Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática
Curso de Graduação em Engenharia de Teleinformática

Pedro de Almeida Lázaro

Concepção de um MPSoC de processadores Java utilizando NoC como mecanismo de comunicação.

Fortaleza – Ceará
Maio 2011

Autor:
Pedro de Almeida Lázaro

Orientador:
Prof. Msc. Jarbas Aryel Nunes da Silveira

Concepção de um MPSoC de processadores Java utilizando NoC como mecanismo de comunicação.

Monografia de Conclusão de Curso apresentada à Coordenação do Curso de Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de **Engenheiro de Teleinformática**.

Fortaleza – Ceará
Maio 2011

Pedro de Almeida Lázaro

Concepção de um MPSoC de processadores Java utilizando NoC como mecanismo de comunicação.

Esta Monografia foi julgada adequada para a obtenção do diploma de Engenheiro do Curso de Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará.

Pedro de Almeida Lázaro

Banca Examinadora:

Prof. Msc. Jarbas Aryel Nunes da Silveira
Orientador

Prof. Dr. Helano de Sousa Castro

Prof. Msc. Ricardo Jardel Nunes da Silveira

Fortaleza, 31 de maio de 2011

Dedico este trabalho primeiramente a Deus, à minha alma gêmea Araci, meu pai e grande tutor educacional Carlos Alberto, meus adoráveis irmãos, a toda minha família, em especial a uma pessoa que está torcendo muito pelo meu sucesso em uma dimensão que nenhuma rede multiprocessada pode alcançar, meu querido tio "Lelei"e aos amigos que sempre acreditaram no meu potencial.

"A mais alta das torres começa no solo."

Provérbio chinês

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AHB Advanced High-performance Bus

AMBA Advanced Microcontroller Bus Architecture

APB Advanced Peripheral Bus

ASB Advanced System Bus

ASIC Application-Specific Integrated Circuit

CBDA Centrally-Buffered, Dynamically-Allocated

CI Circuito Integrado

CLCD Connected Limited Device Configuration

CMP Chip Multi-processor

CPU Central Processing Unit

CISC Complex Instruction Set Computer

DSP Digital Signal Processor

EDA Electronic Design Automation

E/S Entrada e Saída

FCFS First Come First Served

FIFO First-In First-Out

flit unidade de controle de fluxo

FPGA Field-programmable Gate Array

GPP General-Purpose Processor

HOL Head-of-Line

IP Intellectual Property

JOP Java Optimized Processor

JVM Java Virtual Machine

J2ME Java 2 Micro Edition

LRS Least Recently Serve

MPSoC Multi-Processor SoC

NoC Network-on-Chip

OCP Open Core Protocol

OPB On-Chip Peripheral Bus

phit unidade de transferência de dado

RISC Reduced Instruction Set Computer

RTL Register Transfer Level

SAF Store-and-Foward

SCJ Safety Critical Java

SDRAM Sincronous Dynamic Random Access Memory

SMP Symmetric MultiProcessor

SoC System-on-Chip

SOPC System-on-a-Programable-Chip

SRAM Static Random Access Memory

TI Texas Instruments

VCT Virtual Cut-Through

VHDL VHSIC Hardware Description Language

VLIW Very Long instruction Word

VLSI Very-Large-Scale Integration

Resumo

Na era da nanotecnologia, processadores têm ganhado cada vez mais poder de processamento devido a uma maior integração dos transistores. Entretanto, aplicações modernas demandam capacidades de processamento bem superiores, as quais podem ser obtidas através do processamento em paralelo. Estes chips que utilizam múltiplos núcleos, conhecidos como MPSoCs, devem atender às exigências do mercado, como alto poder de processamento, baixo consumo de energia, além de possuírem componentes integrados escaláveis e reutilizáveis. MPSoCs que empregam redes-em-chip surgem como uma alternativa atende-las simultaneamente. Arquiteturas multiprocessadas atendem uma grande faixa de aplicações e algumas delas podem possuir um melhor desempenho através da utilização de redes de interconexão otimizadas para os padrões de comunicação exigido.

Dentro do contexto descrito, o presente trabalho aborda um MPSoC contendo processadores Java para aplicações de tempo real que se comunicam através de uma rede de interconexão chaveada. Na abordagem proposta, cada elemento de processamento executa, em paralelo, uma determinada tarefa da aplicação. O SoC proposto foi implementado em nível RTL, simulado e prototipado em FPGA.

Através das análises e medições que foram realizadas por ferramentas EDA, observou-se melhorias consideráveis no desempenho do SoC utilizando NoC em relação ao SoC utilizando barramento como forma de comunicação. Melhor desempenho na execução da aplicação, baixa potência e maior escalabilidade estão entre essas melhorias.

Palavras-Chave: Multiprocessadores, Redes-em-chip, Processador Java.

Capítulo 1

Introdução

Nos últimos anos, a evolução da nanotecnologia permitiu uma redução considerável nas dimensões dos Circuito Integrado (CI)s devido a um aumento do nível de integração dos transistores, o que concorda com a Lei de Moore(MOORE, 2000). Como consequência imediata, a indústria de semicondutores passou a produzir chips mais complexos e de alto desempenho, a um custo relativamente baixo. Os circuitos fabricados nessa década já possuem dezenas de bilhões de transistores, com dimensão em torno de $45nm$ e frequência de operação acima de $5GHz$. Logo, desenvolver tais sistemas complexos tornou-se um grande desafio para os projetistas.

Considerando esse avanço na tecnologia, projetistas estão desenvolvendo CIs que integram múltiplos componentes heterogêneos, como processadores, memórias e controladores, em um único chip caracterizando um sistema conhecido como System-on-Chip (SoC). No cenário atual da indústria de semicondutores, os componentes integrados devem ser reutilizáveis para garantir um tempo menor para o produto chegar ao mercado, ou seja, um *time-to-market* e um ciclo de vida reduzidos. Assim, as metodologias de projeto adotadas devem ser baseadas no reuso de componentes pré-projetados e pré-verificados, conhecidos como Intellectual Property (IP) *cores* (BERGAMASCHI *et al.*, 2001). Projeto de SoCs são fortemente baseados no reuso desses blocos.

1.1 Motivação

Muitas aplicações que envolvem comunicação sem fio, multimídia e redes possuem uma alta complexidade e demandam uma alta capacidade de processamento. Além disso, o mercado exige que os dispositivos que executam essas aplicações tenham restrições de potência e custo. Esse fato impulsiona o desenvolvimento de sistemas computacionais compostos de vários processadores, que antes eram implementados na forma de *clusters* (i.e. agregado de computadores), agora são implementados em um único CI. Um SoC fornece uma solução em um único chip para todos esses casos. Eles são frequentemente otimizados para melhorar a razão potência/desempenho ou o custo da aplicação (JERRAYA; TENHUNEN; WOLF, 2005).

Enquanto simples processadores podem ser suficientes para aplicações de baixo desempenho, há um número crescente de aplicações que necessitam de múltiplos processadores para alcançar seus requisitos de desempenho. Multi-Processor SoC (MPSoC)s são, portanto, cada vez mais utilizados para construir sistemas integrados.

Um dos tópicos importantes em MPSoCs está relacionado à sua infraestrutura de comunicação intrachip. O desempenho do sistema depende fortemente da forma de comunicação entre os núcleos processadores. Logo, é fundamental que tal arquitetura suporte taxas de comunicações consideráveis e alto paralelismo. Além disso, o crescimento do número de elementos de processamento nos sistemas consideráveis exige o emprego de infraestruturas mais escaláveis.

Tradicionalmente, utilizam-se dois mecanismos de comunicação, o barramento e canais ponto-a-ponto dedicados. Essas abordagens não atendem os requisitos de desempenho de comunicação, que são largura de banda escalável e paralelismo na comunicação, das futuras arquiteturas (**??**). As redes-em-chip, do inglês Network-on-Chip (NoC), consistem em uma abordagem de comunicação baseada no reuso de conceitos bem conhecidos de redes de computadores no domínio intrachip. O emprego de NoCs é imprescindível frente às limitações impostas pelos barramentos, relativas à baixa escalabilidade e ao pouco paralelismo suportado na comunicação (BENINI; MICHELI, 2002).

O elemento de processamento escolhido para compor o MPSoC desenvolvido neste trabalho é um processador Java. Ao contrário de muitos sistemas embarcados que utilizam fortemente linguagem C, tanto em nível de sistema operacional quanto

em nível de aplicação, este processador utiliza linguagem Java (ARNOLD; GOSLING; HOLMES, 2000) que possui características que não são encontradas na linguagem C, como:

- Linguagem orientada a objeto;
- Gerenciamento de memória com *garbage collector*;
- Proteção de memória implícita;
- *Threads*.

A Java Virtual Machine (JVM) (LINDHOLM; YELLIN, 1999) é um *software* implementado em um *hardware* não virtual e em sistemas operacionais padrões. Ela fornece um ambiente no qual os *bytecodes* Java podem ser executados. A JVM pode ser implementada de várias formas e uma delas é a implementação em *hardware*, na qual um processador Java possui *bytecodes* JVM como conjunto de instruções nativas.

Java Optimized Processor (JOP) é uma implementação em *hardware* da máquina virtual Java para pequenos sistemas embarcados voltados para aplicações em tempo real (SCHOEBERL, 2003). O mesmo é implementado como um soft-core em Field-programmable Gate Array (FPGA). O desenvolvimento de um chip com múltiplos processadores Java traz grandes benefícios, uma vez que cada elemento de processamento executa uma tarefa da aplicação. Além disso, Java possui um alto nível de abstração e recursos característicos de sistema operacional, como comunicação entre processos e escalonamento de tarefas. Dessa forma, é possível fazer um uso otimizado dos recursos de *hardware* e obter maior desempenho das aplicações em sistemas embarcados.

1.2 Objetivos

Os objetivos gerais e específicos desta monografia são apresentados a seguir.

1.2.1 Objetivos Gerais

Esta monografia tem como objetivo principal a concepção e implementação de um MPSoC heterogêneo baseado em NoC, no qual cada processador conectado a rede executa uma determinada tarefa dentro da aplicação. Além da

concepção, pretende-se realizar uma comparação de várias métricas de circuitos Very-Large-Scale Integration (VLSI) entre o MPSoC implementado com um SoC que possui os mesmos processadores, porém com a infraestrutura de comunicação diferente.

1.2.2 Objetivos Específicos

Dentre os objetivos específicos desse trabalho encontram-se:

i. Realizar uma revisão bibliográfica sobre MPSoC, NoC e o processador JOP.

ii. Implementar as interfaces de rede que conectam o elemento de processamento ao roteador da rede.

iii. Propor uma arquitetura de MPSoC.

iv. Realizar simulações para validar, testar e medir desempenho.

v. Obter estimativas de utilização de lógica, frequência máxima de operação e potência em FPGA.

1.3 Organização da Monografia

O presente trabalho encontra-se organizado em cinco capítulos. Neste primeiro capítulo foi apresentada uma introdução ao assunto bem como uma discussão da motivação que levou a escolha deste tema e dos objetivos do trabalho proposto.

O capítulo 2 apresenta uma revisão bibliográfica que inicia tratando sobre organizações de MPSoCs. Na sequência, os conceitos que envolvem NoC é apresentado. E por fim, é realizado um estudo do processador para aplicações de tempo real Java.

O capítulo 3 descreve todas as ferramentas utilizadas no desenvolvimento do trabalho, a arquitetura do sistema e a interface de rede proposta. Ainda no mesmo capítulo, são descritos como serão feitas as análises de desempenho, área e potência do MPSoC.

No capítulo 4 são apresentados, comparados e analisados todos os resultados obtidos no desenvolvimento trabalho.

O capítulo 5 apresenta as considerações finais da monografia e as perspectivas futuras na área de concepção de SoCs que utilizam redes de interconexão.

Capítulo 2

Fundamentação Teórica

Com o objetivo de tornar a compreensão deste trabalho mais fácil, algumas definições são necessárias e importantes. Neste capítulo serão descritos alguns fundamentos relacionados a sistemas multiprocessados e a redes de interconexão chaveada, comumente conhecida como NoC. Uma definição de interface de comunicação é dada, bem como exemplos de interfaces existentes no mercado. Adicionalmente, é feita uma breve explicação do funcionamento do processador JOP, que representa o elemento de processamento do MPSoC proposto, além de uma descrição geral de um Chip Multi-processor (CMP) composto por vários núcleos desse mesmo processador.

2.1 Sistemas Multiprocessados

2.1.1 MPSoC

Um MPSoC é um SoC - Um sistema VLSI que incorpora muitos ou todos componentes necessários para uma aplicação - que utiliza múltiplos processadores programáveis como componentes do sistema (WOLF; JERRAYA; MARTIN, 2008). Em geral, eles são compostos por vários processadores embarcados, memórias e núcleos de *hardware* digitais especializados. Todos esses componentes são conectados por meio de uma infraestrutura de comunicação, a qual requer flexibilidade para suportar a conexão de muitos e diversificados elementos de processamento.

