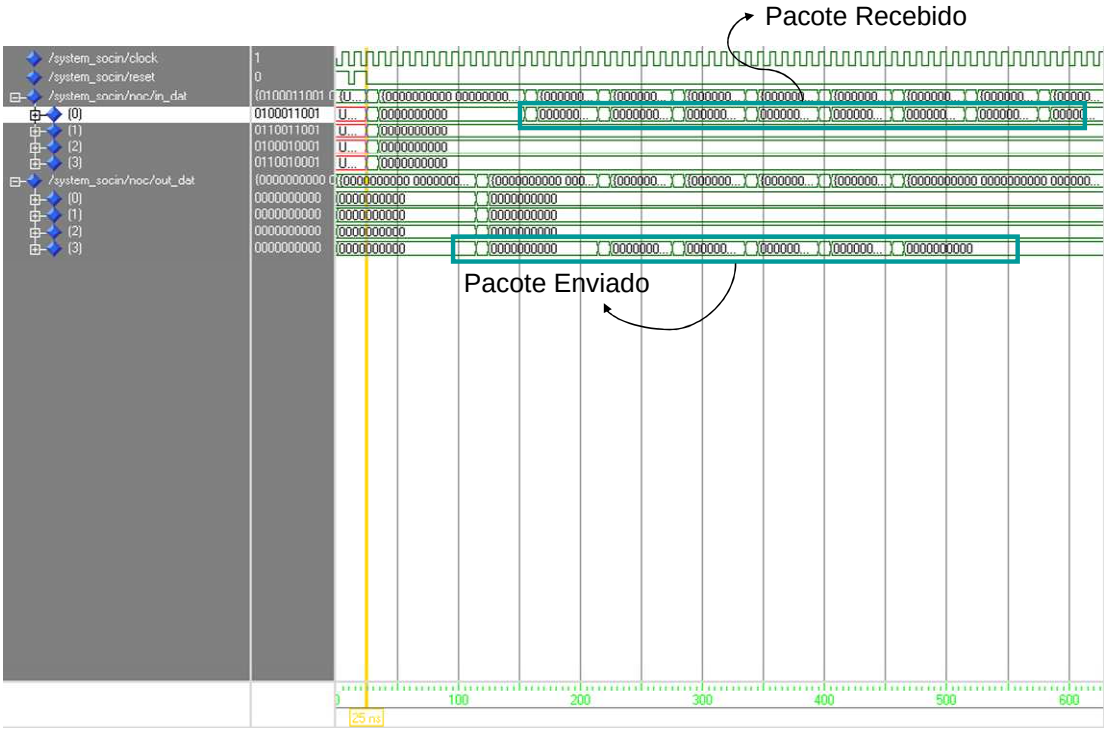


Figura 3. 7: Forma de onda com o envio dos pacotes de teste.

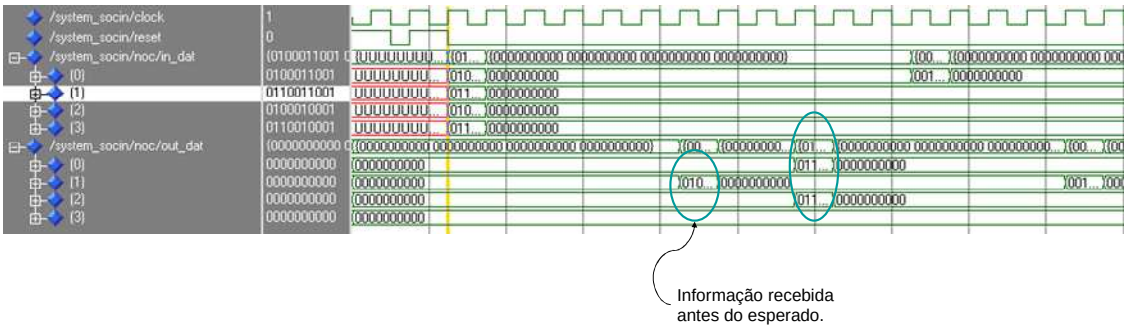


Figura 3. 8: Falha na informação de roteamento da mensagem.

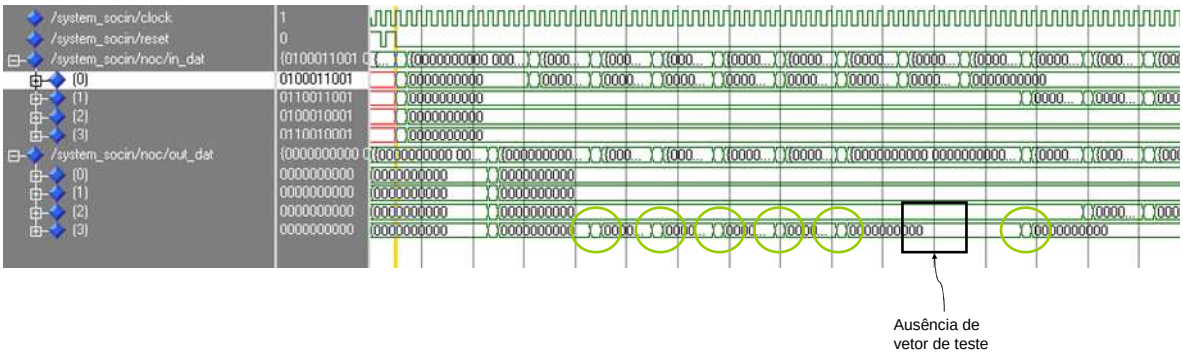


Figura 3. 9: Falha no vetor de teste enviado na rede. O valor lógico que deveria ser 1 em um determinado fio está em nível lógico 0.

3.1.9 Resultados

Todas as falhas compreendidas no modelo de falhas foram detectadas conforme apresentado na tabela a seguir.

Tabela 3. 1: Resultados da simulação do método apresentado.

Experimento	**# ciclos de relógio**	**Tipo de sinalização de erro**		
		Erro de *Payload*	***Timeout***	**Total Detectado**
Falhas nas interconexões de dados: 8.128 falhas injetadas				
Curto-circuito tipo *AND*	182	6092 (74,95%)	2036 (25,05%)	8128 (100%)
Curto-circuito tipo *OR*	182	7616 (93,70%)	512 (6,30%)	8128 (100%)

A Tabela 3.1 apresenta os resultados para a simulação com injeção de falhas. As falhas foram classificadas como erro de *payload* (coluna 3) e erro de *timeout* (coluna 4). Os erros de *timeout* correspondem aos erros ocasionados por falhas que afetam a informação de cabeçalho do pacote, fazendo com que a mensagem não atinja seu respectivo nó destinatário. Essa falha é detectada através de um estouro de contagem em um contador localizado no nó responsável por receber o pacote. Dessa forma, se o pacote não for recebido dentro de um tempo pré-estabelecido, o erro é assinalado. Já os erros de *payload* correspondem aos erros ocasionados por falhas nos demais *flits* do pacote, mantendo a informação de roteamento intacta. Esta detecção é feita através de comparação com valores esperados pelo nó que recebe o pacote.

Primeiramente, as 8128 falhas foram injetadas exaustivamente em todos as 128 interconexões de dados da NoC 2x2. As linhas 1 e 2 da tabela mostram os resultados para as falhas de curto circuito do tipo AND e OR lógico, respectivamente, nos fios de dados apenas, quando a seqüência *Walking One* completa é enviada em cada pacote e os 4 pacotes são enviados simultaneamente. Para cada falha, apenas uma detecção é feita. A seqüência utilizada é capaz de detectar todas as falhas em 182 ciclos de relógio, exercitando todos os caminhos de roteamento da rede em modo funcional.

3.2 Teste Funcional das Interconexões da NoC – Inclusão dos Sinais de Controle

A inclusão das falhas nos sinais de controle complementa o método descrito na seção 3.1, ao levar em consideração falhas anteriormente não contempladas pelo método de teste. A metodologia de teste utilizada para detectar as falhas nos sinais de dados e de controle foi proposto por Cota (2008) e também busca detectar as novas falhas através do envio de mensagens na rede, com esta operando em modo funcional.

3.2.1 NoC Utilizada – Estudo de Caso

A rede utilizada como estudo de caso foi a mesma apresentada na seção anterior (SoCIN), mantendo-se também a mesma topologia de rede (grelha 2D de tamanho 2x2). Manteve-se, também, a mesma largura do canal de comunicação (8 *bits* de dados).

3.2.2 Modelo de Falhas

O modelo de falhas utilizado foi o mesmo apresentado em 3.1, foram incluídos os sinais de controle. Para cada canal de comunicação, existem 4 interconexões de controle (*bop, eop, val, ack*) além das *w* interconexões de dados. Com isso, o número de interconexões compreendidas na análise aumentou, assim como o número total de falhas.

O número total de interconexões consideradas passou a ser 16 * (4 + *w*), sendo que 16 corresponde ao número de canais de comunicação, 4 representa o número de sinais de controle por canal de comunicação e *w* a quantidade de interconexões de dados. Considerando-se *w* = 8, conforme mencionado anteriormente, o número de interconexões analisadas passou a ser 192.

Igualmente, as Equações 1 e 2 tiveram de ser alteradas para compreender as novas interconexões. Com isso, foram obtidas as Equações 8 e 9. Estas equações representam o número total de falhas consideradas no modelo em função da largura do barramento de dados e mantêm-se como a combinação do total de interconexões agrupadas dois a dois, já que as falhas ocorrem aos pares.

$$C_2^w = \frac{(16\cdot(4+w))!}{2!\cdot((16\cdot(4+w))-2)!} \qquad (8)$$

$$C_2^8 = \frac{(16\cdot(4+8))!}{2!\cdot((16\cdot(4+8))-2)!} = 18336 \qquad (9)$$

3.2.3 Ambiente de Simulação

Para simular o teste proposto e validar o método apresentado, o mesmo ambiente de simulação utilizando a ferramenta ModelSim 6.0 foi utilizado. A única modificação foi a inclusão dos sinais de controle no mecanismo de injeção de falha.

3.2.4 Método do Envio de Mensagem na Rede Para o Teste das Interconexões

Basicamente, o método de envio de mensagens na rede para teste das interconexões foi mantido igual. A grande alteração foi a inclusão de uma defasagem no envio dos pacotes por parte dos nós da rede. No método anterior, os pacotes eram todos enviados simultaneamente a partir de um instante inicial. Nesse novo método, com o modelo estendido, o primeiro pacote é enviado pelo nó 0 no instante inicial e o nó seguinte só começa a enviar o seu pacote após um atraso pré-determinado.

3.2.5 Composição do Pacote da Mensagem

Utilizando o método apresentado em 3.1, foi observado que algumas falhas nos fios de controle (*bop* e *eop*) não são detectadas porque todos os pacotes são enviados simultaneamente. Como os 4 pacotes possuem o mesmo tamanho e sua transmissão de início e final ocorrem de forma sincronizada na rede, os fios de *bop* e *eop* para ambas as direções em um mesmo canal recebem sempre os mesmos valores (figura 3.10). Assim, falhas entre esses fios não são detectadas. Para resolver este problema, foi inserida uma defasagem no tempo de partida da transmissão de cada pacote de teste. Por exemplo, o nó 0 envia seu pacote em um instante inicial e somente após o cabeçalho deste pacote ter alcançado seu destino o nó 1 poderá começar a enviar o seu respectivo pacote e assim por diante, conforme ilustrado na Figura 3.11. Nota-se que *strings* adicionais de zeros são colocados após o cabeçalho de forma que o *flit* de cabeçalho e o *flit* de *payload* não atravessem a rede ao mesmo tempo.

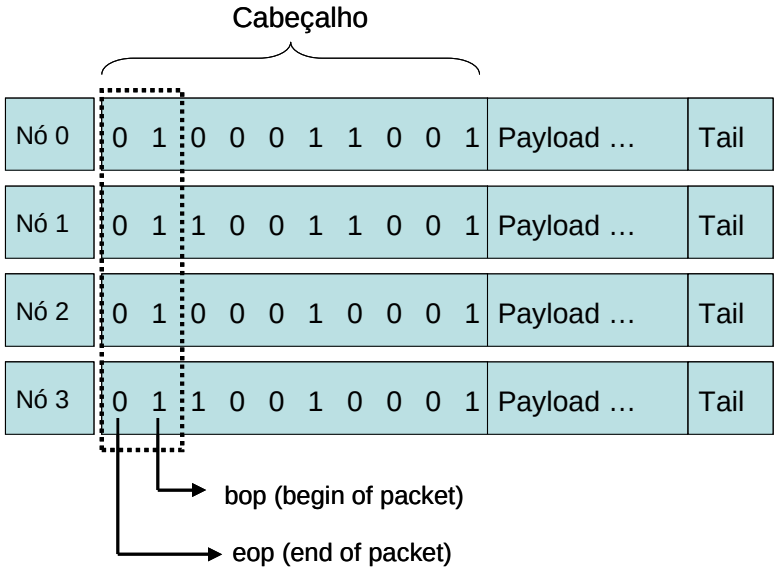


Figura 3. 10: Representação do envio de pacotes do método apresentado em 3.1. Na figura pode ser visto que os *bits* de *bop* (*begin of packet*) e *eop* (*end of packet*) possuem o mesmo valor lógico num mesmo instante de tempo, impossibilitando, assim, a detecção das falhas.

Além disso, algumas falhas nos fios de *ack* e *val* não podem ser detectadas a menos que dois pacotes sejam enviados, um depois do outro, através do mesmo caminho na NoC. A razão é que algumas das falhas só podem se manifestar quando os roteadores estão processando o *flit* terminador (*tail*) de um pacote e no próximo ciclo de relógio o *flit* de cabeçalho de um novo pacote. Assim, além de defasar no tempo a transmissão de pacotes, um segundo pacote deve ser acrescentado à seqüência de teste em cada nó. O segundo pacote possui apenas *flits* de cabeçalho e *tail*, já que todas as outras falhas foram detectadas com o primeiro pacote. *Strings* extras de zeros devem ser adicionados no final do primeiro pacote para garantir que um único *tail* ou cabeçalho esteja atravessando a rede em um dado momento.

A figura 3.11 mostra a aplicação da nova seqüência de teste enquanto a nova organização do pacote de teste é mostrada na figura 3.12. Cada pacote possui nove partes construídas conforme descrito a seguir:

- *Parte 1*: O cabeçalho, conforme definido em 3.1.

- *Parte 2*: Antes de enviar o *payload* do pacote, *strings* de zeros (contendo todos os *flits* em "0") devem ser enviadas para preencher os canais com zeros até que os *flits* de cabeçalho cheguem ao destino correspondente. Cada cabeçalho é separado por z_1 número de *flits* em zero.
- *Parte 3*: *Strings* de zeros adicionais são necessárias para defasar a aplicação do vetor de teste. Assim, z_4 é definido como o número de *flits* com zeros acrescentados a z_1 e antes do *payload*, e é calculado pela Equação 10.

$$z_4 = (3 - i) \cdot (z_1 + 1) \quad (10)$$

- *Parte 4*: O *payload*, conforme definido na Equação 3.
- *Parte 5*: *Strings* de zero adicional para preencher todos os canais com zeros após o *payload*, conforme definido na Equação 4.
- *Parte 6*: Novamente, *flits* de zeros adicionais devem ser incluídos para garantir que apenas um *flit* terminador (*tail*) ou um *flit* de cabeçalho atravesse a rede em um dado momento. Este número de *flits* em zero é chamado de z_5 e é definido pela Equação 11 de acordo com o número do nó (i) e o número de ciclos de relógio necessários para enviar um *flit* de cabeçalho de seu nó fonte até seu nó de destino (z_1). A constante 3 na Equação 11 representa o *flit* terminador acrescido de um *flit* de cabeçalho e outro *flit* terminador (oriundos do segundo pacote).

$$z_5 = i \cdot (z_1 + 3) \quad (11)$$

- Parte 7: O terminador, conforme definido anteriormente.
- Parte 8: O cabeçalho, conforme definido na seção anterior.
- Parte 9: O terminador, conforme definido anteriormente.

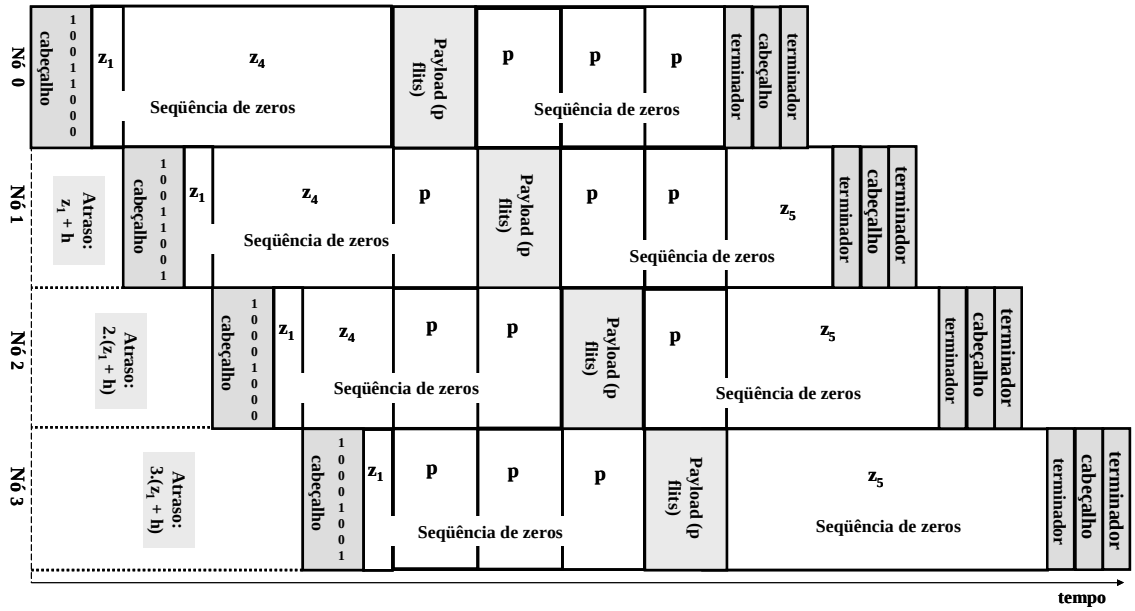


Figura 3. 11: Representação da forma de envio dos pacotes pelos nós em função do tempo. A figura mostra que os quatro pacotes são enviados de forma defasada.

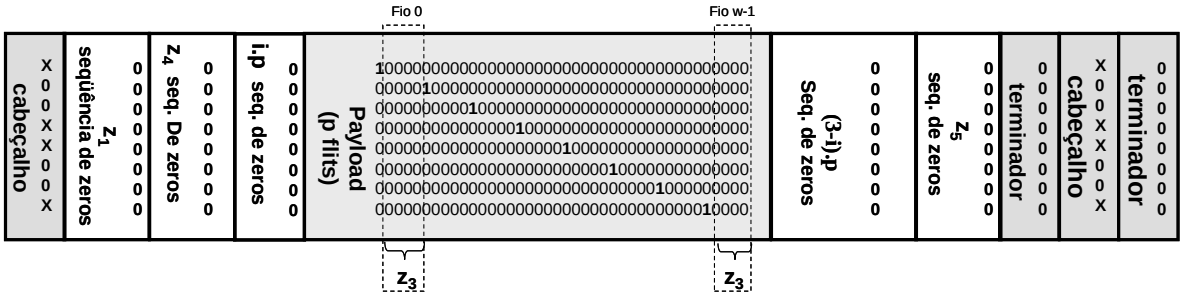


Figura 3. 12: Composição do pacote de teste enviado na rede.

