UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RAIMUNDO VALTER COSTA FILHO

MPNI: UMA INTERFACE DE REDE COM SERVIÇOS DE PROGRAMAÇÃO PARALELA PARA REDES-EM-CHIP

MOSSORÓ - RN
2015

RAIMUNDO VALTER COSTA FILHO

MPNI: UMA INTERFACE DE REDE COM SERVIÇOS DE PROGRAMAÇÃO PARALELA PARA REDES-EM-CHIP

Orientador: Profº Silvio Roberto Fernandes de Araújo, D.Sc.

MOSSORÓ - RN
2015

Catalogação na Fonte
Catalogação de Publicação na Fonte. UFERSA - BIBLIOTECA CENTRAL ORLANDO TEIXEIRA - CAMPUS MOSSORÓ

Costa Filho, Raimundo Valter.
MPNI: uma interface de rede com serviços de programação paralela para redes-em-chip / Raimundo Valter Costa Filho. - Mossoró, 2015.
98f: il.

1. Arquitetura de computadores. 2. Network Interface - (NI). 3. Network-on-chip (NOC). I. Título

RN/UFERSA/BCOT/374 CDD 004.22
C837m

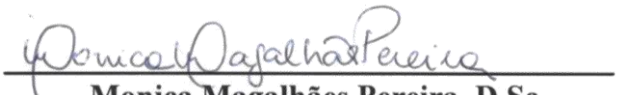
RAIMUNDO VALTER COSTA FILHO

MPNI: UMA INTERFACE DE REDE COM SERVIÇOS DE PROGRAMAÇÃO PARALELA PARA REDES-EM-CHIP

APROVADA EM: 13 de fevereiro de 2015.

BANCA EXAMINADORA

Silvio Roberto Fernandes de Araújo, D.Sc.
Orientador



Monica Magalhães Pereira, D.Sc.
Avaliadora Externa

Danniel Cavalcante Lopes, D.Sc.
Avaliadora do Programa

Dedico este trabalho aos meus filhos e esposa, pacientes companheiros. Aos Sr. Expedito Fechine de Párcio e Sra. Zilma Fechine de Párcio, meus avós maternos, essenciais em minha caminhada. A Deus por permitir que eu viva este sonho.

″...form ever follows function. This is the law.″

Louis H. Sullivan (Arquiteto Americano)

AGRADECIMENTOS

Agradeço fundamentalmente a Deus por ter me dado a oportunidade de vivenciar todas estas experiências e conhecer pessoas tão maravilhosas. Contribuintes direta ou indiretamente na conclusão desta obra. A meus filhos e esposa, acompanhantes fieis de minha caminhada no desenvolvimento deste trabalho, agradeço a paciência e compreensão.

Agradeço a todos os professores do Programa de Pós-graduação UFERSA-UERN que me acompanharam. Agradeço em especial ao professor Leonardo Casillo pelas conversas, materiais e sugestões que me trouxeram à mente a ideia de interface de rede para redes-em-chip e toda a pesquisa que se sucedeu.

Agradecimento especial, também, para os professores componentes da banca, por dedicarem tempo precioso na análise e crítica deste trabalho. Trabalho escrito cuidadosamente, evidenciando o conhecimento que pude reunir durante os últimos três anos na pesquisa sobre interfaces de redes para redes-em-chip.

Agradeço por último, mas não menos importante, ao meu orientador, Professor Sílvio Fernandes, por tudo que tem me ensinado, direto ou indiretamente, como professor e pesquisador. Pela paciência e compreensão que tem dispensado aos meus questionamentos e ideias. Sempre disponível, calmamente enfrentando comigo todas as dificuldades, "escrevendo à luz de velas quase na escuridão. Longe da multidão. Somos um exército, o exército de um homem só. No difícil exercício de viver em paz."Humberto Gessinger / Augusto Licks

RESUMO

As características dos *softcores* e o progresso da tecnologia dos ASICs, especificamente no desenvolvimento dos CPLDs e FPGAs, têm possibilitado o uso de sistemas multiprocessados em diversas aplicações desta tecnologia. Redes-em-chip foram introduzidas como uma opção interessante para sanar possíveis desafios de interconexão entre os dispositivos integrados. Recentemente, interfaces de rede tem surgido abstraindo o gerenciamento da comunicação envolvida no uso das redes-em-chip. Este trabalho apresenta uma abordagem *hardware/software* de uma interface de rede denominada *MultiProcessor Network Interface* (MPNI) para redes-em-chip que oferece serviços baseados no paradigma de troca de mensagens definido pelo padrão *Message-passing Interface* (MPI 3.0). É apresentado, também, os recursos da interface de rede MPNI e simulação contendo 16 nós integrados em um SoC conectados por uma NoC. Adicionalmente são expostos resultados da síntese.

Palavras-chave: Network Interface (NI), Network-on-Chip (NoC), Message-passing Interface (MPI), Multiprocessor-System-on-Chip (MPSoC), Application Specific Integrated Circuit (ASIC), Very-High Integrated Circuit Hardware Description Language (VHDL).

ABSTRACT

Softcores inherent characteristics and the manufacturing technological progress of ASICs, specifically CPLDs and FPGAs, have made possible the usage of multiprocessors systems in many applications of such technology. Network-on-chip was presented as an interesting option for intra-chip interconnection problem and recently network interfaces have emerged abstracting communication management involved by its usage. This work presents a hardware/software approach of a network interface called MPNI (MultiProcessor Network Interface) offering services based on Message-Passing Interface (MPI 3.0) standard to connect processors using a network-on-chip. It also present the MPNI network interface features and a simulation of the MPNI with 16 nodes integrated in a SoC connected by a NoC. Additionaly, synthesis results are presented.

Key-words: Network Interface (NI), Network-on-Chip (NoC), Message-passing Interface (MPI), Multiprocessor-System-on-Chip (MPSoC), Application Specific Integrated Circuit (ASIC), Very-High Integrated Circuit Hardware Description Language (VHDL).

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACM *Association for Computing Machinery*, página 21

API Application Program Interface, página 41

CAPES Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, página 21

CBDA *Centrally-Buffered, Dynamically-Allocated*, página 29

CNI *Core Network Interface*, página 36

CNI-CS *Core Network Interface - Communication Scheduler*, página 37

CNI-PM *Core Network Interface - Power Manager*, página 37

DAMQ *Dinamically Allocated Multi-Queue*, página 29

De-PACK *De-packetizer*, página 37

DEC *Decoder*, página 36

DMA *Direct Memory Access*, página 91

DR/RT *Destination Table/Route Table*, página 37

eMPI embedded Message-Passing Interface, página 41

ENC *Encoder*, página 36

EP Elemento Processante, página 44

FIFO *First In, Firt Out*, página 27

FLIT *Flow Control Unit*, página 31

FPGA *Field Programmable Gate Array*, página 58

HDL *Hardware Description Language*, página 20

HLP *Hight Level Protocols*, página 32

HOL *Head-of-Line blocking*, página 29

IEEE *Institute of Electrical and Electronics Engineers*, página 21

IP *Intellectual Property*, página 27

MPI *Message Passing Interface*, página 18

MPNI *MultiProcessor Network Interface*, página 19

MPNI *Multiprocessor Network Interface*, página 44

MPSoC *Multiprocessor System on Chip*, página 19

NI *Network Interface*, página 19

NoC *Network-on-Chip*, página 44

NoC *descriptionNetwork-on-Chip*, página 19

OCP *Open Core Protocol*, página 36

PACK *Packetizer*, página 36

PAN Protocolo de Alto Nível, página 46

PE *Processing Element*, página 19

QoS *Quality of Service*, página 38

RIB *Routing Information Bits*, página 32

RTL *Register Transfer Level*, página 20

SAF *Store-and-forward*, página 31

SAFC *Statically Allocated, Fully Connected*, página 29

SAMQ *Statically Allocated Multi-Queue*, página 29

SB *Status Bits*, página 46

SO Sistema Operacional, página 64

SoC System-on-Chip, página 18

SoCINfp *System-on-Chip, Interconnection Network fully parameterizable*, página 23

TC *Test Controller*, página 37

VCT *Virtual Cut-throught*, página 31

VHDL *Very High Speed Circuits Description Language*, página 44

XUIRU *eXtensible InteRface Unit*, página 38

SUMÁRIO

1 INTRODUÇÃO

A necessidade de produzir computadores com maior desempenho tem levado ao desenvolvimento contínuo do *hardware* dos processadores desde seu surgimento. A indústria de integração dos circuitos em *chip*, a cada ano, tem batido recordes na miniaturização de circuitos pelo uso de novas técnicas litográficas. Simultaneamente, o aumento na frequência do *clock* destes circuitos balizados pela eficiência crescente das técnicas de resfriamento permitiu uma melhora no desempenho dos processadores modernos. Entretanto, diminuir a escala e aumentar a frequência do *clock* em *chips* torna-se uma tarefa cada vez mais difícil. Isto porque há um limite físico para a miniaturização e aumento na frequência de operação desses circuitos semicondutores. Em resposta a este cenário, os fabricantes vem constatando que é vantajoso paralelizar o processamento e reduzir o *clock* utilizado pelos processadores interconectados. Assim, recentemente, a indústria tem popularizado os *multicores* e multiprocessadores, ambos possibilitando melhorar o desempenho dos computadores do futuro, abrindo caminho para os desenvolvimento de softwares paralelos.

A miniaturização dos componentes eletrônicos projetáveis em silício tem possibilitado incluir cada vez mais circuitos dedicados em uma pastilha. Integrar dezenas e até centenas de processadores em um único *chip* tornou-se possível. Apesar da facilidade em criar *multicores* e multiprocessadores, novos desafios se evidenciam. Um deles é: como interligar enormes quantidades de núcleos viabilizando a troca de informações entre eles sem que grandes atrasos sejam incluídos? O barramento, método comumente empregado para conexão entre dispositivos em uma placa eletrônica ou mesmo intra-chip suporta a interligação de apenas poucas dezenas de elementos, inviabilizando seu uso em sistemas maiores. Assim, como uma solução criativa, surgiram as redes-em-chip (do inglês Network-on-chip, ou NoC) (BENINI; MICHELI, 2002), provendo características como: escalabilidade, troca de mensagens não bloqueante, tolerância a falhas e comunicação seletiva (ZEFERINO, 2003).