MPSoCs constituem um ramo específico e importante de multiprocessadores. Eles não são simplesmente vários processadores tradicionais integrados em um

único chip. São projetados para satisfazer os requisitos específicos de aplicações embarcadas. São muito comuns em sistemas de computação embarcada porque eles permitem conhecer os requisitos de desempenho, custo e consumo de energia (WOLF, 2006).

Classificações

Levando em consideração o ponto de vista do multiprocessamento, um MPSoC é classificado como homogêneo quando os elementos processadores que o compõem são todos da mesma natureza. Por exemplo, um sistema composto por processadores idênticos que permitem exclusivamente a execução de tarefas de software compiladas para tal arquitetura de processador. De outra forma, quando o MPSoC possui elementos de processamento diferentes, como um General-Purpose Processor (GPP) e um Digital Signal Processor (DSP), ele é dito heterogêneo. Nesse caso as tarefas também serão de naturezas distintas (CARVALHO, 2009).

Enquanto MPSoCs homogêneos tendem a simplificar a aplicação de técnicas como migração de tarefas, MPSoCs heterogêneos podem suportar uma variedade maior de aplicações. Para garantir qualidade e desempenho, um decodificador de TV digital, por exemplo, deve ser heterogêneo o suficiente para integrar processadores Reduced Instruction Set Computer (RISC), núcleos de *hardware* dedicados (*e.g. upsampler*) e memórias (*e.g.* Sincronous Dynamic Random Access Memory (SDRAM)). Além disso, cada um desses componentes possui funcionalidades, tamanhos e necessidades de comunicação diferentes, o que demonstra a complexidade desses sistemas.

MPSoCs comerciais e aplicações

Nos anos 90, viu-se o desenvolvimento de muitos projetos VLSI baseados em apenas um processador, criados para aplicações embarcadas como multimídia e comunicações. Diversos processadores do tipo Very Long instruction Word (VLIW) foram desenvolvidos para proporcionar maior nível de paralelismo, juntamente com o auxílio da programação. Outra abordagem comum foi o desenvolvimento do Application-Specific Integrated Circuit (ASIC), que possibilitou a interligação de vários blocos juntos, muito dos quais não eram computadores de uso geral. A arquitetura desses sistemas, muitas vezes correspondia ao diagrama de blocos das aplicações para as quais eles foram desenvolvidos.

O primeiro MPSoC conhecido é o *Lucena Daytona* (ACKLAND *et al.*, 2000). Foi desenvolvido para estações de base sem fio, nas quais o processamento de sinais idênticos é executado nos canais de dados. *Daytona* possui uma arquitetura simétrica com quarto Central Processing Unit (CPU)s ligadas ao barramento de alta velocidade. A arquitetura da CPU é baseada no SPARC V8 com algumas melhoras. Os processadores compartilham um espaço de endereçamento comum na memória e há utilização de memórias *cache*. Sua arquitetura é ilustrada na Figura 2.1.

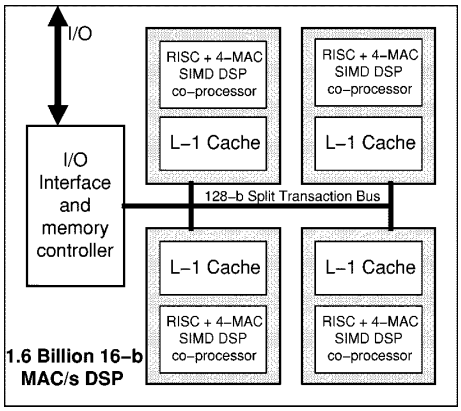


Figura 2.1: MPSoC Lucent Daytona, adaptado de (ACKLAND *et al.*, 2000)

O Processador de rede C-5 (CORPORATION, 2001) foi projetado para uma outra classe importante de aplicações embarcadas, que é o processamento de pacotes nas redes. A arquitetura do C-5 é mostrado na figura 2.2. Os pacotes são tratados pelos canais de processadores que estão agrupados em quatro *clusters* de quatro unidades cada. Três barramentos tratam com diferentes tipos de tráfego no processador. O C-5 usa vários processadores adicionais, alguns dos quais são muito especializados, considerando que o processador executivo é uma CPU RISC.

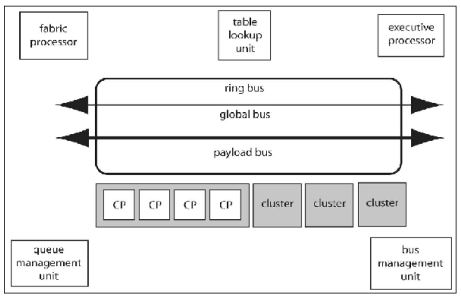


Figura 2.2: Processador C-5, adaptado de (CORPORATION, 2001)

Uma terceira classe importante de aplicações que utilizamMPSoC é o processamento multimídia. Um dos primeiros exemplos dessa classe é o Philips Viper

Nexperia (DUTTA; JENSEN; RIECKMANN, 2001), que está mostrado na figura 2.3. O Viper possui dois processadores: Um MIPS e um processador VLIW Trimedia. O MIPS é um mestre que executa o sistema operacional, enquanto que o Trimedia age como um escravo que realiza os comandos do MIPS. O sistema inclui três barramentos, uma para cada CPU e um para a interface de memória externa. Pontes são utilizadas para conectar os barramentos.

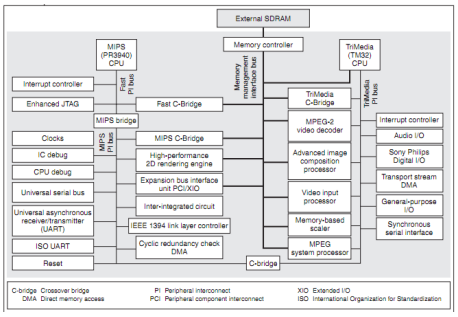


Figura 2.3: Viper Nexperia, adaptado de (DUTTA; JENSEN; RIECKMANN, 2001)

Uma quarta importante classe de aplicações é o MPSoC voltado para telefonia celular. Os primeiros processadores para celulares executavam operações de comunicação e multimídia. A arquitetura Texas Instruments (TI) OMAP (INC, 2004) tem várias implementações. O OMAP 5912, que pode ser visualizado através da figura 2.4, tem duas CPUs: Um ARM9 e um DSP TMS320C55x. O ARM atua como um mestre, e o DSP como escravo que realiza as operações de processamento de sinais.

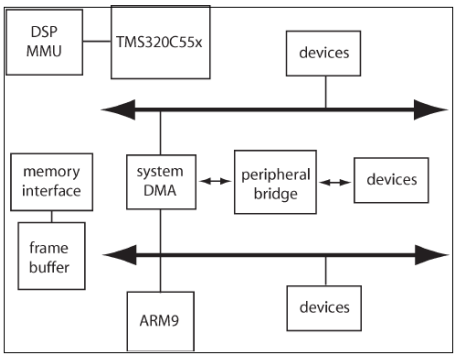


Figura 2.4: TI OMAP 5912, adaptado de (INC, 2004)

Nos últimos anos foram produzidos MPSoCs com quantidade significativas de processadores, como os propostos pela Intel (VANGAL *et al.*, 2007) e pela Tilera (CORPORATION, 2007). O primeiro deles é composto por 80 núcleos de

processamento idênticos, enquanto o outro por 64 núcleos também idênticos. Ambos utilizam NoC como mecanismo de comunicação entre os processadores.

2.1.2 MPSoC versus CMP

É importante ressaltar que MPSoC não é o mesmo que CMP. *Chip Multiprocessors* são componentes que tomam vantagem do aumento da densidade de transistores para incluir mais processadores em um único chip, mas eles não tentam alcançar as necessidades da aplicação. MPSoCs, ao contrário, são arquiteturas customizadas que procuram equilibrar as restrições da tecnologia VLSI com as necessidades da aplicação (JERRAYA; TENHUNEN; WOLF, 2005).

Processadores Multicores comerciais têm uma história mais curta do que os MPSoCs comerciais. Os primeiros *multicores* de propósito geral foram introduzidos em 2005. O Intel Core Duo Processor (GOCHMAN *et al.*, 2006), por exemplo, combina dois cores PENTIUM M melhorados em uma única pastilha silício. Os processadores são relativamente separados, mas compartilham uma cache L2 assim como a lógica de gerenciamento de energia.

2.2 Redes de Interconexão Chaveada

2.2.1 Conceitos Básicos sobre NoCs

Uma rede-em-chip pode ser definida como um conjunto de roteadores e canais ponto-a-ponto que interconectam os núcleos de um sistema integrado de modo a suportar a comunicação entre esses núcleos (ver Figura 2.5). A comunicação entre tais núcleos ocorre através da troca de mensagens geralmente transmitidas na forma de pacotes ao longo da rede (ZEFERINO, 2003a).

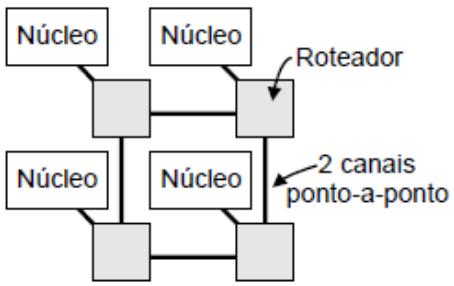


Figura 2.5: Rede-em-chip, adaptado de (ZEFERINO, 2003a)

Cada roteador tem um conjunto de portas que são usadas para conectar os roteadores com os seus vizinhos e com os núcleos de processamento do sistema. As portas utilizadas para conectar os núcleos de processamento são chamadas de portas locais ou terminais. A flexibilidade da NoC deve permitir a conexão de núcleos de diferentes naturezas, *e.g.* GPPs, memórias, dispositivos de Entrada e Saída (E/S) ou aindaIPs específicos. Adicionalmente, cada um deles pode ter características próprias de voltagem, frequência de operação e/ou tecnologia (ZEFERINO; KREUTZ; SUSIN, 2004). Redes-em-chip emergiram como uma solução alternativa para as restrições de arquitetura de comunicação devido as seguintes características (GUERRIER; GREINER, 2000):

i. Consumo de energia eficiente e confiabilidade;

ii. Escalabilidade da largura de banda quando comparada a arquiteturas de barramento tradicionais;

iii. Reusabilidade;

iv. Decisões de roteamento distribuídas.

Apesar das desvantagens como maior custo e latência na comunicação, esses problemas são atenuados pelos consequentes avanços na tecnologia CMOS, que permitiram a redução do custo e consumo dos semicondutores, além de soluções arquiteturais que permitem reduzir a latência da rede e seus efeitos no desempenho da aplicação.

Roteadores e canais de uma NoC

O roteador é o principal bloco construtivo de uma rede-em-chip cuja funcionalidade é encaminhar mensagens transferidas pela rede. É constituído por um conjunto de filas, conhecidos também como *buffers*, e multiplexadores (chaves) conforme ilustrado na Figura 2.6, além de blocos controladores que implementam os mecanismos de comunicação necessários à transferência de mensagens pela rede. Esses componentes podem ser construídos de maneira centralizada ou distribuída. Em geral, a primeira abordagem oferece uma maior taxa de utilização de recursos e a segunda propicia a construção de circuitos mais rápidos e com menor área.

Os roteadores e os núcleos em uma rede-em-chip são interligados por meio de canais ponto-a-ponto unidirecionais. Cada canal constitui um conjunto de fios

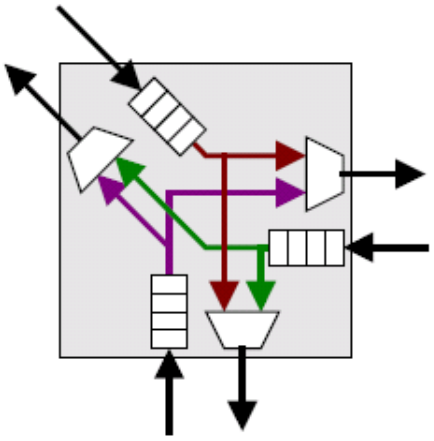


Figura 2.6: Roteador de uma rede-em-chip, adaptado de (ZEFERINO, 2003a)

que transportam os dados da mensagem. Mensagens maiores que o tamanho da palavra do canal devem ser quebradas e transferidas como sequências de palavras formando os pacotes. Adicionalmente, um canal inclui um conjunto de fios de banda lateral, geralmente utilizados para transferir informações sobre o enquadramento da mensagem (começo e fim), paridade da palavra de dado, sinalização de erro, entre outros. O conjunto de fios relacionado à palavra de dados e à banda lateral define a unidade física do canal ou *phit*.

Como um canal é unidirecional, ele conecta um emissor a um receptor. Para sinalizar ao receptor a presença de uma unidade de transferência de dado (phit) no canal, o emissor requer a disponibilização de algum tipo de recurso. Da mesma forma, como o receptor pode ter limitações para consumir um phit transmitido, ele requer outro recurso para sinalizar ao emissor a sua disponibilidade de consumir o phit ou receber um novo phit. Esses recursos são tipicamente implementados através de fios adicionais que são associados ao mecanismo de controle de fluxo, o qual é responsável pela regulação do tráfego de phit no canal. Logo, além dos fios associados ao phit (palavra de dados e banda lateral), um canal inclui um terceiro conjunto de fios para regular o tráfego sobre ele, conforme é ilustrado na 2.7. Como, em geral, em um sistema integrado, cada núcleo precisa enviar e receber mensagens, cada par de componentes conectados na rede (núcleo-roteador ou roteador-roteador) requer dois canais de comunicação unidirecionais, um para cada direção, os quais constituem o que se denomina de enlace (ou *link*).

