UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RODRIGO DA SILVA CARDOZO

Redes-em-Chip de Baixo Custo

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Altamiro Amadeu Susin
Orientador

Porto Alegre, setembro de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Cardozo, Rodrigo da Silva

Redes-em-Chip de Baixo Custo / Rodrigo da Silva Cardozo – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

75 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Altamiro Amadeu Susin.

1.Microeletrônica. 2.Sistemas Embarcados 3.Redes em chip. I. Susin, Altamiro Amadeu. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. José Carlos Ferraz Hennemann
Vice-reitor: Prof. Pedro Cezar Dutra Fonseca
Pró-Reitora de Pós-Graduação: Profa. Valquiria Linck Bassani
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Flávio Rech Wagner
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

É inacreditável alcançar este momento tão importante na minha vida. Momento este que se iniciou com os meus estudos de Engenharia Elétrica na UFRGS. A conquista deste mestrado representa para mim uma grande vitória. Vitória essa que seria impossível sem a ajuda e o apoio de grandes pessoas que me auxiliaram, me auxiliam, e tenho a certeza de que sempre me auxiliarão. A estas pessoas é que eu não posso deixar de agradecer neste momento. Inicio esta seqüência de merecidos agradecimentos pela minha família. Agradeço a minha esposa Cristiane pelo amor e paciência mantida durante este longo período. Agradeço a minha mãe, ao meu pai e minha irmã pelo eterno amor e apoio ao estudo.

Agradeço ao meu orientador, Altamiro Amadeu Susin que sempre confiou no meu trabalho e na minha vontade de crescer. Agradeço ao professor Sergio Bampi por ter sido meu orientador durante a graduação, no começo da minha vida acadêmica e ao professor Luigi Carro pelo apoio dado durante o mestrado. Agradeço também aos professores membros desta banca por terem aceitado fazerem parte desta e assim fazem-me sentir honrado por tais presenças. Agradeço a todos os funcionários do Instituto de Informática e da Biblioteca pelo esmero e disposição que sempre colocaram em seus serviços e aos amigos do GME pelo apoio nos estudos e pelos churrascos feitos.

Aos amigos Fernando Paixão Cortes, Marcio Kreutz, Emerson Machado, Isabel Forquin também fica o meu agradecimento pelo apoio e companheirismo, assim como para Marcos Tabajara e o pessoal da Manutenção Eletrônica da CGTEE. Por fim, agradeço a todas as pessoas que me ajudaram, direta ou indiretamente, durante a realização deste mestrado.

A todos vocês o meu muito obrigado!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ASIC	*Application Specific Integrated Circuits*
ASIP	*Application Specific Instruction Set Processor*
aSoC	*adaptive System-on-Chip*
BFW	*Buffered Wormhole Switching*
bop	*begin-of-packet*
bps	bits por segundo
CBDA	*Centrally-Buffered, Dynamically-Allocated*
CLICHÉ	*Chip-Level Integration of Communicating Heterogeneous Elements*
CMOS	*Complementary Metal-Oxide Semiconductor*
CPE	Codificador de Prioridade Estática
CPLD	*Complex Programmable Logic Device*
CPU	*Central Processing Unit*
DAMQ	*Dynamically-Allocated, Multi-Queue*
DSP	*Digital Signal Processor*
E/S	Entrada-e-Saída
eop	*end-of-packet*
FCFS	*First-Come-First-Served*
FFT	*Fast Fourier Transform*
FIFO	*First-In, First-Out*
FPGA	*Field Programmable Gate Array*

GALS	*Globally Asynchronous and Locally Synchronous*
HOL	*Head-of-Line*
I/O	*Input/Output*
LRS	*Least Recently Served*
MEMS	*Micro-Electro-Mechanical Systems*
NoC	*Network-on-Chip*
PA	Processador Auxiliar
ParIS	*Parameterizable Interconnection Switch*
PC	Processador Central
PE	*Processing Elements*
PG	*Priority Generator*
PHIT	*Phsycal unIT*
PPE	*Programmable Priority Encoder*
RASoC	*Router Architecture for System-on-Chip*
RISC	*Reduced Instructions Set Computer*
RR	*Round-Robin*
SAF	*Store and Forward*
SAFC	*Statically Allocated, Fully Connected*
SAMQ	*Statically Allocated, Mulit Queue*
SoC	*System-on-Chip*
SoCIN	*System-on-Chip Interconnection Network*
SoCIN*fp*	*System-on-Chip Interconnection Network fully parameterizable*
SoCIN*het*	*System-on-Chip Interconnection Network heterogeneous*
SPIN	*Scalable Programmable Integrated Network*
VLIW	*Very Long Instruction Word*
VLSI	*Very Large Scale Integration*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Com as recentes tecnologias de fabricação é possível integrar milhões de transistores em um único chip, permitindo a criação dos chamados *System-on-Chip* (SoCs), que integram em um único chip um grande número de componentes (tipicamente blocos reutilizáveis conhecidos por núcleos). Quanto mais complexos forem estes sistemas, melhores técnicas de projeto serão necessárias para também reduzir o tempo e custo do projeto. Uma destas técnicas, chamada de *Network-on-Chip* (NoC), permite melhorar a performance da comunicação entre os núcleos e, ao mesmo tempo, fornecer uma plataforma de comunicação escalável e que pode ser reutilizada para um grande número de sistemas. Uma NoC pode ser definida como uma estrutura de roteadores e canais ponto-a-ponto que interconectam os núcleos de um sistema, provendo o suporte de comunicação entre eles. Os dados são transmitidos pela rede na forma de mensagens, que podem ser divididas em unidades menores chamadas de pacote. Uma das desvantagens desta plataforma de comunicação é o impacto na área do sistema causado pelos roteadores. Dentro deste contexto, este trabalho apresenta uma arquitetura de roteador de baixo custo, com o objetivo de permitir o uso de NoCs em sistemas onde a área do roteador representará um grande impacto no custo do sistema. A arquitetura deste roteador, chamado de Tonga, é baseada em um roteador chamado RASoC, um *soft-core* para SoCs. Nesta dissertação será apresentada também uma rede heterogênea, baseada na rede SoCIN, e composta por dois tipos de roteadores – RASoC e Tonga. Estes roteadores visam diferentes objetivos: Rasoc alcança uma maior performance comparada ao Tonga, mas ocupa área consideravelmente maior. Potencialmente, uma NoC heterogênea otimizada pode ser desenvolvida combinando estes roteadores, procurando o melhor compromisso entre área e latência. Os modelos desenvolvidos permitem a estimativa de área e do desempenho das arquiteturas de comunicação propostas e são apresentados resultados de performance para algumas aplicações.

Palavras-chave: Microeletrônica, Sistemas Integrados, Redes-em-Chip.

Low Cost Network-on-Chip

ABSTRACT

Current technologies allow the integration of millions of transistors in one chip, creating the so-called Systems-on-Chip (SoCs), with a large number of components (tipically reused blocks known as cores) on a single chip. As more complex systems are developed, better design techniques are also required to reduce design time and costs. One of these techniques aims at improving the communication among the embedded cores by providing a communication platform that is scalable, and can be reused for a number of systems. This platform, called Network-on-Chip (NoC), can be defined as a structured set of routers and point-to-point channels interconnecting the processing cores of a SoC, in order to support communication among them. Data is transferred by means of messages, which can be divided into smaller units called packets. One of the drawbacks of the NoCs is the area overhead caused by the routers. In this context, this work presents a router architecture, named Tonga, that can be customized in order to allow the building of NoCs for systems with as few as dozens cores. The architecture of Tonga is based on the one of the router named RASoC (Router Architecture for SoC), a parametric soft-core for embedded SoCs. We also present a heterogeneous NoC called SoCINhet and is based on the SoCIN network, and is composed by two routers architectures – RASoC and Tonga. These routers target different design constraints: RASoC reaches higher performance when compared to Tonga, but takes considerably more area. Potentially, an optimum and heterogeneous NoC can be developed by mixing these two router architectures. The models allow getting an estimation of area and performance of NoCs. Performance's results for target applications are also obtained and presented in this work.

Keywords: Microeletronics, System-on-Chip, Network-on-Chip.

1 INTRODUÇÃO

O avanço dos processos de fabricação de circuitos integrados tornou possível agregar um maior número de transistores em uma mesma pastilha de silício. Esta tendência, que aumenta a cada ano, está permitindo o surgimento de sistemas complexos denominados *System-on-Chip* (SoC).

A arquitetura de hardware de um SoC pode conter um ou mais processadores dos mais diversos tipos (RISC, VLIW, DSP, até ASIPs), memórias, interfaces para periféricos e blocos dedicados. Os componentes, chamados de núcleos, são interligados por uma estrutura de comunicação que pode variar desde canais ponto-a-ponto dedicados, barramento e, mais recentemente, redes de interconexão.

Com o aumento da integração, é necessário rever as metodologias de projeto para que a complexidade destes sistemas não aumente o tempo de desenvolvimento. Para atender à produtividade necessária que o mercado exige, é preciso que haja um aumento do reuso dos componentes. Assim, a metodologia de projeto de um SoC deve ser baseada em núcleos reutilizáveis, pré-projetados e pré-verificados. Muitos fabricantes oferecem soluções de integração de sistema que incluem bibliotecas completas com núcleos para componentes de processamento e de comunicação.

Para interligar os núcleos são utilizadas diferentes arquiteturas de comunicação. As mais tradicionais são as de canais ponto-a-ponto e barramento. A arquitetura ponto-a-ponto consiste em canais dedicados de comunicação entre os núcleos, o que oferece um melhor desempenho. Já na arquitetura de barramento, os núcleos são conectados a um mesmo canal de comunicação, ou seja, eles compartilham a mesma estrutura para troca de dados, o que causa uma redução no desempenho do sistema. Além disso, com maior número de núcleos conectados aos canais do barramento, a carga capacitiva dos canais é incrementada, resultando em um aumento no tempo e na energia necessários à propagação dos sinais pelos fios do barramento.

Contudo, uma arquitetura do tipo ponto-a-ponto possui reusabilidade limitada, enquanto no barramento a mesma estrutura pode ser reutilizada em diferentes sistemas, reduzindo o tempo de projeto. Devido a isso, e ao baixo custo em silício, o barramento tornou-se a arquitetura de comunicação mais utilizada.

Segundo estudos do ITRS (International *Technology Roadmap for Semiconductors*) (INTERNATIONAL SEMATECH, 2004), até o final desta década, serão feitos projetos com bilhões de transistores com tecnologias de processo de 100 – 50 nm e com freqüências de relógio em torno de 10GHz. Nos futuros sistemas, será possível integrar dezenas, centenas, ou até milhares de núcleos em um mesmo circuito integrado (BENINI, 2002b), permitindo o desenvolvimento de novas aplicações. Entretanto, do ponto de vista da comunicação, o problema destes novos sistemas é a inviabilidade do uso de interconexões dedicadas face às dificuldades envolvidas e à falta de

reusabilidade dessa arquitetura. Já a utilização de uma arquitetura de comunicação por barramentos não atenderá os requisitos de desempenho em comunicação dos futuros SoCs, devido ao aumento da carga capacitiva do canal de comunicação.Além da diminuição da freqüência de operação, o aumento da carga capacitiva dos canais de comunicação do barramento provoca o aumento do consumo de potência dos SoCs, algo não desejado nos sistemas embarcados atuais e futuros.

Torna-se necessário, portanto, o desenvolvimento de novas técnicas para implementar a comunicação dos núcleos, que possam oferecer tanto alta reusabilidade, quanto desempenho satisfatório em comunicação dos futuros SoCs. A solução proposta baseia-se nos conceitos utilizados nas redes de interconexão de computadores paralelos, provendo assim para os sistemas embarcados uma plataforma de comunicação escalável, que ofereça paralelismo, que possa ser reutilizável para vários sistemas e tenha um baixo consumo de potência. Esta plataforma, chamada *Network-on-Chip* (NoC), pode ser definida como uma estrutura de roteadores com canais ponto-a-ponto interconectando os cores do SoC e provendo a comunicação entre eles. Os dados entre os blocos são transferidos por meio de mensagens, que podem ser divididas em unidades menores chamadas pacotes.

Essa arquitetura de comunicação permite que a freqüência de operação do sistema não diminua com o aumento do número de núcleos, ampliando a reusabilidade e o paralelismo. Apesar das vantagens, há um maior custo e aumento da latência na comunicação, que poderão ser atenuados principalmente pelo aumento de transistores disponíveis e por soluções arquiteturais. O ponto em que uma NoC torna-se mais vantajosa que um barramento depende fortemente do número de núcleos do sistema, como foi demonstrado pelos modelos matemáticos apresentados em (ZEFERINO, 2002). Além deste trabalho, em (BECK, 2003) é demonstrado pelo modelo de Sakurai que o consumo de potência de um barramento, para um sistema composto por dez núcleos, é maior do que para as topologias de NoCs estudadas (Spin e Torus), além do barramento apresentar uma menor freqüência de operação.

As arquiteturas de comunicação de uma NoC certamente variam de acordo com as características do sistema, desde o limite de consumo de potência pretendido até o desempenho de comunicação e o custo em área da NoC com relação ao sistema.

Para diminuir o impacto destas redes no custo do sistema é apresentada, nesta dissertação, uma alternativa arquitetural de roteador de baixo custo e uma topologia de rede composta por dois tipos de roteadores. Na tese defendida no Instituto de Informática no ano de 2003 (ZEFERINO, 2003a) foi feita a especificação de uma arquitetura de rede-em-chip e o desenvolvimento de um modelo VHDL parametrizável do seu roteador. Essa rede é denominada SoCIN (*System-on-Chip Interconnection Network*) e sua característica principal é que ela se baseia em um núcleo de roteador configurável, denominado RASoC – *Routing Architecture for System-on-Chip*, cuja largura de canais e profundidade dos *buffers* podem ser dimensionadas em função dos requisitos do sistema.

Baseado no roteador RASoC foi desenvolvida uma arquitetura de roteador, chamado Tonga, com o objetivo de reduzir o impacto do aumento do custo do sistema com o uso de NoCs. Para tanto, foram realizadas algumas alterações no algoritmo de roteamento, na arbitragem e uma multiplexação dos canais internos do roteador. A análise de performance das arquiteturas de comunicação entre os roteadores Tonga, RASoC e barramento foi feita em um ambiente de simulação desenvolvido em C++ (KREUTZ,

2001) e com a descrição do comportamento de comunicação de algumas aplicações de sistema embarcado.

Além do desenvolvimento de uma arquitetura de roteador de baixo custo, foi explorada nesta dissertação uma rede heterogênea, chamada de SoCINhet, composta pelos dois tipos de roteadores: Tonga e RASoC. Com essa rede mista pretende-se obter um melhor resultado entre custo e latência da rede, utilizando roteadores RASoC onde houver uma maior taxa de mensagens e Tonga onde não houver essa necessidade. Um algoritmo de otimização, o método Tabu, procura a melhor combinação de roteadores e a posição mais otimizada dos núcleos.

O Capítulo 2 apresenta uma revisão bibliográfica dos conceitos de SoC e de NoC. É explorado neste capítulo o espaço de projeto das NoCs composto pelas principais topologias de rede e pela classificação das estruturas de comunicação dos roteadores. As principais NoCs encontradas na literatura são mostradas, assim como suas características, com ênfase à rede SoCIN e ao roteador RASoC, que será a base do roteador Tonga e da rede heterôgenea.

O Capítulo 3 apresenta uma descrição do roteador Tonga. São apresentados os métodos usados para otimização de área e os resultados de síntese, incluindo custos e freqüências de operação do roteador e de redes básicas para diferentes configurações dos parâmetros do roteador.

No Capítulo 4 é mostrada a NoC heterogênea, SoCINhet, composta por roteadores RASoC e Tonga e é detalhada a ferramenta de otimização do posicionamento dos roteadores e dos núcleos na rede heterogênea, para uma determinada aplicação. Os resultados obtidos para a rede heterogênea, tanto em área como freqüência, são comparados com as redes compostas por roteadores Tonga e RASoC.

Por fim, no Capítulo 5 são apresentadas as conclusões e considerações finais sobre a reflexão desenvolvida.

2 REVISÃO BIBLIOGRÁFICA

Uma revisão em torno de arquiteturas de comunicação é apresentada a seguir, enfatizando conceitos fundamentais e pontos relevantes para o desenvolvimento do trabalho. Inicialmente, é apresentado o conceito de sistema integrado em um chip e o projeto dele a partir do reuso de blocos pré-projetados. Em seguida são exploradas informações relativas às características, subdivisões e particularidades das Redes-em-Chip. Por fim serão mostradas as principais redes pesquisadas na literatura, entre elas a rede SoCIN, base para o desenvolvimento das reflexões realizadas ao longo desta pesquisa.

2.1 Sistemas Integrados em um chip

O aperfeiçoamento do processo de fabricação CMOS tornou possível produzir chips com uma alta densidade de transistores, possibilitando o projeto de sistemas completos em um único chip, os quais são denominados Sistemas-em-chip ou SoCs (*System-on-Chip*) (BERGAMASCHI, 2000)(BERGAMASCHI, 2001). SoCs podem ser compostos por processadores, memórias, ASICs e tecnologias mais recentes como MEMs, formando sistemas heterogêneos complexos.

Apesar de suas vantagens, a combinação das mais diversas tecnologias em um mesmo sistema aumentará o tempo e o custo dos novos projetos. Enquanto isso, o tempo de lançamento no mercado do produto (*time-to-market*) tende a ser cada vez menor devido à redução do ciclo de vida dos produtos. Em (BERGAMASCHI, 2002), os dados mostram a diminuição do tempo que determinado produto leva para alcançar um volume de vendas de 1 milhão de unidades. Por exemplo, enquanto a TV preto e branco levou em torno de 16 anos para alcançar um milhão de unidades vendidas, o DVD alcançou este mesmo volume de vendas em apenas um ano. Assim, para ser possível atender ao *time-to-market* exigido, a metodologia de projeto de sistemas integrados em um chip é baseada no reuso de blocos previamente projetados e verificados, os quais são denominados núcleos (cores). Estes núcleos são circuitos pré-projetados e pré-verificados que podem ser usados na construção de uma aplicação maior ou mais complexa em um único chip.

Os núcleos podem ser implementados de diferentes maneiras. Eles podem ser descritos em uma linguagem de *hardware* que pode ser mapeada para diversos processos de fabricação (núcleos *soft-core*); em um *netlist* pronto para as etapas de posicionamento e roteamento (núcleos *firm-core*); ou implementados em nível de leiaute, com informações referentes às principais características do circuito (núcleos *hard-core*). Um núcleo, portanto, é resultado de tecnologia, de *software* e de experiência do projetista e, por isso, está sujeito aos direitos autorais. O núcleo é a propriedade intelectual (IP) que o projetista licencia ao usuário.

Os núcleos em um SoC podem ser novos ou herdados de projetos já existentes, bem como ser obtidos de uma ou mais bibliotecas. Se os núcleos são provenientes de fontes independentes, a integração e o teste podem ser difíceis, com a necessidade de reprojeto do núcleo para adequar a uma interface comum.

Em um sistema integrado é necessário ainda ter uma estrutura que permita a comunicação entre os núcleos. Duas destas estruturas são utilizadas com mais freqüência para interconectar os núcleos: fios ponto-a-ponto dedicados e barramentos. Uma arquitetura de fios ponto-a-ponto consiste em estruturas dedicadas que interligam um núcleo ao outro, como mostrado na Figura 2.1. O barramento consiste em um canal compartilhado em que os núcleos do sistema estão ligados, trocando mensagens entre si.

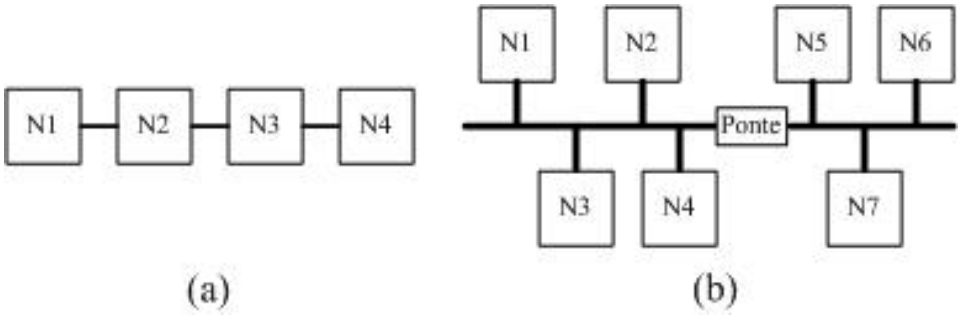

Figura 2.1: Interconexão com: (a) ponto-a-ponto; (b) barramento

Estas arquiteturas de comunicação atendem a grande parte dos sistemas integrados em chips atuais. Porém, a integração de até quatro bilhões de transistores a uma freqüência de 10GHz em um chip está prevista para até 2012 (INTERNATIONAL SEMATECH, 2004). Essa alta integração irá trazer novos desafios para os projetistas de sistemas. Devido aos efeitos físicos da redução dos transistores, serão necessárias maiores habilidade e experiência do projetista para lidar com fenômenos como ruído e a dificuldade de manter um relógio sincronizando todas as partes do sistema, visto que o aumento dos comprimentos dos fios de interconexão e a diminuição dos transistores tornaram mais significativos os atrasos nas interconexões. Além destes fenômenos físicos, o projetista terá que lidar com sistemas heterogêneos cada vez mais complexos, com partes projetadas por diferentes pessoas, com diferentes linguagens e ferramentas (JANTSCH, 2003).

Tabela 2.1: Comparação entre as arquiteturas de comunicação

	Paralelismo	Consumo	Escalabilidade	Reusabilidade
Arquitetura ponto-a-ponto	Possui paralelismo	Baixo consumo em relação ao barramento	Não oferece escalabilidade	Reuso restrito
Barramento	Não possui paralelismo	Maior consumo de energia	Escalabilidade limitada	Reutilizável
NoC	Possui paralelismo	Baixo consumo em relação ao barramento	Ilimitada, basta acrescentar mais roteadores	Reutilizável

Assim, diversos autores (BENINI, 2002a) (DALLY, 2001) (SAASTAMOINEN, 2002) (GUERRIER, 1999) (GUERRIER, 2000) (KUMAR, 2002) (SGROI, 2001) (JANTSCH, 2001) estão propondo o uso de uma rede de interconexões chaveadas dentro do chip, conhecida como *Network-on-Chip* (NoC). As principais vantagens da NoC com relação às outras arquiteturas de comunicação usadas são: (*i*) aumento do reuso, tanto dos núcleos como da plataforma de comunicação; (*ii*) desenvolvimento de um sistema localmente síncrono, mas globalmente assíncrono (*globally asynchronous and locally synchronous* – GALS) (HEMAMI, 1999), eliminando o problema de sincronização do relógio em todo o sistema e permitindo que os núcleos trabalhem com diferentes relógios; e (*iii*) performance constante do relógio com o aumento de núcleos do sistema. Na Tabela 2.1 é feita uma comparação entre as três alternativas de implementação da comunicação de um sistema-em-chip.

Na próxima seção são explorados os conceitos básicos das redes-em-chip, além do espaço de projeto e as principais redes existentes na literatura.

2.2 Conceitos Básicos sobre NoCs

Uma *Network-on-Chip* pode ser definida como um conjunto de estruturas que permite interconectar os núcleos de um sistema. Os núcleos podem ser desde uma simples memória até um sistema completo com processador, memória e dispositivos de E/S, conforme mostrado na Figura 2.2. As duas principais estruturas que compõem uma NoC são os roteadores e os enlaces.

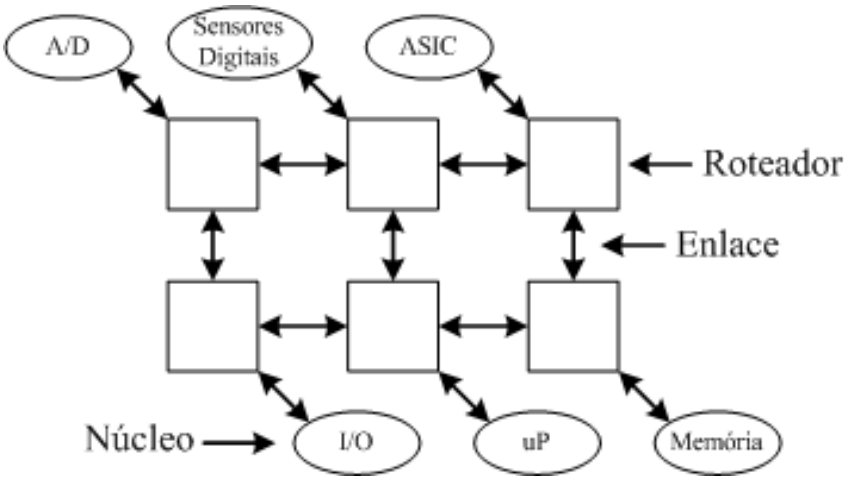

Figura 2.2: Exemplo de uma *Network-on-Chip*

Os enlaces são os canais ponto-a-ponto unidirecionais e assíncronos que conectam um roteador a outro ou a um núcleo do sistema. Estas conexões costumam serem implementadas como dois canais unidirecionais transmitindo mensagens simultaneamente em direções opostas.

O roteador é um conjunto de mecanismos que determinam o caminho da mensagem na rede e é constituído por um *crossbar*, lógica de roteamento e chaveamento e portas de comunicação que interligam os roteadores a outros roteadores e a núcleos do sistema. Cada porta do roteador pode possuir capacidade de memorização das informações que chegam ao roteador e controle de fluxo para regular o tráfego nos enlaces. A topologia da rede é determinada pela forma como os roteadores estão ligados entre si e os núcleos conectados aos roteadores.

Como o roteador é o elemento principal de uma NoC, seu impacto na área final do SoC deve ser minimizado. A exploração do espaço de projeto de uma NoC para uma aplicação determinará a melhor arquitetura de roteador que irá atender à performance exigida pela aplicação com um menor custo em área. Os principais mecanismos de comunicação e as topologias principais de uma NoC são apresentados em uma seção deste capítulo.

2.2.1 A Comunicação em uma NoC

A comunicação entre os dois núcleos, de origem e de destino da informação, é realizada em uma NoC através da troca de mensagens (BENINI, 2002b). Uma mensagem é geralmente composta por: cabeçalho (*header*), carga útil (*payload*) – contendo os dados da mensagem e terminador (*trailer*). O cabeçalho da mensagem indica o início da mensagem e contém as informações utilizadas pelo roteador para encaminhar a mensagem na rede. O terminador aponta o fim da mensagem. Tipicamente as mensagens são quebradas em pacotes que contêm palavras com tamanho igual à largura física do enlace (*phit*), sendo que o pacote possui uma estrutura semelhante à da mensagem.