Nota-se que agora os pacotes são enviados em tempos diferentes. A Equação 12 define o tamanho do pacote de teste para o modelo de falhas estendido de acordo com a descrição acima. No estudo de caso, h=1, z_1=9, w=8, z_3=4, para i = 0 o pacote contém 203 *flits*, para i = 1 o pacote contém 205 *flits*, para i = 2 o pacote contém 207 *flits*, e para i = 3 o pacote contém 209 *flits*.

$$S_i = 2h + 4 \cdot z_1 + 2 \cdot i + 4 \cdot p + 2 \cdot t + 3 \quad (12)$$

O tempo total de teste para a NoC básica é dado pela Equação 13, onde o primeiro termo indica o atraso para a entrega do cabeçalho do pacote do nó 3 (maior pacote de teste), o segundo termo indica o tamanho do pacote de teste do nó 3 e L é a latência para a transmissão do pacote. No estudo de caso, o tempo total de teste é de 250 ciclos de relógio.

$$\textit{Tempo total de Teste} = 3(z_1 + h) + S_{i=3} + L \quad (13)$$

3.2.6 Escalabilidade do método de teste

A escalabilidade do teste é feita da mesma forma como no método de teste apresentado em 3.1. Como conseqüência, o cálculo do número total de falhas que podem ser detectadas é calculado multiplicando-se o número de possíveis falhas de curto circuito em uma vizinhança de 2x2 (dado na Equação 9), pelo número de configurações de teste dados na equação 7.

3.2.7 Resultados

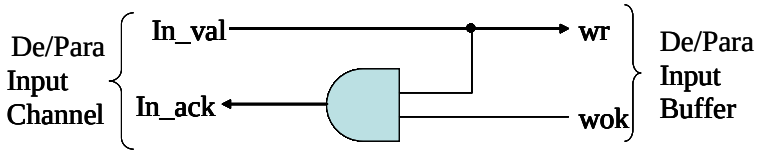
Um total acima de 99% das falhas compreendidas no modelo de falhas foram detectadas conforme apresentado na tabela a seguir.

Tabela 3. 2: Resultados da simulação do teste incluindo os sinais de controle na análise.

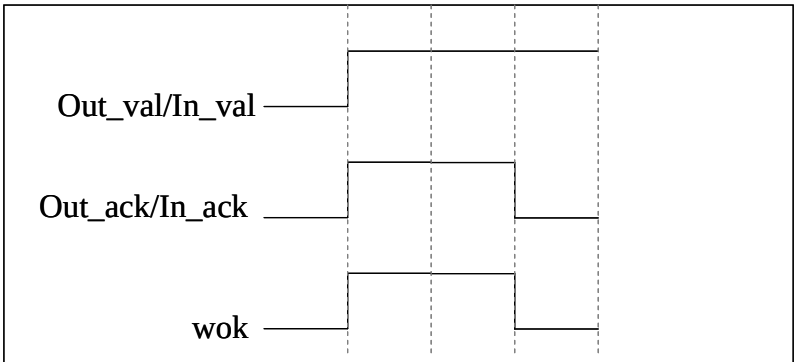
Experimento	**# ciclos de relógio**	**Tipo de sinalização de erro**		
		Erros de *Payload*	**Erros de *Timeout***	**Total de falhas Detectadas**
Falhas nas interconexões de dados e controle: 18.336 falhas injetadas				
Curto-circuito tipo *AND*	250	7049 (38,44%)	11275 (61,49%)	18324 (99,93%)
Curto-circuito tipo *OR*	250	10501 (57,27%)	7235 (42,73%)	18336 (100%)

A tabela mostra os resultados da simulação do teste considerado os sinais de controle, injetando-se 18336 falhas. Para falhas do tipo curto-circuito *OR*, 100% da

detecção é alcançada. Para falhas do tipo curto-circuito *AND*, apenas 12 de 18336 (0.07%) não são detectadas. As falhas não detectadas estão entre o *ack* e *val* num mesmo canal, onde ambos os fios recebem o mesmo valor (conforme ilustrado na figura 3.13). Para este caso, as falhas do tipo *wired-OR* causam erros de *timeout* no nó destinatário porque o *buffer* de entrada não pode indicar quando está cheio, já que não pode setar o *ack* para 0 (figura 3.13 (b)). Nesse caso os pacotes são perdidos e conseqüentemente a falha é detectada. Já no caso de falhas do tipo *wired-AND*, a falha não resulta em erro. Como se trata de uma operação *AND* entre os sinais de *ack* e *val*, quando o *ack* vai para 0, o *val* vai para 0 da mesma forma e vice versa (figura 3.13 (c)), coincidindo com o funcionamento sem falhas (figura 3.13 (a)). É importante notar que esse número pequeno de falhas não detectadas (menos de 1%) não afetam a correta funcionalidade da rede, ou seja, o funcionamento do circuito é o mesmo na presença ou não da falha.

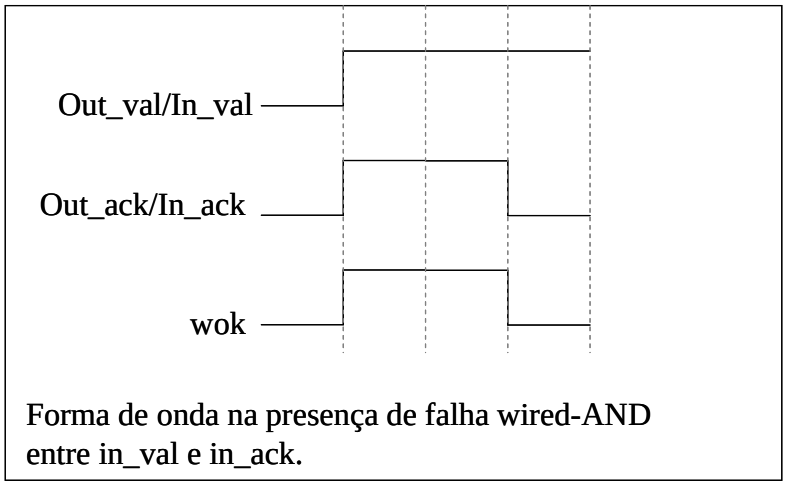


wok: sinal que sinaliza a disponibilidade de espaço para escrita no buffer.

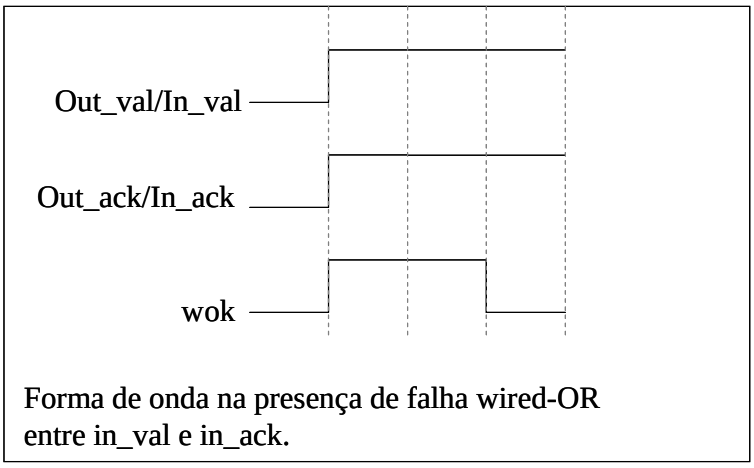


Forma de onda para operação sem falhas.

(a)



(b)



(c)

Figura 3. 13: A figura mostra a representação do controle de fluxo do roteador RASoC e as formas de onda dos sinais de controle durante uma operação: (a) sem falhas; (b) na presença de uma falha do tipo *wired-AND* entre *in_val* e *in_ack*; (c) na presença de falhas do tipo *wired-OR* entre *in_val* e *in_ack*.

RESUMO DO CAPÍTULO 3

O capítulo 3 apresentou métodos funcionais de teste das interconexões de redes-em-chip. Inicialmente foi proposto um teste funcional com o objetivo de testar as interconexões de dados da NoC. Este método de teste baseou-se no envio de mensagens através da rede utilizando a NoC em modo de operação normal. Os vetores de teste são compostos por uma seqüência de detecção e falhas de interconexão (*walking one*).

Em seguida este método foi estendido, atendendo, também, ao teste dos sinais de controle de handshake e do pacote. Para isso, foram propostas alterações na forma como os pacotes que carregam os vetores de teste são enviados na rede. O pacote de teste também foi ligeiramente alterado para aumentar a cobertura de falhas do método.

Como resultado, os métodos apresentaram cobertura de falhas acima de 99% para falhas de curto-circuito do tipo *AND* e do tipo *OR* (*wired-AND* e *wired-OR*)

4 ESTRUTURAS DE TESTE (CIRCUITOS DE BIST)

Nas seções anteriores, as metodologias de teste descritas utilizaram estruturas capazes de gerar os vetores de teste, enviar estes vetores em forma de pacotes na rede, receber os pacotes de teste e analisar os resultados. Este tipo de circuito, quando implementado dentro do chip, permite ao sistema (ou partes dele) se auto-testar, realizando, assim, uma estratégia de BIST (*Built In Self Test*).

Nos ambientes de simulação das seções anteriores, tais estruturas de teste foram apresentadas como circuitos descritos em linguagem VHDL com nível de abstração comportamental. Contudo, a fim de realizar uma análise mais detalhada do impacto dessas estruturas no chip como um todo, os circuitos foram descritos em linguagem VHDL com nível de abstração RTL, de forma que pudessem ser sintetizados.

Pequenas alterações foram necessárias às estruturas de teste a fim de aplicar as duas metodologias de teste descritas em 3.1 (com e sem a inclusão dos sinais de controle no modelo de falhas).

Para implementar as estratégias de teste, geradores de vetores de teste (TDGs – Test Data Generators) e analisadores de resposta (TRAs – Test Response Analyzers) foram incluídos na lógica que conecta os núcleos de IP à rede (interfaces de rede). Um núcleo programável pode, também, fazer esta tarefa, contudo, como nem todos os núcleos são programáveis e a interface de rede é projetada de acordo com o protocolo da rede, a inclusão de estruturas de teste nas interfaces de rede se apresenta como uma solução mais genérica.

4.1 TDG – Características

Os TDGs têm por característica a geração dos vetores de teste e a implementação da política de *handshake* necessária para fazer o envio de mensagens na rede. Isto foi feito através de uma máquina de estados presente em cada TDG. Esta máquina de estados é descrita em linhas gerais na tabela 4.1 para o método de teste descrito em 3.1 (sem os sinais de controle):

Tabela 4. 1 Máquina de estados do TDG.

Estado	**Condição**	**Próximo Estado**	**Sinais (controle de *handshake* – *val*)**	**Sinais (Dados)**	**Comentários**
E1		E2	0	0000000000	Estado de inicialização de

					teste.
E2	(ack = 1) (ack = 0)	E3 E2	1	Cabeçalho	Estado de envio do cabeçalho
E3	(ack = 1) E (Contador = z1) Caso contrário	E4 E3	1	0000000000	Estado de envio do primeiro conjunto de *flits* com conteúdo zero antes dos *flits* de *payload*.
E4	(ack = 1) E (Contador = *payload*) Caso contrário	E5 E4	1	*Payload*	Nesse estado, o valor dos vetores de teste é enviado para a rede.
E5	(ack = 1) E (Contador = z2) Caso contrário	E6 E5	1	0000000000	Estado de envio do conjunto de *flits* com conteúdo zero após os *flits* de *payload*.
E6	(ack = 1) Caso contrário	E1 E6	1	Terminador	Estado de envio do *flit* de terminador.

4.2 TDG – Configuração

Basicamente, o TDG precisa gerar um *flit* de cabeçalho, alguns *flits* com todos os bits em zero antes dos *flits* de *payload*, os *flits* de *payload*, e uma nova série de *flits* com conteúdo zero seguidos de um terminador. Isto é mostrado em detalhes na seção 3.1, para a análise mais simplificada (sem sinais de controle) e na seção 3.2, para a análise que inclui os sinais de controle.

Alguns desses valores devem ser alterados para cada rodada de teste, já que o caminho que deve ser feito pela mensagem depende da posição do nó na rede. Embora para uma NoC “mesh” de tamanho 2x2 todos os pacotes possuam endereços de destino

fixos, para NoCs maiores o endereço deve ser variável, para atender às diferentes configurações de cada rodada de teste. Assim, algumas informações que devem ser configuradas em cada TDG:

- O atraso que deve ser somado a z_1 antes do TDG enviar um pacote de teste (apenas para a análise completa – 3.2),
- O cabeçalho,
- O número de *flits* com todos os *bits* em zero lógico antes do *payload* do pacote de teste (z_2 para a análise simplificada e z_4 para a análise completa),
- O número de *flits* com todos os *bits* em zero lógico depois do *payload* do pacote de teste.

Dados como a largura dos canais de comunicação da rede (w), o *flit* terminador, e o número de ciclos que um *flit* de *payload* leva para atravessar a rede (z_3 conforme definido antes) são constantes, sendo armazenado internamente nos circuitos de teste. As estruturas de teste (TDGs) são configuradas pelo ATE através de cadeias *scan*, sendo que diferentes implementações para esta conexão são apresentadas nos capítulos seguintes.

Para exemplificar a configuração de um TDG, assume-se a configuração mostrada na figura 4.1, onde quatro rodadas de teste são realizadas para uma NoC de tamanho 4x4. Nesse caso, o TDG incluído na interface de rede ligada ao roteador circulado na figura 4.1(a, b, c, d) deve gerar quatro cabeçalhos diferentes, um para cada rodada de teste, devido à sua posição na rede. A posição deste TDG na rede faz com que ele participe das quatro rodadas de teste necessitando quatro cabeçalhos diferentes para o envio das mensagens de teste. Isto justifica a necessidade de que a informação de cabeçalho seja configurável.

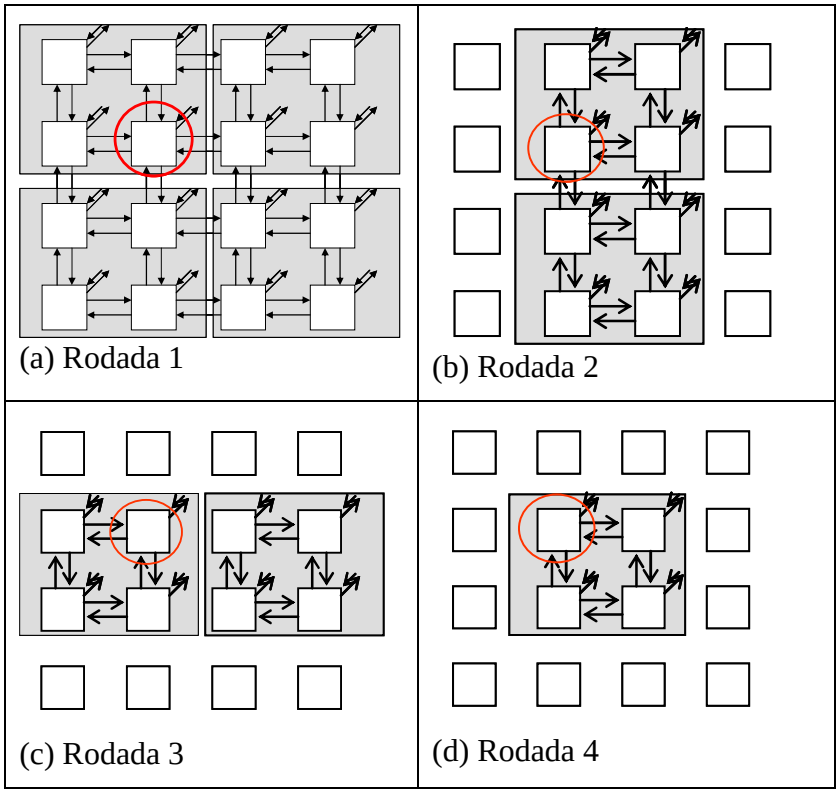
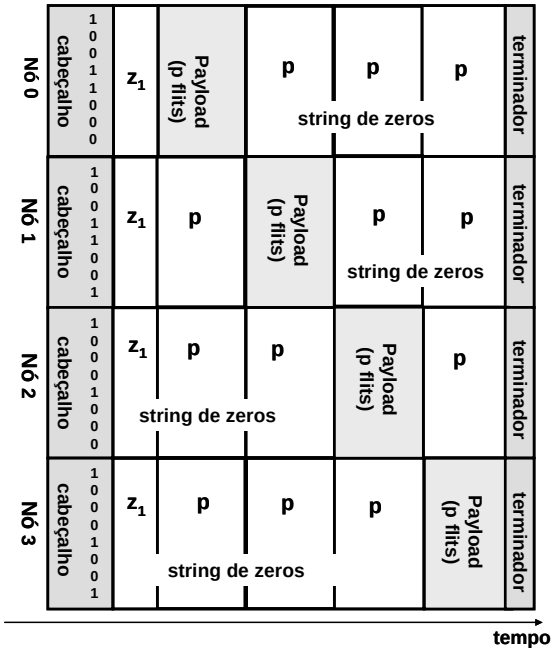
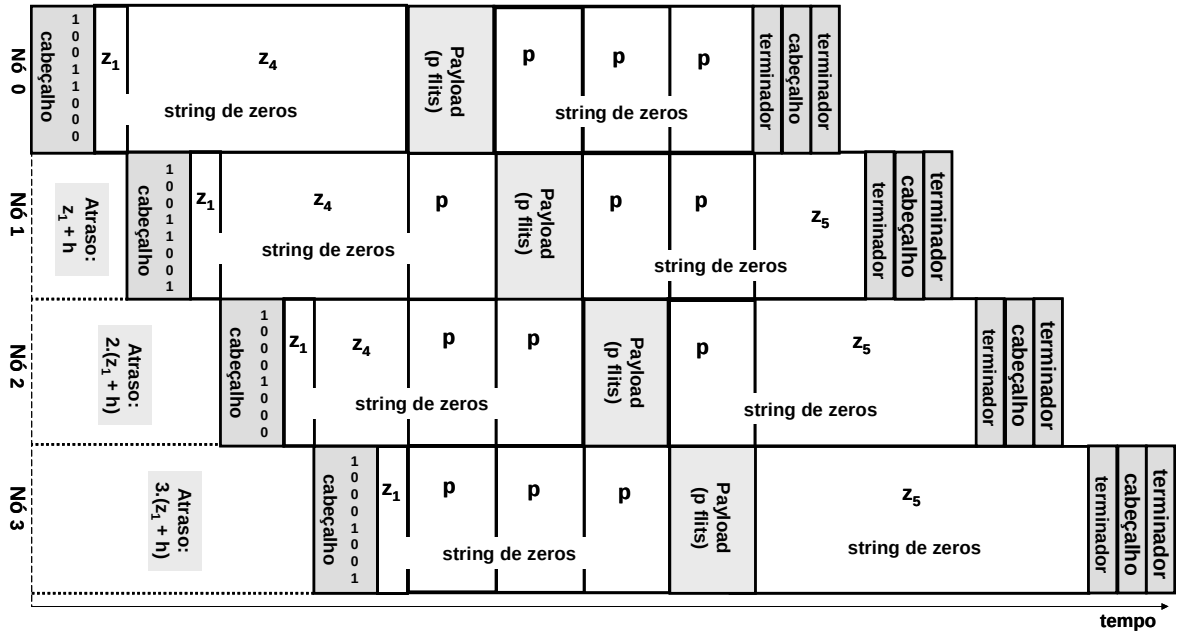


Figura 4. 1: Rodadas de teste para uma NoC de tamanho 4x4.

Tomando como exemplo a rodada de teste da figura 4.1(a), o TDG ligado ao roteador circulado deve enviar o pacote de teste ao roteador no localizado no canto suerior esquerdo. Assumindo-se o formato do pacote de teste para falhas nas interconexões desconsiderando-se os sinais de controle (3.1), nota-se que o primeiro conjunto de *flits* com todos os *bits* em zero z_1 apresenta um valor fixo que pode ser armazenado internamente no TDG. O segundo conjunto de *flits* contendo todos os bits em zero (z_2 dado pela equação 4), contudo, depende da posição do roteador considerando-se sua vizinhança de teste (tamanho 2x2). No caso da figura 4.1(a), essa posição (dada por *i*) é a posição 3. Observa-se que esta posição muda para cada rodada de teste necessitando, conseqüentemente, que este valor seja configurável. O mesmo vale para o conjunto de *flits* com conteúdo zero que seguem os *flits* de *payload*. Este conjunto de *flits* é complementar a z_2.A figura 4.2(a) apresenta os diferentes pacotes de teste enviados através da rede pelo método apresentado em 3.1.



(a)



(b)

Figura 4. 2: Envio dos pacotes na rede em função do tempo. (a) Método de teste sem inclusão dos sinais de controle (3.1); (b) Método de teste incluindo os sinais de controle (3.2).