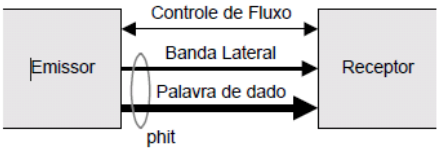


Figura 2.7: Canal de Comunicação, adaptado de (ZEFERINO, 2003a)

2.2.2 Métricas de uma rede

Podem-se utilizar vários atributos de redes em diferentes níveis de abstração para calculá-los e compará-los durante o projeto (WOLF, 2006).

Vazão Preocupação com a vazão máxima disponível e também as variações nas taxas de dados sobre o tempo e os efeitos dessas variações no comportamento da rede.

Latência Muitas vezes há o interesse de conhecer o tempo que um pacote, ou um conjunto deles, leva para ser transportado de uma fonte para um destino. Se a latência pode variar, medidas de melhor e pior caso, média e variância da latência são de fundamental importância.

Consumo de energia Interesse com a quantidade de energia necessária para enviar um bit através da rede. É uma medida típica.

Área e potência Os blocos contidos na rede determinam o seus custo de manufatura. A área do chip é obtida através de duas medidas: área de silício dos transistores e área de metal dos fios. A rede também pode fornecer algumas informações da potência dissipada. Dado que os fios são uma grande parte da rede global, área de metal é uma importante métrica de custo que influencia na capacitância da rede e na potência dinâmica.

2.2.3 *Starvation, Livelock e Deadlock*

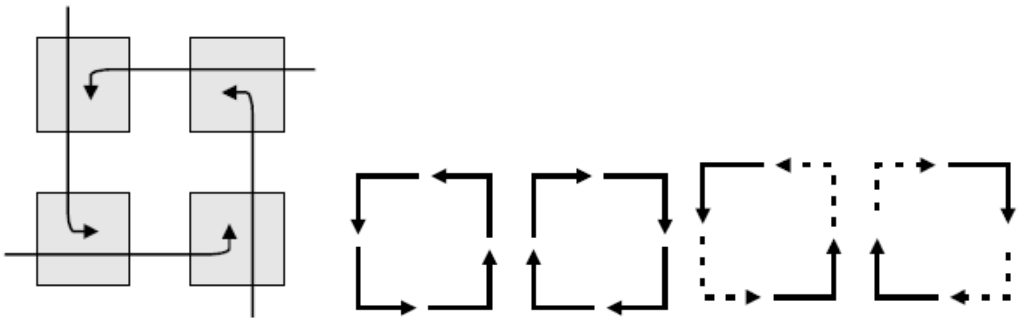
Nas redes-em-chip, há três situações que podem impedir que uma mensagem chegue com sucesso ao seu destinatário: *starvation, livelock e deadlock* (ZEFERINO; KREUTZ; SUSIN, 2004).

Starvation Quando o cabeçalho de um pacote chega em um roteador, o algoritmo de roteamento determina o canal de saída e emite um pedido ao árbitro para

realizar a alocação desse canal. O árbitro utiliza um critério de prioridades para selecionar um dos pedidos, pois podem existir muitos pedidos simultâneos para o mesmo canal. Dependendo do critério utilizado e do nível de tráfego para esse canal de saída, uma mensagem pode ser preterida indefinidamente e nunca ser selecionada para utilizar o canal, sofrendo *starvation*.

Livelock Se um algoritmo de roteamento permitir que qualquer saída seja utilizada para minimizar o *starvation*, pode permitir que a cada roteador ele selecione um canal de saída que afaste a mensagem do seu destinatário e ela nunca chegue ao mesmo. Essa situação configura um *livelock*.

Deadlock Ocorre quando há uma dependência cíclica na rede (Figura 2.8(a)), na qual cada mensagem garantiu a alocação de um canal e requer o uso de outro canal já alocado a outra mensagem. Essas dependências podem ocorrer em dois ciclos, como mostra a Figura 2.8(b). A solução para evitar *deadlock* consiste em proibir a realização de um subconjunto de curvas que a mensagem poderia realizar. Um exemplo desse tipo de solução está mostrado na Figura 2.8(c).



(a) (b) (c)

Figura 2.8: Dependência cíclica entre mensagens em uma NoC, adaptado de (ZEFERINO, 2003a)

A ausência da ocorrência de *stravation* é garantida pelo mecanismo de arbitragem, enquanto que a ausência de *livelock* e *deadlock* são garantidas unicamente pelo algoritmo de roteamento adotado.

2.2.4 Caracterização de uma NoC

Toda rede-em-chip é caracterizada pela sua topologia e seus mecanismos de comunicação. Essas características vêm das redes de interconexões para

computadores paralelos e também são atribuídas no nível intrachip. Os mecanismos de comunicação definem a forma como as mensagens serão transferidas na rede. Os principais mecanismos são controle de fluxo, roteamento, arbitragem, chaveamento e memorização.

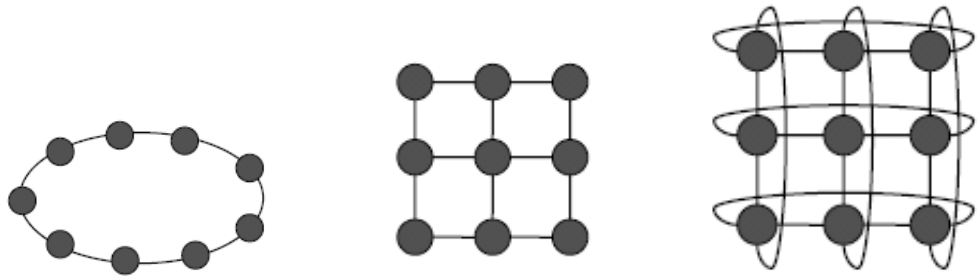
Topologia

A topologia de uma rede-em-chip determina o leiaute físico e as conexões entre os nodos e canais na forma de um grafo, no qual os roteadores são os vértices do grafo e os canais são os arcos. A topologia afeta diretamente no custo e no desempenho da rede como um todo. Uma topologia determina o número de saltos (*hops*) que uma mensagem deve atravessar, bem como o comprimento da interligação entre *hops*, influenciando significativamente na latência da rede. Como atravessar roteadores e *links* incorre no consumo energia, o efeito de uma topologia na contagem dos *hops* também afeta diretamente na potência da rede. Além disso, a topologia determina o número total de caminhos alternativos entre os nós, afetando em como a rede pode espalhar o trafego e, portanto, suportar os requisitos de largura de banda.

Uma topologia de rede pode ser classificada como direta ou indireta. Com uma topologia direta, cada núcleo de processamento está associado a um roteador, logo todos os roteadores são fontes e destinos do trafego. Em uma topologia indireta, os roteadores são diferentes dos núcleos de processamento, apenas os nodos terminais são fontes e destinos do tráfico, nodos intermediários simplesmente chaveiam trafego entre os nodos terminais. Em uma rede direta, pacotes são encaminhados diretamente entre nodos terminais. Com uma rede indireta, pacotes são chaveados indiretamente através de uma série de nodos de roteamento intermediários entre a fonte e o destino.

A figura 2.9 ilustra as topologias diretas mais utilizadas. Dentre elas, as topologias malha (Figura 2.9(b)) e toro (Figura 2.9(c)) fornecem um número menor de saltos em relação à topologia anel (Figura 2.9(a)). A grande maioria das redes-em-chip utilizam topologia malha.

Um exemplo de rede indireta é a rede *Butterfly*, ilustrada na figura 2.10, na qual os círculos representam os nodos terminais (*e.g.* processadores, memorias) e os quadrados os nodos roteadores. Os nodos fonte estão à esquerda e os nodos de destino à direita dos nodos roteadores.



(a) (b) (c)

Figura 2.9: Topologias de rede diretas, adaptado de (JERGER; PEH, 2009): (a) Anel (b) Malha (c) Toro

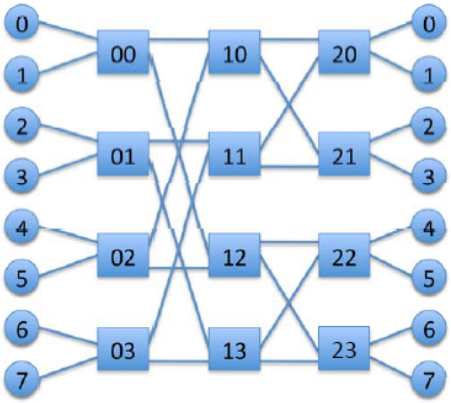


Figura 2.10: Topologia indireta: Butterfly, adaptado de (JERGER; PEH, 2009)

Controle de Fluxo

O controle de fluxo regula a alocação dos *buffers* e dos canais. Ele determina quando os *buffers* e os enlaces são atribuídos as mensagens, a granularidade a qual eles são alocados e como esses recursos são compartilhados entre as muitas mensagens através da rede (JERGER; PEH, 2009). Quando uma unidade de controle de fluxo (flit) não pode prosseguir porque algum recurso que ele necessita está sendo utilizado por outro flit ocorre uma colisão. Nesse caso, alguma política de controle de fluxo deve ser usada para decidir se o pacote deve ser descartado, bloqueado, recebido e armazenado temporariamente ou, então, desviado para outro caminho.

Geralmente as redes de interconexão realizam controle de fluxo em nível de enlace. Em cada terminal do enlace, os nodos podem possuir áreas para armazenamento do dado transferido, normalmente implementadas sob a forma de *buffers* First-In First-Out (FIFO). Se o *buffer* de entrada do receptor estiver cheio, o

transmissor deve manter o dado a ser enviado em seu *buffer* local até que o receptor esteja pronto para realizar a comunicação. Um mecanismo de controle deve então ser usado para bloquear a saída de dados do transmissor. Normalmente, o receptor envia uma informação de controle ao transmissor notificando se está pronto ou não para receber novos dados, conforme a Figura 2.11.

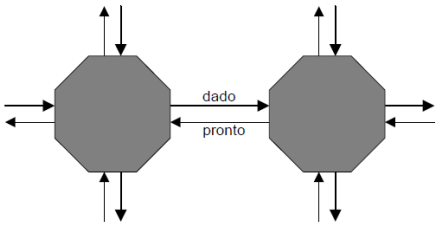


Figura 2.11: Controle de fluxo em nível de enlace, adaptado de (ZEFERINO, 1999)

Os mecanismos de controle de fluxo mais utilizados são descritos abaixo:

Handshake Nesse mecanismo de controle de fluxo, o transmissor ativa um sinal (tx/valid) indicando que um flit está sendo transmitido através do canal. Ao chegar o dado no receptor, o mesmo sinaliza através da ativação de um sinal *ack* se houver espaço disponível em seu *buffer* e armazena o dado.

Baseado em créditos É um tipo de controle que nunca descarta dados. Só ocorre transmissão quando existe espaço suficiente no *buffer* do receptor para armazenar o flit a ser recebido. Esse controle funciona da seguinte forma:

- **i.** O receptor envia ao transmissor uma informação de crédito relativa ao espaço em *buffer* disponível para recepção de dados;
- **ii.** O transmissor inicia uma transmissão apenas quando possuir crédito suficiente;
- **iii.** O crédito do transmissor diminui com o envio de dados e aumenta com o recebimento de mensagens de controle adequadas.

Canais Virtuais Se algum pacote de um *buffer* for bloqueado por conta de uma colisão e o pacote for maior do que o espaço livre disponível no *buffer*, o canal físico também estará bloqueado e nenhum outro pacote poderá utilizar esse canal. Esse problema é conhecido como bloqueio de cabeça de linha (Head-of-Line (HOL)).

Se o *buffer* de entrada for organizado em filas de profundidade menor, denominadas canais virtuais, obtém-se um conjunto de filas que podem ser alocadas independentemente umas das outras. Logo, se um canal virtual estiver bloqueado por algum motivo, o canal físico poderá ainda ser usado por outro canal virtual, resolvendo o problema do bloqueio HOL e aumentando a utilização do canal físico.

Roteamento

O algoritmo de roteamento é utilizado para decidir o caminho que um pacote, ou mensagem, tomará através da rede para alcançar seu destino. O objetivo do algoritmo de roteamento é distribuir uniformemente o tráfego entre os caminhos fornecidos pela topologia da rede, de forma a minimizar a contenção, melhorando a latência da rede e a vazão (JERGER; PEH, 2009).

