NELSON ANTONIO GONÇALVES JUNIOR

ANÁLISE E SIMULAÇÃO DE TOPOLOGIAS DE REDES EM CHIP

MARINGÁ

2010

NELSON ANTONIO GONÇALVES JUNIOR

ANÁLISE E SIMULAÇÃO DE REDES EM CHIP

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. João Angelo Martini

MARINGÁ

2010

"Dados Internacionais de Catalogação-na-Publicação (CIP)"
(Biblioteca Setorial - UEM. Nupélia, Maringá, PR, Brasil)

G635a

Gonçalves Junior, Nelson Antonio, 1984-
Análise e simulação de redes em chip / Nelson Antonio Gonçalves Junior. -- Maringá, 2010.
99 f. : il. (algumas color.).

Dissertação (mestrado em Ciência da Computação)--Universidade Estadual de Maringá, Centro de Tecnologia, Dep. de Informática, 2010.
Orientador: Prof. Dr. João Angelo Martini.

1. Redes em chip (Ciência da Computação) - Análise e simulação. 2. Topologia de redes em chip (Ciência da Computação). I. Universidade Estadual de Maringá. Departamento de Informática. Programa de Pós-Graduação em "Ciência da Computação".

CDD 22. ed. -004.65
NBR/CIP - 12899 AACR/2

Maria Salete Ribelatto Arita CRB 9/858
João Fábio Hildebrandt CRB 9/1140

NELSON ANTONIO GONÇALVES JUNIOR

ANÁLISE E SIMULAÇÃO DE TOPOLOGIAS DE REDES EM CHIP

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em 01/03/2010.

BANCA EXAMINADORA

Prof. Dr. João Angelo Martini
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Ronaldo Augusto de Lara Gonçalves
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Marcos Antonio Cavenaghi
Universidade Estadual Paulista "Júlio de Mesquita Filho" – DCo/UNESP

À Almerinda Pierdoná Gonçalves e Antônia Serrato Castanhari pelo carinho que levarei por toda a vida.

Agradecimentos

Aos meus pais e irmãos, que sempre estiveram ao meu lado e propiciaram um ambiente familiar cheio de afeto e compreensão.

Ao professor João Angelo Martini pela orientação na condução do trabalho, amizade, confiança e todos os ensinamentos que levarei ao longo da vida. Agradeço também ao professor Ronaldo Augusto de Lara Gonçalves pelas contribuições e disposição em prestar qualquer tipo de auxílio.

Aos companheiros do Programa de Pós-Graduação em Ciência da Computação Everton, Maurílio, Mauro e Rogério pelo companheirismo e troca de ideias durante o decorrer deste projeto. À Inês pela dedicação e disposição em esclarecer qualquer dúvida e à Renata, cuja amizade e opiniões contribuíram para o desenvolvimento deste trabalho.

Aos colegas de trabalho Edgard e Guilherme pelas conversas, dicas e trocas de ideias. Let There Be Rock!

Ao CNPq pelo suporte financeiro e a todas as pessoas que direta ou indiretamente contribuíram para a realização deste trabalho.

À Deus.

Resumo

Os avanços nos processos de fabricação de chips têm permitido um constante aumento na quantidade de transistores integrados em uma mesma pastilha de silício, possibilitando a associação de todos os componentes de um computador em um único chip. São os chamados Sistemas em Chip, cuja complexidade vem aumentando frequentemente com a integração de diversos componentes, como mais núcleos de processamento. A comunicação entre esses componentes pode ser realizada através de canais ponto-a-ponto dedicados, mais eficientes, porém com custos maiores, ou através de canais multiponto, denominados barramentos, com custos menores, todavia com desempenho inferior. Nos próximos anos, os sistemas em chip tendem a ficar tão complexos, com centenas de núcleos de processamento, que tais arquiteturas de comunicação se tornarão obsoletas. Nesse sentido é importante a investigação de novas técnicas de comunicação em chip para que esta não se torne um gargalo no desempenho de sistemas em chip. Uma abordagem muito discutida atualmente para garantir essa comunicação é a utilização de redes em chip, que mantêm chaves roteadoras para direcionar os pacotes de dados para seus respectivos destinos e são interligadas de acordo com determinada topologia. Nesse contexto, o presente trabalho busca investigar redes em chip, analisando e comparando o desempenho de tais redes com as topologias Anel, Spidergon, Grelha, Cubo Expresso e Toróide, a fim de mostrar o impacto que a topologia pode ter no desempenho e custo final de uma rede intrachip. Um algoritmo de roteamento semidinâmico para redes Toróide também é apresentado, aproveitando a característica da topologia de possuir mais de um caminho mínimo entre um par de nodos.

Abstract

The advance in chips manufacturing have allowed a constant increase in the number of transistors integrated into a single chip. This allows the combination of the components of a computer on a single chip, introducing the Systems on Chip, whose complexity is often increasing with the integration of several components, like processing cores. The communication between these components can be achieved by point-to-point channels, which are more efficient but more expensive, or through multi-point channels, called bus, which are cheaper, but have lower performance. In the upcoming years, the systems on chip tend to be so complex, with hundreds of processing cores, that these communication architectures will become obsolete. Thus it is important to investigate new paradigms of communication so that the message exchange does not become a bottleneck in the performance of systems on chip. An approach which is discussed nowadays is the use of networks on chip, which keep switches to router data packets to their destination and are interconnected according to a specific topology. In this context, this study aims to investigate such networks, analyzing and comparing the performance of networks on chip with Ring, Spidergon, Mesh, Cube Express and Torus topologies, to show the impact that the topology may have on performance and final cost of a network on chip. A semi dynamic routing algorithm for Torus topologies is also introduced.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

ALUT	*Adaptative Look-Up Table*
ASIC	*Application Specific Integrated Circuit*
BOL	*Begin of Line*
CBDA	*Centrally-Buffered, Dynamically-Allocated*
CI	Circuito Integrado
DAMQ	*Dynamically Allocated Multi-Queue*
DSP	*Digital Signal Processor*
EIB	*Element Interconnection Bus*
EOL	*End of Line*
FIFO	*First In, First Out*
FPGA	*Field-Programmable Gate Array*
GALS	*Globally Asynchronous Locally Synchronous*
GPU	*Graphics Processing Unit*
GSM	*Global System for □óbile Communications*
IBM	*International Business Machines*
IP	*Intellectual Property*
LE	*Logic Element*
MANGO	*Message-passing Asynchronous Network-on-chip providing Guaranteed services over OCP interfaces*
MECS	*Multidrop Express Channels*
NoC	*Network on Chip*
ParIS	*Parametrizable Interconnection Switch*
PPE	*Power Processor Element*
RISC	*Reduced Instruction Set Computer*
RTL	Register Transfer Level
SAMQ	*Statically Allocated Multi-Queue*
SMT	*Simultaneous Multi-Thread*

SoC	*Systems on Chip*
SoCIN	*SoC Interconnection Network*
SPE	*Synergistic Processor Elements*
SPIN	*Scalable Programmable Interconnection Network*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High-Speed Integrated Circuits*
VLIW	*Very Long Instruction Word*

Sumário

CAPÍTULO

1

Introdução

1.1. Considerações Iniciais

O avanço no processo de encapsulamento de transistores tem permitido a integração de milhões destes em uma única pastilha de silício, sustentando a Lei de Moore (Moore, 1998). Esse avanço possibilita a coexistência de diversos componentes em um chip, permitindo a manutenção de sistemas computacionais completos dentro de um único chip. Tais sistemas são denominados Sistemas em Chip (*Systems on Chip*) ou SoCs (Mori; Yamada; Takizawa, 1993), e são comumente utilizados em sistemas embarcados e dispositivos portáveis, uma vez que apresentam baixo consumo de energia e baixo custo final.

Os SoCs são compostos por diversos componentes, como processadores, DSPs, módulos de memória e dispositivos de entrada e saída. Esses componentes necessitam de meios para se comunicarem, de forma a realizar troca de dados. A comunicação entre esses componentes é comumente realizada de duas formas principais: por meio de canais ponto-a-ponto dedicados ou utilizando barramentos. A principal vantagem dos canais ponto-a-ponto é que eles proporcionam melhor desempenho e menor latência, pois a comunicação entre dois componentes ocorre independentemente das demais, o que possibilita comunicações paralelas simultâneas. Porém, tal arquitetura de comunicação possui custo de projeto maior, já que necessita de um projeto específico.

O uso de barramentos provê um canal multiponto compartilhado entre todos os elementos do sistema. A grande vantagem de utilizar arquiteturas multiponto está relacionada

ao custo e ao tempo de projeto, pois tal abordagem garante reusabilidade ao sistema. Alternativas como o uso de múltiplos barramentos e hierarquia de barramentos também são utilizadas, porém são limitadas em função da baixa escalabilidade.

Com o aumento na quantidade de componentes em um mesmo sistema em chip o problema recai na comunicação entre esses componentes, uma vez que os SoCs serão tão complexos em um futuro próximo, a ponto de deixar canais ponto-a-ponto inviáveis e canais multiponto com desempenho muito reduzido (Zeferino, 2003) (Sum, Kumar e Jantsch, 2002).

Novas formas de comunicação entre os elementos de um sistema em chip devem ser encontradas de maneira a proporcionar desempenho eficiente a esses sistemas. Nesse âmbito, muitas pesquisas têm sido realizadas visando trazer conceitos de redes de interconexão de multiprocessadores (Dally e Towles, 2003) para tratar a comunicação em chip. São as chamadas redes em chip (NoCs – *Networks on Chip*) (Dally e Towles, 2002) (Benini e De Micheli, 2002), que buscam equilibrar o desempenho de arquiteturas ponto-a-ponto com a reusabilidade de canais multiponto.

Tais redes têm como elementos principais enlaces (*links*), canais por onde trafegam os dados, e chaves roteadoras (chaves *crossbar* ou *crossbar switches*), responsáveis pelo redirecionamento dos dados através da rede para chegar a determinado destino. A forma como as chaves de uma rede de interconexão são interligadas é indica por sua topologia. A escolha da topologia de uma rede é realizada buscando atingir determinadas metas em alguns parâmetros. Em redes em chip, os principais parâmetros são: desempenho (latência e vazão), consumo de energia, escalabilidade e custo de implementação.

O presente trabalho se introduz nesse contexto, apresentando um estudo comparativo de diferentes topologias para NoCs, comparando-as em relação ao desempenho e custo. Um algoritmo semidinâmico para topologias Toróide também é apresentado, visando uma melhora no desempenho.

1.2. Objetivos

A presente dissertação apresenta um estudo comparativo de redes em chip com a utilização de diferentes topologias para mostrar o impacto que estas podem ter no desempenho e na implementação de uma NoC. Tal estudo compara, sob as mesmas condições, topologias bem documentadas na literatura e outras pouco utilizadas, mostrando a diferença de comportamento entre elas e amparando projetistas na escolha de uma topologia de acordo com os requisitos do sistema em chip a ser projetado.

1.3. Organização

O presente trabalho está organizado da seguinte forma. O Capítulo 2 apresenta os conceitos fundamentais de redes em chip, introduzindo suas principais características e a razão da utilização de tais redes. O Capítulo 3 apresenta diversos exemplos de NoCs e suas peculiaridades. Entre elas, a rede Hermes, utilizada como base para este trabalho. O Capítulo 4 apresenta os detalhes de implementação das quatro redes implementadas no trabalho, bem como do algoritmo de roteamento semidinâmico. O Capítulo 5 mostra os resultados obtidos e a comparação de desempenho e custo entre todas as redes implementadas. O Capítulo 6 apresenta as conclusões obtidas com a realização do trabalho e as propostas de trabalhos futuros.

CAPÍTULO

2

Princípios de Rede de Interconexão

A emergente necessidade de troca de dados eficiente em ambientes intrachip faz com que novos paradigmas de comunicação sejam investigados para suprir as limitações apresentadas nas arquiteturas de comunicação até então utilizadas. O uso de redes em chip é a principal solução apontada pela indústria e pela comunidade científica para garantir que a troca de dados entre os diversos elementos de um SoC não prejudique o desempenho do sistema. Nesse contexto, este capítulo aborda os principais fundamentos de redes em chip, incluindo os ambientes em que ocorre essa comunicação e os principais conceitos sobre redes de interconexão.

2.1. Sistemas em Chip

A possibilidade de integração de milhões de transistores em uma mesma pastilha de silício permite o encapsulamento de todos os componentes de um computador em um mesmo circuito integrado (CI). São os sistemas em chip, comumente utilizados em dispositivos portáveis e em sistemas embarcados, pois apresentam baixo consumo de energia, menor custo final, menos espaço requerido e bom desempenho.

Existem vários tipos de hardware de SoCs, cada um utilizado de acordo com a funcionalidade que o sistema requer. Esses sistemas podem ser classificados de acordo com sua versatilidade e domínio de aplicação:

- Multiprocessadores de propósito geral, ou sistemas multicore (Freitas et al., 2006), são chips de alto desempenho caracterizados por apresentarem diversos núcleos de processamento. Esses sistemas são projetados para executar aplicações de diferentes domínios e normalmente apresentam um conjunto de processadores e unidades de armazenamento homogêneos. Isso permite que a carga de trabalho do sistema possa ser balanceada entre os diversos núcleos de processamento existentes;
- SoCs específicos para aplicação (*Application Specific Integrated Circuit* – ASICs) são sistemas encapsulados em um chip dedicados a realização de determinada tarefa. Em muitos casos, como aplicações móveis, a característica de baixo consumo de energia é muito importante. A maioria desses sistemas é programável, porém o domínio das aplicações normalmente é restrito e as características de software e padrões de comunicação são conhecidos;
- Plataformas SoC (Maxiaguine et al., 2004) são sistemas em chip desenvolvidos para uma determinada família de domínio de aplicações. Exemplos dessa classe de SoC são plataformas para telefones GSM ou controle automotivo. Uma plataforma SoC normalmente é versátil, podendo ser utilizada em diferentes sistemas de diversos fabricantes. As unidades de processamento e armazenamento dessa classe de SoC podem diferir em desempenho e funcionamento, o que torna os padrões de tráfego difíceis de prever;
- Por fim, os *Field-Programmable Gate Arrays* (FPGAs) são hardwares nos quais sua funcionalidade é especificada após a fabricação por meio da conexão e configuração dos componentes, que são conectados por redes reprogramáveis. A configuração da rede determina a função implementada pelo sistema. São comumente utilizados para protótipos de circuitos específicos.

A concepção de um SoC é relativamente complexa e envolve pesquisas em nível de hardware e software. Por isso, os SoCs são apropriados para sistemas com grande volume de produção. Os maiores problemas no projeto de um SoC são (Titri et al., 2007): complexidade e tamanho do projeto, padronização de interface entre os componentes do sistema, complexidade de prototipação e gerenciamento de energia.

Nesse sentido, um elevado nível de reusabilidade é necessário para garantir uma alta taxa de produtividade no projeto. Uma abordagem comum é a reutilização de determinados componentes já desenvolvidos, denominados blocos de propriedade intelectual (IP – *Intellectual Property*). Os blocos IPs podem facilitar muito o desenvolvimento de um projeto,

porém também podem causar atrasos, caso seja necessário a alteração do bloco para que seja possível utilizá-lo em um novo projeto. Esses blocos necessitam de uma arquitetura de comunicação que possibilite a troca de dados entre eles de forma a garantir o funcionamento do sistema.

Além disso, exigências de mercado têm aumentado constantemente a complexidade dos SoCs, uma vez que novas funcionalidades, como aplicações multimídias, de telecomunicações e de rádio frequência, vêm sendo adicionadas a esses sistemas, que devem se adequar às tecnologias emergentes. O problema é que em muitos casos, como plataformas e SoCs de aplicação específica, existe uma limitação na frequência de *clock* de seus componentes, pois estes não podem demandar alto consumo de energia. Então, a melhoria de desempenho por meio do aumento da frequência de *clock* não é viável nesses sistemas, uma vez que isso implicaria em um aumento no consumo de energia e redução do tempo de autonomia da bateria. Uma das alternativas para aumentar o desempenho dos SoCs é desenvolver formas de permitir que cada unidade do sistema realize seu processamento paralelamente, de forma independente. E para garantir desempenho e custo de projeto satisfatórios é necessário que a comunicação entre esses componentes seja feita de forma eficiente, porém sem aumento significativo do custo dos componentes utilizados para a fabricação do sistema. Em termos de nomenclatura, assume-se nesta tese, a denominação *núcleo* para qualquer componente que necessite realizar troca de dados com outro componente do sistema e que, consequentemente, está incluído na arquitetura de comunicação do sistema. Processadores, módulos de memória e dispositivos de entrada e saída são exemplos de elementos que se enquadram nessa categoria.

2.2. Exemplos de Sistemas em Chip

O processador Cell (Pham et al., 2006) é um processador de propósito geral desenvolvido em conjunto pelas empresas Sony, Toshiba e IBM. O projeto foi iniciado em 2001 e tinha como objetivo primário o desenvolvimento de um processador para o console Playstation 3 da Sony. A idéia por trás do Cell é baseada em processadores vetoriais, distribuindo a computação entre vários núcleos de processamento independentes e atingindo uma alta frequência de *clock*.

A arquitetura do Cell é composta por um processador Power PC de 64 bits (*Power Processor Element* – PPE), oito elementos sinergísticos de processamento (*Synergistic Processor Elements* – SPE), um barramento de interconexão (*Element Interconnection Bus* –

EIB), memória e estruturas para entrada e saída de dados, como pode ser observado na Figura 2.1.

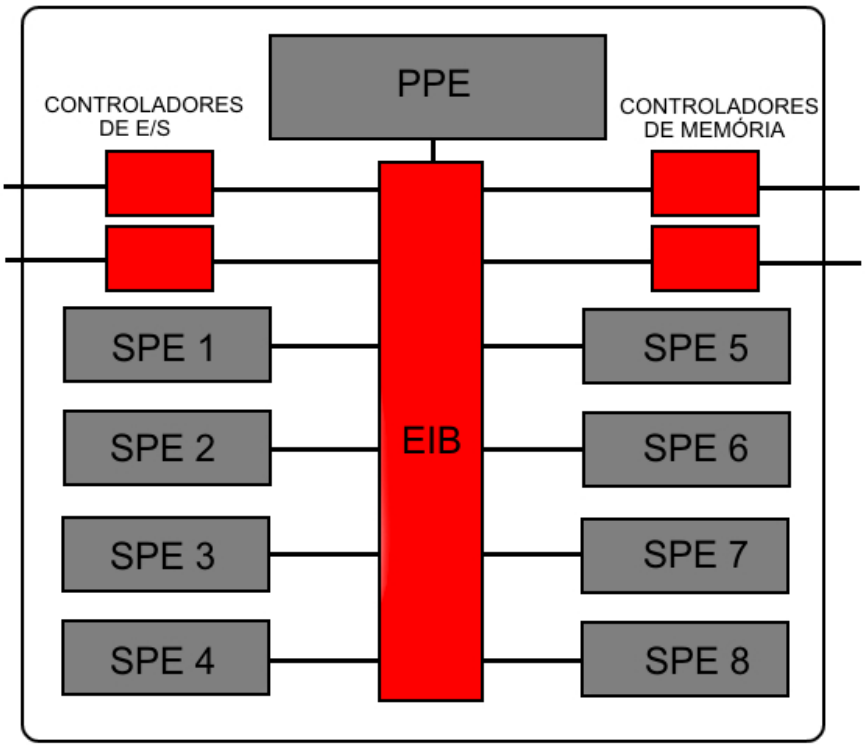


Figura 2.1. Arquitetura do processador Cell

O PPE é um processador SMT (*Simultaneous Multi-Thread*) com duas *threads* de execução. O núcleo possui arquitetura RISC e executa instruções em ordem, mantendo a execução na mesma ordem que é realizada a busca. O PPE é responsável por controlar os SPEs, processadores vetoriais independentes com quatro unidades funcionais de ponto flutuante e quatro unidades funcionais de inteiro cada. A conexão entre o PPE, os SPEs e a interface de controle de memória é realizada pelo EIB, uma rede de interconexão com topologia em anel que possui quatro anéis unidirecionais que interligam todas as portas de dados.

Um outro exemplo de plataforma SoC é o VIPER (Dutta, Jensen e Rieckmann, 2001), desenvolvido pela Philips para aplicações de vídeo e televisão digital. O VIPER possui dois processadores principais, um dele é um processador com arquitetura MIPS RISC de 32 bits e o outro um processador TriMedia VLIW (*Very Long Instruction Word*), também de 32 bits. Somados às CPUs, a plataforma contém coprocessadores multimedia, unidades de entrada e saída e uma arquitetura de barramentos hierárquicos que facilitam o processamento gráfico, de áudio e de vídeo, como pode ser visto na Figura 2.2.

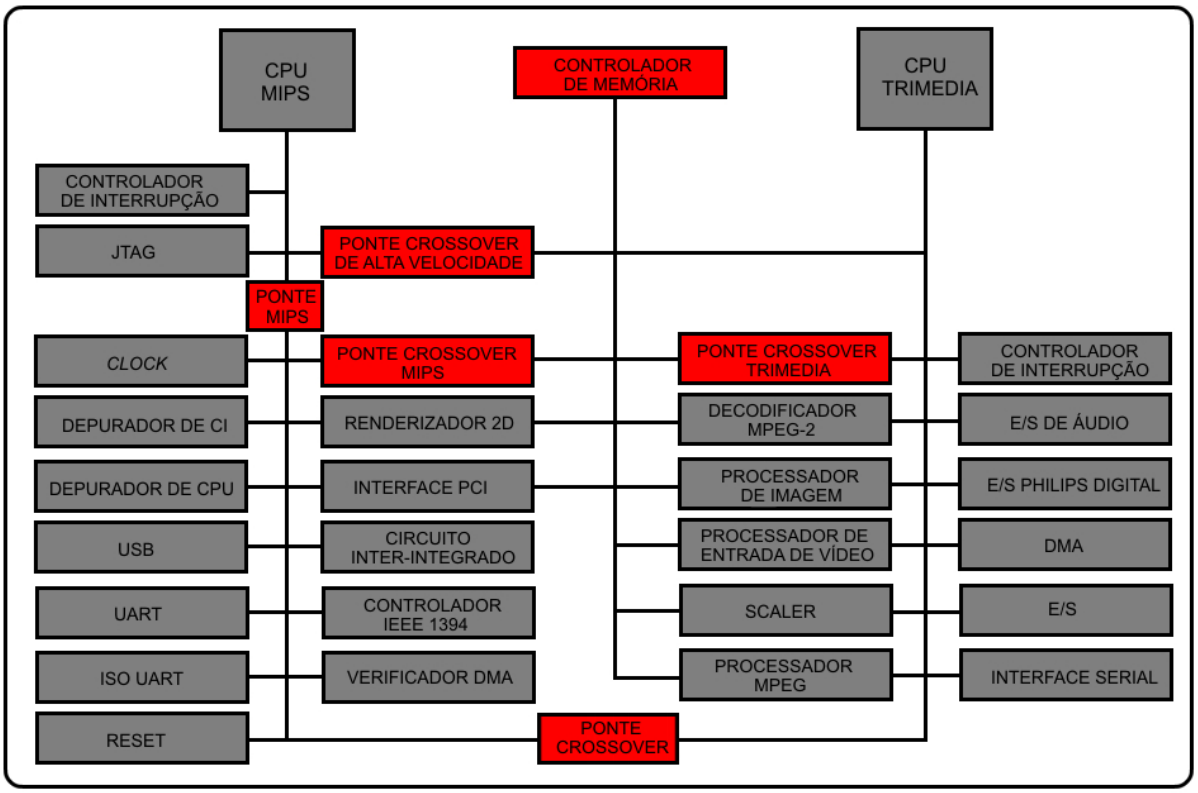


Figura 2.2. Arquitetura da plataforma VIPER

O processador RISC do VIPER é um MIPS PR3940 de 150 MHz, responsável por executar o sistema operacional e diversas tarefas de controle. Já o processador VLIW é um TriMedia TM32, que cuida dos processamentos em tempo real. Além disso, o VIPER possui diversos componentes para otimização de processamento gráfico, como blocos de processamento de vídeo, decodificador de vídeos MPEG-2, processador de entrada de vídeo, entre outros. A comunicação entre cada núcleo do sistema é realizada por um barramento hierárquico dividido em quatro segmentos principais, garantindo menor latência em requisições simultâneas de diferentes componentes do sistema.

Outro exemplo de SoCs são os FPGAs da família Cyclone III da Altera (Altera, 2008a), que são dispositivos programáveis otimizados para proporcionar custo e consumo de energia baixos. Esses dispositivos apresentam milhares de elementos lógicos (*Logic Elements* – LEs) que podem ser configurados para desempenhar uma função específica. A conexão entre os componentes dessa família de FPGA é realizada por uma rede de canais com diferentes velocidades, capazes de otimizar caminhos críticos com a utilização de interconexões mais rápidas.

2.3. Comunicação Intrachip

Os exemplos de SoCs apresentados na Seção 2.2 mostram que tais sistemas apresentam relativa complexidade, incorporando diversos núcleos responsáveis por diferentes operações. Esses sistemas podem inclusive possuir várias unidades de processamento, tais como processadores, microcontroladores, processadores de sinais digitais (*Digital Signal Processors* – DSPs) e processadores gráficos (*Graphics Processing Unit* – GPUs) em um único microchip. Há uma evidente necessidade de comunicação entre esses componentes, uma vez que cada um realiza diferentes atividades do sistema. Normalmente essa comunicação é realizada através de canais ponto-a-ponto ou de canais multiponto, arquiteturas de comunicação apresentadas na Figura 2.3.

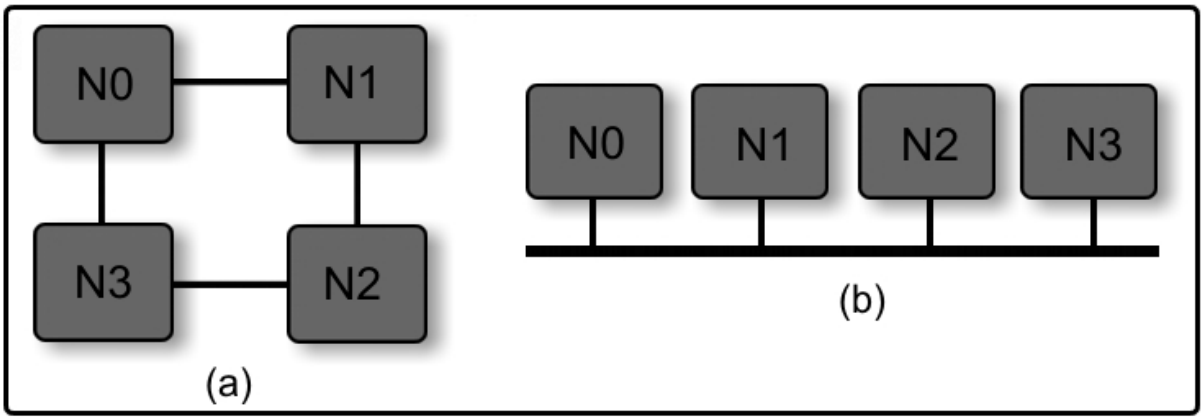


Figura 2.3. Exemplo de arquiteturas de comunicação ponto-a-ponto (a) e multiponto (b)

Na arquitetura ponto-a-ponto, os núcleo são interconectados por meio de canais dedicados. A vantagem dessa forma de comunicação é que, como cada canal interliga dois núcleo diferentes, a comunicação é independente de outros canais, o que possibilita múltiplas conexões simultaneamente. Isso possibilita também que cada canal seja otimizado separadamente, algo que não é possível com a utilização de barramentos.

Além disso, por utilizar conexões mais curtas e, consequentemente, possuir menor carga capacitiva nos fios, essas arquiteturas consomem menos energia, conseguem atingir frequências de *clocks* maiores e permitem menor latência de comunicação (Zeferino e Susin, 2003). O problema dessa abordagem recai no fato dela precisar de um projeto específico e hardware adicional para as portas de comunicação, implicando em um alto custo de projeto e fabricação.

Já nas arquiteturas multiponto, conhecidas como barramentos, o canal de comunicação é compartilhado entre os núcleos do sistema, que disputam o controle e dividem a banda passante do barramento. A principal vantagem desse paradigma de comunicação, que a faz ser

a mais utilizada atualmente (Sum, Kumar e Jantsh, 2002), é a reusabilidade que ela proporciona, uma vez que a mesma estrutura pode ser reutilizada em diferentes sistemas. Essa é uma característica importante, uma vez que o reuso de componentes já projetados e validados garante menor tempo de projeto do sistema.

O baixo custo e os protocolos de controle já estabelecidos fazem com que os barramentos sejam comumente utilizados em SoCs. Entretanto, essa arquitetura de comunicação sofre quanto à escalabilidade, pois sua largura de banda é fixa e a adição de novos núcleos a um barramento faz com que o tempo de resposta diminua, já que a banda do canal também passa a ser compartilhada pelo novo elemento adicionado. Outro problema diz respeito à competição pelo barramento, que pode provocar atrasos no tempo de resposta de determinadas aplicações. Quando múltiplas requisições simultâneas ocorrem no barramento, apenas um núcleo pode ser atendido, o que faz com que outros núcleos com dados prontos para serem enviados fiquem em um estado de espera.

Para contornar as limitações dos sistemas multiponto, as principais soluções encontradas consistem na utilização de múltiplos barramentos (Chen e Sheu, 1991) e de hierarquias de barramentos (IBM, 1999). Os múltiplos barramentos consistem na utilização de diversos canais multiponto, compartilhados entre os núcleos do sistema, como apresentado na Figura 2.4.

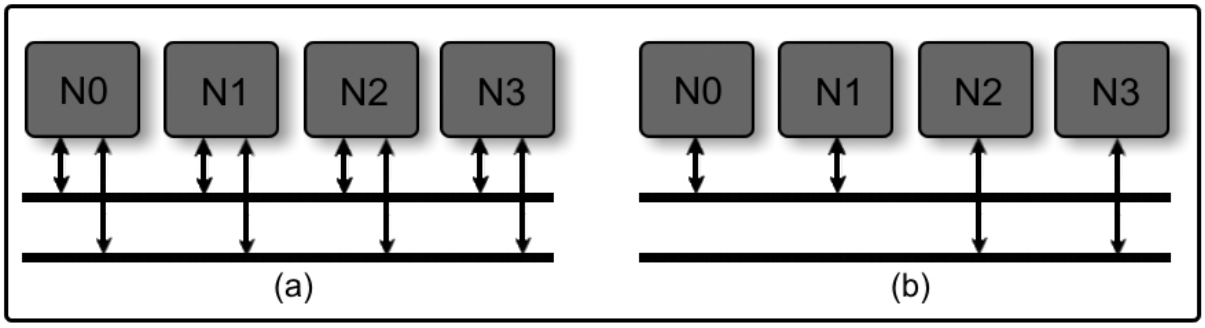


Figura 2.4. Arquitetura de comunicação com múltiplos barramentos: núcleos completamente conectados (a) e núcleos parcialmente conectados (b)

A organização desses canais depende da arquitetura implementada. Os núcleos podem ser conectados a todos os barramentos, como mostra a Figura 2.4(a), o que garante maior disponibilidade de *links* de comunicação. Ou ainda, podem ocorrer conexões parciais, nas quais um determinado número de elementos se conecta a um grupo de barramentos ou a um único barramento, como pode ser observado na Figura 2.4(b). Tais abordagens conseguem aumentar a capacidade de comunicação e largura de banda do sistema.

A abordagem que utiliza hierarquia de barramentos também permite o aumento no número de transações, já que comunicações simultâneas podem ser realizadas com essa arquitetura. Tal sistema consiste no uso de dois ou mais barramentos, com características diferentes, interconectados por um circuito ponte (*bridge*), como mostra a Figura 2.5. A plataforma VIPER, apresentada na Seção 2.2 utiliza tal esquema de comunicação, separando o barramento em áreas diferentes para permitir melhor autonomia de comunicação.

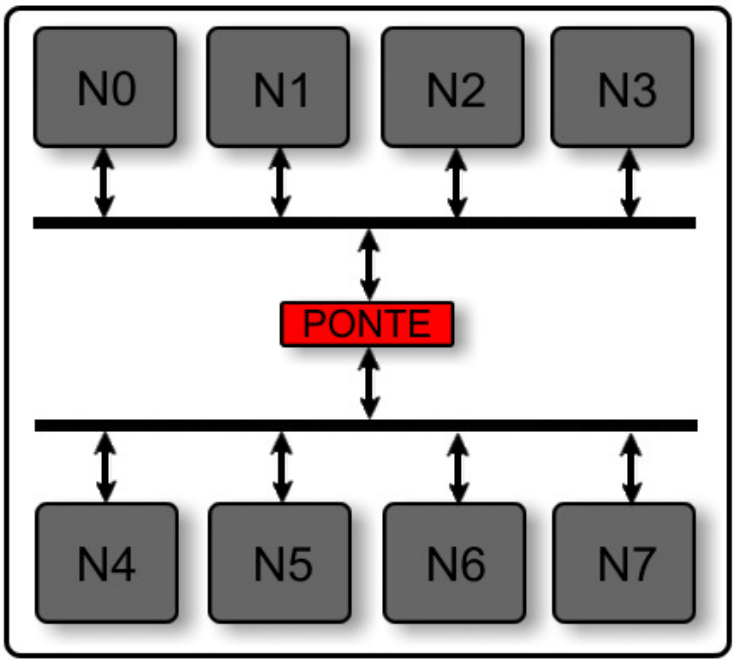


Figura 2.5. Arquitetura de comunicação com hierarquia de barramentos

Assim, componentes mais rápidos podem operar em barramentos com alta velocidade enquanto componentes que operam em frequências mais baixas podem ser conectados a barramentos mais lentos. Isso permite a comunicação paralela dentro de cada subsistema de barramento. Porém, há uma grande perda de desempenho se elementos em barramentos diferentes precisarem se comunicar, já que ambos os barramentos envolvidos na comunicação ficarão ocupados.

O processador Cell buscou uma alternativa diferente das tradicionais. O barramento de interconexão do processador, o EIB, consiste de quatro canais unidirecionais que podem ter até três transações ativas em cada um, totalizando até doze comunicações simultâneas (Ainsworth e Pinkston, 2007). O EIB interliga os nove processadores do Cell, o controlador de memória e duas interfaces de entrada e saída, totalizando doze dispositivos conectados a ele. Dois dos seus canais transportam dados em sentido horário e dois em sentido anti-horário, como apresentado na Figura 2.6.

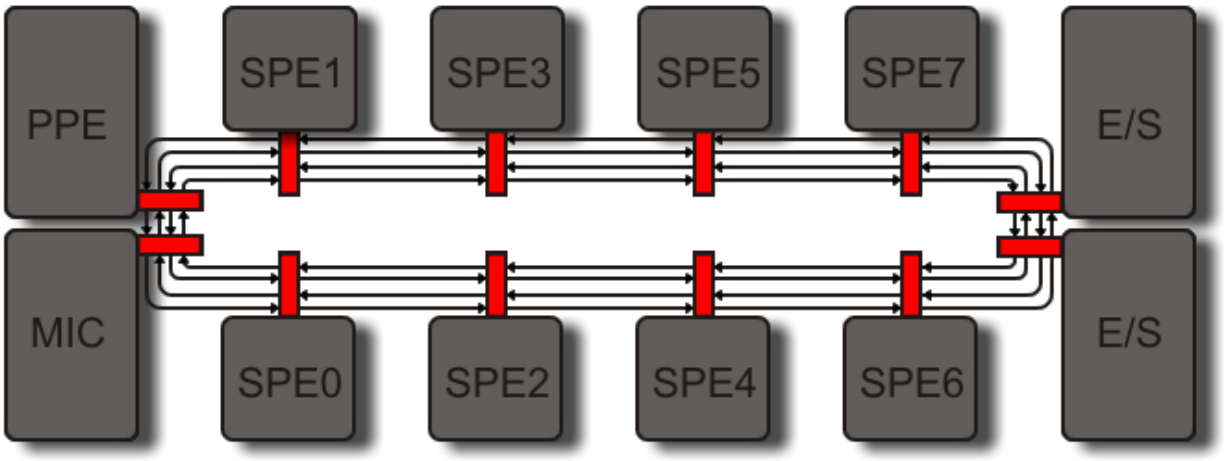


Figura 2.6. Arquitetura de comunicação do processador Cell

Como apresenta núcleos de processamento rápidos, a arquitetura de comunicação do Cell, também deveria ser de alto desempenho. Assim, foram projetados canais com alta largura de banda. O controle de cada canal é concedido por um árbitro central, uma estrutura em forma de estrela que implementa arbitragem *round-robin* com dois níveis de prioridade. A interface de memória possui prioridade maior sobre os outros dispositivos, enquanto os outros componentes recebem acesso igualitário aos canais. Para garantir esse acesso, um barramento de comando gerencia as transações e garante a coerência na distribuição dos acessos não permitindo, por exemplo, a sobreposição de dados enviados em um mesmo canal.