No caso de considerarmos o modelo de falhas apresentado em 3.2, incluindo os sinais de controle, o pacote enviado pelo TDG, bem como a forma como ocorre o envio é alterada. Nesse caso, a primeira informação que deve ser configurada no TDG, além do cabeçalho, é o atraso de envio da mensagem representado na figura 4.2(b). Em seguida, a quantidade de *flits* com conteúdo zero antes do *payload* e depois do *payload* também devem ser configurados no TDG. Na figura 4.2(b), os nós representam os TDGs, que apresentam valores diferentes de acordo com sua posição em uma vizinhança de teste de tamanho 2x2 (TDG 0, TDG 1, TDG 2 TDG 3).

4.3 TRA – Características

O analisador de resposta possui uma estrutura similar ao TDG. O TRA aguarda pelo sinal de *val* do roteador, indicando que existe mensagem a ser recebida, para começar a verificar essa mensagem. Se o sinal de *val* não é recebido dentro de um certo intervalo de tempo, um sinal de *timeout* é habilitado, indicando um erro. Se nenhuma falha de *timeout* é detectada, a cada ciclo de relógio, um novo *flit* é lido e o TRA reproduz o valor esperado para comparação.

Assim como no TDG, uma máquina de estados é utilizada para implementar a política de *handshake*, para recepção das mensagem vindas da rede, e comparação com os valores gerados (valores esperados). Essa máquina de estados, em linhas gerais, é apresentada na tabela 4.2:

Tabela 4. 2: Máquina de estados do TRA.

Estado	**Condição**	**Próximo Estado**	**Sinais (controle de *handshake* –**	**Sinais gerados (para comparação)**	**Comentários**

			***ack*)**		
E1	(val = 0) E (Atraso > previsto)	E8	0	0000000000	Estado de inicialização de teste. Espera a recepção do primeiro *flit*.
	(val = 1)	E2			
	Caso Contrário	E1			
E2	(Val = 1)	E3	1	Cabeçalho	Recebe o primeiro *flit*. Compara com o cabeçalho esperado.
	(val = 0)	E2			
E3	(val = 1) E (Contador = z1)	E4	1	0000000000	Estado de recepção e comparação do primeiro conjunto de *flits* com conteúdo zero antes dos *flits* de *payload*.
	Caso contrário	E3			
E4	(val = 1) E (Contador = *payload*)	E5	1	*Payload*	Nesse estado, o valor dos vetores de teste são recebidos e comparados.
	Caso contrário	E4			
E5	(val = 1) E (Contador = z2)	E6	1	0000000000	Estado de recepção e comparação do conjunto de *flits* com conteúdo zero após os *flits* de *payload*.
	Caso contrário	E5			
E6	(Val = 1)	E7	1	Terminador	Estado de recepção e comparação do *flit* terminador.
	Caso contrário	E6			

E7	(Erro = 1)	E9	0	-	Análise de erro.
	Caso contrário	E1			
E8		E8	0	-	Indicação de erros de *timeout*.
E9		E9	0	-	Indicação de erros de *payload*.

A fim de possibilitar a localização das falhas em nível de interconexão, conforme mencionado no capítulo 5, dois contadores devem ser colocados em cada TRA. Esses contadores indicam o número do *flit* que é analisado. Assim que a primeira discrepância entre o *flit* recebido e o valor esperado é detectada, o primeiro contador é parado, indicando, assim, o número do primeiro *flit* com erro. O segundo contador é parado no caso de um segundo *flit* com erro ser recebido pelo TRA. Ao final de cada ciclo de teste, o ATE pode acessar os valores armazenados nos contadores através de uma cadeia de *scan*.

Sabendo-se o número do *flit* que apresenta erro, sabe-se qual fio apresentou o erro, já que cada *flit* é responsável pela detecção de um subconjunto de interconexões (todas as interconexões bit 0 dos canais, por exemplo).

4.4 TRA – Configurações

Da mesma forma que no TDG, para gerar os dados corretos para comparação com os *flits* recebidos através da rede, alguns valores devem ser configurados nos circuitos de teste, bem como algumas constantes devem ser definidas. As constantes a serem declaradas envolvem o cabeçalho modificado da mensagem (modificado após passagem nos roteadores), terminador, z_3 (o número de ciclos que um *flit* de *payload* leva para atravessar a rede), e *w*. Quanto aos *bits* configuráveis, são eles:

- O número de *flits* com todos os *bits* em zero antes do *payload* do pacote de teste (z_2 para a análise simplificada e z_4 para a análise completa),
- O atraso que deve ser somado a z_1 antes do TDG enviar um pacote de teste (apenas para a análise completa – 3.1.2),
- O número de *flits* com conteúdo zero lógico depois do *payload* do pacote de teste.

O ATE configura cada TRA com um tempo de espera máximo específico para a recepção do pacote (no caso do método de teste incluindo os sinais de controle). Por exemplo, o TRA 0 no nó 0 é programado com um tempo de espera baseado no atraso do pacote enviado pelo nó 3 e na latência da rede. Por outro lado, o TRA no nó 3 é programado com um tempo de espera baseado no atraso do pacote enviado pelo nó 0 e a latência da rede. Em suma, o TRA no nó *i* é programado com um tempo de espera de $(3-i).(z_1+h)+L$.

No TRA, além da configuração do circuito, se faz necessária a coleta das respostas obtidas na análise. Os sinais de erro (2 *bits*) e de o *flag* de *timeout* (1 *bit*), além dos dois

contadores de *flits* com erros (w *bits*) devem ser enviados para o ATE através de cadeias *scan*.

4.5 Configuração das Estruturas de Teste

Conforme visto, as estruturas de teste (TDGs e TRAs) precisam ser conectadas a um testador externo (ATE) para fazer a configuração para as diferentes rodadas de teste, assim como extrair os resultados. A seguir são apresentadas duas formas de conexão entre os circuitos de teste e o equipamento externo: utilizando-se cadeias *scan* e uma envoltória de teste baseado no padrão *boundary scan*. Essas implementações são avaliadas quanto ao impacto em área e tempo de teste.

4.6 Configuração das Estruturas de Teste Através de Cadeias *Scan*

A primeira opção de acesso ao teste é a conexão de todas as estruturas de teste através de uma única cadeia *scan*. O tamanho dessa cadeia é avaliado a seguir para o método de teste simplificado (3.1, sem inclusão dos sinais de controle). De acordo com o método de teste utilizado, o número máximo de *bits* necessários para representar a *string* de *flits* com conteúdo zero antes dos *flits* de *payload* e os *flits* com conteúdo zero depois dos *flits* de *payload* é o mesmo para os TDGs e TRAs e é dado por:

$$zeros\_antes\_do\_payload : \lceil \log_2(z_1 + 3 \cdot w + 3 \cdot w \cdot z_3) \rceil \qquad (14)$$

$$zeros\_depois\_do\_payload : \lceil \log_2(3 \cdot w + 3 \cdot w \cdot z_3) \rceil \qquad (15)$$

O número máximo de *flits* com conteúdo zero antes dos *flits* de *payload* é dado por $z_1 + 3.p$, onde $p = w.(1 + z_3)$, da Equação 14. Já o número máximo de *flits* com conteúdo zero depois dos *flits* de *payload* é dado por $3.p$, ou $3.(w + z_3)$.

Para a configuração do TDG, os seguintes *bits* adicionais são necessários ao registrador *scan*: *start_TDG* (1 *bit*) e cabeçalho (w+2 *bits*). Para a configuração do TRA, *bits* adicionais também são necessários ao registrador de *scan*, são eles: *start_TRA* (1 *bit*), *flag* de *timeout* (1 *bit*), e *flag* de erro (1 *bit*). Nesta análise não foram incluídos no TRA os contadores utilizados na indicação do *flit* afetado pela falha. Estes contadores implicariam em *bits* adicionais na cadeia *scan* dos TRAs.

As equações 16 e 17 representam os tamanhos (como função de w) das cadeias *scan* dos TDGs e dos TRAs, assumindo $z1$=9, $z3$=4, h=1, and L=11, conforme é aplicado à NoC estudo de caso.

$$sc_{TDG} = 3 + w + \lceil \log_2(9 + 15 \cdot w) \rceil + \lceil \log_2(15 \cdot w) \rceil \qquad (16)$$

$$sc_{TRA} = 3 + \lceil \log_2(9 + 15 \cdot w) \rceil + \lceil \log_2(15 \cdot w) \rceil \qquad (17)$$

Para uma cadeia *scan* única, o tempo total de aplicação do teste das interconexões da NoC é dado por:

$\boxed{T = 2 \cdot m^2 \cdot sc + C}$, para m=2 (18)

$\boxed{T = (tr + 1) \cdot (m^2 \cdot sc) + tr \cdot C}$, para m>2 (19)

Onde *TR = 4* (número de rodadas de teste), m^2 é o número de núcleos em uma NoC de tamanho *mxm*, *sc* é o tamanho do registrador usado para a configuração do TDG e do TRA (sc_{TDG} + sc_{TRA}), e *C* é o número de ciclos de relógio de teste requeridos pela seqüência de teste (no estudo de caso 182 ciclos de relógio).

O termo (tr + 1) da Equação 19 representa o número de vezes que ocorrem as configurações de teste e a retirada de dados de resposta. Assim que ocorre uma configuração através do deslocamento dos dados na cadeia de registradores, os dados de resposta são deslocados para fora da cadeia.

4.7 Configuração das Estruturas de Teste Através de uma Envoltória de Teste

Nessa seção, o uso de uma envoltória de teste baseado no padrão *boundary scan* é apresentado para configuração das estruturas de teste. Os *bits* de configuração dessas estruturas (TDGs e TRAs) são colocados nos registradores de *boundary scan* da envoltória, conforme mostrado na figura 4.3. TDI e TDO são portas de entrada e saída seriais para as instruções de teste, para os *bits* de configuração e para retirada dos dados de resposta. A envoltória de teste é compatível com o padrão de teste *Boundary Scan* (IEEE Std. 1149.1) e, além das portas de entrada e saída seriais, possui outras 3 portas de acesso ao teste – TMS, TCK e TRST, que conectam a envoltória de teste ao mundo externo. O procedimento para configurar os TDGs e os TRAs consiste em carregar as instruções no registrador de instruções e carregar os dados de configuração no registrador de *boundary scan*.

Além disso, um registrador de *bypass* contendo um único bit é utilizado para passar os dados através dos TDGs e dos TRAs que compõem a cadeia *scan*, que não estão em uso para uma rodada de teste específica (considerando-se uma NoC de tamanho maior que 2x2).

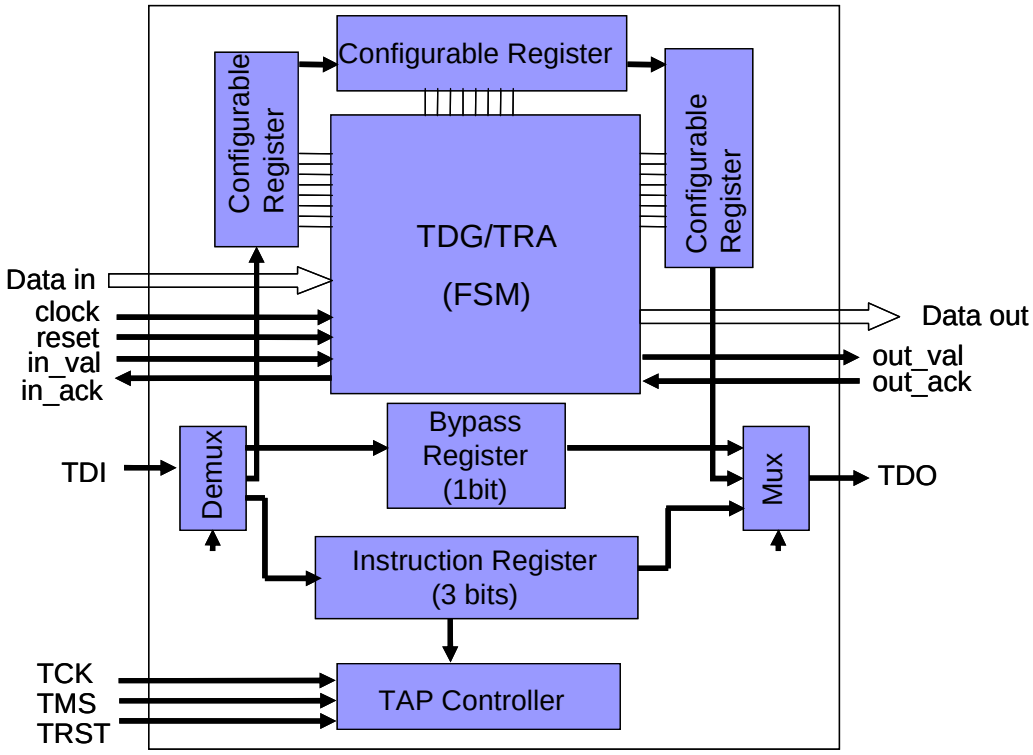


Figura 4. 3: Envoltórias de teste proposto para facilitar o acesso ao teste.

Os sinais de controle desses registradores são administrados pelo Controlador TAP (conforme definido no padrão). Após a ativação do *reset* de teste (TRST), o Controlador TAP é colocado no estado *test logic reset*. Na configuração do TRA, por exemplo, o ATE aplica os sinais de TMS e TCK ao Controlador TAP, mandando-o ao estado *select IR* e então *capture IR*. Nesse estado os *flags* de *timeout* e erro do teste previamente aplicado são carregados nos dois bits mais significativos do registrador de instruções. O ATE, então, manda a máquina de estados do Controlador TAP ao estado *shift IR* que fica nesse estado até que todas as respostas do teste sejam deslocadas para fora da cadeia de registradores e todos os bits das próximas instruções sejam carregados.

Quatro instruções de teste foram implementadas na envoltória proposta: SAMPLE/PRELOAD (obrigatório, mas não apresenta uso nessa aplicação de teste), INTEST (aplica os bits de configuração de teste aos TDGs e TRAs), BYPASS (reduz o tamanho da cadeia de registradores quando TDGs e TRAs não são parte de uma determinada rodada de teste), e SCAN_TEST (torna possível o teste interno das estruturas de teste TDG e TRA).

As instruções de teste se tornam ativas no estado *update IR*. Depois de carregar a instrução, o ATE aplica os sinais necessários à máquina de estados do Controlador TAP para mandá-la ao estado *select DR*. O Controlador TAP é enviado, então, ao estado *shift DR*, onde os bits de configuração são deslocados para dentro da cadeia de registradores *boundary scan*.

O registrador selecionado pelo Controlador TAP depende da instrução programada. No caso das instruções INTEST e SCAN_TEST, o registrador de *boundary scan* é selecionado. O registrador de *bypass* é selecionado na instrução BYPASS e o registrador da cadeia scan interna é selecionado junto com o registrador de *boundary scan* pela instrução SCAN_TEST.

Como na envoltória de *boundary scan* os *flags* de *timeout* e erro de *payload* são colocados dentro do registrador de instruções (IR), as equações 14, 15 e 16 ainda são aplicadas, porém a equação 17 muda para (assumindo *z1*=9 e *z3*=4, conforme o estudo de caso):

$$\boxed{sc_{TRA} = 1 + \lceil \log_2(9 + 15 \cdot w) \rceil + \lceil \log_2(15 \cdot w) \rceil} \qquad (20)$$

Para calcular o tempo de aplicação total de teste imposto por esta metodologia, deve-se levar em conta a redução do tamanho da cadeia *scan* devido à instrução de BYPASS do padrão *boundary scan*. O número de bits reduzidos devido ao *bypass* em cada rodada de teste para uma NoC mxm é dado pelas equações 21, 22 e 23.

Para (m = 2):

$$\boxed{Bypass = 0} \qquad (21)$$

Para (m > 2) e (m par):

$$Bypass = \begin{cases} 0, & Rodada \quad 1 \\ 2 \cdot m, & Rodada \quad 2 \\ 2 \cdot m, & Rodada \quad 3 \\ 2 \cdot m + 2 \cdot (m-2), & Rodada \quad 4 \end{cases} \qquad (22)$$

Para (m > 2) e (m ímpar):

Bypass = (2.m - 1) Todas as rodadas (23)

Então,

$$\text{Tempo\_de\_config} = 4 + 2.m^2.IR + 4 + [m^2 \,.\, sc - 2 \,.\, bypass \,.\, (sc - 1)] + 3 \qquad (24)$$

Onde *IR* representa o número de bits do registrador de instrução (no caso IR = 4) e *sc* representa o tamanho da cadeia dos registradores utilizados para a configuração dos TDGs e dos TRAs (sc_{TDG} + sc_{TRA}), de acordo com as Equações (16 e 20). Os valores constantes na Equação 24 representam o número de ciclos de relógio para ir de um estado a outro na máquina de estados do Controlador TAP. O termo $2.m^2.IR$ representa o tamanho total da cadeia de *scan* associada a todos os registradores de instrução dos TDGs e dos TRAs.

Finalmente, o tempo total de teste é dado por:

$$\boxed{T = Tempo\_de\_config + C + (3 + 2 \cdot m^2 \cdot IR)}, \text{ para m=2} \qquad (25)$$

$$\boxed{T = tr \cdot (Tempo\_de\_config + C) + (3 + 2 \cdot m^2 \cdot IR)}, \text{ para m>2} \qquad (26)$$

Onde *tr=4* (número de rodadas de teste) e C é o número de ciclos de relógio necessários para a seqüência de teste (no caso, 182 ciclos de relógio).

Se as cadeias de *scan* forem separadas para a configuração dos TDGs e dos TRAs, o tempo total de teste pode ser calculado eliminado o fator multiplicativo 2 do termo m^2.IR das Equações 24, 25 e 26, e considerando sc na Equação 24, como o maior tamanho de cadeia *scan* entre os TDGs e os TRAs, isto é, o número de bits do registrador de configuração dos TDGs, dado na Equação 16.

4.8 Teste dos Circuitos de BIST

O teste funcional dos módulos de teste resultou em uma cobertura de falhas de 80% para o TDG e apenas 50% para o TRA. Além disso, uma falha não detectada em um desses módulos, principalmente no TRA, pode não só mascarar algumas falhas nas interconexões da rede (por exemplo, uma falha do tipo *stuck-at-0* no sinal de erro do TRA) como também pode indicar uma falha inexistente (por exemplo, uma falha do tipo *stuck-at-1* no sinal de erro do TRA). Logo, é assumido que os TDGs e os TRAs são testados a priori, juntamente com a interface de rede, usando um método baseado em *scan*. Como o número de *flip flops* nessas lógicas é muito reduzido (83 *flip flops* para TDG e 79 para TRA), eles também podem ser testados usando uma cadeia de *scan* completa para evitar qualquer efeito de mascaramento de falhas durante o teste.

4.9 Experimentos

Essas duas estruturas de teste foram implementadas em VHDL e sintetizadas utilizando-se a ferramenta Encounter RTL Compiler da Cadence usando uma biblioteca de tecnologia de 0.35 μm. Os resultados são mostrados na tabela 4.3.

Os blocos de TDG e TRA representam 49% da área de um roteador da rede (RASoC com 1688 gates). Contudo, os roteadores representam uma pequena parcela na área tal do sistema, uma vez que os núcleos são responsáveis pela maior parte da ocupação do *chip*. Assim, pode-se dizer que o sobrecusto de área das estruturas de teste não causa grande impacto na área do sistema como um todo, tendo em vista os benefícios obtidos com a utilização dos TDGs e dos TRAs.

Tabela 4. 3: Resultados de área das estruturas de teste.