Em geral, o algoritmo de roteamento visa atender a alguns objetivos específicos, os quais têm consequência direta em algumas propriedades da rede de interconexão (ZEFERINO, 1999), como:

- Conectividade - capacidade de rotear pacotes de qualquer nodo fonte para qualquer nodo destinatário.
- Liberdade de *deadlock* e *livelock* - capacidade de garantir que nenhum pacote ficará bloqueado ou circulando pela rede sem atingir o seu destinatário.
- Adaptabilidade - capacidade de rotear pacotes através de caminhos alternativos quando ocorrer congestionamento ou falha em algum componente do caminho em uso.
- Tolerância a falhas - capacidade de rotear pacotes na presença de falhas em componentes. A tolerância a falhas pode ser obtida sem adaptabilidade, roteando-se um pacote em duas ou mais fases.

Existem muitos algoritmos de roteamento que visam atender a requisitos distintos. Os algoritmos podem ser classificados conforme os seguintes critérios:

Quanto ao número de destinos

- *Unicast*: se os pacotes têm um único destino.

- *Multicast*: se os pacotes podem ser roteados para múltiplos destinos.

Quanto ao lugar onde as decisões de roteamento são tomadas

- Centralizado: se os caminhos são estabelecidos por um controlador central.
- Fonte: se o nodo fonte define o caminho a ser seguido pelo pacote antes de injetá-lo na rede.
- Distribuído: se o roteamento é realizado enquanto o pacote atravessa a rede.
- Multifase: quando o nodo fonte computa alguns nodos destinos, mas o caminho é estabelecido de forma distribuída. O pacote pode ser entregue a todos os nodos computados ou somente ao último nodo destino.

Quanto à implementação

- Tabela: se o roteamento é feito a partir de uma consulta a uma tabela em memória.
- Máquina de estados: se o roteamento é realizado a partir da execução de um algoritmo implementado em software ou em hardware.

Quanto à adaptabilidade

- Determinístico: se o algoritmo de roteamento fornece sempre o mesmo caminho entre um determinado par fonte-destino.
- Adaptativo: se o algoritmo de roteamento utiliza informação a respeito do tráfego da rede e/ou do estado dos canais para evitar regiões congestionadas ou com falhas.

Os algoritmos adaptativos, por sua vez, podem ainda ser classificados quanto:

- à progressividade: progressivo, se o cabeçalho sempre avança pela rede, reservando um novo canal a cada passo de roteamento; ou regressivo, se o cabeçalho pode retornar pela rede, liberando canais previamente reservados.

- à minimalidade: mínimo, se o algoritmo de roteamento pode selecionar apenas canais de saída que aproximem cada vez mais o pacote do seu destino; ou não mínimo, se o algoritmo de roteamento pode selecionar canais que levem o pacote a se afastar do seu destino.
- ao número de caminhos: completo, se o algoritmo de roteamento pode utilizar todos os caminhos disponíveis; ou parcial, se apenas um subconjunto desses caminhos pode ser usado.

Um dos algoritmos mais simples e mais utilizados, além de ser livre de *deadlocks* é o algoritmo XY (ZEFERINO, 1999). Seu funcionamento é descrito no parágrafo seguinte.

Em uma rede em grelha 2-D, cada nodo é representado por suas coordenadas (x, y). No algoritmo de roteamento XY, os pacotes são enviados primeiramente na dimensão X e depois na dimensão Y, sendo que apenas uma mudança de dimensão é permitida. Quando um pacote chega a um nodo, o valor dos deslocamentos em x e em y é calculado a partir das coordenadas do nodo corrente e do nodo destino do pacote. Se o valor do deslocamento tanto em x como e em y for nulo, o pacote é enviado ao processador local por meio do seu canal interno. Se o deslocamento em x for não nulo, o pacote é passado adiante através de um canal de saída na dimensão X. Por outro lado, se o deslocamento calculado em x for nulo, mas em y for não nulo, o pacote é então repassado por meio de um dos canais de saída na dimensão Y.

Arbitragem

Arbitragem define qual porta de entrada (ou *buffer* de entrada) poderá utilizar uma determinada porta de saída. Em outras palavras, o roteamento é um mecanismo de seleção de saída e a arbitragem é um mecanismo de seleção de entrada.

A arbitragem é fundamental para a resolução de conflitos decorrentes da existência de múltiplos pacotes competindo por uma mesma porta de saída. O mecanismo de arbitragem deve ser capaz de resolver esses conflitos, selecionando um dos pacotes com base em algum critério e sem levar qualquer pacote a sofrer *starvation*.

O árbitro de um roteador pode ser implementado de forma centralizada ou distribuída (ZEFERINO, 1999). Na abordagem centralizada, ilustrada na Figura

2.12(a), os mecanismos de roteamento e de arbitragem são implementados em um único módulo. Esse módulo recebe os flits de roteamento dos pacotes, executa o roteamento e determina a porta de saída a ser utilizada por cada pacote. A partir disso, ele faz a arbitragem, selecionando os pacotes a serem conectados em cada saída, configura o crossbar e habilita os *buffers* selecionados para os mesmos avançarem seus pacotes. Na abordagem distribuída, o roteamento e a arbitragem são realizados de forma independente para cada porta bidirecional do nodo de chaveamento. Como mostra a Figura 2.12(b) , cada porta possui um módulo de roteamento associado à sua porta de entrada e um módulo de arbitragem na sua porta de saída.

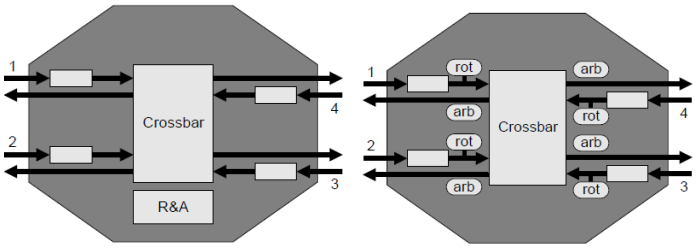


Figura 2.12: Roteamento e Arbitragem, adapatado de (ZEFERINO, 1999):(a)centralizado. (b) distribuído.

Os critérios de arbitragem mais utilizados são descritos abaixo (JERGER; PEH, 2009):

Round Robin Com um árbitro *Round Robin*, a última requisição a ser servida terá menor prioridade na próxima arbitragem. Um exemplo do uso do árbitro *Round Robin* é mostrado na Figura 2.13 na qual há um conjunto de pedidos a partir de quatro solicitadores. O último pedido atendido antes desse conjunto de solicitações foi do solicitador A. Como resultado, o B tem prioridade mais alta no início do exemplo. Logo as solicitações são preenchidas da seguinte forma: B1, C1, D1, A1, D2, A2.

FCFS O algoritmo First Come First Served (FCFS) atribui prioridade mais alta aos pacotes que chegam mais cedo, enquanto que os pacotes que chegam depois possuem prioridade mais baixa.

LRS O mecanismo Least Recently Serve (LRS) favorece a fonte que foi menos servida recentemente.

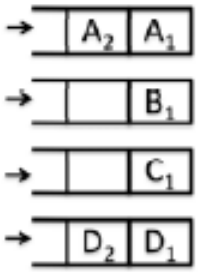


Figura 2.13: Exemplo de uso do árbitro Round Robin, adaptado de (JERGER; PEH, 2009)

Chaveamento

O chaveamento define a forma pela qual os dados são transferidos de um canal de entrada de um nodo para um dos seus canais de saída (ZEFERINO, 2003b). Existem diversas técnicas utilizadas no chaveamento de dados em redes de interconexão, as principais são: circuito chaveado, pacote chaveado, Virtual Cut-Through (VCT) e emphwormhole.

Chaveamento por circuito No chaveamento por circuito, um caminho físico entre a fonte e o destino é estabelecido para a transferência de uma mensagem, sendo mantido até o término da comunicação, a qual é realizada em duas etapas. Na primeira, o nodo fonte injeta na rede um cabeçalho de roteamento com o endereço destino e com algumas informações de controle. Esse cabeçalho avança pela rede, reservando canais físicos para o estabelecimento do circuito. Se um canal desejado já estiver sendo ocupado por outra mensagem, o cabeçalho fica bloqueado até que esse canal lhe seja alocado. Quando o cabeçalho atinge o nodo destino, uma informação de reconhecimento é enviada ao nodo fonte através do caminho de retorno do circuito estabelecido. A partir desse momento, ocorre a segunda etapa da comunicação, a qual consiste na transferência dos dados da mensagem. Durante o avanço do terminador da mensagem em direção ao destino, o caminho alocado pode ser desfeito.

Chaveamento por pacote É muito útil em situações nas quais as mensagens transmitidas entre nodos são frequentes e curtas. Nesse tipo de chaveamento, a mensagem é dividida em pacotes de comprimento fixo, cada um incluindo um cabeçalho com as informações necessárias para o seu roteamento pela rede. Os pacotes são enviados um a um e cada pacote reserva apenas os recursos necessários para avançar de nodo em nodo. Em cada nodo de chaveamento

deve ser fornecido um com capacidade para armazenar um pacote inteiro. Um nodo de chaveamento recebe um pacote, armazena-o em seu *buffer*, identifica o destino da mensagem, seleciona uma porta de saída com base em algum critério de roteamento e repassa o pacote adiante para um nodo de chaveamento adjacente ou para o nodo de processamento local, caso este seja o destino do pacote. Essa técnica é também denominada de armazena-e-repassa (ou Store-and-Foward (SAF)).

Chaveamento VCT O chaveamento VCT é uma alternativa ao chaveamento por pacote. Ele visa reduzir a latência na comunicação quando um pacote chega a um nodo de chaveamento e o canal desejado encontra-se disponível. No chaveamento por pacote, quando isso ocorre, o pacote deve ser armazenado inteiramente para então ser retransmitido pelo canal de saída desejado, mesmo que a porta de saída correspondente a esse canal esteja disponível.

No chaveamento VCT, quando o cabeçalho do pacote contendo as informações de roteamento chega a um nodo de chaveamento e o canal de saída desejado encontra-se disponível, o restante do pacote (corpo de dados e terminador) desvia o *buffer*, reduzindo a latência da comunicação. Um pacote só é armazenado no *buffer* se o canal desejado estiver indisponível. Para isso, o nodo de chaveamento deve ter espaço em *buffer* suficiente para armazenar completamente o pacote bloqueado. No pior caso, quando a rede está muito carregada, o VCT se comporta como o SAF.

Chaveamento *Wormhole* É uma variação do chaveamento VCT que tem como objetivo principal reduzir a quantidade de buffer necessária para manter pacotes bloqueados na rede. Um pacote de mensagem é dividido em flits que avançam pela rede em um modo *pipeline*. Os *buffers* dos nodos de chaveamento tem capacidade para armazenar poucos flits, de modo que se um pacote for bloqueado, seus flits serão mantidos em diferentes nodos na rede. Como a informação de roteamento é incluída nos flits de cabeçalho, os flits de dado devem seguir os primeiros pela rede. Como resultado disso, não é possível se realizar a multiplexação de flits de diferentes pacotes em um mesmo canal lógico. Um pacote deve atravessar completamente um canal antes de liberá-lo para outro pacote. Essa é uma das principais desvantagens dessa técnica de chaveamento, pois a probabilidade de ocorrer o *deadlock* é maior. Entretanto, essa restrição pode ser contornada através do uso de canais virtuais.

A grande vantagem do chaveamento *wormhole* é que as profundidades do *buffer* são menores que as das abordagens SAF e VCT, possibilitando a construção de roteadores pequenos e rápidos. Geralmente o chaveamento *wormhole* é o mais vantajoso com relação à utilização da rede e ao custo dos roteadores, sendo o mais usado atualmente (ZEFERINO, 2003b).

Memorização

Todo roteador com chaveamento por pacote deve ser capaz de armazenar os pacotes destinados a saídas que estejam sendo utilizadas por outros pacotes e, então, realizar o controle de fluxo de modo a evitar a perda de dados recebidos em cada canal de entrada. Isso exige a implementação de algum esquema de memorização para a manutenção dos pacotes bloqueados dentro do roteador.

Algumas estratégias de memorização utilizados nos roteadores são apresentadas:

Memorização centralizada compartilhada Nesse esquema de memorização, um *buffer* centralizado é utilizado para armazenar os pacotes bloqueados em todos os canais de entrada e o espaço de endereçamento é dinamicamente distribuído entre os pacotes bloqueados. Esse *buffer* é denominado Centrally-Buffered, Dynamically-Allocated (CBDA) e, no pior caso, deve oferecer uma largura de banda igual à soma das larguras de banda de todas os canais.

Memorização na entrada Nesse esquema, o espaço de memória é distribuído sob a forma de partições entre os canais de entrada do roteador. Essas partições são implementadas através de *buffers* independentes, cada um com portas de entrada próprias. Há várias opções de implementação para essa abordagem. A mais utilizada é a estratégia *buffer* FIFO.

Essa é a alternativa mais simples e de menor custo. Cada *buffer* FIFO possui um espaço de memória fixo onde os dados são lidos na mesma ordem em que são escritos. Porém, o seu grande inconveniente é o problema do bloqueio HOL, no qual um pacote bloqueado na saída do *buffer* impede o avanço de outro pacote que esteja atrás dele e para o qual a saída desejada esteja disponível, o que resulta em uma subutilização das portas de saída. Na Figura 2.14, é mostrado um roteador com quatro *buffers* FIFO nos canais de entrada.