Em uma rede de comunicação é necessário garantir que os pacotes sempre cheguem a seu destino. Existem três situações que podem impedir que isso ocorra: *deadlock*, *livelock* e *starvation* (DUATO, 1997).

O *deadlock* é definido como uma dependência cíclica entre as solicitações de acesso a recursos de comunicação e armazenamento. A dependência cíclica ocorre quando cada pacote na rede solicita o uso de recursos já ocupado por outro pacote. O *deadlock* pode ser prevenido usando um chaveamento por circuito que reserva todos os recursos da rede necessários para a transmissão do pacote do nodo origem até o destino, gerando uma baixa utilização da rede. No caso de uma rede que aloca os recursos à medida que o pacote avança, uma estratégia é evitar o *deadlock* limitando o número de voltas que o algoritmo de roteamento pode realizar na rede, o que permite uma melhor utilização da rede. A outra estratégia seria a de recuperação, recomendada apenas em casos em que o *deadlock* não é freqüente (DUATO, 1997). Neste caso, são usadas ferramentas de detecção dos *deadlocks* e de realocação dos recursos entre os pacotes. Os pacotes que perdem recursos já alocados podem ser descartados ou terem suas rotas redefinidas (ZEFERINO, 2003a).

Livelock ocorre quando os pacotes ficam circulando na rede sem se aproximarem de seus respectivos destinos. Este problema pode ocorrer em redes com algoritmos de roteamento tolerantes a falhas que usam caminhos não mínimos. A melhor forma de evitar o *livelock* é utilizando roteamento do tipo determinístico, onde a mensagem sempre irá percorrer o mesmo caminho entre um núcleo origem e destino.

Já o processo de *starvation* ocorre quando um pacote armazenado em um *buffer* solicita um canal de saída que permanece bloqueado, porque o canal de saída é sempre alocado para outro de prioridade mais alta. A solução para tornar a rede imune a esse problema é usar um algoritmo de arbitragem que procure reservar sempre uma parte da banda para os pacotes de menor prioridade (ZEFERINO, 2003a).

Outro aspecto que precisa ser levado em conta no processo de comunicação em uma NoC é a avaliação do desempenho de uma rede de interconexão, que pode ser feita pela largura de banda, pela vazão e pela latência da rede.

A largura de banda (*bandwitdh*) refere-se à taxa máxima com a qual a rede de interconexão pode propagar as informações, uma vez que uma mensagem entra na rede. No cálculo da largura de banda costumam ser contados os bits do cabeçalho, da carga útil e do terminador da mensagem, sendo que a unidade de medida utilizada é “bit por segundo” (ou bps).

A vazão (*throughput*) é definida em (DUATO, 1997) como o tráfego máximo aceito pela rede ou, em outras palavras, a quantidade máxima de informação entregue na unidade de tempo. Para que a vazão seja independente do tamanho das mensagens e da rede, seu valor pode ser normalizado, dividindo-o pelo tamanho das mensagens e pelo tamanho da rede.

Já a latência é o tempo envolvido desde o início da transmissão de uma mensagem até o momento em que ela é completamente recebida. A latência é medida em unidades de tempo ou em ciclos de relógio.

2.3 Espaço de Projeto de NoCs

A metodologia no projeto de um sistema em silício baseado em uma NoC consiste em duas fases (KUMAR, 2002). Na primeira fase são definidos o número e os mecanismos de comunicação dos roteadores, além da topologia da rede. Na segunda fase é realizado o mapeamento da aplicação dentro da arquitetura de comunicação definida na primeira fase.

As seleções de topologia e dos mecanismos de comunicação (roteamento, chaveamento, controle de fluxo, arbitragem e memorização) da arquitetura são decisões que cabem ao projetista. Na decisão devem ser levados em consideração fatores como a relação custo/desempenho, escalabilidade e requisitos de performance da aplicação.

As topologias e mecanismos de comunicação usados no projeto de NoCs consistem em um subconjunto do espaço de projeto de uma rede de interconexão chaveada utilizada em computadores paralelos. Em (ZEFERINO, 2003b) são mostradas as principais alternativas aplicadas nas NoCs atuais. Nesta seção são mostradas as principais alternativas de topologia e mecanismos de comunicação utilizadas nas redes-em-chip.

2.3.1 Topologia de NoC

A topologia de rede consiste na estrutura formada pela ligação dos roteadores. Essa organização pode ser expressa na forma de grafo, onde os roteadores são vértices e os canais de comunicação são os arcos (NI, 1993). As topologias utilizadas em NoCs são construídas usando estruturas 2-D e podem ser agrupadas em dois grandes grupos: redes diretas e redes indiretas.

As redes diretas são caracterizadas pela associação de cada roteador a um núcleo, formando um elemento único dentro da rede chamado de nodo. Cada nodo é ligado ponto-a-ponto a outros nodos. Se uma mensagem é enviada, ela passará por vários nodos antes de chegar ao seu destino, utilizando para isso o roteador do nodo.

Uma rede direta pode ter vários níveis de conectividade. Por exemplo, na Figura 2.3 são mostradas as redes diretas mais usadas: a rede grelha e a Torus. Estas redes possuem conexão com, no máximo, quatro nodos vizinhos. Se elas possuíssem um nível ideal de conectividade, cada nodo teria uma conexão para qualquer nodo da rede, o que aumentaria muito o custo da rede e diminuiria a sua escalabilidade.

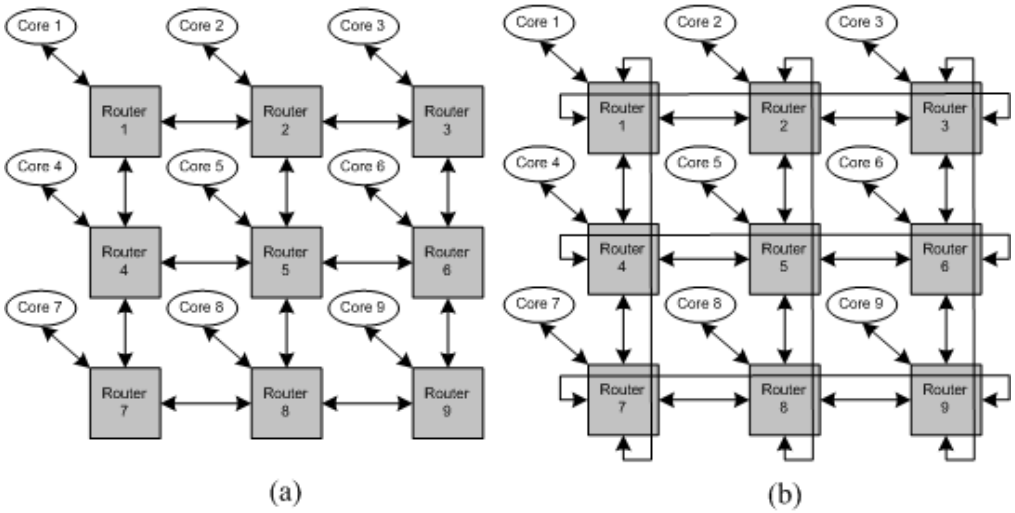

Figura 2.3: Topologia de redes: (a) rede grelha; (b) rede torus

Nas topologias de redes indiretas os roteadores não são necessariamente ligados a um núcleo, sendo que apenas alguns roteadores possuirão conexão com os núcleos do sistema. Entre as redes indiretas destacam-se a matriz de chaveamento (*crossbar* – figura 2.4) e as redes multiestágio. A rede de matriz de chaveamento consiste em uma boa solução em desempenho, mas, para redes muito grandes, sua complexidade e custo cresce ao quadrado em relação ao número de núcleos do sistema, tornando-a muito custosa nestes casos (OST, 2004).

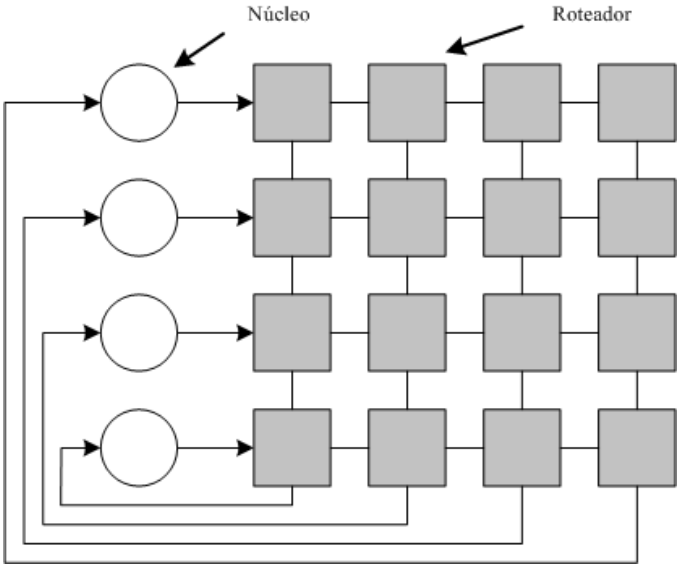

Figura 2.4: Topologia de rede *crossbar* 4x4

2.3.2 Controle de fluxo

O controle de fluxo lida com a alocação de recursos da rede (*buffers* e canais) necessários para uma mensagem avançar, realizando a regulação de tráfego nos canais. Esta política de regulação é implementada nas redes de interconexão em nível de enlace, utilizando *buffers* para armazenamento do dado em transferência. Se o *buffer* de entrada do receptor estiver cheio, o transmissor irá manter os dados no seu *buffer* até que o recurso ocupado no receptor esteja livre.

No caso de redes-em-chip, existem três alternativas mais utilizadas na implementação do controle de fluxo: *handshake*, controle baseado em canais virtuais e controle baseado em créditos (ZEFERINO, 2003b).

O controle de fluxo do tipo *handshake* consiste em duas linhas, uma de validação e outra de reconhecimento (*acknowledge*). Pela linha de validação o nodo emissor avisa que possui mensagem para enviar. O nodo receptor irá confirmar se há espaço em seu buffer através da linha de reconhecimento, estabelecendo a transferência da mensagem.

Utilizado em redes de interconexão com chaveamento *wormhole*, o controle de fluxo baseado em canais virtuais foi concebido com o objetivo de resolver os problemas de *deadlock* nestas redes. Consiste em dividir o *buffer* de entrada em filas independentes de profundidades menores que irão formar os canais-virtuais, procurando desta forma evitar o bloqueio de cabeça de linha. O bloqueio ocorre quando, durante a transmissão de vários *flits* seguidos, o *buffer* de entrada do receptor enche, provocando o bloqueio do canal físico por onde o pacote é transmitido. O uso de canais virtuais permite uma maior utilização do canal físico, que poderá usar um outro canal virtual em caso de bloqueio.

No controle de fluxo baseado em créditos, uma transmissão só ocorre se houver espaço no *buffer* do receptor. O protocolo opera baseado no número de créditos, que corresponde ao espaço no *buffer* que o receptor possui. Assim, o transmissor recebe informação relativa ao número de créditos do receptor e, se este possuir créditos, a mensagem é enviada e o número de créditos diminuído. Se a mensagem é passada adiante, esvaziando uma posição do *buffer*, o número de créditos aumenta novamente.

2.3.3 Roteamento

O algoritmo de roteamento define o caminho pelo qual um pacote será transmitido pela rede, da sua fonte até seu destino, sendo alta a sua influência no desempenho da rede. Além de influenciar no desempenho da rede, ele pode garantir que a rede seja imune a problemas como *deadlock* e *livelock* e capacitá-la para transmitir pacotes, apesar de falhas em seus componentes. Contudo, o algoritmo de roteamento também depende da topologia usada na rede.

Os algoritmos de roteamento são agrupados em diferentes classes, de acordo com características como: localização da tomada de decisão do roteamento; adaptatividade; número de destinos dos pacotes; momento da realização do roteamento; e implementação do algoritmo.

Dependendo de onde são tomadas as decisões de roteamento, ele pode ser centralizado, fonte ou distribuído. No centralizado, as decisões são tomadas por um controlador central. Se o caminho de transmissão da mensagem for decidido pelo roteador de origem, o algoritmo de roteamento é do tipo fonte. Caso a rota seja definida por cada roteador que o pacote atravessar na rede, ele será um algoritmo do tipo distribuído. O roteamento distribuído permite um cabeçalho menor se comparado ao do tipo fonte, pois não é necessário conter no cabeçalho toda a rota do pacote.

O processo de seleção do caminho permite classificar o roteamento em dois tipos: determinístico e adaptativo. Quando o caminho percorrido na troca de mensagens entre dois nodos é sempre o mesmo, o roteamento é chamado de determinístico. Se houver alguma falha de componente ou caminho bloqueado, a mensagem não irá por um caminho alternativo e ficará bloqueada até o recurso ser liberado. Ao contrário, o roteamento adaptativo já prevê vários caminhos possíveis entre o roteador fonte e o

destino. A seleção de um caminho depende de informações como tráfego da rede e estado dos canais, possibilitando assim evitar áreas congestionadas da rede ou com falhas.

O roteamento adaptativo pode ser classificado como parcialmente adaptativo se possibilitar apenas um subconjunto dos caminhos possíveis entre a origem e destino, ou totalmente adaptativo se for possível rotear um pacote por qualquer caminho. Outras classificações dos roteamentos adaptativos dizem respeito à progressividade e à minimalidade. Quanto à progressividade, ele pode ser progressivo se o pacote sempre avança pela rede ou regressivo, se permitir que o pacote retorne pela rede liberando canais previamente reservados. Os roteamentos adaptativos também podem ser classificados quanto à sua minimalidade, em mínimos, ao permitir apenas caminhos que aproximem o pacote cada vez mais do destino, ou não mínimos, se permitirem que o pacote possa ser enviado por um caminho mais longo.

Os algoritmos de roteamento podem ser classificados ainda quanto ao número de destinos: *unicast*, se permitir pacotes com apenas um destino, ou *multicast*, se permitir pacotes com mais de um destino. Quanto ao momento da realização do roteamento, podem ser classificados como dinâmico, no caso de ser realizado no tempo de execução ou estático, se for realizado no tempo de compilação da aplicação. Por fim, quanto à sua implementação, o roteamento será baseado em tabela ao realizar o roteamento após a consulta a uma tabela de memória ou baseado em máquinas de estados se for realizado a partir de algoritmo implementado em *software* ou *hardware*.

2.3.4 Arbitragem

A arbitragem é o mecanismo de comunicação que resolve os conflitos entre duas ou mais mensagens que competem por um mesmo recurso. Ela define qual porta de entrada usará uma determinada porta de saída. Esse mecanismo deve resolver estes conflitos sem que ocorra uma situação de *starvation*, em que uma mensagem fica esperando indefinidamente para avançar na rede.

O árbitro de um roteador pode ser implementado como uma estrutura centralizada ou distribuída. Na centralizada, a arbitragem é feita por um módulo central. Este módulo avalia todas as requisições emitidas pelo circuito de roteamento e executa a arbitragem, tendo uma visão global do uso dos canais de saída.

Já a forma distribuída realiza o roteamento e arbitragem dos canais de modo independente. Cada canal possui seus módulos de roteamento e arbitragem associados às suas portas de entrada e saída, respectivamente. Esta estratégia não permite a visão global dos canais de saída, mas tem como vantagem possibilitar árbitros mais rápidos com uma arquitetura mais simples.

Existem vários mecanismos de arbitragem, baseados em diferentes critérios tais como: prioridades estáticas, dinâmicas, escalonamento por idade, FCFS (*First Come First Served*), LRS (*Least Recently Served*) e RR (*Round-Robin*) (ZEFERINO, 2003a). O esquema de prioridades estáticas considera que cada requisição possui uma prioridade estática, podendo ocorrer que uma requisição de menor prioridade nunca seja selecionada, ocasionando *starvation*. Já o mecanismo de prioridade dinâmica considera as requisições com diferentes prioridades a cada arbitragem. A cada ciclo, portanto, uma requisição mais antiga ganha mais prioridade, o que evitará que ela nunca seja atendida.

Em (SANTOS, 2003) foram implementados em VHDL três tipos de árbitros, baseados em um esquema de prioridade dinâmica, conforme Figura 2.5. Destes três

árbitros, um foi implementado com base em um esquema randômico, enquanto os outros dois usam uma abordagem do tipo *round-robin*.

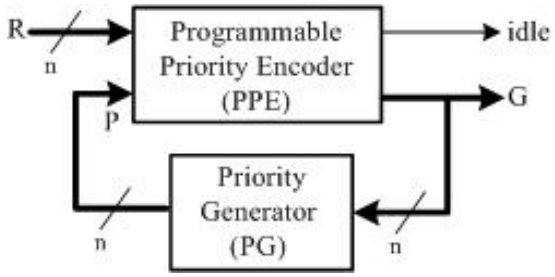

Figura 2.5: Arquitetura básica dos árbitros apresentados em (SANTOS, 2003)

No esquema randômico a arbitragem não depende da requisição atendida anteriormente, pois ele é baseado em um contador em anel que a cada ciclo ativa uma prioridade diferente. Já os baseados em *round-robin* possuem uma função que faz com que a requisição atendida em um ciclo seja a de menor prioridade no ciclo seguinte.

2.3.5 Chaveamento

O chaveamento define como uma mensagem será transferida da entrada de um roteador para um dos seus canais de saída. Os dois métodos de transferência de pacotes utilizados são: (*i*) chaveamento por circuito e (*ii*) chaveamento por pacote.

O chaveamento por circuito (*circuit switching*) estabelece um caminho do núcleo fonte até o núcleo destino para, em seguida, enviar a mensagem pela rede. Neste tipo de chaveamento os canais físicos necessários para a transmissão da mensagem ficam todos reservados, não podendo ser usados até o fim da transmissão. O uso de *buffers* nesta metodologia é mínimo, pois não ocorrerá contenção da mensagem na rede, visto que o caminho já estará todo pré-estabelecido. Os *buffers* serão necessários apenas para conter o cabeçalho responsável por reservar os recursos da rede. A desvantagem deste método é que os canais reservados não podem ser utilizados por outra mensagem, gerando menor utilização da rede.

No chaveamento por pacote (*packet switching*), a mensagem é dividida em pacotes, cada um com um cabeçalho com informações necessárias à sua transmissão pela rede. Os pacotes informam a cada roteador qual caminho seguirão na rede, não existindo um caminho pré-definido e sem estabelecer um circuito fixo do núcleo fonte até o destino. Como não há um caminho pré-definido, é possível uma maior utilização da rede, pois não há reserva de recursos.

O chaveamento por pacote pode ser classificado em: (*i*) chaveamento por pacote *store-and-forward*, (*ii*) chaveamento por pacote *virtual cut-through* e (*iii*) chaveamento por pacote *wormhole*.

Uma rede na qual o roteador recebe completamente um pacote e o armazena em seu *buffer*, antes de identificar qual será o destino e selecionar a porta de saída para transmitir o pacote, possui um chaveamento de pacote *store-and-forward* (armazenar e passar). Este modo exige grande armazenamento nos roteadores, sendo que os *buffers* devem ser dimensionados para o tamanho máximo do pacote. Além disso, há uma sobrecarga na comunicação devido ao tempo que os roteadores precisam para encaminhar cada pacote. A latência de comunicação irá aumentar proporcionalmente com o aumento do pacote, pois este só é repassado após ser completamente armazenado no roteador.

Nas redes com chaveamento por pacote *virtual cut-through*, uma rede pode enviar um pacote no instante em que o canal pelo qual a mensagem será enviada estiver disponível para receber o pacote, reduzindo desta forma a latência da comunicação. O dimensionamento dos *buffers* deve ser feito para conter um pacote inteiro. No pior caso, quando a rede estiver completamente carregada, ela portar-se-á como uma rede de chaveamento por pacote *store-and-forward*, armazenando o pacote inteiro no *buffer* de entrada até a liberação do canal de saída.

O chaveamento por pacote *wormhole* é uma variação do *virtual cut-through* que procura reduzir o tamanho dos *buffers* de armazenamento. Assim, os pacotes são divididos em unidades menores de controle de fluxo, denominadas de *flits*. As unidades de armazenamento são projetadas para armazenar poucos *flits*, de modo que os *flits* restantes do pacote ficarão armazenados nos demais roteadores da rede. Como apenas o *flit* de cabeçalho contém informações de roteamento, os demais *flits* que compõem o pacote seguirão o mesmo caminho do cabeçalho. Neste tipo de chaveamento um canal só é liberado após a passagem de todos os *flits* que compõem o pacote.

2.3.6 Memorização

A memorização é o esquema de filas usado para guardar mensagens destinadas a canais de saídas que já foram requisitados por um outro pacote e que, por isso, estão bloqueadas na rede. Nas NoCs, os dois tipos de memorizações mais usados são a centralizada e distribuída nas entradas.

A memorização centralizada é feita com uma arquitetura de armazenamento que recebe e guarda os pacotes bloqueados de todas as entradas. Esta arquitetura de memorização possui endereçamento dinamicamente distribuído entre os pacotes bloqueados e é chamada de CBDA (*Centrally-Buffered, Dynamically-Allocated*). O projeto deste *buffer* deve ser baseado no número de canais de entradas e saídas do roteador, pois são os canais que determinarão o número de acessos simultâneos de escrita e leitura da memória.

A vantagem da memória centralizada está no fato dela, dinamicamente, alocar os endereços de memória entre os canais de entrada, permitindo que canais que estejam recebendo mais pacotes bloqueados em um determinado instante possam ocupar mais memória. É necessário apenas limitar o espaço de memória que cada canal pode vir a alocar, pois se uma entrada ocupar totalmente o *buffer* irá afetar as outras comunicações (ZEFERINO, 2003a).

Na memorização distribuída cada canal de entrada recebe uma arquitetura independente de memorização dos pacotes bloqueados, sem a possibilidade de compartilhamento dos *buffers* entre os canais. A principal estratégia usada na memorização distribuída é o *buffer* FIFO (*First-In, First-Out*), sendo esta a alternativa de menor custo. Os dados neste buffer são lidos na mesma ordem em que são escritos.

O problema do *buffer* FIFO é que ele pode subutilizar a rede se ocorrer um bloqueio de cabeça de rede (HOL), quando um pacote bloqueado não permite a transmissão de outro pacote cuja saída está livre. Para contornar esse problema os *buffers* SAFC (*Statically Allocated, Fully Connected*) e SAMQ (*Statically Allocated, Multi Queue*) e DAMQ (*Dynamically Allocated, Multi Queue*) podem ser utilizados.

A estratégia consiste basicamente em dividir o *buffer* em partições, cada uma é alocada a uma porta de saída. No *buffer* SAFC estas partições são fixas, o que gera uma menor utilização dele com relação ao *buffer* FIFO. Além disso, o *crossbar* deixa de ser

NxN, onde *N* é o número de portas de saída e entrada, para se tornar um *crossbar* N^2xN, pois cada partição do *buffer* pode ser endereçada para qualquer entrada e o controle de fluxo torna-se mais complexo.

O problema do *crossbar* pode ser resolvido com um *buffer* SAMQ, que multiplexa as partições do *buffer*, reduzindo o custo do *crossbar*, mas sem resolver o problema da menor utilização dos *buffers* e do controle de fluxo mais complexo.

Para solucionar a questão de utilização dos *buffers* pode ser usada uma alocação dinâmica do espaço de memorização. O *buffer* DAMQ utiliza esta técnica, particionando o *buffer* de acordo com a demanda dos pacotes recebidos. As alternativas de memorização resumidas aqui são encontradas em (ZEFERINO, 2003a) e (TAMIR, 1992), que realizam uma comparação entre elas.

2.4 Principais NoCs na literatura

Os primeiros resultados experimentais com redes-em-chip surgiram em 2000 com as redes SPIN (*Scalable Programmable Integrated Network*) (GUERRIER, 2000) e aSoC (*adaptative SoC*) (LIANG, 2000). Deste então, várias redes foram apresentadas em diversos congressos, tais como as redes CLICHÉ (KUMAR, 2002), Octagon (KARIM, 2001) (KARIM, 2002), Proteo (SAASTAMOINEN, 2002) e a rede com topologia *torus* 2D (DALLY, 2001).

No Brasil, foram desenvolvidas duas redes-em-chip: a rede SoCIN (ZEFERINO, 2003b) e a rede Hermes (MORAES, 2003). Recentemente, foi apresentada uma nova versão da SoCIN, a SoCIN*fp* (*System-on-Chip Interconnection Network fully parameterizable*) (ZEFERINO, 2004). A SoCIN*fp* é composta por um roteador (ParIS – *Parameterizable Interconnection Switch*) que possui uma biblioteca de blocos parametrizáveis que permite diferentes implementações dos mecanismos de comunicação.

Nesta seção são apresentadas com mais detalhes algumas destas redes. Foram selecionadas aquelas que possuíam características mais diversas entre si. Por fim, é mostraredo um quadro resumo sobre as diversas características das redes pesquisadas, feitas as devidas considerações sobre elas.

2.4.1 Rede SPIN

A rede SPIN é uma rede indireta com topologia em árvore-gorda quaternária, conforme a Figura 2.6. Os enlaces são bidirecionais com canais de 36 bits, sendo 32 de dados e quatro bits para indicar início e fim do pacote, sinalização de paridade e sinalização de erro de paridade. O formato do pacote da rede SPIN é composto por um *flit* de cabeçalho contendo 10 bits para endereçamento e 22 bits para funções do protocolo de comunicação, uma carga útil variável e um *flit* terminador, indicando o fim da mensagem.

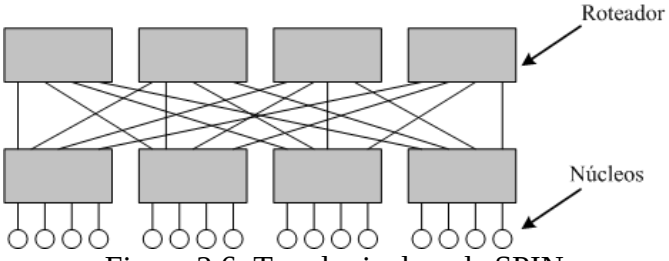

Figura 2.6: Topologia da rede SPIN

O mecanismo de roteamento da rede SPIN é dinâmico (realizado em tempo de execução) e pode ser adaptativo e distribuído ou determinístico e fonte, dependendo do sentido de deslocamento da mensagem. Os pacotes, ao saírem de seu núcleo de origem, sobem a rede, afastando-se dos núcleos. Nesse instante, o mecanismo de roteamento é adaptativo, a mensagem pode ser transmitida por qualquer canal livre, e distribuído, pois são os roteadores que decidem as portas de saída a serem utilizadas. Para os pacotes que descem em direção ao núcleo destino o roteamento é determinístico e fonte, a porta de saída a ser utilizada é identificada no endereço do destinatário e já definida pelo emissor do pacote.