Circuito	**TDG**	**TRA (sem contadores para localização de falhas)**	**TRA (com contadores para localização de falhas)**	**RASoC (5 portas, buffers de 3 posições, largura do canal de dados = 8)**
Tamanho (número de células da biblioteca)	341	401	494	1688

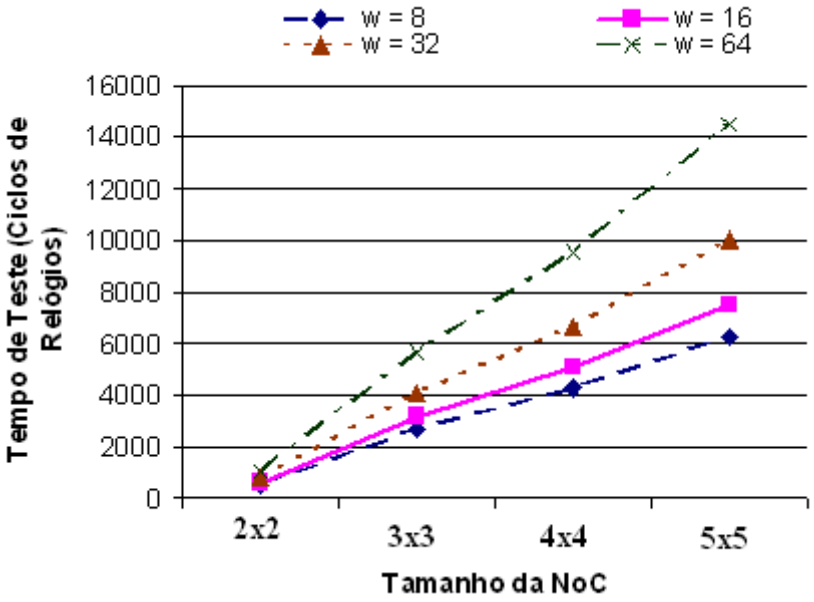
A tabela 4.4 representa os resultado de área (em número de células da biblioteca) das estruturas de teste com as envoltórias de teste utilizadas para a configuração dos TDGs e dos TRAs. Nessa tabela, é apresentada, também, a área do Controlador TAP.

Tabela 4. 4: Resultados de área para as estruturas de teste acrescidas das envoltórias de teste.

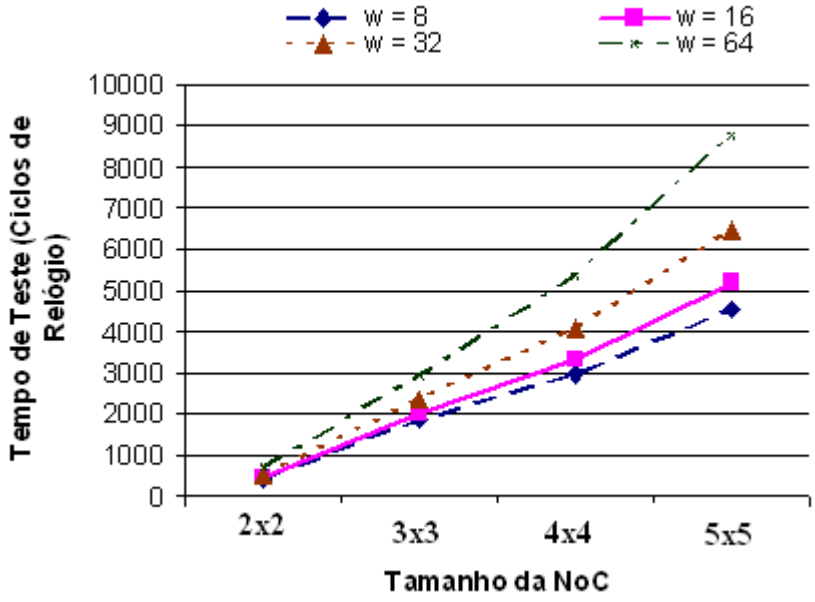
Circuito	TDG + Envoltória de Teste	TRA + Envoltória de Teste	Controlador TAP
Tamanho (número de células da biblioteca)	680	684	116

O tempo de teste também foi avaliado para as três estratégias de configuração propostas e implementadas: cadeia *scan* única; uso de envoltória de teste e cadeia de registradores única (a mesma para os TDGs e os TRAs); e uso de envoltória de teste e cadeias de registradores separadas para os TDGs e os TRAs. O tempo de teste T foi calculado utilizando as equações 18, 19, 25 e 26, para NoCs mxm ($2 \leq m \leq 5$), considerando-se interconexões de dados com w=8, 16, 32 e 64, IR = 4 *bits*, C = 182 ciclos de relógio e tr = 4 rodadas de teste (esse valor é constante), para as três implementações mencionadas.

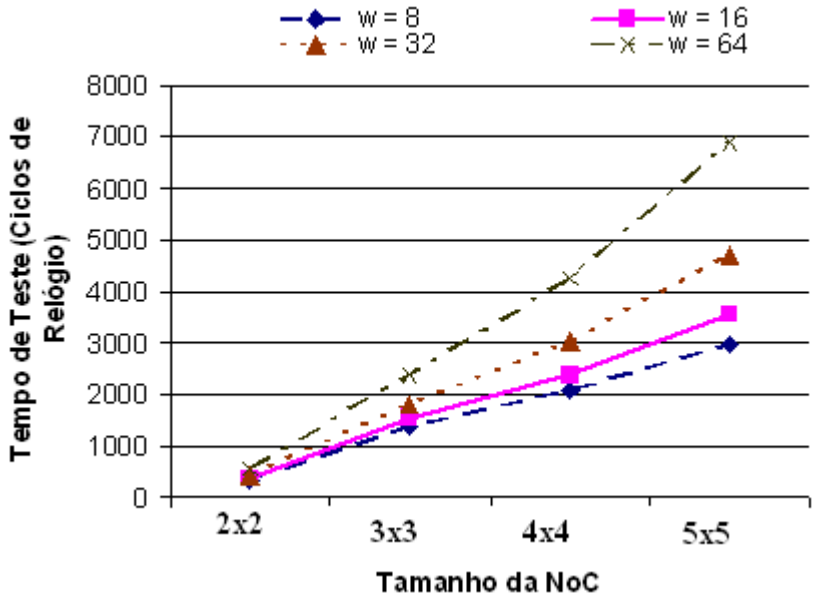
As curvas com os resultados são mostradas na figura 4.4. Nessa figura, fica claro que para todos os valores de w e m, a envoltória de teste aparece como uma solução melhor devido à redução no tempo de configuração de teste obtido principalmente devido à instrução de BYPASS. Os melhores resultados são obtidos com o uso da envoltória de teste e cadeias duplas (uma para os TDGs e outra para os TRAs).



(a)



(b)



(c)

Figura 4. 4: Resultados de tempo total de teste para as diferentes implementações. (a) cadeia *scan* única; (b) uso de envoltória de teste e cadeia única de registradores; (c) uso de envoltória de teste e cadeia dupla de registradores (uma cadeia para os TDGs e outra cadeia para os TRAs).

Por fim na figura 4.5 é mostrada uma comparação entre os três métodos apresentados quanto ao tempo de teste em função da largura do canal. É assumida uma NoC de tamanho 4x4.

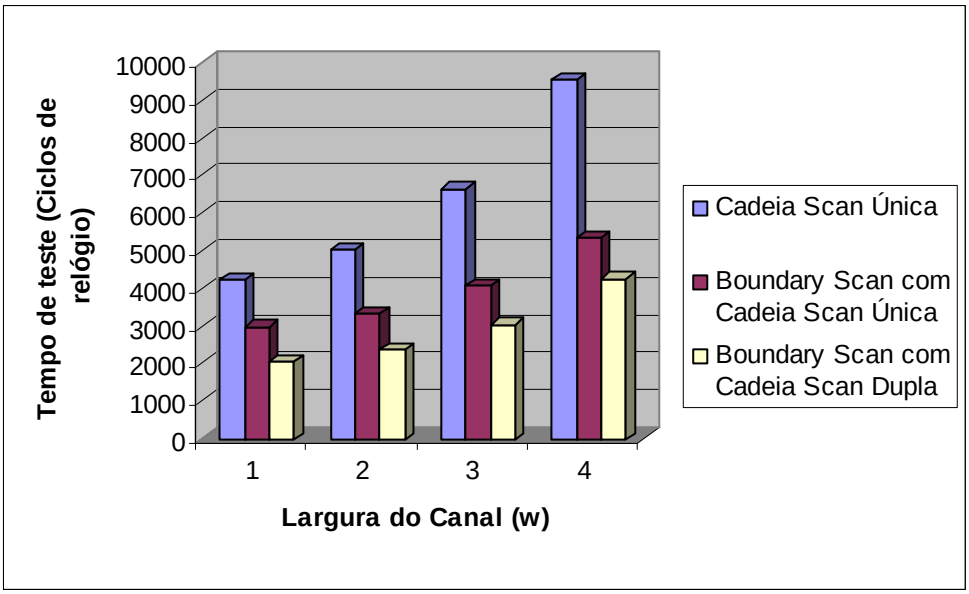


Figura 4. 5: Comparação do tempo de teste entre os diferentes métodos propostos em função da largura do canal para uma NoC de tamanho 4x4.

RESUMO DO CAPÍTULO 4

Nesse capítulo, foram apresentados os circuitos de BIST necessários à estratégia de teste proposta. Estes circuitos são responsáveis pela geração dos vetores de teste e sua aplicação à NoC (TDGs), bem como pelo recebimento dos pacotes de teste e comparação com valores esperados (TRAs).

Um estudo sobre os diferentes métodos de acesso ao teste visando a configuração desses circuitos de BIST encerra o capítulo. Neste estudo são propostos métodos baseados apenas em cadeias *scan*, e métodos baseados no uso de uma envoltória de teste.

5 CAPACIDADE DE LOCALIZAÇÃO DE FALHAS

Utilizando-se o método de teste das interconexões de uma rede em chip apresentado na seção 3.1, foi realizado um estudo sobre a capacidade de localização das falhas detectadas. A partir dos resultados obtidos, foram propostas formas de aumentar essa capacidade, conforme descrito nas seções seguintes.

A localização das falhas pode permitir uma reparação da rede através de mudança no roteamento da mensagem, ou através do uso de interconexões redundantes para fim de tolerar a falha. Estas interconexões extras podem ser ligadas conforme a necessidade a fim de estabelecer um caminho sem falhas para a mensagem enviada através da rede.

5.1 Modelo de Falhas

O modelo de falhas utilizado foi o mesmo apresentado na seção 3.1 e compreende apenas as falhas nos bits de dados das interconexões da NoC. As falhas na lógica dos roteadores não são consideradas. A NoC básica do método apresentada em 3.1 é mostrada novamente na figura 5.1.

De acordo com o modelo de falhas, existem 128 interconexões de dados (assumindo-se a NoC configurada com 8 bits de dados), distribuídas em 16 canais de comunicação, que podem entrar em curto-circuito.

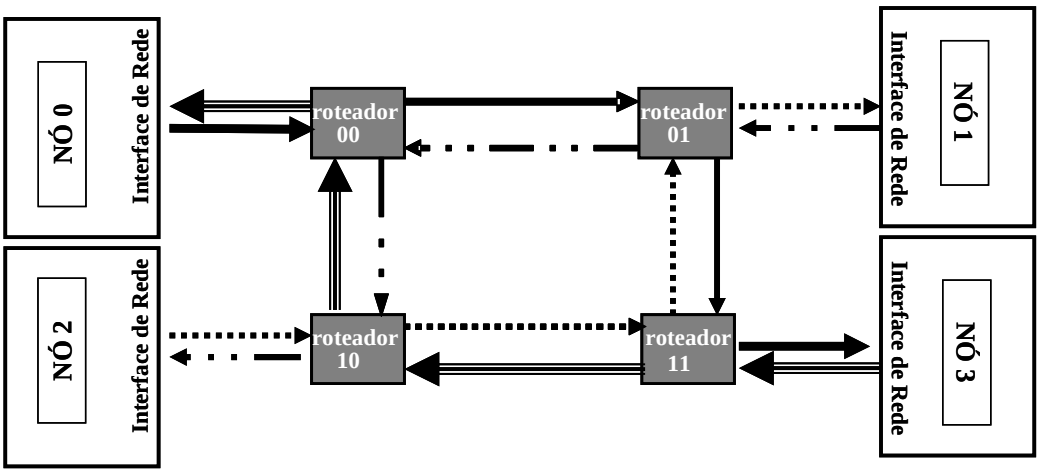


Figura 5. 1: Vizinhança de teste do método apresentado em 3.1.

A fim de simplificar a análise, podemos considerar as falhas em nível de canal de comunicação. Para isso, assume-se que existe uma falha em um canal de comunicação, quando existe pelo menos uma interconexão pertencente a este canal com falha. Nesse caso, pode-se observar que existem 136 casos possíveis de falhas em canais. Desses 136 casos, 120 estão relacionados a falhas em canais distintos, ou seja, falhas de curto

circuito entre interconexões pertencentes a canais distintos, e 16 falhas internas aos canais de comunicação, ou seja, falhas de curto circuito entre duas interconexões pertencentes a um mesmo canal de comunicação. Exemplos de falhas em nível de canal são apresentados a seguir:

- nó00-ao-roteador00 → nó00-ao-roteador10 (canais distintos),
- nó00-ao-roteador00 → nó00-ao-roteador11 (canais distintos),
- nó00-ao-roteador00 → nó00-ao-roteador01 (canais distintos),
- ...,
- nó00-ao-roteador00 → nó00-ao-roteador00 (interno ao canal),
- nó00-ao-roteador01 → nó00-ao-roteador01 (interno ao canal),
- nó00-ao-roteador11 → nó00-ao-roteador11 (interno ao canal),
- ...

As falhas em nível de canal compreendem, em cada caso, várias falhas em nível de interconexão. Por exemplo, para uma falha em nível de canal entre o canal nó00-ao-roteador00 e o canal nó00-ao-roteador10 as seguintes falhas de interconexão são compreendidas:

- nó00-ao-roteador00 (bit 0) → nó00-ao-roteador10 (bit 0),
- nó00-ao-roteador00 (bit 0) → nó00-ao-roteador10 (bit 1),
- ...,
- nó00-ao-roteador00 (bit 0) → nó00-ao-roteador10 (bit *w*);
- nó00-ao-roteador00 (bit 1) → nó00-ao-roteador10 (bit 0),
- nó00-ao-roteador00 (bit 1) → nó00-ao-roteador10 (bit 1),
- ...,
- nó00-ao-roteador00 (bit 1) → nó00-ao-roteador10 (bit *w*);
- ...,
- nó00-ao-roteador00 (bit *w*) → nó00-ao-roteador10 (bit 0),
- nó00-ao-roteador00 (bit *w*) → nó00-ao-roteador10 (bit 1),
- ...,
- nó00-ao-roteador00 (bit *w*) → nó00-ao-roteador10 (bit *w*);

Essas falhas em nível de interconexão representam todas as possíveis falhas de curto circuito para as interconexões de interesse e representam um total de 8.128 falhas.

Essa distinção entre falhas em nível de canal e falhas em nível de interconexão é feita com o objetivo de facilitar o método de localização das falhas.

5.2 NoC Utilizada – Estudo de Caso

A NoC utilizada como estudo de caso é igual à apresentada na seção 3.1 e consiste em uma NoC SoCIN de topologia direta grelha 2D de tamanho 2x2. Estruturas de teste foram ligadas à rede compondo um ambiente de simulação.

5.3 Método de Localização de Falhas

O método de localização de falhas proposto baseia-se na análise das sinalizações de erro por parte das estruturas de teste colocadas nos nós da rede e leva em consideração as falhas em nível de canal. As falhas em nível de canal permitem, a partir da análise das sinalizações de erro, fazer a localização dos canais que apresentam interconexões com falhas. As interconexões com falhas podem, então, ser identificadas através da seqüência de teste utilizada.

Para cada vetor de teste contido na mensagem de teste, um subconjunto de interconexões é testado como, por exemplo, todos as interconexões de bit 0 dos canais envolvidos no caminho da mensagem. Dessa forma, se os flits que contém os vetores de teste responsáveis por testar as interconexões de bit 0 apresentarem erro, eles podem ser marcados, identificando, assim, que há falhas em alguma das interconexões de bit 0. Para haver a localização da falha, é necessário, então, identificar o canal onde a interconexão de bit 0 ocorreu.

A figura 5.2 mostra um bit afetado por uma falha. A falha responsável pelo erro pode ter ocorrido no caminho “a”, “b” ou “c”. Dessa forma é possível identificar o bit que apresenta o erro, porém não é possível localizar a falha.

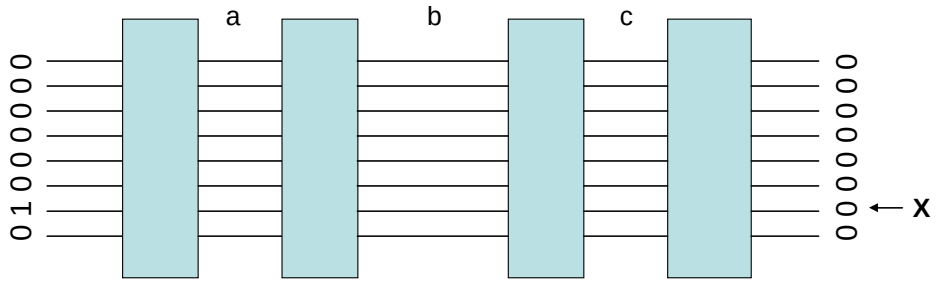


Figura 5. 2: Exemplo de erro em um bit do canal de comunicação. A falha pode ter ocorrido nos caminhos “a”, “b” ou “c”, porém não é possível localizar o exato local da falha.

5.4 Avaliação da Capacidade de Localização de Falhas do Método de Teste das Interconexões Apresentado na Seção 3.1

Para a NoC estudo de caso e com o método de teste proposto na seção 3.1, as sinalizações de erros possíveis são apresentados na tabela 5.1. Nota-se que, por causa do modelo de falhas definido (somente falhas simples de curto circuito são consideradas), no máximo duas estruturas de teste podem sinalizar o erro após uma rodada de teste, o que resulta em 10 possíveis combinações de sinalização de erro.

Se analisarmos as falhas em nível de interconexão, nota-se que o método de teste pode identificar através da seqüência de teste, um subconjunto de interconexões suspeitas de apresentar falhas. Este subconjunto é formado por bits específicos das interconexões de dados que se encontram no caminho da mensagem que causou uma

sinalização de erro. Para localizar a interconexão faltosa dentro desse subconjunto de interconexões, é necessário identificar o canal que apresenta a falha, logo, deve ser feita uma análise considerando falhas em nível de canal.

Usando-se a informação mostrada na tabela 5.2, é possível relacionar a detecção de erro com os canais com falha. Isso pode ser feito através de uma lista dos canais localizados no caminho da mensagem de teste que causa a sinalização de erro.

Por exemplo, se somente o nó11 (nó conectado ao roteador 11) sinaliza o erro, a falha deve estar em um dos seguintes canais: nó00-ao-roteador00, roteador00-ao-roteador01, roteador01-ao-roteador11, roteador11-ao-nó11. Estes canais representam os canais suspeitos *u* e *v* sendo que deve haver uma falha entre dois fios (*i* e *j*) dentro de um desses canais ($u = v$), ou entre dois fios (*i* e *j*) pertencentes a dois canais suspeitos diferentes ($u \neq v$). Assim existem 10 casos possíveis de canais com falha, conforme mostrado na tabela 5.2. Dentre os suspeitos não é possível identificar o responsável pela interconexão com falha, logo, a capacidade de localização das falhas do método de teste não permite a identificação do canal com falha, apresentando, no máximo, um subconjunto de canais suspeitos.

Tabela 5. 1: Combinação de sinalização de erros para o método descrito na seção 3.1.

Hipótese	**Estrutura de teste ligada ao nó da rede**			
	00	**01**	**10**	**11**
1	*Erro*	OK	OK	OK
2	OK	*Erro*	OK	OK
3	OK	OK	*Erro*	OK
4	OK	OK	OK	*Erro*
5	*Erro*	*Erro*	OK	OK
6	*Erro*	OK	*Erro*	OK
7	*Erro*	OK	OK	*Erro*
8	OK	*Erro*	*Erro*	OK
9	OK	*Erro*	OK	*Erro*
10	OK	OK	*Erro*	*Erro*

Estendendo esta análise para todas as falhas de *payload* para a NoC de estudo de caso, obtêm-se uma classificação para a localização da falha para o método apresentado na seção 3.1. As rodadas de teste que apresentam falhas de *timeout* não são consideradas na análise por representarem um caso particular. Esta classificação é apresentada na tabela 5.3. De acordo com essa tabela, para 65% das falhas possíveis, o número de canais suspeitos é de pelo menos 10.