Memorização na saída Consiste em particionar o espaço de memorização entre as saídas, sendo que as partições podem ser implementadas como *buffers* FIFO. O problema é que cada *buffer* deve ser capaz de suportar a demanda simultânea das N entradas. Esta estratégia requer um controle de fluxo interno entre portas de entrada e de saída do roteador.

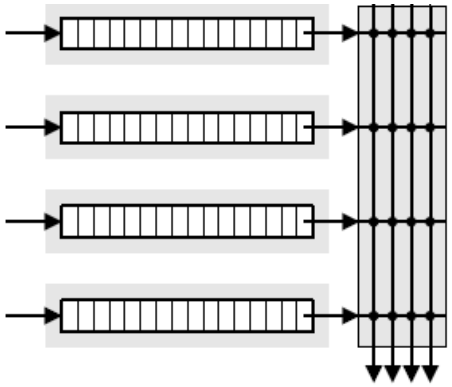


Figura 2.14: *Buffers* FIFO nas entradas do roteador, adaptado de (JERGER; PEH, 2009)

2.2.5 NoC Hermes

O MPSoC proposto utiliza NoC como infraestrutura de comunicação devido aos fatores de escalabilidade e paralelismo. Devido a sua simplicidade e baixo custo, foi empregada a rede interconexão Hermes (MORAES *et al.*, 2004). Além dessas características, um conjunto de ferramentas é disponibilizado (GAPH, 2007) e permite a geração da rede de acordo com diferentes parametrizações, bem como a geração de cenários de tráfego. Também possibilita simulação e prototipação do sistema, bem como avaliações da latência dos pacotes e dissipação de potência. Dentre os parâmetros tem-se as dimensões da NoC, o algoritmo de roteamento, o controle de fluxo, o tamanho dos *buffers* e a largura do flit.

O roteador da Hermes é ilustrado na Figura 2.15. Ele possui cinco portas, quatro delas são conectadas aos roteadores vizinhos, formando uma topologia malha 2D. A outra porta, denominada local, é conectada ao elemento de processamento, ou a um módulo IP. Cada enlace entre os roteadores da NoC possui dois canais de 16 bits de comunicação, um em cada um dos sentidos, permitindo a transmissão bidirecional simultânea entre os roteadores. Os outros parâmetros adotados por *default* são:

i. Chaveamento de pacotes *wormhole*;

ii. *Buffers* de entrada;

iii. Controle de fluxo por *handshake*;

iv. Roteamento XY determinístico.

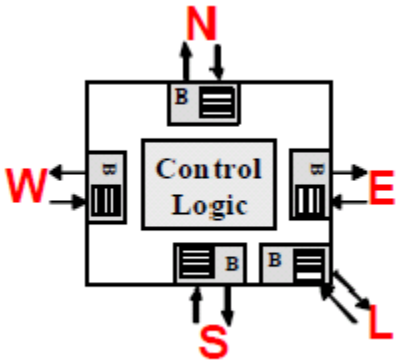


Figura 2.15: Arquitetura do roteador Hermes, adaptado de (MORAES *et al.*, 2004)

O protocolo *handshake* permite a transmissão de um flit de 16 bits a cada dois ciclos de relógio, enquanto que o protocolo baseado em crédito, que pode ser parametrizado utilizando a ferramenta Atlas (GAPH, 2007), realiza a transmissão do mesmo em um ciclo de relógio.

2.3 Interfaces de Comunicação

Muitos padrões de barramentos e de conexões ponto-a-ponto foram propostos nos últimos anos. São bastante usados para comunicar blocos de IP diferentes. Em redes-em-chip, um padrão de comunicação é utilizado para conectar a interface da porta local do roteador à interface do elemento de processamento. Os padrões mais conhecidos e utilizados serão brevemente descritos nessa seção.

O padrão Advanced Microcontroller Bus Architecture (AMBA) é a interconexão utilizada pelos núcleos ARM. Essa especificação define três barramentos diferentes: Advanced High-performance Bus (AHB), Advanced System Bus (ASB) e Advanced Peripheral Bus (APB). O AHB é usado para interconectar memórias internas ao chip, memórias cache e externa ao processador. Dispositivos periféricos são conectados ao barramento APB. ASB é o predecessor do AHB e não é mais recomendado para projetos novos.

Wishbone é um padrão de domínio público utilizado por muitos núcleos de IP de código aberto. Sua especificação ainda segue a tradição dos barramentos de microcomputadores. Entretanto, para interconexão de um SoC, que normalmente é ponto-a-ponto, essa não é a melhor abordagem.

A especificação da interface Avalon é fornecida pela Altera para interconexão de um System-on-a-Programable-Chip (SOPC). Ela define uma grande variedade de interconexão de dispositivos que vão desde uma simples interface assíncrona voltada para conexão direta com Static Random Access Memory (SRAM) até sofisticadas transferências em pipeline com latências variáveis. Essa grande flexibilidade fornece um caminho fácil para conectar um dispositivo periférico a essa interface.

O On-Chip Peripheral Bus (OPB) é um padrão aberto fornecido pela IBM e usado pela Xilinx. O OPB especifica um barramento para interconectar múltiplos mestres e escravos. A implementação do barramento não é diretamente definida na especificação. São sugeridas implementações como anel distribuído, multiplexador centralizado ou uma rede de portas AND/OR.

Sonics Inc. definiu o protocolo Open Core Protocol (OCP) como um padrão aberto e livremente disponibilizado. O padrão agora é manipulado pela parceria internacional OCP (www.ocpip.org).

2.4 Processador Java

Um processador Java, teoricamente, é uma implementação da máquina virtual Java (JVM) em *hardware*. Essa implementação não é completamente em *hardware* porque uma Máquina Virtual Java contém funções complexas como, por exemplo, escalonamento, gerenciamento e comunicação entre processos. O custo de implementar todos esses recursos em *hardware* pode tornar o projeto inviável. Logo, o conceito de um processador Java difere de um processador comum, no qual apenas elementos de *hardware* estão envolvidos. Dessa forma, um processador Java é uma implementação baseada em *hardware* e, possivelmente, em algum *software* (SILVEIRA, 2010).

O JOP (SCHOEBERL, 2003) é uma implementação em *hardware* e *softaware* da JVM, baseada no perfil Java 2 Micro Edition (J2ME) Connected Limited Device Configuration (CLCD) e na especificação Safety Critical Java (SCJ) .Este processador é utilizado em pequenos sistemas embarcados voltados para aplicações

em tempo real. O mesmo é implementado como um *soft* IP *core* em FPGA e, ao contrário da JVM que é uma máquina Complex Instruction Set Computer (CISC), o JOP é internamente uma máquina RISC, logo possui seu próprio conjunto de instruções.

2.4.1 Arquitetura

O JOP na sua configuração típica é composto de quatro blocos (ver Figura 2.16): *core* processador, interface de memória, interface de E/S e o módulo de extensão.

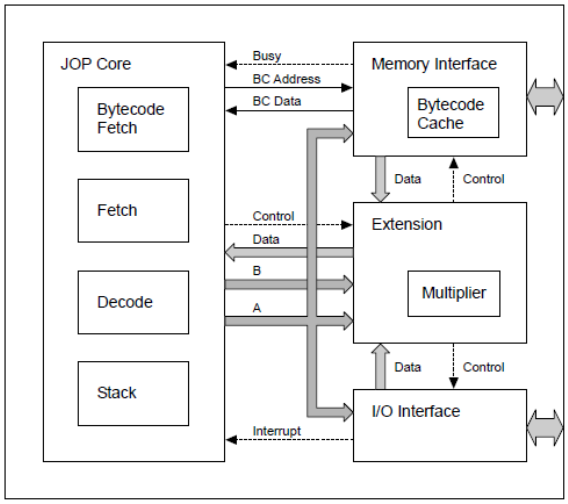


Figura 2.16: Diagrama de blocos do JOP, adaptado de (SCHOEBERL, 2005)

O *core* processador contém os quatro estágios de *pipeline*, que são *bytecode fetch*, *microcode fetch*, *decode* e *execute*. A interface de memória fornece a conexão entre a memória principal e o *core* processador. Ele também contém uma memória *cache* para armazenar os *bytecodes* Java. A interface de E/S contém os dispositivos periféricos, tais como o sistema de tempo e interrupção de *timer* e uma interface serial. O módulo conhecido como extensão executa três funções: contém aceleradores de *hardware* (nesse caso, o multiplicador), controla a memória e os dispositivos de E/S e contém o multiplexador para o dado lido que é carregado no topo da pilha (SCHOEBERL, 2009).

2.4.2 Implementação da JVM no JOP

É importante diferenciar dois conjuntos de instruções: *bytecode* e *microcode*. *Bytecodes* são instruções que compõem um programa Java compilado. Essas

instruções são executadas pela máquina virtual Java. *Microcode* é o conjunto de instruções nativas para o JOP.

Durante a execução, cada *bytecode* Java é traduzido em uma instrução de *microcode*, que é executada em um ciclo de relógio, ou em uma sequência delas, que ocorre quando os *bytecodes* possuem uma complexidade maior. Essa tradução só adiciona um estágio de pipeline ao *core* processador e não resulta em nenhuma uma sobrecarga na execução (SCHOEBERL, 2005).

Alguns *bytecodes*, tais como `new`, que criam e inicializam um objeto, são muito complexos de implementar em *hardware*. Logo, tem de ser emulado por *software* na própria linguagem Java resultando em uma sequencia de *bytecodes* que podem ser traduzidos pelo processador Java.

A Figura 2.17 ilustra um exemplo desse fluxo de dados do contador de programa Java para o *microcode* JOP. O *bytecode* solicitado atua como um índice para a tabela de salto. Essa tabela contém o endereço inicial para a implementação JVM em *microcode*. Esse endereço é carregado no contador de programa JOP para cada *bytecode* executado.

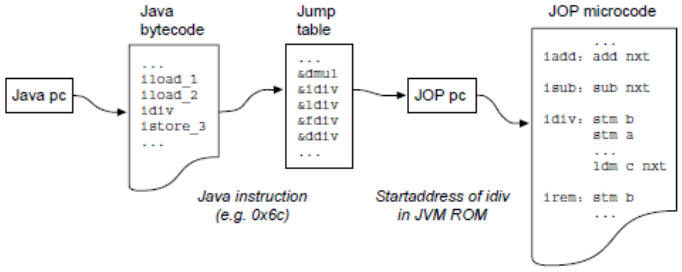


Figura 2.17: Fluxo de dados do Java pc para o JOP *microcode*, adaptado de (SCHOEBERL, 2009)

2.4.3 A interconexão SimpCon

SimpCon (SCHOEBERL, 2007) é a principal interface de interconexão utilizada pelo JOP. Os módulos de E/S e a memória principal são conectados através desse padrão. Esse padrão foi concebido com o intuito de ser simples e eficiente com relação ao recursos de implementação e latência na transação.

SimpCon é um padrão completamente síncrono para interconexões em-chip. Ele é uma conexão ponto-a-ponto entre um mestre e um escravo. O mestre inicia uma transação de escrita ou leitura, que levam apenas um ciclo de relógio, liberando o

mestre para continuar nas operações internas durante uma transação pendente. O escravo tem de registrar o endereço quando necessário por mais de um ciclo. Também registra o dado na leitura e o fornece ao mestre por mais de um ciclo. Isso permite o mestre atrasar a leitura atual se ele estiver ocupado com operações internas.

Os escravo sinaliza o final da transação através de um sinal chamado *ready counter* que fornece uma notificação antecipada, que simplifica a integração de periféricos em mestres em pipeline.

As principais características do SimpCon são fornecidas a seguir:

- Conexão ponto-a-ponto mestre/escravo;
- Operação síncrona;
- Transações de leitura e escrita;
- Liberação antecipada do pipeline para o mestre;
- Transações de pipeline;
- Especificação de código aberto;
- Baixo custo de implementação.

2.4.4 JOPCMP

JOPCMP (PITTER; SCHOEBERL, 2010) implementa um modelo Symmetric MultiProcessor (SMP) de memória compartilhada. Múltiplos processadores Java compõem a base de um CMP homogêneo. A rede de interconexão é responsável por conectar múltiplos processadores à memória. Um árbitro é parte dessa rede e controla o acesso a memória compartilhada. O barramento SimpCon é utilizado para conectar os processadores ao árbitro, e o árbitro ao controlador de memória. Um mecanismo de sincronização é responsável por coordenar o acesso a objetos compartilhados. A Figura 2.18 ilustra a arquitetura típica do JOPCMP.

Uma memória compartilhada é uma memória física onde todos os dados e instruções são armazenadas e acessíveis a todos os processadores. Um controlador de memória conecta as CPUs integradas na FPGA a memória compartilhada *off-chip* através do árbitro.