A rede SPIN possui memorização na entrada dos canais, usando *buffers* com capacidade de armazenar quatro *flits*, e centralizada. A memorização centralizada é implementada com dois *buffers* centrais com capacidade de armazenamento de 18 *flits*, um para cada lado do roteador. Quando um pacote é bloqueado em uma porta de entrada, seus *flits* são movidos para o *buffer* central correspondente, minimizando bloqueios por cabeça de linha (HOL). Essa memorização distribuída/centralizada se deve ao chaveamento do tipo *buffered wormhole* usado. Nesta técnica, a rede terá as características do chaveamento SAF (armazena-e-repassa) ou *wormhole*, de acordo com as condições de tráfego da rede.

A arbitragem implementada nesta rede é distribuída e o algoritmo de arbitragem é baseado em prioridades dinâmicas, com os pacotes de maior prioridade armazenados nos *buffers* centrais.

O controle de fluxo é baseado em créditos. Inicialmente, cada emissor possui um número de créditos equivalentes à capacidade do *buffer* de entrada, que é de quatro posições. Quando um *flit* é enviado, esse crédito é diminuído. Se o receptor transmite esse *flit* adiante, ele notifica o transmissor enviando um crédito através de uma das linhas de controle.

2.4.2 Rede aSoC

A rede aSoC (*adaptative System-on-Chip*) diferencia-se por ser a única rede pesquisada que possui roteamento estático, ou seja, definido em tempo de compilação. Possui topologia do tipo direta, podendo ser implementada em várias topologias, desde as do tipo grelha 2-D até topologias irregulares. O roteador é composto por quatro portas bidirecionais para conexão com nodos vizinhos e uma interface bidirecional para conexão com o núcleo local. O enlace é feito por canais com largura de 32 bits.

Esta rede foi desenvolvida para aplicações, tipicamente, *dataflow*, que possuem um padrão de comunicação estático, o que torna possível definir a maior parte das comunicações durante o tempo de compilação. Com isso, apesar da perda em flexibilidade, há uma redução do *hardware* e maximização das comunicações em um mesmo ciclo de execução, com um aumento do desempenho da comunicação.

Além de estático, o roteamento da rede aSoC é centralizado e determinístico. O roteador da rede possui uma memória interna, a memória de interconexão, onde é guardada a configuração do roteador para cada ciclo de execução. Esta configuração é determinada pelo escalonador no instante da compilação do programa. A posição da memória é indicada por um contador de programa, atualizado a cada ciclo de execução.

O chaveamento na rede aSoC é uma variação do chaveamento por circuito. A principal diferença ocorre no fato da rede aSoC não precisar enviar um cabeçalho reservando recursos, já que os circuitos são estabelecidos pelo algoritmo de

escalonamento antes da execução da aplicação. Estes caminhos são estabelecidos procurando maximizar o paralelismo nas comunicações.

Além da memória de interconexão, que possui tamanho variável conforme as aplicações-alvo do sistema, o roteador da aSoC possui uma memorização mínima de entrada e dois *buffers* FIFO de interface com o núcleo local. Esses *buffers* de interface entre núcleo e roteador compatibilizam a diferença de velocidade entre a rede e o núcleo, sendo usado como controle de fluxo do tipo FIFO.

2.4.3 Rede Octagon

Em (KARIM, 2002) é proposta uma arquitetura de rede chamada Octagon (Figura 2.7.a), que serve para atender ao alto desempenho de aplicações como roteadores de *backbone* da Internet. A arquitetura básica da rede Octagon é composta por oito nodos, formando uma topologia direta do tipo anel-cordal.

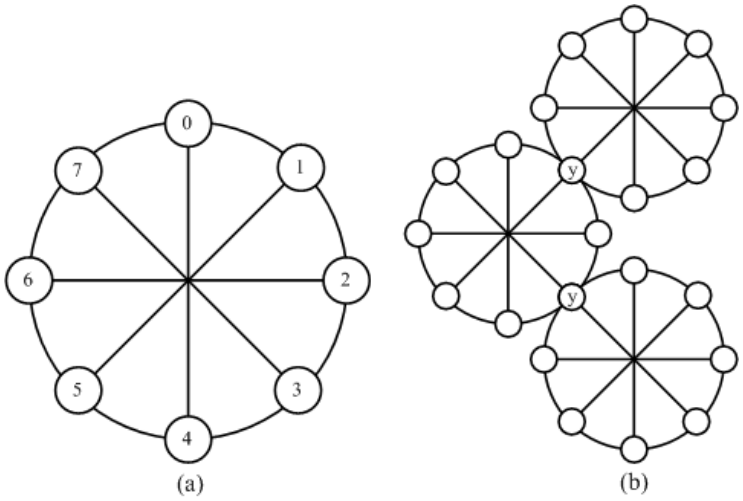

Figura 2.7: (a) Topologia da rede Octagon; (b) Escalabilidade

No exemplo apresentado pelo autor, o roteador OC-768, o sistema processa 57 milhões de instruções por segundo. Para atender a esta performance é necessária uma arquitetura multiprocessada com uma rede de comunicação que suporte uma taxa de dados de 40 Gbit/s. Em muitos sistemas semelhantes ao descrito são usadas redes de comunicação do tipo *crossbar*. O problema é que a *crossbar*, apesar de oferecer uma performance suficientemente alta, requer um grande número de fios para interconexões para permitir o desenvolvimento destes sistemas.

A Figura 2.8 mostra a disposição do leiaute físico de uma rede Octagon e *crossbar*. Além de um menor uso de fios para interconexões, as conexões na rede Octagon são menos complexas que na *crosssbar*.

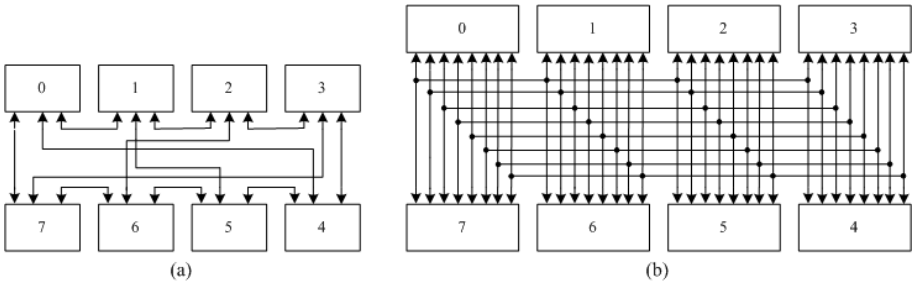

Figura 2.8: Distribuição física das redes: (a) Octagon; (b) *Crossbar*

O aumento do sistema também implica em um menor custo e complexidade para a Octagon. A estratégia de escalabilidade da Octagon consiste em usar um nodo como ponte para outras redes adjacentes. Este nodo ponte é identificado com um y na Figura 2.7.b. Desta forma, o custo da rede aumenta linearmente com o crescimento do número de nodos, enquanto na *crossbar* essa relação é quadrática.

O endereçamento dos nodos é feito por um campo de seis bits. Os três bits mais significativos são usados para identificar a rede adjacente para qual o pacote é destinado, e os três menos significativos identificam o nodo dentro da rede. O algoritmo de roteamento é dinâmico, distribuído e determinístico.

O algoritmo de roteamento permite que o número de enlace a ser percorrido por um pacote seja no máximo dois. Com o acréscimo de redes, este número aumenta. Para uma rede formada por dois anéis, o pacote pode atravessar no máximo quatro enlaces, aumentando para seis em uma rede com três anéis. Apesar do crescimento ser lento comparado com outras redes, ele não permanece constante como a rede *crossbar*.

O *throughput* da rede Octagon é comparada no artigo (KARIM, 2002) com duas arquiteturas, barramento e *crossbar*, através de sistemas equivalentes. O resultado foi um *throughput* 12 vezes maior se comparada com o barramento e 3 vezes maior que o *crossbar*.

O chaveamento na rede Octagon pode ser feito por circuito ou pacote. No chaveamento por circuito, os roteadores da rede memorizam os pacotes nos nodos intermediários caso haja contenção na rede. O modo de chaveamento por circuito usa um mecanismo que escalona as comunicações para que não concorram pelos mesmos recursos, procurando explorar ao máximo o paralelismo da rede.

2.4.4 Arquitetura de Comunicação Toróide Dobrado

Esta arquitetura foi apresentada em (DALLY, 2001) e proposta pelos autores como substituta das arquiteturas de conexões tradicionais dos sistemas integrados. O toróide dobrado é uma variação do toróide 2-D, onde o enlace mais longo é encurtado pela metade, como mostrado na Figura 2.9.

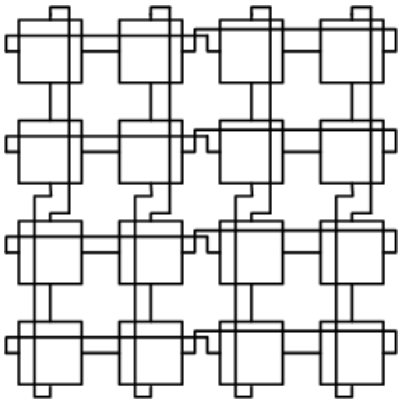

Figura 2.9: Arquitetura de comunicação toróide dobrado

Cada roteador é composto por cinco portas bidirecionais, uma delas interligada ao núcleo, e as outras em conexão com os roteadores vizinhos. Os canais de comunicação são largos, com campo de dados de 256 bits e 38 bits para sinais de controle e identificação da rota.

O roteamento da rede é dinâmico e determinístico e o mecanismo de chaveamento utilizado é o de pacotes do tipo *wormhole*. A memorização é realizada na entrada de cada canal, utilizando canais virtuais. Cada canal físico possui oito canais virtuais, o que minimiza os problemas de bloqueio por cabeça de rede e evita o congelamento da rede por *deadlock.*

2.4.5 Rede Hermes

A rede Hermes é uma arquitetura de comunicação composta por um conjunto de módulos apresentados em (MELLO, 2003). Cada módulo pode ser parametrizável em função de restrições de projeto como largura de palavra, profundidade das filas, topologia da rede entre outras. A parametrização da rede Hermes é gerada de forma automática pela ferramente NoCGen (MORAES, 2004). Atualmente a ferramenta pode gerar topologias diretas do tipo: (*i*) grelha 2-D, (*ii*) torus, (*iii*) torus dobrado e (*iv*) anel. No caso da topologia grelha 2-D, é possível escolher o algoritmo de roteamento da rede. Dentre os algoritmos de roteamento suportados estão: (*i*) XY adaptativo; (*ii*) *West-First Minimal*; (*iii*) *West-First Non-Minimal* e (*iv*) *Negative-First Non-Minimal*. Estes algoritmos de roteamento estão descritos em (GLASS, 1994).

O roteamento da rede pode ser classificado como distribuído e adaptativo e seu chaveamento é o de pacote do tipo *wormhole.* O controle de fluxo da rede Hermes é do tipo *handshake* e a arbitragem é centralizada com prioridade dinâmica, sendo que a prioridade de cada porta depende da última que obteve permissão de chaveamento. A estrutura do pacote da rede Hermes é formada por um *flit* com endereço destino, 1 *flit* com o tamanho da caga útil e uma carga útil pode ter até 255 *flits*.

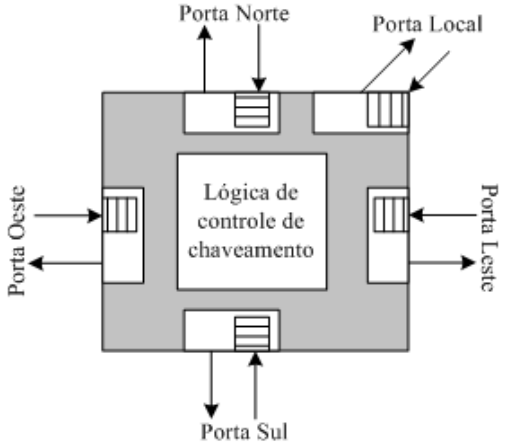

Figura 2.10: Estrutura do roteador Hermes

A Figura 2.10 mostra a estrutura básica do roteador Hermes, composto por cinco portas bidirecionais: Norte, Sul, Leste, Oeste e Local. Cada porta possui canais de entrada e saída, sendo que o canal de entrada possui um *buffer* FIFO para memorização dos pacotes. A largura do canal de dados é parametrizável, assim como a profundidade das filas de entrada. A lógica de controle de chaveamento engloba a arbitragem e lógica de chaveamento.

2.4.6 Rede SoCIN e SoCIN*fp*

A rede-em-chip denominada SoCIN foi desenvolvida no escopo de uma tese de doutorado do Programa de Pós-Graduação em Computação da UFRGS (PPGC-UFRGS) (ZEFERINO, 2003a).

Possui topologia direta, podendo ser configurada como uma grelha 2D, um toróide 2D ou até mesmo como toróide dobrado. Suas principais características são: controle de fluxo do tipo *handshake*; roteamento tipo fonte e determinístico (XY); chaveamento por pacote do tipo *wormhole*; arbitragem distribuída (*round-robin*) e; memorização de entrada usando *buffers* do tipo FIFO.

No chaveamento da rede SoCIN, por pacotes do tipo *wormhole*, as mensagens são transferidas sob a forma de pacotes, que possuem uma palavra de cabeçalho onde se localiza o destino do pacote e um sinal, chamado de bop (*begin-of-packet*), que indica o início do pacote e uma carga útil de tamanho variável. O término de um pacote é indicado por sinal chamado eop (*end-of-packet*) na última palavra da carga útil.

No roteamento XY implementado na rede SoCIN, a comunicação entre um remetente e um destinatário posicionados em linhas e colunas diferentes deve primeiro atravessar todos os canais X até atingir a coluna onde o destinatário está posicionado. Neste instante, a mensagem passa a ser transmitida na direção Y até chegar a seu destinatário.

O controle de fluxo é baseado no protocolo de *handshake*. Cada canal físico liga um emissor a um receptor. Uma linha val é ativada pelo emissor para sinalizar a presença de um dado no canal e o receptor ativa a linha ack para sinalizar que o dado foi recebido.

A arbitragem é feita de forma distribuída, onde cada canal de saída possui um arbitro *round-robin*. Um canal de entrada, ao receber um cabeçalho de pacote, executa o algoritmo de roteamento e envia uma requisição ao árbitro do canal de saída selecionado por esse algoritmo. O árbitro irá determinar qual requisição possui mais proridade, já que pode haver mais de uma requisição para o mesmo árbitro. Isso é feito com a aplicação de um critério de prioridades em que o canal selecionado em um ciclo passa a ter menor prioridade no ciclo seguinte de arbitragem.

Em cada canal de entrada existe um FIFO (*First-In, First-Out*) responsável pela memorização dos *phits* que chegam em cada porta de comunicação. Esses *buffers* possuem p posições com n + 2 bits em cada posição.

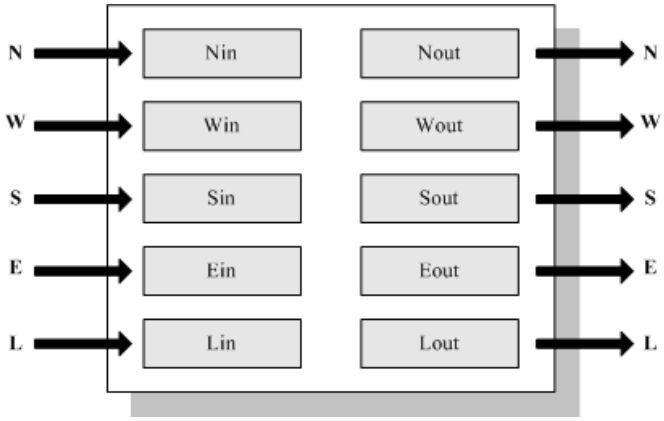

Figura 2.11: Estrutura do roteador RASoC

A rede SoCIN é composta pelo roteador RASoC (*Router Architecture for Systems-on-chip*), um *soft-core* descrito em VHDL com as seguintes características parametrizáveis: largura dos canais de comunicação (*n*), profundidade dos *buffers* (*p*) e largura de informação de roteamento no cabeçalho do pacote (*m*). O roteador possui cinco portas de comunicação bidirecional, com uma porta dedicada à comunicação com o núcleo local e quatro para a comunicação: N (*North*), S (*South*), W(*West*) e E (*East*).

Internamente, o roteador RASoC possui um *crossbar* 5x5 parcial que implementa apenas 20 das 25 conexões possíveis dele. Isso ocorre por que não é permitido a um canal de entrada ser conectado ao canal de saída associado à mesma porta. Tanto o *crossbar* como as estruturas de roteamento, arbitragem e controle de fluxo são distribuídas em módulos associados aos canais de entrada e de saída de cada porta bidirecional, conforme mostrado na Figura 2.11. Estes módulos são baseados em dois circuitos denominados *Input Channel* e *Output Channel*.

Tabela 2.2: Comparação entre SoCIN e SoCIN*fp*

	Roteamento	Controle de Fluxo	Memorização	Arbitragem
SoCIN	Determínistico	Handshake	Na entrada	Dinâmica
SoCIN*fp*	Determínistico ou parcialmente adaptativo	*Handshake* ou baseado em créditos	Entrada ou Saída	Dinâmica ou Estática

Mais recentemente, em (ZEFERINO, 2004), foi apresentada a continuação deste trabalho, que resultou na rede SoCIN*fp*. A rede é formada por um roteador chamado ParIS, composto por blocos parametrizáveis que permitem diferentes alternativas e implementações para os circuitos usados no roteador, como mostrado na Tabela 2.2.

2.4.7 Quadro Comparativo e Considerações

Neste capítulo são apresentados os conceitos essenciais relativos a redes paralelas de comunicação implementadas em chip e algumas das redes encontradas na literatura, assim como as suas principais características. A Tabela 2.3 apresenta um resumo com as principais características destas NoCs.

Em geral, as NoCs propostas são baseadas em conceitos bem estabelecidos na área de redes de interconexão. Usam topologias planares, chaveamento *wormhole* e roteamento dinâmico, embora algumas também usam chaveamento por circuito e roteamento adaptativo. Uma exceção é a rede aSoC, que possui um roteamento estático, o que a diferencia das demais redes.

A rede grelha é a que mais predomina na literatura. A opção deve-se ao fato da topologia ser de fácil implementação usando as atuais tecnologias planares dos CIs, o que possibilita uma estratégia de chaveamento simplificada e permite a construção de redes facilmente escaláveis. A torus bidirecional permite diminuir o diâmetro da rede, mas isso tem um custo em conexão e tamanho de fio. Uma outra opção é o torus 2D dobrado, que permite uma redução do aumento do comprimento dos fios quando comparado com a torus bidirecional.

Para redes nas quais é requerido um desempenho maior, podem ser usadas topologias alternativas como a rede SPIN e a rede Octagon. Estas redes oferecem uma menor latência e diâmetro de rede, se comparadas às redes grelha e torus. A rede SPIN adota a topologia de árvore gorda, enquanto, a rede Octagon sugere o uso da topologia anel cordal.

Entre as redes pesquisadas, a maioria utiliza como esquema de memorização filas de entrada. Esta opção permite filas mais simples, que ocupam menos área. Porém, filas de entrada apresentam o problema de bloqueio *head-of-line* (HOL). Se um *flit* ficar

bloqueado, todos os demais que estiverem na fila ficarão também bloqueados. Uma solução é utilizar canais virtuais, como os utilizados em (DALLY, 2001). Outro parâmetro importante é o tamanho da fila, que implica na necessidade de solução em termos de compromisso entre a contenção da rede, latência de entrega dos pacotes e sobrecarga de área da chave. Filas grandes conduzem a pouca contenção de rede, alta latência de pacote e chaves com bastante área. Em contrapartida, filas pequenas implicam em situações opostas.

Tabela 2.3: Principais características das NoCs

Redes	Topologia	Chaveamento	Roteamento	Arbitragem	Memorização	Controle de Fluxo
SPIN	Árvore gorda	Wormhole	Adaptativo	Distribuída	Entrada e centralizada	Baseado em Crédito
AsoC	Direta	Variação do chaveamento por circuito	Estático Determinístico	-	Entrada e centralizada	Tipo FIFO
Hermes	Grelha 2D	Wormhole	Parcialmente adaptativo	Centralizada Dinâmica	Entrada	Handshake
Octagon	Anel cordal	Circuito ou Pacotes	Determinístico	Centralizada	-	-
SoCIN*fp*	Direta	Wormhole	Determínistico ou parcialmente adaptativo	Dinâmica ou Estática	Entrada ou Saída	*Handshake* ou baseado em crédito
Toróide dobrado	Torus Dobrado	Wormhole	Determinístico	Distribuída	Entrada	Canais Virtuais

As escolhas arquiteturais feitas pelo projetista buscam desenvolver uma NoC com boa relação de custo e performance e, ao mesmo tempo, atender às necessidades dos futuros SoCs: baixo consumo de potência, reusabilidade, escalabilidade, paralelismo, distribuição do re*lógio* entre outras.

3 TONGA: UM ROTEADOR DE BAIXO CUSTO

Como mostrado no capítulo anterior, o avanço dos processos submicrônicos possibilitou a integração em um único chip de milhões de transistores e a criação de sistemas completos em um chip chamados de *System-on-Chip* (SoCs). Quanto mais complexos os sistemas ficam, melhores técnicas de projeto são necessárias para reduzir o custo e o tempo de projeto. Uma destas técnicas, denominada *Network-on-Chip*, baseia-se nos conceitos utilizados nas redes de interconexão de computadores paralelos, provendo, assim, uma plataforma de comunicação escalável para os sistemas embarcados, que ofereça paralelismo, que possa ser reutilizável para vários sistemas e tenha um baixo consumo de potência.

O principal destas arquiteturas de NoC é que elas atendam a SoCs com um grande número de núcleos e com alta performance, características destes futuros sistemas. Entretanto, pensando nos SoCs atuais, que requerem ao mesmo tempo escalabilidade, paralelismo e baixo consumo de potência aliados a um baixo custo de área, é desenvolvida nesta dissertação um roteador de baixo custo chamado Tonga.

Baseado no roteador RASoC e na rede SoCIN, o Tonga utiliza uma técnica de multiplexação dos canais de comunicação que permite que duas portas da interface do roteador com a rede utilizem a mesma estrutura de comunicação. A alternativa arquitetural utilizada no Tonga busca atender principalmente a um menor custo em área.

Assim como o RASoC, o Tonga foi desenvolvido como um *soft-core* em VHDL parametrizável em três dimensões: largura dos canais de comunicação, profundidade dos buffers e largura da informação de roteamento no cabeçalho do pacote. A estrutura do roteador é distribuída e modularizada, sendo que cada módulo possui uma interface que permite o reuso do bloco na implementação de alternativas arquiteturais.

O presente capítulo apresenta a arquitetura do roteador, assim como resultados da síntese do roteador em FPGA. O desempenho do roteador foi avaliado usando uma ferramenta em C++ apresentada em (KREUTZ, 2001). Os resultados também serão apresentados neste capítulo.

3.1 Comportamento do Tonga em uma rede

Assim como o roteador RASoC, o Tonga tem como interface de rede cinco portas bidirecionais, uma para cada direção (Norte, Sul, Leste, Oeste e Local). A porta Local conecta o núcleo ao roteador enquanto as demais portas fazem a interface com os roteadores vizinhos.

Utilizando o roteador Tonga é possível construir redes do tipo planar e direta, como a grelha 2D e toróide, mostradas na Figura 3.1. Nestas redes cada roteador é ligado a um núcleo, sendo este conjunto chamado de nodo.

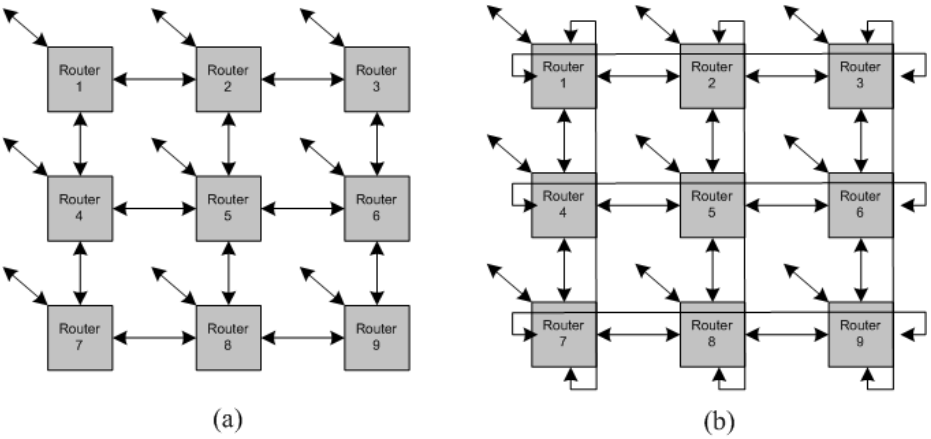

Figura 3.1: Redes Tonga tipo (a) grelha; (b) toróide

Os arcos que interligam os roteadores na Figura 3.1 são os enlaces da rede. Os enlaces do Tonga seguem o padrão da rede SoCIN, que por sua vez, são baseados na rede SPIN (GUERRIER, 2000) e são compostos por dois canais unidirecionais *simplex*. A largura física destes canais (*phit*) é de *n*+2, sendo *n* bits de dados e dois bits marcadores de início e fim de pacote (*bop* e *eop*). O parâmetro *n* deve ser definido em função dos requisitos do sistema, sendo que, no primeiro *flit*, estes bits contêm os sinais necessários para o transporte dos dados do pacote e sinais adicionais a serem manipulados por um protocolo de mais alto nível, tais como verificação de paridade e testes da rede.

A rede utiliza o chaveamento por pacotes do tipo *wormhole*. Cada roteador da rede tem capacidade para armazenar poucos *flits* de um pacote bloqueado. Assim, quando o cabeçalho de um pacote chega a um roteador e a porta de saída necessária ao seu encaminhamento não está disponível, o roteador absorve os *flits* possíveis de serem armazenados no espaço disponível em seu *buffer* e os demais *flits* são mantidos nos *buffers* dos roteadores anteriores no caminho do pacote.