Tabela 5. 2: Lista dos canais suspeitos de apresentarem falhas (falhas de curto circuito entre fio *i* do canal *u* e fio *j* do canal *v* para o exemplo citado no texto).

	Canal *u* **(contendo o fio *i*)**	**Canal *v*** **(contendo o fio *j*)**
1	nó00-ao-roteador00	nó00-ao-roteador 00
2	nó00-ao-roteador 00	roteador00-ao-roteador 01
3	Nó00-ao- roteador 00	roteador01-ao-roteador 11
4	Nó00-ao-roteador 00	roteador11-ao-nó11
5	roteador00-ao-roteador 01	roteador00-ao-roteador01
6	roteador00-ao–roteador 01	roteador01-ao-roteador11
7	roteador00-ao-roteador 01	roteador11-ao-nó11
8	roteador01-ao-roteador 11	roteador01-ao-roteador11
9	roteador01-ao-roteador 11	roteador11-ao-nó11
10	roteador11-ao-nó11	roteador11-ao-nó11

A análise apresentada na tabela 5.3 considera todas as falhas em nível de interconexão. Contudo, como não é possível identificar o canal que possui a interconexão faltosa, foram encontrados dois subconjuntos de falhas um com 10 canais suspeitos e outro com 16 canais suspeitos.

Nota-se que a mesma análise pode ser feita para falhas que acarretam erros de *timeout*, contudo, as falhas que provocam erros de *payload* já demonstram que o método possui uma baixa capacidade de localização de falhas. Dessa forma, foram consideradas alternativas para o teste funcional buscando além da alta cobertura de falhas, um aumento na capacidade de localização das falhas. Essas alternativas são apresentadas na próxima seção.

Tabela 5. 3: Classificação dos resultados de localização de falhas para o método de teste descrito na seção 3.1.

Erros de *Payload*		**Erros de *Timeout***	**Total**
10 pares de canais suspeitos de apresentar falhas	16 pares de canais suspeitos de apresentar falhas		
1043 (12,83%)	4285 (52,72%)	2800 (34,45%)	8128 (100%)

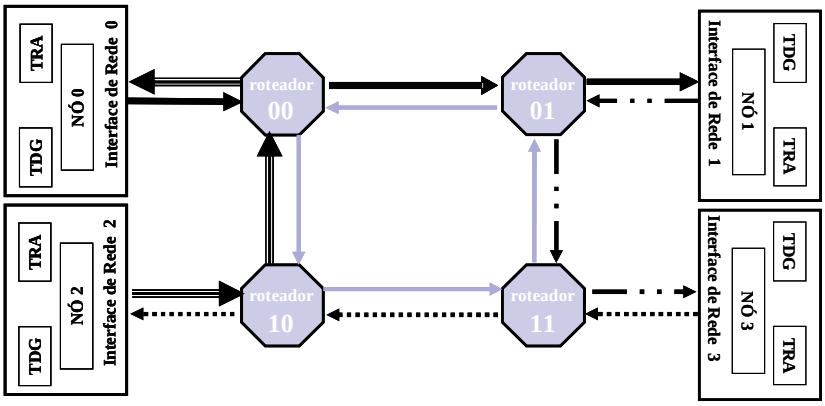
5.5 Método de Teste Modificado Visando a Localização de Falhas

A baixa capacidade de localização das falhas no método de teste funcional apresentado em 3.1 é uma conseqüência do caminho feito pela mensagem de teste através da rede. Visando reduzir o tempo de teste, o método descrito em 3.1 utiliza o menor número de ciclos de envio de mensagens na rede que obtenha 100% de detecção de falhas. Contudo, nesse método de teste, um pacote enviado de uma estrutura de teste (nó da rede) é analisado após dois *hops* através da NoC, o que acarreta no alto número

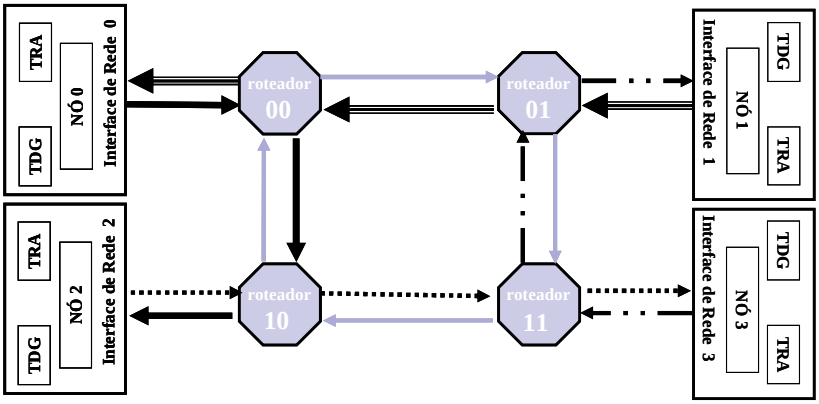
de canais suspeitos durante o método de localização das falhas, já que o caminho da mensagem envolve 4 canais. Assim, para aumentar a capacidade de localização de falhas, foram estabelecidos diferentes caminhos e formas de envio da mensagem de teste através da rede, de forma que menos *hops* fossem necessários do envio até a análise da mensagem e, conseqüentemente, menos canais fossem envolvidos nesse caminho.

O novo esquema de teste utiliza os mesmos vetores de teste da seção 3.1, mas utiliza dois ciclos de envio de mensagens na rede, conforme mostrado na figura 5.3. No primeiro ciclo, os pacotes são enviados no sentido horário do nó 00 ao nó adjacente, por exemplo: o nó 00 envia para o nó 01, o nó 01 envia para o nó 11, o nó 11envia para o nó 10, e o nó 10 envia para o nó 00, conforme a figura 5.3(a). Nesse primeiro ciclo, os canais de comunicação do roteador00-ao-roteador10, roteador10-ao-roteador11, roteador11-ao-roteador01 e roteador01-ao-roteador00 não são utilizados. É aproveitada a característica do roteador RASoC da SoCIN que mantém os canais inutilizados em nível lógico “0”. Esta característica é baseada no *reset/preset* dos *buffers* de entrada e saída dos roteadores e podem ser modificados facilmente, se necessário.

No segundo ciclo, os pacotes são enviados na direção oposta, por exemplo: o nó 00 envia para o nó 10, o nó 10 envia para o nó 11, o nó 11 envia para o nó 01, e o nó 01 envia para nó 00, conforme a figura 5.3 (b). Nesse segundo ciclo, os canais que foram utilizados no primeiro ciclo de envio não são utilizados. Dessa forma, após dois ciclos, todos os canais foram exercitados.



(a)



(b)

Figura 5. 3: Ciclos de envio de mensagem para o método de teste modificado visando localização de falhas. (a) Ciclo 1; (b) Ciclo 2.

Utilizando-se a metodologia de localização de falhas (descrita em 5.3), foi aplicado o esquema de teste modificado, com dois ciclos de envio de mensagens na rede. Novamente, são consideradas, para simplificar a análise em um primeiro momento, apenas as falhas que resultam em sinalizações de erro de *payload*, já que essas representam o maior conjunto de falhas. Uma análise similar pode ser feita para as falhas que resultam em erros de *timeout*.

A análise de localização de falhas é feita combinando as sinalizações de erro do primeiro ciclo de envio de mensagens com as sinalizações de erro apresentadas do segundo ciclo de envio e novamente serão consideradas as falhas em nível de canal para a localização dos canais com falha.

Exemplificando, se um erro é detectado pelo nó 01 no primeiro ciclo, existem 3 possíveis canais suspeitos: nó00-ao-roteador00, roteador00-ao-roteador01, e roteador01-ao-nó01. Se, no segundo ciclo, o nó 11 sinaliza um erro, três outros canais são suspeitos: nó10-ao-roteador10, roteador10-ao-roteador11 e roteador11-ao-nó11. Como os canais conectados aos nós não apresentam falhas em pelo menos um dos ciclos de envio, eles podem ser descartados como suspeitos de apresentarem as falhas, mantendo apenas os canais que ligam roteadores a roteadores como possíveis suspeitos de apresentarem falhas. Assim, para este exemplo, é possível dizer que existe uma falha de curto circuito (devido ao modelo de falhas adotado) entre fios nos canais de comunicação roteado00-ao-roteador01 e um fio do canal roteador10-ao-roteador11.

Na presença de falhas de curto-circuito aos pares, a aplicação da seqüência de teste *walking one* aos dois canais mencionados, causa um erro na checagem de um *flit* do *payload* pelo nó 01 no ciclo 1 e outro erro na checagem de um *flit* do *payload* pelo nó 11 no ciclo 2. Assim, para identificar os bits que tiveram seu valor lógico alterado, é necessário identificar qual foram os flits faltosos do pacote de teste.

Contudo, nem todos os casos podem ter as interconexões faltosas identificadas diretamente. Se, por exemplo, o nó 01 sinaliza um erro no ciclo 1, os canais de comunicação suspeitos são nó00-ao-roteador00, roteador00-ao-roteador01, e roteador01-ao-nó01. Se o nó 10 indica um erro no ciclo 2, a lista de canais suspeitos é, então, composta pelos seguintes canais: nó00-ao-roteador00, roteador00-ao- roteador10 e roteador 10-ao-nó10. Nesse caso, a sinalização de erro é associada a quatro possíveis pares de canais faltosos conforme apresentados na tabela 5.4. A tabela 5.5 apresenta a análise de localização de falhas em nível de canal para todas as sinalizações de erro de *payload*.

Tabela 5. 4: Canais suspeitos de falha para o exemplo apresentado.

Canais suspeitos
roteador00-ao-roteador01 roteador00-ao- roteador10
roteador00-ao- roteador10 nó00-ao- roteador00
roteador00-ao-roteador01 Nó00-to- roteador00
nó00-to-roteador00 nó00-to-roteador00

Da análise feita para localização de falhas levando-se em consideração as falhas que resultam em sinalizações de erro de *payload*, observam-se três possibilidades de relação entre a sinalização do erro e a falha que a ocasionou, conforme mostra a tabela 5.5. O primeiro grupo corresponde aos casos resolvidos (R), em que as falhas são localizadas totalmente correspondendo a um único par de canais suspeitos. O segundo grupo corresponde aos casos não resolvidos (N), onde a sinalização do erro se relaciona a pelo menos dois pares de canais suspeitos. O terceiro grupo corresponde às combinações de sinalização de erros que são impossíveis de ocorrer (I) para o modelo de falhas assumido. Das 136 possíveis falhas entre canais de comunicação, o grupo resolvido (76 combinações de sinalização de erro na tabela 5.5) corresponde a 76 casos e o grupo não resolvido (18 combinações de sinalização de falhas na tabela 5.5) corresponde a 60 pares de canais faltosos. Os 26 casos restantes de sinalizações de erro não são possíveis de ocorrer (I) nos ciclos 1 e 2 e, dessa forma, não estão relacionados a nenhuma falha possível. Para os casos não resolvidos, foram aplicadas novas técnicas apresentadas posteriormente neste trabalho.

Uma vez que o canal com falha é identificado, pode-se facilmente obter a interconexão com falha utilizando-se a informação da seqüência de teste aplicada conforme explicado anteriormente. Se fios a mais (extras) são colocados na NoC, é possível manter a rede funcionando utilizando-se estas interconexões no lugar da interconexões com falhas, sem necessitar de reconfiguração de algoritmos de roteamento da mensagem. Para isso, a identificação do fio com falha se faz necessária além da detecção desta no processo de teste da rede.

Tabela 5. 5: Possíveis combinações de sinalização de erro por parte das estruturas de teste após os dois ciclos de envio de mensagem de teste na rede (R = Resolvido, N = Não resolvido, I = Impossíveis de ocorrer para o modelo de falhas adotado).

	Ciclo 1 (nó #)											
Ciclo 2 (nó #)	**Erro**	**00**	**01**	**10**	**11**	**00 / 01**	**00 / 10**	**00 / 11**	**01 / 10**	**01 / 11**	**10 / 11**	**Não detectado**
	10 / 11	R	R	R	R	R	R	*I*	*I*	R	R	R
	01 / 11	R	R	R	R	R	R	*I*	*I*	R	R	R
	01 / 10	*I*	**N**	**N**	*I*	*I*	*I*	*I*	**N**	*I*	*I*	R
	00 / 11	**N**	*I*	*I*	**N**	*I*	*I*	**N**	*I*	*I*	*I*	R
	00 / 10	R	R	R	R	R	R	*I*	*I*	R	R	R
	00 / 01	R	R	R	R	R	R	*I*	*I*	R	R	R
	11	**N**	R	R	**N**	R	R	**N**	*I*	R	R	R
	10	R	**N**	**N**	R	R	R	*I*	**N**	R	R	R
	01	R	**N**	**N**	R	R	R	*I*	**N**	R	R	R
	00	**N**	R	R	**N**	R	R	**N**	*I*	R	R	R
	Não Detectado	R	R	R	R	R	R	R	R	R	R	I

Dentre os casos não resolvidos, foram identificados 3 grupos diferentes considerando-se o número de possíveis falhas associadas a uma sinalização de erro. O primeiro grupo foi associado com 4 canais suspeitos (é o caso do exemplo cujo resultado é apresentado na tabela 5.4), o segundo grupo foi associado com 3 canais suspeitos de apresentarem falha, e o terceiro grupo é associado com 2 canais suspeitos. Utilizando a mesma idéia inicial de alterar os caminhos da mensagem de teste na rede, foram implementados outros ciclos de envio de mensagem a fim de solucionar cada um dos casos.

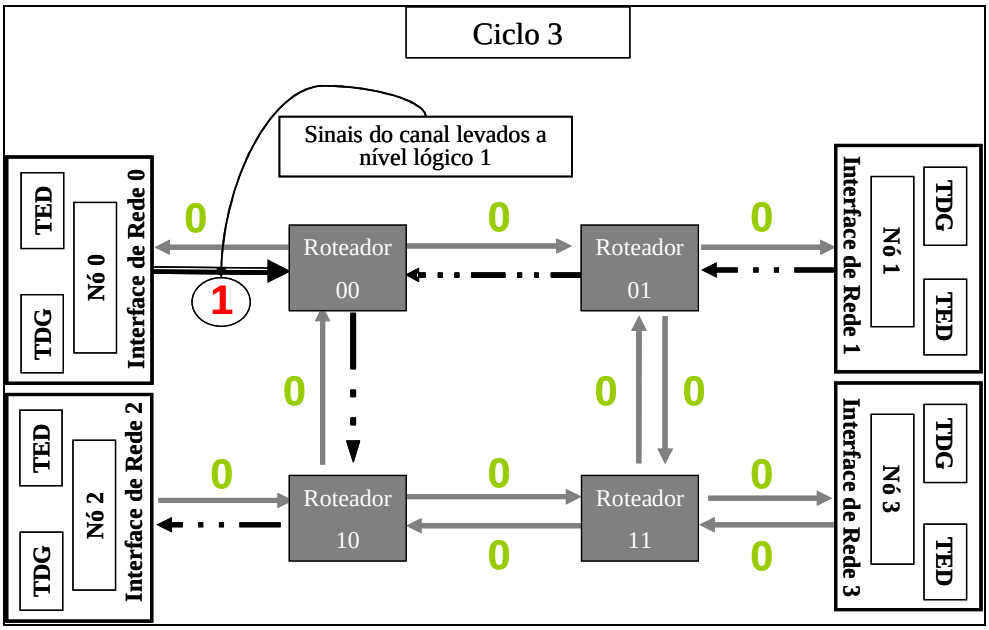
5.6 Extensão do Método de Localização de Falhas para Casos Não Resolvidos

A fim de aumentar a capacidade de localização das falhas ainda mais, foram estabelecidos novos ciclos de envio de mensagens de teste para atender aos casos não resolvidos.

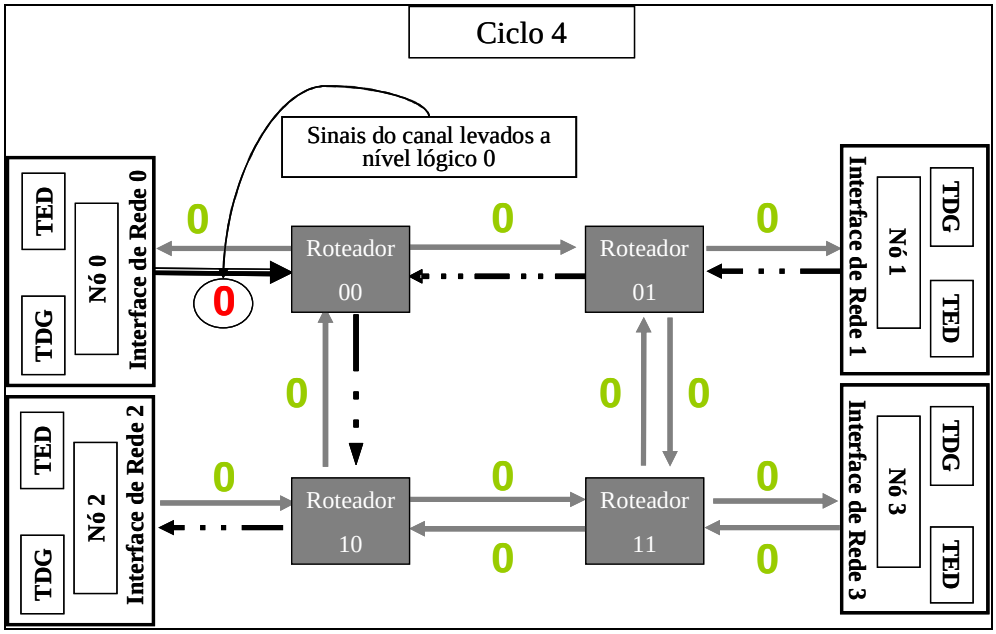
Os ciclos extras para o exemplo da tabela 5.4 são mostrados nas figuras 5.4(a), 5.4(b) e 5.4(c), respectivamente. Com os novos ciclos torna-se possível localizar todas os casos que mantinham-se não solucionados anteriormente, após os dois ciclos de teste. Os vetores de teste utilizados nos ciclos extras, consistem em pacotes com *flits* contendo todos os *bits* em zero (o suficiente para manter preenchido o caminho da mensagem com zeros), e um *flit* contendo todos os *bits* em “1” no meio do pacote. Esta seqüência de teste é capaz de detectar curtos-circuitos nos canais suspeitos e é consideravelmente menor que a seqüência de teste dos ciclos 1 e 2.

Cada ciclo extra exclui um suspeito da lista de suspeitos. Assim, para o subgrupo com 4 canais suspeitos de conter interconexões faltosas, três ciclos extras são necessários para localizar a falha. Para o subgrupo com 3 suspeitos, dois ciclos extras são necessários e para o subgrupo com 2 possibilidades de canais com falhas apenas um ciclo extra é necessário.