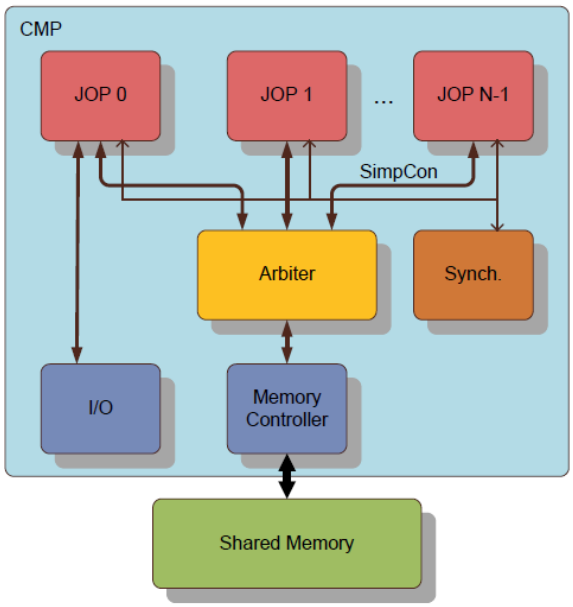


Figura 2.18: Diagrama de blocos do JOPCMP, adaptado de (PITTER; SCHOEBERL, 2010)

Sistemas SMP de memória compartilhada necessitam de um mecanismo de sincronização. CPUs trocam dados ao ler e escrever dados de objetos compartilhados. A fim de assegurar que uma CPU tenha acesso exclusivo a tal objeto, a sincronização é necessária. O módulo Syncronizer é responsável por realizar essa sincronização.

Um característica importante do uso desse sistema CMP é a execução de uma *thread* por núcleo de processamento. Nessa configuração, sobrecargas de escalonamento são evitadas e cada núcleo pode alcançar 100% utilização sem perder um *deadline*. Para explorar essa característica sem o uso de um escalonador, um mecanismo é fornecido para registrar objetos, que implementam a interface `Runnable`, para cada núcleo. Quando os núcleos são habilitados, eles executam o método `run` da interface `Runnable` como método principal deles.

Cada núcleo contém um conjunto de dispositivos de E/S locais, necessários para o sistema de execução, como por exemplo a interrupção de temporizador e suporte a sincronização). A interface serial para carregamento de programas e um dispositivo de E/S de propósito geral são conectados apenas no primeiro núcleo.

2.5 Resumo do Capítulo

Este capítulo apresentou, de maneira resumida, alguns fundamentos necessários e importantes para a compreensão e concepção do MPSoC proposto. Conceitos relacionados às redes-em-chip foram bastante enfatizados, pois eles ajudarão a compreender melhor os resultados obtidos. No capítulo seguinte será descrita a NoC proposta, bem como as interfaces de rede implementadas para a concepção do SoC, além das ferramentas necessárias.

Capítulo 3

Metodologia

Neste capítulo são descritas as ferramentas utilizadas, a arquitetura e funcionamento do SoC desenvolvido e o método de comparação entre o MPSoC proposto e o JOPCMP descrito na seção 2.4.4.

3.1 Ferramentas utilizadas

Para o desenvolvimento deste trabalho foram necessárias algumas ferramentas conhecidas como Electronic Design Automation (EDA), que são ferramentas de *software* voltadas para projetos de sistemas eletrônicos como placas de circuito impresso e circuitos integrados.

Primeiramente foi utilizado o *software* Modelsim ®para validação comportamental e funcional do projeto através de simulações lógicas. O Modelsim é um *software* para simulação de circuitos eletrônicos da empresa Mentor Graphics. Foi utilizado o Modelsim PE que uma é versão gratuita para estudantes. Após o projeto ser validado, o mesmo foi utilizado para a medição do desempenho dos SoCs comparados.

Para as análises de utilização lógica, frequência máxima de operação e consumo de potência em FPGA, foi utilizado o *software* ISE ®da empresa Xilinx. Essa ferramenta EDA é utilizada para a realização de síntese lógica e implementação para FPGAs da Xilinx além de fornecer informações sobre características temporais, utilização lógica e potência do circuito implementado. Foi utilizado o ISE 10.1 disponibilizado através do programa Universitário Xilinx.

3.2 MPSoC proposto

O MPSoC proposto consiste em um conjunto de processadores JOP conectados à rede-em-chip através de uma interface de rede, que é a ligação entre o elemento de processamento e o roteador. A NoC Hermes, descrita na seção 2.2.5, é a rede de interconexão utilizada pelos elementos de processamento para a comunicação entre eles. Todos os componentes integrados do sistema estão descritos em VHSIC Hardware Description Language (VHDL) e são codificados no nível Register Transfer Level (RTL), que permite a realização de síntese lógica para FPGAs e CIs.

O MPSoC desenvolvido tem o mesmo propósito do CMP explicado na seção 2.4.4. Entretanto, eles diferem quanto a infraestrutura de comunicação, que no JOPCMP é caracterizada por um barramento controlado por um árbitro.

3.2.1 Arquitetura

Como as redes-em-chip são arquiteturas escaláveis, foram projetadas três arquiteturas ilustradas na figura 3.1.

Foram desenvolvidas configurações de rede com dimensões 2x2, 3x3 e 4x4 com três, oito e quinze processadores idênticos, respectivamente, como mostram as figuras 3.1(a), 3.1(b), 3.1(c). Em todas as configurações está presente um controlador de memória, o que torna o MPSoC heterogêneo, segundo a classificação descrita na seção 2.1.1.

Duas abordagens de NoC foram desenvolvidas para comparação com o JOPCMP. Uma delas, utiliza *handshake* como mecanismo de controle de fluxo e é referenciada como JOPNoC. Na outra abordagem, o mecanismo baseado em créditos é usado e é referenciada como JOPNoC-BC. Esses mecanismos de controle de fluxo estão explicados na seção 2.2.4.

3.2.2 Funcionamento

Assim como no JOPCMP, a cada processador conectado a NoC é atribuída uma tarefa (*thread*) da aplicação Java que é executada em paralelo a outras tarefas. O processo de *boot-up* é o mesmo para todos os processadores até a execução das primeiras instruções de *microcode*. Apenas uma CPU, chamada de CPU_0, é designada para realizar a inicialização do sistema e apenas ela está em execução

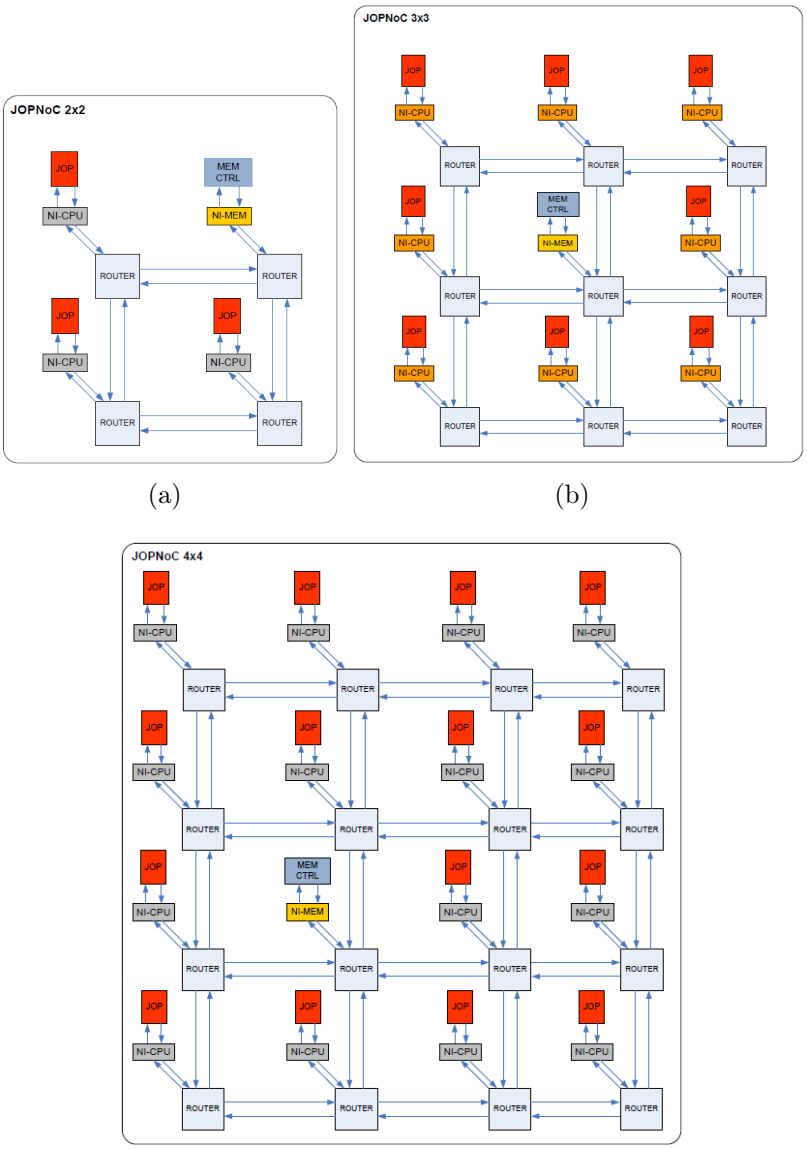


Figura 3.1: Arquiteturas do MPSoC: (a) NoC 2x2. (b) NoC 3x3. (c) NoC 4x4

nesse momento. As outras CPUs aguardam até a CPU_0 terminar a inicialização e executar o método `main` até o ponto em que as tarefas (*threads*) são iniciadas.

Como as instruções da aplicação estão contidas na memória externa compartilhada, cada núcleo do JOP se comunica unicamente com o controlador de memória, que é o módulo responsável pela interface com a memória externa. Portanto, não há comunicação entre processadores. Devido a essa característica na comunicação entre os nodos da rede, o controlador de memória foi atribuído a um roteador localizado na região central da topologia da NoC para que o número de

saltos na transferência de pacotes tenha um valor médio mínimo. Na Figura 3.1 é possível identificar a posição do controlador de memória nas três configurações desenvolvidas.

Quando algum JOP necessita acessar a memória compartilhada, um pacote cujo conteúdo são informações de endereço, dado, controle e uma identificação do roteador ligado a esse processador, é enviado para o roteador conectado ao controlador de memória. Ao receber essas informações, o controlador de memória realiza a operação de escrita ou leitura na memória e na sequência transmite um pacote referente ao dado lido ou à confirmação de escrita para o processador que solicitou o acesso a memória externa.

3.3 Interfaces de rede

Para a concepção do MPSoC , foi necessária a implementação das interfaces de rede que conectam a porta local do roteador ao núcleo de processamento. Foram desenvolvidas duas interfaces de rede: uma para conectar as CPUs e outra para conectar o controlador de memória. A interface de rede ligada ao JOP é chamada de NI-CPU e a interface de rede que conecta o controlador de memória é chamada de NI-MEM.

3.3.1 Arquitetura e funcionamento das interfaces de rede

As interfaces propostas possuem as seguintes características:

- Interface padrão SimpCon para os elementos de processamento.
- Empacotamento e transmissão dos sinais da interface SimpCon.
- Recepção e montagem dos pacotes recebidos em sinais do padrão Simpcon.
- Pacotes compostos de flits de 16 bits.
- Mecanismo de sincronização para aguardar o término de uma operação sobre a memória.

A Figura 3.2 mostra o diagrama de blocos da interface de rede NI-CPU.

O funcionamento desse módulo é descrito a seguir:

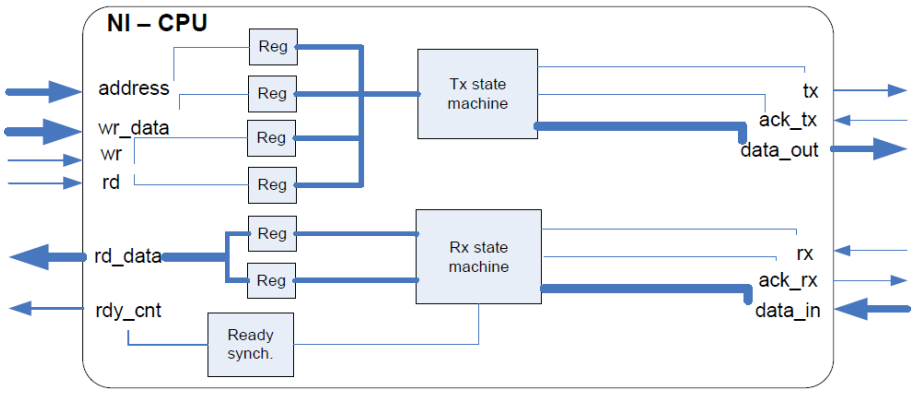


Figura 3.2: Arquitetura da NI-CPU

i. Quando ocorre uma solicitação de escrita ou leitura na memória externa, os sinais mestre do SimpCon, que são `address`, `wr_data`, `wr` e `rd`, são particionados e armazenados em registros de 16 bits. Nesse mesmo ciclo de relógio, a máquina de estados de transmissão inicia envio dos dois flits de cabeçalho, que indicam o roteador de destino, que sempre será o roteador ligado ao controlador de memória, e a quantidade de flits de carga útil. Na sequência, os flits de carga útil são enviados. Quando a operação é de escrita, são necessários quatro flits de carga útil para comportar todos os sinais do mestre. Já na operação de leitura bastam dois flits, pois o envio do sinal `wr_data`, cujo tamanho é de 32 bits, não é necessário.

ii. Após serem completadas as transações de escrita ou leitura, o pacote referente ao dado lido ou a confirmação da escrita é recebido pela interface da porta local. A máquina de estados de recepção lê e armazena dois flits de carga útil. Se a operação requisitada for de leitura, esses dois flits são montados e atribuídos ao sinal `rd_data` e representa o dado lido. Entretanto, se a operação for de escrita, os dois flits recebidos apenas representam a confirmação da escrita. Mesmo na operação de escrita, dois flits são necessários porque é a quantidade mínima de flits de carga útil permitida pela infraestrutura da NoC Hermes.

iii. Um mecanismo de sincronização é necessário para sinalizar ao processador que um dado lido está disponível ou que um dado já foi escrito na memória compartilhada. Após o início de uma operação de acesso a memória, o JOP fica em um estado de espera até que o mecanismo de sincronização altere o valor do sinal escravo `rdy_cnt` (*ready counter*) sinalizando ao processador que ele pode continuar suas operações internas.