A memorização dos *flits* é feita nos canais de entrada por buffers FIFOs distribuídos entre os canais de entrada, onde os dados são lidos na mesma ordem em que são escritos. Como já visto, apesar dessa abordagem apresentar como limitação o problema do bloqueio de cabeça de linha, é a mais econômica para a implementação de roteadores de baixo custo.

Cada buffer FIFO pode ser parametrizável quanto à sua capacidade de armazenar *p flits* de *n* bits, de acordo com os requisitos do sistema. Quanto maior for a profundidade *p* do *buffer* maior será o custo do roteador. Em compensação, a contenção na rede será menor, pois os pacotes irão ocupar menos canais quando bloqueados.

Na rede Tonga é usada a mesma técnica de controle de fluxo utilizada na rede SoCIN, isto é, baseada no protocolo de "aperto de mão" (ou *handshake*), em que o emissor informa a intenção de enviar um dado ao receptor através de uma linha de validação (*val*) e o receptor confirma a disponibilidade de espaço em buffer para receber esse dado através de uma linha de reconhecimento (*ack*). O sinal *val* é setado pelo emissor quando há *flit* pronto para ser transmitido e a linha *ack* é setada pelo receptor se houver espaço no *buffer* para memorizar o *flit*. Dessa forma, a transmissão só ocorre após um acordo entre os envolvidos na negociação (emissor e receptor).

A implementação do protocolo de aperto de mão nas redes SoCIN e Tonga é derivada do protocolo FIFO associado aos *buffers* de entrada (ZEFERINO, 2002),

conforme mostrado na Figura 3.2. A saída *rok* do buffer emissor indica a presença de dados a serem lidos no *buffer* emissor. Esta saída está conectada pela linha *val* à entrada de controle *wr* do buffer receptor. A entrada *wr* comanda a escrita do *flit* recebido e essa escrita só irá ocorrer se houver espaço disponível para tal, o que é indicado pelo sinal *wok*. A linha *ack* que retorna ao emissor será acionada apenas se *wok* e *wr* estiverem altos.

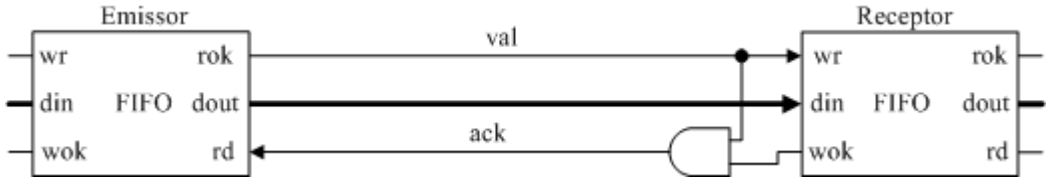

Figura 3.2: Controle de fluxo baseado no protocolo de aperto de mão.

3.1.1 Multiplexação dos canais internos de comunicação

A proposta do roteador Tonga é reduzir o custo do sistema com o acréscimo de uma rede-em-chip, procurando tornar essa opção mais interessante para sistemas em que a área do roteador represente parcela significativa da área total ocupada. Com este objetivo, o roteador Tonga divide suas estruturas de comunicação entre as cinco portas de comunicação, ou seja, é usada a mesma estrutura para transmitir e receber pacotes pelas portas Norte e Oeste (canal A) e pelas portas Sul e Leste (canal B), enquanto a porta Local possui um canal dedicado (canal L).

Cada porta terá um período de tempo (*slot time*) para utilizar o canal de comunicação. Após este período, o canal transmitirá os dados da outra porta de comunicação, como mostrado na Figura 3.3. O período de tempo que cada porta usa o canal de comunicação é configurável de acordo com a aplicação integrada pela rede, permitindo uma otimização da performance da rede. Se uma comunicação em um dos canais (A ou B) não tiver sido finalizada, é permitido ao canal concluir a transmissão antes de chavear para a próxima porta.

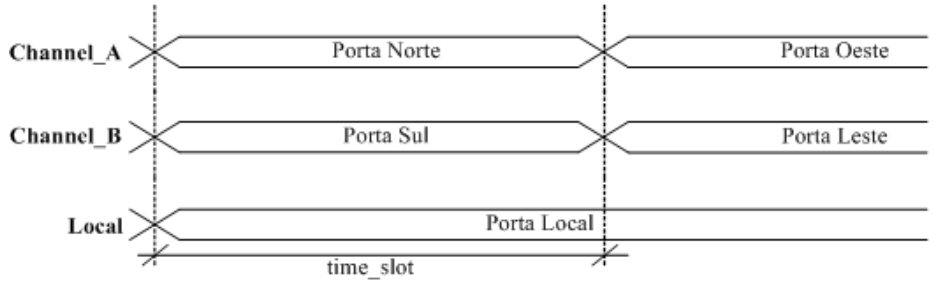

Figura 3.3: Uso dos canais do Tonga.

Por exemplo, na Figura 3.4 um pacote (*pck1*) é enviado pela porta N pelo canal A. Depois disso, um segundo pacote (*pck2*), que entra pela mesma porta, é enviado. Apesar do tempo de envio ter excedido o tempo reservado para esta porta, o pacote é inteiramente enviado antes do uso do canal ser passado para a porta W, onde o pacote *pck3* já espera seu envio.

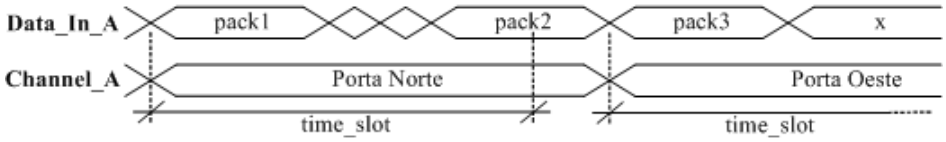

Figura 3.4: Tempo extra para transmissão do pacote.

3.1.2 Roteamento

O roteamento da rede Tonga é baseado no roteamento XY (DUATO, 1997), o mesmo usado na rede SoCIN. Na implementação do roteamento nas redes, um pacote deve percorrer todo o caminho horizontal até chegar na coluna onde está o nodo destino. A partir desse instante, o pacote começa a se deslocar na vertical em direção ao nodo destino, sem poder tomar mais a direção horizontal.

A definição das rotas por onde os pacotes serão transmitidos é feita em tempo de execução (roteamento do tipo dinâmico). Os pacotes contêm apenas um destinatário (*unicast*) e a definição da rota é feita pelo emissor do pacote através de uma consulta em tabela de roteamento (roteamento do tipo fonte baseado em tabela). Quanto à adaptatividade, o roteamento é do tipo determinístico, pois os pacotes sempre seguem o mesmo caminho entre os mesmos pares de nodos fonte-destino.

Este tipo de roteamento é muito usado nas redes de interconexão do tipo grelha e toróide e possui como vantagem garantir à rede liberdade de *deadlock* a um baixo custo de implementação. A desvantagem é que este algoritmo limita as rotas possíveis que um pacote pode seguir entre dois nodos, diminuindo assim a utilização da rede.

A diferença no roteamento das duas redes está na sua implementação. Na rede SoCIN o emissor do pacote consulta uma tabela de roteamento que indica o número de deslocamentos que o pacote deve realizar nas direções *X* e *Y* até chegar a seu destino. O número de deslocamentos em cada direção e o sentido destes deslocamentos são escritos no cabeçalho do pacote. As informações são usadas para determinar a cada roteador por qual porta o pacote deve ser transmitido. Após ser utilizada, a informação de deslocamento é atualizada pelo roteador. Quando os valores forem nulos, o roteador que recebeu o pacote o envia para o núcleo conectado a ele.

O cabeçalho contém quatro campos para informação de roteamento, dois para cada direção. Os campos *Xdir* e *Xmod* indicam o sentido e a direção do deslocamento em X, enquanto *Ydir* e *Ymod* indicam direção e sentido do deslocamento Y. Os campos *Xdir* e *Ydir* definem se o deslocamento do pacote é no sentido leste ou oeste para a direção X, e norte ou sul para a direção Y.

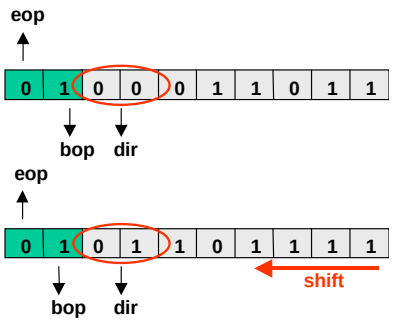

Figura 3.5: Algoritmo de Roteamento

Na rede Tonga, simplificou-se a atualização do cabeçalho e a decodificação da direção, procurando assim obter um menor custo em área. A informação de roteamento contida no cabeçalho do pacote na rede Tonga é composta por um conjunto de bits agrupados de dois em dois. Como mostrado na Figura 3.5, os dois bits mais à esquerda indicam a porta por onde o pacote será transmitido pelo roteador. Depois de lido, o cabeçalho do pacote é atualizado pelo roteador através de um deslocamento dos bits de

endereçamento. Os dois bits mais significativos usados para determinar a rota do pacote são descartados e os bits menos significativos do pacote são preenchidos com “11”.

A porta de saída por onde o pacote será transmitido depende da porta de entrada por onde o pacote chegou na rede, conforme mostrado na Tabela 3.1. Os bits de endereço indicam se o pacote deve ser transferido para o nodo que está à direita, à esquerda ou a frente da porta em que o pacote chegou ao roteador. Se ele estiver indicando “11” o pacote é enviado ao núcleo conectado à porta local do roteador. Quando o pacote é enviado do núcleo para a rede, os bits de roteamento assumem um significado diferente, indicando diretamente a porta por onde o pacote deve ser enviado.

Tabela 3.1: Direções de roteamento

Cabeçalho	Direção	
	Porta Local	Outras Portas
00	N	Direita
01	S	Esquerda
10	E	Em frente
11	W	Extrair a mensagem

Por exemplo, se o pacote estiver chegando pela porta leste e o cabeçalho estiver indicando “00” o pacote será roteado pela porta à direita, neste caso a porta norte, como mostrado na Figura 3.6. No caso dele estar chegando ao roteador pela porta oeste os mesmos “00” irão indicar que o pacote deve ser transmitido pela porta sul. Os pacotes que chegam pelas portas norte ou sul não podem ser transmitidos pelas direções “00” e “01”, devido às restrições do algoritmo de roteamento XY.

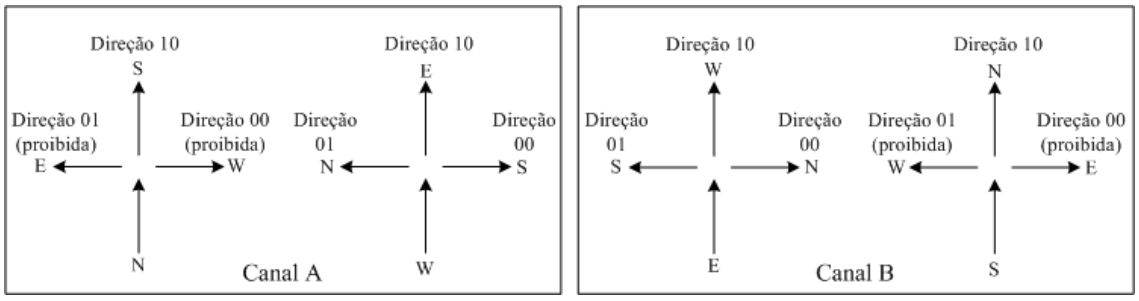

Figura 3.6: Direção muda de acordo com o canal de entrada

A escolha da multiplexação das portas Norte/Oeste e Sul/Leste deve-se em grande parte ao algoritmo XY. Qualquer outra opção de multiplexação possui uma mesma redução de custo, mas a combinação Norte/Oeste e Sul/Leste apresenta blocos de entrada e saída mais balanceado. Isso é demonstrado na Figura 3.7.

Para a transmissão de um pacote, a porta de entrada gera uma requisição de uso para a porta de saída por onde o pacote será enviado. Na Figura 3.7 estão representadas as requisições geradas pelas portas de entrada e as requisições de uso recebidas pelas portas de saída, sem levar em consideração a porta Local.

Na multiplexação Norte/Oeste e Sul/Leste são gerados e recebidos o mesmo número de sinais de requisição pelos canais A e B. A porta Norte envia apenas um sinal de

requisição destinado a porta Sul, assim como a porta Sul só envia um sinal de requisição para a porta Norte. No caso dos sinais recebidos pelas portas de saída, a porta Oeste recebe apenas o sinal de requisição da porta Leste, da mesma forma que a porta Leste recebe apenas o sinal de requisição da porta Oeste. Isso se deve as restrições do algoritmo XY, que não permite deslocamentos na direção X quanto o pacote já está sendo transmitido na direção Y.

Se fosse adotada a multiplexação Sul/Norte e Oeste/Leste, os canais A e B iriam enviar e receber um diferente número de requisições, gerando desta forma blocos de diferentes tamanhos, como mostrado na Figura 3.7. As características do algoritmo XY foram decisivas na escolha das portas a serem multiplexadas, de forma a permitir que os canais A e B tenham uma estrutura similar.

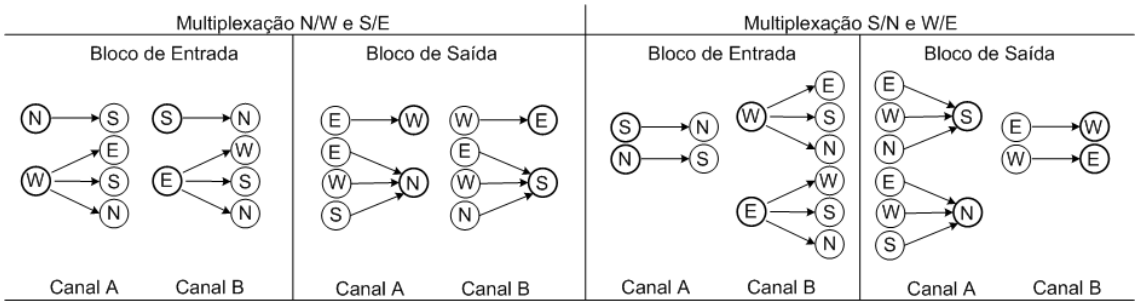

Figura 3.7: Sinais de requisição para diferentes tipos de requisição

3.1.3 Formato do pacote

A Figura 3.8 mostra o formato do pacote usado pelo Tonga. Ele é composto por um *flit* de cabeçalho identificado pelo bit da banda lateral *bop* (*begin-of-packet*) e por um número ilimitado de *flits* que compõem a carga útil do pacote, sendo que o último *flit* da carga útil é também o terminador do pacote e é indicado pelo bit da banda lateral *eop* (*end-of-packet*). Conforme pode ser observado na Figura 3.8, o cabeçalho é marcado pelo bit *bop* em 1, enquanto que o terminador é marcado pelo bit *eop* em 1.

Assim como no cabeçalho da rede SoCIN, a rede Tonga também utiliza o cabeçalho dividido em duas partes. A primeira parte do cabeçalho, composta de *m* bits, é chamada de RIB (*Routing Information Bits*) e é reservada para informação de roteamento. A segunda parte é reservada para protocolos de mais alto nível e se chama HLP (*High Level Protocol*) e pode ser usada, por exemplo, para identificar a *thread* que enviou o pacote em um processador *multithreaded*, implementar protocolos de detecção de erro de paridade ou realizar testes na rede.

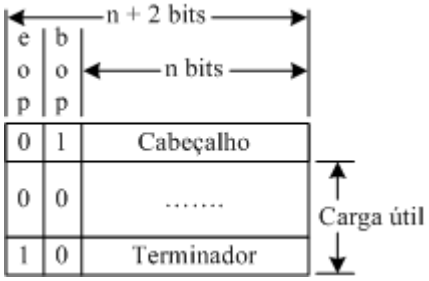

Figura 3.8: Formato do pacote das redes Tonga e SoCIN

A principal diferença no cabeçalho ocorre no campo RIB. Enquanto a rede SoCIN divide o campo RIB em duas partes (Figura 3.9), uma para indicar quantos deslocamentos o pacote irá ter em X (campos Xdir e Xmod) e a outra para indicar os deslocamentos em Y (campos Xdir e Xmod), a rede Tonga divide o campo de

endereçamento em conjuntos de dois bits. Estes pares indicam o caminho que o pacote deve seguir em cada etapa do seu deslocamento na rede até chegar ao seu destino. Esta diferença se deve ao fato das redes possuir em diferentes algoritmos de roteamento, como mostrado na seção 3.

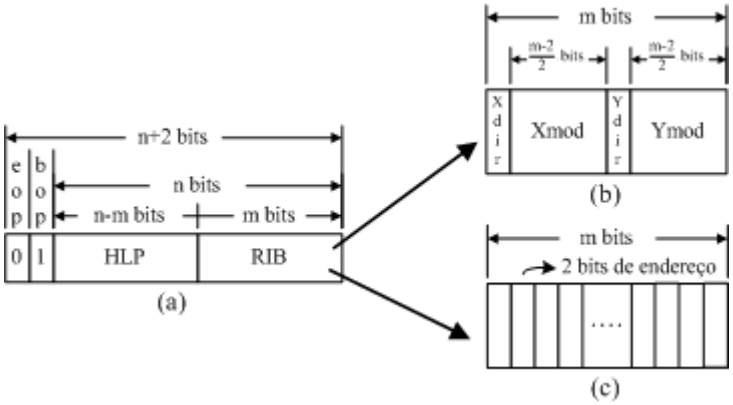

Figura 3.9: (a) Formato do cabeçalho das Redes Tonga e SoCIN; (b) Campo RIB da rede SoCIN; (c) Campo RIB da rede Tonga

O valor definido para *m* limita o tamanho da rede. Assim, para a rede SoCIN o número máximo de deslocamentos e a dimensão máxima da rede podem ser calculadas conforme mostrado em (ZEFERINO, 2003a) e são iguais a:

$$Xmod_{max} = Ymod_{max} = 2^{(m/2-1)} - 1 \quad (3.1)$$

$$k_{max} = 2^{(m-2)/2} \quad (3.2)$$

Na rede Tonga, a distância máxima de nodos que um pacote pode atravessar também varia com m e é dada por:

$$Maximum\_Nodes = \left(\frac{m}{4} + 1\right)^2 \quad (3.3)$$

No Tonga, por exemplo, se o tamanho dos dados transmitidos for igual a 8, o número máximo de roteadores que o pacote pode passar é de 4 e o tamanho máximo da rede é de uma grelha 3x3. Para um canal de 16 bits, a maior rede que pode ser feita irá ter 25 nodos em uma topologia grelha 5x5.

3.1.4 Arbitragem

A rede Tonga utiliza um esquema de arbitragem distribuída onde cada um dos três canais de saída possui um árbitro para seleção do *buffer* de entrada do roteador que transmitirá o seu conteúdo através do canal. O árbitro da rede Tonga foi apresentado em (SANTOS, 2003) e é baseado em um critério de prioridade dinâmica randômica.

O critério de prioridade randômica não depende da última comunicação realizada pelo canal, sendo o critério de prioridade modificado a cada ciclo de relógio. No roteador RASoC é aplicado um critério de prioridade dinâmico baseado em *Round-Robin*, em que a prioridade de uso do canal é definida a cada ciclo de arbitragem. Assim, o canal que realizou a última transmissão passa a ter o menor nível de prioridade na arbitragem seguinte.

A utilização de um árbitro randômico reduz o custo do árbitro, pois elimina toda a lógica de controle que atualiza a ordem de prioridade da arbitragem seguinte baseada na arbitragem corrente. A desvantagem deste árbitro é ser passível de deixar um pacote esperando indefinidamente para ser transmitido, ou seja, este árbitro pode levar algum pacote a sofrer com o problema de *starvation*. Como o objetivo do roteador Tonga é reduzir ao máximo a área consumida pelo roteador para ser usado em sistemas onde a área do roteador irá representar grande impacto, decidiu-se aplicar o critério randômico apesar deste não ser livre de *starvation*. Além do mais, em (SANTOS, 2003) são descritos três tipos diferentes de árbitros com a mesma interface de sinais e com estruturas básicas similares à mostrada na Figura 2.5 da seção 2.3.4. Desta forma pode-se implementar qualquer um dos dois árbitros de prioridade dinâmica rotativa do tipo *Round-Robin* apresentados neste trabalho sem alterar a interface do árbitro com o roteador e nem a sua estrutura básica, acresentando um custo maior de área ao roteador. Nas seções seguintes são apresentadas com mais detalhes a arquitetura do roteador e da estrutura do árbitro.

3.2 Arquitetura do Roteador Tonga

O roteador Tonga possui uma arquitetura baseada em uma abordagem distribuída. Os mecanismos que efetuam o roteamento, arbitragem e controle de fluxo, além do controle de multiplexação, são distribuídos em blocos associados aos canais de entrada e saída.

Conforme é ilustrado na Figura 3.10, os roteadores Tonga e RASoC possuem cinco portas bidirecionais denominadas L (Local ou local), N (North ou norte), E (East ou leste), S (South ou sul) e W (West ou oeste).

Enquanto o roteador RASoC utiliza blocos de entrada (sufixo *in*) e saída (sufixo *out*) dedicados para cada canal, o roteador Tonga compartilha o mesmo canal de entrada e saída com as portas norte e oeste (canal A) e portas sul e leste (canal B). A porta local permanece com um canal dedicado, como mostrado na Figura 3.10. O tempo de uso que cada porta irá ter para transmitir os seus pacotes é dado por um controle de multiplexação (bloco *Multiplexer Controller*).

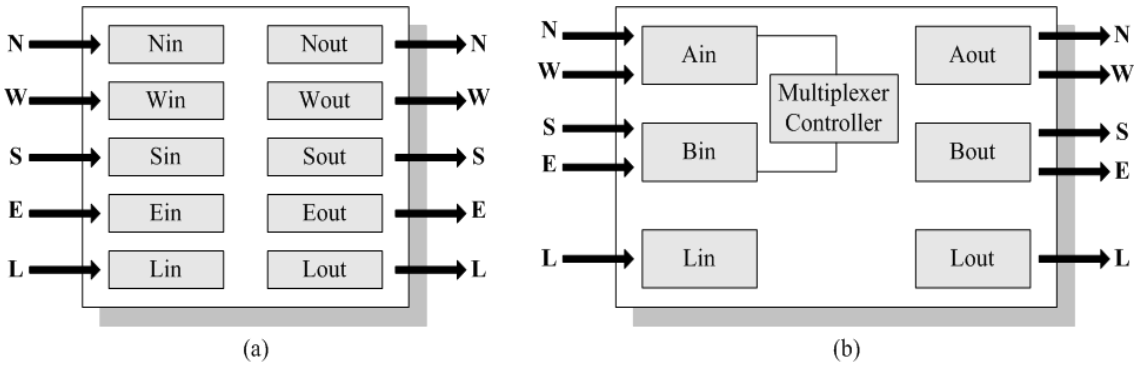

Figura 3.10: Blocos dos roteadores (a) RASoC; (b) Tonga

Este compartilhamento dos canais de comunicação foi realizado com o objetivo de reduzir a área do roteador. Em vez de usar cinco canais de entrada e saída, o roteador Tonga utiliza apenas três canais internos de comunicação.

A técnica, além de reduzir o número de canais internos, também diminui o tamanho do *crossbar* que implementa as conexões possíveis entre os canais de entrada e saída. Como pode ser observado na Figura 3.11.a, internamente o roteador RASoC possui um

crossbar 5×5 parcial que implementa apenas 20 das 25 conexões que poderiam ser realizadas em um crossbar com essas dimensões. As cinco conexões que não são implementadas são aquelas que interligariam o canal de entrada ao seu próprio canal de saída. Em outras palavras, um pacote que chega ao canal de entrada da porta N de um roteador, por exemplo, não pode ser encaminhado ao canal de saída dessa porta. Nesse caso, os únicos canais de saída possíveis de serem utilizados por esse pacote são aqueles associados às portas L, W, E e S.

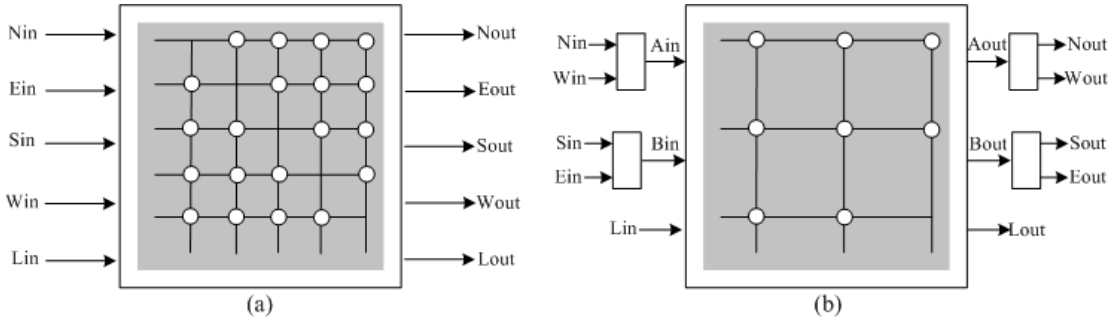

Figura 3.11: Interface e crossbar interno dos roteadores (a) RASoC; (b) Tonga

No Tonga, esse *crossbar* passou a ser de dimensão 3x3, como pode ser visto na Figura 3.11.b. Das nove conexões possíveis do *crossbar*, apenas uma não é implementada, a que interligaria o canal de entrada local ao seu próprio canal de saída. Todas as demais conexões são implementadas, incluindo as que conectam os canais de entrada A e B a seus próprios canais de saída. Isso se deve ao fato dos canais atenderem a duas portas e não apenas a uma, como ocorria na RASoC. Assim, a porta leste, que utiliza o canal A, pode precisar enviar pacotes pela porta norte, que também está conectada ao canal A.

O roteador Tonga é composto por três blocos. Os blocos *Input Channel e Output Channel* estão relacionados aos canais de entrada e saída, respectivamente. Estes blocos também fazem parte do roteador RASoC, de onde foram herdados. O terceiro módulo é o *Multiplex Controller*. A implementação destes blocos foi feita na forma de um modelo VHDL parametrizável que possui como parâmetros de síntese a largura do canal de dados (*n*), a largura da informação de roteamento (*m*) e a profundidade dos *buffers* de entrada (*p*). No código VHDL esses parâmetros são chamados respectivamente de *DATA_WIDTH*, *ROUTE_WIDTH* e *FIFO_DEPTH*.