É justamente por causa do barramento de comando que o EIB sofre limitações de latência e largura de banda (Pinkston e Ainsworth, 2007). Isso porque a complexidade do EIB faz com que ele necessite de longas fases de comando e envio que são implementadas justamente neste barramento. Este é compartilhado entre todas as unidades participantes do EIB e provê concorrência limitada, o que aumenta a latência no tráfego de pacotes e reduz a largura de banda efetiva do sistema. Embora os canais de comunicação provenham muita largura de banda, o barramento de comando limita o uso dessa banda. A alta frequência de *clock* que o processador é capaz de atingir faz com que essas restrições não sejam sentidas no desempenho do Cell, porém para sistemas que devem trabalhar em freqüências baixas, como sistemas portáveis, talvez não seja uma arquitetura tão interessante.

Por um lado, têm-se conexões ponto-a-ponto, que maximizam o desempenho no chip, mas sua complexidade de projeto eleva muito o custo de desenvolvimento. Por outro, a utilização de barramentos, alternativa preferencial para comunicação intra-chip, é limitada em relação ao desempenho e escalabilidade, pois perde desempenho com o crescimento do sistema e não oferece paralelismo na comunicação (Lee et al., 2005). Há deficiências mesmo

com o uso de barramentos hierárquicos, os quais exigem um agrupamento específico para cada caso, levando à perda da flexibilidade e generalidade do barramento (Guerrier e Greiner, 2000). Além disso, uma transação entre sub-unidades do sistema de barramentos ocupa diversos recursos do sistema e exige hardware adicional, aumentando o custo do sistema (Zeferino, 2003). O sistema de comunicação do Cell surge como uma alternativa, porém pode não ser tão eficiente em sistemas de menor desempenho, uma vez que possui alta complexidade de implementação e limitações quanto à latência de entrega de pacotes por causa das políticas de controle que o barramento de comando deve seguir.

Portanto, outras formas de comunicação são requeridas para a comunicação intrachip, garantindo performance e escalabilidade para os SoCs. Uma forma de atender tais requisitos é por meio do uso de redes de interconexão chaveadas, trazendo conceitos de interconexão de multiprocessadores para dentro do chip. Tais redes consistem em ligações ponto-a-ponto entre roteadores, que se conectam a núcleos de processamento e aos demais componentes do sistema e atendem aos requisitos de paralelismo na comunicação, alta frequência de operação, escalabilidade e reusabilidade.

2.4. Redes de Interconexão

O surgimento de redes de interconexão de multiprocessadores foi motivado pelo crescimento da complexidade de determinadas aplicações. A busca por tempos computacionais aceitáveis para execução dessas aplicações demanda a exploração de paralelismo na execução.

Tais aplicações podem ser encontradas em diversos domínios. Grama et al. (2003) enumeram alguns desses domínios, citando: (i) aplicações em engenharia como o projeto de máquinas de combustão interna e de micro e nano sistemas; (ii) aplicações científicas, como caracterização funcional de genes e proteínas; e (iii) aplicações em sistemas computacionais, como detecção de invasão em redes, criptografia e tarefas computacionais com grandes bases de dados, que quando divididas e executadas por diversos processadores obtêm resultados mais rápidos.

As redes de interconexão são fundamentais no desempenho de um sistema paralelo, tendo influência na latência, custo e tolerância a falhas do sistema. Elas devem prover desempenho adequado para qualquer aplicação para não causar sobrecarga de dados em algum componente da interconexão.

Com a evolução dos sistemas em chip e o aumento de sua complexidade, essas redes passam a ser utilizadas como sistema de comunicação intrachip, utilizando os mesmos

princípios das redes de interconexão de multiprocessadores, mas apresentando algumas diferenças de projeto, devido a características distintas presentes em sistemas em chip. Quando utilizadas em SoCs, essas redes são denominadas Redes em Chip (*Network on Chip* – NoC).

Entre as diferenças de projeto, deve-se garantir uma baixa latência de comunicação, de forma que a comunicação não consuma muitos ciclos de *clock*. Isso porque as restrições de consumo de energia de determinados sistemas em chip não permitem que eles atinjam altas frequências de *clock*. Esse consumo deve ser baixo, pois muitas vezes esses sistemas são sustentados por baterias que devem possuir o maior tempo possível de autonomia.

Outra diferença que uma rede em chip pode ter em relação a uma rede de interconexão para multiprocessadores, é que muitas vezes o sistema em que a NoC está inserida possui um domínio de aplicações muito pequeno, sendo específico para determinada atividade, ao contrário de redes para multiprocessadores que dão ênfase à comunicação de propósito geral. Assim, para obter melhor desempenho em um SoC específico para uma determinada aplicação é necessário mapear os padrões de comunicação e projetar a rede de forma a otimizar os caminhos críticos da troca de mensagens durante a execução da aplicação.

Redes de interconexão são caracterizadas de acordo com sua topologia e seus protocolos. A topologia indica a estrutura na qual seus roteadores são interligados, enquanto os protocolos definem a forma como serão efetuadas as transferências de dados pela rede. A escolha da topologia e dos protocolos de uma rede são realizadas buscando atingir determinadas metas em alguns parâmetros. Em redes em chip, os principais parâmetros são: desempenho (latência e vazão), consumo de energia, escalabilidade e custo de implementação (De Micheli e Benini, 2006).

As redes de interconexão podem ser classificadas em dois grupos principais, de acordo com sua topologia. Redes diretas, ou estáticas, mantêm conexões fixas (ponto-a-ponto) entre os núcleos da rede, que são diretamente conectados por meio de roteadores. Já nas redes indiretas, ou dinâmicas, as conexões passam por chaves roteadoras que indicam para qual caminho o dado será enviado. Enquanto nas redes diretas um canal conecta dois roteadores que estão diretamente associados à seus respectivos núcleos, nas redes indiretas podem existir canais que interligam fases de roteadores, sem que estes estejam diretamente ligados a qualquer núcleo.

2.4.1 Redes de Interconexão Diretas

As redes de interconexão diretas mantêm um roteador diretamente associado a cada núcleo da rede. Os roteadores mantêm ligações fixas, ponto-a-ponto, para uma determinada quantidade de roteadores vizinhos, também associados a outros núcleos. Dessa forma, se dois núcleos não adjacentes precisarem se comunicar, será necessária a passagem da mensagem por roteadores intermediários até que a mensagem chegue ao seu destino.

Os canais que conectam os roteadores podem ser unidirecionais ou bidirecionais. No primeiro caso, o tráfego possui um fluxo único, partindo de um roteador A até um roteador B, não sendo possível trafegar dados na direção oposta, de B até A. Já nos canais bidirecionais isso é possível, um roteador pode utilizar o mesmo canal tanto para envio quanto para recebimento de dados.

Em comparação aos sistemas de barramento, as redes diretas possuem vantagem em relação à escalabilidade. Se um novo núcleo for adicionado ao sistema, ele não utilizará parte da largura de banda do sistema, como nos barramentos, mas aumentará a largura da banda total do sistema com a adição de novos canais de comunicação.

Existem algumas propriedades básicas dessas redes que são utilizadas para defini-las. Denomina-se nodo o par roteador/núcleo. O grau de um nodo é o número de canais que conectam o nodo aos seus vizinhos. O diâmetro da rede é a distância máxima entre dois nodos. Uma rede pode ser dita regular, quando todos os nodos possuem o mesmo grau. A Figura 2.7 apresenta exemplos de redes de interconexão diretas.

A mais simples é a rede Linear (Figura 2.7a), na qual todos os nodos, exceto os nodos extremos, são conectados à direita e à esquerda por uma conexão simples. Os pacotes enviados são repetidos de nodo a nodo até chegar ao seu destino. Conectando-se as extremidades da rede linear tem-se uma rede Anel (Figura 2.7b), na qual a comunicação ocorre da mesma maneira que na rede Linear, porém de forma mais eficiente já que o espaço entre os nodos extremos não existe mais e a distância entre dois nodos qualquer é, no máximo, igual à distância destes nodos em uma topologia linear. Na rede Grelha (Figura 2.7c), um nodo se conecta aos seus nodos adjacentes, disponibilizados em forma de grelha. A interligação dos nodos extremos em uma rede Grelha resulta em uma rede Toróide, ou Toro, (Figura 2.7d). As redes Hipercubo apresentam um nodo interligado a outro dependendo do grau da rede. Na Figura 2.7e pode ser observada uma rede Hipercubo de grau 3, na qual cada nodo é interligado a outros três. Na rede Totalmente Conectada (Figura 2.7f) cada nodo é conectado a todos os outros. Essa seria a rede ideal, pois garante entrega rápida das

informações, que não passam por nodos intermediários. Porém, a rede possui custo elevado, tanto de projeto, quanto para a adição de novos nodos na rede. Já nas redes em Árvore Binária (Figura 2.7g) cada nodo interliga outros dois, existindo ainda um nodo raiz no topo da hierarquia.

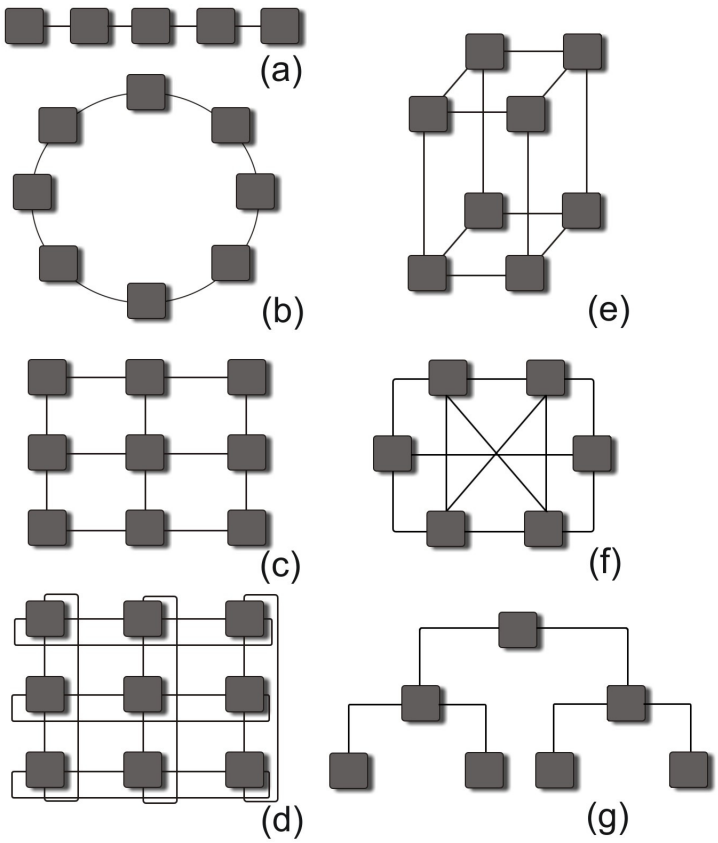


Figura 2.7. Topologias de redes de interconexão diretas: Rede Linear (a), Rede Anel (b), Rede Grelha (c), Rede Toróide (d) Rede Cubo de Grau 3 (e), Rede Completamente Conectada (f) e Rede em Árvore (g)

2.4.2 Redes de Interconexão Indiretas

As redes indiretas utilizam estágios de chaves roteadoras para conectar os núcleos de um sistema. Esses roteadores, também denominados *switches* ou *crossbar switches*, são os elementos básicos de chaveamento de uma rede de interconexão. Eles são apenas responsáveis pelo fornecimento de um caminho para a conexão entre dois nodos da rede. Assim como as redes diretas, as redes indiretas podem assumir diferentes topologias. Existem três topologias de interconexão básicas para redes dinâmicas: *Crossbar*, Estágio Único e Multiestágio.

As redes *Crossbar* são muito rápidas, pois permitem conexões simultâneas entre todas as suas entradas e saídas. Todavia, seu custo é elevado, pois requer uma chave *crossbar* em cada uma das interconexões da rede. Supondo uma rede que conecta processadores a módulos de memória, para *p* processadores e *m* módulos de memória são necessárias *p x m* chaves, como pode ser observado na Figura 2.8, que apresenta uma rede *crossbar* 4x4, com 4 processadores e 4 módulos de memória. Neste exemplo são necessários 16 *switches* para a constituição da rede. Isso a torna uma rede com baixa escalabilidade, já que seu custo de área cresce com uma proporção de N^2. A adição de um novo processador na rede, por exemplo, implicaria no acréscimo de mais quatro roteadores.

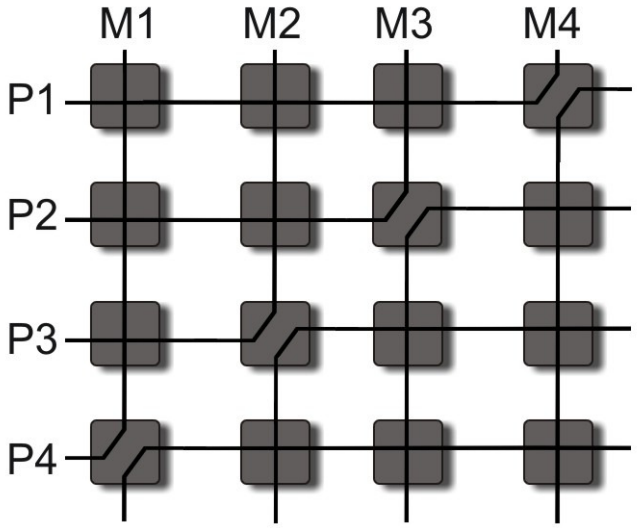


Figura 2.8. Rede Crossbar 4x4

Nas redes de Estágio Único existe apenas um único estágio de chaves *crossbar* entre as entradas e saídas da rede, enquanto que na rede Multiestágio existem diversas fases de roteadores. Na rede de estágio único o dado circula um número de vezes pela rede até conseguir estabelecer a comunicação entre a entrada e a saída. Isso é feito utilizando um padrão de conexão. Dessa forma, o dado deverá circular pela rede através da execução das operações até encontrar o destino, sendo que o *switch* é responsável pelo redirecionamento do dado para o próximo destino em cada fase de execução.

Já na rede Multiestágio os padrões de conexão são utilizados para conectar um estágio de roteadores a outro. Na Figura 2.9 pode ser observada uma rede de interconexão Multiestágio 8x8 que apresenta o padrão *Shuffle-Exchange*. Ela utiliza três estágios de chaves *crossbar* 2x2, com duas entradas e duas saídas, para interligar as entradas às saídas da rede.

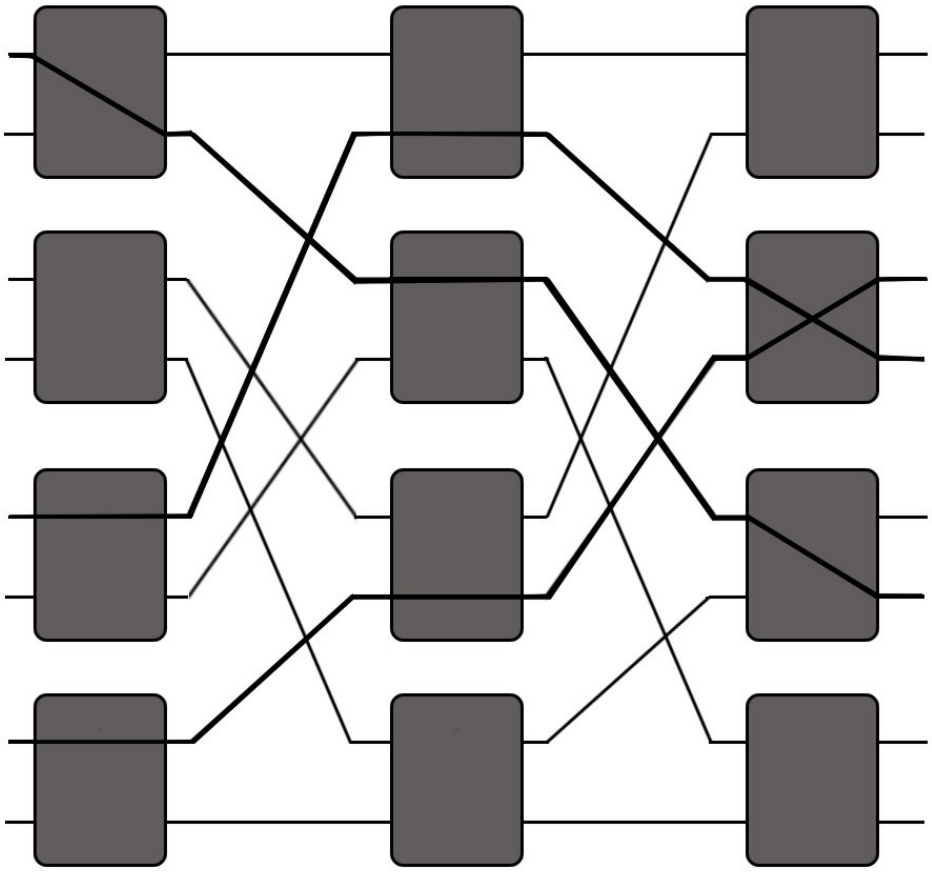


Figura 2.9. Rede Multiestágio Shuffle-Exchange 8x8

A Figura 2.9 mostra ainda como um determinado número de caminhos pode ser estabelecido simultaneamente pela rede com a modificação das configurações dos roteadores. No caso, três caminhos são utilizados simultaneamente. É possível notar que um único *switch* pode mapear dois dados de entradas diferentes ao mesmo tempo, desde que as saídas requisitadas sejam distintas.

A configuração que o roteador irá assumir depende sempre do endereço de destino da mensagem. Cada bit do endereço de destino é utilizado para rotear a mensagem por um estágio. O bit mais a esquerda do endereço de destino é utilizado para rotear a mensagem pelo primeiro estágio, o segundo bit no segundo estágio e assim por diante. Para rotear a mensagem, verifica-se o bit do endereço de destino que controla o roteamento do estágio. Se esse bit for zero, a mensagem é roteada para a saída superior da chave roteador, já se o bit for um, a mensagem é roteada para a saída inferior.

As redes indiretas são a base para a comunicação em FPGAs (Kilts, 2006). Isso porque os FPGAs possuem um grande número de elementos computacionais dinamicamente configuráveis que devem suportar padrões de comunicação complexos e reprogramáveis.

Com exceção dos FPGAs, a maioria das propostas de implementação de arquiteturas de comunicação em sistemas em chip é de redes diretas com uma topologia ortogonal (De Micheli e Benini, 2006), na qual é possível organizar os nodos em um espaço ortogonal de dimensão *n* de forma que cada *link* produza um deslocamento em uma única dimensão. Exemplos de redes ortogonais são as redes Grelha e Toróide. Essas redes são normalmente utilizadas, pois o roteamento é simples e pode ser eficientemente implementado em hardware.

Além da topologia, existem diversos fatores que influenciam no desempenho de uma rede de interconexão. São os casos da largura de banda dos canais, da estratégia de *switching*, do mecanismo de controle de fluxo e dos próprios roteadores.

2.4.3 Canais

Os canais (*links* ou enlaces) entre os nodos consistem de fios ou fibras ópticas que transportam um sinal analógico de um nodo da rede até outro. Um canal é caracterizado por fatores como largura (*w*), frequência (*f*), latência e tempo requerido para um bit ir da fonte *x* até o destino *y* (Dally e Towles, 2003). A latência de um *link* é determinada por seu comprimento e pela velocidade de propagação dos dados. Já a largura de banda de um canal (*b*) é medida através da equação $b = w * f$. Usualmente todos os canais possuem a mesma largura de banda.

2.4.4 Estratégia de *Switching*

A estratégia de *switching* determina como uma mensagem atravessa a rede. Na técnica de *Circuit Switching*, o caminho de uma mensagem saindo da fonte até seu destino é estabelecido antes da comunicação ser realizada (Ould-Khaoua e Min, 2001). O caminho é reservado e só então a comunicação é estabelecida. Dessa forma, enquanto houver comunicação a rota permanecerá reservada.

Para o estabelecimento de uma rota, a fonte envia um cabeçalho *flit* através da rede. O *flit*, ou unidade de controle de fluxo (*flow control unit*), é a menor unidade de informação que pode ser transferida através de um canal, sendo que o cabeçalho *flit* de um pacote contém o seu destino (Peh e Dally, 2000).

Quando um cabeçalho chega a um roteador, o caminho anteriormente percorrido por esse cabeçalho é reservado. Após a chegada do cabeçalho ao seu destino, um *flit* de reconhecimento é enviado de volta à fonte, garantindo o estabelecimento da rota completa e

permitindo assim, a inicialização da comunicação entre os núcleos. Porém, caso o cabeçalho tente passar por um canal que já está reservado, ele envia um sinal de volta a origem, desbloqueando os *links* anteriormente reservados por ele.

As vantagens dessa técnica são a possibilidade da entrega de mensagens com uma latência garantida, uma vez que a conexão tenha sido estabelecida (Ould-Khaoua e Min, 2001) e a facilidade de implementação (Dally e Towles, 2003).

Outra estratégia de *switching* muito utilizada é a *Packet Switching*, na qual uma mensagem é dividida em vários pacotes, que são roteados individualmente da fonte até o destino (Tripathi e Lipovski, 1979). Além de dados, os pacotes contêm a rota que devem seguir e um número de sequência para a remontagem da mensagem após todos os pacotes chegarem ao seu destino. Tal estratégia necessita de mais alguns recursos por parte da chave *crossbar*, que geralmente contém *buffers* para armazenar os pacotes que chegam e os que devem sair. *Packet Switching* permite melhor aproveitamento dos recursos da rede, já que os *buffers* e canais são ocupados somente quando um pacote os está percorrendo, desalocando-o no momento em que deixam de usá-los (Kumar, Dias e Jump, 1983). O *Packet Switching* pode ser subdividido em submétodos, de acordo com a forma que os pacotes são transmitidos na rede.

A técnica mais simples é denominada *Store-and-Forward* (Gerla e Kleinrock, 1980). Nela, ao receber um pacote, o roteador primeiramente o armazena em seu *buffer* para então decidir o que fazer com ele. É verificado se o canal por ele requisitado está sendo utilizado. Em caso negativo o pacote é então enviado ao seu próximo destino.

Um método alternativo ao *Store-and-Forward* é o *Virtual Cut-Through* (Kermani e Kleinrock, 1979). Nesse método, os pacotes que estão chegando a um nodo intermediário são automaticamente roteados para o próximo nodo desde que seja possível a conexão entre os nodos. Em outras palavras, o pacote só será armazenado no *buffer* caso o canal ou o *buffer* da próxima chave *crossbar* estejam ocupados. Implementada no final dos anos 80 com o barateamento dos circuitos integrados, tal técnica apresenta um ganho de desempenho quanto à latência da rede, uma vez que, no pior caso, ela terá a mesma performance da técnica *Store-and-Forward*.

Outra técnica é a *Wormhole* (Dally e Seitz, 1987). Semelhante à *Virtual Cut-Throught*, tal técnica difere ao não armazenar o pacote inteiro em *buffers*, armazenando apenas alguns *flits*. Os pacotes são divididos em *flits*, sendo o cabeçalho enviado antes do restante do pacote. Ao chegar a um roteador, o cabeçalho *flit* é enviado para sua rota especificada e todos os *flits* restantes do pacote seguem-no. Para que isso possa ocorrer, três condições devem ser

satisfeitas: (i) o *link* até o próximo nodo deve estar disponível, (ii) deve haver espaço para o cabeçalho *flit* no *buffer* da próxima *crossbar switch* e (iii) um *flit* de largura de banda do canal deve estar disponível. Caso o cabeçalho chegue a um nodo e uma dessas condições não seja satisfeita ele é armazenado em um *buffer* e fica bloqueado. Dessa forma o resto do pacote para de avançar e fica retido no canal, bloqueando-o. Em relação ao *Virtual Vut-Throught*, o *Wormhole* utiliza melhor os espaços reservados para *buffer*, já que os *buffers* são utilizados apenas para armazenamento de *flits*.

Esse bloqueio da rede provocado pelo *Wormhole* pode ser evitado com a utilização de canais virtuais (Dally, 1992). Os canais virtuais permitem que diversos pacotes utilizem o mesmo canal físico, aumentando significativamente o desempenho da rede (Silla e Duato, 2000).

Um canal virtual é um *buffer* capaz de manter *flits* ou pacotes associados a um *link* físico. Diversos canais virtuais podem estar associados a um único canal físico. Assim, se um pacote *A* está bloqueado e ocupa o canal virtual *C1*, um outro pacote *B* pode vir pelo mesmo *link*, mas através do canal virtual *C2*. Dally e Towles (2003) comparam os canais virtuais a pistas de uma estrada: uma rede de interconexão sem canais virtuais seria como uma estrada com uma única pista, enquanto um *link* com dois canais virtuais seria como uma estrada com pista dupla. A adição de canais virtuais não aumenta o número de *links* ou *switches* de uma rede, mas apenas divide a largura de banda de um *link* pelo número de canais virtuais implementados.

2.4.5 Mecanismo de Controle de Fluxo

O mecanismo de controle de fluxo determina quando uma mensagem ou pacote segue sua rota. Esse controle é necessário quando duas ou mais mensagens tentam utilizar um mesmo recurso na rede. Por exemplo, quando tentam utilizar o mesmo canal simultaneamente.

A forma mais simples e de menor custo para realizar o controle de fluxo é através da não utilização de *buffers* (Kruskal e Snir, 1983). Nesse caso, deve existir alguma maneira arbitrária de escolher qual pacote irá seguir pela rede. O mecanismo deve ainda especificar o que fazer com o pacote "perdedor", pois como não existe *buffer* ele não pode ficar esperando o canal ser desocupado. Assim, o pacote deve ser descartado ou redirecionado para uma outra saída. A segunda opção nem sempre é possível, uma vez que, dependendo da topologia da rede, só existe uma rota possível de uma fonte até um destino. Já se o pacote for descartado,

então ele deve ser retransmitido pela fonte, o que não é uma boa alternativa, já que os recursos gastos com tal pacote até enão deverão ser gastos novamente.

A adição de *buffers* nas entradas e/ou saídas de uma chave roteadora evita o problema de um pacote ter que ser descartado, já que poderá ficar armazenado enquanto espera a desocupação do canal desejado. Assim, ao chegar a um nodo intermediário, o pacote é armazenado em um *buffer* até que os recursos necessários sejam liberados. Existem três tipos principais de mecanismos de controle de fluxo (Dally e Towles, 2003): Baseado em Créditos, *Slack Buffer* e *Ack*/*Nack*.

O mecanismo de controle de fluxo Baseado em Créditos (Cozzani, Giordano, Pagano e Russo, 1996) permite a transmissão de dados apenas se a rede possuir recursos suficientes (espaço em *buffer* e capacidade nos canais) para que a comunicação ocorra. Tal abordagem funciona sobre cada canal que conecta dois nodos, um nodo transmissor e outro receptor. Cada nodo deve possuir *buffers* separados para cada conexão. Assim, a transmissão é controlada por créditos gerados pelo receptor, que previnem seu *buffer* de receber mais dados do que o suportado. Antes de enviar dados pelo canal, o transmissor deve ser habilitado pelo receptor, que envia os créditos que podem ser utilizados. Esses créditos refletem a capacidade de seu *buffer*. Cada vez que dados são enviados pelo transmissor, os créditos são decrementados. Caso os créditos cheguem a zero, dados não poderão ser mais transmitidos até que o nodo receptor utilize os recursos recebidos, liberando espaço em seu *buffer* e incrementando a contagem dos créditos novamente. Essa abordagem não descarta dados, uma vez que eles só serão enviados caso a comunicação possa ser estabelecida.

Já no controle de fluxo baseado em *Slack Buffer* (Nishimura et al., 2000), também conhecido como controle de fluxo *On/Off* ou *Stop and Go*, é mantido um bit de controle no *buffer* do nodo receptor que indica se o nodo pode ou não receber dados. Um limite máximo e mínimo de capacidade de dados são estabelecidos em cada *buffer*. Caso o *buffer* atinja seu limite máximo um sinal *off* (ou *stop*) é enviado ao nodo transmissor, indicando que o *buffer* está cheio e não pode mais receber dados. O sinal se alterará para *on* (ou *go*) quando o *buffer* atingir o limite mínimo, o que indica que ele está vazio e, consequentemente, apto a receber dados.

Já o mecanismo de controle de fluxo *Ack/Nack* (Niswar e Thamrin, 2005), ou *Handshake*, não faz controle do espaço utilizado no *buffer* do nodo receptor. Assim que o dado está disponível para envio, o nodo transmissor envia um cópia de seus dados. Caso exista espaço no roteador receptor, este envia um sinal de confirmação ao transmissor (*Ack*), que descarta sua cópia dos dados. Porém, se não houver espaço no nodo receptor, ele enviará

um sinal que o dado não pode ser recebido (*Nack*) e o transmissor deve reenviá-lo. O transmissor realiza o reenvio até receber um sinal *Ack* como resposta. Apesar de ser de implementação mais simples, essa retransmissão dos dados descartados ocasiona uso ineficiente da largura de banda da rede.

2.4.6 Roteadores

Responsáveis por determinar a direção dos dados em uma rede de interconexão, as chaves roteadoras também podem realizar outras funções como armazenamento de dados, quando uma porta de saída requisitada está ocupada, e o redirecionamento do roteamento para aliviar congestionamento em redes que permitam tal mecanismo.

A cada nodo da rede, um roteador deve ser associado junto com um endereço que deve identificar de forma única esse nodo. Assim, quando um nodo deseja enviar determinado dado, ele deve conhecer o endereço do nodo destino para que os dados possam se enviados pela rede. Caso receba um dado não endereçado a si, um roteador verifica esse endereço para encaminhar o dado para a direção correta. Essa direção é determinada de acordo com um algoritmo de roteamento implementado no roteador.

Os roteadores geralmente consistem de um conjunto de portas de entrada, um conjunto de portas de saída, uma rede interna conectando cada entrada a todas as saídas, *buffers* internos e lógica de controle para a conexão de cada ponto (Culler e Singh, 1999). Em cada porta de entrada existe um receptor e em cada porta de saída um transmissor. Além disso, podem existir *buffers* nas portas de entrada e nas portas de saída. A Figura 2.10 mostra a organização geral de uma *crossbar switch* 2x2.

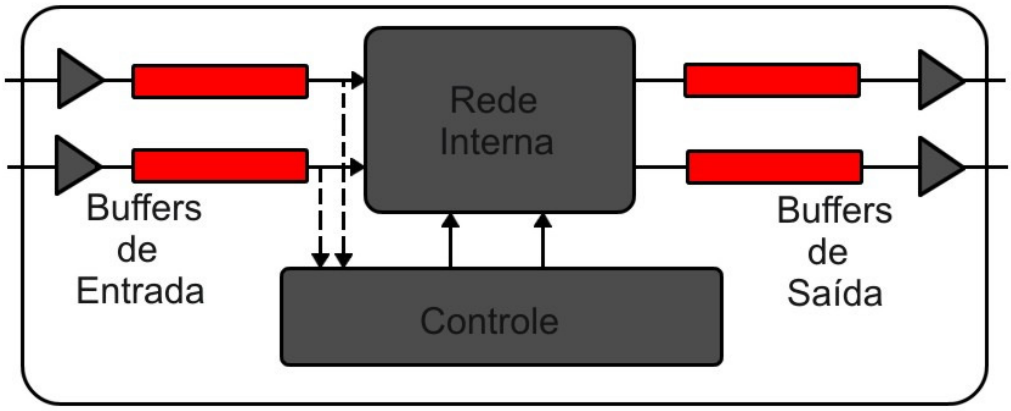


Figura 2.10. Organização básica de um roteador

A rede interna da chave é responsável pela conectividade entre cada porta de saída e entrada. Ela redireciona os dados de entrada para a porta de saída desejada de forma realizar a comunicação entre dois nodos diferentes. Normalmente, a rede interna é projetada de forma regular, para diminuir o caminho dos canais de comunicação e reduzir o atraso das transmissões.

Buffers usualmente são utilizados para o armazenamento de dados que não podem ser imediatamente roteados. O maior problema das redes de interconexão em chip é que os *buffers* possuem alto custo quanto ao consumo de energia (De Micheli e Benini, 2006), assim, em geral, implementações de NoCs têm limitações quanto ao armazenamento.

O uso de *buffers* nos roteadores pode ocorrer de cinco formas distintas: utilizar *buffers* somente nas portas de entrada, somente nas portas de saída, em ambas, não utilizar *buffers* ou utilizar um *buffer* central compartilhado.

A utilização de *buffers* somente em portas de entrada ou saída ou em ambas, depende do método de controle de fluxo escolhido. No método por créditos, por exemplo, são necessários *buffers* em ambas as portas enquanto nos outros métodos apresentados anteriormente são necessários *buffers* somente nas portas de entrada ou saída.

A não utilização de *buffers* faz com que o roteador apenas repasse as mensagens que chegam sem qualquer tipo de controle. O problema ocorre quando dois ou mais pacotes tentam seguir pelo mesmo canal de saída. Nesse caso, o *switch* poderá rotear apenas um dos pacotes e o outro poderá ser descartado ou roteado para outra saída, se a rede possibilitar que isso seja feito.

Já com a utilização de *buffers,* a organização deles também é importante. A maneira mais simples de implementação de um *buffer* é a utilização de *buffers* FIFO (*First In, First Out*). Nessa organização, os *buffers* são lidos na ordem em que recebem dados. Um atraso na comunicação ocorre caso a primeira posição do *buffer* seja ocupada por um pacote que não pode ser imediatamente roteado. Assim, os outros pacotes do *buffer* também ficarão bloqueados, mesmo se o *link* que desejam utilizar estiver desocupado. Uma forma de contornar esse problema é a utilização de *buffers Statically Allocated Multi-Queue* (SAMQ) e *Dynamically Allocated Multi-Queue* (DAMQ), que mantêm portas de leitura para todas as posições do *buffer* e elas são mutiplexadas de acordo com a disponibilidade do canal de saída. A melhora no desempenho com *buffers* SAMQ e DAMQ é obtida com um aumento no custo de implementação do *buffer*, que terá um maior número de portas de leitura. As restrições de custo da rede devem ser levadas em conta nesse caso, uma vez que muitos *buffers* normalmente são utilizados nas implantações de redes em chip.

Outra forma de implentação de *buffers* é a utilização de um *buffer* central compartilhado entre as portas de entrada e saída. Tal implementação denominada *Centrally-Buffered, Dynamically-Allocated* (CBDA) deve possuir no mínimo 2 * *n* posições no *buffer* para possibilitar a leitura simultânea de *n* portas de entrada e *n* portas de saída (Tamir e Frazier, 1992).

Já nos roteadores que utilizam a organização *Output Queued* (Mneimneh, Sharma e Siu, 2001), os *buffers* são ligados às saídas das chaves. Assim, quando um pacote chega, ele é alocado diretamente no *buffer* de sua saída. Dessa forma, a porta de saída envia os dados contidos em seu *buffer*, que pode ser uma simples fila FIFO, sem precisar procurar os pacotes nos *buffers* de entrada. O problema consiste na complexidade de implementação, uma vez que é necessário um controle de fluxo interno entre as portas de entrada e saída do roteador (Zeferino, 2003). O ganho de desempenho, porém pode ser significativo (Gonçalves et al, 2007).

2.4.7 Algoritmo de Roteamento

O algoritmo de roteamento da rede indica para qual caminho o roteador deve encaminhar um pacote recebido. O roteamento é realizado com base no endereço de destino do pacote. Os algoritmos de roteamento devem garantir a entrega dos pacotes ao seu destino evitando três problemas principais: *deadlock*, *livelock* e *starvation*.

O *deadlock* ocorre quando há uma dependência cíclica entre roteadores requisitando determinado recurso da rede, de forma que nenhum deles consiga progresso algum. Já o *livelock* ocorre quando pacotes ficam circulando permanentemente na rede sem conseguir chegar a seu destino. Esse problema acontece em algoritmos adaptativos que utilizam caminhos não-minímos para rotear os pacotes e pode ser evitado com estratégias de restrição de desvios no roteamento. O *starvation* ocorre quando uma porta requisita um recurso da rede, mas nunca é atendida por possuir baixa prioridade. Esse problema pode ser evitado utilizando políticas de arbitragem eficientes que garantam que mais cedo ou mais tarde todos os recursos requisitados serão atendidos.