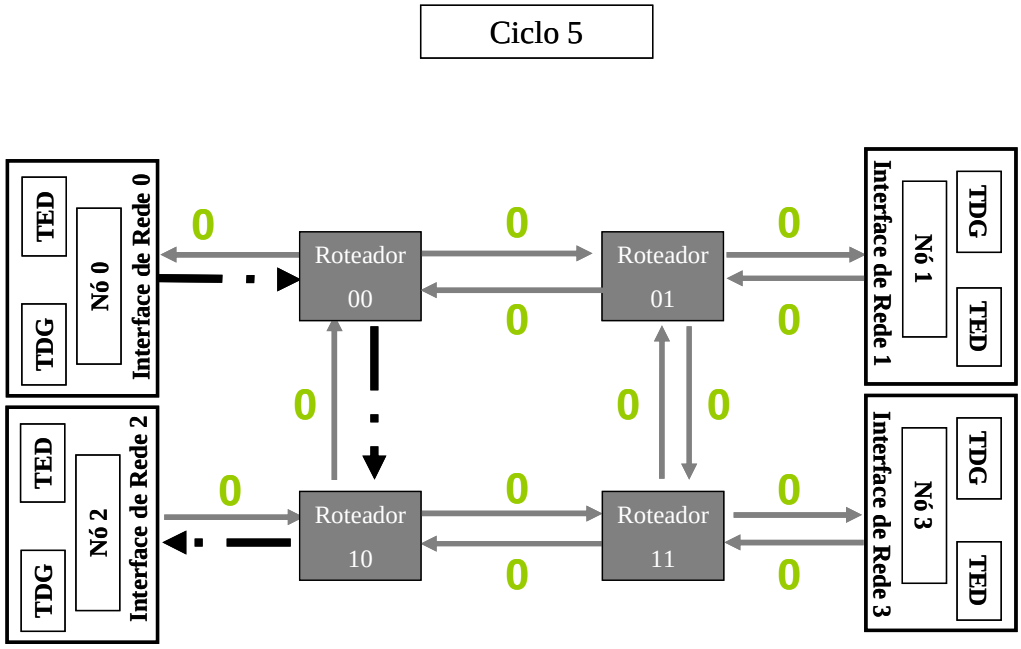
Nota-se que, nos ciclos extras, todos os fios dentro de canais suspeitos que estejam conectados aos nós mas não pertençam ao caminho da mensagem devem ser colocados em um valor predeterminado (ou “0” ou “1”) conforme mostrado na figura 5.4(a) e 5.4(b). Se esses canais estiverem envolvidos na falha, então, o valor imposto (“0”) dominará a falha (para falhas do tipo *wired-AND*) e garantirá que a estrutura de teste indique um erro. Se um “1” lógico é imposto, então, o canal não provará a dominância na falha e por conseqüência não provocará o erro, permitindo que os outros suspeitos sejam checados.


Ciclo 3
Sinais do canal levados a nível lógico 1
Interface de Rede 0
TED
Nó 0
TDG
Interface de Rede 1
Nó 1
TDG
TED
Interface de Rede 2
TED
Nó 2
TDG
Interface de Rede 3
Nó 3
TDG
TED
Roteador 00
Roteador 01
Roteador 10
Roteador 11
1
0

(a)


Ciclo 4
Sinais do canal levados a nível lógico 0
Interface de Rede 0
TED
Nó 0
TDG
Interface de Rede 1
Nó 1
TDG
TED
Interface de Rede 2
TED
Nó 2
TDG
Interface de Rede 3
Nó 3
TDG
TED
Roteador 00
Roteador 01
Roteador 10
Roteador 11
0

(b)



(c)

Figura 5. 4: Ciclos de teste adicionais visando o aumento da capacidade de localização de falhas.(a) Ciclo 3; (b) Ciclo 4; (c) Ciclo 5.

Colocar em um valor lógico predeterminado um dado canal pode ser feito de forma simples em canais que se conectam aos nós da rede e enviam as mensagens através dela, já que estes canais atuam como os *drivers* do canal. Os canais de comunicação ligados ao nó e que recebem as mensagens da rede, contudo, não tem como controlar diretamente o nível lógico do canal, já que neste caso, quem atua como *driver* do canal é o roteador ligado na outra extremidade.

Uma alternativa para controlar o valor de um determinado canal é simplesmente mandar uma mensagem através dele (se este canal não estiver sendo utilizado) durante a duração do teste. Uma segunda alternativa para poder controlar o canal do roteador é através de comandos de set/reset de forma que este controle o valor do canal de comunicação quando não estiver sendo utilizado para comunicação.

Considerando-se o exemplo de caso não resolvido que resultou na lista da tabela 5.4 e assumindo falhas do tipo *wired-AND*, o próximo ciclo extra de envio de mensagem aplicado é o ciclo 3. Neste ciclo, e para os canais suspeitos listados, o pacote é roteado do nó01 passando pelo roteador01, roteador00, roteador10 até chegar à estrutura de teste do nó 10. O canal que conecta o nó 00 ao roteador 00 deve ser colocado em nível lógico “1”. Nota-se que este canal faz parte da lista de suspeitos. A seqüência enviada pelo nó 01 preenche o caminho da mensagem com zeros, mandando *flits* contendo todos os *bits* com “0”. No meio do pacote é enviado um *flit* contendo todos os *bits* em “1”. Assim que este *flit* chega ao canal que conecta o roteador 00 ao roteador 01 ele somente será afetado se a falha for entre os canais roteador00-ao-roteador01 e o canal roteador00-ao-roteador10. Logo, se o nó 10 detectar um erro no ciclo 3, a falha é localizada, caso contrário um suspeito da lista de suspeitos é eliminado e outro ciclo de envio de mensagens é aplicado.

O ciclo 4 é muito similar ao ciclo 3, com a mesma mensagem e caminho de mensagem na rede. Contudo, o nível lógico ao qual o canal nó00-ao-roteador00 é

colocado é “0” ao invés de “1”. Neste ciclo, quando o *flit* contendo todos os *bits* em “1” atinge o canal roteador00-ao-roteador10, ele somente será afetado se a falha for entre os canais roteador00-ao-roteador10 e nó00-ao-roteador00. Nesse caso, o nó10 detecta o erro e a falha é identificada. Caso contrário outro suspeito é eliminado da lista e o ciclo extra 5 é aplicado.

No ciclo 5, a mesma mensagem é enviada na rede, entretanto, o caminho da mensagem passa a ser diferente do caminho utilizado nos dois ciclos anteriores. O pacote é roteado do nó 00 ao nó 10 passando pelo roteador 00 e roteador 10. Quando o *flit* contendo todos os *bits* em “1” é enviado, ele somente será afetado pela falha se esta for entre fios de dois canais diferentes, eliminando, assim, a possibilidade de a falha ser entre dois fios dentro do mesmo canal. Nesse caso, a detecção de erro sinalizada pelo nó 10 torna possível a identificação da falha entre as duas possibilidades remanescentes.

O caminho da mensagem e os canais envolvidos nos ciclos extras dependem da lista de suspeitos. Para falhas que acarretem sinalizações de erro de *timeout*, o mesmo tipo de classificação de suspeitos da tabela 5.5 pode ser feito. A classificação da relação das falhas que ocasionam erro de *timeout* com as possíveis falhas resultaram em 72, associados a 30 pares de canais suspeitos. Isso é possível, já que mais de uma sinalização de erro aponta para a mesma falha. É observado, também, um total de 24 casos não resolvidos associados a 28 canais suspeitos.

5.7 Ambiente de Simulação

Para poder avaliar o método proposto, uma rede SoCIN de topologia grelha e tamanho 2x2 foi implementada, juntamente com um mecanismo de injeção de falhas baseado em simulação capaz de injetar as falhas compreendidas no modelo de falhas. O ambiente de simulação utilizado foi similar ao ambiente descrito em 3.1, contudo, este foi adaptado de forma a utilizar a tabela de análise de sinalização de erros (tabela 5.5), permitindo a validação do método de localização de falhas.

A simulação foi realizada com a ferramenta ModelSim, onde os sinais internos puderam ser manipulados usando o comando de *signal_force* utilizado no *testbench*. A lógica de localização das falhas foi inserida, também, no circuito de *testbench*, sendo utilizada para validar o método.

5.8 Resultados

O algoritmo de localização de falhas proposto deve ser executado por um equipamento testador externo (ATE). No experimento realizado, o ambiente de simulação fez o papel de um equipamento de teste externo. A localização de falhas foi feita analisando as respostas das estruturas de teste nos nós da rede, a partir da tabela de localização de falhas (tabela 5.5) e comparando o *flit* que apresenta erro com o resultado esperado. Conforme mencionado anteriormente, a seqüência *walking one* foi utilizada para definir os vetores de teste e detectar os fios faltosos. Outra seqüência pode ser utilizada, aplicando-se uma análise similar.

As 8.128 possíveis falhas de curto-circuito em todos os 128 fios de dados da NoC 2x2 foram injetadas individualmente.

A tabela 5.6 apresenta os resultados do método de localização de falhas para todas as 8.128 falhas injetadas. A tabela é dividida em dois grupos: localizados e não

completamente localizados. O grupo de falhas localizadas é dividido em subgrupos formados pelas falhas detectadas nos primeiros dois ciclos de teste e as falhas detectadas após os ciclos extras de teste. As falhas também são divididas em falhas que causam erros de *payload* e falhas que causam erros de *timeout*.

Da tabela 3.4.6, observa-se que a capacidade de localização de falhas do método funcional com as alterações propostas pode ser bem eficiente (93%). Para os casos não resolvidos, que representam os restantes 7,32% das falhas, alguns casos podem estar relacionadas a dois pares de canais com falhas e outros relacionados com até três casos de canais com falhas.

Tabela 5. 6: Resultados das simulações para validação do método de localização de falhas.

Falhas Localizadas			Falhas não localiazdas	Total
Erros de payload		**Erros de Timeout**	**Erros de Timeout**	
Localizadas nos dois primeros ciclos	**Localizadas com ciclos extras**	**Localizadas nos dois primeiros ciclos**	**Não resolvidas**	
4089 (50.31%)	2611 (32.12%)	833 (10.25%)	595 (7.32%)	8128 (100%)

RESUMO DO CAPÍTULO 5

Este capítulo, avaliou a capacidade de localização de falhas do método proposto para teste das interconexões da NoC baseado no envio de mensagens através da rede. O método de localização de falhas baseia-se na análise das sinalizações de erro por parte de cada circuito TRA.

Foram propostas alterações no envio das mensagens na rede a fim de aumentar a capacidade de localização das falhas na rede. Foram implementados múltiplos ciclos de envio de pacotes de teste na rede passando por caminhos diferentes. Como resultado, foi obtida uma capacidade de localização de falha para as falhas de curto-circuito nas interconexões acima de 90%.

6 INCLUSÃO DO TESTE DOS ROTEADORES

Conforme mencionado em capítulos anteriores, o teste das redes-em-chip (NoCs) pode ser dividido em duas partes: o teste dos roteadores e o teste das interconexões. Até agora, os métodos de teste funcional apresentados neste trabalho abordaram apenas as falhas nas interconexões da NoC. Neste capítulo, utilizando-se o método de teste descrito no capítulo 3, é avaliada a cobertura de falhas para as falhas nos roteadores da rede. Foram levadas em consideração falhas nas diferentes partes que compõem a lógica dos roteadores da NoC utilizada como estudo de caso (lógica de roteamento, lógica de arbitragem, FIFOs).

Visando aumentar a cobertura de falhas, são acrescentados ao método de teste descrito no capítulo 3, diferentes caminhos para envio de mensagens de teste na rede. O impacto desses novos caminhos de teste foi avaliado para as falhas do roteador, bem como para as falhas nas interconexões da rede.

6.1 NoC Utilizada – Estudo de Caso

A rede utilizada como estudo de caso foi a mesma apresentada nas seções anteriores (SoCIN), mantendo-se também a mesma topologia de rede (grelha 2D). Manteve-se, também, a largura do canal de comunicação (8 *bits* de dados), além dos 2 *bits* de controle. O tamanho da rede utilizada nas simulações, contudo, passou a ser 3x3, já que este tamanho permite um conjunto maior de configurações de teste que se mostraram interessantes para detectar as falhas nos roteadores.

Os roteadores utilizados foram descritos com 5 portas – norte (N), sul (S), leste (E), oeste (W) e a conexão local (L) ligada ao núcleo. Os *buffers* de entrada foram configurados como tendo 3 posições de 10 *bits* (8 *bits* de dados, além dos *bits* de controle de começo e fim do pacote – *bop* e *eop*). Cada canal possui igualmente estes 10 *bits*, além de 2 *bits* responsáveis pelo controle de fluxo através do *handshake* (*ack* e *val*).

6.2 Modelo de Falhas

O modelo de falhas utilizado para o teste nas interconexões foi o mesmo apresentado no capítulo 3.2. Contudo, foram acrescentadas a este modelo, as falhas na lógica do roteador. Para isso, foram consideradas falhas do tipo *stuck-at* em nós dos circuitos dos roteadores. Elementos do circuito que apresentam falhas do tipo *stuck-at* apresentam valores lógicos fixos nos nós afetados, conforme mencionado no capítulo 2. No caso de uma falha de *stuck-at-1*, o nó afetado (entradas ou saídas de portas lógicas) apresenta valor lógico fixo em “1”. De forma similar, uma falha de *stuck-at-0* representa um valor lógico fixo em “0” no nó afetado.

O roteador central de uma NoC 3x3 foi sintetizado (utilizando-se a ferramenta ISE 9.1 da Xilinx) resultando em uma *netlist* (VHDL mapeado) com 595 pontos de injeção de falhas do tipo *stuck-at*, entre saídas de *flip-flops* e demais portas lógicas (nós do circuito). O modelo de falhas resultante apresentou 1.918 falhas considerando-se as falhas do tipo *stuck-at-0* e *stuck-at-1*.

6.3 Ambiente de Simulação

Para simular o teste proposto e validar a metodologia apresentada, o mesmo ambiente de simulação descrito no capítulo 3.1 foi utilizado. Foi incluído no mecanismo de injeção as falhas do tipo *stuck-at* nos nós do circuito do roteador (595 pontos de injeção de falhas).

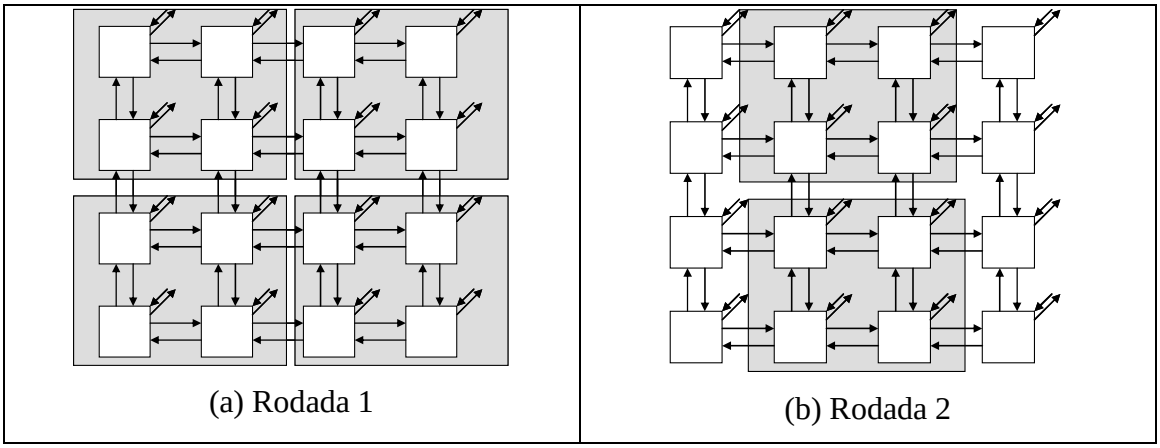
6.4 Teste dos Roteadores da Rede

O método de teste descrito no capítulo 3.1 busca detectar falhas apenas nas interconexões de uma NoC. Durante este teste, contudo, a lógica dos roteadores também é exercitada, uma vez que a NoC opera em modo funcional. Assim, falhas no interior dos roteadores podem ser detectadas pelo mesmo teste desde que as falhas nesses roteadores provoquem um erro na transmissão da mensagem na rede. Este erro deve ser detectado pelas estruturas de teste responsáveis pela recepção e comparação da mensagem com os valores esperados.

Entretanto, considerando-se que a seqüência de teste usada foi feita objetivando-se apenas o teste das interconexões, não é possível assumir que os mesmos vetores de teste e as configurações de teste (caminhos da mensagem na rede) garantam alta cobertura de falhas para a lógica dos roteadores.

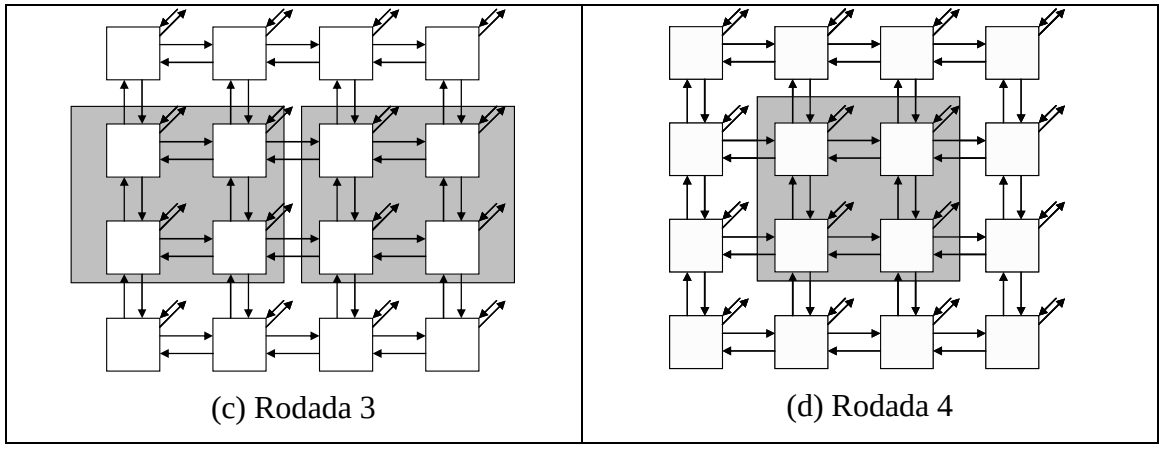
Conseqüentemente, foi necessário alterar a seqüência e configurações de teste visando um teste eficiente para falhas nos blocos internos dos roteadores, incluindo a lógica de controle (para arbitragem e roteamento) e as FIFOs. A seguir são apresentadas as modificações acrescentadas ao método de teste a fim de aumentar a cobertura de falhas considerando-se o novo modelo.

Apenas com a finalidade de estabelecer uma seqüencia nos métodos de teste, são apresentados na figura 6.1 (a), (b), (c) e (d) as rodadas de teste utilizadas no teste das interconexões de uma NoC 4x4. Dando seqüência ao trabalho, esta NoC 4x4 é utilizada como exemplo para os métodos de teste apresentados a seguir.



(a) Rodada 1

(b) Rodada 2



(c) Rodada 3 (d) Rodada 4

Figura 6. 1: Teste das interconexões de uma NoC 4x4.

Os resultados de cobertura de falhas para este método são apresentados na tabela 6.1.

Tabela 6. 1: Resultado do teste das interconexões de uma NoC 2x2 (conforme apresentado em 3.2).

Tipo de Experimento	**Tamanho da Seqüência de Teste (# Ciclos de Relógio)**	**# Falhas Detectadas (% Cobertura de Falhas)**
Rodadas de Teste na Lógica de Roteamento		
Falhas nas interconexões de dados e controle: 18.336 falhas injetadas		
Curto-circuito tipo *AND*	182	18324 (99,93%)
Curto-circuito tipo *OR*	182	18336 (100%)

6.5 Teste dos Roteadores da Rede – FIFOs

As FIFOs de entrada dos canais de comunicação dos roteadores são estruturas que podem ser facilmente acessadas considerando-se o teste funcional. Para isso, é necessário aplicar vetores de teste (incluídos na mensagem transmitida através da rede) apropriados em conteúdo e volume, de forma que seja possível preencher todos os bits de todas as posições da FIFO, com níveis lógicos “1” e “0”. Dessa forma, se algum bit de alguma posição da FIFO apresentar uma falha do tipo *stuck-at*, o valor armazenado será diferente do valor original da mensagem, o que permite a detecção da falha.

A seqüência de teste utilizada no método da seção 3.2 é rica em termos de volume, já que todas as posições das FIFOs são mantidas preenchidas durante a aplicação do teste, porém é pobre em termos de conteúdo, uma vez que os *flits* contendo todos os *bits* em zero dominam a seqüência de teste. O conteúdo pobre da mensagem transmitida permite que *bits* de determinadas posições da FIFO não recebam nunca o valor lógico

"1", ocasionando uma não detecção de falhas do tipo *stuck-at-0* (o comportamento normal é o mesmo do comportamento com a falha).

Por essa razão, foi proposto o acréscimo, à seqüência de teste da seção 3.2, de um pacote de teste com *payload* contendo *flits* com todos os *bits* em "1" lógico, capaz de preencher completamente todas as posições da FIFO. A idéia é garantir que o "1" e o "0" lógico sejam atribuídos a todos os bits de todas as posições da FIFO, incluindo os bits de *bop* e *eop*.