Ao contrário do que ocorre no roteador RASoC, no Tonga os blocos de entrada e saída dos canais A, B e local, apesar de semelhantes, apresentam algumas diferenças estruturais. Por isso, existem três tipos de blocos *Input Channel* e *Output Channel*, um para cada canal. Estas estruturas estão interligadas por três barramentos, conforme mostrado na Figura 3.12.

O bloco *Input Channel* é responsável por determinar qual porta transmitirá um pacote e por enviar uma requisição para o bloco de saída que transmitirá o pacote. Neste módulo está implementado o controle de fluxo de entrada, o *buffer* de entrada e o algoritmo de roteamento.

A execução da multiplexação das portas de entrada é feita pelo bloco *Multiplex Controller*. A estrutura implementada no *Multiplex Controller* definirá como será realizada a multiplexação dos blocos e quanto tempo cada porta terá para usar o canal de comunicação para transmitir seus dados.

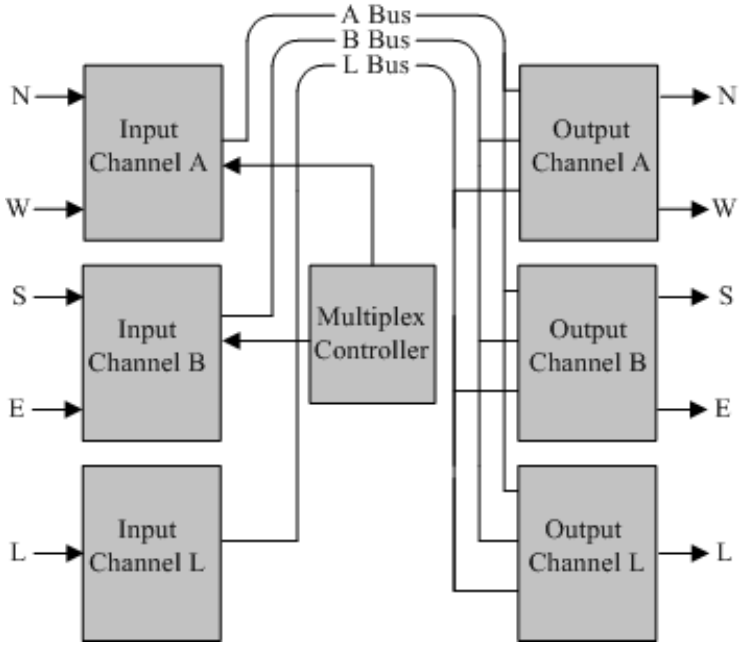

Figura 3.12: Arquitetura interna do Tonga

O terceiro bloco é o *Output Channel*, responsável por gerenciar as portas de saída. A seleção da porta de saída por onde os pacotes serão enviados é feita por este bloco, assim como o atendimento das requisições dos blocos *Input Channel*. Para tanto, cada bloco *Output Channel* possui um árbitro baseado na arquitetura apresentada em (SANTOS, 2003).

3.3 Blocos *Input Channel* e *Multiplex Controller*

O bloco *Input Channel* é dividido em outros quatro blocos: *Input Flow Controller*, *Input Buffer*, *Input Controller*, *Input Rd Switch* e um multiplexador. Como explicado anteriormente, há três blocos *Input Channel* usando uma estrutura similar. A diferença entre estas três estruturas é apenas o bloco *Input Controller* e a presença ou não do multiplexador, como mostrado na Figura 3.13.

O multiplexador recebe dados de duas portas, além de receber e transmitir os sinais de controle de fluxo *ack* e *val*. A seleção da porta que utilizará o canal de comunicação é feita pelo sinal *sel_channel*, gerado pelo bloco *Multiplex Controller*.

O bloco *Multiplex Controller* consiste em um contador simples que define o tempo pelo qual cada porta utilizará o canal de comunicação. Este tempo, chamado de *time slot*, é configurável de acordo com o sistema a ser sintetizado.

Se o canal estiver transmitindo algum pacote no momento em que o tempo de uso acabar, não será concedido o uso para a outra porta, até que todo o pacote seja transmitido. Para verificar essa condição, o sinal de seleção passa por um *flip-flop D* habilitado por uma porta NOR, mostrado nas Figuras 3.13.b e 3.13.c.

As entradas da porta OU recebem os sinais de confirmação *gnt* enviados pelos blocos *Output Channel*. Estes sinais indicam se algum canal de saída está sendo usado pelo bloco *Input Channel*, ou seja, um sinal *gnt* no nível lógico ‘1’ indicará que há dados sendo transmitidos pelo canal de entrada e a saída da porta NOR continuará em ‘0’. Nesta situação o sinal de seleção será mantido no seu valor anterior, pois a entrada de habilitação do *flip-flop D* estará em zero.

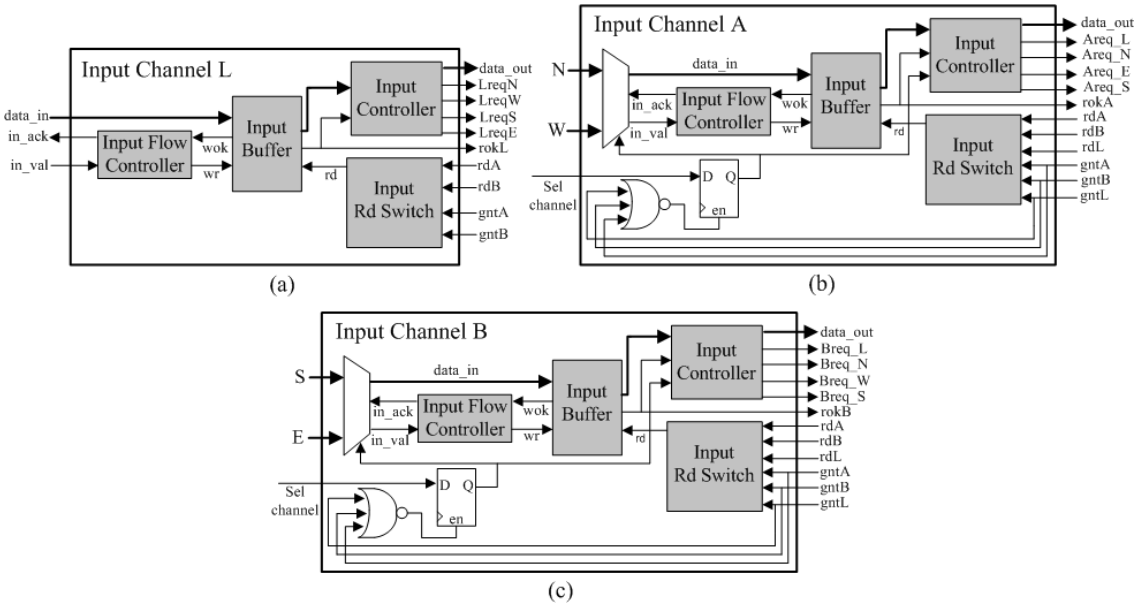

Figura 3.13: Módulos (a) Input Channel Local; (b) Input Channel A; (c) Input Channel B

O bloco *Input Flow Controller* faz a interface entre a rede e o *buffer* de entrada, implementando o protocolo de controle de fluxo *handshake* usado para regular o fluxo de dados que chegam na porta de entrada e adaptando o protocolo de aperto de mão ao protocolo FIFO. Nos dois protocolos, o controle de fluxo é realizado por meio do uso de dois sinais de controle: um de requisição (ou validação), no sentido emissor-receptor, e outro de retorno (ou reconhecimento), no sentido receptor-emissor. Como a diferença entre os dois protocolos resume-se à temporização do sinal de retorno, a lógica necessária a essa adaptação é bastante simples. No protocolo FIFO, o sinal de retorno consiste em um bit de condição que informa, antecipadamente, a habilidade de receber um novo dado, sendo ativado independentemente de haver ou não uma requisição. No protocolo de aperto de mão, o sinal de retorno só é ativado após o recebimento de uma requisição e somente se o receptor for capaz de receber o dado a ser enviado. Para o emissor, a ativação desse sinal é interpretada como uma confirmação de entrega do dado sendo transmitido. Com base nisso, a lógica de adaptação dos protocolos é constituída por uma operação lógica “E” que condiciona a ativação do sinal de retorno do protocolo de aperto de mão (*ack*) à ativação do sinal de requisição desse protocolo (*val*) e do sinal de retorno do protocolo FIFO (*wok*), conforme é ilustrado na Figura 3.14.

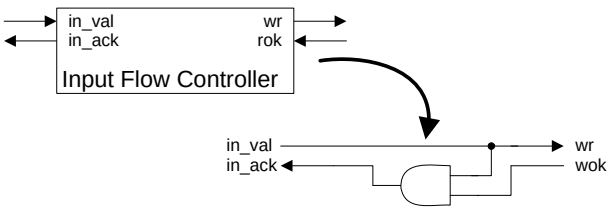

Figura 3.14: Estrutura do bloco IFC.

A memorização dos dados que chegam ao canal de comunicação é feita no bloco *Input Buffer*. Este bloco consiste em um *buffer* FIFO com profundidade parametrizável na síntese do roteador. A estrutura do roteador permite também construir roteadores com *buffers* de diferentes profundidades.

O algoritmo de roteamento XY apresentado anteriormente está implementado no bloco *Input Controller*. A porta de saída selecionada pelo algoritmo para enviar o pacote depende da porta de saída por onde o pacote chegou no roteador e da informação de roteamento contida no cabeçalho do pacote. Além disso, o roteamento XY não permite que o pacote se desloque em uma direção vertical enquanto este não fizer todo seu deslocamento na horizontal.

O bloco *Input Controller* também verifica os bits de início e fim de pacote (*bop* e *eop*). Quando o *flit* que chega à porta tem setado o bit *bop*, o roteador lê os dois primeiros bits do cabeçalho que indicam a porta por onde o pacote deve ser enviado e desloca para a esquerda todo o cabeçalho do pacote. Os demais *flits* são transmitidos para o próximo nodo através do caminho já estabelecido pelo primeiro *flit*.

Depois de executar todo o algoritmo de roteamento, o bloco *Input Controller* envia um sinal de requisição (*req*) para o *Output Channel* selecionado. Quando um sinal de confirmação (*gnt*) é recebido como resposta, os canais são conectados e o pacote pode ser transmitido pelo canal de saída se o sinal *read* estiver setado. O sinal *read*, que chega ao bloco *Input Rd Switch*, é enviado pelo bloco *Output Flow Controller* e indica a disponibilidade do *buffer* do canal de entrada para onde o pacote está sendo enviado.

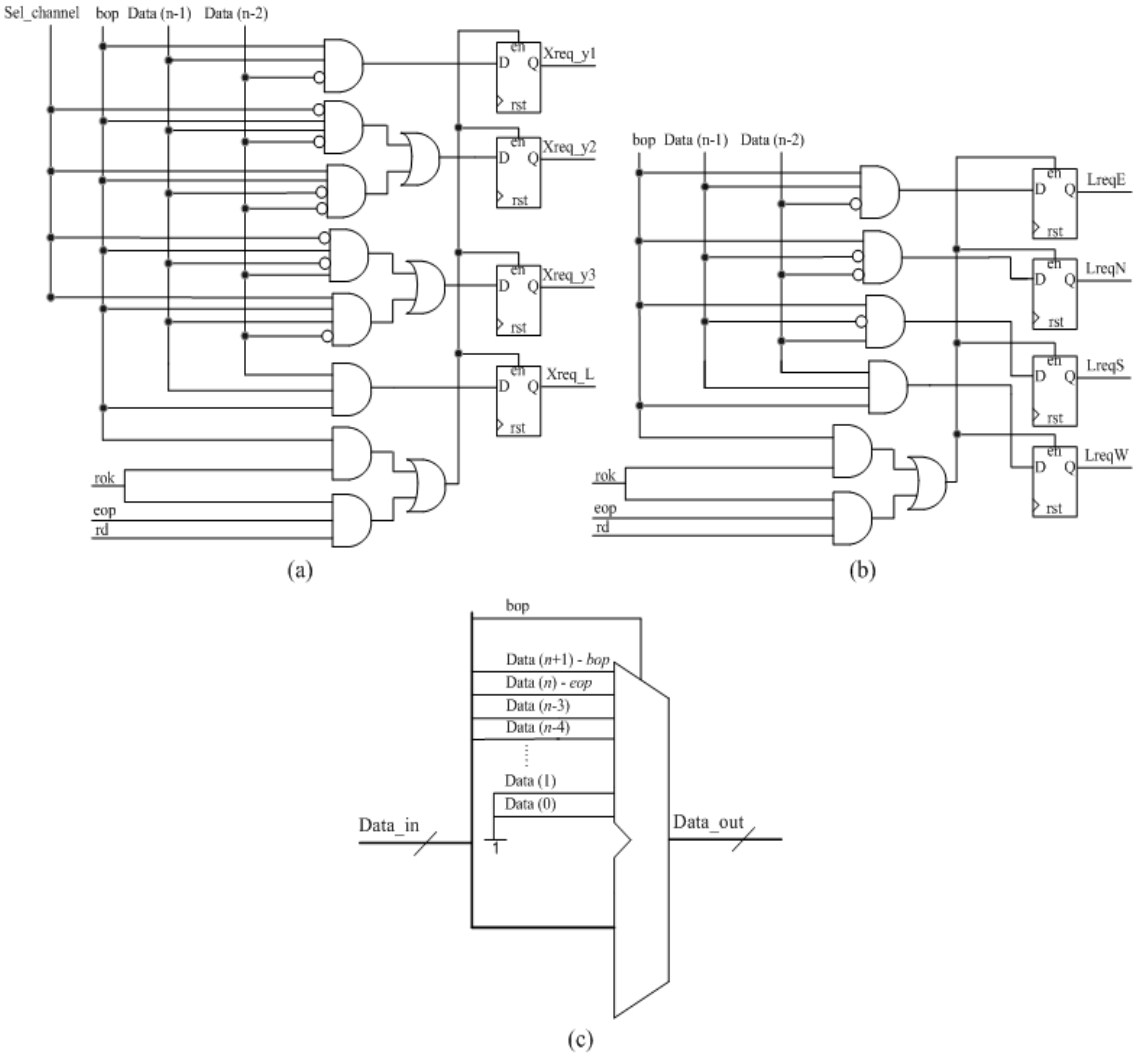

Figura 3.15: Circuitos (a) *Input Controller* A e B ; (b) *Input Controller* L; (c) circuito de atualização do cabeçalho dos blocos *Input Controller* A, B e L

Na Figura 3.15 estão os circuitos de geração da requisição e atualização do cabeçalho que compõem os blocos *Input Channel*. A geração da requisição ocorre quando o sinal de *bop* e o sinal *rok* estão em ‘1’, indicando a presença do primeiro *flit* de um pacote e a disponibilidade do *flit* para leitura no *buffer*. Desta forma, os *flip-flops* D são habilitados, permitindo a atualização dos sinais de requisição.

O sinal de requisição não muda até o bit *eop* e o sinal *rd* indicarem que este é o último *flit* do pacote e o *flit* chegar ao *buffer* de destino. Quando estes dois sinais estiverem em ‘1’, os *flip-flops* são habilitados novamente, zerando a requisição gerada.

A atualização do pacote é feita pelo circuito da Figura 3.15.c. O circuito consiste em um multiplexador de duas entradas para uma saída, onde a seleção é feita pelo sinal *bop*. Se o *bop* estiver ativo, é selecionada a entrada do multiplexador onde o cabeçalho é atualizado, descartando os dois bits usados no roteamento e setando os bits menos significativos do cabeçalho. Caso o *bop* seja zero, isso irá indicar que o *flit* contém carga útil e será transmitido sem alterações em seu conteúdo.

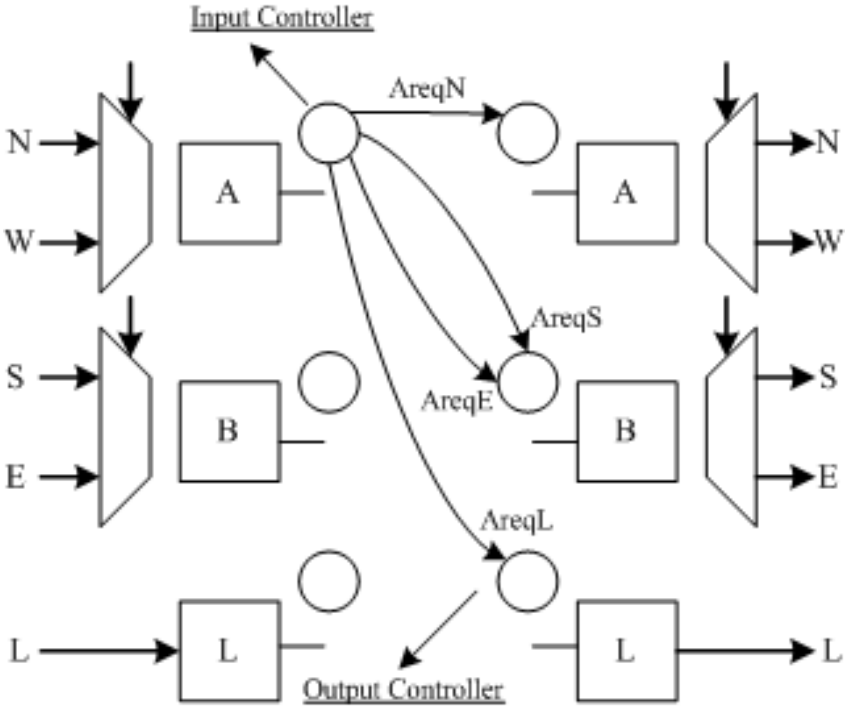

Figura 3.16: Requisições geradas pelo Input Channel A

As requisições emitidas pelo *Input Controller* são limitadas a quatro. Por exemplo, o canal A irá gerar requisição para as portas norte, sul, leste e local, como mostrado na Figura 3.16. Não é gerada uma requisição do canal A para transmitir pela porta oeste devido às restrições do algoritmo de roteamento, que só permite deslocamento na vertical depois de ter percorrido todo o caminho na horizontal. Por isso, a porta oeste pode transmitir pela porta norte, mas a porta norte jamais irá usar a porta oeste, não ocorrendo nunca uma requisição do canal A para transmitir pela porta oeste.

Na Tabela 3.2 estão indicadas as requisições que cada canal pode gerar. O sinal indica qual canal está requisitando e a porta que ele quer usar. Por exemplo, ‘AreN’ quer dizer canal A requer o uso da porta norte. Já os sinais de confirmação (gnt) e leitura (rd) são de três por *Input Channel*, um para cada *Output Channel*. A exceção é o *Input Channel local*, que recebe apenas dois sinais de confirmação e leitura, já que não pode transmitir por seu próprio canal de saída.

Tabela 3.2: Sinais de requisição no Tonga

	Sinais de Requisições		
	Canal A	Canal B	Canal Local
Canal A	AreqN	BreqN	LreqN
	-	BreqW	LreqW
Canal B	AreqS	BreqS	LreqS
	AreqE	-	LreqE
Canal Local	AreqL	BreqL	-

3.4 Bloco *Output Channel*

O bloco Output Channel é composto pelos blocos Output Data Switch, Output Controller, Output Flow Controller, Output Rok Switch e, dependendo do canal implementado, o bloco Output Demux. Assim como no Input Channel, há três blocos Output Channel usando uma estrutura similar, com algumas diferenças (Figura 3.17). No canal Local há apenas dois sinais de grant, rok, req e de dados (data), enquanto nos canais A e B existem três deles. Além disso, o canal L não possui o bloco Output Demux.

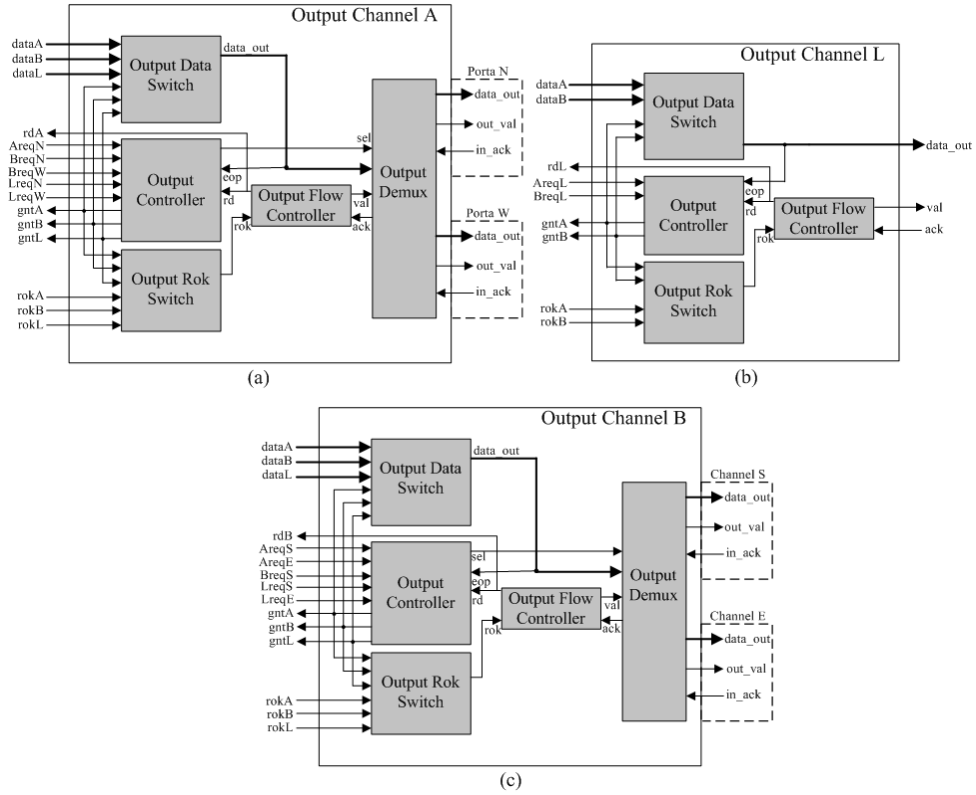

Figura 3.17: Módulos (a) Output Channel Local; (b) Output Channel A; (c) Output Channel B.

A arbitragem é executada pelo bloco *Output Control*. O algoritmo de arbitragem seleciona um dos sinais de requisição emitidos pelos canais de entrada e envia um sinal de concessão do canal (*gnt*). O árbitro utilizado no Tonga é baseado na arquitetura apresentada em (SANTOS, 2003) e mostrada na Figura 3.18. Ele é composto por dois blocos: PPE (*Programmable Priority Encoder*) e PG (*Priority Generator*).

O bloco PPE é composto por um conjunto de células de arbitragem. Estas células possuem três entradas: P, R e Imed_in. A entrada R recebe uma das requisições dos blocos *Input Controller* e a entrada P recebe um dos sinais P_n gerados pelo bloco *Priority Generator* do árbitro. Dos *n* sinais P gerados pelo bloco *Priority Generator*, apenas um estará ativo, indicando qual das requisições tem prioridade no ciclo de arbitragem corrente. A entrada Imed_in indica se a célula anterior gerou ou não um sinal de concessão.

Apenas uma saída s_G_n é ativada em cada ciclo de arbitragem. Ela só é ativa em duas condições: (*i*) se P_n e R_n estiverem ativos ou (*ii*) se R_n e $Imed\_out_{n-1}$ estiverem ativos. O sinal Imed_out é ativado pela célula para indicar à próxima célula que não foi gerado um sinal de concessão (s_G_n), passando, dessa forma, a prioridade do ciclo de arbitragem para a outra célula.

O critério de prioridade utilizado no bloco PG é do tipo randômico, implementado com a utilização de um contador em anel onde apenas um dos bits está setado. A cada ciclo de relógio este bit é deslocado para a saída seguinte, alterando a prioridade do circuito. Para se obter árbitros com diferentes critérios de prioridade, basta alterar o bloco PG. Em (SANTOS, 2003) são apresentados árbitros com critérios de arbitragem baseados em uma estrutura *Round-Robin*, mas o que possui um custo menor em área é aquele com prioridade do tipo randômica.

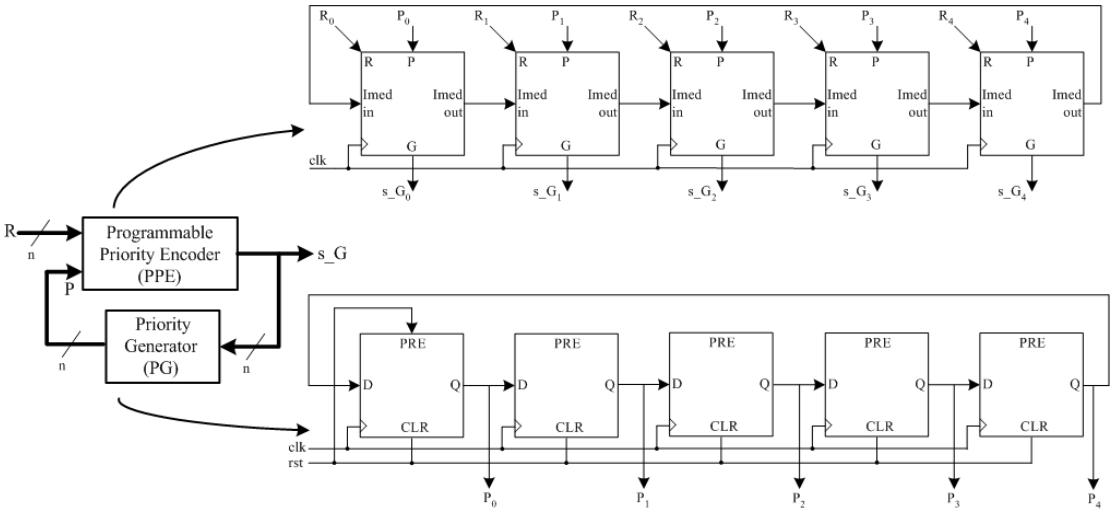

Figura 3.18: Arquitetura do árbitro com critério de prioridade randômica

Os blocos *Output Channel* A e B utilizam um árbitro com cinco células de arbitragem, como mostrado na Figura 3.18, e recebem cinco requisições geradas pelos blocos de entrada. Porém, as requisições geram apenas três sinais de reconhecimento: *gntA*, *gntB* e *gntL*. Como duas requisições partem do mesmo canal de entrada *Input Channel*, a ativação de apenas um deles pode gerar o mesmo sinal de concessão. Por exemplo, os sinais *AreqS* e *AreqE* (canal A requisita o uso da porta Sul e canal A requisita o uso da porta Leste) ativarão a mesma saída *gntA*, sinalizando a este bloco que pode transmitir seus dados pelo canal de saída requisitado. Assim, as duas saídas

s_G relacionadas ao mesmo canal passam por uma porta lógica OU, gerando um único sinal *gnt*.