Existem diversas taxonomias para classificar os algoritmos de roteamento. Essas taxonomias são baseadas nos seguintes critérios: (i) número de destinatário de cada pacote; (*ii*) local de decisão do roteamento; (iii) implementação; (iv) adaptabilidade; (v) progressividade; (vi) minimalidade; e (vii) número de caminhos (Duato, Yalamanchili e Ni, 1997).

De acordo com o número de destinos de cada pacote, um algoritmo de roteamento pode ser classificado em *unicast*, quando os pacotes possuem um único destino, ou *mulitcast*, quando os pacotes podem ser entregues para mais de um destino.

Considerando o local no qual a decisão de roteamento é tomada, o roteamento pode ser determinado por um controlador central na rede (roteamento centralizado), pelo roteador de origem (roteamento na origem) ou as decisões por ser tomadas por cada roteador, conforme eles recebem os pacotes (roteamento distribuído). Um esquema híbrido também é possível, denominado multifase. Nessa abordagem, o roteador de origem calcula os nodos de destino que poderão receber os pacotes a serem enviados, mas o caminho é traçado de forma distribuída.

Os algoritmos de roteamento podem ser implementados de diferentes formas. Os algoritmos podem ser baseados em tabelas, realizando consultas em uma tabela de rotas para determinar o caminho ou implementados em hardware ou software, de acordo com uma máquina finita de estados.

Em relação à adaptabilidade, os algoritmos podem ser classificados como determinísticos ou adaptativos. Algoritmos determinísticos sempre provêm o mesmo caminho para determinado par de nodos. Já os algoritmos adaptativos utilizam informações do tráfego da rede ou condições dos canais para evitar regiões congestionadas ou com falhas da rede.

Algoritmos adaptativos ainda podem ser classificados de acordo com sua progressividade. Algoritmos progressivos sempre enviam os cabeçalhos dos pacotes adiante, reservando um novo canal a cada operação de roteamento. Já algoritmos regressivos permitem também que os cabeçalhos possam retornar pela rede, desalocando canais previamente reservados.

Em níveis mais baixos, o roteamento pode ser classificado de acordo com sua minimalidade, como mínimos ou não-mínimos. Algoritmos mínimos apenas enviam os pacotes por meio de canais que os deixam mais próximos de seu destino. Já algoritmos não-mínimos podem enviar os pacotes para canais que os afastem de seu destino.

Finalmente, em relação ao número de caminhos, um algoritmo adaptativo pode ser classificado como completo, se puder utilizar todos os caminhos disponíveis, ou parcial, se apenas um subconjunto desses caminhos puder ser considerado. A Figura 2.11 apresenta as classificações dos algoritmos de roteamento e as relações entre elas.

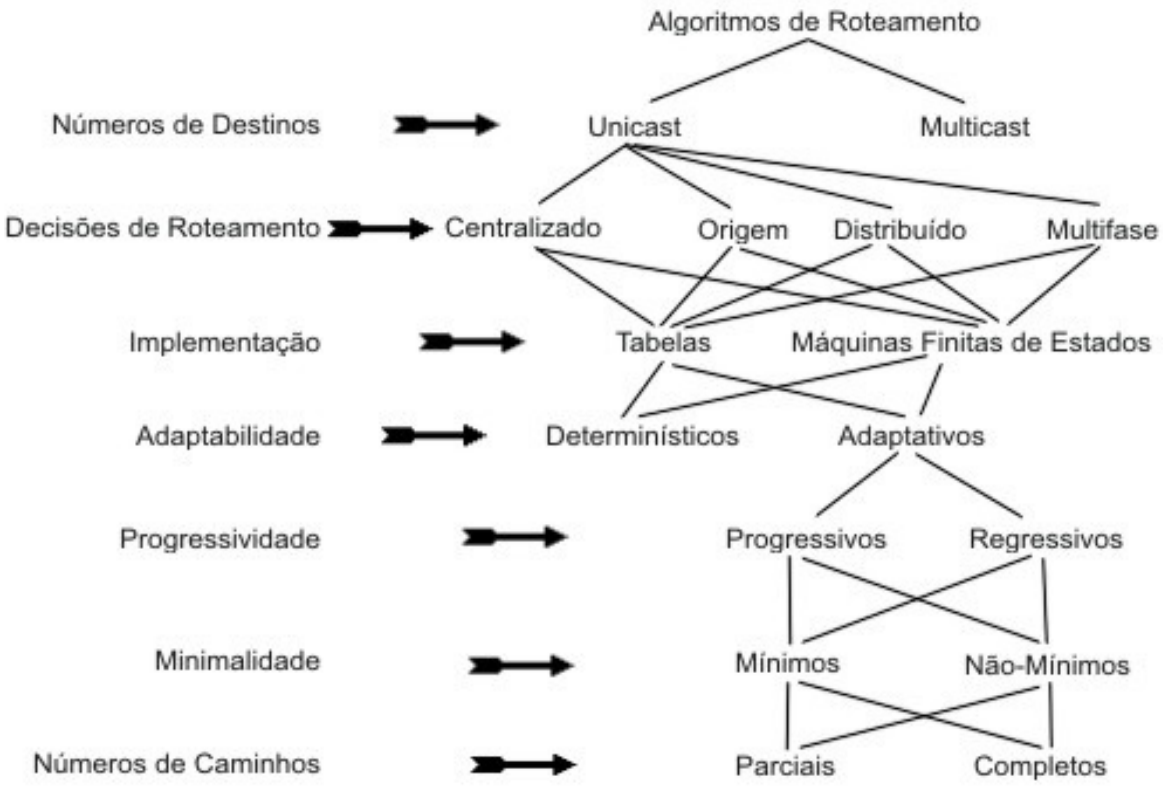


Figura 2.11. Classificações dos algoritmos de roteamento

2.5. Considerações Finais

Neste capítulo foram apresentados os fundamentos de redes de interconexão, introduzindo os principais conceitos do tema e abordando as diversas características desse paradigma de comunicação. A ênfase de redes de interconexão voltadas para SoCs foi dada, apresentando as principais diferenças entre NoCs e redes de interconexão para processadores paralelos.

CAPÍTULO

3

Estado da Arte de Redes em Chip

A necessidade de novas arquiteturas de comunicação intrachip vem abrindo grandes alternativas para a pesquisa de novas formas de comunicação para tais sistemas. A abordagem mais aceita atualmente é a utilização de conceitos de redes de interconexão de multiprocessadores para realizar a comunicação entre núcleos de um chip. Porém, apesar de utilizar boa parte da teoria de interconexão de computadores paralelos, as redes em chip têm características próprias que as distinguem das anteriores.

Entre as diferenças, a principal está no fato de que existe a limitação de área nos chips, exigindo que as chaves roteadoras das NoCs sejam pequenas e rápidas, a ponto de não causar atrasos na comunicação entre os núcleos. Além disso, essas chaves não podem ser complexas, pois normalmente sistemas em chip têm restrições quanto ao consumo de energia. Assim, os algoritmos de roteamento e lógica de controle e arbitragem devem manter um compromisso entre simplicidade e eficiência nessa nova abordagem de comunicação. Redes em chip vêm sendo propostas recentemente e algumas delas são discutidas neste capítulo.

3.1 SPIN

Uma das primeiras propostas de redes de interconexão para sistemas em chip foi a rede SPIN (*Scalable Programmable Interconnection Network*) (Adriahantenaina et al., 2003). É uma

rede indireta que utiliza topologia de Árvore Gorda (*Fat Tree* ou *Folded Butterfly*), para prover roteamento simples e eficaz. Tal topologia é apresentada na Figura 3.1.

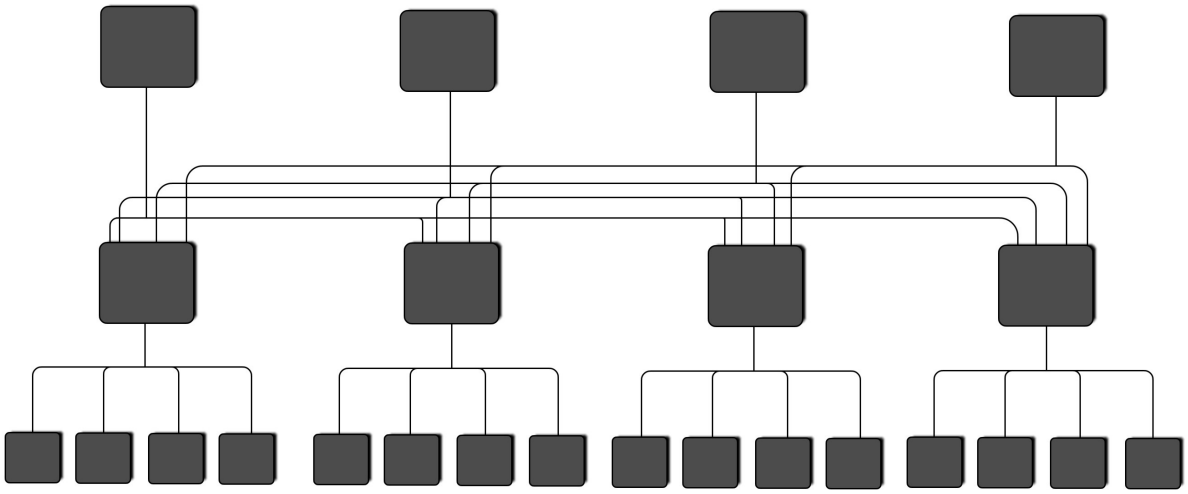


Figura 3.1. Topologia utilizada pela Rede SPIN

A topologia Árvore Gorda utiliza uma estrutura de árvore na qual todos os núcleos se concentram nos nodos folhas. Os outros níveis da árvore são consistituídos por roteadores responsáveis por encaminhar os pacotes para a subárvore onde se encontra o núcleo destino. A árvore possui várias raízes para evitar que o tráfego congestione em um único roteador. Tal topologia produz uma rede não-bloqueante de alto desempenho. Entretanto, para um grande número de núcleos, o *layout* de uma rede com essa topologia é mais complexo em comparação às redes com topologias Toróide ou Grelha.

A rede SPIN contém dois canais unidirecionais por *link*, permitindo que os pacotes sejam enviados nas duas direções. Nas primeiras implementações dessa rede cada canal possuía 36 bits de largura, sendo 32 bits de dados e o restante utilizado para paridade e identificação de erros no tráfego. Na SPIN cada pacote é definido como sequências de 32 bits. No cabeçalho, um campo de oito bits indica o destino do pacote, limitando a comunicação da rede a, no máximo, 256 núcleos.

Essa rede utiliza *Wormhole* como estratégia de *switching* e roteamento adaptativo e distribuído. Quando um pacote chega a um roteador, este escolhe um dos canais disponíveis para enviar o pacote baseado no endereço de destino. O processo continua até que o pacote chegue a um roteador ancestral do núcleo destino. Quando isso ocorre o pacote é encaminhado através de um caminho determinístico, o único entre o roteador em que o pacote se encontra e seu destino.

Cada roteador na SPIN contém oito portas, cada uma com um par de canais, um de entrada e outro de saída. O roteador, denominado RSPIN, possui um *buffer* de quatro posições

em cada canal de entrada e dois *buffers* de 18 posições compartilhado entre todas as portas de saída. Esses *buffers* possuem uma prioridade maior no caso de competição entre eles e um *buffer* de uma porta de entrada, o que permite a redução da latência de entrega dos pacotes. A Figura 3.2 apresenta a estrutura básica do roteador RSPIN.

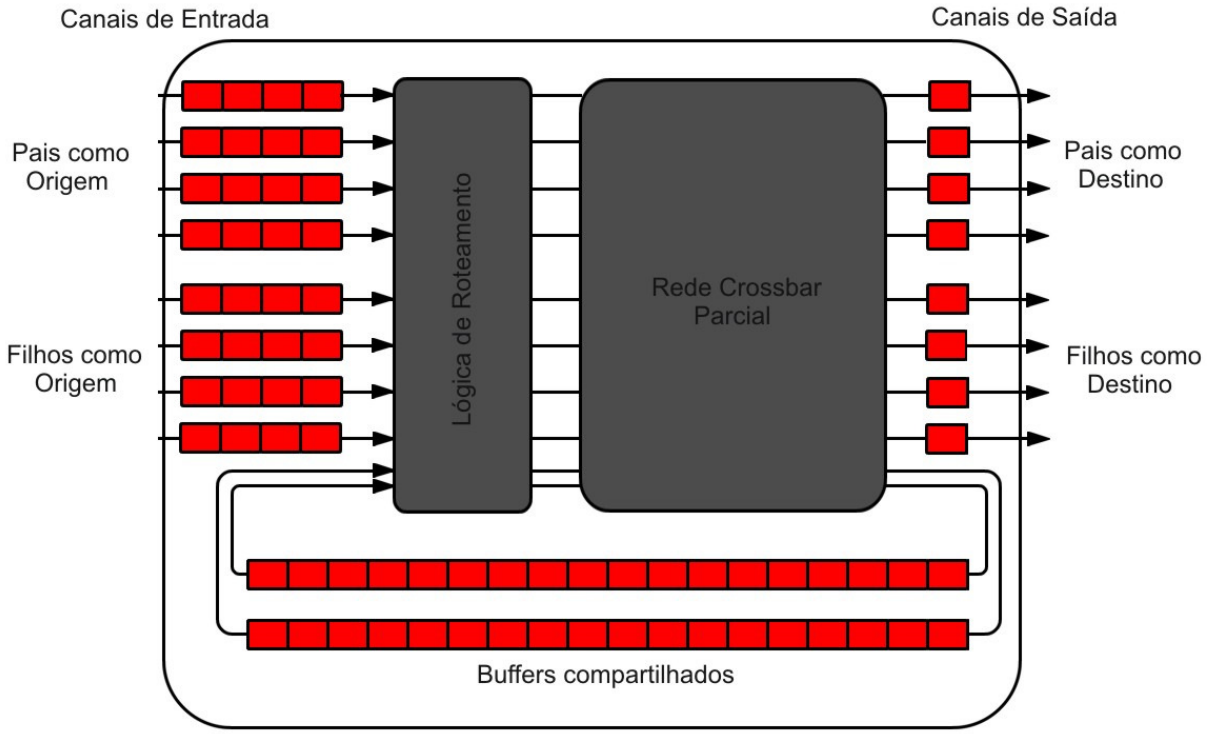


Figura 3.2. Estrutura do roteador RSPIN

Para o roteamento dos pacotes das portas de entrada para as portas de saída, o roteador possui uma rede *crossbar* interna que implementa apenas as conexões permitidas pelo algoritmo de roteamento. Apenas pacotes vindos das folhas podem ser encaminhados para os nodos pais e apenas os pacotes que veem dos pais podem ser direcionados para os filhos. São esses últimos pacotes que podem utilizar os *buffers* de saída, quando o canal de saída requisitado está ocupado.

3.2 Octagon

A rede Octagon (Karim, Nguyen e Dey, 2002) foi desenvolvida pela STMicroeletronics para processadores de rede. Sua topologia é baseada em anel e sua configuração básica consiste de oito nodos e doze *links* bidirecionais, como apresentado na Figura 3.3. Isso permite que a comunicação entre qualquer par de nodos seja realizada com no máximo dois saltos, passando

por um único nodo intermediário. Para isso, um algoritmo que sempre indica o caminho mínimo entre dois nodos é utilizado.

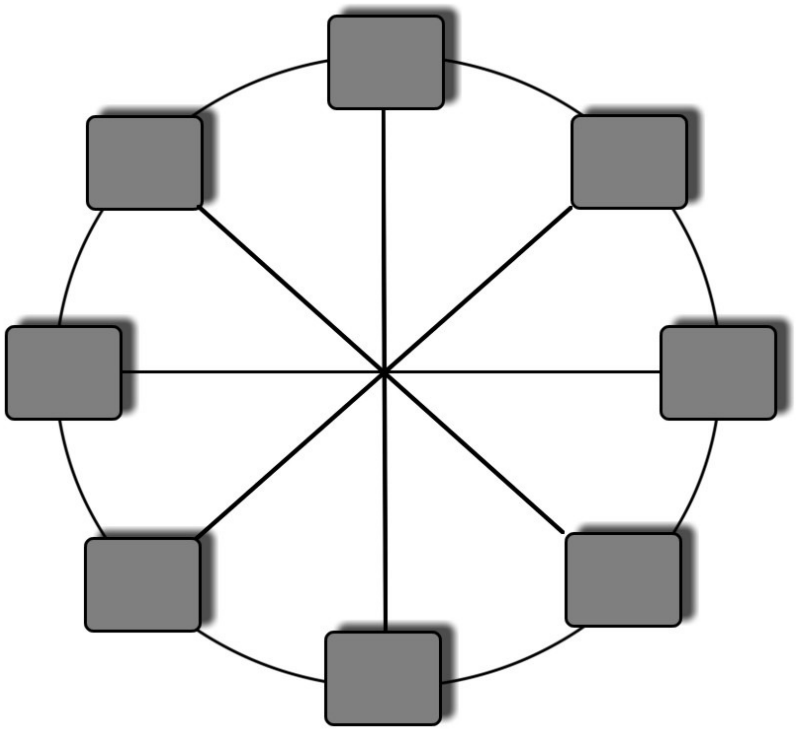


Figura 3.3. Estrutura básica de uma rede Octagon

Os pacotes na Octagon podem ser de tamanhos fixos ou variáveis. Essa rede pode operar tanto em modo *packet* quanto em modo *circuit switching*. Para permitir a operação no modo *circuit switching* um escalonador denominado *best-fit* foi desenvolvido. O escalonador consiste em um protocolo de comunicação orientado a conexão que pode acomodar simultaneamente diversas conexões que não se sobrepõem. Para isso, cada roteador possui três filas de requisições de saída, uma para cada porta do roteador. O escalonador verifica as primeiras posições de todas as filas e estabelece as conexões por ordem de espera. Todas as conexões possíveis de requisições nas primeiras posições das filas são estabelecidas. Quando um pacote termina de ser transportado o escalonador é reativado para verificar se existem novos pedidos de conexão. Essa estratégia permite melhorar o desempenho do sistema e utilização dos nodos em relação a alguns protocolos de comunicação que bloqueiam o nodo requerente até que seu pedido possa ser atendido. Entretanto, cada nodo da Octagon deve possuir uma fila grande o suficiente para evitar perdas de pacotes.

Buscando maior escalabilidade de forma a possibilitar que mais núcleos sejam utilizados em uma rede Octagon, é possível interconectar vários anéis com a utilização de um

nodo ponte, que não é ligado a nenhum núcleo, mas conecta anéis adjacentes, como mostra a Figura 3.4.

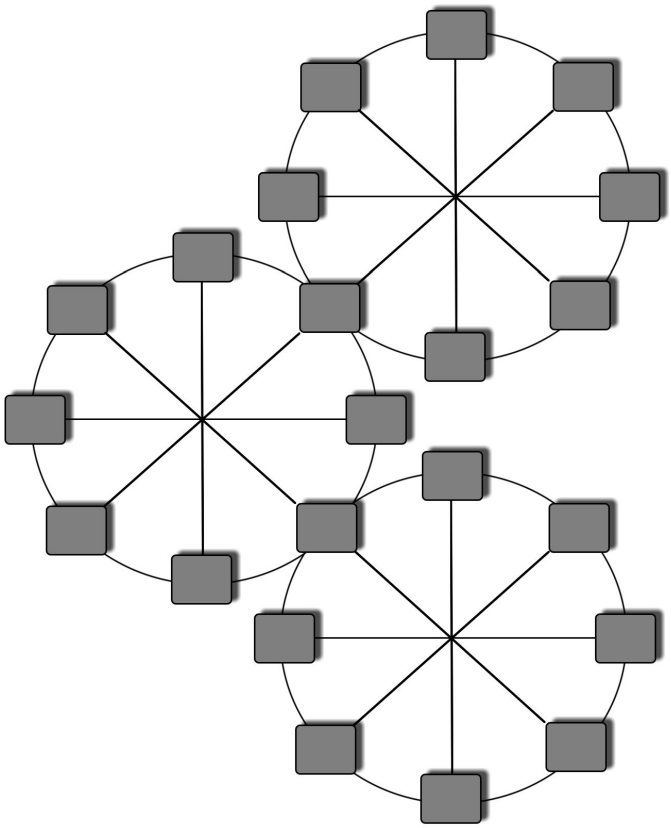


Figura 3.4. Octagon com três níveis de anéis

Com essa configuração são necessários bits adicionais para determinar o endereço de destino do pacote. Parte do endereço deve indicar em qual anel o nodo destino se encontra e outra parte indica qual dos nodos do anel é o destino. Assim, ao receber um pacote cujo destinatário está em outro anel, o roteador receptor deve encaminhá-lo para o roteador ponte correspondente.

Essa adição de mais anéis faz com que o número máximo de saltos entre um par de roteadores aumente na ordem de *2x*, sendo *x* o número de anéis. Assim, com 16 nodos (um nodo ponte e quinze nodos compostos por roteadores e núcleos) o número máximo de saltos é quatro e com 24 nodos (dois nodos pontes e 22 nodos constituídos de roteadores e núcleos) o número máximo de saltos é seis.

3.3 Spidergon

Spidergon é uma rede comercial, baseada na *Octagon,* proposta pela STMicroeletronics (Coppola et al., 2004) com o objetivo de oferecer uma topologia fixa e otimizada para

implementações de baixo custo em SoCs com vários núcleos de processamento. É uma rede regular, escalável e com topologia direta.

A rede Spidergon conecta um número par de nodos com *links* bidirecionais em forma de anel tanto no sentido horário quanto no sentido anti-horário. Alem disso há a adição de canais que cruzam a rede conectando pares de roteadores. A quantidade de *links* da rede é $N = 2*n$, sendo n o número de nodos da rede. Cada nodo i, $0 <= i < N$, é conectado a outros três nodos da seguinte forma: (i) no sentido anti-horário o nodo se conecta ao nodo (i-1 mod N); (ii) em sentido horário ao nodo (i+1 mod N); (iii) cruzando a rede ao nodo (i+n mod N). A Figura 3.5 apresenta a topologia da rede Spidergon com 16 nodos.

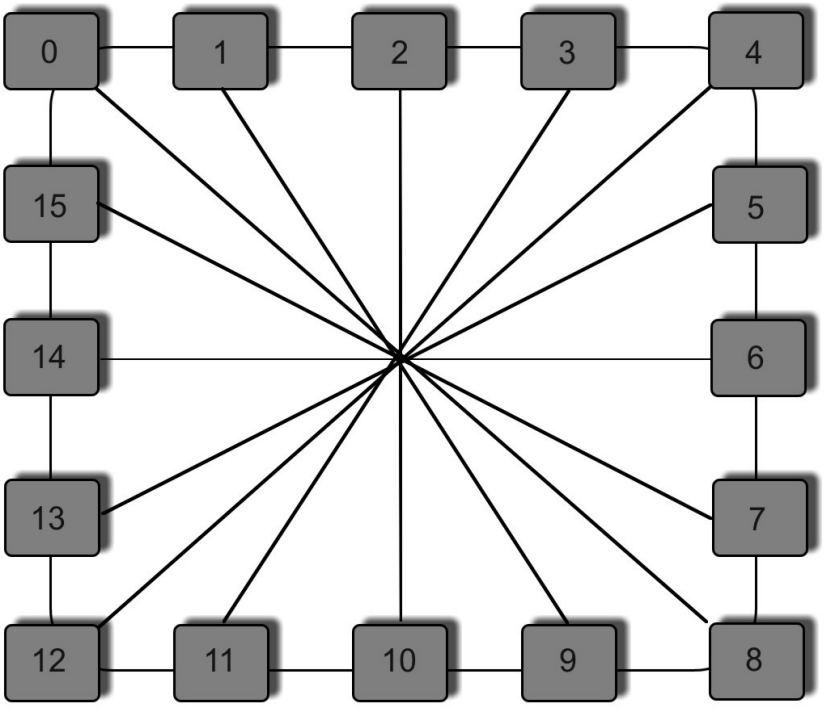


Figura 3.5. Topologia da rede Spidergon

Com tal organização, a rede é capaz de proporcionar bom diâmetro, pequeno número de conexões, sistema de roteamento simples, canais com o mesmo comprimento e a utilização de roteadores homogêneos em toda a rede (Moadeli et al., 2007). A Spidergon utiliza comunicação *packet switched*, dividindo as mensagens em pacotes, e estratégia *Wormhole*. Em relação à Octagon, a principal diferença é que a Spidergon é capaz de realizar qualquer comunicação com *log N* saltos, não apresentando a estrutura baseada em subanéis da Octagon.

O algoritmo de roteamento é determinístico e sempre busca o menor caminho para realizar a comunicação entre dois nodos. A ideia do algoritmo é verificar o endereço de

destino do pacote e encaminhá-lo pelo canal adequado dependendo dessa distância. Se ela for maior que *N*/4, é utilizado o canal que cruza a rede, caso contrário são usados os *links* laterais, com o pacote sendo enviado para a direção adequada. Os canais do anel contêm dois canais virtuais cada, que são utilizados apenas como escape para evitar a ocorrência de *deadlock* na rede.

Um outro modelo de rede, denominado Quark, foi proposto com base na reed Spidergon. A rede Quark (Moadeli, Maji e Vanderbauwhede, 2009) apresenta algumas modificações para garantir melhor desempenho. As alterações são: (i) a adição de um canal paralelo ao canal que cruza a rede, permitindo que sejam realizadas comunicações simultâneas entre um par de roteadores nas duas direções; (ii) todas as portas do roteador são tratadas ao mesmo tempo, assim comunicações paralelas que envolvam um mesmo roteador, mas portas diferentes, podem ser estabelecidas ao mesmo tempo; e (iii) os roteadores podem receber e enviar *flits* ao mesmo tempo. A topologia resultante para a rede Quark com oito nodos é apresentada na Figura 3.6.

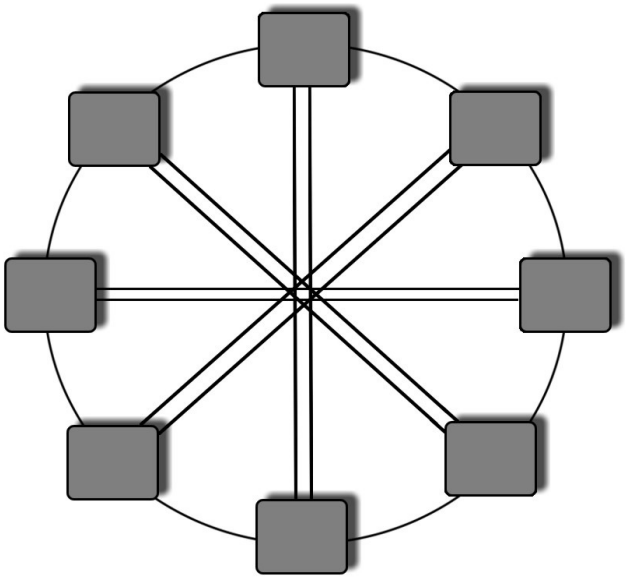


Figura 3.6. Topologia da rede Quark

3.4 SoCIN

A rede SoCIN (*SoC Interconnection Network*) (Zeferino e Susin, 2003) é uma rede de interconexão com chaveamento por pacote do tipo *Wormhole* e topologia Grelha. Ela utiliza o algoritmo de roteamento XY, que garante que não haverá *deadlock* na rede e é simples de implementar.

Os *links* da SoCIN possuem dois canais unidirecionais com direções opostas. Cada canal contém *n* bits para a transmissão de dados, além de 2 bits adicionais que indicam se o *flit* é o cabeçalho ou o final do pacote. Além disso, são utilizados bits de controle de fluxo, que indicam a validade (*val*) e a confirmação do recebimento de um pacote (*ack*). A Figura 3.7 apresenta a organização dos *links* da SoCIN.

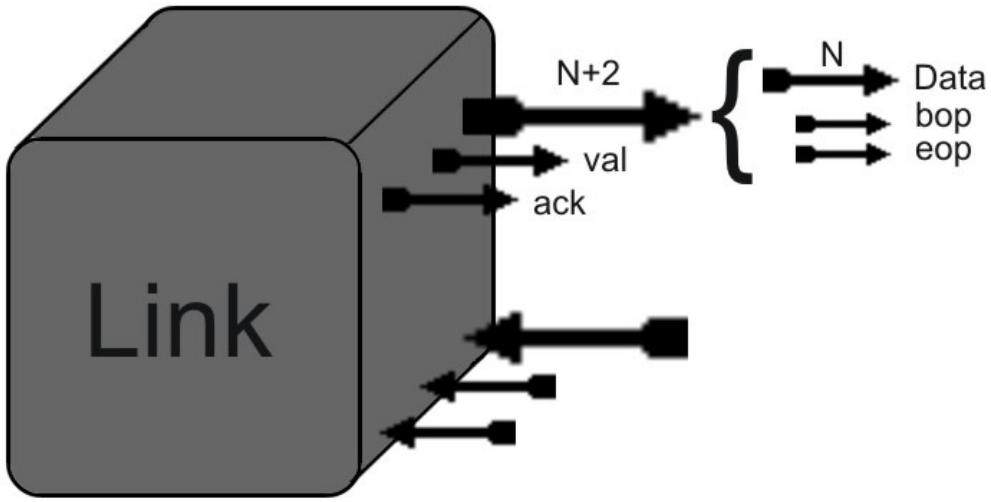


Figura 3.7. Organização de um link na rede SoCIN

O algoritmo de roteamento utilizado pela SoCIN é o XY. Nele, um pacote deve primeiro ser encaminhado através do eixo X até encontrar a coluna correspondente ao seu destino, para então ser encaminhado pelo eixo Y até encontrar o nodo destinatário. Uma vez que o pacote passa a utilizar a direção Y ele não pode mais retornar para a direção X. Isso garante que não exista *deadlock* na rede. A Figura 3.8 apresenta o funcionamento do algoritmo, mostrando dois caminhos percorridos por diferentes pacotes.

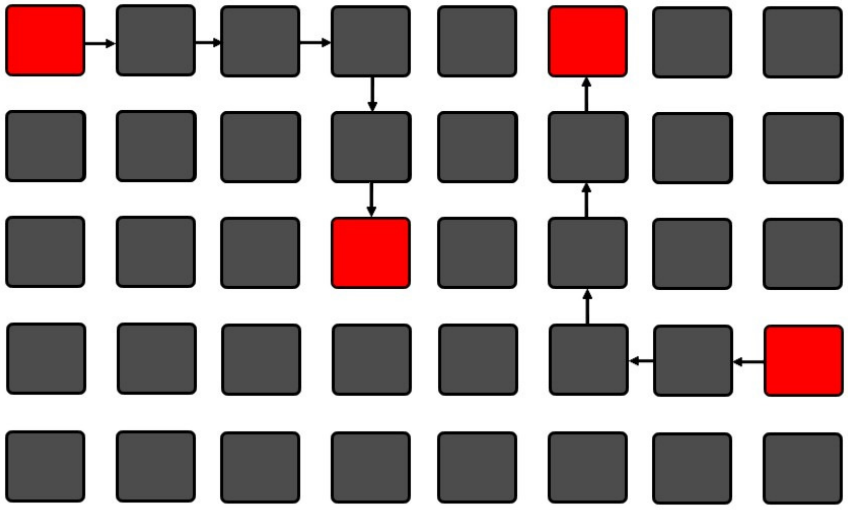


Figura 3.8. Algoritmo XY

O pacote da SoCIN é formado por um *flit* cabeçalho, um ilimitado número de *flits* de dados e um *flit* que indica o término do pacote. Cada *flit* contém n+2 bits, sendo n a quantidade de bits de dados. Os dois bits adicionais são denominados *bop* (*begin of packet*) e *eop* (*end of packet*). No cabeçalho, o bit *bop* é ativado para 1, indicando o início de um pacote; e no terminador do pacote o bit *eof* é ativado para 1, indicando que o pacote terminou de ser transmitido. Em todos os outros casos esses bits são marcados com o valor '0'. O cabeçalho carrega as informações de roteamento, contendo a quantidade de nodos que devem ser percorridos *(Xmod* e *Ymod)* e a direção a ser tomada (*Xdir* e *Ydir*) em cada eixo. A Figura 3.9 exemplifica a implementação do roteamento na rede SoCIN.

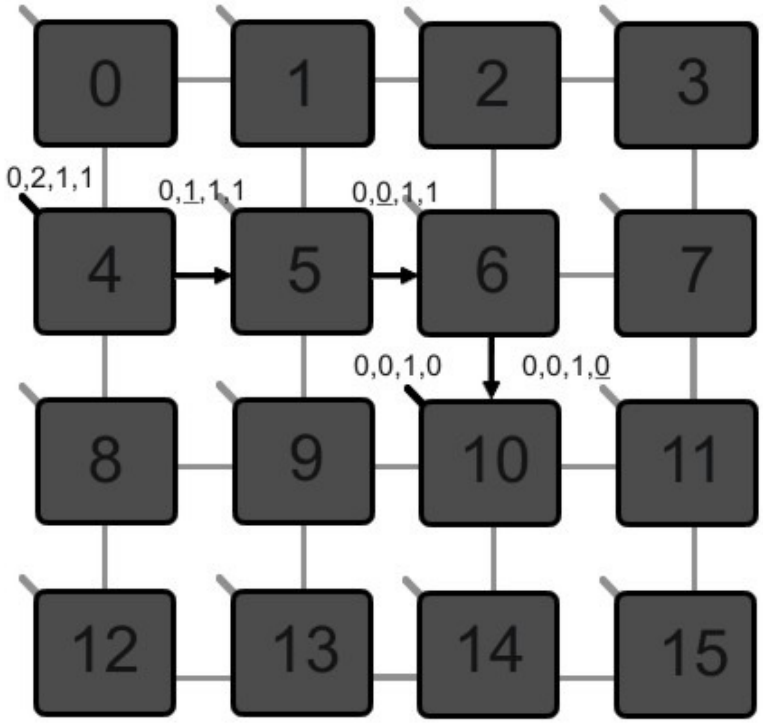


Figura 3.9. Exemplo de roteamento na rede SoCIN

Na Figura 3.9 cada rótulo associado a um nodo contém os valores *Xdir*, *Xmod*, *Ydir* e *Ymod*, nessa sequência. No eixo X, 0 indica a direção leste e 1 a direção oeste, enquanto no eixo Y, 0 indica a direção norte e 1 a direção sul. O pacote é inserido pelo roteador 4 e tem como destino o roteador 10. Ao ser inserido na rede, o cabeçalho do pacote contém *Xmod* = 2, indicando que mais dois nodos devem ser percorridos pelo eixo X. Então, o roteador envia o pacote pela direção leste, uma vez que *Xdir* = 0. O roteador 5 recebe o pacote, decrementa o valor de *Xmod* e o compara. Como o valor é maior que 0, no caso o valor é 1, o pacote deve ser encaminhado através do eixo X para a direção indicada por *Xdir*. Assim, o pacote é encaminhado ao roteador 6, que também decrementa *Xmod*. O valor de *Xmod* igual a 0 indica

que ele está alinhando ao roteador destino no eixo X e que então o pacote deve ser encaminhado através do eixo Y. Os mesmos procedimentos ocorrem no eixo Y até que *Ymod* chegue ao valor 0.

Para possibilitar o roteamento é necessário que o roteador que insere um pacote na rede tenha conhecimento da quantidade de nodos entre ele e o roteador destino, para que esses valores possam ser inseridos no cabeçalho. Para isso, uma tabela de roteamento, contendo todos os roteadores da rede, é consultada. Assim, é possível determinar os valores de *Xmod*, *Ymod*, *Xdir* e *Ydir* no momento de construir o cabeçalho.

A última versão da SoCIN conta com o roteador ParIS (*Parametrizable Interconnection Switch*) (Zeferino, Santo e Susin, 2004), que pode ser parametrizado de forma a atender diferentes expectativas de uma rede a ser implementada, seja otimizando custo ou desempenho. O roteador contém cinco portas, quatro para a comunicação com outros roteadores e uma porta para comunicação interna, conectando o roteador ao núcleo de processamento ao qual ele está associado. Cada porta de comunicação possui dois módulos, um com um canal de entrada e outro com um canal de saída. Dependendo da posição do roteador na rede nem todas as portas são necessárias. Assim, essas portas não precisam ser sintetizadas, reduzindo o custo no momento da implementação. O que indica se uma porta deve ou não ser parametrizada são parâmetros definidos no roteador.