Para poder aplicar valores diferentes ("1" e "0") nos *bits* de *bop* e *eop* de cada posição da FIFO, os pacotes que entram nas FIFOs devem ser defasados de forma que o primeiro e o último *flit* do pacote (que carrega os bits de *bop* e *eop*) sejam recebidos pelas diferentes posições da FIFO. Dessa forma, se a FIFO for configurada com *n* posições, devem ser enviados *n* pacotes com a correta defasagem para que todos os bits recebam valores lógicos "1" e "0".

O resultado parcial de cobertura de falhas para as falhas de *stuck-at* na lógica de um roteador roteador central de uma NoC 3x3 são apresentados na tabela 6.2. Também são apresentados os resultados parciais para o método de teste que visa detectar as falhas em todas as posições das FIFOs (o acréscimo na cobertura de falhas apresentado na tabela se dá devido às falhas nas FIFOs que passaram a ser detectadas com o novo método).

Tabela 6. 2: Resultado parcial de cobertura de falhas para o teste das FIFOs de um roteador central de uma NoC 3x3.

Tipo de Experimento	**Tamanho da Seqüência de Teste (# Ciclos de Relógio)**	**# Falhas Detectadas (% Cobertura de Falhas)**
Falhas na lógica do roteador central de uma NoC 3x3 (1.918 Falhas injetadas)		
Seqüência Walking-One (Rodadas 1, 2, 3 e 4) [3.1.2] *Falhas Stuck-at*	1.144	1.467 (76,49%)
Seqüência para teste nas FIFOs : Falhas na lógica do roteador central de uma NoC 3x3 (1.918 Falhas injetadas)		
Seqüência Walking-One [3.1.2] seguida de um pacote de teste com 3 flits com todos os bits em 1. (Rodadas 1, 2, 3 e 4) *Falhas Stuck-at*	1.164	1.497 (78,05%)

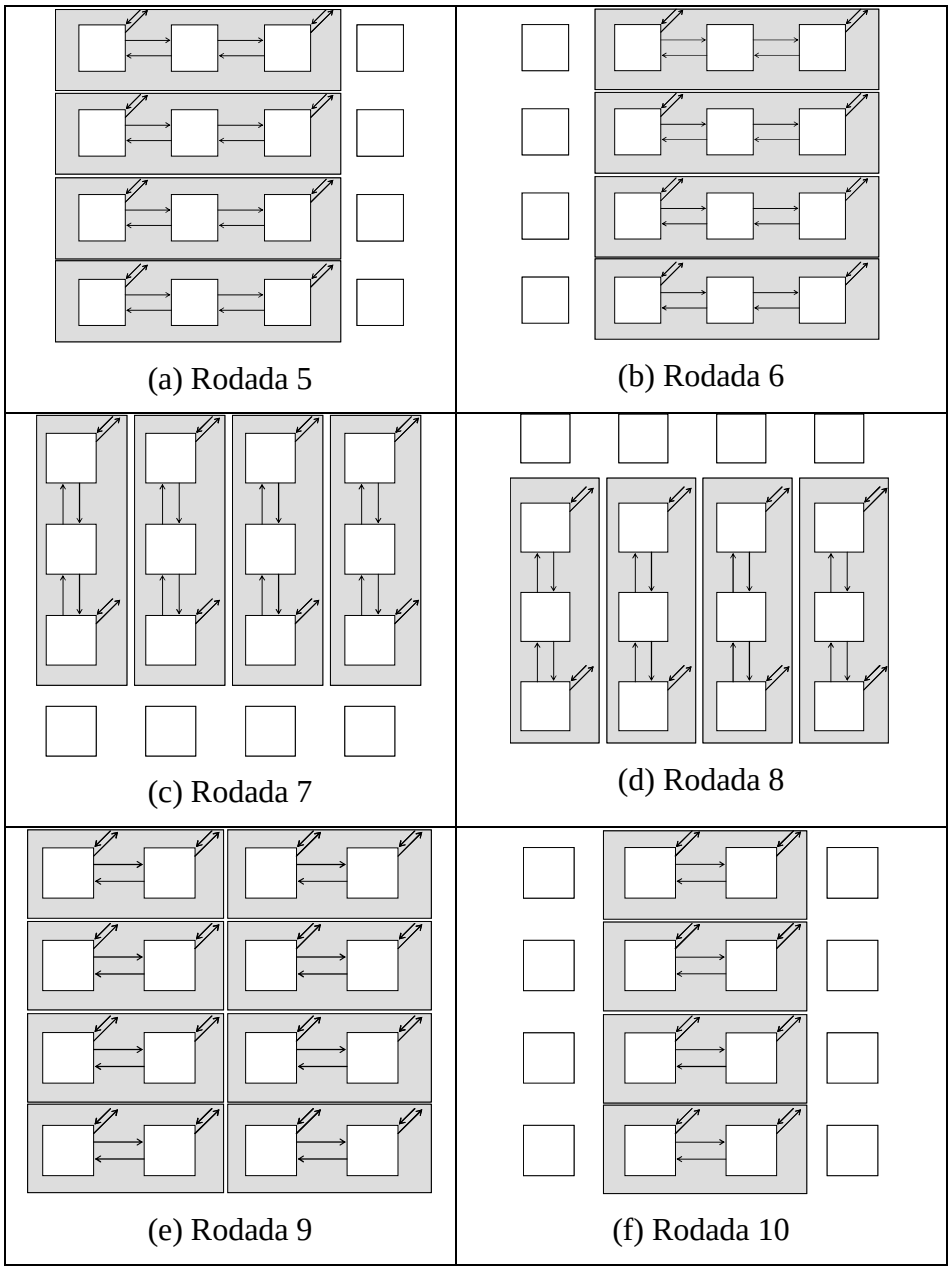
6.6 Teste dos Roteadores da Rede – Lógica de Roteamento

O teste da lógica de controle, com o roteador operando em modo funcional, pode requisitar um número muito alto de configurações de teste. Isso ocorre, já que para exercitar todo o circuito, precisa-se atender a todas as possibilidades de roteamento e arbitragem. Levando-se em consideração apenas a lógica de roteamento existem *p*(p-*

1)-4 configurações de roteamento XY válidos para cada roteador, onde p é o número de portas presentes no roteador (no máximo 5).

Uma análise cautelosa das configurações de roteamento levam a conclusão que 8 de 16 configurações válidas para os roteadores não estão presentes no teste das interconexões. São elas: L→N (para os roteadores 10 e 11, conforme a Figura 6.1), L→S (para os roteadores 00 e 01), E→L (para os roteadores 00 e 10), W→L (para os roteadores 01 e 11) e N→S, S→N, E→W e W→E (para os roteadores internos em topologias de NoCs maiores que 2x2).

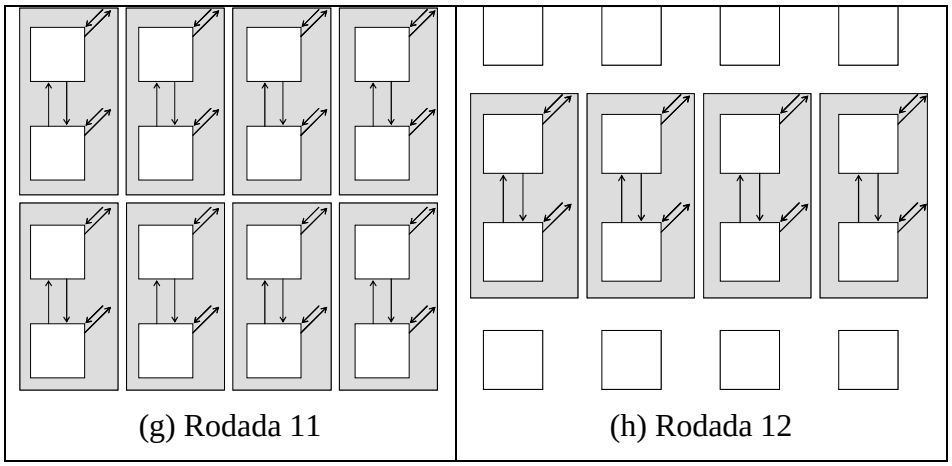
As novas rodadas de teste mostradas na figura 6.2 para uma NoC 4x4 cobrem as seguintes configurações de roteamento adicionais: rodadas 5-6 cobrem E→W e W→E, rodadas 7-8 cobrem N→S e S→N, rodadas 9-10 cobrem E→L e W→L, rodadas 11-12 cobrem L→N e L→S.



(a) Rodada 5 (b) Rodada 6

(c) Rodada 7 (d) Rodada 8

(e) Rodada 9 (f) Rodada 10



(g) Rodada 11 (h) Rodada 12

Figura 6. 2: Rodada de teste adicionais para detecção de falhas na lógica de roteamento.

Essas configurações de teste adicionais garantem que a lógica de roteamento de um roteador da rede seja exercitada de forma que as falhas presentes nela causem algum erro na transmissão da mensagem ocasionando a detecção dessa falha.

Para diferentes políticas de roteamento (que não a XY), uma análise similar deve ser feita considerando todas as possibilidades de roteamento permitidas e estabelecendo caminhos de envio da mensagem que exercitem essas possibilidades.

Os resultados parciais (incrementais) para o método descrito acima, considerando o acréscimo na cobertura de falhas devido a falhas detectadas na lógica de roteamento são apresentados na tabela 6.3.

Tabela 6. 3: Resultados parciais do método de teste visando as falhas na lógica de roteamento de um roteador central em uma NoC 3x3.

Tipo de Experimento	**Tamanho da Seqüência de Teste (# Ciclos de Relógio)**	**# Falhas Detectadas (% Cobertura de Falhas)**
Falhas na lógica de roteamento do roteador central de uma NoC 3x3 (421 falhas injetadas)		
Pacote de teste com 1 flit com todos os bits em 1 e 1 flit com todos os bits em 0 (Rodadas 5, 7, 9, 10, 11 e 12) *Falhas Stuck-at*	28	189 (9,85%) (resultado incremental)

6.7 Teste dos Roteadores da Rede – Lógica de Arbitragem

Levando-se em consideração as falhas na lógica de arbitragem, todas as possibilidades de arbitragem predefinidas pela política utilizada pelo roteador devem ser

exercitadas. No estudo de caso foi utilizada uma política de arbitragem do tipo *Round-Robin*, conforme descrita no capítulo 1.

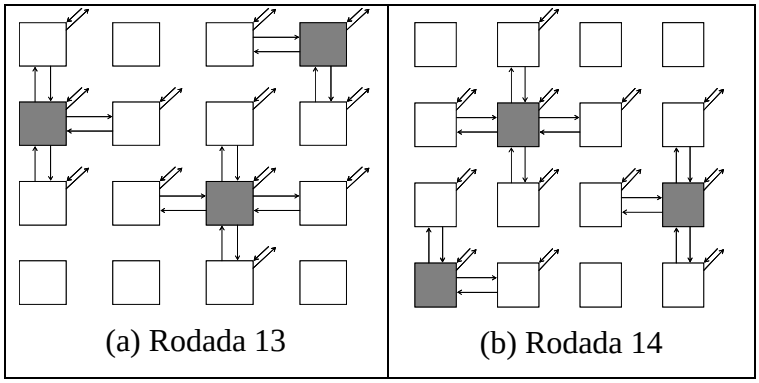
Para isso, é necessário que o roteador receba múltiplas mensagens de tal forma que haja uma requisição simultânea, por mais de um canal de entrada, de canais de saída do roteador. Por exemplo: mensagem recebida pelo canal Sul e mensagem recebida pelo canal Leste fazem requisição para utilizar o canal de saída Norte simultaneamente.

Utilizando-se o método de teste proposto em 3.1 nota-se que não há requisição simultânea de canais de saída em um determinado roteador da rede. Isso ocorre devido à forma como a mensagem de teste é enviada através da rede e faz com que a maioria das falhas que afetam a lógica de arbitragem não sejam detectadas. Para cobrir todas as possibilidades de arbitragem (considerando-se a política utilizada pelo roteador RASoC), seria necessário um conjunto muito grande de configurações de teste (quando comparado às configurações para exercitar a lógica de roteamento). Isto resultaria em um número igualmente grande de rodadas de teste, podendo tornar-se proibitivo, dadas as restrições de tempo de teste do circuito.

Assim, neste trabalho, apenas o subconjunto dessas possibilidades de configurações de teste que envolvem, simultaneamente, todos os canais de entrada de um dado roteador requisitando uma mesma saída é considerado para validação do método.

Nesse caso, para cada roteador da rede, no máximo *p* rodadas de teste são necessárias para o teste da lógica de arbitragem (onde *p* equivale ao número de portas do roteador). Para um roteador completo com 5 portas, por exemplo, as seguintes possibilidades de arbitragem serão exercitadas: as portas E, S, W e N requisitam a porta L; as portas L, E, S e W requisitam a porta N; as portas L e E requisitam a porta W; as portas W, N, L e E requisitam a porta S; e as portas W e L requisitam a porta E.

A fim de reduzir o tempo de teste, deve-se aproveitar ao máximo o paralelismo em cada configuração de teste. A figura 6.3 mostra as configurações de teste necessárias para o teste da lógica de arbitragem de uma NoC 4x4. Os roteadores marcados com cinza na figura são os roteadores sob teste (testados em paralelo com a mesma configuração de teste), que recebem a mensagem de teste dos roteadores adjacentes. Nota-se que, para a topologia estudada, existem roteadores com diferentes números de portas utilizadas na rede.



(a) Rodada 13 (b) Rodada 14

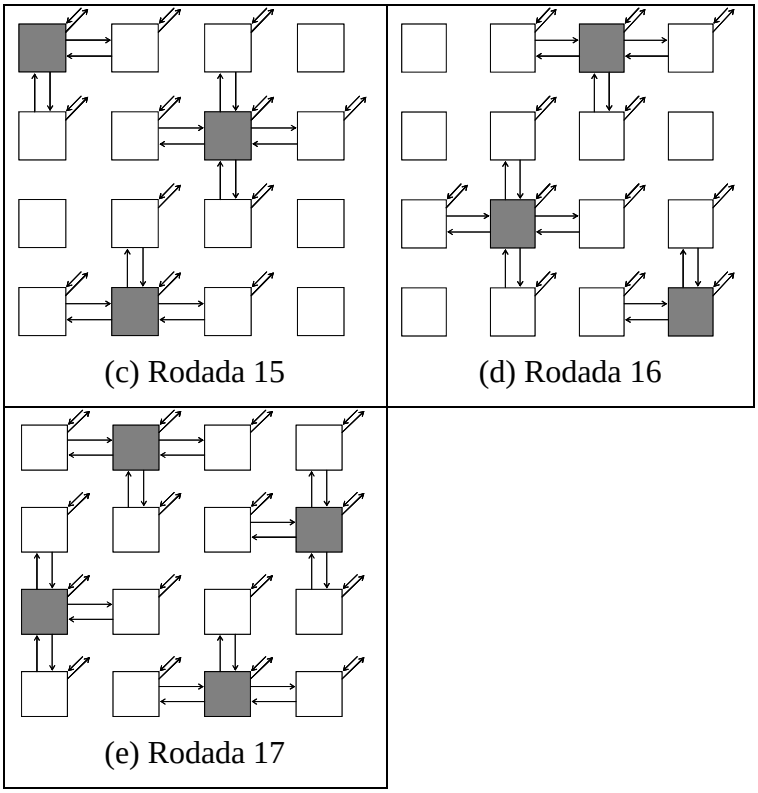


Figura 6. 3: Configurações de teste adicionais para detecção de falhas na lógica de arbitragem.

Para cada configuração figura 6.3((a), (b), (c), (d), (e)) são estabelecidas rodadas de teste. Cada rodada de teste implementa uma das possibilidades de arbitragem mencionadas anteriormente. O número de rodadas em uma configuração de teste é dado pelo maior número de portas em um roteador sob teste. Nas configurações de teste (a), (b), (c) e (d) na figura 6.3 possuem 5 rodadas de teste cada e a configuração (e) possui 4 rodadas de teste. Assim, para a NoC 4x4 da figura 6.3, um total de 24 rodadas de teste adicionais são necessárias.

Também pode ser mostrado que 12, 21 e 25 são, respectivamente, o número de rodadas de teste adicionais necessários para o teste da lógica de arbitragem dos roteadores de uma NoC 2x2, 3x3 e 5x5, respectivamente. Para redes maiores, um crescimento quase linear é esperado.

Como estas rodadas de teste adicionais somente exercitam a lógica de arbitragem, a seqüência de teste pode conter diferentes tipos de *flits* (com todos os *bits* em “0”, metade dos *bits* em “0” e metade em “1”, metade dos *bits* em “1” e metade dos *bits* em “0” e com todos os *bits* em “1”, por exemplo), de forma que seja possível distinguir o comportamento sem falhas do comportamento com falhas. Esses tipos de *flits* são mostrados na tabela 6.4.

Tabela 6. 4: Representação dos *flits* de *payload* do pacote enviado para deteção de falhas na lógica de arbitragem.

	Representação
***Flit* com todos os *bits* em 0.**	**00000000**
***Flit* com todos os *bits* em 1.**	**11111111**

***Flit* com metade dos *bits* em 0 e metade em 1**	**00001111**
***Flit* com metade dos *bits* em 1 e metade em 0**	**11110000**

Os resultados parciais para o método descrito acima, considerando o acréscimo na cobertura de falhas devido a falhas detectadas na lógica de arbitragem são apresentados na tabela 6.5.

Tabela 6. 5: Acréscimo na cobertura de falhas devido ao método de teste que visa as falhas na lógica de arbitragem de um roteador central de uma NoC 3x3.

Tipo de Experimento	**Tamanho da Seqüência de Teste (# Ciclos de Relógio)**	**# Falhas Detectadas (% Cobertura de Falhas)**
Rodadas de Teste na Lógica de Arbitragem: falhas na lógica de arbitragem de um roteador central meu ma NoC 3x3 (232 falhas injetadas)		
1 flit com todos os bits em 0, 1 flit com todos os bits em 1, 1 flit com metade dos bits em 1 e 1 flit com metade dos bits em 0 *Falhas Stuck-at*	294	93 (4,85%) (resultado incremental)

6.8 Teste dos Roteadores da Rede – Interconexões

Nota-se que se a seqüência de teste original for aplicada nas configurações de teste das rodadas 5-8, a vizinhança definida originalmente para simplificar o teste das interconexões (NoC de tamanho 2x2) pode ser estendida de forma que pares de curto-circuito entre todos os canais de comunicação chegando/partindo do mesmo roteador serão, também, cobertos.

Nota-se, também, que as rodadas 5-8 não se aplicam a NoCs 2x2, as rodadas 5 e 6 são as mesmas para NoCs 3x3 e, para NoCs maiores que 4x4, uma rodada de teste adicional na direção W→E e outra na direção N→S são necessárias. Como as rodadas de teste 9-12 são necessárias apenas para o teste das demais possibilidades de roteamento, uma seqüência de teste simples e curta pode ser aplicada, contendo apenas *flits* com todos os *bits* em “0” e *flits* com todos os *bits* em “1”, por exemplo. Para estas rodadas de teste, é possível notar que as rodadas 10 e 12 não se aplicam a NoCs 2x2 e para NoCs maiores, o número de rodadas de teste é mantido o mesmo.

A figura 6.4 mostra como a vizinhança compreendida pelo modelo de falhas é estendida. A figura 6.4 (a) mostra a vizinhança proposta no método de teste das interconexões. Já as figuras 6.4 (b) e 6.4 (c) mostram a vizinhança estendida devido as rodadas de teste 5-8.