A partir do início do século XXI, redes-em-chip têm sido objeto de intensa pesquisa. Aspectos como arquitetura dos roteadores, topologias da rede, mecanismos de roteamento, arbitragem, etc. têm sido insistentemente pesquisado com o intuito de adequar o uso dessas redes às diversas aplicações existentes (LEE et al., 2008). Porém, assim como as redes que as inspiraram, redes de comutação de pacotes internas a um *chip* possuem aspectos particulares análogos às populares redes de computadores (BENINI; MICHELI, 2002). Entretanto, é necessário a definição de protocolos e mecanismos para o gerenciamento do meio de interconexão. Esta necessidade fez surgir as interfaces de rede, com o objetivo principal de tornar transparente o tratamento das comunicações e

fazer das redes-em-chip um meio facilmente adaptável aos mais distintos dispositivos que as utiliza (BHOJWANI; MAHAPATRA, 2003).

As interfaces de rede para redes-em-chip merecem profunda discussão. Neste sentido, esta dissertação aborda alguns aspectos considerados chave na utilização de interfaces de rede, focando em aplicações multiprocessadas e no paradigma de troca de mensagens, tomando o padrão MPI 3.0 (FORUM, 2012) como um norte na concepção de uma interface de rede apropriada para compor projetos de *multicores*/multiprocessadores.

1.1 CONTEXTUALIZAÇÃO

Existem muitas aplicações suficientemente complexas onde interconexão por barramento ou *crossbar* não são aplicáveis. Com o aumento do número de blocos *Intellectual Property* (IP) integrados nos projetos em silício atualmente, a tecnologia de rede-em-chip inegavelmente proporciona uma boa opção de conectividade desses núcleos. Assim, profissionais da área de projeto de sistemas em chip (SoC , do inglês System-on-Chip) precisam tem empregado as redes-em-chip não somente por sua funcionalidade de comunicação, mas, também, devido a confiabilidade e a facilidade em utilizar subsistemas *plug-and-play* para compor projetos, reduzindo cada vez mais o *time-to-market* (BHOJWANI; MAHAPATRA, 2006).

No contexto de multiprocessadores e redes-em-chip, supõe-se que, assim como as camadas de *hardware* e *software* presentes nas redes de computadores, estruturas semelhantes devem estar presentes nas aplicações internas ao *chip* ou intra-chip. Esta abordagem agiliza e torna transparente as questões inerentes à conectividade entre os dispositivos processantes. Estas estruturas gerenciam a comunicação, simplificam a conexão dos nós e ampliam a empregabilidade deste método de comunicação em projetos de circuitos integrados. No início do uso das redes-em-chip falava-se do simples empacotamento dos dados para que estes pudessem ser roteados através da rede. Lee et al. (2008) ressaltam que esse modelo poderia ser melhorado, apontando como tendência o desacoplamento entre computação e comunicação. Lee et al. (2008) e outros (RADULESCU et al., 2004; BHOJWANI; MAHAPATRA, 2006; JOVEN et al., 2008; GARCIA, 2009; MATOS, 2010; STEFAN; WINDT; GOOSSENS, 2010; CARARA, 2011; MELO, 2012) propuseram diferentes arquiteturas com o mesmo fim: abstrair a complexidade do uso das redes-em-chip.

As interfaces de rede mantêm a complexidade da comunicação em núcleo da rede, onde os projetistas especializados em redes-em-chip identificam e implementam inovações neste nível, enquanto usuários desta tecnologia se concentram nas aplicações que as utilizam. Esta virtualização da rede implica em flexibilidade, segurança, redução

de custos, escalabilidade, tolerância a falhas e modulariza o sistema, criando um ambiente onde parte das ações de comunicação são executadas, reduzindo a complexidade dos dispositivos em camadas superiores.

Do ponto de vista do usuário de sistemas baseados em redes-em-chip e interfaces de rede, a transparência de questões intrinsecamente ligadas à execução da comunicação permite mais inovação. Estimulando a produção de uma maior variedade de sistemas e popularizando o uso das redes-em-chip/interfaces de rede com o intuito de atender a demanda das mais distintas aplicações.

1.2 PROBLEMÁTICA

Apesar de intensa pesquisa no desenvolvimento das redes-em-chip e relativamente pouco ter sido discutido acerca do desenvolvimento de interfaces de rede para apoio aos IPs e *Processing Elements* (PEs) interconectados por uma NoC (*Network-on-Chip*) (MATOS, 2010), temos algumas propostas que já delineiam a estrutura básica das soluções a serem empregadas nesta área. Em contrapartida, as técnicas propostas por Radulescu et al. (2004), Bhojwani e Mahapatra (2006), Garcia (2009), Matos (2010), Stefan, Windt e Goossens (2010), Carara (2011), Melo (2012), são genéricas, visam servir a uma gama de aplicações sem especificamente dar suporte a sistemas usuais de multiprogramação, por exemplo.

A maioria das propostas pesquisadas na produção deste trabalho não oferecem suporte especificamente a operações de processamento paralelo a fim de virtualizar as estruturas de comunicação disponíveis. Assim, é sugerido como solução a interface de rede *MultiProcessor Network Interface* (MPNI). Esta oferece serviços baseados no padrão *Message-Passing Interface* (MPI) (FORUM, 2012), exigindo dos processadores apenas operações *load/store* sobre um sistema de memória distribuído. Neste esquema, nenhuma alteração física do processador ou da rede-em-chip é requerida, sendo necessária apenas a programação das ações de comunicação na própria memória local compartilhada entre o processador e a interface de rede.

Esta solução dispõe de meios para que processadores conectados a uma rede-em-chip mantenham comunicação sem que, para isto, gerenciem diretamente as operações de comunicação. Esta proposta é uma camada em *hardware* projetada para trabalhar com o processador por meio de operações *load* e *store* em uma memória de rascunho compartilhada entre ambos. O processador manipula uma estrutura de dados específica presente na memória de rascunho solicitando esses serviços e a interface de rede acessa e interpreta, de maneira autônoma, as informações, gerenciando a execução da comunicação. Tal interface de rede (do inglês *network interface* - NI) implementaria

serviços de passagem de mensagens parametrizáveis em tempo de projeto. Oferecendo, deste modo, opções de subconjuntos de serviços MPI, possibilitando a escolha otimizada dos serviços necessários ao desenvolvimento de cada projeto. Essa abordagem mostra-se útil especialmente por reduzir a complexidade inerente ao uso de redes-em-chip como meio de interconexão, criando uma camada dedicada ao suporte de operações de comunicação em *Multiprocessor System on Chip* (MPSoC), reduzindo carga de trabalho do processador.

1.3 OBJETIVO GERAL

Desenvolver uma interface de rede para redes-em-chip, utilizando a abordagem *hardware/software*, dedicada ao gerenciamento das comunicações intra-chip de maneira autônoma. Parte das ações de comunicação disponíveis tem como guia a especificação Message-Passing Interface 3.0, outras tem em mente as necessidades observadas no uso de redes-em-chip, algo não previsto na discussão do padrão citado.

1.4 OBJETIVOS ESPECÍFICOS

Para alcançar o objetivo foi necessário concluir os seguintes objetivos específicos:

a) Implementar uma interface de rede sintetizável em nível RTL utilizando uma linguagem HDL;

b) Identificar e disponibilizar um conjunto de operações de comunicação baseados no padrão MPI 3.0 compatíveis às necessidades das redes intra-chip;

c) Propor operações de comunicação pertinentes ao ambiente de comunicação intra-chip não previsto na discussão do padrão MPI 3.0;

d) Desenvolvimento de um método para passagem de parâmetros entre dispositivos finais de comunicação e a interface de rede;

e) Analisar vantagens e desvantagens no uso de interfaces de redes conjugadas às redes-em-chip;

f) Analisar a aplicabilidade do paradigma de passagem de mensagens para sistemas de comunicação baseados em redes-em-chip;

1.5 MÉTODO DE PESQUISA

Este tópico discute os aspectos metodológicos utilizados na presente dissertação, como a caracterização da pesquisa e seus procedimentos.

1.5.1 CARACTERIZAÇÃO DA PESQUISA

Conforme apresentado por Silva e Menezes (2001) pesquisa é um "conjunto de ações, propostas para encontrar a solução para um problema, que têm por base procedimentos racionais e sistemáticos". Este autor ainda descreve que as pesquisas podem ser categorizadas com base na natureza ou finalidade (pesquisa básica e aplicada), no objetivo (exploratória, descritiva e explicativa), na análise do problema (qualitativa e quantitativa) e nos procedimentos técnicos empregados (pesquisa bibliográfica, pesquisa documental, pesquisa experimental, levantamento, estudo de caso, *survey*, pesquisa *expost-facto*).

Esta dissertação caracteriza-se, em relação à natureza da pesquisa, como sendo uma aplicada pois objetiva gerar conhecimentos, para aplicação prática, dirigidos à solução de problemas específicos (SILVA; MENEZES, 2001) como a caracterização e implementação de uma interface de rede para redes-em-chip aplicáveis em MPSoCs *softcores*.

No que tange à abordagem, é qualitativa e quantitativa. A perspectiva quantitativa, de acordo com Silva e Menezes (2001), considera que "tudo pode ser quantificável, o que significa traduzir em números opiniões e informações para classificá-las e analisá-las". Assim sendo, nesta pesquisa, a abordagem quantitativa é destacada na análise latência incrementada ao sistema, área em chip, potência utilizada e *clock* máximo de operação. Isto é, fatores que impactam no desempenho do sistema que emprega esta solução. No tocante a análise qualitativa, tem-se o levantamento das soluções de interfaces de rede existentes para embasar a escolha de técnicas adequadas aplicáveis na implementação de uma interface adaptável ao processamento paralelo com base no paradigma de passagem de mensagens.