Tabela 3.3: Geração das saídas do *Output Controller*

	Output Controller A		Output Controller B		Output Controller L
Sinais do árbitro	Saída *grant* ativa	Entrada sel do Output Demux	Saída *grant* ativa	Entrada sel do Output Demux	Saída *grant* ativa
s_G_0	GntA	sel => Norte	gntA	sel => Sul	gntA
S_G_1	GntB	sel => Norte	gntA	sel => Leste	gntB
S_G_2	gntB	sel => Oeste	gntB	sel => Sul	-
S_G_3	gntL	sel => Norte	gntL	sel => Sul	-
S_G_4	gntL	sel => Oeste	gntL	sel => Leste	-

No caso do bloco *Output Channel* L são recebidas apenas duas requisições de uso: *AreqL* e *BreqL*. A resposta à solicitação de uso do canal é feita neste caso pelos sinais *gntA* e *gntB*, respectivamente.

A seleção da porta pela qual o pacote é destinado é realizada pelo bloco *Output Demux*. A saída *sel* do *Output Controller* indica ao demultiplexador do *Output Demux* por qual porta de saída o pacote será transmitido. Este sinal *sel* é gerado com base nas saídas s_G do árbitro. Na Tabela 3.3 é mostrado qual sinal *gnt* é ativado pelos sinais s_G e a porta apontada pela saída *sel* do *Output Controller*.

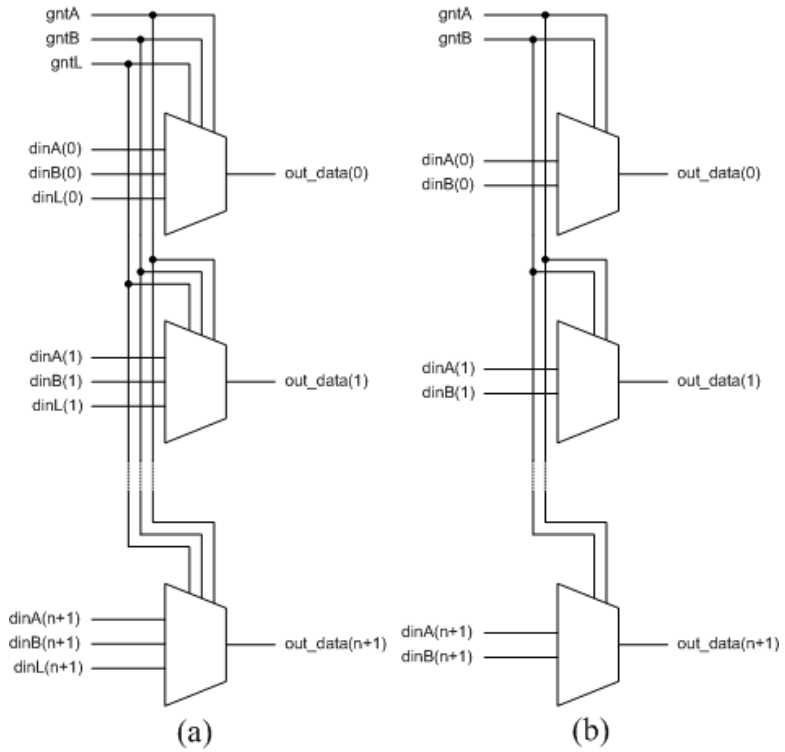

Figura 3.19: (a) *Output Data Switch* dos canais A e B; (b) *Output Data Switch* do canal L

Depois de selecionar uma das requisições, o árbitro envia o sinal de g*rant* para o canal selecionado e também para os blocos *Output Data Switch* e *Output Rok Switch*, comandando estes blocos.

O bloco *Output Data Switch* conecta o sinal de dados (*data*) do canal selecionado à entrada do *Output Demux*, que envia o pacote para uma das duas portas de saída. Este bloco é composto nos canais A e B por *n*+2 multiplexadores 3x1 de 1 bit (Figura 3.19.a) e *n*+2 multiplexadores 2x1 de 1 bit para o canal L (Figura 3.19.b).

A seleção do sinal *rok* do bloco *Input Buffer* do canal de entrada conectado ao canal de saída é feita pelo bloco *Output Rok Switch*. A seleção, assim como no bloco *Output Data Switch*, é feita utilizando os sinais de confirmação de seleção recebidos do bloco *Output Controller*. O circuito consiste em um multiplexador 3x1 de 1 bit para os canais A e B (Figura 3.20.a) e um multiplexador 2x1 de 1 bit para o canal L (Figura 3.20.b).

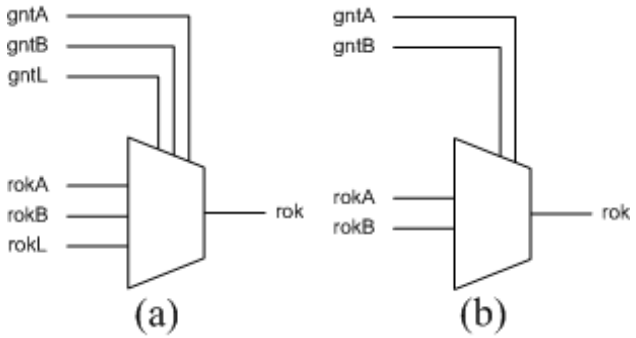

Figura 3.20: (a) *Output Rok Switch* dos canais A e B; (b) *Output Rok Switch* do canal L

No roteador RASoC os circuitos eram compostos por multiplexadores 4x1 de 1 bit, sendo um para cada canal, totalizando cinco blocos *Output Data Switch* e *Output Rok Switch*. No roteador Tonga, com a multiplexação dos canais, foi possível reduzir o tamanho dos multiplexadores e o total de blocos usados no roteador.

Os sinais *rok* e *rd* que passam pelo *Output Flow Controller* correspondem aos sinais de controle de fluxo *val* e *ack*, respectivamente. O bloco tem como função fazer a interface dos sinais que controlam o fluxo de dados, fazendo a adaptação dos protocolos *handshake* e FIFO. Embora neste circuito ele não implemente nenhuma lógica, foi criado no roteador RASoC para implementação de técnicas alternativas de controle de fluxo como, por exemplo, controle de fluxo baseado em créditos. Neste caso o bloco *Output Flow Controller* poderia implementar um contador de crédito para contabilizar o espaço disponível no *buffer* do canal de entrada conectado a este canal de saída (ZEFERINO, 2002).

3.5 Resultados da Síntese do Roteador

Nesta seção, são apresentados os resultados da síntese do roteador Tonga para diferentes configurações dos parâmetros referentes à largura do canal e ao tamanho do *buffer*. Os resultados estão baseados na síntese do modelo VHDL em FPGA com utilização da versão 4.2 SP1 Web Edition do ambiente QUARTUS II da Altera (ALTERA, 2005). As quantidades de células lógicas consumidas em cada configuração são mostradas, bem como parâmetros de desempenho obtidos através da ferramenta análise de temporização. Nos resultados apresentados o FPGA alvo das sínteses foi o dispositivo EP2A15F672C9 (ALTERA, 2002).

Para uma melhor análise de área o ideal seria sintetizar o roteador para um ASIC, utilizando um software como o Leonardo Spectrum. A desvantagem de utilizar FPGA na análise de área é que na síntese do roteador diferentes estruturas que teriam diferentes custos em área, tais como um multiplexador 4:1 e um multiplexador 2:1, irão ocupar o mesmo número de células lógicas. Nas medidas feitas neste capítulo optou-se por sintetizar o circuito para FPGA por ser o QUARTUS a ferramenta de síntese disponível durante o trabalho. Apesar disto, no Capítulo 4 são mostrados alguns resultados de síntese que foi possível fazer no Leonardo Spectrum.

Tabela 3.4: Resultados de síntese do roteador Tonga

	DATA_WIDTH		
	8	16	32
FIFO_DEPTH	#LCs	#LCs	#LCs
2	306	458	762
3	373	589	964
4	430	670	1150
8	664	1064	1864
16	1070	1792	3230

O roteador Tonga foi sintetizado em diferentes larguras de canais (parâmetro DATA_WIDTH na descrição VHDL) e tamanho dos *buffers* (parâmetro FIFO_DEPTH na descrição VHDL). Os resultados são mostrados na Tabela 3.4 e ilustrados no gráfico da Figura 3.21.

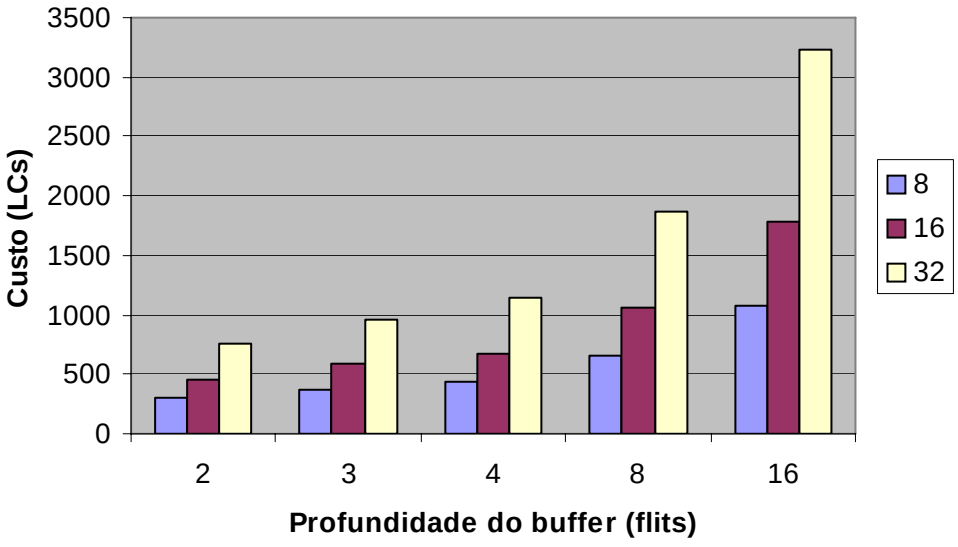

Figura 3.21: Área do roteador Tonga para diferentes configurações

Um forte impacto na área é notado com o aumento dos *buffers*. O circuito de memorização tende a ter um custo adicional grande com o aumento de sua profundidade, devido ao multiplexador da saída do *buffer* e ao registrador que seleciona a posição a ser lida.

A Figura 3.22 mostra também os resultados de freqüência máxima obtidos na síntese. Há uma tendência da diminuição da freqüência com o aumento da profundidade do *buffer* que não se reproduz em todas as configurações. Por exemplo, na configuração de 16 bits de canal, ocorre um aumento da freqüência quando a profundidade do *buffer* passa de duas para três posições. No caso do tamanho do canal, também não é possível identificar uma relação com a freqüência, pois, para cada profundidade do *buffer*, diferentes larguras de canais apresentaram freqüência maior. Os efeitos podem ser atribuídos ao compilador e à arquitetura do FPGA e também apareceram na síntese da RASoC (ZEFERINO, 2003a).

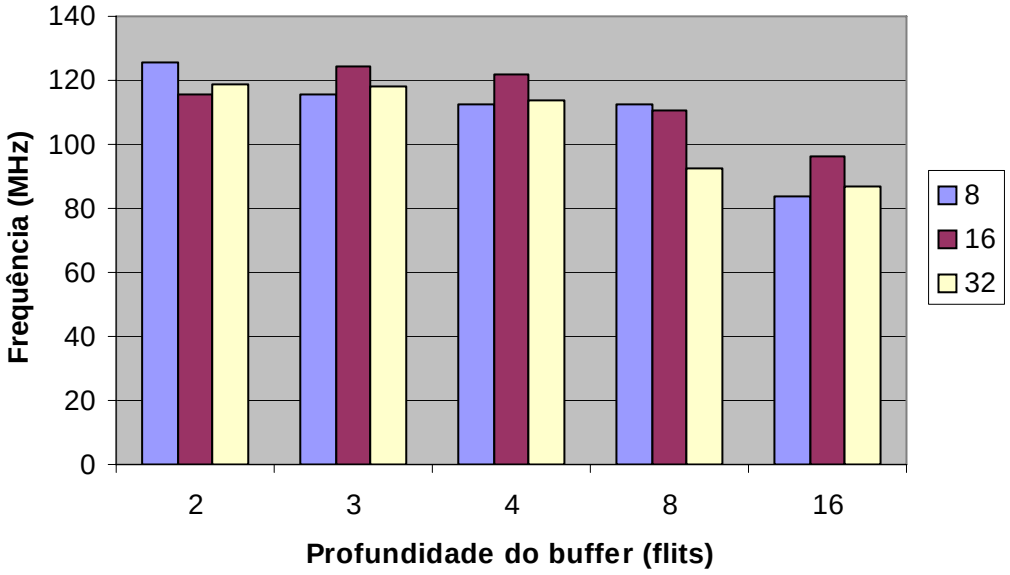

Figura 3.22: Freqüência do roteador Tonga para diferentes configurações

3.5.1 Comparação da síntese entre os roteadores Tonga e RASoC

Para permitir a comparação entre os dois roteadores e, dessa forma, analisar o ganho de área obtido, o código VHDL do roteador RASoC foi sintetizado a partir da alteração dos parâmetros de largura de canal e tamanho do *buffer*.

Comparando a área em *logic cells* de ambos os roteadores nota-se uma redução de área média de 50%. A maior redução, de 55%, ocorre para uma configuração com largura de canal de 8 bits e tamanho dos *buffers* de 2 posições, enquanto o menor ganho em área, de 42%, ocorre para um canal de 32 bits e *buffers* de 8 posições, como é mostrado na Figura 3.23.

Ao ser observada a Figura 3.23 nota-se que, com o aumento do tamanho dos canais, o ganho em área diminui. A causa da redução é presença de multiplexadores e demultiplexadores que controlam o uso dos canais de comunicação no roteador Tonga e que ocuparão uma área maior se for aumentada a largura dos canais de comunicação.

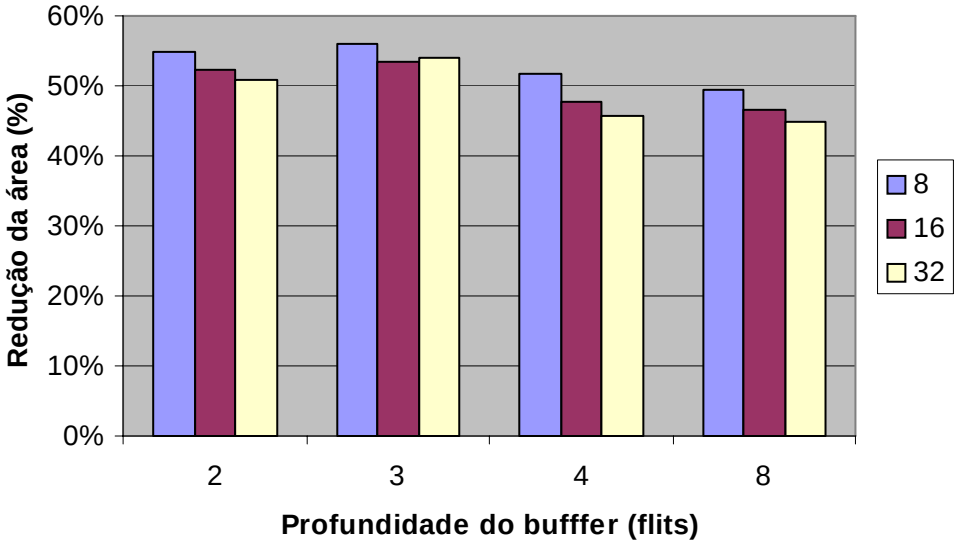

Figura 3.23: Redução de área do Tonga em comparação ao RASoC

Com relação à frequência, o Tonga apresenta um ganho de performance de, em média, 40%. O maior ganho é de 51% para um canal de 16 bits e *buffers* de 8 posições, enquanto o menor ganho ocorre para um canal de 8 bits com *buffers* de 3 posições.

O aumento na freqüência de operação compensará o aumento da latência da comunicação. A análise comparativa do comportamento da comunicação dos dois roteadores será mostrada a seguir.

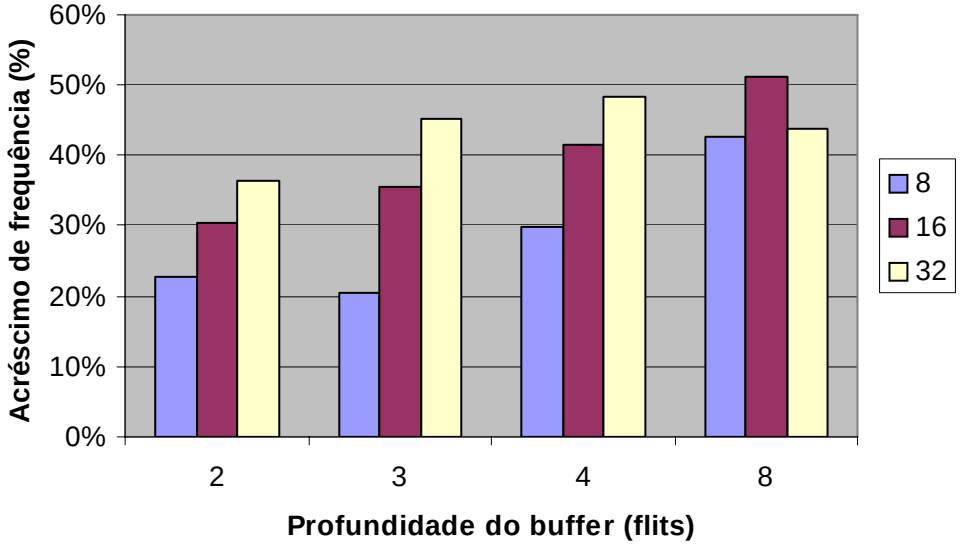

Figura 3.24: Acréscimo de freqüência do Tonga em comparação RASoC

Pode-se comparar o que significa para um SoC utilizar um roteador Tonga ou RASoC em termos de acréscimo de área. Para tanto, a área de cada roteador é comparada com relação a um microprocessador FemtoJava de 8 bits, desenvolvido na UFRGS (ITO, 1999).

A Tabela 3.5 mostra a redução do impacto da área ao ser utilizado o roteador Tonga no lugar do RASoC. O roteador RASoC é responsável por 32% da área ocupada em um sistema composto por processadores *FemtoJava*. Se for usado um roteador Tonga, o impacto na área total cai para 17%. Neste sistema, o roteador é responsável por um grande parcela da área final do sistema, tornando atraente o uso de um roteador de baixa área como o Tonga.

Tabela 3.5: Percentual de área ocupada pelos roteadores em um SoC

	Área roteador (canal de 8 bits e *buffer* de 2 *flits*)	Área FemtoJava de 8 bits	Área Total (LCs)	Percentual da área do roteador
RASoC	680	1418	2098	32%
Tonga	306	1418	1724	17%

3.6 Análise da Comunicação da Arquitetura Tonga

A análise de performance de comunicação da arquitetura Tonga é feita por meio da utilização de um simulador desenvolvido em C++, com uso do Borland C++ Builder 5, e apresentado em (KREUTZ, 2001). A análise objetiva compara o desempenho da arquitetura Tonga com o de um barramento e da arquitetura RASoC.

O simulador extrai o número de ciclos de relógio necessários para uma aplicação ser executada em uma arquitetura de comunicação, tanto um barramento como uma NoC. As principais classes do simulador são: *classe app*, que define o comportamento de comunicação da aplicação e executa esse comportamento sobre a arquitetura de comunicação através de simulação; *classe router*, definidora da arquitetura de comunicação de uma rede chaveada, contendo especificações para o roteador das mensagens, a política de arbitragem e controle de fluxo; *classe messages*, que define a estrutura de uma mensagem.

A definição do comportamento de comunicação da aplicação é feita através dos vetores *id* (número do core destino da mensagem) e *width* (tamanho da mensagem), onde cada posição corresponde a uma mensagem. O vetor *core_pos* define a posição de cada core na rede. Os índices do vetor correspondem ao número do núcleo e o conteúdo à sua posição. A ordem de execução das mensagens é definida pelo vetor *cr*. O conteúdo de cada posição corresponde ao número do core que recebe a mensagem. Todos esses vetores estão na classe app, em que é descrito o comportamento da comunicação.

Assim, para definir uma aplicação neste simulador, é necessário descrever nos vetores a origem e o destino de cada mensagem, o tamanho da mensagem e também determinar a posição dos núcleos na rede. É possível também usar um algoritmo do tipo Tabu para determinar a melhor posição dos núcleos.

Para comparar o desempenho da RASoC com o Tonga foi necessário descrever a arquitetura do roteador Tonga na classe *router*. Nossa descrição foi realizada com base na descrição já existente do roteador RASoC, considerando apenas três canais de comunicação internos.

A aplicação selecionada para a avaliação de performance foi uma FFT (*Fast Fourier Transform*), cujo algoritmo está descrito em (QUINN, 1994).

3.6.1 Descrição da Aplicação FFT

Na versão implementada da FFT, a aplicação é composta por dois tipos de núcleos. O primeiro tipo, chamado de *fft*, é composto pelos núcleos onde a FFT é processada em paralelo. O segundo tipo é chamado de *sync* e sua função é ordenar a FFT, repassando o seu resultado para um terminal de saída.

A aplicação possui um modelo de comunicação chamado *butterfly*, excelente comportamento de comunicação para testar uma arquitetura paralela, pois todos os roteadores são usados durante a sua execução.

O algoritmo é executado em quatro passos: (i) permutação dos pontos da FFT entre os núcleos; (ii) processamento dos pontos em paralelo, sem comunicação entre os núcleos; (iii) comunicação entre os núcleos, trocam resultados e processam os dados novamente; (iv) sincronização dos resultados para um terminal de saída. O primeiro passo pode ser eliminado se os pontos forem lidos pelos núcleos na ordem permutada.

Durante a primeira etapa do processo, em cada interação os núcleos *fft* devem trocar informação sobre todos os pontos com um outro núcleo *fft*. A troca é feita através dos canais de comunicação que possuem largura igual ao tamanho dos dados a serem trocados. No fim do processo com comunicação, os núcleos *fft* enviam o resultado final para o núcleo *sync*. A Figura 3.25 demonstra as etapas de uma FFT de 8 pontos.

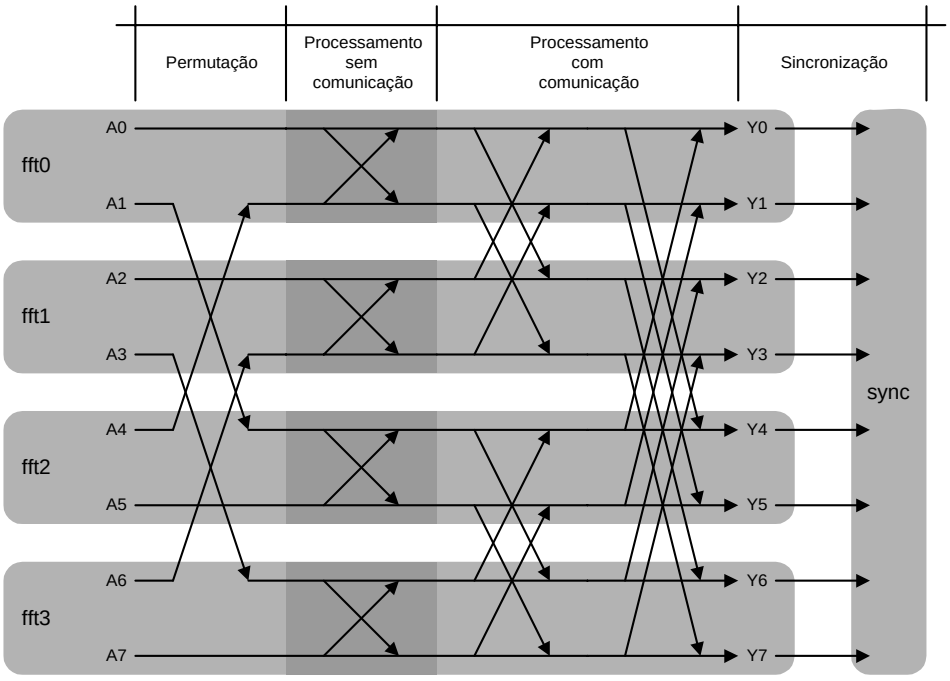

Figura 3.25: FFT de 8 pontos

3.6.2 Comparação de performance das arquiteturas

Para analisar a performance das redes, foram descritas duas aplicações FFT no simulador, uma de 8 e outra de 16 pontos. Elas foram executadas uma vez sobre três tipos de plataforma: um barramento monolítico, uma rede composta por RASoC e outra composta por roteadores Tonga.

A aplicação de 8 pontos é composta por 5 núcleos, sendo 4 núcleos *fft* e um *sync*, distribuídos em uma rede grelha 3x2. Já a FFT de 16 pontos possui 9 núcleos, 8 núcleos *fft* e 1 *sync*, distribuídos em uma rede grelha 3x3. Os roteadores foram configurados com uma profundidade de *buffer* de 4 posições e canal de 8 bits.

Os resultados obtidos são mostrados na Tabela 3.6. Para a FFT de 8 pontos, a diferença na redução da performance entre o Tonga e o barramento é pequena. Enquanto o Tonga tem uma performance 8% mais lenta que o desempenho da rede RASoC, o barramento apresenta uma redução de 13%. A diferença acentua-se em uma aplicação maior, como a FFT de 16 pontos, passando a rede Tonga a apresentar uma redução da performance de 30%, enquanto o barramento tem uma performance 53% pior em relação a uma rede RASoC (Figura 3.26).

Tabela 3.6: Comparação da performance em ciclos

	FFT de 8 pontos (ciclos)	FFT de 16 pontos (ciclos)
RASoC	45	75
Tonga	49	107
Barramento	52	160

O decréscimo da performance ocorre devido à multiplexação dos canais internos, o que provoca um aumento da contenção de pacotes. Se duas mensagens chegam ao mesmo tempo para os canais Norte e Oeste ou Sul e Leste, uma delas será transmitida enquanto a outra ficará bloqueada até o canal estar livre.