3.5 Nostrum

A Nostrum (Millberg et al., 2004) é uma NoC com topologia Grelha e chaveamento *packet switching* do tipo *Wormhole*. Assim como na SoCIN, o roteamento é realizado antes de inserir um pacote na rede, com a direção e quantidade de saltos inseridos no cabeçalho do pacote. Entretanto, a rede Nostrum não segue o algoritmo XY, uma vez que o pacote não necessita ser encaminhado primeiramente pelo eixo X. O pacote pode seguir por qualquer um dos eixos, sempre seguindo a direção indicada no cabeçalho e desde que não atinja a mesma linha ou coluna do nodo destino. Se isso ocorrer o pacote só poderá percorrer um dos eixos, até chegar ao nodo destino.

A Nostrum fornece ainda regras de roteamento para tentar melhorar o desempenho da rede e evitar áreas congestionadas, que normalmente ocorrem na parte central da rede (Nilson, 2002). Assim, informações da carga de cada nodo são compartilhadas entre os roteadores vizinhos. Essas informações auxiliam no momento em que um roteador toma a decisão da direção para a qual será enviado o pacote. Tomando como exemplo um pacote que pode ir

pela direção norte no eixo Y ou pela direção oeste no eixo X, o roteador pode, a partir das informações de tráfego recebidas, enviar o pacote para a direção com menor tráfego de pacotes. As informações de tráfego podem ser fornecidas por meio da quantidade de pacotes roteados em determinado período de tempo.

Outra técnica implementada na Nostrum que visa a melhora de desempenho são os circuitos virtuais. Se um roteador deseja enviar uma grande quantidade de pacotes para outro, um caminho entre os dois roteadores pode ser reservado com largura de banda fixa. Dessa forma, todos os pacotes que chegarem a um roteador tendo como origem o roteador que fez o pedido do circuito virtual, serão imediatamente encaminhados para a porta de saída indicada pelo caminho, que sempre estará livre. Essa conexão pode oferecer uma latência garantida, uma vez que todos os pacotes chegarão no mesmo intervalo de tempo. O caminho reservado pode ser o caminho mais curto ou o menos congestionado.

Quando requisitando um circuito virtual, o roteador de origem envia pacotes de controle aos roteadores que fazem parte do caminho, começando pelo roteador mais próximo. Ao receber um desses pacotes, o roteador responde ao roteador de origem, seja para aceitar ou negar o circuito virtual. A Figura 3.10 apresenta a alocação de um circuito virtual entre dois nodos.

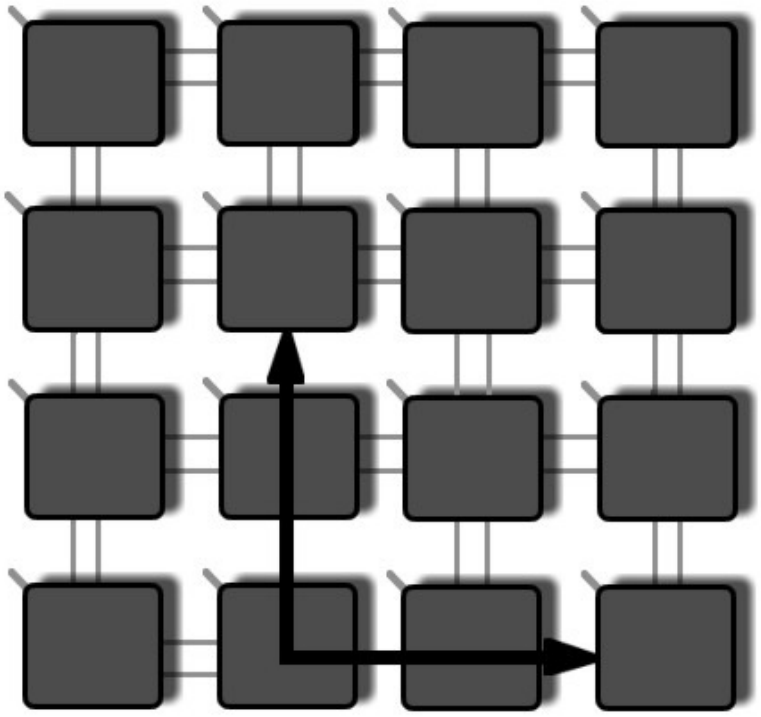


Figura 3.10. Circuito virtual na rede Nostrum

O circuito virtual é terminado assim que determinada quantidade de pacotes seja transmitida. Essa quantidade é determinada nos pacotes de controle, enviados para estabelecer o circuito. Para evitar que o estabelecimento de circuitos virtuais cause muito atraso na entrega de pacotes que estão sendo roteados normalmente, o número de circuitos virtuais que podem ser utilizados simultaneamente é limitado. Essa limitação depende do tamanho da rede. Além disso, os circuitos não podem ser sobrepostos em um mesmo eixo para evitar conflitos.

3.6 Xpipes

Xpipes (Bertozzi e Benini, 2004) é uma rede de interconexão voltada para SoCs que operam na escala dos gigabytes. A rede constitui uma biblioteca com componentes altamente parametrizavéis (interface de rede, roteador) que são utilizados na fase de projeto para compor uma NoC. Isso significa que diferentes topologias, regulares ou não, podem ser especificadas usando as estruturas da Xpipes. O objetivo da rede é permitir a criação de arquiteturas de comunicação para sistemas específicos, inclusive sistemas heterogêneos, nos quais nem sempre nodos vizinhos precisam se comunicar.

O grau de parametrização da rede permite especificar valores tanto para parâmetros globais da rede quanto para parâmetros específicos para os roteadores (Jalabert, Murali, Benini e De Micheli, 2004). Entre os parâmetros globais estão o tamanho do *flit*, o espaço de endereçamento dos nodos, o número máximo de saltos entre dois nodos específicos, o número de bits para controle de fluxo, entre outros. Já como parâmetros específicos estão o número de portas de cada roteador, o número de canais virtuais e o comprimento de cada *link*, a largura dos *buffers* nas portas de saída e o conteúdo das tabelas de roteamento para determinar o caminho dos pacotes.

Na Xpipes os pacotes são divididos em *flits*, que contêm um campo que permite distinguir se o *flit* é um cabeçalho ou um dado. A rede utiliza a técnica *Wormhole* como estratégia de chaveamento e algoritmo de roteamento estático. O roteamento é determinado por meio de uma tabela de roteamento, baseada no endereço de destino, existente em cada roteador. Cada rota é representada por um conjunto de bits de direcionamento. Dessa forma, cada roteador encaminha os *flits* pertencentes a determinado pacote à porta de saída correspondente baseado no destino do pacote. Isso permite simplificar a implementação do roteador, já que não são necessárias tomadas de decisões dinamicas. Em contrapartida, o custo da tabela de roteamento pode ser grande caso existam muitos nodos na rede.

Os *links* de NoCs irregulares podem conter diferentes comprimentos, podendo variar o número de ciclos de *clock* para percorrê-los. Para otimizar a vazão dos dados, a Xpipes implementa uma espécie de *pipeline* nos *links*, subdividindo-os em segmentos básicos que levam um ciclo de *clock* para serem percorridos. Isso permite melhor identificar a latência da rede, que pode ser analisada considerando cada segmento de um *link* e essa análise pode ser independente dos diferentes tamanhos de *links*.

A arquitetura suporta ainda os controles de fluxo *Ack/Nack* e *Stop*/*Go* e utiliza *buffers* nas portas de saída dos roteadores. Os roteadores também são divididos em *pipelines* como mostra a Figura 3.11, que apresenta o módulo de saída de uma das portas de um roteador com quatro portas de entrada.

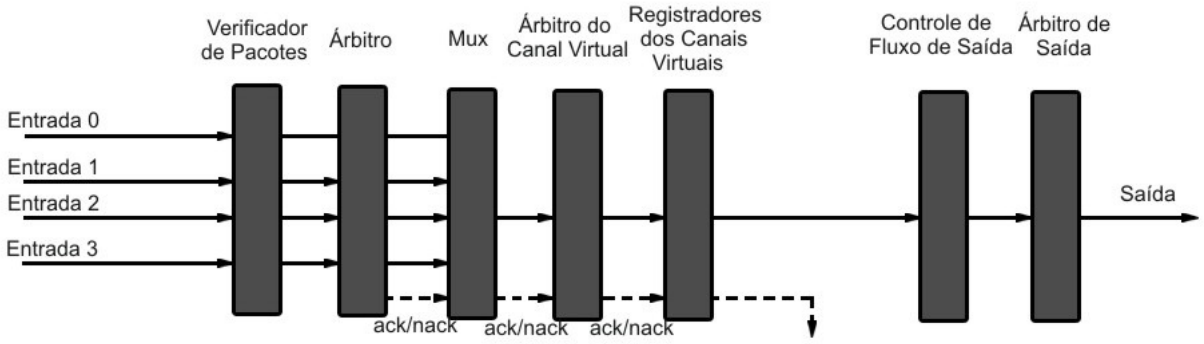


Figura 3.11. Roteador da rede Xpipes

O primeiro estágio do *pipeline* verifica o cabeçalho dos pacotes que chegam às diferentes portas de entrada para determinar quais pacotes serão roteados pela porta analisada. Apenas pacotes que correspondam à porta em questão passam para o segundo estágio, que é responsável por resolver conflitos baseado em uma política de escalonamento *round-robin*. Nesse estágio, um sinal *nack* é gerado e enviado através das portas caso existam pacotes que não foram aceitos no momento. O terceiro estágio é apenas um multiplexador, que seleciona a porta de entrada priorizada no estágio anterior. O quarto estágio mantém o estado dos registradores dos canais virtuais para determinar se os *flits* podem ou não serem armazenados nos registradores. Um *flit* cabeçalho é enviado para o registrador com maior espaço disponível, seguido pelo restante dos *flits* que compõem o pacote. É no próximo estágio que ocorre o armazenamento dos *flits* nos *buffers*. Um sinal *ack/nack* indica se o *flit* foi armazenado ou não com sucesso. O sexto estágio é responsável por tratar da saída da porta. Nele, um *flit* é transmitido ao próximo roteador para verificar a disponibilidade da chave para que a transmissão continue. O sétimo e último estágio corresponde a multiplexadores que selecionam por qual canal virtual será estabelecida a comunicação com o próximo roteador.

3.7 MANGO

A rede MANGO (*Message-passing Asynchronous Network-on-chip providing Guaranteed services over OCP interfaces*) (Bjerregaard e Sparso, 2005) é uma NoC assíncrona que permite a implementação de sistemas GALS (*Globally Asynchronous Locally Synchronous*) em SoCs. Seus roteadores e *links* não utilizam sinais de *clock* para sincronizar as operações. A rede pode operar em modo "serviço garantido" (*guaranteed service*) (Bjerregaard e Sparso, 2006) com latência de entrega de um pacote conhecida e modo "melhor tentativa" (*best effort*), no qual os pacotes competem por *links* para serem enviados através da rede.

Para prover serviço garantido, a rede implementa canais virtuais reservados exclusivamente para o estabelecimento de circuitos virtuais. Pacotes em modo serviço garantido e melhor tentativa compartilham o mesmo *link* físico, mas a banda do *link* é dividida entre eles, separando o tráfego da rede.

Os circuitos virtuais são estabelecidos com a reserva de uma sequência de canais virtuais ao longo de um caminho. Dados que passam por esse caminho terão latência garantida, já que os canais virtuais são utilizados apenas para a comunicação do circuito virtual em questão. A Figura 3.12 apresenta a estrutura do roteador MANGO.

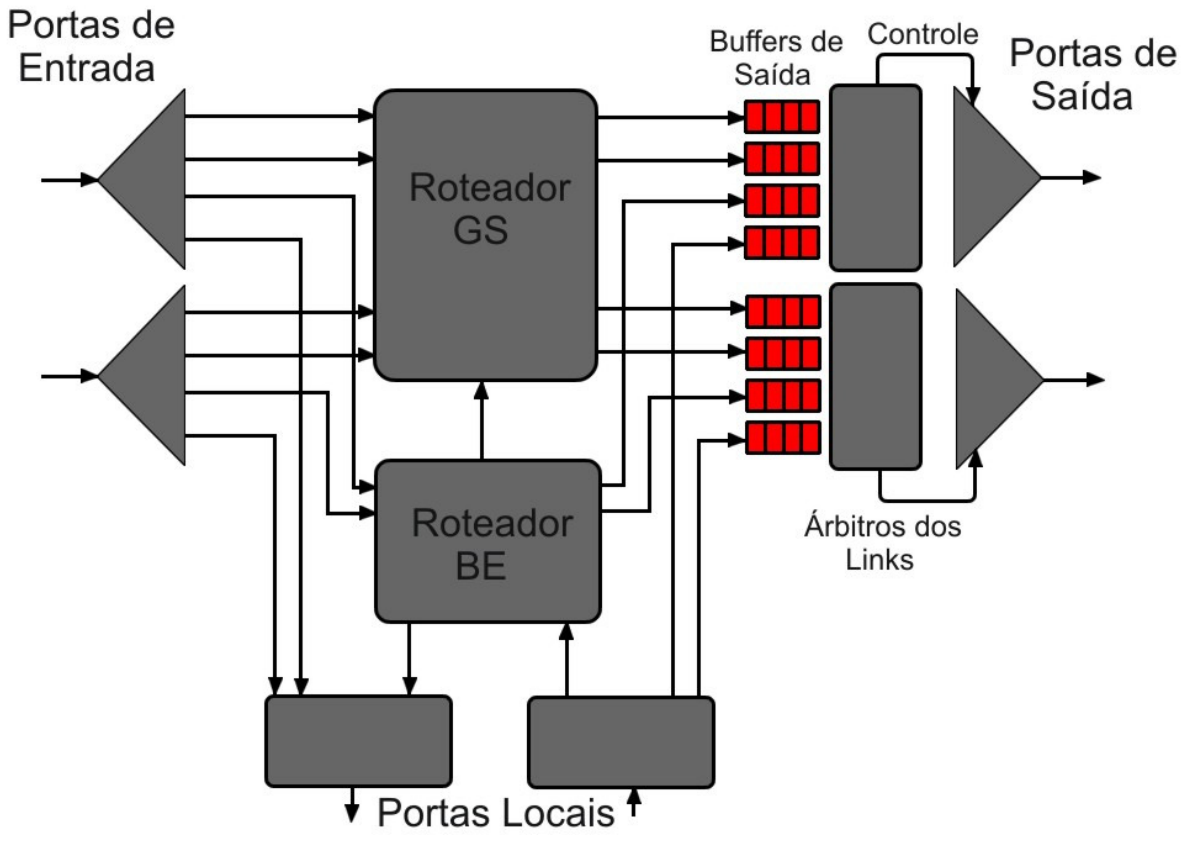


Figura 3.12. Roteador MANGO

Como pode ser observado na Figura 3.12, o roteador apresenta um conjunto de portas unidirecionais. Duas dessas portas são responsáveis pela comunicação local por meio de um adaptador de rede. As portas restantes são utilizadas para conectar roteadores vizinhos, sendo algumas portas de entrada e outras de saída. Cada uma dessas portas implementa um determinado número de canais virtuais independentes, utilizados para separar os dois modos de roteamento da MANGO.

Internamente o roteador é dividido em dois módulos separados que implementam cada um dos modos de roteamento. O roteador GS é responsável pelo modo de serviço garantido e o roteador BE trata do modo de melhor tentativa. O roteador BE encaminha os pacotes dinamicamente de acordo com um caminho de roteamento definido no cabeçalho do pacote. O controle de fluxo utilizado por esse roteador é baseado em créditos e o roteador BE é capaz de aceitar pacotes de variados tamanhos. Um *flit* cabeçalho contém os dados do roteamento. Após estabelecer a comunicação com determinada porta, a mesma decisão de roteamento será aplicada para todos os pacotes que passarem pelo roteador até que este detecte um *flit* terminador, indicado por um bit de controle.

Já o roteador GS envia sequências de dados que não possuem cabeçalhos por meio de caminhos definidos estaticamente. Para evitar que *flits* que chegam de diferentes canais virtuais bloqueiem uns aos outros é necessário realizar um controle dos canais virtuais. Assim, um *flit* só é transmitido se houver espaço disponível no *buffer* do canal virtual alvo. Para estabelecer o caminho virtual pacotes BE são enviados para os roteadores que farão parte do circuito e a interface de programação de cada roteador é utilizada para reservar o caminho, armazenando informações sobre o canal virtual para o qual os *flits* serão encaminhados e de qual canal virtual os *flits* chegam. Assim, ao detectar um *flit* vindo de determinado canal virtual ele é imediatamente enviado ao canal virtual de saída que faz parte do circuito. O roteador MANGO implementa os *buffers* de armazenamento nas portas de saída e assim como a rede Xpipes não possui topologia fixa.

3.8 Hermes

Hermes (Moraes et al., 2004) é uma infraestrutura para a implementação de NoCs com topologia Grelha. Consiste de *links* e roteadores com cinco portas bidirecionais cada, sendo quatro para a comunicação externa com outros roteadores e uma porta para a comunicação local com o núcleo ao qual o roteador está associado.

Os canais correspondem às direções Norte, Sul, Leste e Oeste, para as quais os pacotes podem ser enviados, como mostra a Figura 3.13. Os canais são divididos em módulos de entrada e saída, permitindo o recebimento e envio de dados por um mesmo canal. Internamente, as chaves são estruturadas com *buffers* FIFO nas portas de entrada e com um módulo de controle que determina a direção que um pacote irá tomar e implementa a política de escalonamento.

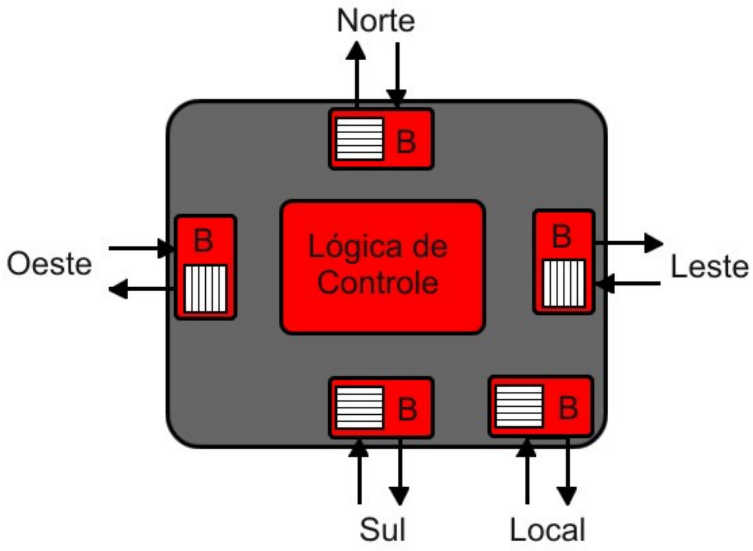


Figura 3.13. Roteador Hermes

Como estratégia de s*witching*, a Hermes utiliza *packet switching* com o método *Wormhole*, não sendo necessário armazenar pacotes inteiros em um roteador caso eles não possam ser imediatamente roteados e diminuindo o tamanho dos *buffers* a serem implementados em hardware.

Quando um roteador recebe um *flit* cabeçalho, o árbitro é executado para garantir prioridade a uma das cinco portas de entradas. As cinco portas podem solicitar roteamento simultaneamente, entretanto apenas um roteamento por vez pode ser tratado. Uma abordagem dinâmica é utilizada na arbitragem. A prioridade de uma porta depende de um índice atribuído a cada uma das portas. A porta leste possui índice 0, a porta Oeste índice 1, a porta Norte índice 2, a porta Sul índice 3 e porta Local índice 4. A prioridade é garantida à porta com o próximo índice da porta que acabou de ter seu roteamento realizado. Caso a porta Local (índice 4) tenha acabado de ter seu roteamento concedido, a prioridade no próximo roteamento será das portas Leste, Oeste, Norte, Sul e Local, nessa ordem. De forma análoga, caso tenha sido concedido o roteamento à porta Norte, a lista de prioridades conterá as portas Sul, Local, Leste, Oeste e, por fim, a porta Norte. Isso previne o *starvation*, uma vez que

eventualmente todas as portas terão prioridade de roteamento, mas não faz qualquer distinção de prioridade entre as portas.

O algoritmo de roteamento utilizado é o XY. O algoritmo é executado assim que é concedida a permissão de conexão a uma das portas de entrada. Cada um dos nodos é associado a um endereço *xLyL*, que indica a posição do nodos nos eixos X e Y. Ao inserir um pacote na rede para ser transmitido, o nodo origem insere no cabeçalho do pacote o endereço *xTyT*, que indica o endereço do nodo destino na rede. A Figura 3.14 apresenta a disposição dos endereços em uma rede Hermes 4x4.

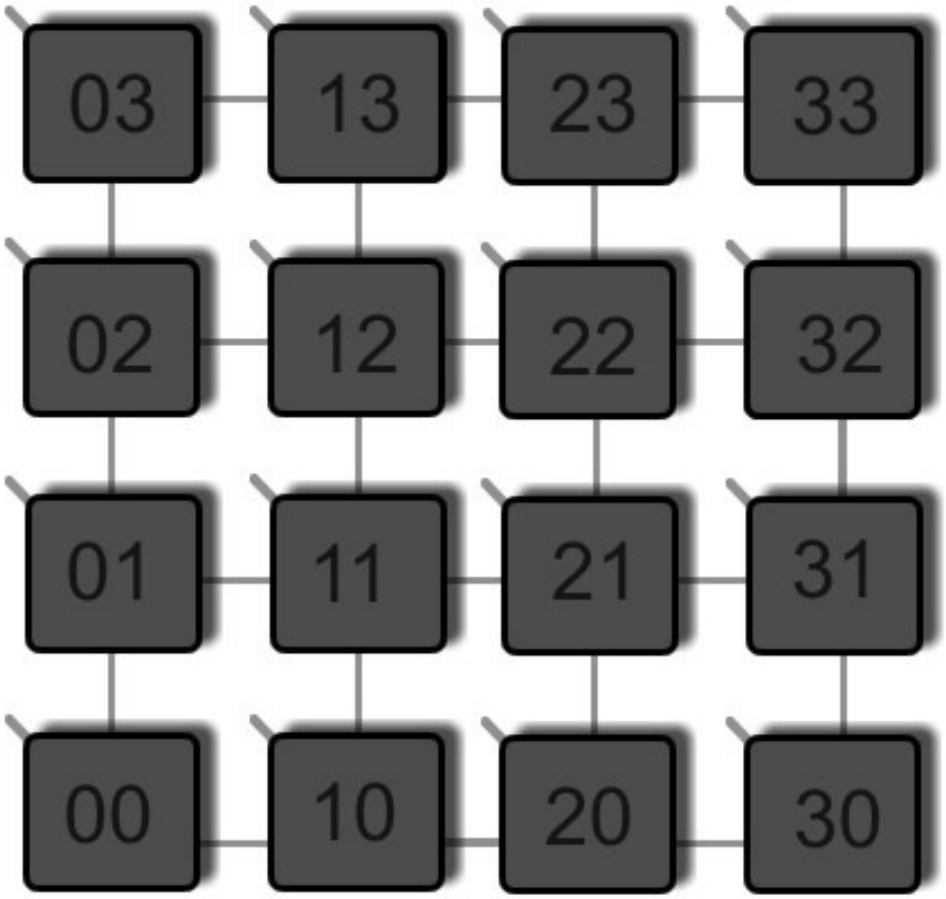
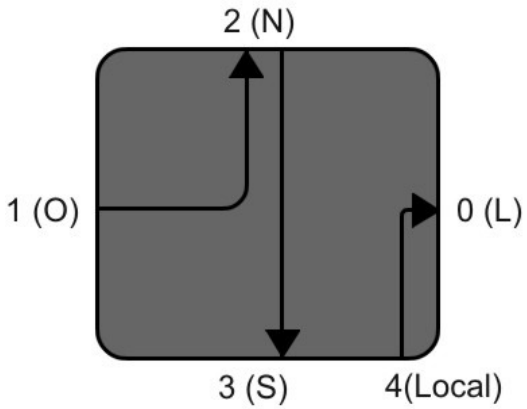


Figura 3.14. Endereçamento dos nodos em uma rede Hermes 4x4

Ao receber um *flit* cabeçalho, o algoritmo de roteamento compara o endereço *xTyT* no cabeçalho com o endereço do próprio nodo. Caso *xLyL* seja igual a *xTyT* significa que o destino do pacote é o nodo atual e o pacote é encaminhado à porta local. Caso contrário é analisado o endereço no eixo X, comparando o valor de *xT* com o valor de *xL*. Essa comparação determina se o pacote será encaminhado para a porta Leste, caso $xL < xT$, para a porta Oeste, caso $xL > xT$, ou se o pacote já está alinhado no eixo X. Se essa última condição for verdadeira, deve ser analisado o endereço em relação ao eixo Y, comparando os valores de

yL com *yT*. Essa comparação determina se o pacote será roteado para a porta Sul, caso *yL* > *yT* ou para a porta Norte, caso *yL* < *yT*. Caso a porta escolhida esteja ocupada, o *flit* cabeçalho, bem como os *flits* subsequentes serão bloqueados e armazenados em *buffers* para posterior envio.

Para o estabelecimento das conexões, uma tabela de roteamento é armazenada com três vetores que indicam conexões estabelecidas e portas livres para conexão. O vetor *in* indica a conexão de uma porta de entrada a uma porta de saída. O vetor *out* mostra a conexão de uma porta de saída a uma porta de entrada. O vetor *free* indica se uma porta está ocupada ou não. A Figura 3.15 exemplifica o uso desses vetores.



	0(L)	1(O)	2(N)	3(S)	4 (Local)
Free	0	1	0	0	1
In	-	2	3	-	0
Out	4	-	1	2	-

(b)

Figura 3.15. Exemplo de configuração de tabela de roteamento na rede Hermes

O roteador da Figura 3.15(a) apresenta três conexões simultâneas. Tomando como exemplo a conexão entre a porta Norte (índice 2) e a porta Sul (índice 3), essa conexão é indicada na tabela de roteamento, Figura 3.15(b), com os valores 3 na coluna da porta Norte no vetor *in* e com o valor 2 na coluna da porta Sul no vetor *out*. Apesar de conter informações redundantes, isso garante melhor desempenho do algoritmo de roteamento. O vetor *free* contém valores 0 nas portas ocupadas no momento.

O encerramento de uma conexão é realizado por meio de cinco contadores, um para cada porta de entrada, existentes em cada roteador. O contador é inicializado quando o segundo *flit* de um pacote chega à porta de entrada. Esse *flit* indica a quantidade de *flits* que compõem o pacote. Assim, sempre que um *flit* é recebido pelo roteador o contador é decrementado. Quando esse valor chega a zero a conexão pode ser encerrada com a tabela de roteamento sendo atualizada.

A comunicação entre um par de roteadores é realizada por dois *links* unidirecionais que conectam ambos os roteadores nos dois sentidos. A Figura 3.16 apresenta a interface de

comunicação de dois roteadores. A porta de entrada é composta pelos sinais *rx*, *data_in* e *ack_rx* enquanto a porta de saída é composta pelos sinais *tx*, *data_out* e *ack_tx*.

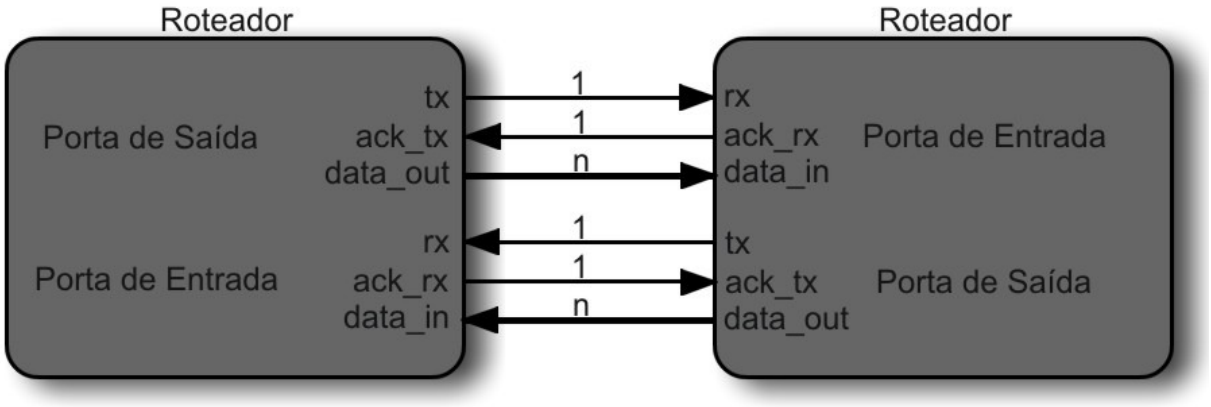


Figura 3.16. Interface de comunicação do roteador Hermes

Após ser concedida a prioridade de comunicação a uma porta de saída de um roteador origem, esta ativa o sinal *tx* da porta escolhida indicando que possui dados a serem enviados por essa porta. O roteador destino recebe a solicitação de comunicação por meio do sinal *rx* e, após verificada a disponibilidade de espaço em seu *buffer*, determina que a comunicação pode ser instituída. Isso é indicado por meio do sinal *ack_rx*. Ao receber essa confirmação no sinal *ack_tx* o roteador origem inicia a transmissão enviando os dados através de *data_out* e retira o *flit* de seu *buffer*, podendo tratar o próximo *flit* da fila.

A rede Hermes é complemente parametrizável sendo possível determinar suas dimensões, tamanho do *flit* e profundidade dos *buffers* durante a geração da rede. A criação é realizada com a ferramenta MAIA (Ost et al., 2005), um *framework* capaz de gerar a rede de acordo com os parâmetros fornecidos. A rede pode ainda ser gerada com a topologia Toróide, todavia um algoritmo livre de *deadlock* ainda não é apresentado. A ferramenta gera os arquivos VHDL em nível de transferência de registradores (RTL), tratando das portas e *buffers* de cada roteador de acordo com sua posição na rede. Justamente por oferecer essa facilidade de criação da rede e por ser bem documentada, a rede Hermes foi escolhida para a realização deste trabalho.

3.9 Considerações Finais

O presente capítulo apresentou a descrição de algumas redes de interconexão em chip propostas recentemente. O objetivo do capítulo foi discutir redes bem estabelecidas na literatura apresentando os principais conceitos empregados no desenvolvimento de NoCs. Isso

não implica que as técnicas de implementação de redes em chip estejam restritas àquelas apresentadas neste capítulo. Outras redes podem ser facilmente encontradas com características distintas. No entanto, as características gerais de NoCs foram aqui apresentadas e exemplificadas.

CAPÍTULO

4

Implementação

Apesar de serem baseadas em redes de interconexão para multiprocessores, que possuem protocolos de comunicação bem definidos e arquiteturas amplamente analisadas, as redes em chip trazem conceitos relativamente novos, uma vez que a implantação dessas redes em um único chip necessita de requisitos diferentes e, consequentemente, possuem implicações distintas no momento de sua implementação. As pesquisas nessa área estão no começo (Concer, 2009), logo análises mais profundas das estruturas propostas até então são necessárias para melhor explorar essa nova abordagem de comunicação em sistemas em chip.

Nesse sentido, o presente trabalho foca a análise de diferentes topologias para NoCs comparando algumas das principais topologias propostas na literatura em relação ao desempenho e custo sob condições similares de simulação e implementação. O atual capítulo aborda as questões de implementação de tais redes, discutindo os principais pontos de sua implantação. Um algoritmo semidinâmico live de *deadlock* para redes com topologia Toróide também é apresentado.

4.1 Ambiente de Implementação

Tendo como objetivo avaliar as diferentes topologias sob condições igualitárias, um mesmo cenário de implementação foi utilizado para todas elas. Assim, os mesmos protocolos de comunicação e arbitragem foram utilizados em todas as implementações.

A rede inicial foi gerada com a infraestrutura Hermes, que utiliza topologia Grelha. Essa rede também é utilizada como parâmetro de comparação com as outras redes desenvolvidas, o que permite uma melhor análise de melhorias ou pioras que as alterações na rede original trouxeram. A rede Grelha gerada foi uma rede 4x4, com quatro roteadores em cada eixo X e quatro roteadores em cada eixo Y, totalizando 16 roteadores. A Figura 4.1 apresenta a configuração dessa rede.

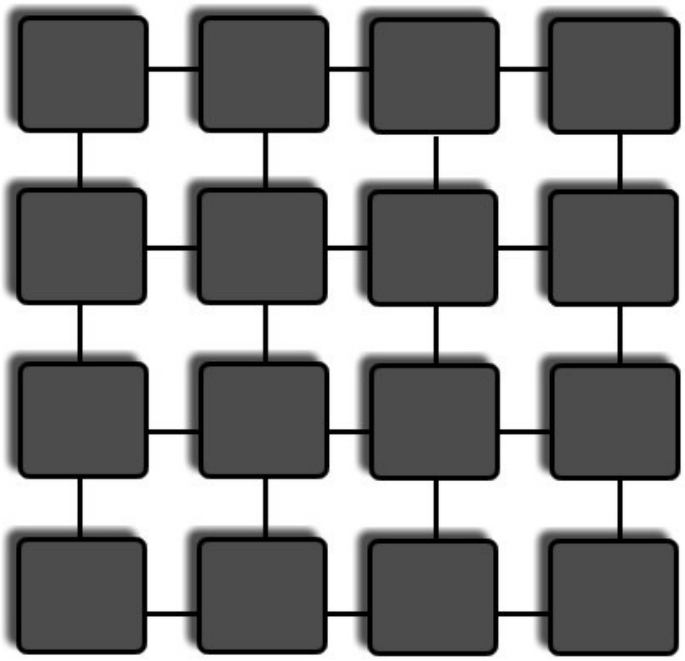


Figura 4.1. Rede Grelha 4x4

A chave da rede Hermes permite a comunicação com até quatro portas externas e mais uma porta local. Tomando como exemplo o nodo com endereço 22, ele se comunica com os nodos 12, 32, 21 e 23. Nos rotedores localizados em posições extremas em cada eixo as portas são isoladas para diminuir o custo da chave. A chave 00, por exemplo, possui as portas Oeste e Sul isoladas, uma vez que elas não são utilizadas. Isso é realizado por meio do aterramento das variáveis de entrada do roteador como mostra a Figura 4.2.

```
ack_txN0000(0)<=ack_rxN0100(1);
data_inN0000(1)<=(others=>'0');
rxN0000(1)<='0';
ack_txN0000(1)<='0';
data_inN0000(3)<=(others=>'0');
rxN0000(3)<='0';
ack_txN0000(3)<='0';
```

Figura 4.2. Isolamento de portas não utilizadas no roteador 00

Como é possível observar na Figura 4.2, as variáveis de entradas das portas Oeste, que possui índice 1, e da porta Sul, índice 3, recebem valores zero. Além disso, os *buffers* são gerados de acordo com as portas de cada chave. Assim, a chave 00 possui apenas três *buffers*, para as portas Leste, Norte e Local. Isso otimiza o custo de implementação da rede em hardware, pois a síntese da chave inclui apenas estruturas que serão utilizadas na rede. A Figura 4.3 apresenta a hierarquia dos principais módulos de uma rede Hermes 3x3, exemplificando duas chaves e seus respectivos *buffers*.

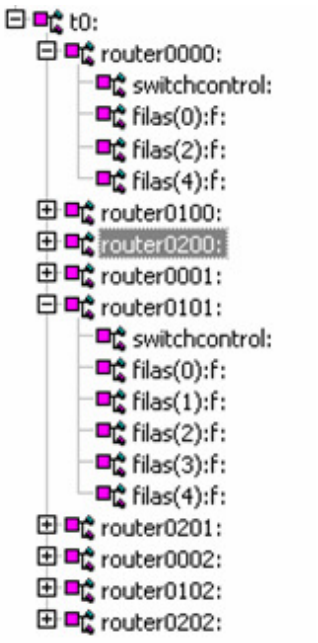


Figura 4.3. Hierarquia do módulos da rede Hermes 3x3

Como pode ser observado na Figura 4.3, o roteador 00 possui três *buffers,* enquanto que o roteador 11 apresenta cinco. Isso permite a otimização da síntese da rede e permite melhor aproveitamento de espaço no chip. É possível notar também que cada roteador possui um módulo de controle associado, responsável por estabelecer e tratar comunicações necessárias. Nas implementações realizadas cada *buffer* possui profundidade 16, o que os possibilita armazenar até 16 *flits* simultaneamente.