Porém, nas pesquisas realizadas nas bases de dados disponíveis (IEEE-xplore, ACM, portal de periódicos da CAPES, Google *Scholar*, teses e dissertações), apresenta-se com características do tipo exploratórias, dado que a pesquisa exploratória visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses, abrangendo o levantamento bibliográfico, bem como análise de exemplos que estimulem a compreensão (GIL, 2010).

A natureza exploratória da pesquisa é verificada visto que, para definição dos serviços específicos de multiprocessamento, é realizado um levantamento bibliográfico de soluções já praticadas na área. A pesquisa exploratória promove a familiaridade com

o problema objetivando torná-lo explícito ou a construir hipóteses. Assim, é por meio do contato com experiências práticas no campo pesquisado, reconhecendo soluções já aplicadas por outros pesquisadores, que é possível estimular a compreensão acerca do tema (GIL, 2010).

1.5.2 PROCEDIMENTOS DA PESQUISA

Os procedimentos para pesquisa foram divididos em cinco etapas. A primeira etapa consiste em levantamento bibliográfico com o objetivo de elencar os principais autores considerados nesta dissertação, elencando vantagens e desvantagens das técnicas propostas já publicadas por estes. Essa pesquisa é balizada em publicações recentes da área divulgadas em congressos, livros, teses e dissertações. Na segunda etapa o padrão MPI 3.0 é analisado verificando possíveis soluções empregáveis às aplicações sobre redes-em-chip. A terceira etapa abrange a proposição de uma técnica para permitir a interface entre a rede e o processador sem promover mudanças estruturais em nenhum dos componentes utilizados. A quarta etapa é a implementação de um modelo físico utilizando a linguagem VHDL. A quinta etapa consiste em analisar aspectos de *design* do circuito como área em *chip*, potência requerida, latências incluídas no processo e a empregabilidade da MPNI para promover abstração necessária ao processador *softcore* SICXE utilizando a rede-em-chip SoCIN-*fp*.

1.6 ORGANIZAÇÃO DA DISSERTAÇÃO

Este trabalho está estruturado em capítulos. Após este capítulo introdutório, que trata da contextualização do tema, da problemática, dos objetivos, bem como apresenta a organização da dissertação, tem-se o Capítulo 2, dedicado à revisão da literatura, acerca dos conceitos teóricos relativos à Message-passing Interface e Redes-em-chip. O Capítulo 3 lista os principais autores considerados neste trabalho, ressaltando as características marcantes de suas propostas, pontos fortes, fracos e técnicas identificadas como interessantes para esta pesquisa.

O Capítulo 4 discorre acerca da interface de rede-em-chip MPNI abrangendo todas as suas dimensões: arquitetural, acesso a memória, serviços disponíveis na arquitetura e métodos para solicitação de serviços. Os resultados de síntese do circuito e as latências inseridas no processo de comunicação são avaliados no Capítulo 5. O Capítulo 6 expõe as conclusões, limitações e recomendações para trabalhos futuros. Além dos capítulos elencados, fazem parte deste trabalho: as referências bibliográficas

consultadas, bem como o apêndice A referente à arquitetura interna da interface de rede.

2 REFERENCIAL TEÓRICO

São apresentados nesta seção alguns tópicos de que suportam um melhor entendimento deste trabalho. Primeiro será abordado o padrão MPI 3.0, detalhando suas premissas e soluções empregáveis na computação paralela pelo uso do paradigma de passagem de mensagens. Na subseção seguinte será feito um breve estudo das redes-em-chip, tais como características, vantagens e desvantagens conhecidas. A seguir é apresentada a rede *System-on-Chip, Interconnection Network fully parameterizable* (SoCINfp), esta rede utiliza o roteador *Parameterizable Interconnection Switch* (ParIS) e será útil na execução dos testes da interface de rede proposta.

2.1 MESSAGE-PASSING INTERFACE

O *Message-passing Interface* é considerado um padrão comercial para o desenvolvimento de projetos que necessitem trocar mensagens entre pontos distintos do sistema. Este padrão foi definido através de um processo aberto por uma comunidade de empresas desenvolvedoras, cientistas da computação e desenvolvedores de aplicações que envolvem computação paralela. O padrão propõe a definição clara de rotinas que podem ser implementadas eficientemente tanto em *hardware* quanto em *software* de maneira a permitir escalabilidade do sistema que o utilize. O MPI estabelece, em termos práticos, uma base para desenvolvimento de código portátil, eficiente e flexível. Atualmente o padrão MPI especifica mais de 300 tipos de operações de comunicação entre pontos de um sistema.

Conforme o manual MPI 3.0 (FORUM, 2012), o MPI oferece algumas características importantes aos sistemas de computação paralela que o segue. Permite comunicação eficiente, transferência de dados entre memórias distribuídas, permite sobreposição de computação e comunicação, além de distribuir processamento para coprocessadores. Pode ser usado em um ambiente heterogêneo, ou seja, sistemas onde os coprocessadores são de diferentes modelos e fabricantes. Propõe uma comunicação confiável, deixando para os subsistemas a tarefa de lidar com as falhas na comunicação decorridas. Define uma interface que pode ser projetada por diferentes desenvolvedores em distintas plataformas com mínimas mudanças nestes sistemas.

O texto completo do padrão MPI 3.0 dispõe de vários exemplos utilizando os mais distintos comandos em rotinas desenvolvidas com o objetivo de explicar a dinâmica das operações disponíveis. A redação a seguir foi baseada no referido documento,

entretanto volta a atenção para os aspectos importantes do padrão que serão utilizados no desenvolvimento do sistema *hardware/software* descrito nesta dissertação.

2.1.1 Comunicação Ponto-a-Ponto

Envio e recebimento de mensagens por processos é a base de qualquer sistema MPI. As operações básicas neste contexto são as *SEND* e *RECEIVE*. Uma operação *SEND* requer a determinação de um *buffer* na memória de onde os dados serão enviados ao destino. O tamanho desta região de memória deve ser definido a fim de permitir o envio da correta quantidade de dados. São informados, também, um destino e o grupo de comunicação a qual o processo está ligado. Este grupo deve conter os dispositivos comunicantes, de outra forma, não é possível a passagem de dados entre origem e destino.

Para todo dado gerado por uma operação *SEND* é necessário a existência de uma operação *RECEIVE* homóloga no destino. Este procedimento minimamente requer, em complemento ao *SEND*, informações de processo destino, *buffer* na memória para escrita dos dados recebidos e grupo de comunicação no qual origem e destino estão inscritos.

O MPI descreve ainda um *tag* (número ordinal que especifica o conjunto de dados transferidos) e o tipo de dado comunicado. Porém, aqui objetiva-se uma abordagem simplista dos comandos *SEND* e *RECEIVE*, sendo uma adaptação do padrão MPI 3.0 com o objetivo de explicar os parâmetros mínimos para a utilização nesta dissertação.

Por fim, o mecanismo de *SEND/RECEIVE* ponto-a-ponto suporta "*loop* de dados", isto é, destino e origem é o mesmo comunicador. Em MPI, porém, não é seguro realizar tal operação com *SEND/RECEIVE* bloqueantes. Esta ação pode levar a falhas por *deadlock*.

2.1.2 Comunicação Bloqueante e Não-bloqueante

Em comunicação por passagem de mensagens existem necessidades específicas. É o caso do bloqueio ou não do processo solicitante da comunicação. Há aplicações em que o processo seja bloqueado a fim de evitar execução de código antes que a mensagem seja efetivamente enviada. Em outros casos, o processo que solicitou a comunicação pode continuar executando enquanto a mensagem é encaminhada. Estas duas modalidades são definidas pelos termos bloqueante e não-bloqueantes, respectivamente. A técnica de bloqueio é aplicada àquelas operações que envolvem envio de dados.

É certo que o desempenho de muitos sistemas de computação paralela pode ser melhorado pela sobreposição de computação e comunicação. Isto é verdade em sistemas onde a comunicação pode ser executada de forma autônoma por um controlador inteligente (FORUM, 2012). Esta característica é conseguida por comunicações não-bloqueantes. Em MPI o bloqueio de processos possui quatro modos: *standard*, *buffered*, *synchronous* e *ready*.

Conforme é descrito pelo padrão, uma chamada de *SEND* não-bloqueante inicia uma operação de envio de dados, mas não é suficiente para completá-la. A chamada de *SEND* pode terminar antes mesmo da mensagem ser devidamente entregue para o destino. Similarmente, uma chamada de *RECEIVE* não-bloqueante pode terminar sem que os dados tenham sido recebidos pelo destino que executa tal chamada. Assim, para ambos casos, as conclusões das operações estão condicionadas a chamadas separadas para certificar se a operação de *SEND*/*RECEIVE* foi concluída propriamente.

Exceto no modo *ready* onde as chamadas *SEND* requerem uma confirmação de que o destino já aguarda pela operação *SEND* para que esta seja completada na origem, uma chamada *SEND* pode ser concluída quando os dados postados são retirados do *buffer* de envio. A depender do modo de operação, este evento pode ter outros significados. Se o modo é síncrono, a chamada *SEND* só é concluída quando a respectiva chamada *RECEIVE* é iniciada no destino e o remetente é informado deste evento. Caso o modo seja *buffered*, operações *SEND* serão completadas independentemente da existência de uma chamada *RECEIVE* no destino. Para isso, basta que o *buffer* de envio de dados seja esvaziado na origem. No modo *standard*, a chamada retorna depois que os dados são copiados no *buffer*, não precisando este ser esvaziado.

Perceba que, pelo padrão, um *SEND* não-bloqueante pode ser completado por uma rotina *RECEIVE* bloqueante e vice-versa.

2.1.3 Grupos de Comunicação

Os grupos de comunicação são interessantes no tocante a troca de mensagens. Eles delimitam quais processos estão ligados e se permite uma organização do sistema em diferentes contextos de comunicação. Isto é, um mesmo conjunto de elementos processantes podem compor diferentes grupos sobre contextos de comunicação distintos. Um grupo de comunicação define um universo de comunicação: mensagens são sempre recebidas e enviadas dentro de um mesmo contexto de comunicação.