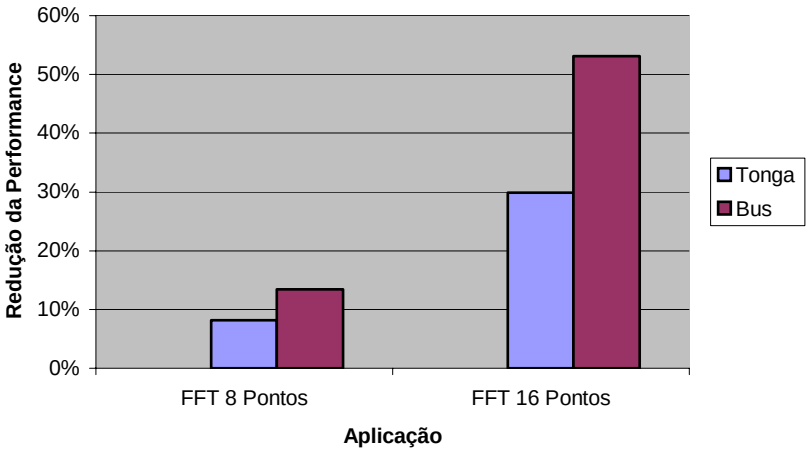

Figura 3.26: Redução da performance frente a rede RASoC

Levando em consideração apenas o número de ciclos, a rede Tonga terá uma performance pior que a RASoC. Mas, como mostrado nos resultados de síntese, a freqüência de operação do Tonga é maior que a da rede RASoC. Dessa forma, a diferença do tempo de execução da aplicação diminuirá, e o Tonga pode, inclusive, apresentar um tempo de execução menor.

Na Tabela 3.7 são mostrados os tempos de execução das aplicações. A freqüência utilizada corresponde a de um roteador de *buffer* de 4 *flits* e canal de 8 bits.

Tabela 3.7: Tempo de execução da aplicação

FFT 8 Pontos				
		Freqüência (MHz)	Número de ciclos	Tempo de execução (μs)
	RASoC	86	45	0,52
	Tonga	103	49	0,47
FFT 16 Pontos				
	RASoC	86	75	0,87
	Tonga	103	107	1,04

Os dados mostram que a rede Tonga executou a aplicação FFT de 8 pontos 10% menos rápido que a rede RASoC. Já para a aplicação de 16 pontos, a rede RASoC apresentou um desempenho melhor, e o Tonga foi 15% mais lento, como mostrado na Figura 3.27.

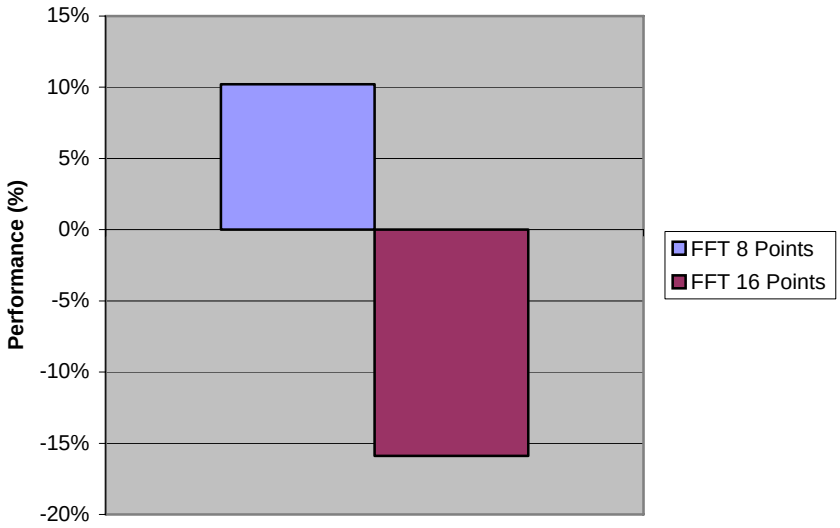

Figura 3.27: Performance da rede Tonga frente a rede RASoC

3.7 Considerações

Neste capítulo foi apresentado o roteador Tonga, baseado na rede SoCIN e no roteador RASoC, e que constitui um esforço no sentido de diminuir a área ocupada pela arquitetura de comunicação. O roteador Tonga manteve as características básicas da RASoC e teve como principal alteração a multiplexação dos canais de comunicação.

A multiplexação permitiu uma redução média de 50% de área e um aumento na freqüência de operação. A redução da área do roteador é bem significativa em um sistema de poucos núcleos ou núcleos pequenos, como no caso de um composto por microprocessadores *FemtoJava* (ITO, 1999), como demonstrado. O Tonga pode, portanto, tornar mais atraente o uso de NoCs para sistemas menores, onde a área do roteador torna a opção por NoC proibitiva.

Como desvantagem, o compartilhamento do canal de comunicação aumentará a contenção de pacotes na rede, reduzindo a performance. Entretanto, a redução de performance pode ser compensada pela freqüência de operação mais alta permitida pelo Tonga. Nas simulações realizadas foi demonstrado que para algumas aplicações o tempo de execução pode ser menor em uma rede Tonga, o que prova que o uso de um ou outro roteador depende das características de comunicação da aplicação e da área dos seus núcleos frente ao roteador.

Para análises futuras seria adequado efetuar novas comparações utilizando uma aplicação com um padrão de comunicação diferente da FFT. Novas comparações devem ser feitas utilizando redes maiores e condições de tráfego mais variada.

No próximo capítulo é apresentada uma rede heterogênea composta por ambos os roteadores. A idéia é aproveitar as características de comunicação similares deles e, através de um posicionamento automático, utilizar o roteador RASoC apenas em áreas da rede com maior tráfego de mensagens, mantendo o maior número de roteadores Tonga na rede. Procura-se, assim, obter uma rede rápida e que ocupe menos área.

4 EXPLORAÇÃO DE ESPAÇO DE PROJETO EM REDES-EM-CHIP (NOC) HETEROGÊNEA

Neste capítulo é apresentada uma NoC heterogênea como uma alternativa para reduzir o acréscimo na área total do sistema causada pelos roteadores que compõem uma arquitetura NoC. Utilizando uma combinação de roteadores com diferentes características de área e latência, pode-se alcançar uma rede com uma alta performance e um baixo impacto de área. Para tanto, foi utilizado um algoritmo de otimização que fornece uma arquitetura de rede compatível com as necessidades de performance, mantendo o custo de área tão pequeno quanto for possível.

A rede heterogênea desenvolvida é baseada na rede SoCIN (ZEFERINO, 2003a), e por esta razão chama-se SoCINhet. Ela é composta por duas arquiteturas de roteadores – RASoC (ZEFERINO, 2003a) e Tonga. Estes roteadores miram em diferentes necessidades de projetos. O RASoC alcança performances mais altas quando comparado com o Tonga, mas ocupa uma área consideravelmente maior.

Potencialmente, uma NoC heterogênea e otimizada pode ser desenvolvida utilizando estas duas arquiteturas de roteadores. Para tanto, foi utilizado o algoritmo *Tabu-search* para encontrar o compromisso procurado entre latência e área. Este algoritmo procura de forma exaustiva a melhor combinação de roteadores e de posicionamento dos núcleos para uma determinada aplicação.

No decorrer deste capítulo é apresentado um resumo das características arquiteturais dos roteadores Tonga e RASoC, o algoritmo de otimização proposto e os resultados correntes da rede SoCINhet. Os resultados foram obtidos usando aplicações-alvo cujo paralelismo nas comunicações torna interessante o uso de uma NoC como plataforma de comunicação.

4.1 Otimização Arquitetural e Redes-em-Chip (NoC) Heterogênea

Nesta seção, é proposto o uso de uma rede-em-chip heterogênea, composta pelos roteadores RASoC e Tonga. Estes roteadores possuem semelhantes características de comunicação, o que facilita sua integração em uma mesma rede, mas diferentes características de área e latência. Uma rede composta apenas por roteadores Tonga ocupará uma menor área, mas possivelmente terá um desempenho menor que uma utilizando RASoC.

A proposta da rede heterogênea é procurar um compromisso de área e performance compatível com a aplicação-alvo. Para tanto, torna-se necessário detectar pontos da rede onde ocorra um maior trafego na comunicação. Nestes pontos pode-se substituir o roteador Tonga por um RASoC, o que irá diminuir a latência da rede nestes pontos chaves. Um algoritmo do tipo *Tabu-search* é usado para encontrar automaticamente

uma combinação otimizada entre os dois tipos de roteadores, em relação à latência e restrição de área. Já que ambos roteadores são desenvolvidos para uma topologia regular, a rede heterogênea resultante será de tipo grelha. Na Figura 4.1, um exemplo de uma configuração 2x2 de tal NoC é demonstrada.

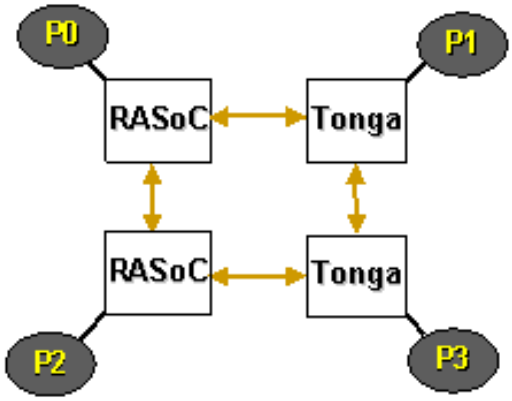

Figura 4.1: *Network-on-Chip* Heterogênea

4.1.1 Especificação dos roteadores

O roteador RASoC possui até cinco portas bidirecionais (Local, Norte, Sul, Oeste e Leste), as quais são compatíveis com topologias grelha-2D e torus. Possui um esquema de roteamento do tipo XY. O controle de fluxo é baseado no protocolo *handshake* (aperto de mão).

Cada porta possui um canal dedicado de comunicação. A arbitragem é distribuída, possuindo um árbrito exaustivo *round-robin* (RR) em cada canal de saída do roteador. A organização da memória é baseada em uma arquitetura de armazenamento de entrada do tipo FIFO.

A arquitetura do roteador Tonga é baseada no RASoC. A principal diferença entre eles recai sobre a maneira que os canais são roteados. Neste sentido o roteador Tonga pode ser visto como RASoC de baixo custo.

Outra diferença é quanto ao cabeçalho dos roteadores Tonga e RASoC, que possuem estruturas diferentes, como visto no Capítulo 2. Para permitir a integração dos dois roteadores na rede é necessário usar o endereçamento de pacote do roteador RASoC. Para tanto, basta substituir o bloco *Input Controller* do Tonga pelo usado no roteador RASoC. Esta mudança representará um acréscimo de área insignificante no Tonga, já que o bloco *Input Controller* representa em média apenas 1% da área total dos roteadores.

Para o Tonga, uma arquitetura multiplexada foi empregada como uma decisão de projeto a fim de reduzir a área. A Figura 4.2 mostra a idéia básica por traz da arquitetura Tonga. A idéia é usar a mesma arquitetura para gerenciar portas Norte e Oeste (nomeadas canal A), e outra para Sul e Oeste (chamada de canal B) e uma terceira para porta Local (Canal C). Os canais A e B incluem um bloco que seleciona uma das portas multiplexadas. A operação de multiplexação destes canais é baseada em um contador, o qual reserva uma fatia de tempo configurável para cada porta.

Ambos roteadores foram implementados em VHDL. A entidade de nível superior define os parâmetros de largura do canal de dados, a profundidade do FIFO e também o tempo de multiplexação para o Tonga.

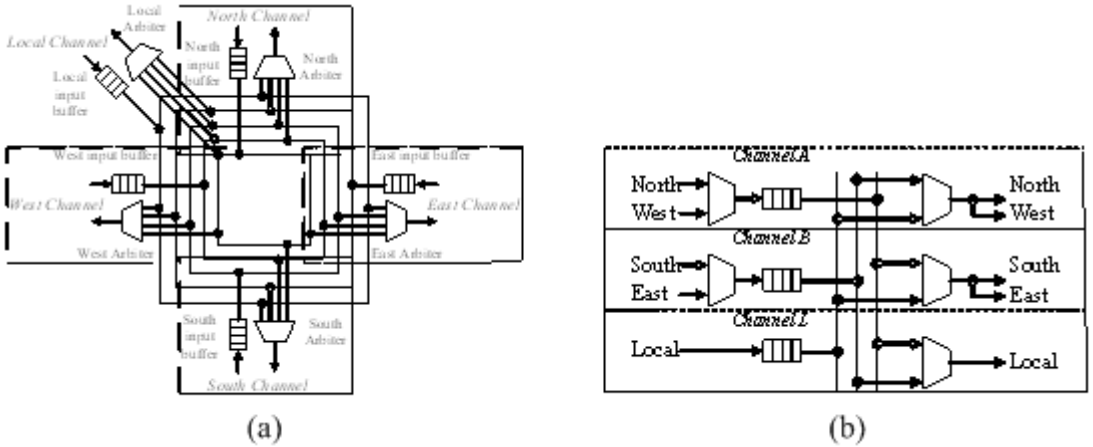

Figura 4.2: Arquitetura dos roteadores (a) RASoC e (b) Tonga.

A Tabela 4.1 mostra a área obtida a partir de ambas arquiteturas de roteadores com *buffers* de quatro posições. Pode ser observado que aproximadamente 40% da área economizada foi alcançada por Tonga em comparação com o RASoC. Os modelos foram sintetizados para a biblioteca de células desenvolvida pela AMS para sua tecnologia 0.35μ CMOS (AMS, 2001) utilizando-se o *software* Leonardo Spectrum 3.

Tabela 4.1: Área dos Roteadores

		Channel Width		
		8	16	32
RASOC	Area (μm2)	157266	248539	431176
	Freqüência (MHz)	111	100	82
TONGA	Área (μm2)	100937	154809	263063
	Freqüência (MHz)	120	143	133

Além da economia de área, o tonga apresentou um aumento da freqüência em relação a RASoC. Este comportamento da freqüência pode ser usado em uma rede heterogênea. A NoC é um sistema desenvolvido para ser localmente síncrono mas globalmente assíncrono (GALS), permitindo, assim, que os núcleos operem em diferentes freqüências de operação. Em uma rede SoCINhet pode-se ter os roteadores Tonga e RASoC operando em diferentes freqüências na rede. Para tanto, deve-se observar que o protocolo *handshake* utilizado é uma variação síncrona do *handshake* assíncrono e talvez isto não seja possível sem uma alteração no controle de fluxo.

4.1.2 Estratégias de Otimização da Rede SoCINhet

A otimização da rede heterogênea é realizada por meio da melhor combinação de roteadores e do posicionamento otimizado dos núcleos. Assim, são assumidas duas estratégias de otimização pelo algoritmo *Tabu-search.*

Na primeira estratégia é procurada a menor latência possível para uma determinada aplicação, considerando o menor custo em área da rede. Assim, o algoritmo posiciona os núcleos da melhor maneira possível em uma rede formada apenas por roteadores Tonga, de forma a obter a menor latência.

Na implementação da primeira otimização têm-se S recursos equivalentes ao posicionamento dos núcleos: S = {C0, C1, ..., Cn}; onde S [8] = {C4} significa que o núcleo C4 está ligado à porta local do roteador "8". Um movimento no algoritmo Tabu significa troca de núcleo, o qual deixa uma nova solução no espaço da pesquisa. Os movimentos (mudanças) são gerados aleatoriamente. Para cada solução todas as comunicações são executadas, com avaliação do resultado de performance encontrado.

Se a latência necessária não for encontrada, o algoritmo passa a executar uma segunda estratégia, com aplicação de uma pesquisa bidimensional. No momento em que os núcleos de processamento são movidos para diferentes portas locais da rede, os tipos de arquiteturas de roteador são simultaneamente modificados. Para este propósito, dois recursos são definidos para o espaço da pesquisa Tabu: S1 e S2.

```
Tabu_Search(resources S)  {
   select an initial solution:
       y ∈ Y e y* = y; k= 0; T = ∅
2  if (S(y)–T) == ∅
       go to 4;
   else
       t = t + 1;
   select best Y' = OPTIMUM(s(y): s ∈ S(y)-T);
   y = Y';
   if k(y) < k(y*)  // y* → best solution
      y* = y;
   if t > nt
4     stop;
   else
      update T;
   go to 2;
}
```

Figura 4.3: Pseudo-código para o algoritmo *Tabu-search*

O recurso S1 significa posicionamento dos núcleos (como na primeira estratégia) e S2 indica o tipo de cada roteador na rede. Se tivermos um vetor S2 = {T_{Tonga}, T_{RASoC},...,T_{Tonga}}, ele nos indicará que o roteador "1" é do tipo RASoC, pois S2 [1] = {T_{RASoC}}. Agora, o Tabu tem dois movimentos (mudanças) no espaço da pesquisa: troca de núcleos (para S1) e recolocação do tipo de roteador, como mostrado na Figura 4.3.

A rede, inicialmente configurada apenas com roteadores Tonga, tem os roteadores progressivamente substituídos pelos de arquitetura RASoC, com a descoberta de novas soluções.

4.2 Análise da Comunicação na Rede SoCINhet

A fim de obter dados experimentais da performance da rede SoCINhet, foi utilizada a descrição do comportamento de comunicação das seguintes aplicações: FFT de 16 pontos, uma aplicação de segmentação de imagem e uma aplicação randômica. A aplicação FFT 16 pontos é explicada no capítulo anterior e em (QUINN, 1994).

As redes foram descridas e simuladas em C++, em uma versão do simulador apresentado em (KREUTZ, 2001). Nesta versão do simulador, os modelos são compatíveis com a abordagem *System C* (GRÖTKER, 2002), já que eles foram descritos seguindo os princípios de design baseado em interface (ROWSON, 1997). A tradução de nossos modelos para *System C* é direta e imediata.

As aplicações serão executadas e otimizadas usando o algoritmo *Tabu-search*, com uma combinação de roteadores mais otimizados para cada aplicação. A aplicação de

segmentação de imagem, chamada de *SegImag* (BORIN, 2005) (MARCON, 2005), é mostrada a seguir, assim como os resultados experimentais obtidos com as aplicações.

4.2.1 Descrição da aplicação *SegImag*

Uma das aplicações utilizadas como alvo é a de segmentação de imagem para reconhecimento de objetos, batizada de *SegImag* (MARCON, 2005). Esta aplicação tem o objetivo de acelerar o processo de identificação do número de objetos de uma imagem. Para tanto, a imagem original deve ser particionada em segmentos, onde cada segmento é tratado concorrentemente por um processador auxiliar (PA). Além da PA, a *SegImag* contém dois outros elementos de processamento (*processing elements* ou PEs) que são: uma memória externa (ME) e um processador central (PC). As comunicações da aplicação para um número qualquer de PAs estão ilustradas na Figura 4.4.

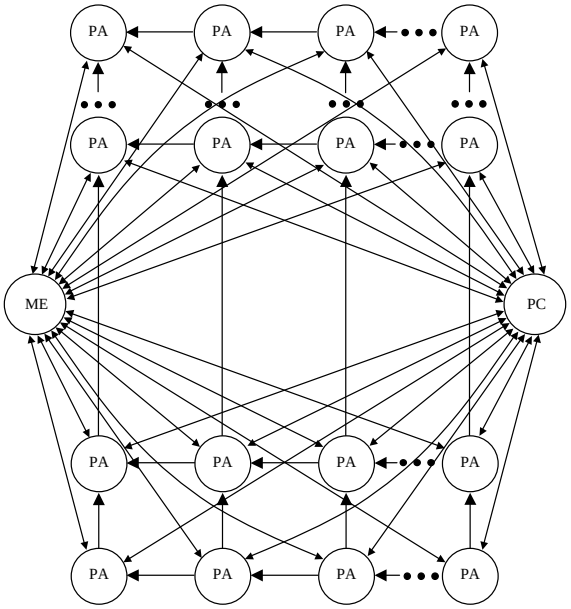

Figura 4.4: Grafo de comunicação genérico para o problema de segmentação de imagens.

As características da aplicação *SegImag* permitem que o número de PAs seja parametrizável. *SegImag* é uma aplicação onde podem ser comparadas implementações com maior computação ou comunicação, ou seja, pode ser analisada a relação entre a computação de cada PA com o número de PAs e o *overhead* causado pela comunicação entre eles.

A imagem que será segmentada encontra-se inicialmente na ME. Ela é transferida para os PAs (um segmento para cada PA), que contabilizam em paralelo a quantidade de objetos de seu segmento. Na etapa inicial, cada PA calcula os objetos de seu segmento e atribui para os pixels da imagem uma numeração, gerando uma associação de pixel com objeto. A seguir, cada PA realiza uma primeira comunicação com seu vizinho esquerdo, enviando a coluna esquerda (já numerada) para este verificar se existem objetos adjacentes nas fronteiras. O mesmo é feito com o PA vizinho acima. Esta etapa é necessária para que os objetos vizinhos não sejam contabilizados mais de uma vez. Em seguida, todos os PAs enviam uma mensagem para o PC com o número de objetos encontrados e os pares de segmentos adjacentes. O PC pega todas as numerações individuais, gera uma numeração global, compacta essa numeração retirando as redundâncias geradas pelos segmentos adjacentes e devolve para os PAs a

nova numeração. Os PAs substituem a nova numeração no seu segmento de imagem e retransmitem a imagem contabilizada para a ME.

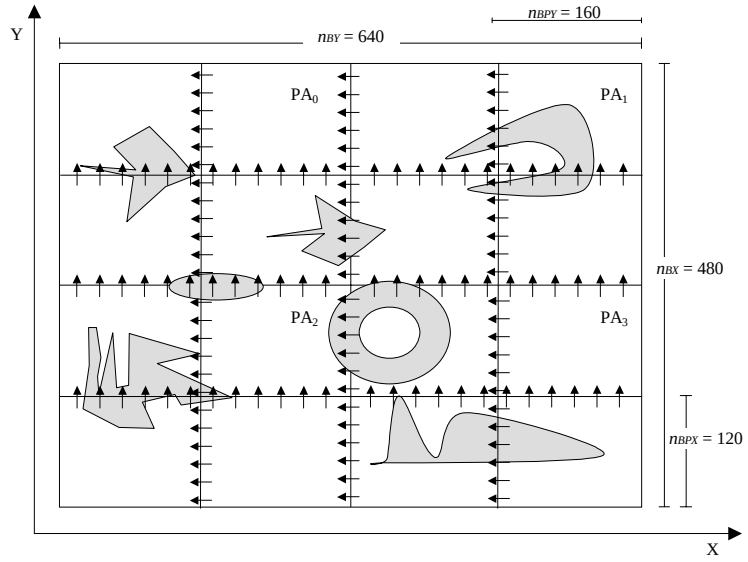

Figura 4.5: Exemplo de imagem com segmentação e os correspondentes PAs para cada segmento.

A aplicação *SegImag* será exemplificada aqui com uma imagem de 640x480, uma taxa de processamento (tp) de 15 quadros/seg, e 16 PAs (npa). Assim, o número de bytes em X (nbx) é 640 e o número de bytes em Y (nby) é 480. Como existem 4 PAs para cada dimensão, o número de bytes que cada PA deve processar em X (nbpx) é 160 e o número de bytes que cada PA deve processar em Y (nbpy) é 120. As características da imagem e cada segmento são apresentadas na Figura 4.5. O tamanho das mensagens em cada comunicação é calculado na Tabela 4.2.

Tabela 4.2: Aplicação de segmentação de imagens, considerando uma imagem de 640x480 bytes e apenas um quadro.

PA↔ ME		Número de bytes total da imagem (nbi) = 640x480 = 307200 bytes
		Número de bytes de cada segmento (nbs) = nbi / npe = 307200 / 16 = 19200 bytes
PA ↔ PA	Em X	Número de bytes da imagem em Y (nby) = 480 bytes
		Número PAs em Y (npay) = 4
		Número de bytes na fronteira de cada PA para Y (nbpy) = nby / npAy = 480 / 4= 120 bytes
	Em Y	Número de bytes da imagem em X (nbx) = 640 bytes
		Número PAs em X (npax) = 4
		Número de bytes na fronteira de cada PA para X (nbpx) = nbx / npAx = 640 / 4 = 160 bytes
PA ↔ PC		Número de bytes de controle de vizinhança (ncv) = 128 bytes (estimado com base no tamanho dos objetos em imagens típicas - equivale a 64 pares)

Assim, no exemplo usado, a aplicação de imagem é composta pelos 16 núcleos PAs, um núcleo onde a memória está armazenada e um processador de controle responsável pela reunião dos segmentos de imagens. As comunicações em paralelo indicam que uma boa forma de interligar estes núcleos pode ser através de uma rede-em-chip, que permitirá a troca de dados em paralelo entre os núcleos de processamento auxiliar.

4.2.2 Análise dos resultados da rede SoCINhet

As Figuras 4.6 e 4.7 mostram respectivamente, os resultados para latência média e de área (considerando largura de canal de 32 bit) para a aplicação testada. Nestas figuras, "RASoC" e "Tonga" identificam as redes otimizadas apenas através do posicionamento dos núcleos, enquanto "SoCINhet" identifica a rede heterogênea encontrada pelo algoritmo *Tabu-search* trocando o tipo de roteador e a posição dos núcleos. A execução da aplicação foi realizada apenas uma vez, com os roteadores com seus buffers vazios na condição inicial.

Como pode ser observado, um compromisso entre latência e área foi encontrado para a aplicação *SegImag*, já que a latência é mais próxima (aproximadamente 10%) de uma rede RASoC (o qual é a solução mais rápida possível) e, ao mesmo tempo, com uma redução de área de aproximadamente 20%, quando comparada com a rede RASoC. Neste caso, o algoritmo encontrou uma rede heterogênea composta por 10 roteadores de tipo RASoC e 10 roteadores de tipo Tonga. Para obter tal resultado o algoritmo Tabu efetuou 43 interações, cada um com 10 soluções.