Por apresentar menor complexidade de implementação e menor custo, o controle de fluxo adotado nos testes e implementações foi o *Handshake*. O algoritmo de roteamento da rede Hermes é o XY, como mencionado no Capítulo 3. Para possibilitar a execução do algoritmo, são atribuídos endereços a cada um dos roteadores. O endereçamento em uma rede 4x4 é realizado com a utilização de oito bits, como apresentado na Figura 4.4. Cada roteador recebe um endereço único que indica sua posição na rede.

```
constant N0000: integer :=0;
constant ADDRESSN0000: std_logic_vector(7 downto 0) :="00000000";
constant N0100: integer :=1;
constant ADDRESSN0100: std_logic_vector(7 downto 0) :="00010000";
constant N0200: integer :=2;
constant ADDRESSN0200: std_logic_vector(7 downto 0) :="00100000";
constant N0300: integer :=3;
constant ADDRESSN0300: std_logic_vector(7 downto 0) :="00110000";
constant N0001: integer :=4;
constant ADDRESSN0001: std_logic_vector(7 downto 0) :="00000001";
constant N0101: integer :=5;
constant ADDRESSN0101: std_logic_vector(7 downto 0) :="00010001";
constant N0201: integer :=6;
constant ADDRESSN0201: std_logic_vector(7 downto 0) :="00100001";
constant N0301: integer :=7;
constant ADDRESSN0301: std_logic_vector(7 downto 0) :="00110001";
constant N0002: integer :=8;
constant ADDRESSN0002: std_logic_vector(7 downto 0) :="00000010";
constant N0102: integer :=9;
constant ADDRESSN0102: std_logic_vector(7 downto 0) :="00010010";
constant N0202: integer :=10;
constant ADDRESSN0202: std_logic_vector(7 downto 0) :="00100010";
constant N0302: integer :=11;
constant ADDRESSN0302: std_logic_vector(7 downto 0) :="00110010";
constant N0003: integer :=12;
constant ADDRESSN0003: std_logic_vector(7 downto 0) :="00000011";
constant N0103: integer :=13;
constant ADDRESSN0103: std_logic_vector(7 downto 0) :="00010011";
constant N0203: integer :=14;
constant ADDRESSN0203: std_logic_vector(7 downto 0) :="00100011";
constant N0303: integer :=15;
constant ADDRESSN0303: std_logic_vector(7 downto 0) :="00110011";
```

Figura 4.4. Endereçamento dos roteadores na rede Hermes 4x4

No exemplo apresentado na Figura 4.4, os quatro primeiro bits indicam a posição do roteador no eixo X e os quatro últimos bits estabelecem a posição do roteador no eixo Y. Tomando como exemplo o roteador 23 é possível verificar o endereçamento. Os quatro primeiro bits da constante ADDRESSN0203, que indica o endereço do roteador, possuem o valor “0010”, que representa o valor 2 em binário. Já os quatro últimos bits possuem o valor “0011”, que representa o valor 3 em binário. Esses dados são utilizados para comparar o endereço destino de um pacote com o endereço do roteador. Essa comparação determina a direção que o pacote deve seguir ou se ele já chegou em seu destino, pois quando o endereço especificado no cabeçalho do pacote é igual ao endereço do roteador indica que o pacote está em seu destino. O modo de endereçamento e a forma como é realizada a comparação dos endereços tiveram que ser alteradas para permitir a implementação de algumas das topologias analisadas.

A partir desses parâmetros e da rede Hermes gerada pelo *framework* MAIA, alterações foram realizadas modificando a topologia da rede e verificando as implicações que ocorreram

no desempenho e custo da rede. As topologias implementadas são discutidas nas seções seguintes.

4.2 Toróide

A topologia Toróide é baseada na Grelha, mas mantém conexões interligando os nodos nas extremidades. A Figura 4.5 apresenta a rede implementada, também com quatro roteadores por eixo.

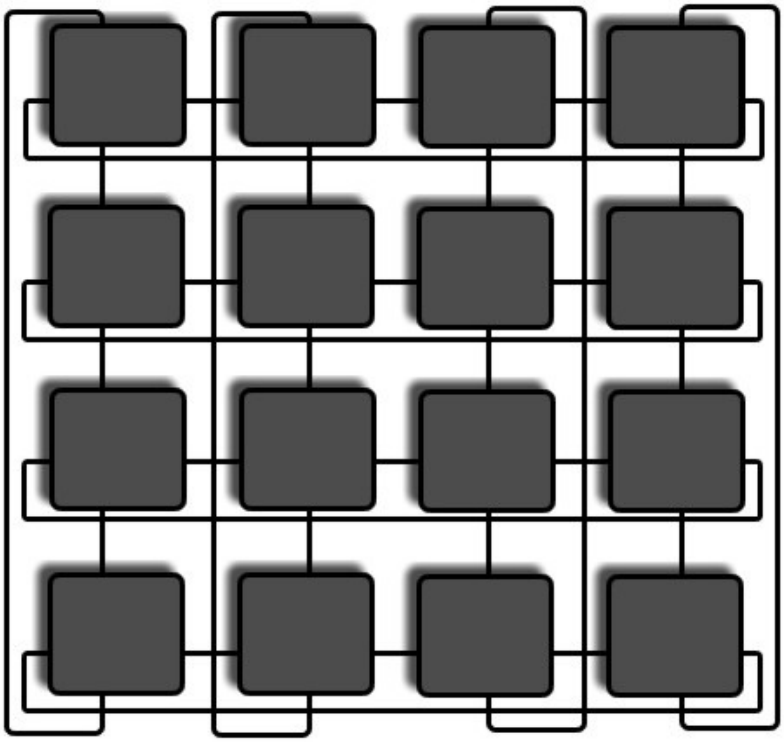


Figura 4.5. Rede com topologia Toróide 4x4

Inicialmente foram criados os *links* interligando as extremidades. Isso é realizado conectando as entradas aterradas na topologia Grelha. A interconexão é realizada verificando o endereço do roteador em determinado eixo. Isso é realizado por meio de quatro constantes utilizadas nos parâmetros da rede, apresentadas na Figura 4.6.

```
constant MIN_X : integer := 0;
constant MIN_Y : integer := 0;
constant MAX_X : integer := 3;
constant MAX_Y : integer := 3;
```

Figura 4.6. Valores mínimo e máximo por eixo em uma rede Toróide 4x4

Com esses valores é possível estabelecer as conexões que faltam para formar uma rede Toróide. Assim, se o endereço de um roteador *LxLy* no eixo X for igual a *MIN_X*, o roteador não possui conexões na porta Oeste originalmente. Dessa forma, essa porta deve ser conectada à porta Leste do roteador *MAX_X Ly*. De forma análoga, se o endereço de um roteador *LxLy* no eixo X for igual a *MAX_X*, o roteador não possui conexão na porta Leste e deve ser conectado à porta Oeste do roteador endereçado por *MIN_X Ly*. As conexões das portas no eixo Y são realizadas da mesma maneira. Um roteador *LxLy* não possuirá conexões na porta Sul caso *Ly* for igual *MIN_Y*. Assim, essa porta deve ser conectada à porta Norte do roteador *Lx MAX_Y*. Por fim, caso *Ly = MAX_Y*, o roteador não possui conexões na porta Norte e essa porta deve ser conectada à porta Sul do roteador *Lx MIN_Y*.

Estabelecidas as conexões, é necessário alterar o algoritmo de roteamento para utilizar os *links* recém criados. O algoritmo desenvolvido tem como base o algoritmo XY anteriormente descrito. Um pacote percorre inicialmente o eixo X até encontrar um nodo com o endereço indicado como destino do pacote no nesse eixo. Então, o pacote segue pelo eixo Y até encontrar seu destino. A diferença introduzida no algoritmo XY para rede Toróide é que ele teve que adaptado para aproveitar os *links* que interligam as extremidades. O roteamento parte do princípio de comparar a quantidade de roteadores entre o roteador em questão e o roteador alvo. Se entre eles existir mais da metade dos roteadores no mesmo eixo, significa que o caminho mais curto é utilizando os *links* das extremidades. Dessa forma, considerando *LxLy* o endereço do roteador que está processando o roteamento e *TxTy* o endereço do roteador alvo indicado no cabeçalho, o roteamento é realizado de acordo com as seguintes condições:

- Caso *Lx < Tx*: Se *Tx-Lx > MAX_X/2*, a direção no eixo X será a oeste. Caso contrário a direção neste eixo será a leste;
- Caso *Lx > Tx: Se Lx-Tx <= MAX_X/2*, a direção no eixo X será a oeste. Caso essas as condições não sejam satisfeitas a direção será a leste;
- *Caso Ly < Ty: Se Ty-Ly > MAX_Y/2*, a direção no eixo Y será a sul. Caso contrário a direção neste eixo será a norte;
- Caso *Ly > Ty: Se Ly-Ty <= MAX_Y/2*, a direção no eixo Y será a sul. Caso essas as condições não sejam satisfeitas a direção será a norte.

A Figura 4.7 exemplifica algumas dessas situações.

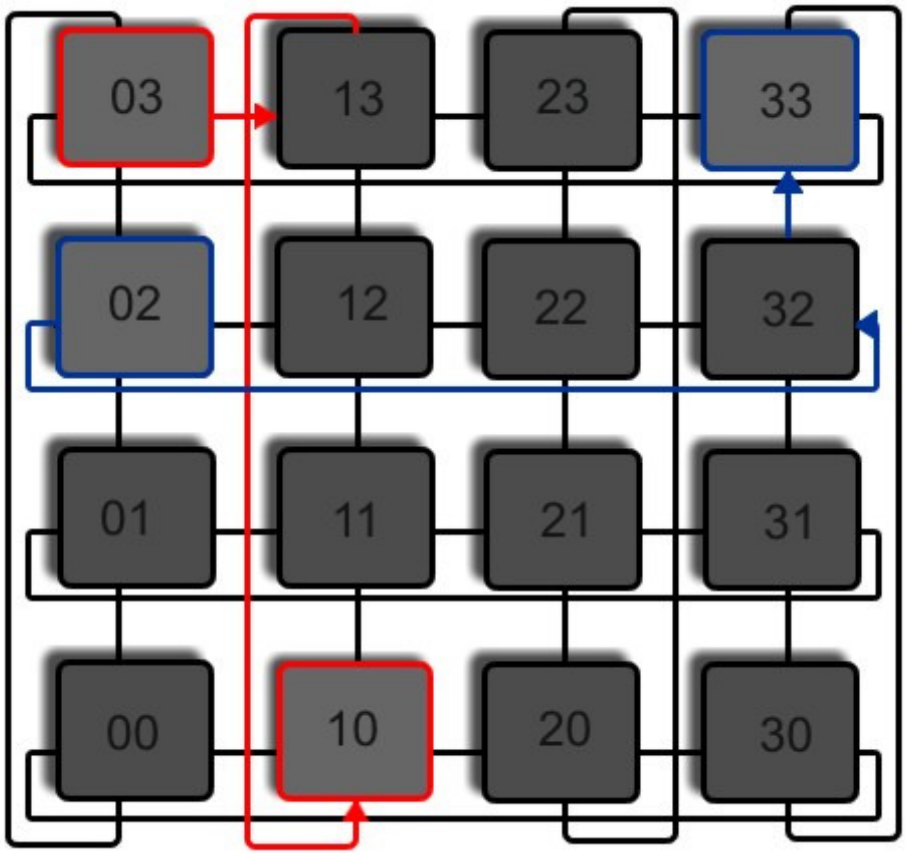


Figura 4.7. Exemplo de roteamento na rede Toróide

A Figura 4.7 apresenta duas rotas a serem traçadas simultaneamente. Na primeira, o roteador 03 deseja enviar um pacote para o roteador 10. Assim, analisando o eixo X, $Lx = 0$, $Tx = 1$ e $Tx\text{-}LX = 1$. Como $Lx < Tx$ e $Tx\text{-}Lx < MAX\_X/2$, a direção tomada é a leste, com o pacote indo para o roteador 13. Nesse roteador $Lx = Tx$, então o eixo Y é analisado. Nesse caso $Ly = 3$ e $Ty = 0$. Então, como $Ly > Ty$ e $Ly\text{-}Ty > MAX\_Y/2$, a direção a ser tomada é a norte, fazendo com que o pacote chegue a seu destino.

No segundo exemplo da Figura 4.7, o roteador origem é o 02 e o roteador destino é o 33. Dessa forma, tem-se inicialmente $Lx = 0$ e $Tx = 3$. Como $Lx < Tx$ e $Tx - Lx > MAX\_X/2$, o pacote é enviado pela porta oeste. Chegando ao roteador 32 é verificado apenas o eixo Y. Assim, tem-se $Ly = 2$ e $Ty = 3$. Como $Ly < Ty$ e $Ty\text{-}Ly < MAX\_Y/2$ o pacote toma a direção norte, chegando a seu destino.

Essa implementação garante um algoritmo livre de *deadlock*, uma vez que um pacote sempre seguirá a mesma direção em um mesmo eixo, não sendo possível a ele entrar em um ciclo. A rede foi validada por meio de simulações funcionais, que verificavam a conexão de cada um dos roteadores da rede. A Figura 4.8 apresenta um exemplo de conexão entre os nodos 00 e 03, realizando a conexão pelo novo canal inserido.

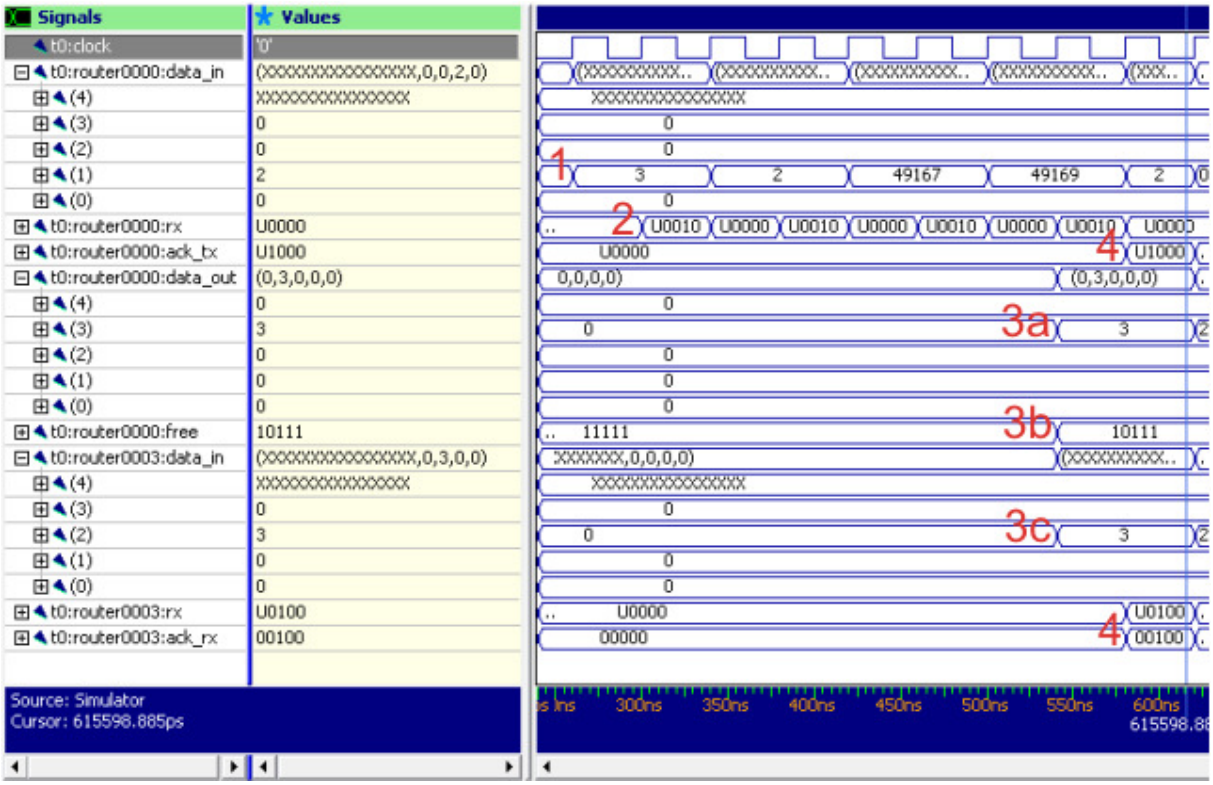


Figura 4.8. Transmissão de um pacote entre os roteadores 00 e 03 na rede Toróide

A transmissão ocorre por meio dos seguintes passos:

1. O roteador 00 recebe um pacote na porta oeste (índice 1), endereçado para o roteador 03;

2. O bit *rx* da porta 1 é ativado para 1, indicando que há um dado a ser transmitido na porta Oeste;

3a. O dado é transmitido pela porta sul (índice 3);

3b. O vetor de roteamento é atualizado, indicando que a porta Sul está ocupada;

3c. O roteador 03 recebe o dado através da porta Norte (índice 2);

4. O sinal *ack_rx* da porta Norte é atualizado no roteador 03, indicando que a transmissão do *flit* foi completada. O roteador 00 recebe esse sinal por meio da entrada *ack_tx*, que é também atualizada.

4.3 Anel

Constituída apenas de duas interconexões externas, as chaves na rede Anel foram adaptadas para conter três portas: uma porta para a comunicação local e outras duas para a comunicação com nodos vizinhos. A Figura 4.9 apresenta a configuração de tal roteador, que conta também com três *buffers* posicionados nas portas de entradas.

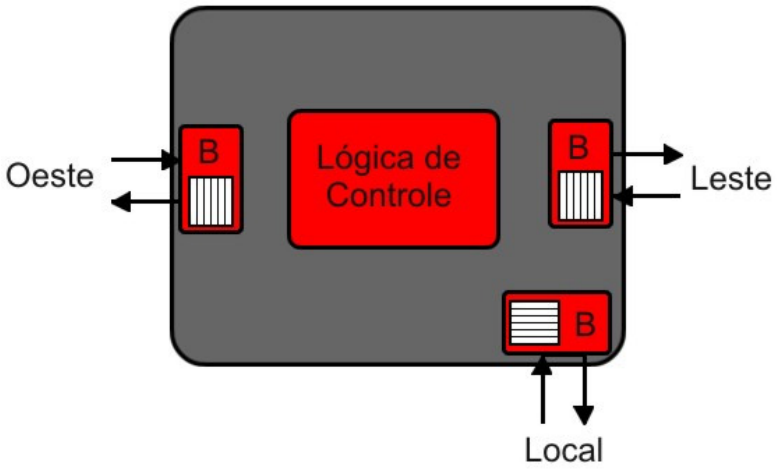


Figura 4.9. Configuração do roteador na rede Anel

A interligação dos nodos é realizada considerando apenas um eixo. Dessa forma, o endereçamento de cada roteador teve que ser alterado para possibilitar o roteamento em anel. Os endereços foram determinados utilizando um dos eixos, no caso o eixo X. Assim, para cada roteador foi atribuído um único valor que determina sua posição na rede. A Figura 4.10 apresenta o endereçamento utilizado na rede analisada, que conta com 16 roteadores.

```
constant NO000: integer :=0;
constant ADDRESSN0000: std_logic_vector(7 downto 0) :="00000000";
constant NO100: integer :=1;
constant ADDRESSN0100: std_logic_vector(7 downto 0) :="00010000";
constant NO200: integer :=2;
constant ADDRESSN0200: std_logic_vector(7 downto 0) :="00100000";
constant NO300: integer :=3;
constant ADDRESSN0300: std_logic_vector(7 downto 0) :="00110000";
constant NO400: integer :=4;
constant ADDRESSN0400: std_logic_vector(7 downto 0) :="01000000";
constant NO500: integer :=5;
constant ADDRESSN0500: std_logic_vector(7 downto 0) :="01010000";
constant NO600: integer :=6;
constant ADDRESSN0600: std_logic_vector(7 downto 0) :="01100000";
constant NO700: integer :=7;
constant ADDRESSN0700: std_logic_vector(7 downto 0) :="01110000";
constant NO001: integer :=8;
constant ADDRESSN0800: std_logic_vector(7 downto 0) :="10000000";
constant NO101: integer :=9;
constant ADDRESSN0900: std_logic_vector(7 downto 0) :="10010000";
constant NO201: integer :=10;
constant ADDRESSN1000: std_logic_vector(7 downto 0) :="10100000";
constant NO301: integer :=11;
constant ADDRESSN1100: std_logic_vector(7 downto 0) :="10110000";
constant NO401: integer :=12;
constant ADDRESSN1200: std_logic_vector(7 downto 0) :="11000000";
constant NO501: integer :=13;
constant ADDRESSN1300: std_logic_vector(7 downto 0) :="11010000";
constant NO601: integer :=14;
constant ADDRESSN1400: std_logic_vector(7 downto 0) :="11100000";
constant NO701: integer :=15;
constant ADDRESSN1500: std_logic_vector(7 downto 0) :="11110000";
```

Figura 4.10. Endereço dos roteadores na rede Anel

Apenas os quatro primeiros bits de cada endereço são considerados. Para estabelecer a conexão entre os nodos é comparada a posição do roteador no anel com os menores e maiores endereços existentes na rede. A interconexão é realizada da seguinte forma:

- Se *Lx* <> *MAX_X*, a porta leste (índice 0) se conecta à porta oeste (índice 1) do roteador *Lx* + 1;
- Se *Lx* = *MAX_X*, a porta Leste se conecta à porta Oeste do roteador *MIN_X*;
- Se *Lx* <> *MIN_X*, a porta Oeste se conecta à porta Leste do roteador *Lx* – 1;
- Se *Lx* = *MIN_X*, a porta oeste se conecta à porta Leste do roteador *MAX_X*.

Apenas os quatro primeiros bits de cada endereço são considerados no momento da execução do algoritmo de roteamento. O roteamento compara o endereço do roteador analisado com o endereço destino indicado no cabeçalho do pacote. A direção é determinada seguindo o mesmo princípio do roteamento de um único eixo na rede Toróide:

- Se *Lx* < *Tx* e *Tx*-*Lx* > *MAX_X/2*, o pacote será enviado em sentido anti-horário;
- Se *Lx* < *Tx* e *Tx*-*Lx* <= *MAX_X/2*, o pacote será enviado em sentido horário;
- Se *Lx* > *Tx* e *Lx*-*Tx* <= *MAX_X/2*, o pacote será enviado em sentido anti-horário;
- Se *Lx* > *Tx* e *Lx*-*Tx* > *MAX_X/2*, o pacote será enviado em sentido horário.

Assim, um pacote sempre é enviado para a direção que possui menos nodos entre o roteador analisado e o roteador destino. O algoritmo também é livre de *deadlock*, uma vez que um pacote sempre é enviado em uma mesma direção até encontrar seu destino. O algoritmo não permite que um pacote mude sua direção de roteamento, uma vez que ele tenha sido introduzido na rede pelo roteador origem.

4.4 Spidergon

A rede Spidergon foi implementada com base na rede Anel. O mesmo modelo de endereçamento foi utilizado, no qual é considerado apenas um dois eixos. Entretanto, a rede Spidergon conta com uma porta adicional, denominada *cross*, utilizada para encurtar a distância entre qualquer par de nodos. A interconexão dessas portas é realizada conectando um nodo *i* qualquer ao nodo *i* + *N/2,* caso *i* < *N/2*, ou ao nodo *i* – *N/2*, caso *i* >= *N/2*. *N* indica o número total de nodos da rede. A rede Spidegon implementada com 16 roteadores possui a configuração apresentada na Figura 4.11.

A Spidergon é uma rede de grau três, com cada nodo conectado a outros três nodos adjacentes. As alterações fazem com as chaves sejam homogêneas, o que indica que a mesma

configuração de chave é utilizada em todos os nodos, que mantém três conexões externas e uma porta Local.

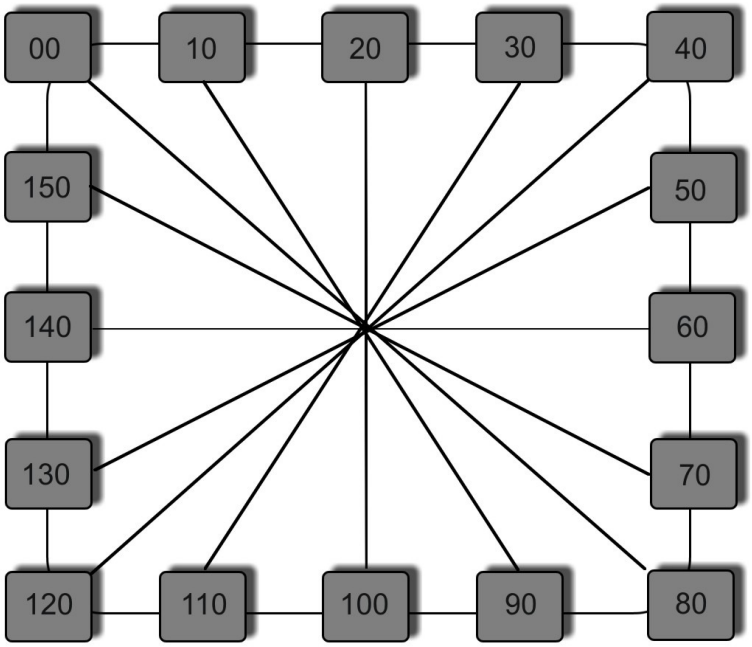


Figura 4.11. Rede Spidergon com 16 roteadores

O algoritmo de roteamento escolhido é determinístico, fazendo com que para um determinado par de roteadores o mesmo caminho seja sempre escolhido. Esse caminho é mínimo, contendo a menor distância entre os dois roteadores. O algoritmo verifica a distância entre o roteador analisado e o destino do pacote. Se a distância entre os dois for maior que $\lceil N/4 \rceil$, significa que o caminho mais curto para um pacote chegar a seu destino é utilizando o *link* ligado à porta *cross*. Dessa forma, a porta em questão é escolhida pelo algoritmo de roteamento. Caso contrário, uma das duas portas laterais será escolhida.

O algoritmo utiliza o método *cross-first*, no qual um pacote primeiro cruza a rede, se necessário, para então ser encaminhado pelas portas laterais do roteador. Isso significa que o único nodo que utilizará o *link cross* durante o roteamento de um pacote é o próprio nodo origem. O roteamento em sentido horário ou anti-horário pelas portas laterais ocorre da mesma maneira que na rede Anel. O algoritmo também é livre de *deadlock*, uma vez que sempre que um pacote é encaminhado para a porta *cross*, não é possível que ele retorne por esse mesmo *link*. A validação do novo canal inserido foi realizada e um exemplo é apresentado na Figura 4.12, mostrando a transmissão de um pacote entre os roteadores 00 e 70.

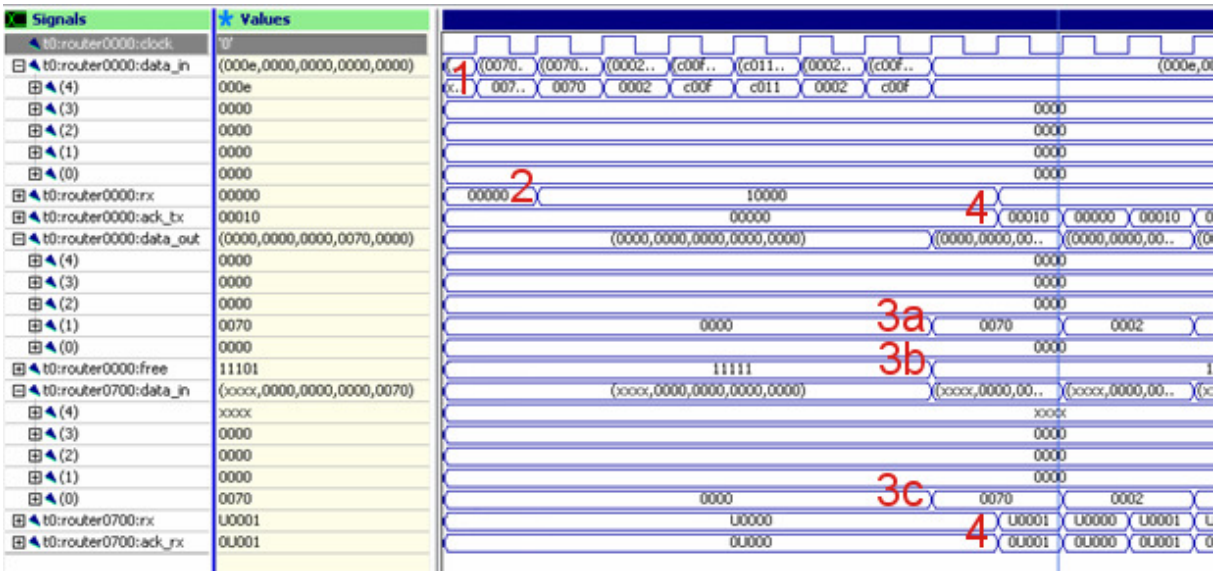


Figura 4.12. Transmissão de um pacote entre os roteadores 00 e 70 na rede Spidergon

A transmissão ocorre por meio dos seguintes passos, indicados na Figura 4.12:

1. O roteador 00 contém um pacote na porta local (índice 4), endereçado para o roteador 70;

2. O bit ‘rx’ da porta 4 é ativado para 1, indicando que há um dado a ser transmitido na porta local;

3a. O dado é transmitido pela porta norte (índice 2);

3b. O vetor de roteamento é atualizado, indicando que a porta norte está ocupada;

3c. O roteador 70 recebe o dado através da porta sul (índice 3);

4. O sinal ‘ack_rx’ da porta sul é atualizado no roteador 70, indicando que a transmissão do *flit* foi completada. O roteador 00 recebe esse sinal por meio da entrada ‘ack_tx’, que é atualizada.

O algoritmo é livre de *deadlock*, pois uma vez utilizado um dos *links* que cruzam a rede, o algoritmo garante que esse *link* não será mais utilizado e o roteamento no próximo roteador será utilizando uma das portas laterais. Além disso, assim que o roteamento passa a ser pelo sentido horário ou anti-horário, a direção não é mais alterada até que o pacote chegue a seu destino.

4.5 Hipercubo

O uso de uma topologia Hipercubo em uma rede intrachip foi proposta na rede MECS (Grot et al., 2009). A rede utiliza uma topologia Cubo Expresso (Dally, 1991) na qual um

Hipercubo é inserido dentro de outro, como mostra Figura 4.13, que apresenta uma rede com 16 nodos.

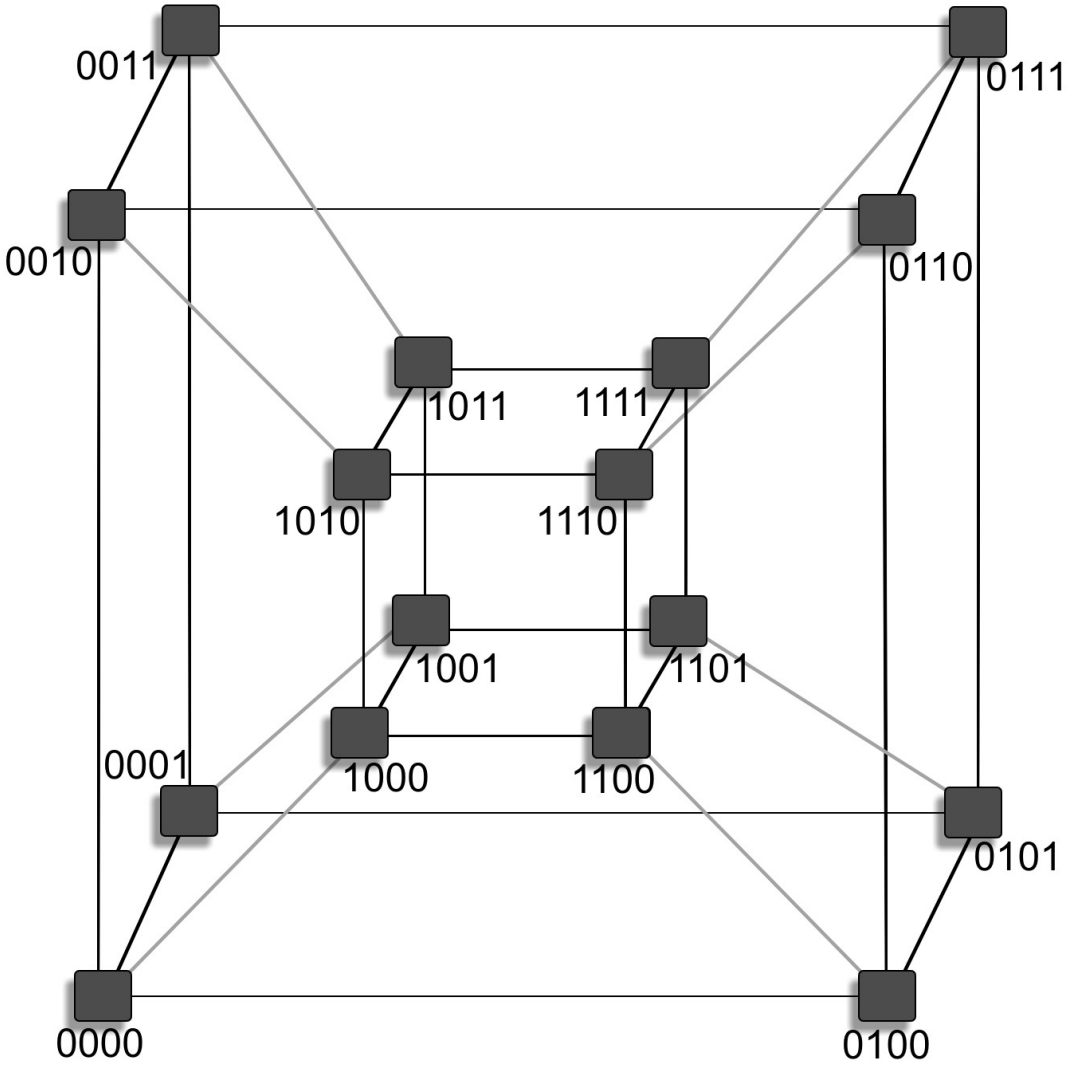


Figura 4.13. Rede Cubo Expresso com 16 nodos

A Figura 4.13 apresenta também o endereçamento utilizado na implementação da rede. Por tratar de estruturas interligadas em forma de cubo, a rede passa a considerar as dimensões X, Y e Z, correspondentes aos eixos existentes na rede. O endereço, composto por oito bits, contém quatro informações diferentes. Os dois primeiros bits indicam de qual Cubo o nodo faz parte, 00 indica que o nodo está no cubo externo e 01 que o nodo faz parte do cubo interno. Os dois próximos bits indicam a posição do nodo no eixo X do cubo em que ele está situado. Os dois bits seguintes indicam a posição do nodo no eixo Y e os dois últimos bits a posição do nodo no eixo Z. Dessa forma, com oito bits é possível endereçar um rede de até 32 nodos. A Figura 4.14 apresenta a definição do endereçamento utilizado no código VHDL implementado.

```
constant N0000: integer :=0;
constant ADDRESSN0000: std_logic_vector(7 downto 0) :="00000000";
constant N0100: integer :=1;
constant ADDRESSN0100: std_logic_vector(7 downto 0) :="00010000";
constant N0001: integer :=2;
constant ADDRESSN0001: std_logic_vector(7 downto 0) :="00000001";
constant N0101: integer :=3;
constant ADDRESSN0101: std_logic_vector(7 downto 0) :="00010001";
constant N0010: integer :=4;
constant ADDRESSN0010: std_logic_vector(7 downto 0) :="00000100";
constant N0110: integer :=5;
constant ADDRESSN0110: std_logic_vector(7 downto 0) :="00010100";
constant N0011: integer :=6;
constant ADDRESSN0011: std_logic_vector(7 downto 0) :="00000101";
constant N0111: integer :=7;
constant ADDRESSN0111: std_logic_vector(7 downto 0) :="00010101";
constant N1000: integer :=8;
constant ADDRESSN1000: std_logic_vector(7 downto 0) :="01000000";
constant N1100: integer :=9;
constant ADDRESSN1100: std_logic_vector(7 downto 0) :="01010000";
constant N1001: integer :=10;
constant ADDRESSN1001: std_logic_vector(7 downto 0) :="01000001";
constant N1101: integer :=11;
constant ADDRESSN1101: std_logic_vector(7 downto 0) :="01010001";
constant N1010: integer :=12;
constant ADDRESSN1010: std_logic_vector(7 downto 0) :="01000100";
constant N1110: integer :=13;
constant ADDRESSN1110: std_logic_vector(7 downto 0) :="01010100";
constant N1011: integer :=14;
constant ADDRESSN1011: std_logic_vector(7 downto 0) :="01000101";
constant N1111: integer :=15;
constant ADDRESSN1111: std_logic_vector(7 downto 0) :="01010101";
```

Figura 4.14. Endereçamento da rede Cubo Expresso com 16 nodos

O algoritmo de roteamento implementado é determinístico. Considerando um roteador com endereço *LcLxLyLz*, o algoritmo faz a verificação de cada parte do endereço destino *TcTxTyTz* do pacote. Se a parte do endereço analisada não for igual à parte do roteador analisado, o pacote é encaminhado para o outro roteador do eixo. Cada roteador possui quatro conexões conectando o roteador ao outro cubo, a outro roteador no eixo X, a outro roteador no eixo Y e a outro roteador no eixo Z. Assim, o algoritmo não precisa determinar a direção que um pacote irá tomar, mas apenas para qual eixo, e consequentemente qual porta, o pacote será enviado. As portas originais foram renomeadas para PortaC, PortaX, PortaY e PortaZ, indicando o eixo ao qual o roteador está conectado. O funcionamento do algoritmo de roteamento é apresentado na figura 4.15.

```
Se Lc <> Tc => PortaC := dado;
Senão, se Lx <> Tx => PortaX := dado;
Senão, se Ly <> Ty => PortaY := dado;
Senão, se Lz <> Tz => PortaZ := dado;
Senão => Porta Local := dado.
```

Figura 4.15. Algoritmo de roteamento da rede Cubo Expresso

O algoritmo é livre de *deadlock*, pois um pacote é sempre roteado no máximo uma vez em cada eixo, garantindo que após ser encaminhado para outro eixo não há como ele retornar por esse mesmo eixo. O algoritmo também garante um máximo de quatro saltos no pior caso de roteamento, que é quando um pacote deve ser roteado através dos quatro eixos.