A Figura 3.3 mostra o diagrama de blocos da interface de rede NI-MEM.

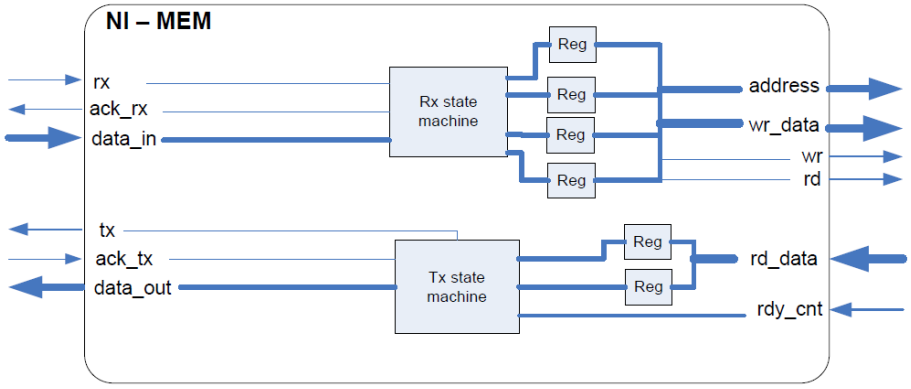


Figura 3.3: Arquitetura da NI-MEM

O módulo NI-MEM é similar ao módulo NI-CPU e funciona da seguinte forma:

i. Pacotes que chegam de solicitações de acesso à memória externa são lidos pela interface da porta local e armazenados em registros de 16 bits através da máquina de estados de recepção. Se a solicitação for de escrita, quatro flits de carga útil são armazenados, montados e atribuídos aos sinais `address` e `wr_data` e nesse momento o sinal `wr` é ativado. Todos esses sinais são ligados nos pinos de entrada do controlador de memória que então realiza a escrita. Caso a operação solicitada seja leitura, apenas dois flits de carga útil são armazenados, montados e atribuídos ao sinal `address` e nesse instante o sinal `rd` é ativado realizando a leitura do dado através do controlador.

ii. Após serem finalizadas as operações de escrita ou leitura, o dado lido é particionado em dois flits de carga útil e então é enviado para o roteador que solicitou o acesso a memória, via flits de cabeçalho, através da máquina de estados de transmissão. Quando a operação é de escrita, o último dado lido que fica armazenado é enviado apenas para sinalizar que a operação de escrita foi realizada, pois esse dado não será utilizado pelo processador.

3.4 Método de Comparação dos SoCs

Como o MPSoC proposto nesta monografia e o CMP desenvolvido por (PITTER; SCHOEBERL, 2010) possuem as mesmas características com diferença apenas na infraestrutura de comunicação, é possível a medição, comparação e análise das

métricas de desempenho na execução de uma aplicação, bem como de utilização lógica, frequência máxima de operação e potência dissipada em FPGA.

Para a comparação do desempenho dos SoCs, foi criada uma aplicação em Java que atribui tarefas a alguns ou a todos os processadores do sistema. O desempenho do JOPCMP com três, oito e quinze processadores foi comparado com o desempenho dos MPSoCs JOPNoC e JOPNoC-BC com dimensões 2x2, 3x3 e 4x4, respectivamente, utilizando o tempo de execução da aplicação medido através de uma simulação RTL.

As outras métricas foram obtidas e comparadas através da utilização do código RTL de cada SoC e do *software* ISE 10.1. As métricas de utilização lógica e frequência máxima de operação foram alcançadas através da síntese lógica para FPGA dos códigos RTL de cada SoC. Após o término desse processo, uma tabela é mostrada no ISE contendo a quantidade de células lógicas que foram utilizadas para a construção do circuito, bem como a frequência máxima de operação para o mesmo. Após o processo de síntese, foi feita a implementação que realiza o posicionamento e o roteamento das ligações entre as células lógicas. Com a finalização dessa etapa, medidas de potência foram extraídas do circuito através da ferramenta XPower que é integrada ao ambiente de desenvolvimento.

3.5 Resumo do Capítulo

Neste capítulo, foram descritas as ferramentas utilizadas para simulação, validação e extração das métricas dos SoCs comparados. A arquitetura e o funcionamento do MPSoC proposto foram explicados, bem como das interfaces de rede que foram implementadas para possibilitar a comunicação dos elementos de processamento. Também foi detalhado o método de comparação de métricas importantes dos SoCs. Essas métricas são apresentadas e analisadas no próximo capítulo.

Capítulo 4

Resultados

Os resultados das medidas realizados nos SoCs descritos no capítulo 3 são apresentados, comparados e analisados neste capítulo.

4.1 Tamanho dos *Buffers*

Como a profundidade do *buffer* influencia na latência da rede, na área e na potência do circuito, a escolha da mesma é importante durante o projeto. Após a avaliação de vários tamanhos de *buffer*, foram escolhidos aqueles que forneceram um bom custo benefício entre desempenho, custo e potência do *hardware*. Os tamanhos dos *buffers* presentes nos roteadores ligados aos processadores e no roteador conectado ao controlador de memória são mostrados na Tabela 4.1.

NoC	Rot CPU	Rot Ctrl Mem
2x2	4 flits	6 flits
3x3	6 flits	22 flits
4x4	8 flits	50 flits

Tabela 4.1: Tamanhos dos *buffers* utilizados pelos MPSoCs

Como o roteador conectado ao controlador de memória recebe pacotes de todos os processadores presentes na rede, o tamanho dos *buffers* presentes no mesmo é bem superior ao tamanho dos *buffers* instanciados nos roteadores ligados às CPUs. Essas profundidades são utilizadas na medição de todas as métricas mostradas nesse capítulo.

4.2 Desempenho na execução de uma aplicação

Primeiramente foi realizada uma simulação envolvendo o JOPCMP e as duas abordagens do JOPNoC, na qual a aplicação em execução inicia duas tarefas que são atribuídas a dois processadores. Como a execução do método `main` sempre é feito pela CPU_0 (seção 3.2.2), essa aplicação necessita apenas de três CPUs no sistema. Para uma frequência de relógio de $50MHz$, o desempenho de cada SoC para as três dimensões é mostrado na Tabela 4.2.

	JOPCMP	JOPNoC	JOPNoC-BC
3 CPUs (2x2)	$21ms$	$57,4ms$	$48,5ms$
8 CPUs (3x3)	$37,3ms$	$58,3ms$	$48,7ms$
15 CPUs (4x4)	$69,8ms$	$60,1ms$	$50,2ms$

Tabela 4.2: Desempenho para execução de duas tarefas (3 CPUs em execução).

Esses tempos medidos foram plotados em um gráfico que mostra o tempo de execução da aplicação descrita com relação ao número de CPUs presentes no sistema. A Figura 4.1 apresenta esse gráfico.

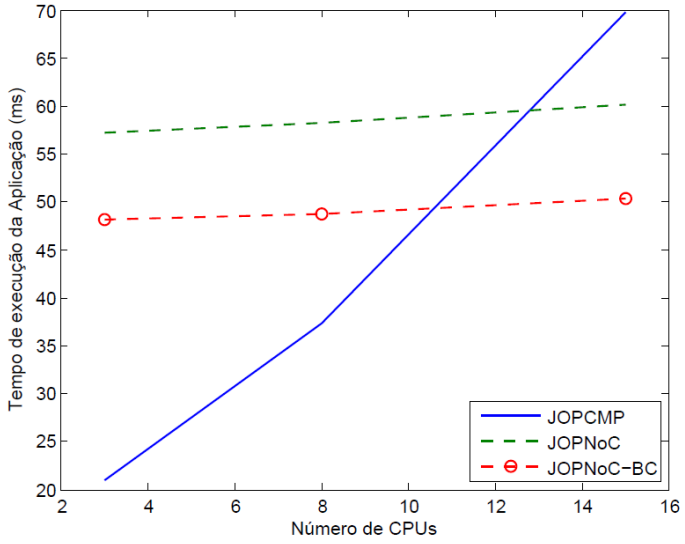


Figura 4.1: Gráfico de desempenho para 3 CPUs em execução

Outra medida de desempenho foi realizada utilizando a mesma aplicação, porém com pequenas alterações para atribuir tarefas a todas as CPUs disponíveis nos três tamanhos implementados para cada SoC. Os tempos de execução para essa aplicação são mostrados na Tabela 4.3.

	JOPCMP	JOPNoC	JOPNoC-BC
3 CPUs (2x2)	$21ms$	$57,2ms$	$48,2ms$
8 CPUs (3x3)	$47,6ms$	$80,6ms$	$67,8ms$
15 CPUs (4x4)	$94ms$	$90,4ms$	$75,4ms$

Tabela 4.3: Desempenho com todas as CPUs em execução.

Para todas as CPUs em execução, foi obtido um histograma de desempenho para cada dimensão dos SoCs que é ilustrado na Figura 4.2.

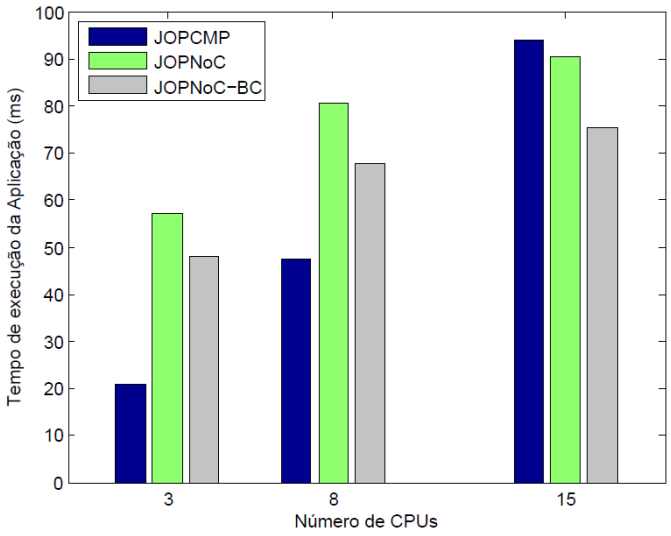


Figura 4.2: Histograma de desempenho para todas as CPUs em execução

4.2.1 Avaliação do desempenho

Os resultados mostram que para um número pequeno de CPUs, o desempenho do CMP, que utiliza uma conexão multiponto, é superior ao desempenho apresentado pelas duas NoCs. Isso se deve a latência que é inserida pela comunicação entre os roteadores. Porém, para a simulação que utilizou quinze processadores, o desempenho dos dois MPSoCs foi maior. O desempenho inferior do JOPCMP com quinze processadores se deve ao número de núcleos conectados ao barramento ser grande, logo o intervalo de tempo que cada CPU espera para ter acesso a memória compartilhada se torna bem significativo. Como as redes-em-chip possuem paralelismo na comunicação, a variação da latência com o aumento do número de processadores é menor que a variação apresentada pelo JOPCMP como podem ser constatadas através das inclinações das curvas mostradas na Figura 4.1 .

Para a aplicação que requisita todas as CPUs do sistema para execução, a variação da latência com o aumento do número de processadores apresentada pelos MPSoCs se torna maior em relação a execução de aplicação que requisita apenas três processadores. Esse aumento da latência é causado pelo maior número de processadores enviando e recebendo pacotes, ocasionando um aumento no tráfego da rede. Mesmo com todos os núcleos de processamento em execução, o desempenho dos MPSoCs foram superiores ao apresentado pelo JOPCMP para quinze processadores. Esses tempos podem ser verificados através do histograma ilustrado na Figura 4.2.

4.3 Lógica utilizada e frequência máxima de operação

Para a obtenção dos valores de utilização lógica e frequência máxima de operação em FPGA, foram sintetizados os três projetos descritos em VHDL utilizando o ISE 10.1. A FPGA escolhida foi a XC2PVP30-7FF896 da família Virtex 2 Pro e as quantidades de células lógicas utilizadas e a frequência máxima de operação para as arquiteturas com três CPUs (2x2)são mostradas na Tabela 4.4.