O conjunto de elementos de um grupo é identificado por um *rank* específico. Este número ordinal identifica cada componente do grupo a fim de permitir a determinação de destino da mensagem dentro de um mesmo grupo. Em MPI, inicialmente todos os

elementos processantes compõe um mesmo grupo, o chamado MPI_COMM_WORLD. Neste, cada processo é definido por um dado *rank* único.

2.2 REDES-EM-CHIP

As redes intra-chip surgiram com o objetivo de prover comunicação escalável, não bloqueante, ser tolerante às falhas e prover comunicação seletiva (ZEFERINO, 2003). A utilização deve ser avaliada considerando vantagens e desvantagens para que haja um correto dimensionamento da rede para a aplicação em vista. Assim, na utilização de NoCs, é preciso levar em consideração algumas dessas características, listo separadamente como vantagens e desvantagens na utilização de NoCs.

Vantagens:

1. Comunicação não-bloqueante: representa uma vantagem desse método de comunicação, pois pode permitir que vários dispositivos em uma rede se comuniquem sem que nenhum dos fluxos de dados afete o desempenho dos outros (ZEFERINO, 2003);
2. Escalabilidade: não possui limite de dispositivos a serem conectados, não impactando na latência de maneira a impedir a sua utilização (BENINI; MICHELI, 2002; ZEFERINO, 2003);
3. Reusabilidade: permite o reuso da mesma estrutura de comunicação para projetos distintos (MATOS, 2010);
4. Comunicação seletiva: podendo a largura de banda ser adequada para cada tipo de informação (ZEFERINO, 2003).
5. Uso de conexões ponto a ponto curtas: diminui os efeitos de capacitâncias parasita devido a alta frequência de chaveamento (BENINI; MICHELI, 2002);
6. Alta disponibilidade da conexão: permite caminhos alternativos caso partes da rede se torne permanentemente ou temporariamente indisponível;
7. Confiabilidade da comunicação: permite suporte a codificação do canal e, consequentemente, dar garantias de integridade da comunicação entre destinos;

Desvantagens:

1. Área em chip: roteadores devem ser otimizados, de outro modo eles oferecerão expressivo consumo de área em chip, pois a sua instanciação ocorrerá tantas vezes quanto necessário à expansão da rede (GUERRIER; GREINER, 2000);

2. *Overhead* da comunicação: produzido pelo método de codificação do canal e protocolos utilizados no gerenciamento dos pacotes (ZEFERINO, 2003);

3. Interfaces de rede: necessidade do uso de interfaces para executarem serviços de gerenciamento da comunicação e tornar o processo transparente aos dispositivos que as utiliza (MATOS, 2010);

4. Latência: produzida pelas recorrentes retransmissões e esperas dos pacotes por recurso para serem encaminhados (MATOS, 2010);

5. Complexidade: devido aos diversos aspectos que devem ser endereçados no projeto de sistemas que utilizem esse método de interconexão (MATOS, 2010).

2.2.1 Roteador

O principal componente em uma rede intra-chip é o roteador. Este representa um monômero, isto é, uma pequena parte do todo, composto por inúmeras outras partes iguais. O roteador é responsável por receber o pacote do *Intellectual Property* (IP) local e encaminhar, de acordo com seu cabeçalho, para as portas de saída da rede (exemplo: norte, sul, leste, oeste), em direção ao destino.

Um roteador típico é composto por: *buffers*, meio de interconexão, enlaces e árbitros. O *buffer* é geralmente uma FIFO (acrônimo para *First In, Firt Out*) onde os pacotes são colocados e retirados com objetivo de aguardar a vez para serem encaminhados pela rede. Os *buffers* produzem o efeito de memorização das redes e a sua profundidade varia de acordo com o esquema em uso. O meio de interconexão responde pela ligação entre os *buffers* de cada porta. O meio mais comum de interligação é o *crossbar* por ser muito eficiente em termos de área e capacidade de realizar conexões ponto-a-ponto entre um número relativamente reduzido de comunicadores. Os árbitros fazem o encaminhamento dos pacotes do *buffer* de chegada para uma porta de saída. São estes os responsáveis pela solução de problemas de condições disputa entre os pacotes. Por fim, temos os enlaces, responsáveis por conectar as portas (norte, sul, leste, oeste, etc.) dos roteadores ponto-a-ponto entre si.

2.2.2 Topologias

As redes-em-chip também são definidas pelas formas em que os roteadores estão ligados entre si. Essa formação pode ser representada tipicamente por grafos (ZEFERINO, 2003), onde os nós representam os roteadores e as arestas, os enlaces que os conectam.

As topologias típicas utilizadas por NoCs são: a grelha 2-D n-dimenssional, o toróide e o hypercubo 3-D (ZEFERINO, 2003). Na figura 1 estão demonstradas as topologias tipicamente encontradas em redes intra-chip.

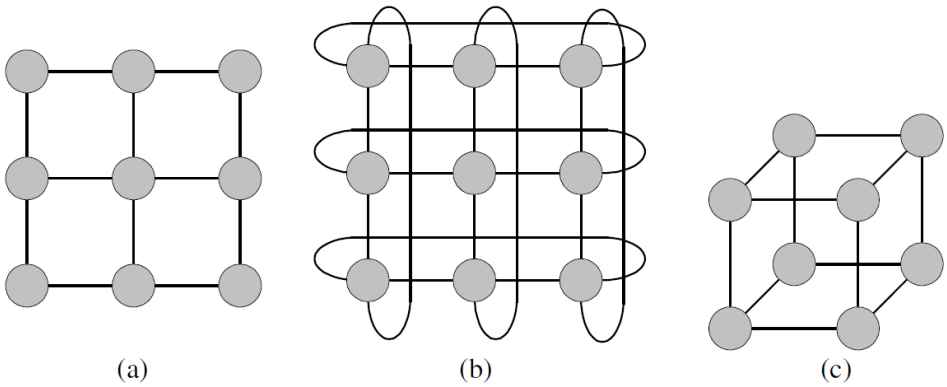


Figura 1 – Topologias diretas: (a) grelha 2-D, (b) toróide 2-D e (c) hypercubo 3-D

Fonte: Adaptado de Zeferino (2003)

Existem ainda duas classes de redes distintas: as diretas e as indiretas. As primeiras estão associadas a estruturas onde cada roteador está ligado a um destino local e entre si através de enlaces. A comunicação dos pacotes é realizada através de *hops* pela estrutura da rede, sem sofrer interferência dos dispositivos locais. Esse é o tipo de rede que será utilizado neste trabalho. Em contrapartida, as topologias indiretas são montadas conectando diversos roteadores entre si, sem que estes possuam necessariamente dispositivos finais de comunicação localmente. Esse é o caso das redes multiestágio usadas em telecomunicações por sua eficiência e versatilidade para pequenas redes.

2.2.3 Controle de Acesso

Os pacotes que trafegam pelas redes-em-chip comumente entram em condições de disputa por um canal de comunicação ou *buffer* que eventualmente já está sendo utilizado por outro pacote. Essas situações devem ser gerenciadas através de determinadas regras,

permitindo que os pacotes sejam encaminhados sem que ocorram perdas de pacotes no caminho ao destino. Essas regras, ou mais comumente chamadas de políticas (ZEFERINO, 2003), são o controle de fluxo de dados em uma rede.

Exitem alguns tipos de controle de fluxo utilizados em NoCs, porém os mais conhecidos são o *handshake* e o baseado em créditos. O controle de fluxo baseado em *handshake* é realizado através de três etapas. A primeira via é usada para encaminhamento dos dados, a segunda via informa a validade dos dados ao receptor. A terceira via é utilizada pelo receptor para informar que o recebimento já foi concluído. O controle de fluxo baseado em créditos utiliza, também, três vias. A diferença é que utiliza-se a terceira via para informar o emitente a quantidade de créditos disponíveis, isto é, a quantidade de espaços no *buffer* que sobraram para serem utilizados.

2.2.4 Memorização

Os *buffers* são necessários em redes intra-chip por basicamente manterem a persistência dos pacotes durante um período de disputa por algum canal já ocupado. Este papel comumente é desempenhado por FIFOs, por serem de simples implementação. As FIFOs, porém, possuem desvantagem quanto ao encaminhamento de pacotes na rede, problema conhecido como *Head-of-Line blocking* (HOL) . Este é um fenômeno observado em pacotes que, impedidos de avançarem para determinada porta de saída, acabam por comprometer todos que estão na mesma fila e que sairiam por outras portas já desocupadas.

Como uma saída ao HOL existem outras técnicas baseadas em FIFOs modificadas. É o caso da *Statically Allocated, Fully Connected* (SAFC), *Statically Allocated Multi-Queue* (SAMQ) e *Dynamically Allocated Multi-Queue* (DAMQ). Essas variações de FIFO visam resolver os problemas causados por pacotes bloqueados na cabeça das filas (ZEFERINO, 2003). Na figura 2 é mostrado o esquema das FIFOs modificadas.

Há, também, porém menos comum, a utilização de *buffer* centralizado compartilhado (ZEFERINO, 2003). São chamados de *Centrally-Buffered, Dynamically-Allocated* (CBDA), em outras palavras é possível alocar memória dinamicamente entre os canais do roteador (ZEFERINO, 2003). Essa abordagem garante uma melhor utilização da memória disponível no roteador devido a otimização na utilização da memória disponível.