4.6 Algoritmo Semidinâmico para Topologia Toróide

Em busca de uma melhora de desempenho da rede Toróide também foi implementado um algoritmo semidinâmico para o roteamento dos pacotes. O algoritmo é baseado no modelo XY, porém um pacote não precisa ser necessariamente roteado por todo o eixo X inicialmente. Caso um roteador LxLy precise enviar um pacote para um roteador TxTy, sendo Lx <> Tx e Ly <> Ty, mas não haja possibilidade momentânea de enviar o pacote pela porta correspondente no eixo X, o algoritmo envia o pacote primeiramente pelo eixo Y. A Figura 4.16 exemplifica o funcionamento desse algoritmo.

Como pode ser observado na Figura 4.16, duas rotas são estabelecidas simultaneamente. Na primeira o roteador 03 deseja enviar um pacote para o roteador 10. O roteador origem inicialmente tenta enviar o pacote pelo eixo X, tentando estabelecer uma conexão com o roteador 13. Porém, não é possível estabelecer uma conexão com esse roteador, seja porque o *link* está ocupado ou porque não existe espaço em seu *buffer* de armazenamento. Nesse caso, o algoritmo semidinâmico tenta estabelecer uma conexão por meio do eixo Y. No caso, ele consegue uma conexão com o roteador 00, que então entrega o pacote para o roteador destino, o roteador 10. A mesma situação ocorre quando se tenta estabelecer uma conexão entre os roteadores 01 e 32. O caminho percorrido pelo algoritmo XY seria 01 – 31 – 32. Porém, com a impossibilidade de estabelecer uma conexão entre os roteadores 01 e 31, o caminho estabelecido pelo algoritmo semidinâmico é 01 – 02 – 32. Como pode ser observado, o algoritmo sempre utiliza a mesma quantidade de saltos para realizar a entrega dos pacotes, independente de qual caminho ele siga inicialmente.

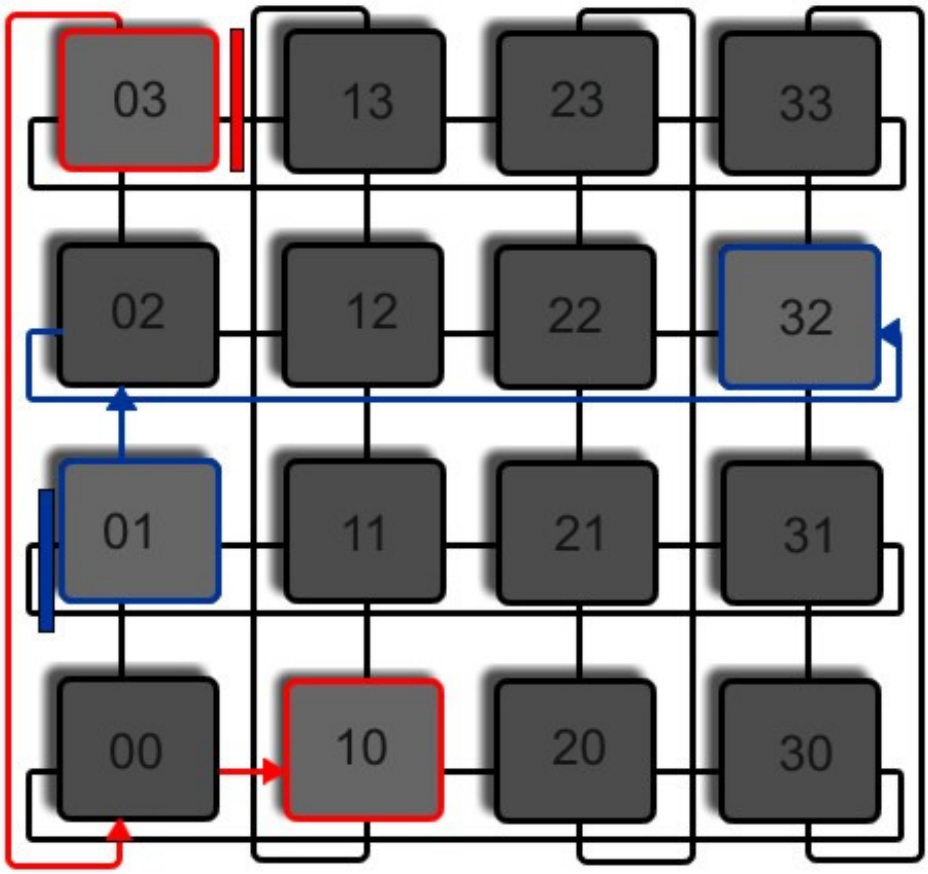


Figura 4.16. Exemplo de funcionamento do algoritmo semidinâmico

Para possibilitar o funcionamento do algoritmo, é necessária a utilização de sinais adicionais, *buffer_tx* e *buffer_rx*, nas portas de entrada e saída de um roteador. A Figura 4.17 apresenta a nova configuração de uma conexão entre dois roteadores.

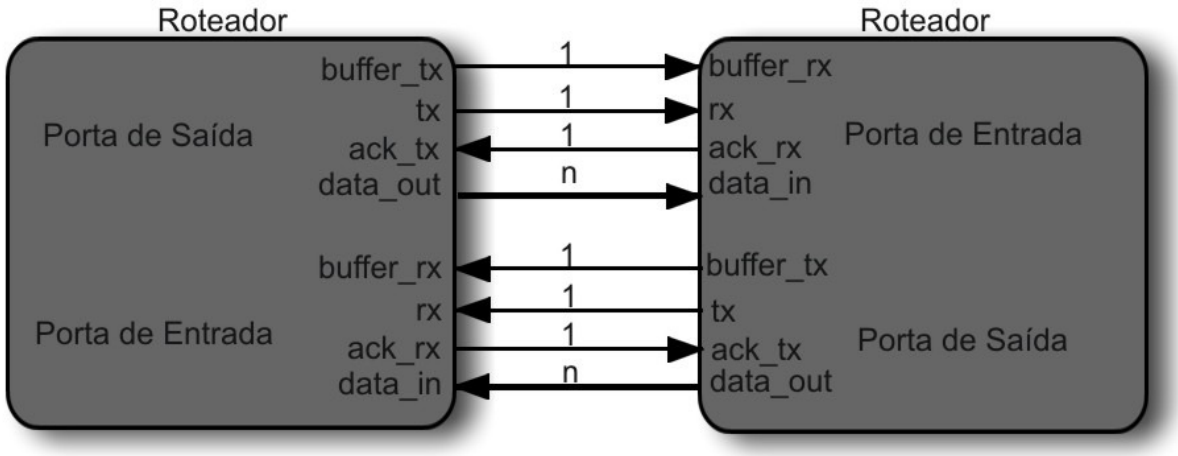


Figura 4.17. Configuração das portas de entrada e saída para possibilitar a execução do algoritmo semidinâmico

Os sinais adicionais indicam a disponibilidade de espaço no *buffer* do roteador vizinho. Assim, caso o *buffer* de uma porta de entrada esteja cheio, ele enviará um sinal indicando que não há mais espaço em seu *buffer*. De posse dessa informação, o algoritmo tenta estabelecer a conexão com o roteador vizinho no eixo Y, caso *Ly* <> *Ty*. Dessa forma, o pacote será roteado pelo eixo X em um próximo passo do roteamento. Entretanto, se *Ly* = *Ty*, o roteador armazena o pacote em seu *buffer* e aguarda a liberação da porta no eixo X. A definição da direção que um pacote irá tomar ocorre da mesma forma que no algoritmo XY. Isso garante que uma única direção será tomada em um eixo, evitando assim a ocorrência de *deadlock*.

O algoritmo desenvolvido é não-determinístico, uma vez que para um par de nodos, caminhos diferentes podem ser estabelecidos na comunicação. A validação do algoritmo foi realizada e um exemplo de conexão estabelecida entre dois roteadores é apresentada na Figura 4.18.

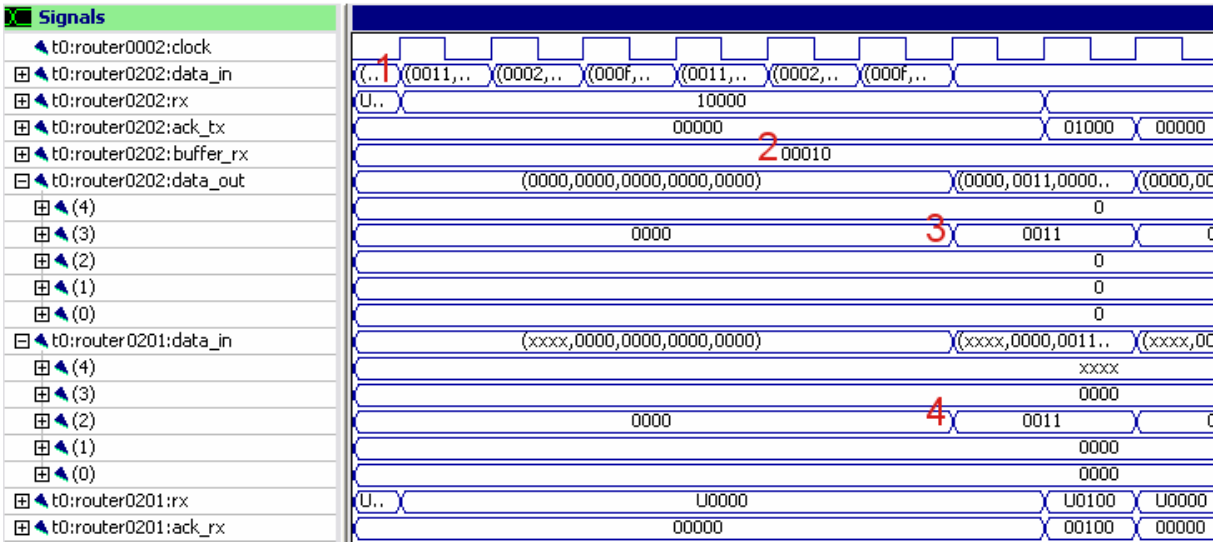


Figura 4.18. Transmissão de pacotes utilizando o algoritmo semidinâmico

1. O roteador 22 (0202) recebe um pacote endereçado para o roteador 11.
2. A transmissão normalmente ocorreria enviando o pacote para o roteador 12, situado a oeste do roteador 22 (porta 1). Porém, é verificado que o sinal *buffer_rx* dessa porta está ativo, indicando que o pacote não pode ser enviado por ela.
3. Então é estabelecida uma conexão por meio da porta 3 do roteador, que o interliga ao roteador 21 (0201), situado ao sul do roteador em questão.
4. O pacote então é recebido pelo roteador 21, que agora deverá enviá-lo por meio do eixo X até encontrar o seu destino.

4.7 Considerações Finais

O presente capítulo apresentou detalhes de implementação das topologias desenvolvidas a partir dos roteadores e protocolos de comunicação da rede Hermes. Topologias presentes na literatura foram adaptadas para permitir sua implementação. Um algoritmo de roteamento semidinâmico para topologias Toróide também foi apresentado, objetivando a redução do tempo de espera de pacotes que não podem ser imediatamente roteados. Exemplos utilizados na validação das implementações também foram apresentados, exemplificando o correto funcionamento das modificações realizadas.

CAPÍTULO

5

Resultados

Neste capítulo são discutidos os experimentos realizados para viabilizar a análise das redes desenvolvidas nos quesitos desempenho e custo. O custo é analisado por meio da síntese dos circuitos implementados. Já o desempenho é analisado nos quesitos latência e vazão. A latência indica o tempo gasto para um pacote chegar ao seu destino, iniciando a partir do momento em que o cabeçalho entra na rede e terminando assim que o último *flit* do pacote chega ao seu destino. Nos testes realizados a latência foi medida em ciclos de *clock*.

Já a vazão trata da quantidade de dados transmitidos durante determinado intervalo de tempo. Nas simulações realizadas a vazão é indicada pela taxa média de *flits* enviados durante o tempo de simulação.

5.1 Comparação de Desempenho entre Topologias

Para realizar a análise de desempenho foram inseridos pacotes nas portas locais de cada roteador, com um endereço de destino. A inserção e a análise foram realizadas com a ferramenta Traffic Mbps, presente no *framework* MAIA. A ferramenta permite a criação de tráfegos específicos ou aleatórios, indicando a quantidade de pacotes a serem enviados por cada roteador, o tamanho dos pacotes e a taxa de transmissão dos canais, entre outros fatores. A Figura 5.1 introduz um diagrama de bloco com os componentes utilizados para a realização de uma simulação.

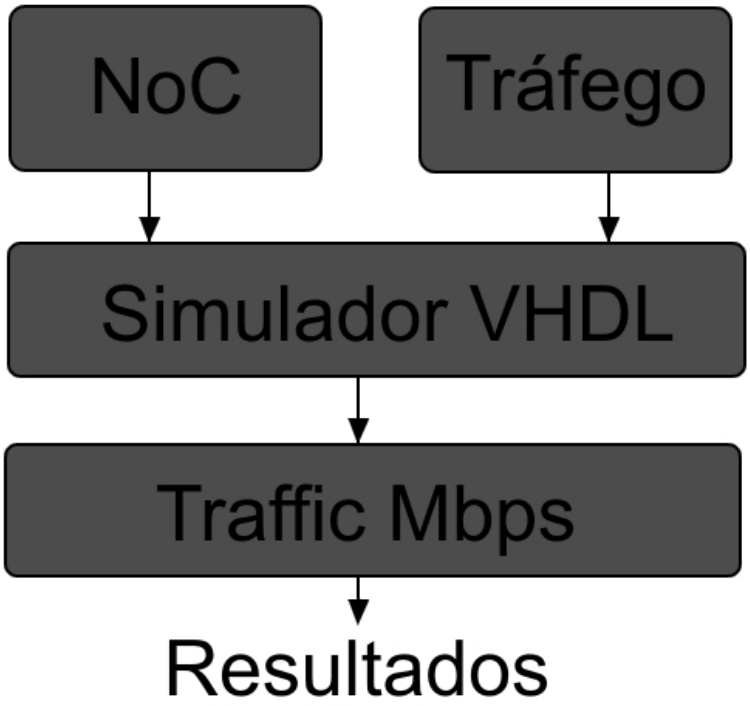


Figura 5.1. Componentes utilizados na execução das simulações

Como pode ser observado na Figura 5.1, as entradas da simulação são a rede implementada e o tráfego gerado com a ferramenta Traffic Mbps. Esses dois itens são utilizados por um simulador VHDL que executa o comportamento da rede de acordo com o tráfego gerado. O simulador utilizado neste trabalho foi o Modelsim SE, versão 6.4a (Mentor Graphics, 2009). A simulação é monitorada pelo Traffic Mbps, que analisa os dados que passam pelas portas de entrada e saída da rede analisada. O Traffic Mbps é descrito em System C, o que requer um simulador com capacidade de executar as duas linguagens simultaneamente.

Nas redes implementadas o modo de geração e análise do tráfego teve que ser alterado, uma vez que, com exceção da rede Toróide, o modo de endereçamento de cada rede é diferente. Assim, para prover resultados não dependentes da particularidade de um único tráfego, para cada configuração de tráfego, três diferentes tráfegos foram gerados. Os resultados aqui apresentados indicam a média desses tráfegos. As configurações utilizadas nas simulações são apresentadas na Tabela 5.1. É possível observar na tabela que se buscou a variação da quantidade de pacotes inseridos na rede, da quantidade de *flits* por pacote (tamanho do pacote) e da taxa de transmissão de cada simulação.

Tabela 5.1. Configurações das simulações realizadas

Descrição	Pacotes / roteador	Flits / pacote	Taxa de transmissão
Simulação 01	100	15	10% (80 Mbps)
Simulação 02	100	15	30% (240 Mbps)
Simulação 03	100	15	50% (400 Mbps)
Simulação 04	100	15	70% (560 Mbps)
Simulação 05	100	15	90% (720 Mbps)
Simulação 06	1000	15	10% (80 Mbps)
Simulação 07	1000	15	30% (240 Mbps)
Simulação 08	1000	15	50% (400 Mbps)
Simulação 09	1000	15	70% (560 Mbps)
Simulação 10	1000	15	90% (720 Mbps)
Simulação 11	100	30	10% (80 Mbps)
Simulação 12	100	30	30% (240 Mbps)
Simulação 13	100	30	50% (400 Mbps)
Simulação 14	100	30	70% (560 Mbps)
Simulação 15	100	30	90% (720 Mbps)
Simulação 16	1000	30	10% (80 Mbps)
Simulação 17	1000	30	30% (240 Mbps)
Simulação 18	1000	30	50% (400 Mbps)
Simulação 19	1000	30	70% (560 Mbps)
Simulação 20	1000	30	90% (720 Mbps)

Em todas as simulações as chaves foram configuradas com 16 posições em seus *buffers*, operavam a uma frequência de *clock* de 50 MHz e cada *flit* continha 16 bits. Os outros parâmetros foram variados. São quatro classes principais de simulações, contendo 100 pacotes por roteador e 15 *flits* por pacote, 1000 pacotes por roteador e 15 *flits* por pacote, 100 pacotes por roteador e 30 *flits* por pacote e 1000 pacotes por roteador e 30 *flits* por pacote. Além disso, em cada uma das classes foi alterada a taxa de transmissão dos dados, variando entre 10%, 30%, 50%, 70% e 90%. A taxa de transmissão indica o tempo que um roteador ficará ocioso antes de inserir novos pacotes na rede. Uma taxa de 100% indica que os pacotes são inseridos na rede em sequência, sem qualquer intervalo entre eles. Já uma taxa de 50% indica que existe um atraso entre a inserção de dois pacotes, levando o dobro de tempo para

inseri-los em relação a uma taxa de 100%. Dessa forma é possível analisar as redes sobre diferentes condições, seja com tráfego ameno ou intenso.

5.1.1 Latência

A Figura 5.2 apresenta a comparação da latência entre as cinco redes desenvolvidas considerando as configurações de simulações de 1 a 5, com 100 pacotes por roteador e 15 *flits* por pacote, totalizando 1.600 pacotes e 24.000 *flits* inseridos em cada simulação. Os resultados são apresentados de acordo com a quantidade média de ciclos de *clock* que os pacotes demoram para chegar ao seu destino a partir de um roteador fonte.

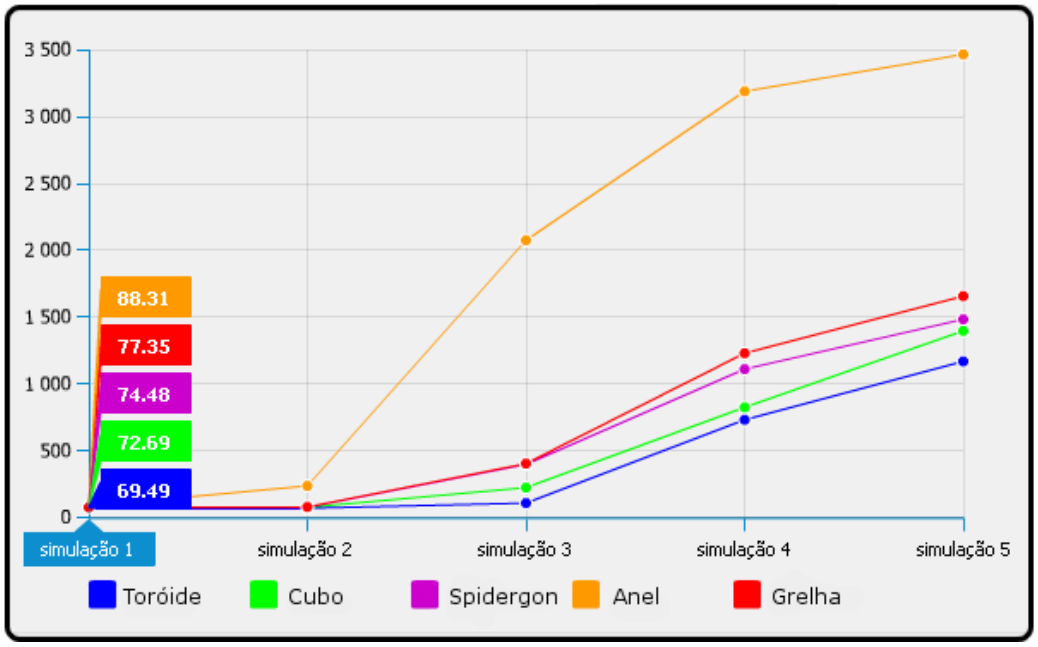


Figura 5.2. Latência nas simulações de 1 a 5

Como pode ser observado na Figura 5.2, a rede Anel obteve desempenho inferior que às demais. As outras quatro redes mantiveram valores aproximados com taxas de transmissão de 10 e 30%. Com taxa de transmissão de 50% a rede Toróide obteve melhor desempenho, com latência de 108,70. A rede Cubo Expresso (indicada pela legenda Cubo no gráfico) obteve desempenho inferior à Toróide, com 226,15. As redes Grelha e Spidergon obtiveram desempenhos semelhantes, com uma diferença de apenas 7 ciclos de *clock* entre as simulações dessas redes. Já com taxas de 70 e 90%, a Spidergon se sobressaiu em relação a Grelha, obtendo melhor desempenho que ela. Com taxa de transmissão de 90% a rede Spidergon obteve desempenho próximo ao da rede Cubo, porém ambas tiveram desempenho pior que a rede Toróide.

Já a Figura 5.3 apresenta a comparação da latência média nas simulações de 6 a 10, inserindo 1.000 pacotes por roteador e 15 *flits* por pacote, totalizando 16.000 pacotes e 240.000 *flits* inseridos na rede.

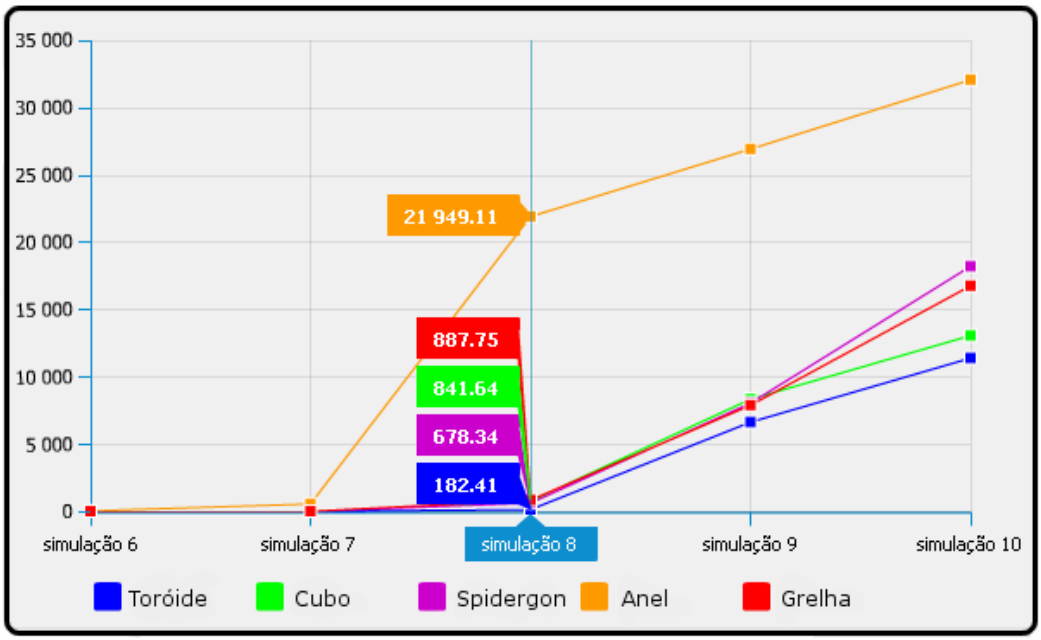


Figura 5.3. Latência nas simulações de 6 a 10

Como pode ser observado na Figura 5.3, os resultados mantiveram-se semelhantes com taxas de transmissões inferiores a 50%. Com taxa de 50% a rede Toróide obteve significativa melhora em relação às demais, permanecendo com melhor desempenho em todas as outras simulações. A rede Spidergon obteve uma queda de desempenho quando utilizado tráfego intenso. Com taxas até 50%, essa rede mantinha-se com melhor desempenho que as redes Grelha e Cubo. Porém, com taxas de transmissão de 70 e 90%, essas redes superam a Spidergon no quesito latência. Com taxa de 90% há ainda uma significativa melhora com o uso da rede Cubo em relação à Grelha, que não consegue acompanho o desempenho da primeira com tráfego intenso.

A Figura 5.4 apresenta a latência média nas simulações de 11 a 15. Nessas simulações foram introduzidos 100 pacotes por roteador e 30 *flits* por pacote, totalizando 1.600 pacotes e 48.000 *flits*.

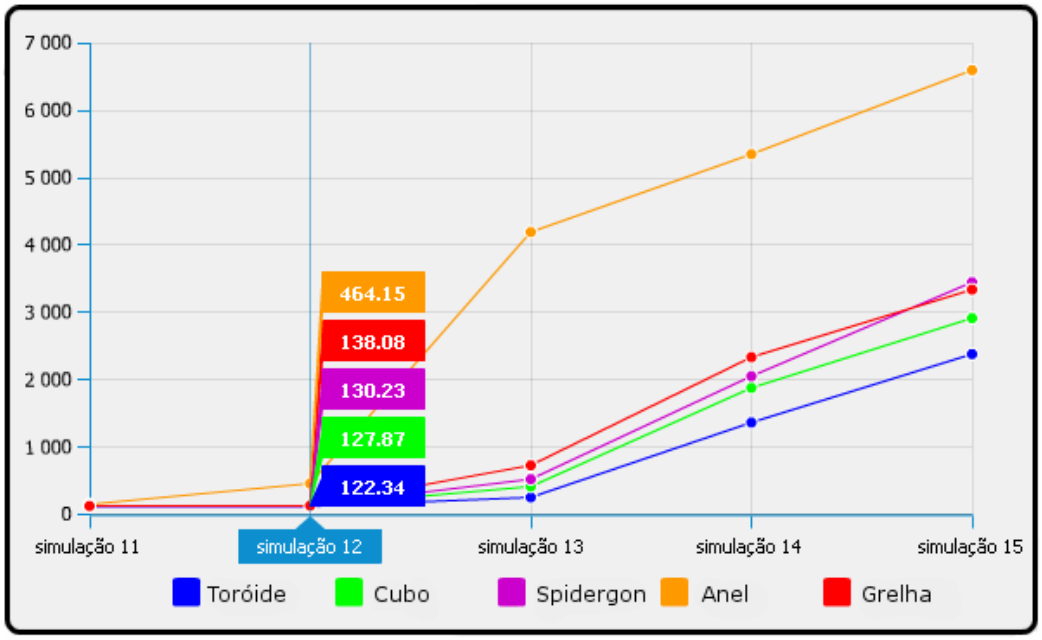


Figura 5.4. Latência nas simulações de 11 a 15

Os resultados mostraram equilíbrio no desempenho das redes Cubo e Spidergon com taxas de transmissão de até 70%. Já com taxa de 90%, simulando um fluxo de dados intenso, a rede Spidergon sofre queda de desempenho, ficando atrás da rede Grelha. A rede Toróide mantém os melhores resultados.

A Figura 5.5 apresenta a latência média nas simulações de 16 a 20, nas quais cada roteador inseria 1.000 pacotes, sendo que cada pacote continha 30 *flits*. Isso totaliza 16.000 pacotes inseridos na rede com um total de 480.000 *flits*.

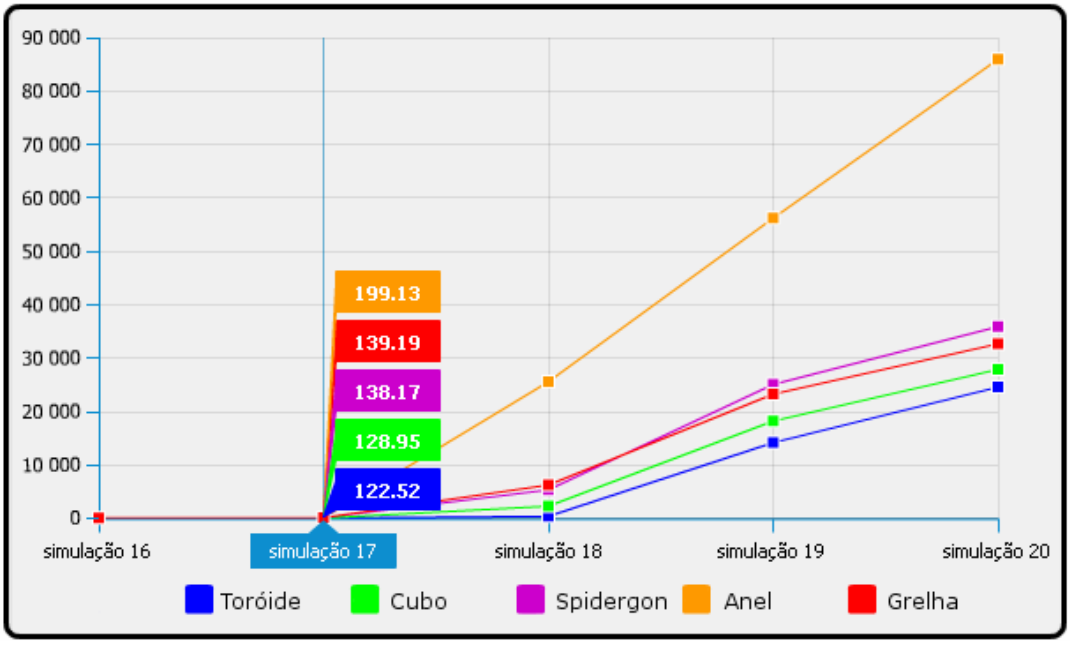


Figura 5.5. Latência nas simulações de 16 a 20

Como pode ser observado na Figura 5.5, com exceção da rede Anel, as outras redes mantiveram-se com desempenho equiparado com taxas de transmissão de 10% e 30%. Já com taxa de 50%, as redes Spidegon e Grelha obtiveram resultados semelhantes, com vantagem para a Spidergon. Ambas ficaram atrás das redes Cubo e Toróide, que obtiveram melhores desempenhos. Já com taxas de 70% e 90%, a rede Grelha superou a Spidergon, entregando os pacotes em espaço de tempo mais curto.

5.1.2 Latência em Relação à Rede Grelha

A presente seção apresenta os resultados comparativos das redes implementadas com a rede Grelha, mostrando a melhora ou piora de desempenho que essas redes tiveram no parâmetro latência. A Figura 5.6 apresenta a comparação do desempenho entre a rede Grelha, a rede que originou as demais e a rede Anel. A figura apresenta a taxa de ganho ou perda da rede Anel em relação à Grelha considerando as quatro diferentes classes de simulações. Valores negativos indicam desempenho pior.

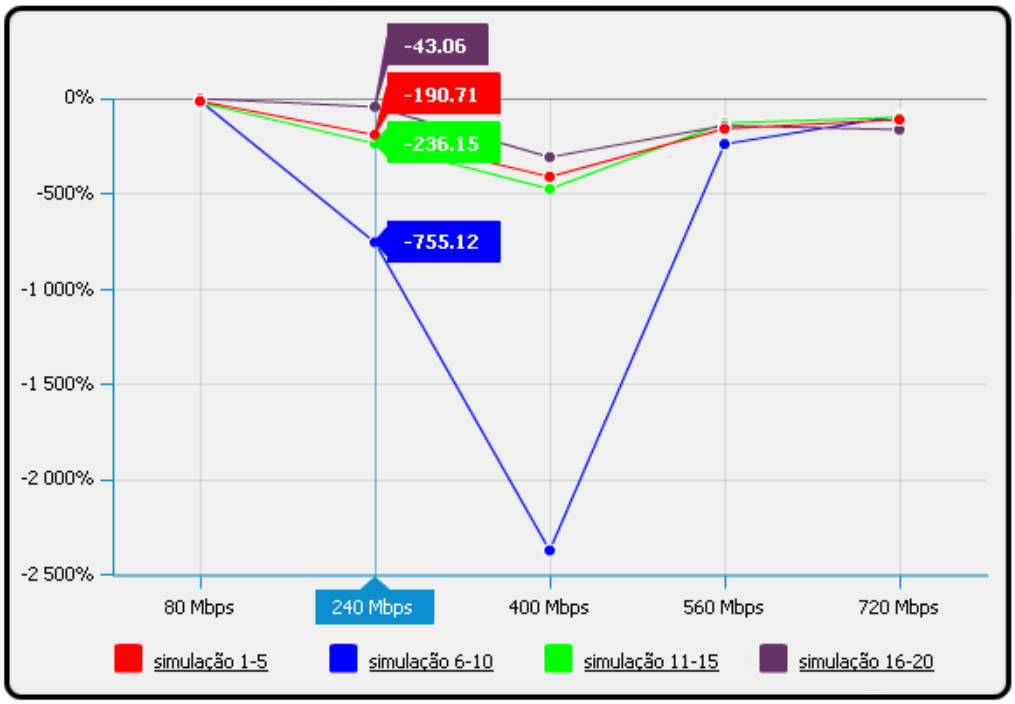


Figura 5.6. Latência da rede Anel em relação à rede Grelha

Em todas as simulações, a rede Anel obteve desempenho pior. A diferença foi grande, passando de 2.000% de perda no pior caso, na simulação 8. É possível identificar o resultado de uma simulação seguindo sua linha no gráfico. Por exemplo, cada um dos cinco pontos presentes na linha simulação 1-5 representam uma simulação diferente. No caso as

simulações 1 (80 Mbps), 2 (240 Mbps), 3 (400 Mbps), 4 (560 Mbps) e 5 (720 Mbps), nessa ordem. A diferença de desempenho não é sentida apenas quando utilizada taxa de transmissão de 10%. Nesse caso, a rede Anel consegue controlar o fluxo de pacotes que transitam na rede sem que exista considerável atraso em relação à rede Grelha. Já a Figura 5.7 apresenta a comparação de desempenho da latência entre a rede Grelha a rede Toróide.