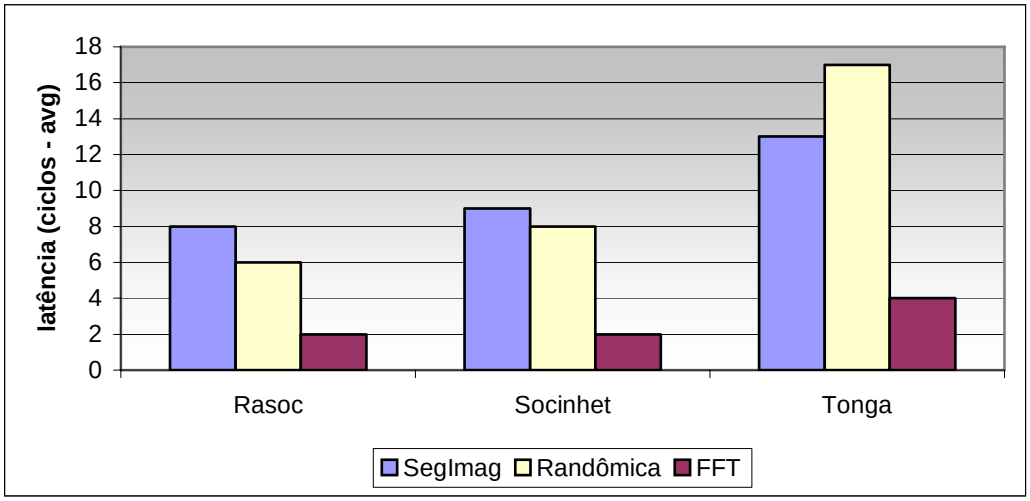

Figura 4.6: Latência das redes

Um resultado semelhante foi obtido com a aplicação randômica, distribuída em uma rede 4x4. A latência média apresentou uma redução próxima a 50% se comparada com a rede Tonga, enquanto a área é 25% menor do que a rede RASoC.

Para a aplicação FFT, uma latência mais baixa pode ser encontrada pela Rede Tonga, em comparação com a RASoC. Embora isto pudesse ser um resultado inesperado, pode ser explicado pela otimização de colocação dos núcleos. Provavelmente para rede Tonga, o *Tabu-search* pode encontrar a melhor solução. O mesmo ocorreu com a SoCINhet. Isto ressalta a relevância da colocação dos núcleos na performance. Quanto a área, a otimização feita na SoCINhet resultou em uma rede composta de 5 roteadores de tipo RASoC e 4 do tipo Tonga, o qual traz uma economia de área de 17% em comparação com a rede RASoC.

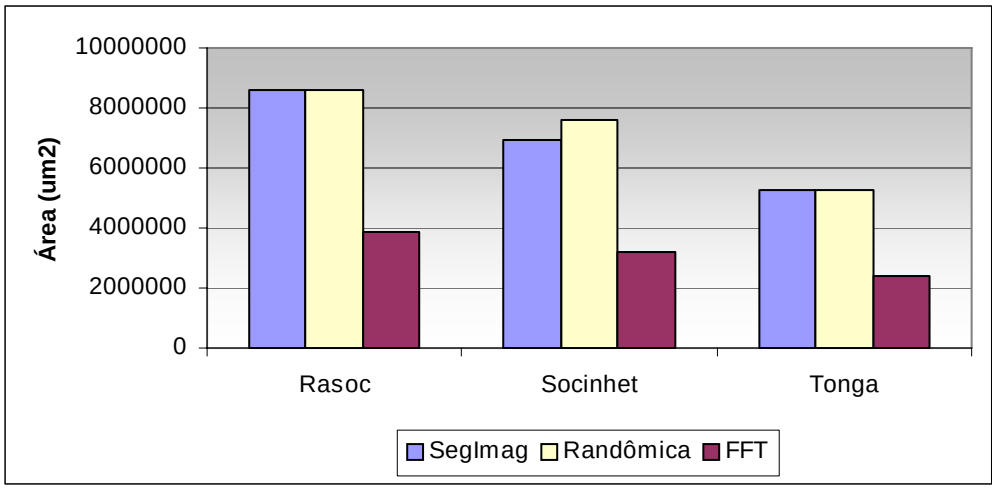

Figura 4.7: Área ocupada pelas redes.

A Figura 4.8 descreve a relação entre latência e área encontrada pelo algoritmo Tabu para as aplicações testadas. O acréscimo de “0%” de área significa um NoC configurada somente com roteadores Tonga, ou seja, é a rede com menor área e maior latência. A medida que ocorre um acréscimo de área, ocorre uma diminuição da latência média. Isso significa que com a inclusão de roteadores RASoC, posicionados estrategicamente na rede, o desempenho da rede melhora Os pontos com maior acréscimo de área e menor latência representam uma rede apenas com roteadores RASoC. Como pode ser observado, quando NoCs heterogêneas são consideradas, uma relação otimizada de latência e área pode ser encontrada.

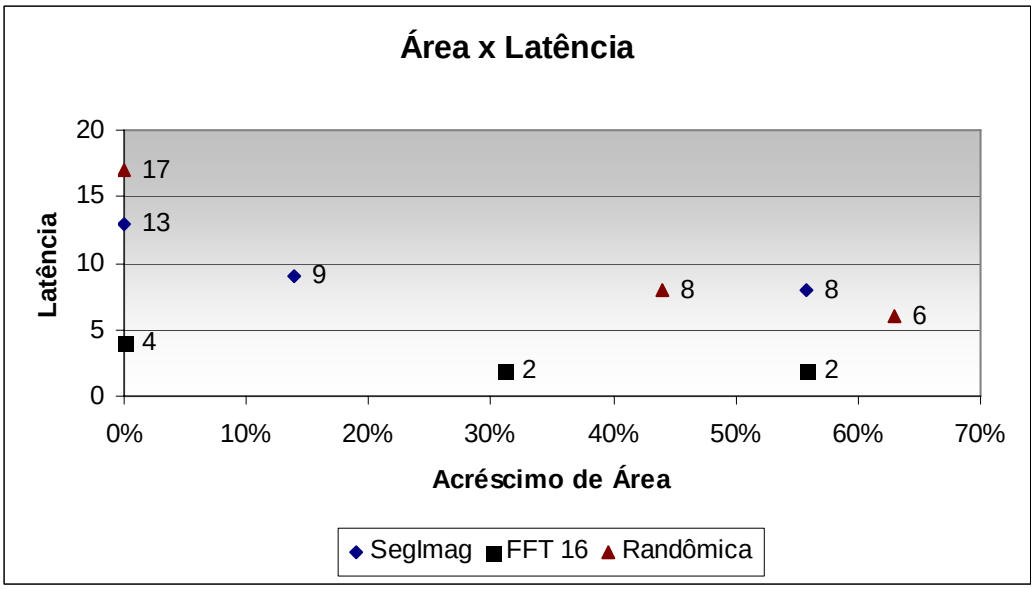

Figura 4.8: Relação Área x Latência.

Na distribuição dos tipos de roteadores na *SegImag*, nota-se que os núcleos processador central e memória foram conectados a roteadores do tipo RASoC, como mostrado na Figura 4.9. Estes núcleos são centralizadores de informação, onde todos os núcleos irão buscar informação (memória) ou enviar e buscar resultados (processador central). Assim, pode-se ver que o algoritmo detectou na troca de roteadores os pontos de maior tráfego e utilizou nestes pontos o roteador que oferece um maior paralelismo na comunicação.

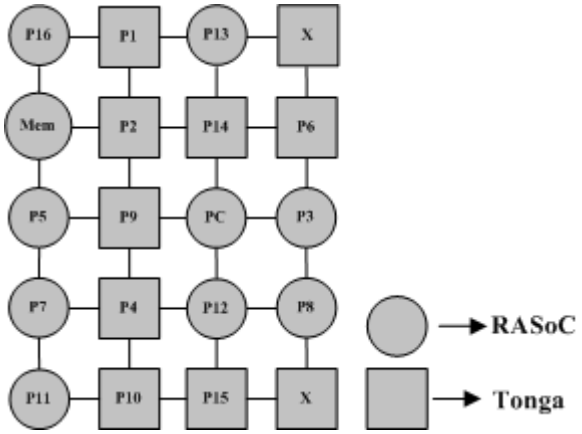

Figura 4.9: Aplicação *SegImag* distribuída em uma SoCINhet.

Foram testadas também configurações de redes maiores. Para este propósito, foram usadas topologias de rede de dimensão 5x5, 8x8 e 10x10. Nestas redes foi executada a aplicação de imagem, já que a sua estrutura é totalmente escalonável. A imagem pode ser dividida entre mais elementos processados, cada um sendo responsável por uma porção menor da imagem. Já que está sendo utilizado um simulador em C++, até mesmo quando uma topologia 10x10 for executada por um algoritmo Tabu de 40 interações, o algoritmo de otimização completo é executado em aproximadamente 10 segundos em uma máquina Pentium IV com 512 MB de RAM.

Os resultados para redes maiores são mostrados na Tabela 4.3. Nestas tabelas, a latência é dada em número médio de ciclos, área em um² e R/T é o número de cada tipo de roteador (R – RASoC, T - Tonga) usado em cada rede.

Tabela 4.3: Resultados para diferentes *SegImag*

		Socinhet	Tonga	Rasoc
NoC 5x5	Latência Média	12	13	10
	Área	8,1	6,5	10,7
	R/T	11/13	0/25	25/0
NoC 8x8	Latência Média	30	33	19
	Área	18,5	16,8	27,5
	R/T	10/54	0/64	64/0
NoC 10x10	Latência Média	21	23	17
	Área	31,6	26,3	43,1
	R/T	32/68	0/100	100/0

Como pode ser observado na tabela, o compromisso entre área e performance é mantido até mesmo quando redes maiores são usadas. Pode-se concluir que existe um espaço a ser explorado na direção das redes heterogêneas, já que elas representam uma oportunidade de priorizar uma restrição e, ao mesmo tempo, reduzir área. A economia da área obtida para redes 5x5, 8x8 e 10x10 são respectivamente 20%, 35% e 27% em comparação com um NOC composto somente por roteadores RASoC.

4.3 Considerações

A rede SoCINhet apresentada neste capítulo foi um esforço desenvolvido na exploração do espaço de projeto em redes-em-chip. Aplicando dois roteadores com protocolos de comunicação compatíveis, mas com diferentes características de custo de área e performance, procurou-se gerar uma rede que tivesse um compromisso entre estes dois fatores.

O roteador Tonga foi desenvolvido a partir da arquitetura do roteador RASoC. Através da multiplexação de canais obteve-se uma redução média de 50% de área e um aumento de 30% na freqüência de operação. Mas este ganho em área teve seu preço na diminuição do paralelismo do roteador e conseqüente aumento do bloqueio dos pacotes na rede.

A rede heterogênea proposta parte do principio de que uma rede formada exclusivamente por roteadores Tonga atenderá aos requisitos básicos de performance de uma aplicação. Desta forma, a solução para a implementação da comunicação desta aplicação inicialmente será aquela com menor custo em área. Se a rede não atender aos requisitos de performance, um algoritmo poderá localizar os pontos de maior tráfego da rede. Nestes pontos o algoritmo substituirá progressivamente os roteadores Tonga por RASoC até atingir a performance exigida.

O algoritmo usado para efetuar esta busca foi o *Tabu-search*. Ele faz uma busca em duas dimensões, de forma exaustiva, substituindo o roteador e reposicionando os núcleos para, em seguida, avaliar o resultado desta troca. Como é um algoritmo de busca exaustiva, o número de trocas a ser executada é determinante para o resultado a ser obtido.

Na avaliação da rede heterogênea, foi utilizada três aplicações: FFT 16 pontos, SegImag e uma aplicação randômica. Os resultados mostram um bom compromisso entre área e performance. Para um acréscimo de área, haverá também uma melhora da performance da aplicação, como esperado. Mas este acréscimo será otimizado na medida exigida pela aplicação-alvo.

A atividade de pesquisa apresentada neste capítulo gerou a base de um trabalho aceito no International Symposium on Circuits and Systems no ano de 2005 (CARDOZO, 2005). Neste artigo é apresentado todo o conceito da rede SoCINhet e os resultados experimentais obtidos.

5 CONCLUSÃO

Este texto apresentou um conjunto de estudos e atividades de pesquisa centrados na comunicação em sistemas integrados em um único chip. No capítulo inicial foi realizada uma revisão bibliográfica sobre os conceitos que fundamentam as redes-em-chip, consideradas a melhor alternativa para os sistemas integrados futuros. Para embasar esta afirmação, foram mostradas as limitações das abordagens das duas alternativas mais utilizadas atualmente na comunicação de sistemas *intrachip*: barramentos e canais ponto-a-ponto dedicados. No estudo realizado sobre redes-em-chip foram apresentados os principais tipos de mecanismos de comunicação e sua classificação. Explorou-se também as principais arquiteturas de redes-em-chip atuais, analisando as opções de desenvolvimento utilizadas na criação destas redes.

Uma rede de comunicação *intrachip* apresenta uma série de vantagens, tais como freqüência de operação do sistema inalterável com o aumento do número de núcleos, alta reusabilidade e paralelismo nas comunicações. Apesar destas vantagens, há um maior custo em área e aumento da latência na comunicação. A pesquisa desenvolvida nesta dissertação procurou diminuir o acréscimo de área da rede no sistema, possibilitando o uso de NoCs em sistemas menores. Nestes sistemas a área dos roteadores será responsável por grande parte da área total do sistema, podendo inviabilizar seu uso.

Para diminuir o impacto destas redes no custo do sistema foi apresentado inicialmente uma alternativa arquitetural de roteador de baixo custo chamado Tonga. Baseado na rede SoCIN e no roteador RASoC, o roteador Tonga manteve as características básicas da RASoC e teve como principal alteração a multiplexação dos canais de comunicação. Como conseqüência foi possível reduzir em média 50% da área em relação a RASoC, além de um aumento médio de freqüência de 40%. A redução da área do roteador é bem significativa em um sistema de poucos núcleos ou núcleos pequenos, como no caso de um composto por microprocessadores *FemtoJava* (ITO, 1999), como demonstrado. O Tonga pode, portanto, tornar mais atraente o uso de NoCs para sistemas menores.

A multiplexacão dos canais de comunicação apresenta uma desvantagem. Com o compartilhamento do canal de comunicação ocorrerá um aumento da contenção de pacotes na rede, reduzindo a performance da rede. A análise de performance das arquiteturas de comunicação entre os roteadores Tonga, RASoC e barramento foi feita em um ambiente de simulação desenvolvido em C++ (KREUTZ, 2001) e com a descrição do comportamento de comunicação de algumas aplicações-alvo. Os resultados mostraram que a rede Tonga apresenta uma performance superior ao barramento, mas inferior a uma rede composta por RASoC.

Entretanto, a redução de performance pode ser compensada pela freqüência de operação mais alta permitida pelo Tonga. Considerando o tempo de execução, em vez do numero de ciclos de relógio na avaliação da performance, notou-se que para algumas aplicações o tempo de execução pode ser menor em uma rede Tonga. Assim, o uso de um ou outro roteador dependerá diretamente das características de comunicação da aplicação e da área dos seus núcleos frente ao roteador.

Na seqüência do trabalho, de posse destes dois roteadores com protocolos de comunicação semelhantes e com diferentes características de custo de área e performance, foi desenvolvida uma rede que torna possível explorar o compromisso entre estas características. Assim, com o propósito de explorar este espaço de projeto, foi proposta uma rede heterogênea chamada SoCINhet.

O propósito desta rede é atender a performance exigida pela aplicação com o mínimo custo em área. A rede com menor custo em área é composta apenas por roteadores Tonga. Caso ela não atenda os requisitos de comunicação da aplicação, o algoritmo *Tabu-search* executará uma procura em duas dimensões, substituindo o tipo de roteador e reposicionando os núcleos na rede. O resultado final será uma rede com os dois tipos de roteadores otimizada para uma aplicação-alvo.

Na avaliação da rede heterogênea, foram utilizadas três aplicações: FFT 16 pontos, *SegImag* e uma aplicação randômica. Os resultados mostraram um bom compromisso entre área e performance. Para um acréscimo de área, haverá também uma melhora da performance da aplicação, como esperado. Mas este acréscimo será otimizado na medida exigida pela aplicação-alvo.

Diversos trabalhos futuros podem vir a ser desenvolvidos a partir desta dissertação. Entre eles, a validação da SoCINhet em VHDL e o aprofundamento das análises de desempenho e a avaliação de potência das redes. Quando ao aprofundamento das análises de desempenho, observa-se que nesta dissertação não foi considerado intervalo entre pacotes gerados. A rede trabalhou com carga de 100% sem levar em conta o processamento dos dados pelos núcleos. Além disso, recomenda-se uma simulação em que a aplicação seja executada mais de uma vez para analise da latência média. Isso seria interessante, pois permitiria observar o comportamento da rede com os roteadores em uma situação diferente da inicial, com os *buffers* vazios.

Ao longo do desenvolvimento deste trabalho diversos artigos foram submetidos e aceitos para publicação. Em (CARDOZO, 2004) é apresentado o roteador Tonga e mostrado os resultados obtidos. A rede SoCINhet gerou um trabalho aceito no *International Symposium on Circuits and Systems* no ano de 2005 (CARDOZO, 2005) onde é apresentado todo o conceito da rede SoCINhet e os resultados experimentais obtidos. Além destes, em (KREUTZ, 2005) o roteador Tonga é aplicado em uma análise sobre o compromisso entre performance e potência.

REFERÊNCIAS

ALTERA. **APEXII Programmable Logic Family Data Sheet**. San Jose, 2002.

ALTERA **Quartus II Software Release Notes – Quartus II version 4.2 Service Pack 1**. San Jose, 2005.

AMS. **0.35 µm CMOS Standard Cell Databook**. Unterpremstätten: Austria Micro System, 2001. 338p.

BECK FILHO, A. C. S.; CARRO, L. **Comparação entre Diferentes Topologias de Comunicação entre SOC**. 2003. 62 f. Relatório de Pesquisa (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

BENINI, L.; DE MICHELI, G. Powering Networks on Chips. In: INTERNATIONAL SYMPOSIUM ON SYSTEM SYNTHESIS, 14., Montreal, 2002. **Proceedings…** Los Alamitos: IEEE Computer Society, 2002. p. 33-38.

BENINI, L.; DE MICHELI, G. Networks on Chips: A New SoC Paradigm. **IEEE Computer**, [S.l.], v.35, n.1, p. 70-78, Jan. 2002.

BERGAMASCHI, R. A.; LEE, W. R. Designing Systems-on-Chip Using Cores. In: DESIGN AUTOMATION CONFERENCE, 37., 2000, Los Angeles. **Proceedings**... New York: ACM Press, 2000. 819p. p.420-425.

BERGAMASCHI, R. A. et al. Automating the Design of SoCs Using Cores. **IEEE Design & Test of Computers**, [S.l.], v.18, n.5, p.32-45, Sept.-Oct.2001.

BERGAMASCHI, R. A. The A to Z of SoCs. In: ESCOLA DE MICROELETRÔNICA, 4., 2002, Florianópolis. **Anais...** [S.l.:s.n.], 2002.

BORIN, A. **Segmentação de Imagens em Paralelo para Uso em um Sistema com Múltiplos Elementos de Processamento**. Disponível em: <http://warrior.eletro.ufrgs.br/producao/outros/Relatorios_tecnicos/segimage.pdf>. Acesso em: maio 2005.

CARDOZO, R. S. et al. Tonga: A Low Cost Router for NoCs. In: WORKSHOP IBERCHIP, 10., 2004, Cartagena de Indias. **Memórias**. [S.l.: s.n.], 2004.

CARDOZO, R. S. et al. Design Space Exploration on Heterogeneous Network-on-Chip. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005, Kobe. **Proceedings…** Los Alamitos: IEEE Computer Society, 2005.

DALLY, W. J.; TOWLES, B. Route Packets, Not Wires: On-Chip Interconnection Networks. In: DESIGN AUTOMATION CONFERENCE, 38., LasVegas, 2001. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2001. p.684-689.

DUATO, J. et al. **Interconnection Networks:** An Engineering Approach. Los Alamitos, CA: IEEE Computer Society Press, 1997. p. 515.

GUERRIER, P.; GREINER, A. A Scalable Architecure for System-on-Chip Interconnections. In: SOPHIA-ANTIPOLIS MICRO-ELECTRONICS CONFERENCE, 1999, France. **Proceedings…** Sophia Antipolis : [s.n.], 1999. p.90-93.

GUERRIER, P.; GREINER, A. A generic architecture for on-chip packet-switched interconnections. In: DESIGN AUTOMATION AND TEST IN EUROPEAN CONFERENCE, 2000, Paris. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2000. p.250-256.

GLASS, C.; NI, L. The Turn Model for Adaptative Routing. **Journal of the Association for Computing Machinery**, [S.l.], v. 41, n.5, p. 874-902, Sept. 1994.

GRÖTKER, T. et al. **System Design with SystemC**. [S.l.]: Kluwer Academic, 2002.

HEMAMI A. et al. Lowering power consumption in clock by using globally asynchronous locally synchronous design style. In: DESIGN AUTOMATION CONFERENCE, 36., New Orleans, 1999. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1999. p.873-878.

ITO, S. A. et al. Designing a Java Microcontroller to Specific Applications. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 12., 1999. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 1999. p.12–15.

INTERNATIONAL SEMATECH. **International Technology Roadmap for Semiconductors**. 2003. Disponível em: <http://public.itrs.net>. Acesso em: 2004.

JANTSCH, A. et al. Networks On Chip. In: EUROPEAN SOLID STATE CIRCUIT CONFERENCE, 2001. **Proceedings...** [S.l.: s.n.], 2001.

JANTSCH, A.; TENHUNEN, H. **Network On Chip.** [S.l.]: Kluwer Academic, 2003.

KARIM, A. N. F.; DEY, R. R. S. On-chip Communication Architecture for OC-768 Network Processors. In: DESIGN AUTOMATION CONFERENCE, 38., LasVegas, 2001. **Proceedings...** New York: ACM Press, 2001.

KARIM, A. N. F.; NGUYEN, A.; DEY, R. R. S. An Interconnect Architecture for Networking Systems on Chips. **IEEE Micro,** [S.l.], v.22, n.5, p.36-45, Sept.-Oct. 2002.

KREUTZ, M. E. et al. Communication Architectures for System-on-Chip. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, 14., 2001, Pirinópolis - Brasil. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.14-19.

KREUTZ, M. E. et al. Design Space Exploration Comparing Homogeneous and Heterogeneous Network-on-Chip Architectures. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, 18., 2005, Florianópolis - Brasil. **Proceedings...** Los Alamitos: IEEE Computer Society, 2005.

KUMAR, S. et al. A Network on Chip Architecture and Design Methodology. In: INT. SYMPOSIUM ON VERY LARGE INTEGRATION SCALE, 2002, Pittsburg. **Proceedings...** Los Alamitos: IEEE Computer Society, 2002. p.105-112.

LIANG, J. et al. aSOC: A Scalable, Single-Chip Communication Architecture. In: INTERNATIONAL CONFERENCE ON PARALLEL ARCHITECTURES AND COMPILATION TECHNIQUES, 2000, Philadelphia. **Proceedings...** Los Alamitos: IEEE Computer Society, 2000.

MARCON, C. A. **Particionamento e Mapeamento de Aplicações em Infra-Estruturas de Comunicação Intrachip.** 2005. Proposta de Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MELLO, A.V. ; MÖLLER, L.H. **Arquitetura Multiprocesada em SoCs:** estudos diferentes topologias de conexão. 2003. 120p. Trabalho de Conclusão, FACIN-PUCRS. Disponível em: <http://www.inf.pucrs.br/~moraes/papers/tc_multiproc.pdf>. Acesso em: 2004.

MORAES, F. G. et al. A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping. In: INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, 2003, Darmstadt. **Proceedings...** [S.1.: s.n.], 2003.

MORAES, F. G. et al. NOCGEN - Uma Ferramenta para Geração de Redes Intra-Chip Baseada na Infra-Estrutura HERMES. In: WORKSHOP IBERCHIP, 10., 2004, Cartagena. **Proceedings…**[S.l.:s.n.], 2004. v. 1, p. 210-216.

NI, L. M.; McKINLEY, P. K. A Survey of Wormhole Routing Techniques in Direct Networks. **IEEE Computer Magazine,** [S.l.], v.26, n.2, p.62-76, Feb. 1993.

OST, L.; MORAES, F. G. **Redes Intra-Chip Parametrizáveis com Interface Padrão para Síntese em Hardware.** 2004. Dissertação (Mestrado em Ciência da Computação) - Faculdade de Informática – PUCRS, Porto Alegre.

QUINN, M. J. **Parallel Computing**: Theory and Practice. New York: Mc-Graw Hill, 1994. 446p.

ROWSON, J.A.; SANGIOVANNI-VINCENTELLI, A. Interface-based Design. In: DESIGN AUTOMATION CONFERENCE, 34., Anaheim, 1997. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1997. p.178 - 183.

SAASTAMOINEN, I., et al. Interconnect IP Node for Future System-on-Chip Designs. In: WORKSHOP ON ELECTRONIC DESIGN TEST AND APPLICATION, DELTA, 2002. **Proceedings...** [S.l.: s.n.], 2002. p. 113-116.

SANTOS, F. G. M. E. et al. Distribuded Arbiters for Network-on-Chip. In: MICROELECTRONICS SEMINAR, 2003, Novo Hamburgo. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 2003. p.169-172.

SGROI, M. et al. Addressing the System-on-Chip Interconnect Woes Through Communication-Based Design. In: DESIGN AUTOMATION CONFERENCE, DAC, 2001. **Proceedings…** [S.l.: s.n.], 2001. p. 667-672.

TAMIR, Y.; FRAZIER, G.L. Dinamically-Allocated Multi-Queue Buffers for VLSI Communication Switches. **IEEE Transactions on Computers,** [S.l.], v. 41, n. 6, p.725-737, June 1992.

ZEFERINO, C. A. et al. A Study on Communication Issues for Systems-on-Chip. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS, 15., 2002, Porto Alegre. **Proceedings...** Los Alamitos: IEEE Computer Society , 2002. p.121-126.

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

ZEFERINO, C. A.; SUSIN, A. A. A SoCIN: Parametric and Scalable Network-on-Chip. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS, 16., 2003, São Paulo. **Proceedings...** Los Alamitos: IEEE Computer Society , 2003. p.169-174.

ZEFERINO, C. A. et al. ParIS: A Parameterizable Interconnect Switch for Network-on-Chip. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS, 17., 2004, Pernambuco. **Proceedings...** Los Alamitos: IEEE Computer Society , 2004. p.204-209.

APÊNDICE (EM CD-ROOM)

Código VHDL do roteador Tonga – D:\Tonga\
Código VHDL do Bloco Input Channel A – D:\Input_ChannelA\
Código VHDL do Bloco Input Channel B – D:\Input_ChannelB\
Código VHDL do Bloco Input Channel L – D:\Input_ChannelL\
Código VHDL do Bloco Multiplex Controller – D:\Multiplex_Controller\
Código VHDL do Bloco Output Channel A – D:\Output_ChannelA\
Código VHDL do Bloco Output Channel B– D:\Output_ChannelB\
Código VHDL do Bloco Output Channel L – D:\Output_ChannelL\