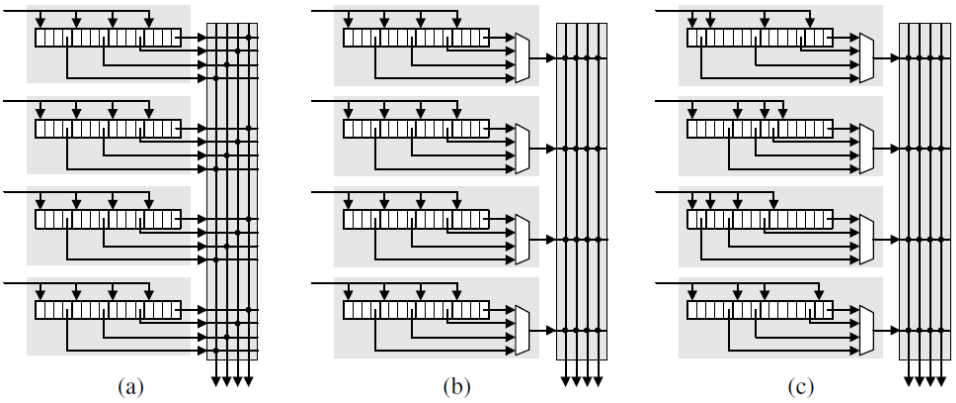


Figura 2 – FIFO modificada: (a) SAFC, (b) SAMQ e (c) DAMQ

Fonte: Adaptado de Zeferino (2003)

2.2.5 Roteamento

O roteamento é a atividade de planejar o caminho que o pacote percorrerá dentro da rede da origem ao destino. Os pacotes passam de roteador a roteador através de *hops*. A decisão de qual método de roteamento utilizar depende da topologia utilizada e visa garantir conectividade, adaptatividade, tolerância a falhas, ausência de *deadlock*, *starvation* e *livelock* (ZEFERINO, 2003). Em redes, o problema de *deadlock* é observado quando pacotes tentam acessar simultaneamente recursos aos quais originalmente cada elemento já está ocupando. Assim, ocorre uma dependência mútua onde cada pacote esperará indefinidamente pelo outro e, dessa forma, não avançam na rede. *Starvation* pode ocorrer em esquemas de arbitragem onde pacotes são privilegiados em detrimento de outros. Isto é, havendo dois fluxos contínuos de informação para um mesmo destino onde as prioridades dos pacotes são diferentes, o fluxo de menor prioridade esperará indefinidamente pelos pacotes do fluxo de maior prioridade. Por fim, o problema de *livelock* ocorre quando um pacote permanece trafegando na rede devido aos canais que o ligam ao nó destino nunca estejam disponíveis.

O roteamento pode ser classificado com base em alguns critérios, tais como: momento da realização de roteamento, local onde as decisões de roteamento são tomadas, adaptatividade, número de destinatários e método de implementação (baseado em tabela ou máquina de estados) (ZEFERINO, 2003). O roteamento mais comum em redes-intrachip é o XY. Esse é um algoritmo determinístico, isto é, garante que o pacote que sai de uma dada origem passe sempre pelo mesmo caminho. A decisão de qual caminho é utilizado pelo pacote é tomado em cada roteador por onde trafega, sendo classificado como distribuído. Esse método de roteamento possui qualidades importantes que o torna comum em aplicações com NoC, pois garante ausência de *deadlock* e *livelock*.

2.2.6 Arbitragem

Frequentemente, em redes-em-chip, pacotes chegam em uma condição de disputa pelos canais de comunicação. Para resolver esses conflitos no roteador existem os árbitros. Esses basicamente determinam qual *buffer* de entrada terá permissão para transmitir o pacote à porta de saída primeiro.

A arbitragem pode trabalhar em dois contextos: centralizado e distribuído. Na implementação centralizada o roteamento e a arbitragem são realizados por um único módulo. Essa abordagem tem vantagens quando o método de roteamento pode resolver condições de disputa por uma porta de saída. Em contrapartida demanda mais tempo já que as duas tarefas são executadas sequencialmente (ZEFERINO, 2003). Isto é, tem-se que processar o roteamento resolvendo condições de disputa, para, em seguida, ser decidido conflitos remanescentes entre pacotes.

A arbitragem distribuída, mais comum nas redes intra-chip, são realizadas sem nenhum vínculo com o roteamento. Essa técnica age apenas na distribuição do recurso aos pacotes que desejam utilizá-lo. Assim, no canal de entrada temos o roteamento sendo realizado, enquanto que, no canal de saída, temos o árbitro decidindo qual pacote será transmitido por vez.

Existem, também, mecanismos de prioridade diferenciados que faz com que um pacote seja encaminhado primeiro frente a outros que estão no *buffer* de entrada. Como consequência disso, é possível que pacotes fiquem parados por tempo indeterminado em certos pontos da rede, esperando para serem encaminhados. Essa situação é conhecida como *starvation*. É importante frisar que bons métodos de arbitragem devem impedir esse problema.

2.2.7 Chaveamento

Assim como nas redes de computadores, as redes-em-chip possuem dois métodos de chaveamento: o de circuitos e o de pacotes. Naturalmente cada uma dessas abordagens possui vantagens e desvantagens. Como as NoCs geralmente utilizam o chaveamento de pacotes, esse será o método que abordaremos em mais detalhes.

Os chaveamentos mais conhecidos são o *Store-and-forward* (SAF), o *Virtual Cut-through* (VCT) e o *Wormhole*. A diferença prática entre esses métodos de chaveamento consiste na maneira de armazenar os pacotes, ou parte deles, durante a transmissão. Pondo em foco essa questão, a memorização da rede pode ser otimizada de acordo com as características da comunicação, diminuindo a necessidade de *buffers* profundos e,

consequentemente, reduzindo área em chip.

Roteadores que implementam o chaveamento SAF, por exemplo, recebem todo o pacote antes que o mesmo seja encaminhado para a porta de saída e continue a rota ao destino. O método de chaveamento VCT (KERMANI; KLEINROCK, 1979) faz com que o pacote possa desviar do *buffer* de entrada, repassando diretamente para a porta de saída, caso esta esteja liberada. Note que, caso a porta de saída esteja sendo utilizada, o pacote deverá ser armazenado da mesma forma. Assim o VCT pode funcionar, no pior caso, como um SAF.

O chaveamento *wormhole*, proposto inicialmente por Dally e Seitz (1986), é o mais utilizado nas redes intra-chips. É uma variação do método VCT puro (ZEFERINO, 2003) e permite que o pacote, agora dividido em *Flow Control Units* (FLITs) , menor parte do pacote que pode ser comunicado separadamente pela rede), seja encaminhado. Assim, o pacote pode estar divido entre diversos roteadores da rede, utilizando de maneira otimizada a memória disponível. Outro fato é que, para manter a sequência de recebimento dos FLITs, eles trafegam da origem ao destino pelo mesmo caminho do FLIT de cabeçalho do pacote. Além disso, os pacotes são encaminhados um por vez por canal, para que não ocorra mistura dos FLITs de pacotes distintos. Isso resulta em maiores riscos de *deadlock*, porém podem ser controlados através do uso de canais virtuais (ZEFERINO, 2003).

2.2.8 A Rede Intra-chip SOCINfp

A SoCINfp é uma rede-em-chip descrita no nível de transferência entre registradores (RTL - *Register-transfer Level*) e parametrizável em tempo de projeto. Esta é formada pelo roteador ParIs. Esta arquitetura de roteador trouxe a possibilidade de parametrização de algumas características a fim de otimizar custo e desempenho da implementação da rede intra-chip. O roteador ParIS pode ser conectado em topologia direta nos formatos grelha 2-D e toróide 2-D. O controle de fluxo pode ser baseado em *handshake* ou crédito. O roteamento pode ser escolhido entre o determinístico XY e o parcialmente adaptativo *West-first*. A técnica de chaveamento é o *wormhole*. A arbitragem dinâmica distribuída é baseada em *round robin* e a memorização é realizada na porta de entrada do roteador (ZEFERINO; SANTO; SUSIN, 2004).

Para este trabalho, a topologia da rede SoCIN-*fp* foi montada usando o roteador ParIS na configuração grelha 2-D. O controle de fluxo escolhido foi o baseado em *handshake*. O algoritmo de roteamento, para garantir a inexistência de condições de *livelock* e *deadlock*, foi escolhido o roteamento determinístico XY. A arbitragem, com o intuito de assegurar inexistência de condições de *starvation* foi utilizado o algoritmo

round robin sem distinção de prioridade entre os pacotes e as FIFOs foram configuradas para profundidade de 4 FLITs. A figura 3 mostra um esquema da topologia da rede SoCIN-*fp*, uma rede 4x4, e o formato do pacote utilizado pela rede. No detalhe, dentro dos roteadores, pode-se perceber o endereço de cada roteador na estrutura.

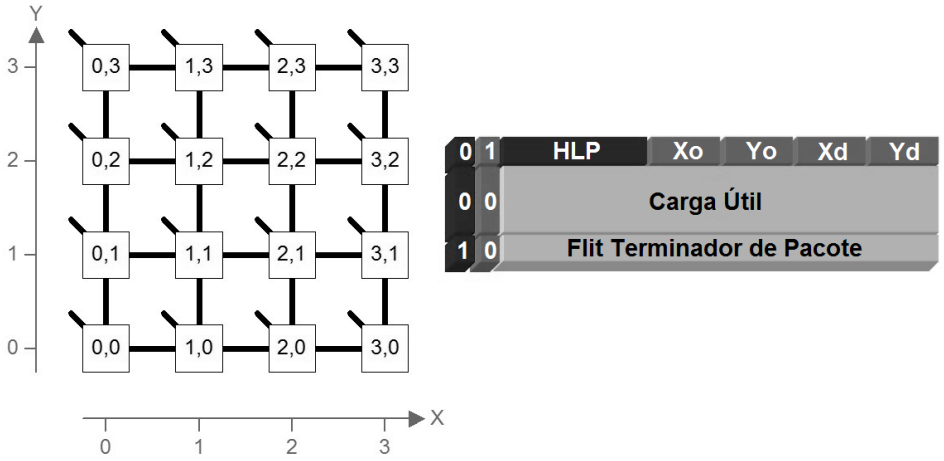


Figura 3 – Endereçamento e formato do pacote utilizado na rede SoCINfp

Fonte: Adaptado de Zeferino, Santo e Susin (2004)

O pacote contém n+2 bits, onde n é a largura do enlace físico. Os dois bits mais significativos são utilizados para controle do FLIT de dados são o *end of package*, indicando cabeça (01), corpo (00) e fim do pacote (10). O FLIT de cabeçalho contém informações do protocolo, chamados de *bits* de *Hight Level Protocols* (HLP), e endereçamento do pacote, chamados de *Routing Information Bits* (RIB). Os demais FLITs são utilizados para transportar dados pela rede (ZEFERINO; SANTO; SUSIN, 2004). Os bits de HLP serão, conforme é sugerido, utilizados pela interface de rede proposta nesse trabalho para endereçar aspectos de troca de informações de protocolo. Os bits de HLP, entretanto, em certos casos, não são suficientes, fazendo com que a interface de rede utilize os FLITs de *payload* para trocar as informações adicionais que não couberem no espaço dispensado ao HLP.