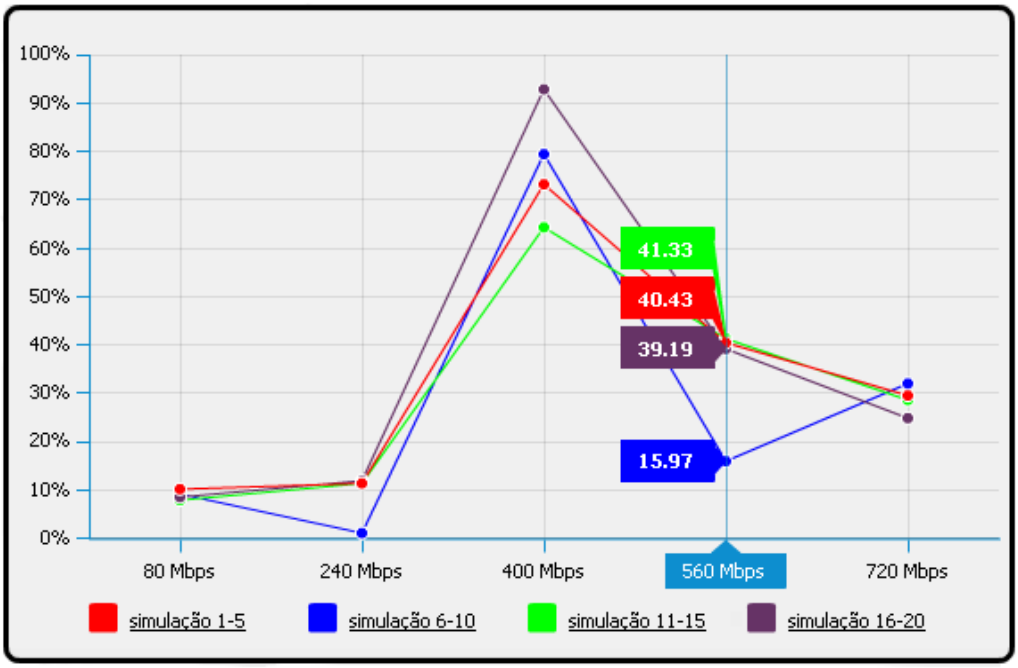


Figura 5.7. Latência da rede Toróide em relação à rede Grelha

A rede Toróide apresentou ganhos de desempenho em todos os testes quanto à latência. Com taxas de transmissão de 10% e 30% a rede Toróide obteve em média 10% de ganho na latência. Os principais ganhos ocorreram com taxa de transmissão de 50%, na qual as diferenças se acentuavam ainda mais com maiores quantidades de pacotes. Isso pode ser observado na Figura 5.7, que apresenta as melhores taxas de ganhos nas simulações 8 e 18, que continham 16.000 pacotes cada e utilizavam taxa de transmissão de 50%. Com taxa de transmissão de 90%, simulando tráfego intenso, os ganhos ficaram entre 25% e 32% com o uso da rede Toróide.

Já a Figura 5.8 apresenta a comparação da latência das redes Grelha e Spidergon. Como pode ser observado, a rede Spidergon apresenta ganho de desempenho quanto à latência na maioria dos casos.

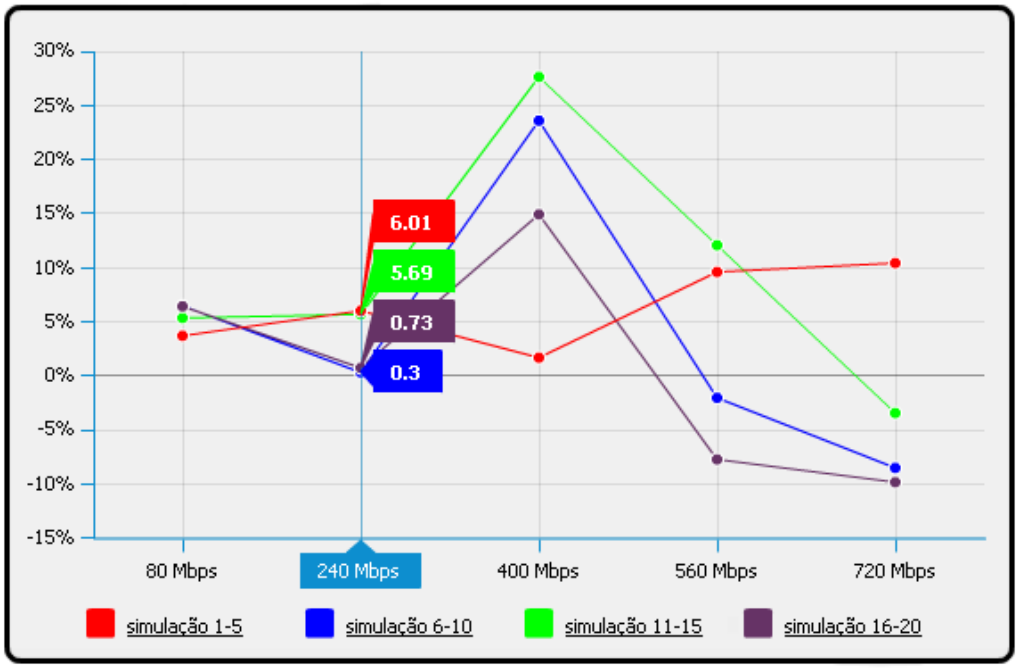


Figura 5.8. Latência da rede Spidergon em relação à rede Grelha

Entretanto, é possível notar que com grandes quantidades de pacotes e taxas de transmissão altas há uma queda de desempenho da rede Spidergon, que apresenta desempenho pior que a rede Grelha nesses casos. A Figura 5.9 apresenta a comparação da latência da rede Cubo com a rede Grelha.

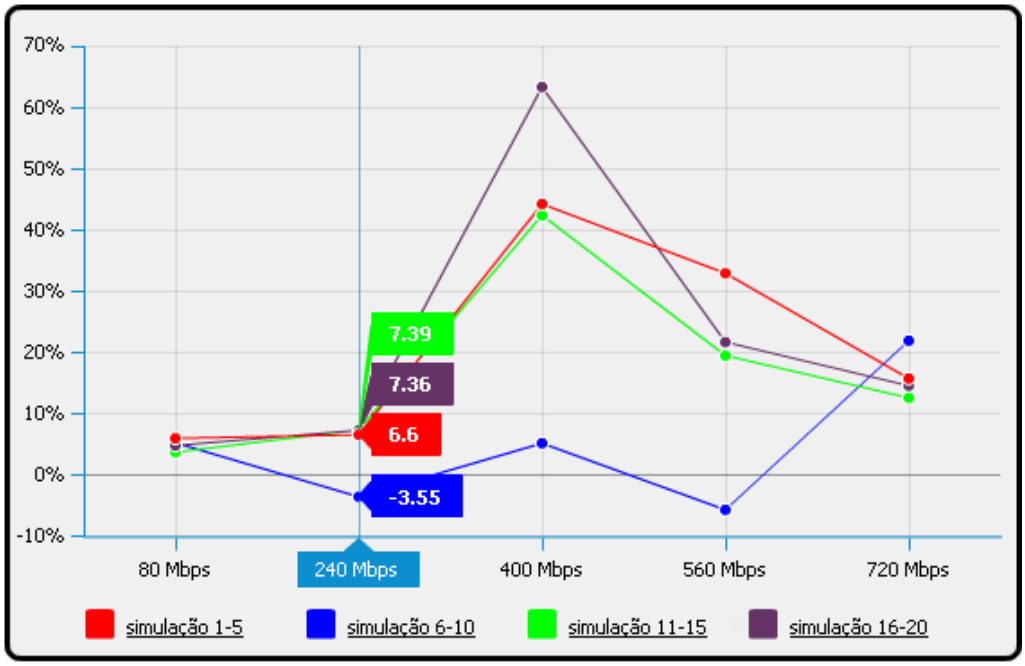


Figura 5.9. Latência da rede Cubo em relação à rede Grelha

A rede Cubo também apresenta ganho de desempenho em relação à Grelha na maioria dos casos. Com taxas de transmissão inferiores à 50%, os ganhos ficam pouco abaixo dos

10%. A única exceção foram as simulações 7 e 9, nas quais a rede Cubo obteve desempenho pior que a rede Grelha.

5.1.3 Vazão

A Figura 5.10 apresenta a comparação de desempenho das cinco topologias analisadas considerando a taxa de vazão das simulações de 1 a 5, que inserem 100 pacotes na rede, cada um com 15 *flits*. Nas simulações a vazão é indicada pela Equação 5.1.

(100 x TamanhoPacote x CiclosPorFlit)/(TempoAtual-TempoAnterior) ... (5.1)

TamaPacote indica a quantidade de *flits* de um pacote. *CiclosPorFlit* representa a quantidade de ciclos de *clock* gastos para transmissão de um *flit*. *TempoAtual* indica o tempo de chegada do último *flit* do pacote atual e *TempoAnterior* o tempo de chegada do último *flit* do pacote anterior.

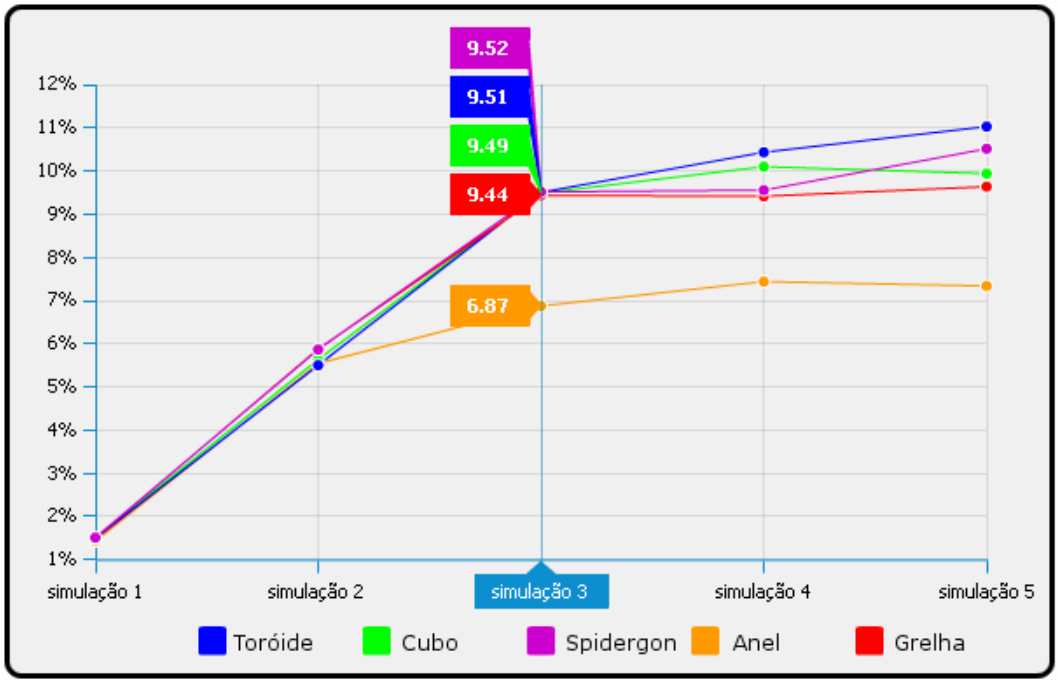


Figura 5.10. Vazão nas simulações de 1 a 5

Como pode ser observado na Figura 5.10, com taxas de transmissão de 10% e 30%, os resultados permanecem semelhantes, com pequena vantagem de desempenho para a rede Spidergon. Com taxa de 50% a rede Anel obtém uma queda em relação às demais, as quais mantêm desempenho muito próximos. Já com taxas mais altas, a rede Spidergon sofre uma queda na performance. Com taxa de 70% ela mantém valores próximos aos da rede Grelha,

enquanto que as redes Cubo e Toróide possuem melhores desempenhos. Já com taxa de 90%, a rede Cubo perde desempenho em relação a rede Spidergon, que é cerca de 0,5% mais eficiente nessa configuração.

A Figura 5.11 apresenta a comparação de desempenho das redes implementadas considerando as simulações de 6 a 10, nas quais cada roteador introduz 1.000 pacotes com 15 *flits* cada.

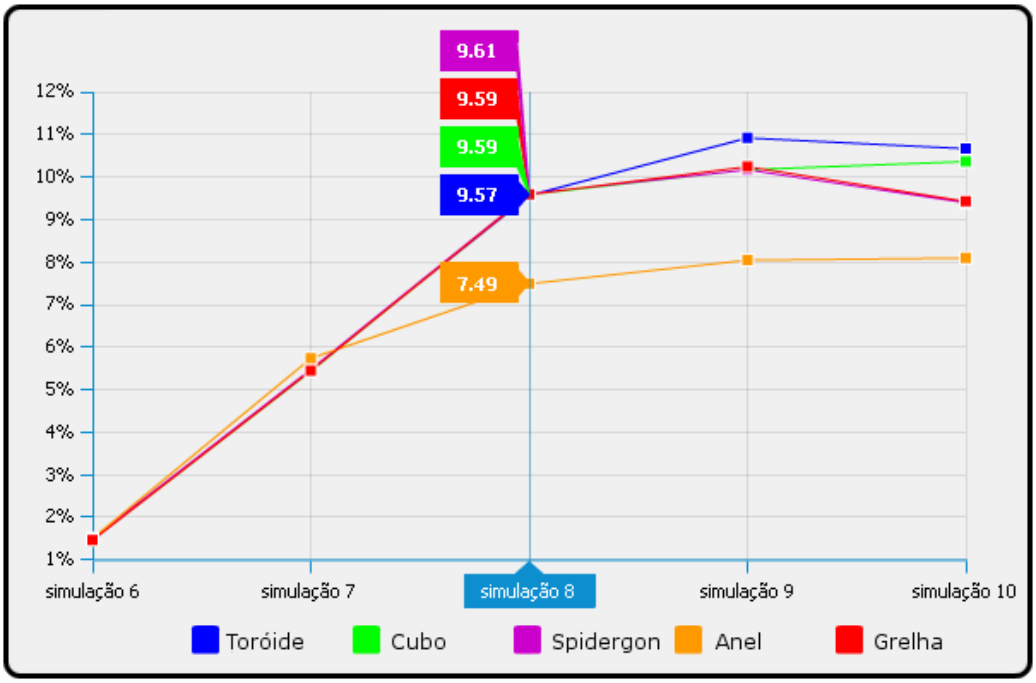


Figura 5.11. Vazão nas simulações de 6 a 10

Com pacotes maiores a taxa de vazão das redes aumenta em relação às simulações anteriores. Mas, mesmo assim, com taxas de 10%, 30% e 50% os resultados mantêm-se semelhantes, com leve vantagem para a rede Spidergon. Já com taxa de 70%, a rede Toróide obtém melhor desempenho, com cerca de 0,7% a mais de vazão em relação às redes Grelha, Cubo e Spidergon, que possuem resultados próximos (10,24%, 10,17% e 10,17% respectivamente). Para taxa de transmissão de 90% os melhores resultados foram das redes Toróide e Cubo, com 10,67% e 10,37% de taxa de vazão. Já as redes Spidergon e Grelha perderam um pouco de desempenho com essa alta taxa de transmissão com resultados cerca de 1,3% abaixo da rede Toróide, mas muito próximos entre si. A Figura 5.12 apresenta a comparação da vazão considerando uma inserção total de 1.600 e 48.000 *flits* na rede, sendo 100 pacotes por roteador e 30 *flits* por pacote.

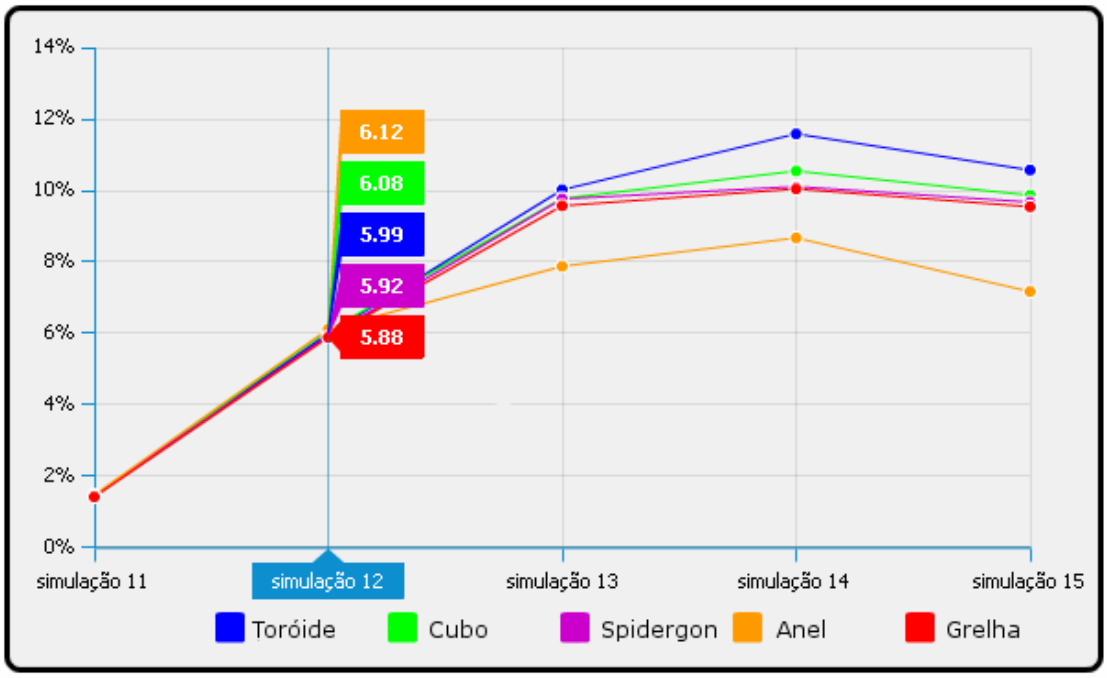


Figura 5.12. Vazão nas simulações de 11 a 15

Com uma maior quantidade de *flits* inseridos, as diferenças de desempenho tendem a se evidenciar mais. Entretanto, com baixas taxas de transmissão, a rede Anel mostrou-se eficiente, apresentando maior vazão que as demais. Todavia, seu desempenho caiu drasticamente quando foi aumentada a taxa de transmissão, uma vez que as poucas possibilidades de roteamento dessa rede acabam congestionando-a. Com taxas de transmissão iguais ou superiores a 50% a rede Toróide novamente obtém melhor desempenho, atingindo mais de 1% a mais de taxa de vazão que a rede Cubo na simulação 14. As outras redes mantém resultados bem próximos. Apenas com taxa de transmissão de 70% que a rede Cubo se destaca em relação à Grelha e à Spidergon durante essa configuração de testes.

Já a Figura 5.13 apresenta a comparação de desempenho no quesito vazão considerando a inserção de 1.000 pacotes por roteador, cada um contendo 30 *flits*. Considerando taxas de transmissão baixas (10% e 30%), a rede Anel novamente mantém bom desempenho em relação às demais, mantendo taxa de vazão de 5,77% na simulação 17 contra 5,55% da rede Toróide, que manteve a segunda melhor média. Já com taxas superiores os resultados mantêm-se semelhantes aos das outras simulações, com a rede Toróide possuindo melhor desempenho, seguida pela rede Cubo e as redes Spidergon e Grelha mantendo resultados próximos.

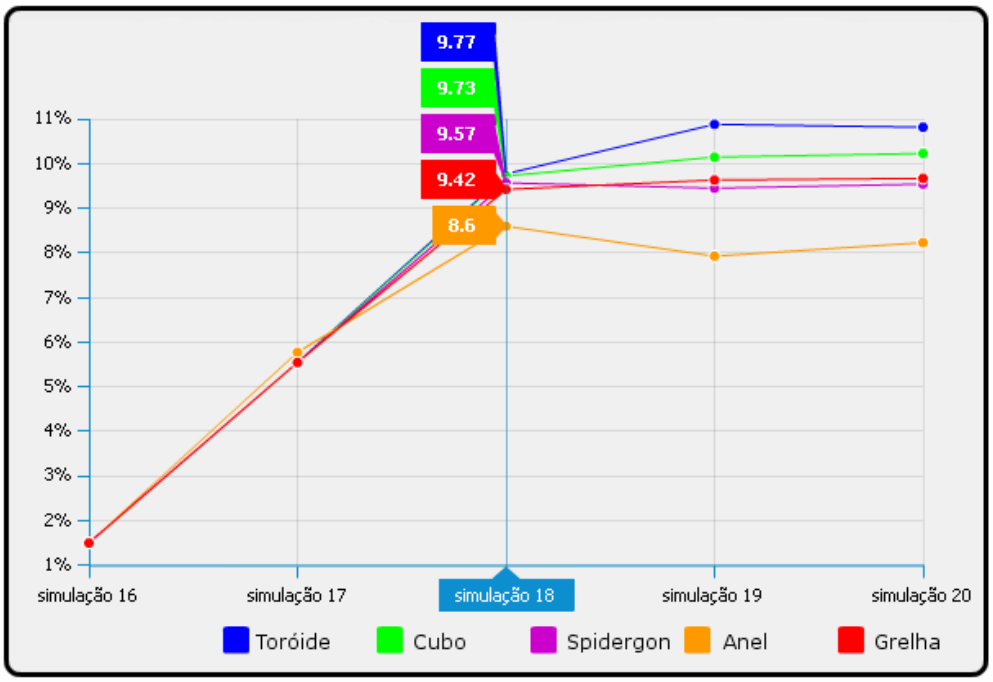


Figura 5.13. Vazão nas simulações de 16 a 20

5.1.4 Vazão em Relação à Rede Grelha

A Figura 5.14 apresenta a diferença de desempenho no quesito vazão da rede Anel em relação à rede Grelha. Valores negativos indicam que o desempenho da rede Grelha foi melhor.

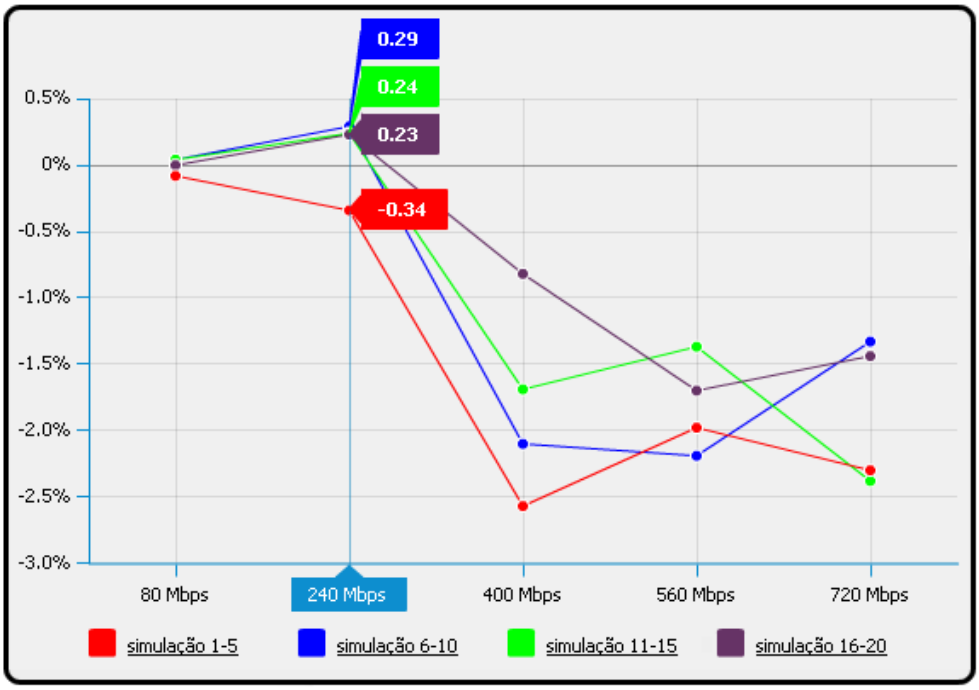


Figura 5.14. Vazão da rede Anel em relação à rede Grelha

Como pode ser observado na Figura 5.14, a rede Anel consegue apresentar alguns resultados melhores que a rede Grelha utilizando taxas de transmissão de 10% e 30% . Com pacotes constantemente sendo inseridos, a rede não consegue manter uma boa taxa de vazão e a entrega de pacotes constantemente sofre atrasos, ficando com desempenho muito abaixo da rede Grelha. A Figura 5.15 apresenta a comparação das redes Grelha e Toróide.

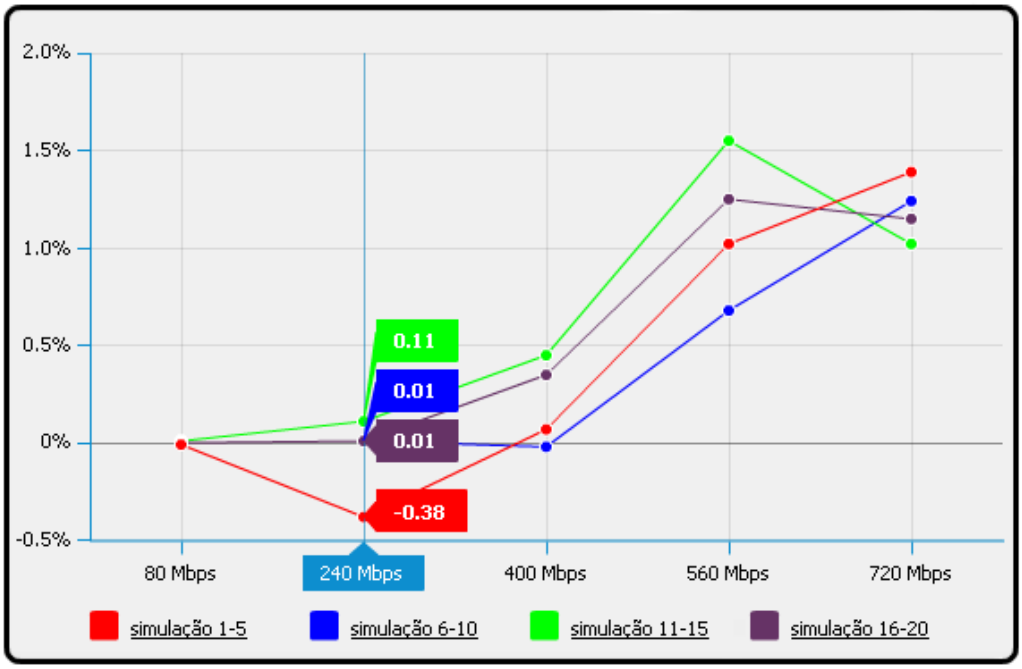


Figura 5.15. Vazão da rede Toróide em relação à rede Grelha

Como pode ser observado na Figura 5.15, a alteração da topologia Grelha para a topologia Toróide apresentou ganho de desempenho quanto à vazão na maioria dos casos. A rede Toróide obteve resultados ruins apenas na simulação 2, considerando 100 pacotes por roteador e pacotes com 15 *flits* e taxa de transmissão de 30%. A rede Toróide mostra-se mais eficiente com taxas de transmissão altas, uma vez que seus caminhos adicionais ajudam a desafogar o tráfego, que é constante com essas taxas. A diferença atinge seu pico na simulação 14, quando chega a pouco mais de 1,5%. Já a Figura 5.16 apresenta a comparação dos resultados da vazão das redes Grelha e Spidergon.

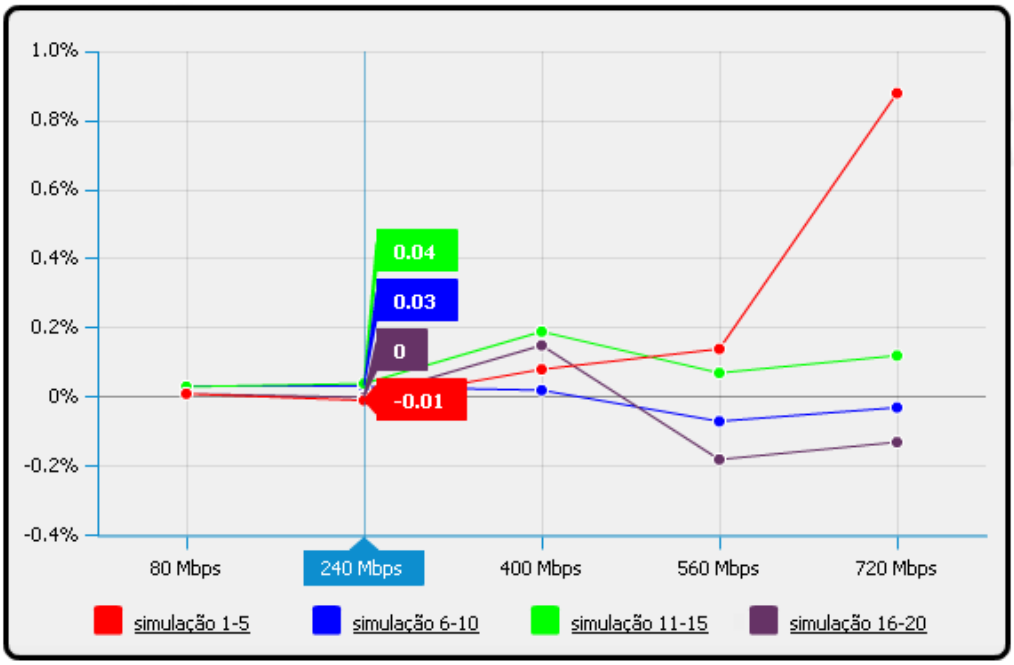


Figura 5.16. Vazão da rede Spidergon em relação à rede Grelha

O desempenho das redes Spidergon e Grelha foram semelhantes. Em nenhuma das simulações houve diferença maior que 1%. Entretanto, fica claro que com um taxa de transmissão média (50%), a rede Spidergon leva vantagem independente da quantidade de pacotes e *flits* inseridos na rede. Já com alta taxa de transmissão e uma grande quantidade de pacotes inseridos (1.000), a rede Spidergon sofre queda de desempenho, ficando abaixo da Grelha. A Figura 5.17 apresenta a diferença da vazão das redes Grelha e Cubo.

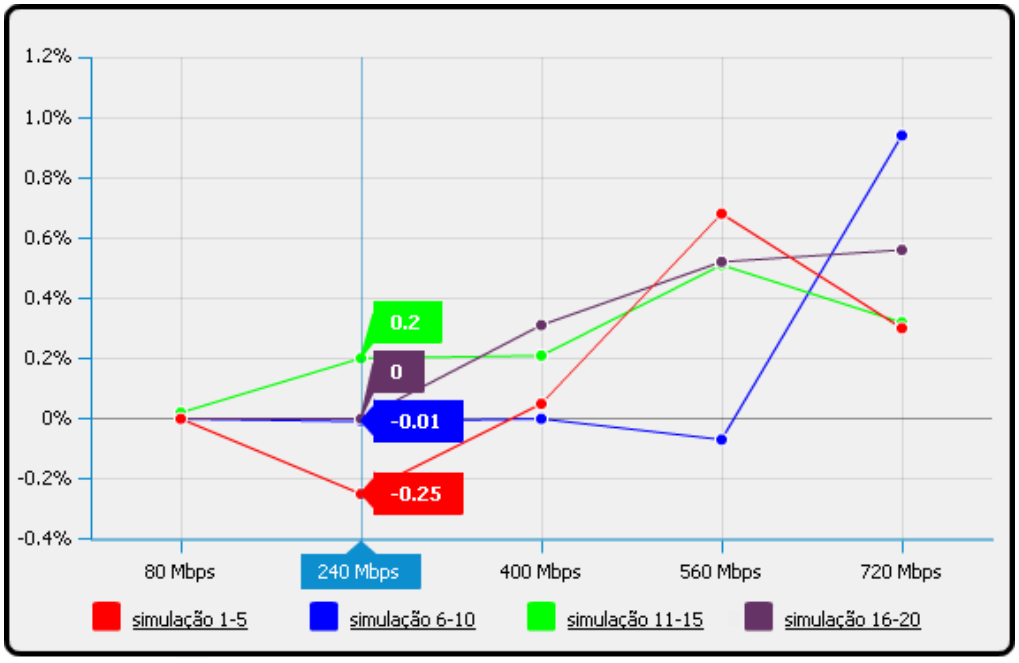


Figura 5.17. Vazão da rede Cubo em relação à rede Grelha

Como pode ser observado na Figura 5.17, em algumas situações, a rede Grelha possui desempenho melhor, como nas simulações 2 e 9. Entretanto, na maioria dos casos a rede Cubo possui melhor desempenho, chegando a quase 1% na maior diferença, que ocorre na simulação 10.

De acordo com os resultados apresentados, é possível notar que a rede Anel só apresenta resultados aceitáveis com taxas de transmissão de 10% e 30%. Já com taxas de transmissão mais altas, há uma grande perda de desempenho dessa rede, que possui roteadores com apenas duas portas para conexões externas. Esse fato sobrecarrega os *buffers* dos roteadores e causa grande atraso na entrega dos pacotes. A topologia não se mostra eficiente para sistemas de propósitos gerais, podendo ser mais eficiente em redes cujo tráfego e taxa de transmissões são bem definidos.

A adição de *links* cruzando a rede e interligando pares de nodos forma a topologia Spidergon, que diminui a quantidade máxima de saltos para realizar uma comunicação e provoca grande melhora no desempenho. A estrutura das interligações da rede faz com que exista aproveitamento dos *links* ociosos, provocando ganho de desempenho significativo em simulações com taxas de transmissão de 10%, 30% e 50%. No entanto, com taxas mais altas, a rede não consegue suportar o tráfego constante e apresenta uma pequena perda de desempenho em relação às redes Grelha, Toróide e Cubo, que apresentam mais *links* capazes de desafogar o tráfego constante.

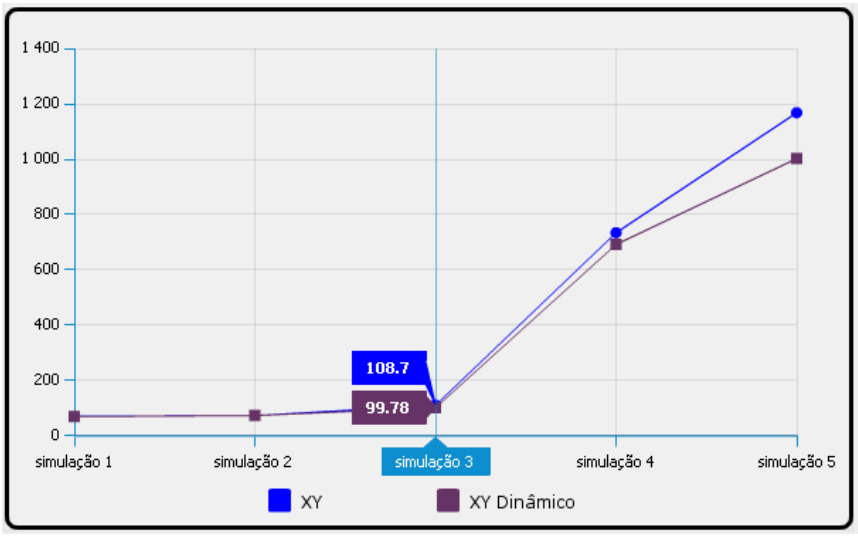
A rede Cubo apresenta bons índices de desempenho. A arquitetura da topologia mostra-se eficiente em relação às redes Anel, Spidergon e Grelha, principalmente quando utilizadas altas taxas de transmissão. Com taxas de 70% e 90%, essa rede consegue tratar melhor o tráfego intenso do que a rede Spidergon, apresentando melhores resultados em tráfego constante.

Já a organização Toróide apresentou os melhores resultados na maioria das configurações. A organização da rede, que reduz bastante a distância entre dois pares de nodos, aliada ao aproveitamento das quatro portas de saída de cada roteador faz com que haja uso eficiente de todos os recursos oferecidos pela rede, melhorando seu desempenho. Um dos principais fatores para essa melhora é que, enquanto na rede Grelha existe uma concentração do tráfego na área central, sobrecarregando os roteadores situados nessa área, na rede Toróide esse tráfego é balanceado por todos os nodos. O principal problema da rede, porém, é que não é possível garantir uma latência na entrega dos pacotes, uma vez que o comprimento dos canais de comunicação não são homogêneos. Isso implica que a comunicação entre dois nodos interligados por *links* nas extremidades da rede demora mais ciclos de *clock* do que a

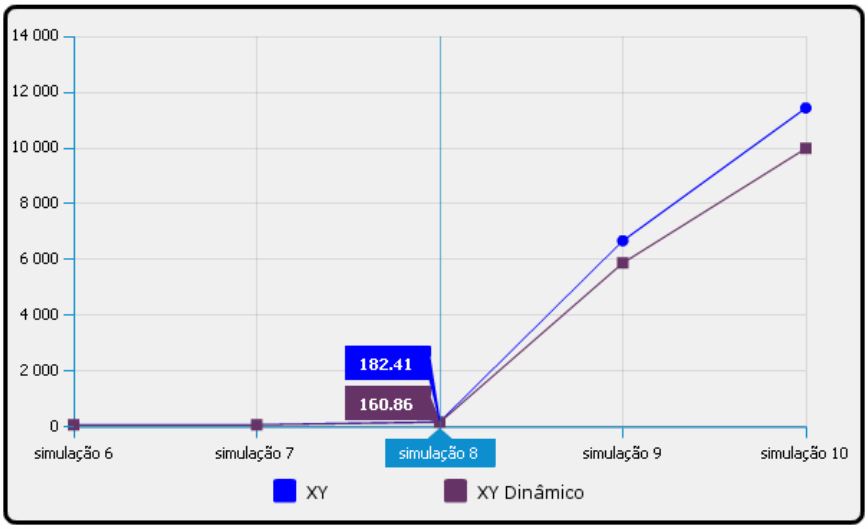
comunicação entre dois nodos vizinhos situados na área interna da rede. Tal problema pode ser crucial para algumas aplicações que necessitem de um tempo de resposta garantido durante a comunicação entre dois núcleos.