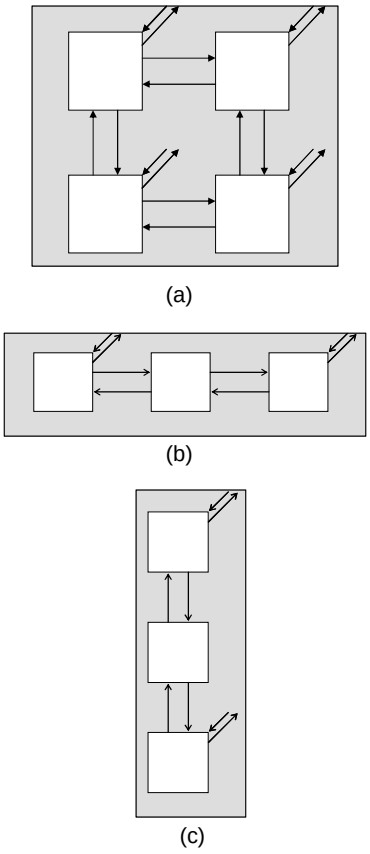


Figura 6. 4: (a) Vizinhança de teste proposta no método de teste das interconexões; (b) e (c) vizinhança de teste estendida devido às rodadas extra para o teste dos roteadores.

Os resultados incrementais na vizinhança estendida considerando falhas nas interconexões da NoC são mostradas na tabela 6.6.

Tabela 6. 6: Resultados incrementais de cobertura de falhas nas interconexões da vizinhança estendida da NoC sob teste.

Tipo de Experimento	**Tamanho da Seqüência de Teste (# Ciclos de Relógio)**	**# Falhas Detectadas (% Cobertura de Falhas)**
Falhas nas interconexões (bits de dados e controle) das portas s W-E (Rodada 5) e nas portas N-S (Rodada 7)do roteador central de uma NoC 3x3 (4,608 falhas injetadas)		
Seqüência Walking-One [3.1.2] (Rodadas 5 e 7) *Curto-circuito tipo AND*	518	4.602 (99,87%)
Seqüência Walking-One [3.1.2] (Rodadas 5 e 7) *Curto-circuito tipo OR*	518	4.608 (100%)

6.9 Resultados

Os experimentos realizados com o objetivo de validar o método descrito acima resultaram na tabela 3.2. A tabela 6.7 apresenta os resultados da simulação de falhas em uma NoC SoCIN de tamanho 3x3.

Conforme a descrição apresentada em 3.2, as 18.336 falhas foram injetadas exaustivamente em todas as 192 interconexões da topologia básica (2x2) da rede, sendo que os resultados foram apresentados na seção 3.2. Com o modelo de falhas estendido compreendendo falhas na lógica do roteador, conforme apresentado nesta seção, acrescentou-se os resultados apresentados na tabela.

As falhas do tipo *stuck-at* foram injetadas exaustivamente nos *flip flops* e portas lógicas de um roteador central em uma rede de tamanho 3x3 sob teste, enquanto as 4 rodadas de teste de interconexões (3.2) foram aplicadas. De acordo com a primeira linha da tabela 3.2.1, 1.467 de 1.918 falhas *stuck-at* foram detectadas, resultando em 76,49% de cobertura de falhas. Como esperado, muitas das falhas do tipo *stuck-at-0* na FIFO de entrada não foram detectadas devido à dominância de zeros na seqüência aplicada. Contudo, acrescentando-se um pacote com três *flits* com todos os *bits* em 1 à seqüência inicialmente utilizada, foi possível detectar todas as falhas do tipo *stuck-at* na FIFO de entrada.

É importante ressaltar que nenhuma rodada de teste adicional é necessária e que o tamanho da seqüência de teste aumenta muito pouco. Conforme indicado na linha 2 da tabela 6.7, a cobertura de falhas aumenta para 78,05%, ainda assim mantendo 421 falhas não detectadas.

Para aumentar ainda mais a cobertura de falhas, novas rodadas de teste que exercitam outras partes do roteador foram implementadas, algumas para testar a lógica de roteamento e outras para a lógica de arbitragem. Começando com o roteamento, as rodadas de teste da figura 6.2 foram aplicadas a uma NoC de tamanho 3x3 (lembrando que, para uma topologia de 3x3, as rodadas de teste 6 e 8 não são aplicados).

Primeiramente, foram avaliados os efeitos na cobertura de falhas de interconexão aplicando as rodadas de teste 5 e 7, considerando uma maior vizinhança. Para as duas rodadas de teste, 518 ciclos de relógio foram necessários para cobrir 100% de todos os pares de curtos-circuitos em uma vizinhança E→W e W→E (rodada 5), N→S e S→N (rodada 7) de um roteador. Esse resultados são apresentados nas linhas 3 e 4 da tabela 6.6.

Voltando à lógica do roteador, as rodadas 9 a 11 foram aplicados utilizando-se pacotes com um *flit* com todos os *bits* em 0 e um *flit* com todos os *bits* em 1. As 421 falhas restantes foram injetadas no roteador central da NoC de topologia 3x3. Para cada rodada de teste, 4 ciclos de relógio (cabeçalho + 2 *flits* de *payload* + terminador) além da latência de 3 ciclos de relógio, são necessários para aplicar a seqüência. De acordo com a linha 5 da tabela 3.2.1, 189 novas falhas foram detectadas e a cobertura de falhas subiu para 87,9%. Ainda assim, 232 falhas permaneceram não detectadas.

Finalmente, as rodadas de teste para a detecção de falhas na lógica de arbitragem foram aplicadas utilizando-se pacotes com diferentes *payloads* (um *flit* com todos os *bits* em 0, um *flit* com todos os *bits* em 1, um *flit* com metade dos *bits* em 0 e metade dos *bits* em 1 e um *flit* com metade dos *bits* em 1 e metade dos *bits* em 0). Neste caso, cada pacote possui 3 *flits* (*header* + *payload* com um dos *flits* mencionados + terminador).

Tabela 6. 7: Resultados dos experimentos para validação da metodologia proposta.

Tipo de Experimento	**Tamanho da Seqüência de Teste (# Ciclos de Relógio)**	**# Falhas Detectadas (% Cobertura de Falhas)**
Falhas na lógica do roteador central de uma NoC 3x3 (1.918 Falhas injetadas)		
Seqüência Walking-One (Rodadas 1, 2, 3 e 4) [3.1.2] *Falhas Stuck-at*	1.144	1.467 (76,49%)
Seqüência para teste nas FIFOs : Falhas na lógica do roteador central de uma NoC 3x3 (1.918 Falhas injetadas)		
Seqüência Walking-One [3.1.2] seguida de um pacote de teste com 3 flits com todos os bits em 1. (Rodadas 1, 2, 3 e 4) *Falhas Stuck-at*	1.164	1.497 (78,05%)
Rodadas de Teste na Lógica de Roteamento		
Falhas nas interconexões (bits de dados e controle) das portas s W-E (Rodada 5) e nas portas N-S (Rodada 7)do roteador central de uma NoC 3x3 (4,608 falhas injetadas)		
Seqüência Walking-One [3.1.2] (Rodadas 5 e 7) *Curto-circuito tipo AND*	518	4.602 (99,87%)
Seqüência Walking-One [3.1.2] (Rodadas 5 e 7) *Curto-circuito tipo OR*	518	4.608 (100%)
Falhas na lógica de roteamento do roteador central de uma NoC 3x3 (421 falhas injetadas)		
Pacote de teste com 1 flit com todos os bits em 1 e 1 flit com todos os bits em 0 (Rodadas 5, 7, 9, 10, 11 e 12) *Falhas Stuck-at*	28	1686 (87,9%)
Rodadas de Teste na Lógica de Arbitragem: falhas na lógica de arbitragem de um roteador central meu ma NoC 3x3 (232 falhas injetadas)		
1 flit com todos os bits em 0, 1 flit com todos os bits em 1, 1 flit com metade dos bits em 1 e 1 flit com metade dos bits em 0 *Falhas Stuck-at*	294	1779 (92,75%)

O pior caso do tamanho da seqüência de teste para as configurações de teste (a), (b), (c) e (d) da figura 6.3, é o que um dos roteadores com no máximo 4 canais de entrada requisitam 1 canal de saída (a configuração em formato de cruz). Para estes roteadores,

quando as portas E, S, W e N requisitarem a porta L, ou as portas L, E, S e W requisitarem a porta N, ou as portas W, N, L e E requisitarem a porta S, será permitida a utilização da saída a uma depois da outra e os 4 pacotes de 3 *flits* de cada entrada requisitante alcançará o destino após a latência de 3 ciclos de relógio. Para estas 3 configurações, o tamanho da seqüência de teste será: 3 configurações * (4 canais de entrada * 3 *flits* + 3 ciclos de latência) = 45 ciclos de relógio. Quando as portas L e E requisitarem a porta W, ou as portas W e L requisitarem a porta E, a mesma lógica pode ser aplicada, contudo com menos configurações e menos canais de entrada requisitantes. Para estas duas configurações, o tamanho da seqüência de teste será de: 2 configurações * (2 canais de entrada * 3 *flits* + 3 ciclos de latência) = 18 ciclos de relógio.

RESUMO DO CAPÍTULO 6

No capítulo 6, o método de teste das interconexões foi avaliado em termos de falhas de colagem na lógica dos roteadores da NoC. A partir dessas análises, foram propostas alterações método de teste, visando aumentar a cobertura de falhas considerando-se o novo modelo de falhas.

O novo método de teste utiliza novos pacotes de teste e estabelece diferentes caminhos para estes pacotes. Estas novas estratégias visam exercitar mais partes da lógica dos roteadores como o circuito de roteamento e arbitragem. As FIFOs também são testadas em sua totalidade através do método proposto.

Foi obtida uma cobertura de falhas acima de 90% para falhas de colagem na lógica dos roteadores.

7 CONCLUSÕES

As redes-em-chip têm se mostrado como uma solução para os gargalos na comunicação dos módulos de IP em sistemas-em-chip. Aproveitando as características intrínsecas dessa estrutura de comunicação, torna-se possível avaliar diferentes estratégias de teste visando alta cobertura de falhas com um tempo de teste reduzido quando comparado a estratégias tradicionais de teste, o que impacta diretamente na qualidade e no custo final desses circuitos integrados.

As estratégias de teste descritas ao longo desse trabalho utilizam a própria funcionalidade da rede para testar defeitos de manufatura do *chip*. Inicialmente, é abordado o teste das interconexões das redes, onde o envio de mensagens na rede é utilizado para detectar falhas. Considera-se um modelo de falhas compreendendo falhas do tipo curto-circuito (*wired-AND* e *wired-OR*). Uma seqüência de teste é apresentada a fim de exemplificar a estratégia de teste adotada.

O método foi validado através de simulações e obteve uma cobertura de falhas de 100% para o modelo de falhas adotado. Dessa forma, o método foi estendido a um modelo de falhas compreendendo falhas em um conjunto maior de interconexões da rede (incluindo sinais de dados e de controle de *handshake*). Esta nova análise atingiu uma cobertura de falhas de 99%.

Estruturas de teste, compondo a estratégia de BIST, foram avaliadas em termos de impacto de área. Juntamente com estas estruturas, foi analisado um mecanismo de acesso ao teste (para configuração das estruturas de teste e teste interno desses circuitos) obtendo-se um tempo estimado de teste em função do tamanho da rede e levando-se em consideração tempos de configuração, aplicação do teste e extração dos resultados.

A alta cobertura de falhas da metodologia de teste apresentada motivou a investigação de sua eficácia quando são acrescentadas falhas na lógica dos roteadores ao modelo de falhas. A aplicação dos mesmos vetores de teste com a mesma forma de envio de mensagens na rede considerando o novo modelo de falhas resultou em uma cobertura de 76% da falhas. Observando-se a possibilidade de aumentar a cobertura de falhas com o aumento das possibilidades de caminhos de roteamento (e, por conseqüência, um maior exercício da lógica interna dos roteadores), foram propostos novos caminhos para o envio de mensagem na rede. Com os novos caminhos propostos, atingiu-se mais de 90% das falhas internas aos roteadores (considerando-se falhas do tipo “stuck-at-0” e “stuck-at-1”).

Os bons resultados quanto à detecção de falhas levaram à análise sobre a capacidade de localização dessas falhas. Uma vez que as falhas detectadas possam ser localizadas abre-se a possibilidade de tolerar estas falhas permitindo que o sistema funcione, mesmo que apresentando algum defeito de fabricação. Um algoritmo de localização de

falhas foi proposto, obtendo uma capacidade de localizar acima de 80% das falhas nas interconexões.

Os resultados obtidos demonstram que as metodologias de teste podem ser aplicadas aos circuitos-alvo como alternativa, ou complementares a metodologias tradicionais de teste.

REFERÊNCIAS

ABRAMOVICI, M.; BREUER, M. A.; Friedman, A. D. **Digital Systems Testing and Testable Designs**. New York: IEEE, 1990.

AKTOUF, C. A Complete Strategy for Testing an On-Chip Multiprocessor Architecture, **IEEE Design and Test of Computers**, vol. 19-1, 2002.

ALAGHI, A. et al. Online NoC Switch Fault Detection and Diagnosis Using a High Level Fault Model. **Proceedings Defect and Fault-Tolerance in VLSI Systems, DFT '07**, pp. 21-29, Setembro 2007**.**

AMORY, A. M. et al. Reducing test time with processor reuse in network-on-chip based systems. **Proceedings of 17th Symposium on Integrated Circuits and Systems Design, SBCCI 2004**, pp. 111-116, 2004.

AMORY, A. M. et al. A Scalable Test Strategy for Network-on-Chip Routers, **Proceedings of the IEEE International Test Conference (ITC)**, pp. 591-599, 2005.

AXEL, J.; HANNU, T. **Networks on Chip.** Dordrecht: Kluwer, 2003.

BENGTSSON, T. et al. Off-Line Testing of Delay Faults in NoC Interconnects. **Proceedings of 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, DSD 2006**, pp.677-680, 2006.

BENINI, L.; DE MICHELI, G. Networks on chips: a new SoC paradigm. **Computer**, Volume 35, Issue 1, pp. 70-78, Janeiro 2002.

BJERREGAARD, T., MAHADEVAN, S. A Survey of Research and Practices of Network-on-Chip. **ACM Computing Surveys, CSUR**, New York, Volume 38, n. 1, 2006.

CHENG, W.T.; LEWANDOWSKI, J.L.; WU, E. Diagnosis for Wiring Networks, **International Test Conference, ITC**, pp. 565-571, 1990.

CONCATO, C. et al. Improving Yield of NoC-Based SoCs Through Fault-Diagnosis-and-Repair of Interconnect Faults, **IEEE International On-Line Test Symposium, IOLTS 2009**, pp. 61-66, 2009.

COTA, E.; CARRO, L.; LUBASZEWSKI, M. Reusing an on-chip network for the test of core-based systems. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, New York, Volume 9, Issue 4, pp. 471-499, Outubro 2004.

COTA, E. et al. Redefining and Testing Interconnect Faults in Mesh NoCs. **Proceedings of International Test Conference, ITC 2007**, pp. 1-10, 2007.

COTA, E. et al. A High-Fault-Coverage Approach for the Test of Data, Control and Handshake Interconnects in Mesh Networks-on-Chip. **IEEE Transactions on Computers**, Volume 57, n. 9 pp. 1202-1215, Setembro 2008.

D'ALESSANDRO, C. et al. Multiple-Rail Phase-Encoding for NoC. **Proceedings of 12th IEEE international Symposium on Asynchronous Circuits and Systems**, 10 pp. – 116, 2006.

GRECU, C. et al. Methodologies and Algorithms for Testing Switch Based NoC Interconnects, **20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems**, DFT, pp. 238-246, 2005.

GRECU, C. et al. BIST for Network-on-Chip Interconnect Infrastructures, **24th IEEE VLSI Test Symposium, VTS 2006**, 6pp, 2006.

GRECU, C. et al. On-line Fault Detection and Location for NoC Interconnects, **Proceedings of the 12th IEEE International On-Line Testing Symposium (IOLTS'06)**, pp. 145-150, 2006**.**

GRECU, C. et al. Essential Fault-Tolerance Metrics for NoC Infrastructures. **Proceedings of 13th IEEE International On-Line Testing Symposium, IOLTS 2007**, pp. 37-42, 2007.

GUERRIER, P.; GREINER, A. A Generic Architecture for On-Chip Packet-Switched Interconnections. **Proceedings of Design, Automation and Test in Europe Conference and Exhibition, DATE 2000**, pp. 250-256, 2000.

HERVÉ, M. et al. NoC Interconnection Functional Testing: Using Boundary-Scan to Reduce the Overrall Testing Time, **10th Latin American Test Workshop, LATW 2009**, 6pp., 2009.

HERVÉ, M. et al. Diagnosis of Interconnect Shorts in Mesh NoCs, **Proceedings of 3rd ACM/IEEE International Symposium on Networks-on-Chip, NOCS 2009**, pp. 256-265, 2009.

HOSSEINABADY, M. et al. A concurrent testing method for NoC switches, **Proceedings of Design, Automation and Test in Europe, DATE '06**, Volume 1, Issue 6 pp. 6-10, Março 2006.

HOSSEINABADY, M. et al. Using the Inter- and Intra-Switch Regularity in NoC Switch Testing," **Proc. Design Automation and Test in Europe(DATE)**, pp. 361-366, 2007.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERING. **IEEE std 1149.1**: IEEE Standard Test Access Port and Boundary Scan Architecture.t. New York, 2001.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERING. **IEEE std P1500-2005**: IEEE Standard Testability Method for Embedded Core-Based Integrated Circuit. New York, 2005.

JUTMAN, A.; UBAR, R.; RAIK, J. New Built-In Self-Test Scheme for SoC Interconnect. **Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics, WMSCI**, Volume 4, pp. 19-24, 2005.

LEHTONEN, T., LILJEBERG, P., PLOSILA, J. Fault Tolerance Analysis of NoC Architectures. **Proceedings of IEEE International Symposium on Circuits and Systems, ISCAS 2007**, pp.361-364, 2007.

LEHTONEN, T., LILJEBERG, P., PLOSILA, J. Online Reconfigurable Self-Timed Links for Fault Tolerant NoC. **VLSI Design**, Volume 2007, 13 pp., 2007.

LI, M.; JONE, W.; ZENG, Q. An Efficient Wrapper Scan Chain Configuration Method for Network-on-Chip Testing. **IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures**, Volume 0, 6 pp, Março 2006.

LIU, C. et al. Reuse Based Test Access and Integrated Test Scheduling for Network-on-Chip. **Proceedings of Design, Automation and Test in Europe (DATE)**, pp. 303-308, 2006.

MEDIRATTA, S. D., DRAPER, J. Characterization of a Fault-tolerant NoC Router. **Proceedings of IEEE International Symposium on Circuits and Systems, ISCAS 2007**, pp.381-384, 2007.

MUHAMMAD, A.; WELZL, M.; HESSLER, S. A Fault Tolerant Mechanism for Handling Permanent and Transient Failures in a Network on Chip. **Fourth International Conference on Information Technology, ITNG 2007**, pp. 1027-1032, 2007.

NAHVI, M.; IVANOV, A. Indirect Test Architecture for SoC Testing. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, Volume 23, Issue 7, pp.1128-1142, Julho 2004.

OST, L. C. **Redes Intra-Chip Parametrizáveis Com Interface Padrão para Síntese em Hardware**. 2004. 116 f. Dissertação de Mestrado (Mestrado em Ciência da Computação) – PUCRS, Porto Alegre.

PANDE, P. P.; DE MICHELI, G. Design, Synthesis, and Test of Networks on Chips. **IEEE Design & Test**, IEEE Computer Society Press, Los Alamos, Volume 22, Issue 5, pp. 404-413, Setembro 2005.

PARK, D. et al. Exploring Fault-Tolerant Network-on-Chip Architectures, **International Conference on Dependable Systems and Networks, DSN 2006**, pp. 93-104, 2006.

RAIK, J., GOVIND, V., UBAR, R. An External Test Approach for Network-on-a-Chip Switches. **Proceedings of 15th Asian Test Symposium, ATS 2006**, pp.437-442, 2006.

VERMEULEN, B. et al. Bringing Communication Networks On Chip: Test and Verification Implications, **IEEE Communications Magazine**, Volume: 41 Issue: 9, Page(s): 74-81, Setembro 2003.

ZEFFERINO, C. A.. **Redes em Chip: Arquiteturas e Modelos para Avaliação de Área e Desempenho**. 2003. 194 f. Tese de Doutorado (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ZORIAN, Y. Testing the Monster Chip. 1999. **IEEE Spectrum**, Page(s): 54-60, Julho, 2009