2.3 RESUMO DO CAPÍTULO

Neste capítulo foi revisado parte do padrão MPI 3.0, onde foram relacionadas algumas características interessantes ao desenvolvimento da interface de rede proposta nesta dissertação. Foram observadas, também, as características principais de uma rede-em-chip. Estes conceitos visam produzir uma fundamentação acerca do tema para posterior utilização desta técnica de interconexão no desenvolvimento deste trabalho.

3 TRABALHOS RELACIONADOS

Nesta seção são apresentadas algumas propostas de interface de rede. A primeira desenvolvida por Matos (2010), é uma interface parametrizável em tempo de projeto e genérica que visa interligar IPs tornando a comunicação transparente ao dispositivo final. Em seguida é abordada a proposta de Bhojwani e Mahapatra (2006) a qual demonstra uma interface para MPSoCs endereçando algumas características importantes nesses tipos de aplicações. Na proposta de Melo (2012) consideramos a perspectiva em camadas adotada pelo autor. Na discussão da proposta de Lee et al. (2008) identifica-se a simplificação das configurações da interface de rede e o que a torna eficiente neste processo. Por fim, a proposta de (JOVEN et al., 2008) é analisada em termos do uso personalizável do sistema completo: rede, interface de rede e processador, tendo em vista a especificação de todas as partes da solução de intercomunicação empregada.

3.1 PROPOSTA DE MATOS

A interface proposta em (MATOS, 2010) está sugerida no esquema da figura 4. Ela propõe uma via *fullduplex* de comunicação com o roteador e o *Processing Element* (PE), ou elemento processante, ou processador. Os dados recebidos da rede são desempacotados por uma unidade específica, fornecendo apenas os dados úteis para o elemento processante. O envio de dados à rede, porém, é condicionado à saída direta do elemento processante. Este, por sua vez, encaminha os dados para uma FIFO que serve como *buffer* para acoplar os fluxos de saída do PE com a capacidade de absorção de dados pela rede. Esta FIFO pode ser usada na entrada do PE, caso o elemento processante possua capacidade de consumo dos dados menor que o volume entregue pela rede.

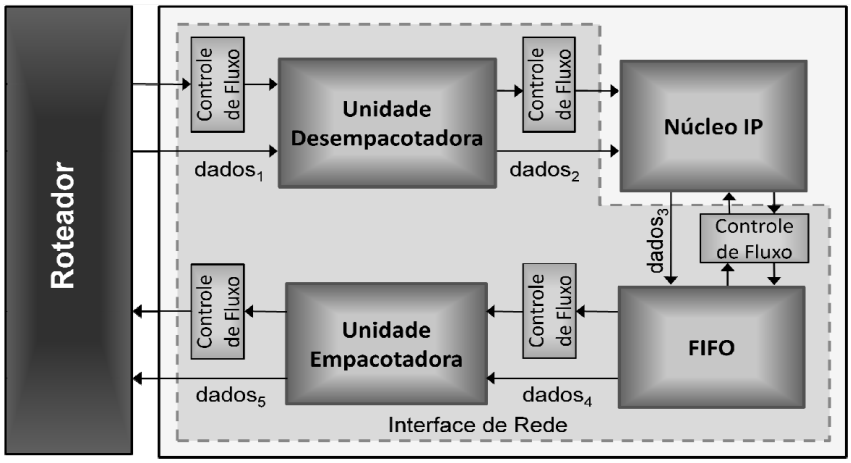


Figura 4 – Estrutura de blocos da interface de rede proposta por Matos (2010)

Fonte: Adaptado de Matos (2010)

Matos (2010) optou por desenvolver apenas as unidades de empacotamento e desempacotamento, realizando o controle de fluxo baseado em *handshake*. Este modelo faz o uso de duas FIFOs posicionadas na entrada e na saída do PE/IP. O papel dessa memorização é acoplar o fluxo de dados entre o dispositivo e a rede de interconexão. Dependendo da aplicação, as FIFOs utilizadas podem ser bastante profundas. Apesar de conferir a capacidade de comunicação bidirecional simultânea com a rede, o uso de FIFOs dedicadas para cada fluxo resulta em uma desvantagem. O esquema de FIFO utilizado por Matos (2010) trabalha exclusivamente com a entrada ou a saída, não podendo ser compartilhada entre ambos os fluxos. Essa abordagem pode provocar o uso não otimizado da memória existente no sistema.

O protocolo de troca de dados entre a interface de rede e o elemento processante baseia-se em dois canais de dados (entrada e saída da interface) com um controle de fluxo através de *handshake*. Isso significa que o elemento processante deverá ser modificado para trabalhar nesse contexto ou haverá a necessidade de um controlador para acoplar o dispositivo à interface de rede. Gerando, assim, necessidade de mais hardware e, consequentemente, mais dissipação de potência e consumo de área em silício.

Conforme descrito em (MATOS, 2010), o esquema em questão não suporta o gerenciamento das operações do protocolo de comunicação da rede (destino, origem, tipo de pacote, controle de erro, etc.). Tanto esta função como as possíveis operações conjuntas com os outros elementos da rede devem ser endereçadas pelos dispositivos conectados às interfaces propostas por ela.

Outro aspecto do esquema é que, para aplicações onde o elemento processante pudesse receber dados de diferentes pontos da rede, seriam necessários mais FIFOs de entrada para ordenarmos os FLITs que chegariam pela rede. Essas FIFOs deveriam ser reservadas para cada ponto de origem da rede, não podendo ser aproveitadas para outro fim senão o recebimento de FLITs de um determinado produtor de dados. É importante ressaltar, também, que, caso a aplicação sugira uma persistência dos dados, é obrigatório o uso de uma terceira memória dedicada à guarda desses dados para o PE.

É usado um FLIT específico para acondicionar todas as informações do protocolo de comunicação utilizadas. Este FLIT é definido pela interface de rede, porém é composto por informações de protocolo definidas estaticamente em tempo de projeto, não modificando o destinatário das mensagens enviadas pela rede em tempo de execução. É usado, também, um protocolo baseado em rótulos para ordenar e identificar as mensagens enviadas pela rede.

3.2 PROPOSTA DE BHOJWANI ET. AL.

Bhojwani e Mahapatra (2003) relatam que, apesar de existirem soluções como as redes em chip, elas precisam de interfaces especializadas que facilitem a empregabilidade

desse meio de interconexão. As vantagens inerentes à manipulação de pacotes, ao invés de comutação de circuitos, acarretam no emprego de empacotadores que aumentam ainda mais a latência dessa opção de interconexão de IP. Ele testa três arquiteturas de soluções para interface de rede: bibliotecas de *software*, módulo intra-IP e baseado em *wrapper*. Em (BHOJWANI; MAHAPATRA, 2003) é mostrado um comparativo dos resultados esperados para os três diferentes tipos de abordagem, seguindo a discussão dos *trade-offs* a serem resolvidos para a construção de interfaces de rede: área, latência, complexidade e flexibilidade. Para o desenvolvimento da interface proposta por ele, a *Core Network Interface* (CNI), é utilizado o modelo baseado em *wrapper*.

No desenvolvimento de uma interface CNI, Bhojwani e Mahapatra (2006) ressaltam a transparência ou não das operações de rede para o IP utilizado. Ressalta-se a vantagem da diminuição da latência, tendo em vista que o tempo de formação de pacote não seria computado na latência da rede. Desconsiderando, assim, atrasos produzidos pelo processo de empacotamento. Outro ponto é que a interface de rede é beneficiada em função da redução de complexidade. Em contrapartida existe o aumento da complexidade do IP em duas linhas: a primeira consiste em especificar os parâmetros de empacotamento e a segunda é a capacidade de programação mínima requerida para que o IP execute as operações de configuração do pacote. Esta desvantagem soma-se à necessidade de modificação do IP para abrigar as funcionalidades necessárias para as operações de rede. Na figura 5 é apresentada a estrutura da interface CNI, proposta em (BHOJWANI; MAHAPATRA, 2006).

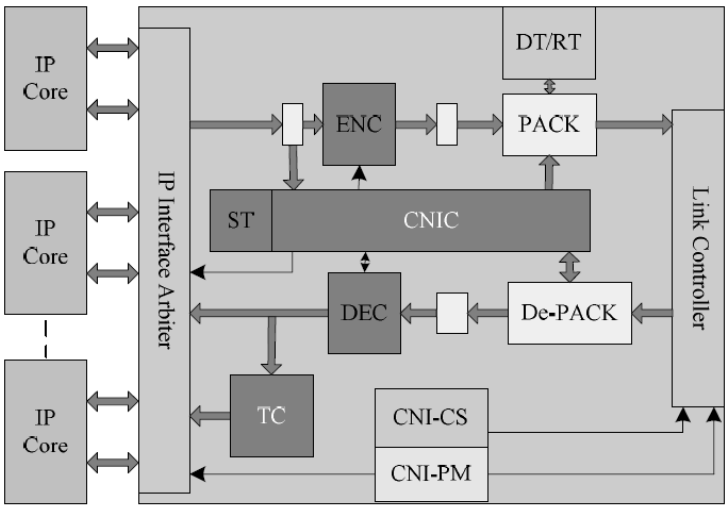


Figura 5 – Estrutura de blocos da interface de rede proposta por Bhojwani e Mahapatra (2006)

Fonte: Adaptado de (BHOJWANI; MAHAPATRA, 2006)

A estrutura é formada pelo árbitro que funciona como um *proxy* para comunicação entre os IPs conectados a este e à rede. O árbitro implementa uma interface básica do protocolo *Open Core Protocol 2.0* (OCP 2.0). Os blocos *Encoder* (ENC) e *Decoder*

(DEC) executam a codificação do canal. Essa funcionalidade pode ser desabilitada manualmente em fase de projeto, dependendo dos níveis de erros encontrados nos pacotes e no custo total do projeto em área e potência dissipada. O empacotamento e desempacotamento são realizados com base nos dois módulos *Packetizer* (PACK) e *De-packetizer* (De-PACK), respectivamente. O empacotamento é realizado com base em uma tabela de roteamento chamada de *Destination Table/Route Table* (DT/RT) que definem a tradução de endereços entre o barramento de IPs conectados à CNI e a rede. Existem, também, módulos específicos que podem ser habilitados como o *Core Network Interface - Communication Scheduler* (CNI-CS), o *Core Network Interface - Power Manager* (CNI-PM) e o *Test Controller* (TC). O módulo CS é responsável pelo gerenciamento dos canais virtuais da rede. O módulo PM gerencia a energia utilizada pelo PE/CNI, proporcionando a melhoria da eficiência energética do sistema. O TC oferece meios para teste e diagnóstico dos *cores* ligados à CNI.