5.2 Desempenho do Algoritmo XY Semidinâmico para Topologia Toróide

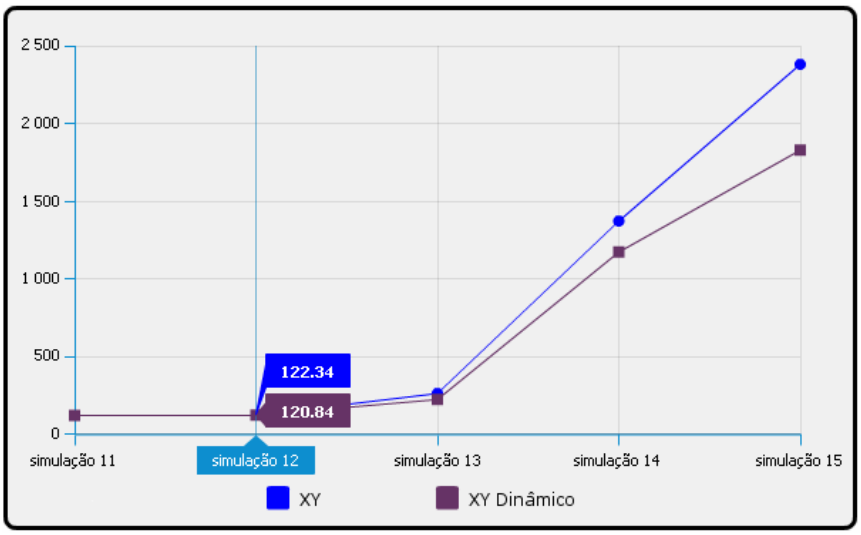
Os resultados das simulações utilizando o algoritmo de roteamento semidinâmico para a rede Toróide são descritos nesta seção. A Figura 5.18 apresenta a comparação de desempenho da rede Toróide utilizando o algoritmo XY e o algoritmo XY semidinâmico considerando o quesito latência.



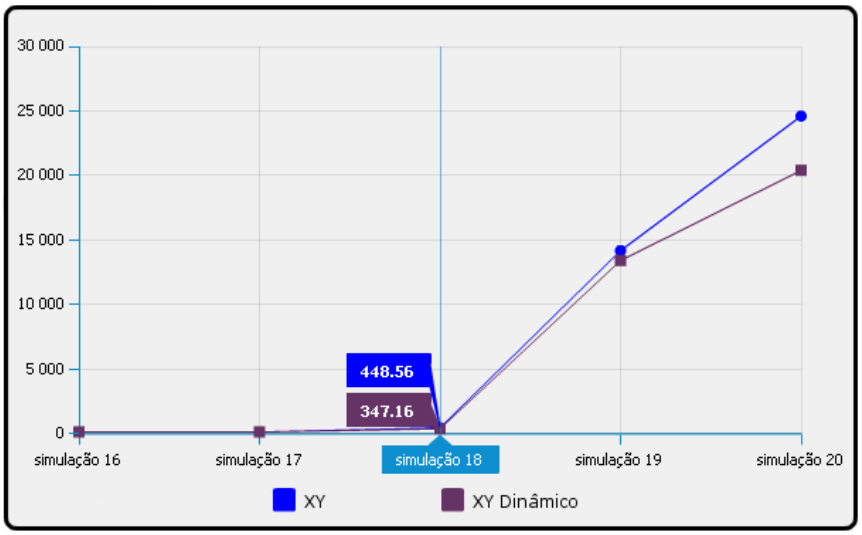
(a)



(b)



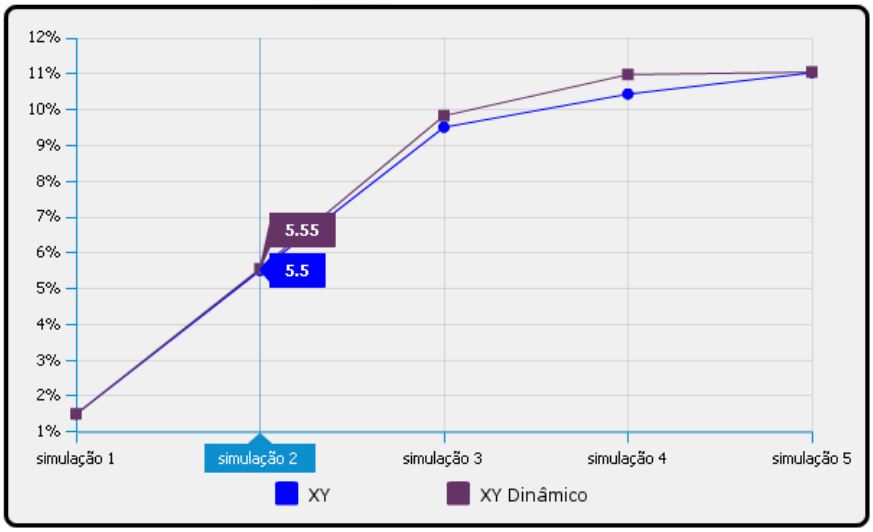
(c)



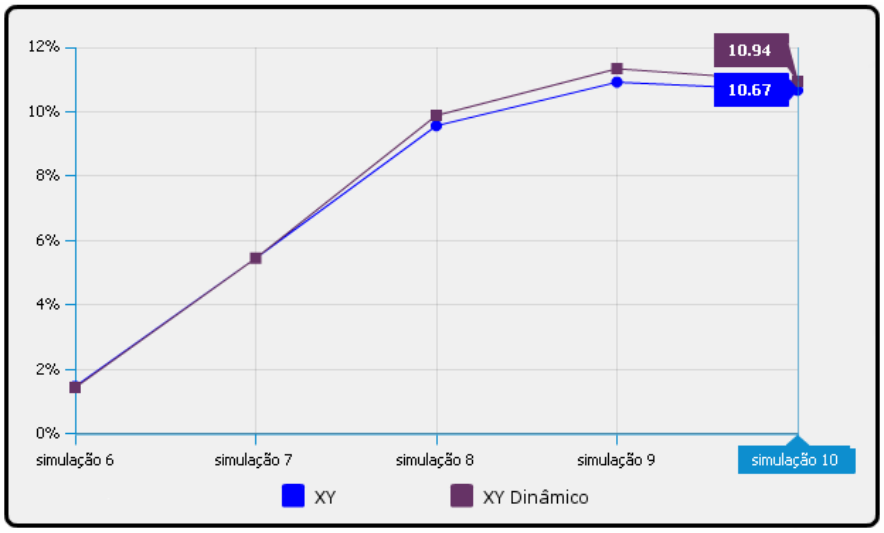
(d)

Figura 5.18. Latência da rede Toróide com os algoritmos XY e XY semidinâmico

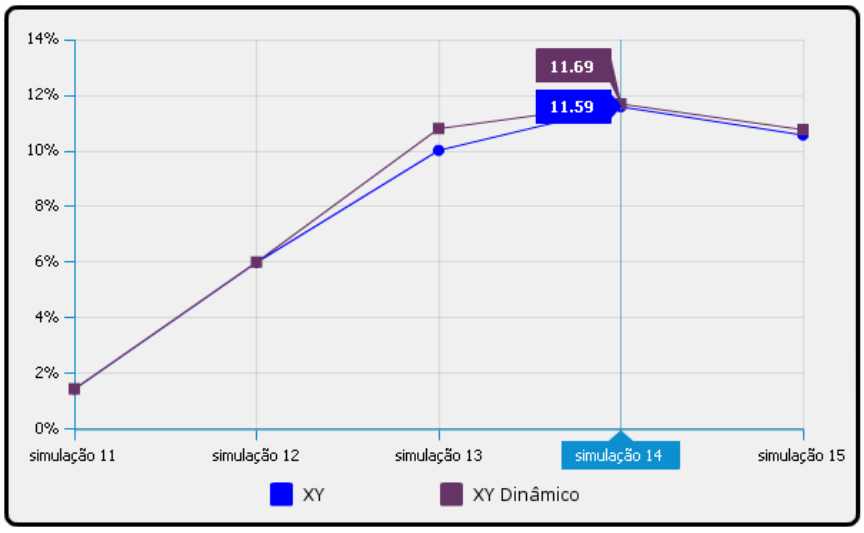
Como mostra a Figura 5.18, a alteração do algoritmo apresenta ganhos em todos os casos, uma vez que no pior caso, o desempenho do algoritmo semidinâmico é igual ao algoritmo XY original. Em alguns casos a diferença é pequena, principalmente com taxas de transmissão menores que 50%. Porém, a diferença de desempenho tende a aumentar em casos críticos, utilizando altas taxas de transmissão e grandes quantidades de pacotes nas simulações. Na simulação 20, por exemplo, que apresenta o caso extremo de tráfego, a utilização do algoritmo semidinâmico reduz em cerca de 4.000 ciclos de *clock* o tempo médio de entrega de pacotes, obtendo um ganho de aproximadamente 20% no desempenho da rede. Já a Figura 5.19 apresenta a comparação da vazão da rede Toróide utilizando o algoritmo XY semidinâmico.


12%
11%
10%
9%
8%
7%
6%
5%
4%
3%
2%
1%
5.55
5.5
simulação 1
simulação 2
simulação 3
simulação 4
simulação 5
XY
XY Dinâmico

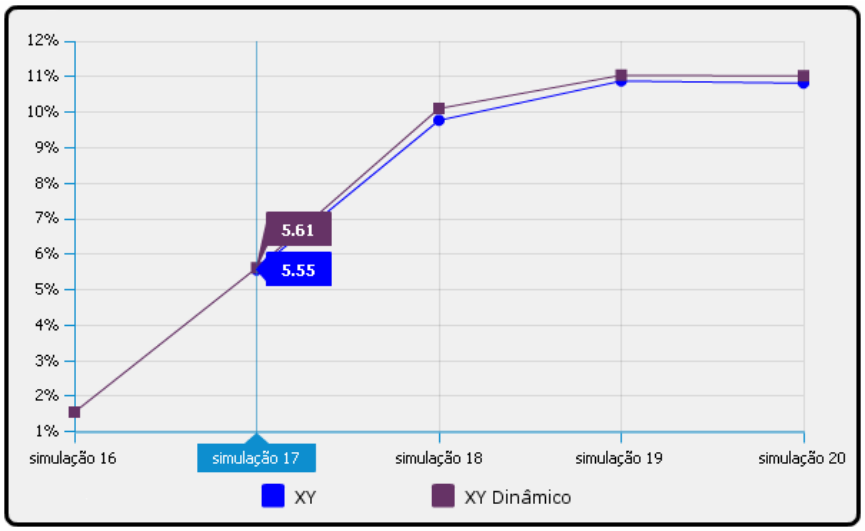
(a)


12%
10%
8%
6%
4%
2%
0%
10.94
10.67
simulação 6
simulação 7
simulação 8
simulação 9
simulação 10
XY
XY Dinâmico

(b)


14%
12%
10%
8%
6%
4%
2%
0%
11.69
11.59
simulação 11
simulação 12
simulação 13
simulação 14
simulação 15
XY
XY Dinâmico

(c)



(d)

Figura 5.19. Vazão da rede Toróide com o algoritmo XY e o algoritmo XY semidinâmico

Como pode ser observado na Figura 5.19, para diferentes tipos de testes, ambos os algoritmos de roteamento obtiveram comportamento semelhante em relação à vazão. O algoritmo semidinâmico obteve desempenho levemente melhor. A maior diferença foi obtida na simulação 13, na qual o algoritmo semidinâmico é 0,8% mais eficiente no quesito vazão.

Os resultados indicam que a alteração do algoritmo para melhor aproveitar as saídas dos roteadores mostra-se eficiente, melhorando o desempenho de uma rede Toróide, que já apresentava melhor desempenho entre as topologias analisadas. O ganho de desempenho é evidenciado em casos de tráfego constante, uma vez que o algoritmo semidinâmico não fica restrito a um único caminho entre determinado par de roteadores. É justamente essa alternativa de rotas proporcionada pelo algoritmo semidinâmico que garante o melhor aproveitamento dos *links* disponíveis e, consequentemente, melhores taxas de latência e vazão.

5.3 Comparação de Área

A análise do custo é um fator importante durante a implementação de NoCs, uma vez que trata-se de sistemas que serão inseridos em um único chip e não podem ocupar muito espaço. Nesse sentido foi realizada a síntese lógica de cada uma das redes implementadas para estimar o espaço utilizado por elas em um chip. A síntese foi realizada com o software Quartus II (Altera, 2008b) tendo como alvo o FPGA EP3SE50F484C2, da família Stratix III da Altera (Altera, 2008c), um dispositivo de alto desempenho encapsulado com tecnologia 65 nm.

A estimativa de custo em silício é realizada de acordo com a quantidade de ALUTs (*Adaptative Look-Up Tables*) e registradores lógicos utilizados durante a síntese do código VHDL. As ALUTs são os blocos básicos do dispositivo alvo, constituídas de funções lógicas dentro delas. Já os registradores constituem o elemento básico para armazenamento de dados. A Tabela 5.2 apresenta os custos de implementação em silício de cada rede analisada.

Tabela 5.2. Custo de implementação das redes analisadas

Rede	ALUTs	Registradores Lógicos
Anel	7.347	3.465
Spidergon	9.970	4.667
Grelha	10.120	5.093
Cubo	12.945	6.243
Toróide	13.324	6.561
Toróide c/ Algoritmo Dinâmico	13.412	6.671

Como pode ser observado na Tabela 5.2, o custo de implementação aumenta de acordo com a quantidade de recursos utilizados por cada roteador da rede. Por exemplo, a rede Spidergon conta com um total de 48 portas para a comunicação entre os 16 roteadores da rede analisada, enquanto a rede Toróide conta com 64 destas portas. Isso implica em um aumento de cerca de 25% a mais de espaço utilizado por um rede Toróide em relação a uma rede Spidergon.

As alterações realizadas tomando como base uma rede Grelha provocaram economia de espaço nas redes Anel e Spidergon, mas também requerem mais em outros casos, como nas redes Cubo e Toróide. Essa estimativa de espaço deve ser considerada durante a escolha de uma rede, uma vez que o espaço em chip nem sempre é um recurso abundante em sistemas em chip. Assim, os requisitos da rede a ser implantada devem ser analisados para equilibrar o desempenho, a área e o custo de implementação da rede.

5.4 Considerações Finais

O presente capítulo apresentou os resultados das simulações realizadas com a utilização de cinco diferentes topologias, além de um algoritmo alternativo que melhora o desempenho de redes com topologia Toróide. As implementações das diferentes topologias tiveram como

base uma rede Grelha e as alterações necessárias foram realizadas em sua estrutura e algoritmo de roteamento para compará-las. Vale ressaltar que a escolha da topologia a ser utilizada em um sistema em chip depende muito dos requisitos do sistema. Para sistemas nos quais o desempenho é exclusivamente importante, a rede Toróide seria a solução ideal dentre as apresentadas, uma vez que não haveria preocupação alguma com custo e espaço.

Porém, se o custo for um fator importante, outras alternativas podem ser consideradas. Caso o comportamento da aplicação seja conhecido e apresenta taxa de transmissão abaixo do 50%, a rede Spidergon pode ser considerada. Se a taxa de transmissão for realmente baixa, a rede Anel também pode ser utilizada, uma vez que apresenta custo muito menor que as outras redes.

Já para taxas de transmissão altas, as redes Grelha e Cubo seriam melhores soluções. A opção por uma ou outra rede também recairia ao custo da rede, já que a rede Grelha apresenta custo menor.

CAPÍTULO

6

Conclusões e Trabalhos Futuros

O presente trabalho apresentou um conjunto de pesquisas relacionadas a uma emergente abordagem para o tratamento da comunicação intrachip. As redes em chip (NoCs) são apontadas como arquiteturas capazes de otimizar a troca de mensagens entre diferentes elementos de um sistema em chip, sem que essa comunicação prejudique o desempenho do sistema. Nesse sentido optou-se pelo estudo e avaliação de diferentes topologias de redes em chip de forma a analisar os principais impactos que diferentes topologias causam sobre o desempenho e o custo de um sistema em chip. Este capítulo aponta as conclusões obtidas com este trabalho e discute oportunidades de pesquisas para a continuidade da pesquisa.

6.1. Visão Geral

Exigências de mercado fazem com que a complexidade de sistemas em chip aumente cada vez mais. Caracterizados por apresentar todas as funcionalidades de um sistema computacional em um único circuito integrado, esses sistemas podem não prover desempenho eficiente quando adicionados muitos elementos a eles, uma vez que as arquiteturas de comunicação normalmente utilizadas sofrem quanto à escalabilidade e desempenho.

Nesse sentido, diversas pesquisas têm sido realizadas para introduzir novos paradigmas de comunicação baseados em redes de interconexão de multiprocessadores. As redes em chip tentam mesclar alta escalabilidade e paralelismo na comunicação com um custo baixo. Tendo como elementos básicos *links* e chaves roteadoras, a forma como os elementos

de uma rede em chip são dispostos, indicado pela topologia da rede, tem forte influência sobre o desempenho, o projeto e a implementação da rede. Nesse contexto o presente trabalho buscou avaliar diferentes topologias de NoCs em relação ao desempenho e custo. Uma rede já desenvolvida e bem estabelecida foi utilizada como base para as implementações. A Hermes é uma infraestrutura parametrizável com topologia Grelha utilizada para a avaliação de redes em chip. A partir de seus componentes foram desenvolvidas redes que utilizam as topologias Anel, Spidergon, Cubo Expresso e Toróide, além de um algoritmo semidinâmico para redes Toróide, capaz de melhorar o fluxo de tráfego de dados e otimizar o desempenho dessa rede.

Para a implementação dessas redes foram realizadas alterações na estrutura da rede Hermes, uma vez que suas interligações tiveram que ser revistas, e em seu módulo de controle, com novos algoritmos de roteamento implementados para cada rede.

6.2 Análise dos Resultados

A análise das redes procurou avaliá-las sob dois importantes parâmetros: desempenho e custo. A avaliação de desempenho é fundamental, pois esse é justamente o principal problema das arquiteturas de comunicação atuais de sistemas em chip, que não conseguem acompanhar o aumento na complexidade desses sistemas sem prejudicar o desempenho. O desempenho foi avaliado em relação à latência e vazão. A latência indica o tempo que um pacote gasta para ser entregue enquanto a vazão trata da quantidade de dados que podem ser transmitidos em um intervalo de tempo. Tais quesitos tratam de dois importantes aspectos do comportamento de uma rede: o tempo de entrega de um pacote, que deve ser sempre o menor possível, e o fluxo do tráfego da rede, que deve ser otimizado para que pacotes não fiquem aguardando para serem enviados.

Já o custo é importante para determinar a área ocupada pela rede em chip. Sistemas em chip muitas vezes possuem restrições quanto a área, pois, em geral, são utilizados como componentes de um sistema maior, cujas limitações de espaço devem ser respeitadas. A maior utilização de recursos de hardware também acaba provocando um aumento no gasto do consumo de energia e, muitos desses sistemas podem ter restrições quanto ao gasto por serem sustentados por baterias.

Dentre as redes analisadas, a rede Anel obteve resultados aceitáveis quando comparada às outras redes apenas quando foram utilizadas taxas de transmissão de 10%. Com taxas maiores essa rede não foi capaz de suportar o tráfego, causando muito atraso na entrega de pacotes. Isso se deve às restrições de adjacência dos nodos da rede, que possui grau 2. No

pior caso um pacote pode levar $\lfloor N/2 \rfloor$ saltos para ser entregue, sendo N o número de nodos da rede. Isso significa que o pacote deve passar por pelo menos metade dos nodos da rede para ser entregue. Essa distância entre os nodos aliada a constantes requisições de comunicação faz com que a rede seja sobrecarregada e, consequentemente, não entregue os pacotes em tempo adequado. Os resultados indicam que a rede Anel pode não ser adequada para aplicações de propósito geral. Porém, caso a taxa de transmissão e o comportamento do tráfego da rede sejam conhecidos e a taxa de transmissão seja baixa, a rede Anel pode ser uma solução, uma vez que apresenta custo bem abaixo das demais. As restrições e especificações do sistema indicariam se o uso de uma rede Anel é viável ou não.

Já a rede Spidergon utiliza uma estrutura anel, com o acréscimo de canais que cruzam a rede e conectam pares de roteadores. Com isso, o número máximo de saltos para a entrega de um pacote cai para $\lceil N/4 \rceil$ nessa rede, que apresenta grau 3. Em relação à Grelha, a Spidergon apresentou melhores resultados quando foram utilizadas taxas de transmissão menores ou igual a 50%. O principal fator responsável por esse resultado é a homogeneidade dos roteadores e dos padrões de interconexão, que fazem com que o maior caminho que um pacote pode percorrer seja uniforme, independente do nodo origem. Na Grelha isso não ocorre, uma vez que o que determina a maior distância que um pacote pode percorrer é a posição do nodo origem na rede. Em uma rede Grelha com $N = m * n$ roteadores, sendo m o número de colunas e n o número de linhas da rede, essa distância pode chegar a $m+n-2$. Entretanto, quando são utilizadas taxas de transmissão superiores a 50%, a rede Spidergon não consegue manter o mesmo desempenho em relação à Grelha. Isso ocorre porque há uma sobrecarga nos *links* que compõem o anel exterior da rede, que não conseguem desafogar o tráfego. O posicionamento dos nodos da rede permite tratar melhor essa constante requisição por comunicação. Outra vantagem da rede Spidergon em relação à Grelha é o custo de implementação que ela apresenta, que é cerca de 3,6% menor.

Uma rede pouco explorada na literatura, mas que apresentou resultados melhores que as redes até então descritas nesta seção é a rede Cubo Expresso, que organiza os nodos em hipercubos interligados. A organização dessa rede mostrou-se eficiente principalmente quando utilizadas altas taxas de transmissão. Em relação à rede Grelha, quando utilizadas altas taxas, percebe-se um congestionamento do tráfego na área interna da rede. Os roteadores posicionados nessa área ficam mais sobrecarregados que os roteadores posicionados nos cantos da rede. Já na rede Cubo Expresso, o tráfego é distribuído de maneira uniforme por toda a rede, o que permite melhor aproveitamento dos canais quando o tráfego é intenso.

Todavia, esse ganho de desempenho requer um custo maior de implementação, já que por ser uma rede de grau 4, ela requer que cada chave roteadora possua quatro portas para a comunicação externa.

A rede com melhor desempenho nos testes realizados foi a rede Toróide. A organização dessa rede reduz bastante a distância entre qualquer par de nodos e, com o uso de *links* adicionais conectando os nodos extremos, resolve o problema de congestionamento na área central da rede Grelha, fazendo com que o tráfego possua o mesmo comportamento por toda a rede. O problema da rede Toróide consiste na utilização de canais com comprimentos diferentes, uma vez que os *links* que conectam nodos extremos devem ser mais compridos que os demais. Isso implica que a comunicação entre dois pares de nodos pode não ter a mesma latência, mesmo se ocorrer sob as mesmas condições. O custo da rede Toróide também foi o maior dentre as redes analisadas, o que indica que restrições de área não podem ser um fator primordial em sistemas que utilizam essa rede.

Com base na rede Toróide foi proposto um algoritmo XY semidinâmico que explora a existência de mais de um caminho mínimo entre um par de nodos da rede. O uso do algoritmo faz com haja uma redução no tempo médio de entrega dos pacotes, uma vez que pacotes que anteriormente deveriam aguardar a liberação de um *link* podem ser roteados por outro canal. O algoritmo aumenta o desempenho, porém também aumenta o custo da rede, sendo voltado para sistemas cujas limitações de área e custo sejam pequenas.

As análises de desempenho e custo aqui apresentadas indicam uma comparação de desempenho e custo sob as mesmas condições. É impossível determinar, entretanto, que determinada rede seja ideal para o uso em sistemas em chip e descartar as outras redes analisadas. A escolha de uma topologia requer uma análise minuciosa das restrições apresentadas pelo sistema e do comportamento das aplicações por ele executadas. Para sistemas que não possuem qualquer restrição de custo e espaço, a rede Toróide com o uso do algoritmo XY semidinâmico seria a rede ideal dentre as analisadas. Porém, se o custo ou o espaço forem um fator determinante no projeto, é importante considerar o uso de outras redes. A taxa de transmissão utilizada apresenta-se como um fator determinante. Para taxas baixas, a melhor relação custo/desempenho é apresentada pela rede Spidergon. A rede Anel pode ser considerada desde que a taxa de transmissão não ultrapasse os 10%. Já se utilizadas taxas altas a rede Grelha ou a Cubo, essa última com maior custo, devem ser consideradas.

6.3 Trabalhos Futuros

Redes em chip constituem uma área de pesquisa relativamente nova e ainda oferece muitas oportunidades de estudos. Na sequência deste trabalho pretende-se realizar melhorias na estrutura das redes implementadas para que seja possível, por parametrização, a geração automática de redes de diferentes tamanhos e configurações com as topologias aqui descritas.

A adaptação dessas redes a diferentes parâmetros para serem analisados também surge como estudo futuro. Diferentes métodos de controle de fluxo e de organizações de *buffers*, por exemplo, podem ser implementados e avaliados quanto ao custo e desempenho.

Algoritmos dinâmicos para a rede Spidergon também podem ser propostos e investigados. Essa rede apresentou ótima relação de custo/desempenho em relação às demais, mas apresenta degradação de desempenho quando simulada em tráfego intenso. Um algoritmo adaptativo que melhor explore os canais *cross* da rede pode ser desenvolvido para verificar se há uma melhora no desempenho com alta taxa de transmissão.

Referências Bibliográficas

ADRIAHANTENAINA, A. et al. SPIN: a Scalable, Packet Switched, on-Chip Micro-Network. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, Munich, pp. 70-73, 2003.

AINSWORTH, T.W.; PINKSTON, T.M. Characterizing the Cell EIB On-Chip Network. In: *IEEE Micro*, v. 27, n. 5, pp. 6-14, 2007.

ALTERA CORPORATION. *Cyclone III Device Handbook*. San Jose: Altera Corporation, v. 1, 2008, 364 p.

ALTERA CORPORATION. *Introduction to Quartus II Software Version 8.0*. San Jose: Altera Corporation, 2008, 246 p.

ALTERA CORPORATION. *Stratix III Device Handbook*. San Jose, 2008, 518 p. (Technical Report SIII5v1-1.5)

BENINI, L.; DE MICHELI, G. Networks on Chip: A New SoC Paradigm. *Computer*, v.35, n.1, pp. 70-78, 2002.

BERTOZZI, D.; BENINI, L. Xpipes: A Network-on-Chip Architecture for Gigascale System-on-Chip. *IEEE Circuits and Systems Magazine*, v. 4, n. 2, pp. 18-31, 2004.

BJERREGAARD, T.; SPARSO, J. A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip. In: *Proceedings of the Design, Automation and Test in Europe*, v. 2, pp. 1226-1231, 2005.

BJERREGAARD, T.; SPARSO, J. Implementation of Guaranteed Services in the MANGO Clockless Network-on-Chip. In: *IEE Proceedings – Computers and Digital Techniques*, v. 153, n. 4, pp. 217-229, 2006.

CHEN, W.T.; SHEU, J.P. Performance Analysis of Multiple Bus Inerconnection Networks with Hierarchical Requesting Model. *IEEE Transactions on Computers*, v. 40, n. 7, pp. 834-842, 1991.

CONCER, N. *Design and Performance Evaluation of Network-on-Chip Communication*

Protocols and Architectures. Tese de PhD, Bologna: University of Bologna, 2009. 202 p.

COPPOLA, M. et al. Spidergon: a Novel on-Chip Communication Network. In: *Proceedings of the International Symposium on System-on-Chip*, Tampere, pp. 15, 2004.

COZZANI, I.; GIORDANO, S.; PAGANO, M.; RUSSO, F. A Performance Analysis of a Credit Based Flow Control Mechanism Loaded by Self-Similar Traffics. In: *Proceedings of the IEEE ATM Workshop*, San Francisco, pp. -, 1996.

CULLER, D.E.; SINGH, J.P. *Parallel Computer Architecture: A Hardware/Software Approach*. 1. ed. São Francisco, EUA: Morgan Kaufmann, 1999, 1025 p.

DALLY, W. J.; TOWLES, B. *Principles and Practices of Interconnection Networks*. São Francisco, EUA: Morgan Kaufmann, 2003. 550 p.

DALLY, W. J.; TOWLES, B. Route Packets, Not Wires: On-Chip Interconnection Networks. In: *Proceedings of Design Automation Conference*, Las Vegas, pp. 684-689, 2002.

DALLY, W.J. Express Cubes: Improving the Performance of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, v. 40, n. 9, pp. 1016-1023, 1991.

DALLY, W.J. Virtual Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, v. 3, n. 2, pp. 194-205, 1992.

DALLY, W.J.; SEITZ, C.L. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, v. 36, n. 5, pp. 547-553, 1987.

DE MICHELI, G.; BENINI, L. *Network On Chips: Technology and Tools*. São Francisco, EUA: Morgan Kaufmann, 2006. 395 p.

DUATO, J.; YALAMANCHILI, S.; NI, L. *Interconnection Networks: An Engineering Approach*. 1. ed. Los Alamitos: IEEE Computer Society Press, 1997, 515 p.

DUTTA, S.; JENSEN, R.; RIECKMANN, A. Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems. *IEEE Design & Test of Computers*, v. 18, n. 5, pp. 21-31, 2001.

FREITAS, H.C. et al. Previsão de Comunicação em Network-on-Chip para Arquiteturas Multi-core. In: *Anais do IV Workshop de Processamento Paralelo e Distribuído*, Porto Alegre, 2006.

GERLA, M.; KLEINROCK, L. Flow Control: A Comparative Survey. *IEEE Transactions on Communication*, v. COM-28, n. 4, pp. 553-574, 1980.

GONÇALVES, N. A.et al. Análise de Performance de Crossbar Switch com a Utilização de HDL. In: *Anais do VIII Workshop em Sistemas Computacionais de Alto Desempenho*, Gramado, 2007, pp. 19-26.

GRAMA, A. et al. *Introduction to Parallel Computing*. Addison-Wesley, 2ª edição, 2003. 656 p.

GROT, B. et al. Express Cube Topologies for on-Chip Interconnects. In: *Proceedings of the 15th IEEE International Symposium on High Performance Computer Architecture (HPCA'09)*, Raleigh, pp. 163-174, 2009.

GUERRIER, P.; GREINER, A.A. Generic Architecture for on-Chip Packet-Switched Interconnections. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, Paris, pp. 250-256, 2000.

IBM. *The CoreConnect Bus Architecture*. White Paper, 1999. 8 p.

JALABERT, A.; MURALI, S.; BENINI, L.; DE MICHELI, G. XpipesCompiler: A Tool for Instantiating Application Specific Networks on Chip. In: *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, v. 2, pp. 884-889, 2004.

KARIM, F.; NGUYEN, A.; DEY, S. An interconnect architecture for networking systems on chips. *IEEE Micro*, v. 22, n. 5, pp. 36-45, 2002.

KERMANI, P.; KLEINROCK, L. Virtual Cut-Through: A New Computer Communication Technique. *Computer Networks*, v. 3, pp. 267-286, 1979.

KILTS, S. *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Hoboken, EUA: Wiley & Sons, 2007. 352 p.

KRUSKAL, C.P.; SNIR, M. The Performance of Multistage Networks for Multiprocessors. *IEEE Transactions on Computer*, v. C-32, n. 12, pp. 1091-1098, 1983.

KUMAR, M.; DIAS, D.M.; JUMP, J.R. Switching Strategies in a Class of Packet Switching Networks. *ACM SIGARCH Computer Architecture News*, v. 11, n. 3, pp.284-300, 1983.

LEE, S.J. et al. Packet-Switched On-Chip Interconnection Network for System-On-Chip Applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, v. 52, n. 6, pp. 308-312, 2005.

MAXIAGUINE, A. et al. Tuning SoC Platforms for Multimedia Processing: Identifying Limits and Tradeoffs. In: *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'04)*, Stockholm, pp. 128-133, 2004.

MENTOR GRAPHICS, *ModelSim SE User's Manual: Software Version 6.4a*. Wilsolville: Mentor Graphics Corporation, 2008, 1036p.

MILLBERG, M. et al. The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip. In: *Proceedings of the 17th International Conference on VLSI Design*, Mumbai, pp. 693-696, 2004.

MNEIMNEH, S. S.; SHARMA, V.; SIU, K. Y. On Scheduling Using Parallel Input-Output Queued Crossbar Switches with no Speedup. In: *Proceedings of IEEE Workshop on High Performance Switching and Routing*, Dallas, pp. 317-323, 2001.

MOADELI, M. et al. Communication Modelling of the Spidergon NoC with Virtual Channels. In: *Proceedings of the International Conference on Parallel Processing*, Xian, pp.

76-84, 2007.

MOADELI, M.; MAJI, P.; VANDERBAUWHEDE, W. Quarc: A High-Efficiency Network on-Chip Architecture. In: *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA'2009)*, Bradford, pp.98-105, 2009.

MORAES, F. et al. HERMES: An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip. In: *Integration the VLSI Journal*, v. 38, n. 1, pp. 69-93, 2004.

NILSON, E. *Design and Implementation of a hot-potato Switch in a Network on Chip*. Tese de Mestrado, Sweden: Royal Institute of Technology - IMIT/LECS, 2002. 66 p.

NISHIMURA, S. et al. High-Speed Network Switch RHiNET-2/SW and Its Implementation with Optical Interconnections. *Technical Digest of Hot Interconnects*, v. 8, pp. 31-38, 2000.

NISWAR, M.; THAMRIN, A.H. Rate-based congestion control mechanism for multicast communication. In: *Proceedings of the 4th WSEAS International Conference on Telecommunications and Informatics*, Prague, article n. 14, 2005.

OST, L. et al. MAIA - A Framework for Networks on Chip Generation and Verification. In: *Proceedings of the Asia and South Pacific Conference on Design Automation (ASP-DAC'05)*, Shanghai, pp. 49-52, 2005.

OULD-KHAOUA, M.; MIN, G. Circuit Switching: An Analysis for k-Ary n-Cubes with Virtual Channels. *IEEE Proceedings – Computers and Digital Techniques*, v. 148, n. 6, pp. 215-219, 2001.

PEH, L.S.; DALLY, W.J. Flit-Reservation Flow Control. In: *Proceedings of 6th International Symposium on High-Performance Computer Architecture (HPCA'00)*, Toulouse, pp. 73-84, 2000.

PHAM, D.C. et al. Overview of the Architecture, Circuit Design, and Physical Implementation of a First-Generation Cell Processor. In: *IEEE Journal of Solid-State Circuits*, v. 41, n. 1, pp. 179-196, 2006.

PINKSTON, T.M.; AINSWORTH, T.W. On Characterizing Performance of the Cell Broadband Engine Element Interconnect Bus. In: *Proceedings of the First International Symposium on Networks-on-Chip (NOCS'07)*, Princeton, pp. 18-29, 2007.

SILLA, F.; DUATO, J. On the Use of Virtual Channels in Networks of Workstations with Irregular Topology. *IEEE Transactions on Parallel and Distributed Systems*, v. 11, n. 8, pp. 813-828, 2000.

SUM, Y.R.; KUMAR, S.; JANTSCH, A. Simulation and Evaluation for a Network on Chip. In: *Proceedings of 20th NORCHIP Conference*, Copenhagen, pp. 7-12, 2002.

TAMIR, Y.; FRAZIER, G.L. Dynamically-allocated multi-queue buffers for VLSI communications switches. *IEEE Transactions on Computers*, v. 41, n. 6, pp. 725-737, 1992.

TITRI, S. et al. Open Cores Based System on Chip Platform for Telecommunication

Applications: VoIP. In: *Proceedings of International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'07)*, Rabat, pp. 245-248, 2007.

TRIPATHI, A.R.; LIPOVSKI, G.J. Packet Switching in Banyan Networks. In: *Proceedings of 6th International Symposium in Computer Architecture (ISCA'79)*, pp. 160-167, 1979.

ZEFERINO, C.A. *Redes-em-Chip: Arquiteturas e Modelos para Avaliação de Área e Desempenho*. Tese de Doutorado, Porto Alegre: UFRG, 2003. 242 p.

ZEFERINO, C.A.; SANTO, F.G.M.E; SUSIN, A.A. ParIS: a Parameterizable Interconnect Switch for Networks-on-Chip. In: *Proceedings of the 17th Symposium on Integrated Circuits and Systems Design*, Pernambuco, pp. 204-209, 2004.

ZEFERINO, C.A.; SUSIN, A.A. SoCIN: A Parametric and Scalable Network-on-Chip. In: *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design*, São Paulo, pp. 169-174, 2003.