	4 Input LUTs	Slice FFs	f_{max}
CMP	11185	4437	$79MHz$
JOPNoC	12740	6284	$123MHz$
JOPNoC-BC	12696	5945	$86MHz$

Tabela 4.4: Utilização lógica e frequência máxima de operação

4.3.1 Avaliação da utilização de FPGA

Como as interconexões de um barramento envolvem custo mínimo de área, os SoCs que utilizam NoC possuem uma utilização de células lógicas superior devido à lógica relativa ao circuito do roteador. As relações de utilização de células lógicas de FPGA dos MPSoCs JOPNOC e JOPNOC-BC com o JOPCMP são mostradas na Tabela 4.5.

Esses resultados mostram um ganho bem maior na utilização de Slice Flip Flops pelas redes-em-chip. Essa porcentagem bem superior em relação ao JOPCMP é causada pelo uso de *buffers* na arquitetura do roteador, pois eles são implementados como um conjunto de registradores.

	4 Input LUTs	Slice FFs
JOPNoC/JOPCMP	$13,90\%$	$41,63\%$
JOPNoC-BC/JOPCMP	$13,50\%$	$33,98\%$

Tabela 4.5: Relação de utilização lógica entre os SoCs

Como a diferença entre o JOPNoC e o JOPNoC-BC reside apenas no mecanismo de controle de fluxo, a diferença de utilização lógica entre eles é bem pequena. Através dos dados na Tabela 4.4, tem-se 46 LUTs e 339 Slice Flip Flops a mais no JOPNoC.

4.4 Extração de Potência

Para obtenção da extração de potência dissipada na FPGA foi executada a implementação do circuito e em seguida a ferramenta XPower foi executada. Características que influenciam na potência como temperatura, condições de operação e tipo de processo foram configuradas como *default*. A extração da potência foi realizada para uma frequência de $50MHz$ e os valores obtidos para os três SoCs com 3 processadores (2x2) são mostrados na tabela 4.6.

	Potência Quiescente	Potência Dinâmica	Potência Total
JOPCMP	$0,103W$	$0,385W$	$0,488W$
JOPNoC	$0,103W$	$0,570W$	$0,673W$
JOPNoC-BC	$0,103W$	$0,558W$	$0,661W$

Tabela 4.6: Potência dissipada

4.4.1 Avaliação do consumo de potência em FPGA

A potência dinâmica de um chip está relacionada com a mudança dos níveis lógicos aplicados na entrada do transistor. Logo, circuitos sequenciais consomem mais energia que os circuitos combinacionais porque são alimentados por um sinal de relógio que varia constantemente. Como nos SoCs que utilizam NoC, o uso circuitos sequenciais é maior que no JOPCMP, pois o número de Slice Flip Flops é superior, a potência dinâmica dissipada por esses circuitos é maior. A Tabela 4.7 exibe as relações de potência dinâmica dissipada em FPGA dos MPSoCs JOPNOC e JOPNOC-BC com a do JOPCMP.

	Potência Dinâmica
JOPNoC/JOPCMP	48,05%
JOPNoC-CB/JOPCMP	44,93%

Tabela 4.7: Relação de potência dinâmica dissipada entre os SoCs

Circuitos de memórias internas ao chip descritos em linguagem de descrição de *hardware* são implementados pela ferramenta de síntese utilizando Flip Flops. Esse problema pode ser minimizado com o uso de blocos de memória pré-projetados. Esses blocos são otimizados em termos de área e consumo de potência.

4.5 Resumo do Capítulo

Neste capítulo, foram apresentados, comparados e analisados os resultados de desempenho obtidos através de simulação funcional. Dados de métricas de utilização lógica, frequência máxima de operação e potência de FPGAs foram mostrados e comparados para cada SoC. Uma justificativa foi dada para os valores obtidos e algumas soluções foram propostas. No próximo capítulo são apresentadas as considerações finais e as perspectivas de trabalhos futuros.

Capítulo 5

Conclusões e Trabalhos Futuros

Neste capítulo, primeiramente são apresentadas as contribuições desta monografia. Em seguida, algumas conclusões são apresentadas e alguns trabalhos futuros são propostos.

5.1 Contribuições

Uma das contribuições deste trabalho consiste na concepção e implementação de uma organização de MPSoC heterogêneo de processadores Java baseado em NoC. A outra contribuição, e mais importante, é a implementação das interfaces de rede que permitem a comunicação dos processadores com o controlador de memória através dos roteadores.

5.2 Conclusões

Nesta monografia, foi proposto um MPSoC composto de processadores JOP que compartilham uma memória externa no qual os dados que trafegam pela infraestrutura de comunicação, caracterizada por uma rede-em-chip, são instruções e dados de uma aplicação Java. Todos os módulos presentes nesse SoC foram descritos em linguagem de descrição de *hardware*, o que permitiu simulações, testes e implementação em FPGA.

No presente trabalho, foram avaliadas métricas de desempenho, custo, frequência máxima de operação e potência dissipada para os SoCs comparados. Quando se trata de aplicações para sistemas embarcados que necessitam de um alto processamento

em paralelo e baixo consumo, essas métricas são de fundamental importância no projeto do MPSoC.

Com relação à métrica de desempenho, constata-se que o uso de redes de interconexão chaveada realmente fornece uma comunicação de baixa latência e com alta escalabilidade, como mostram os resultados de desempenho para um número grande de elementos de processamento. As duas abordagens do MPSoC possuem custo e consumo de potência maiores, entretanto problemas de área e potência podem ser mitigados através da alta disponibilidade de transistores com o uso de tecnologias modernas e utilização de blocos de IP, respectivamente.

Através dos resultados de comparação obtidos, conclui-se que para aplicações que exigem uma grande computação e que só podem ter seus requisitos alcançados com o uso de muitos processadores, o emprego de NoC é bem vantajoso, pois permite a execução de muitas tarefas em paralelo em um tempo menor.

5.3 Trabalhos Futuros

Como a NoC Hermes é uma rede do tipo melhor esforço, as tarefas executadas pelos processadores perdem a previsibilidade, pois os pacotes sofrem atrasos arbitrários na rede. Uma proposta de trabalho futuro consiste na implementação de uma infraestrutura de rede-em-chip otimizada para aplicações de tempo real. Para isso pretende-se aplicar modificações nos mecanismos de comunicação e na topologia da NoC Hermes para fornecer serviços de vazão garantida.

Uma vez concluída a implementação de uma NoC com características de tempo real, outro trabalho futuro consiste no projeto de um CI desse MPSoC buscando melhorias de custo e potência através de técnicas de leiaute de CIs.

Referências Bibliográficas

ACKLAND, B. *et al.* A single-chip, 1.6-billion, 16-b mac/s multiprocessor dsp. *Solid-State Circuits, IEEE Journal of*, v. 35, n. 3, p. 412 –424, mar 2000. ISSN 0018-9200.

ARNOLD, K. *et al. The Java Programming Language, Third Edition.* [S.l.]: Addison-Wesley, 2000. ISBN 0-201-70433-1.

BENINI, L.; MICHELI, G. D. Networks on chips: A new soc paradigm. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 35, p. 70–78, January 2002. ISSN 0018-9162. Disponível em: <http://portal.acm.org/citation.cfm?id=619071.621885>.

BERGAMASCHI, R. *et al.* Automating the design of socs using cores. *Design Test of Computers, IEEE*, v. 18, n. 5, p. 32 –45, Sep 2001. ISSN 0740-7475.

CARVALHO, E. L. de S. *Mapeamento Dinâmico de Tarefas em MPSoCs Heterogêneos baseados em NoC.* Tese (Doutorado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2009. Disponível em: <http://tede.pucrs.br/tde_busca/arquivo.php?codArquivo=2360>.

CORPORATION, C.-P. *C-5 Network Processor Architecture Guide.* may 2001. North Andover, MA. Disponível em: <http://cache.freescale.com/files/netcomm/doc/ref_manual/C5NPD0-AG.pdf>.

CORPORATION, T. $TILE64^{TM}$ *Processor.* August 2007. Santa Clara, CA, EUA. Product Brief Description. Disponível em: <http://www.tilera.com/sites/default/files/productbriefs/PB010_TILE64_Processor_A_v4.pdf>.

DUTTA, S. *et al.* Viper: A multiprocessor soc for advanced set-top box and digital tv systems. *Design Test of Computers, IEEE*, v. 18, n. 5, p. 21 –31, sep-oct 2001. ISSN 0740-7475.

GAPH. *Atlas - An Environment for NoC Generation and Evaluation*. 2007. Porto Alegre, RS, BRA. Disponível em: <http://www.inf.pucrs.br/~gaph/AtlasHtml/AtlasIndex_us.html>.

GOCHMAN, S. *et al.* Introduction to intel core duo processor architecture. *Intelligence/sigart Bulletin*, 2006.

GUERRIER, P.; GREINER, A. A generic architecture for on-chip packet-switched interconnections. In: *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*. [S.l.: s.n.], 2000. p. 250 –256.

INC, T. I. *OMAP5912 Multimedia Processor Device Overview and Architecture Reference Guide*. mar 2004. Dallas, TX. Disponível em: <http://focus.tij.co.jp/jp/lit/ug/spru748c/spru748c.pdf>.

JERGER, N. E.; PEH, L.-S. *On-Chip Networks*. [S.l.]: Morgan and Claypool, 2009. (Synthesis Lectures on Computer Architecture).

JERRAYA, A. *et al.* Guest editors' introduction: Multiprocessor systems-on-chips. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 38, p. 36–40, July 2005. ISSN 0018-9162. Disponível em: <http://portal.acm.org/citation.cfm?id=1079829.1079866>.

LINDHOLM, T.; YELLIN, F. *The Java Virtual Machine Specification*. 2nd. ed. [S.l.]: Addison Wesley, 1999.

MOORE, G. E. Readings in computer architecture. In: HILL, M. D. *et al.* (Ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. cap. Cramming more components onto integrated circuits, p. 56–59. ISBN 1-55860-539-8. Disponível em: <http://portal.acm.org/citation.cfm?id=333067.333074>.

MORAES, F. *et al.* Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integr. VLSI J.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 38, p. 69–93, October 2004. ISSN 0167-9260. Disponível em: <http://portal.acm.org/citation.cfm?id=1056481.1056486>.

PITTER, C.; SCHOEBERL, M. A real-time java chip-multiprocessor. *ACM Trans. Embed. Comput. Syst.*, ACM, v. 10, n. 1, p. 9:1–34, 2010.

SCHOEBERL, M. Jop: A java optimized processor. In: *On the Move to Meaningful Internet Systems 2003: Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2003)*. Springer, 2003. v. 2889, p. 346–359. Disponível em: <http://www.jopdesign.com/doc/jtres03.pdf>.

SCHOEBERL, M. *JOP: A Java Optimized Processor for Embedded Real-Time Systems*. Tese (Doutorado) — Vienna University of Technology, 2005. Disponível em: <http://www.jopdesign.com/thesis/thesis.pdf>.

SCHOEBERL, M. Simpcon - a simple and efficient soc interconnect. In: *Proceedings of the 15th Austrian Workhop on Microelectronics, Austrochip 2007*. [S.l.: s.n.], 2007.

SCHOEBERL, M. *JOP Reference Handbook: Building Embedded Systems with a Java Processor*. [S.l.]: CreateSpace, 2009. 362 p.

SILVEIRA, R. J. N. da. *FT-JOP: UM PROCESSADOR JAVA PARA APLICAÇÕES DE TEMPO REAL EM SISTEMAS EMBARCADOS TOLERANTES A FALHAS*. Agosto 2010. Disponível em: <http://www.lesc.ufc.br/teses/papers/dissertacao-Jardel.pdf>.

VANGAL, S. *et al.* An 80-tile 1.28tflops network-on-chip in 65nm cmos. In: *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*. [S.l.: s.n.], 2007. p. 98 –589. ISSN 0193-6530.

WOLF, W. *High-Performance Embedded Computing*. San Franciso, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

WOLF, W. *et al.* Multiprocessor system-on-chip (mpsoc) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, v. 27, n. 10, p. 1701 –1713, October 2008. ISSN 0278-0070.

ZEFERINO, C. *et al.* Rasoc: a router soft-core for networks-on-chip. In: *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*. [S.l.: s.n.], 2004. v. 3, p. 198 – 203 Vol.3. ISSN 1530-1591.

ZEFERINO, C. A. *Redes de Interconexão para Multiprocessadores.* Porto Alegre, Brasil: [s.n.], 1999. Disponível em: <http://www.inf.pucrs.br/~noc/wiki3/lib/exe/fetch.php?media=docs:qualificacao_zeferino.pdf>.

ZEFERINO, C. A. *Introdução às Redes-em-Chip.* [S.l.], 2003.

ZEFERINO, C. A. *Redes-em-Chip: Arquiteturas e Modelos para Avaliação e Desempenho.* Tese (Doutorado) — Universidade Federal Rio Grande do Sul, Porto Alegre, Brasil, 2003. Disponível em: <http://www.inf.pucrs.br/~noc/wiki3/lib/exe/fetch.php?media=docs:tese_zeferino.pdf>.