Bhojwani e Mahapatra (2003) divide o processo de comunicação via pacote como um conjunto das seguintes atividades: preparação do pacote, transmissão do pacote e tratamento de pacote recebido. O primeiro estágio é contado desde quando o processador identifica a necessidade de comunicação com um componente externo até a entrega do pacote completo para ser enviado pelo roteador local. A fatia da latência para este estágio é reflexo da codificação utilizada e dos protocolos de comunicação a serem resolvidos. O estágio de tratamento do pacote recebido possuirá latência proporcional, pois executa basicamente a operação complementar do primeiro estágio. O estágio de transmissão do pacote é resultado dos atrasos inerentes à rede-em-chip utilizada, topologia, mecanismos de troca de mensagens, etc., não sendo objeto de análise neste trabalho.

A preparação do pacote no modelo de Bhojwani e Mahapatra (2003) segue os seguintes passos:

1. Traduz o endereço solicitado pelo PE em endereço real e localização da memória na rede;
2. Prepara o cabeçalho do pacote colocando o endereço dos dados e o roteador a ser acessado;
3. Examina a instrução que requer o acesso e seta as *flags* necessárias para essa instrução;
4. Põe o endereço efetivo dos dados a serem lidos no pacote;
5. Caso use algum código de detecção e correção de erros, calcula-se e insere no pacote;
6. Monta e entrega o pacote para a rede.

Na aplicação montada em (BHOJWANI; MAHAPATRA, 2003), é considerado um cenário de memória compartilhada onde o processador central acessa uma memória de endereçamento único espalhada por toda a rede. A natureza distribuída do espaço de memória é transparente ao processador central. Quando a execução de programa acessa uma posição de memória, a interface de rede é responsável por achar na rede a memória correspondente e executar a operação de transferência de operando, entregando o dado a ser processado para o PE central.

Os principais resultados obtidos por Bhojwani e Mahapatra (2003) foram a comparação das latências entre os três métodos propostos e a atenção para o custo em área especialmente requerida pela memória utilizada para suportar as bibliotecas de comunicação em relação ao custo de implementação dos dispositivos em hardware dedicado.

3.3 PROPOSTA DE MELO

Melo (2012) apresenta uma implementação de interface de rede para redes-em-chip organizada em camadas, conforme a Figura 6, que propõe um conjunto de serviços em *hardware* para auxiliar implementações de protocolos de comunicação de camadas superiores. Esta interface disponibiliza serviços de interligação entre um protocolo de barramento e uma rede-em-chip. Assim, como toda interface de rede, está disponível o serviço de empacotamento e desempacotamento de dados. Além disso, é previsto, também, suporte à técnicas de QoS (*Quality of Service*) disponíveis na rede SoCIN-Q (BEREJUCK; ZEFERINO, 2009). Adicionalmente é proposto um controle de integridade dos dados encaminhados e um sistema de codificação para reduzir chaveamento de *bits* entre *Flits* sucessivos transmitidos pela rede e, assim, economizar em uso de energia e de *buffers* a serem empregados pela rede. A interface de rede apresentada em (MELO, 2012) é batizada de XIRU (acrônimo da expressão: *eXtensible InteRface Unit*) .

A integridade dos dados é controlada pela interface por meio do uso de um *bit* de paridade em cada *Flit* e de um *Flit* terminador contendo o *checksum*. O serviço de codificação, conforme é comprovado pelo autor, requer um considerável uso de recurso em silício, sendo comparado à área requerida para o sistema de *buffers* da rede. Assim, é identificado um *trade-off* no tocante a utilização de sistemas de codificação mais eficientes em *hardware* e a utilização de *buffers* distribuídos pela rede.

A conexão entre a interface XIRU e núcleo processante é garantida por uso do protocolo AVALON, protocolo proprietário do fabricante ALTERA. Este, bastante difundido entre projetistas de circuitos integrados, é compatível com centenas de dispositivos e garante ao projeto desta interface de rede conectividade com os mais diferentes tipos de núcleo. A Figura 7 mostra o sistema exemplificado para a validação da referida interface de rede.

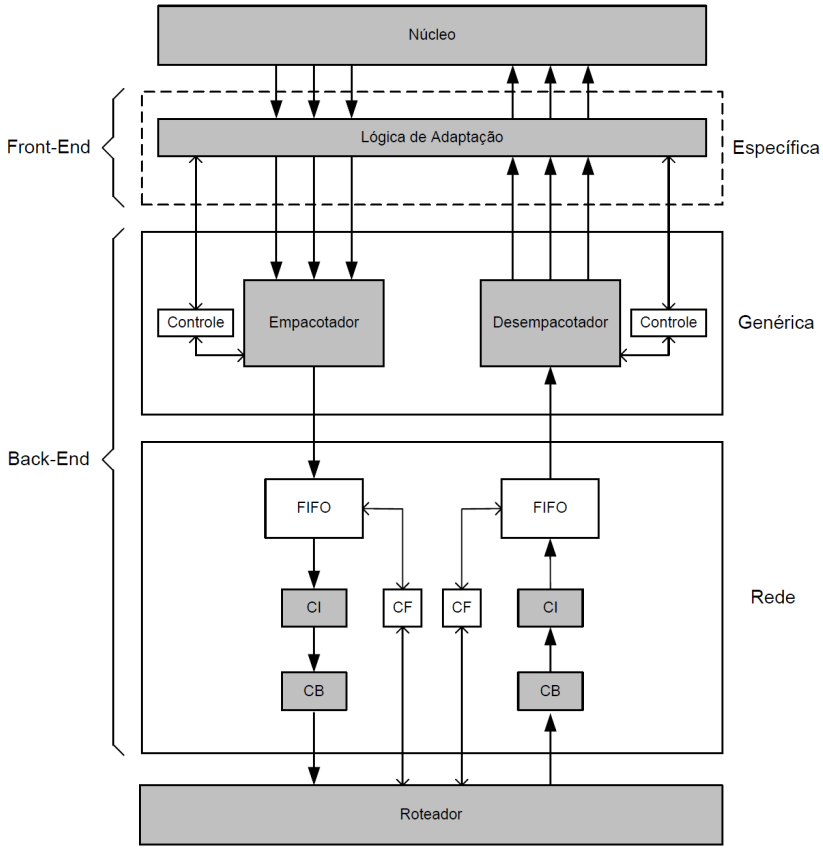


Figura 6 – Modelo em camadas proposto por Melo (2012)

Fonte: Adaptado de Melo (2012)

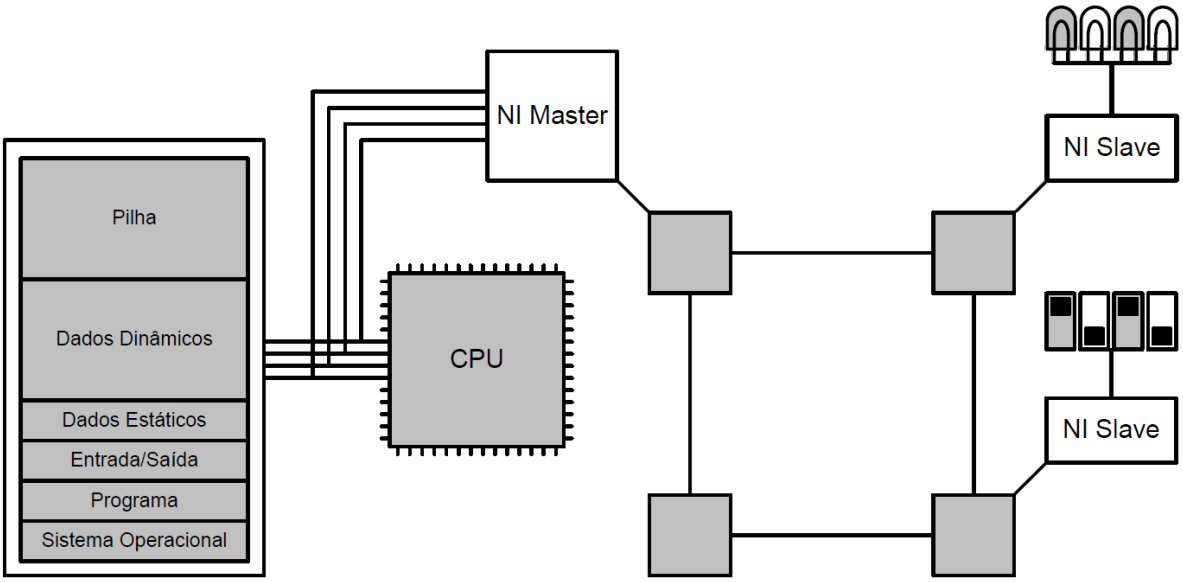


Figura 7 – Exemplo de sistema proposto por Melo (2012)

Fonte: Adaptado de Melo (2012)

No esquema da Figura 7 a CPU é uma instância do processador NIOS II conectado pela interface XIRU *Master*. A interface realiza as operações de tradução entre atividades do barramento AVALON e da rede-em-chip. Ligados em outros pontos da rede estão os sistemas de entradas e saídas conectadas por meio de interfaces XIRU *Slave*, trocando informação com o processador NIOS II.

3.4 PROPOSTA DE LEE ET. AL.

Lee et al. (2008) propõe uma interface de rede para suporte a projetos com o modelo de processador compatíveis com processadores *softcores* OpenRISC. Este processador é distribuído gratuitamente sob regras da GNU *Lesser General Public License* (LGPL). A Figura 8 mostra o esquema de blocos da interface montada em conjunto com um processador OpenRISC com memória de programa e dados separadamente. O processador configura a interface de rede por meio de um canal de controle específico que dá acesso a registradores de parametrização da interface.

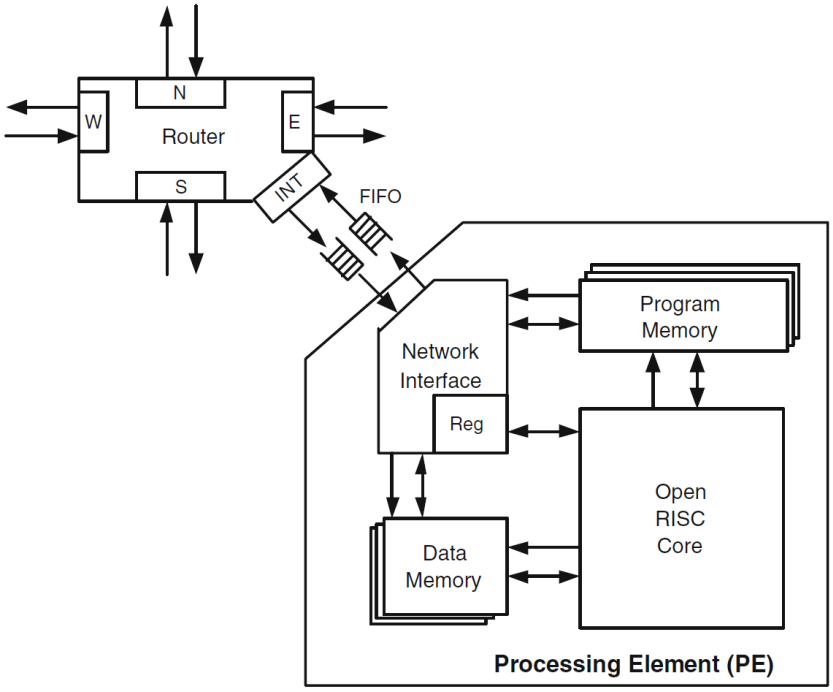


Figura 8 – Detalhe do sistema proposto por Lee et al. (2008)

Fonte: Adaptado de Lee et al. (2008)

A interface de rede consiste em um módulo empacotador, um módulo desempacotador e uma interface com o núcleo processante. Assim como as propostas de interfaces vistas, esta está localizada entre o roteador da NoC e o processador. Os registradores da interface são vistos através de um esquema de mapeamento em memória dos registradores de configuração. Isto é, os registradores estão ligados ao barramento do lado do processador sendo acessados por meio de endereços específicos.

A Figura 9 mostra a organização interna da arquitetura da interface. Nela é possível identificar registradores de *status*, de configuração de envios e configuração de recebimentos de dados provindos da rede. Estão presentes, também, os módulos de construção e tradução de cabeçalhos. Estes são responsáveis por criar a sequência de *bits* que identificará a mensagem na rede e passará informações de protocolo para garantir a entrega dos dados pela rede. O *Flit controller* responde pelas atividades de efetivar o empacotamento e desempacotamento, seja na entrada ou na saída da a rede-em-chip.

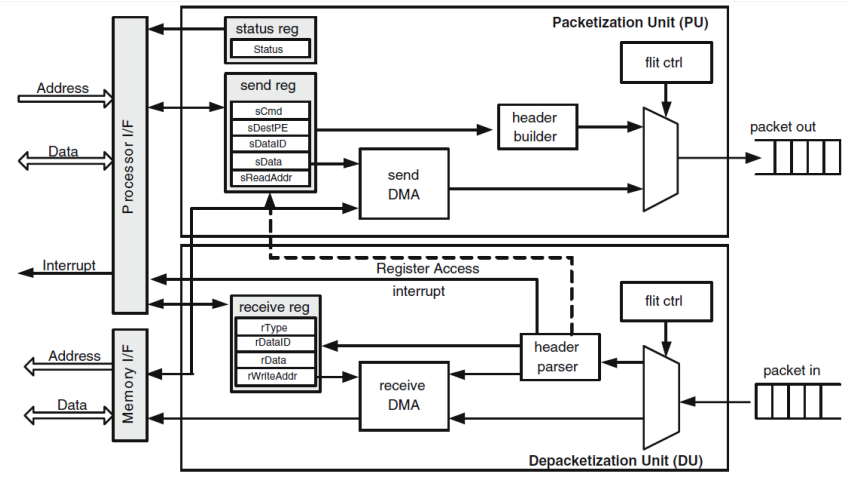


Figura 9 – Arquitetura da interface de rede proposta por Lee et al. (2008)

Fonte: Adaptado de Lee et al. (2008)

Esta interface propõe uma maneira simples de conectar núcleos. A proposta é conectá-los com um mínimo de remodelamento do processador integrado no projeto. Uma das características principais desta interface é a transferência de dados memória/rede e rede/memória realizadas em bloco por meio do mecanismo de acesso direto à memória (DMA), estas operações ocorrem em ciclo único, objetivando alto desempenho.

3.5 PROPOSTA DE JOVEN ET. AL.

A solução proposta por Joven et al. (2008) aplica uma metodologia automatizada de geração de um MPSoC completo, isto é, integração de IP *core*, rede-em-chip, interface de rede, *drivers* para a interface e *software* da aplicação para explorar o espaço de projeto e a potencialidade dos componentes empregados. Na medida em que seguem o fluxo de projeto conforme mostrado na Figura 10, os projetistas são guiados na tarefa de desenvolver um sistema que contemple a aplicação, os blocos IPs e os subsistemas de comunicação (rede-em-chip e interface de rede), seguindo uma abordagem *hardware/software* para o tratamento da comunicação entre estes blocos.

Para o desenvolvimento do *driver* que fará uso da interface de rede proposta, é disponibilizada uma API (*Application Program Interface*) denominada eMPI (*Embedded*

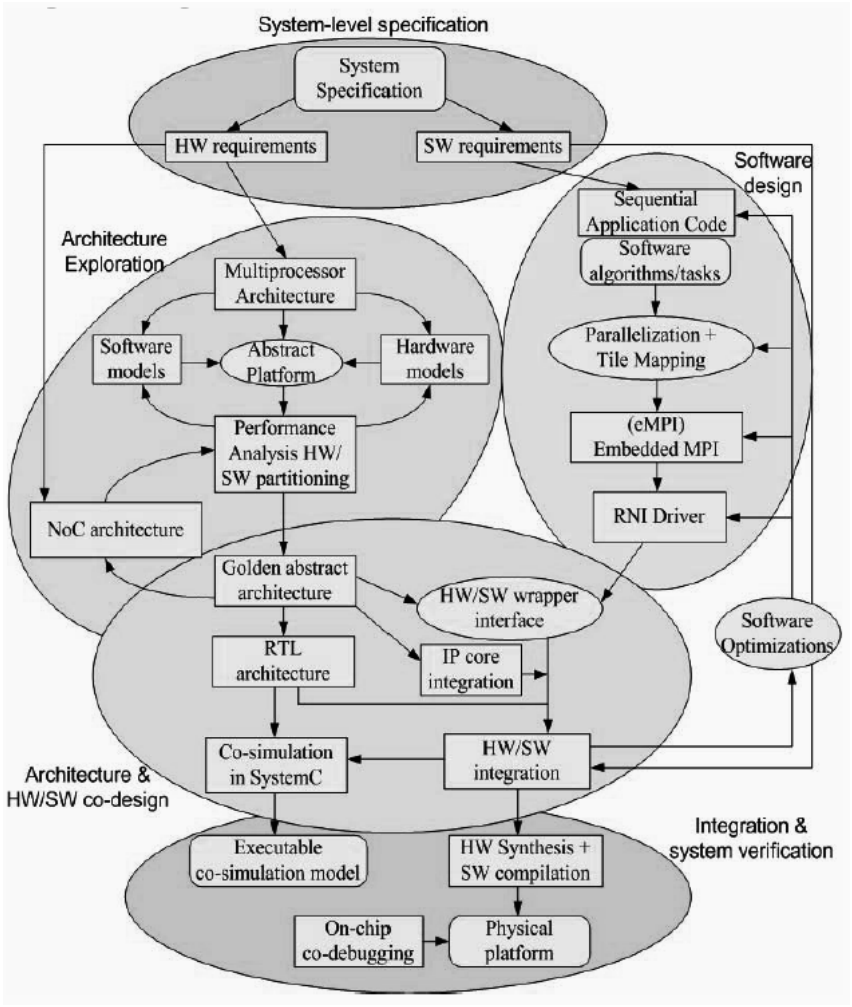


Figura 10 – Fluxo de projeto adotado por Joven et al. (2008)

Fonte: Adaptado de Joven et al. (2008)

Message-Passing Interface) . Esta visa homogeneizar as requisições e respostas às atividades de comunicação realizadas pela aplicação que é executada distribuidamente pelo sistema.

A API eMPI é um subconjunto das transações disponíveis no padrão MPI discutido em (WALKER; DONGARRA, 1996). Foram implementadas as funções eMPI_Init (inicialização de grupos de comunicação), eMPI_Finalize (encerramento de grupos de comunicação), eMPI_Comm_size (define o tamanho do grupo de comunicação), aMPI_Comm_rank (define o número ordinal que identifica o nó no grupo de comunicação), eMPI_Send (envio de mensagem com bloqueio do nó solicitante) e eMPI_Recv (recebimento de mensagem com bloqueio do nó solicitante).

A interface de rede utilizada por Joven et al. (2008) possui duas versões: *Master/Slave* e *slave*, conforme é visto na Figura 11-A. Pela proposta do referido autor, há duas formas de conectar uma rede-em-chip ao dispositivo final. Uma primeira forma é pelo uso da interface *Slave*, conforme visto na Figura11-B. Quando os dados são repassados para a interface *Slave*, estes são enviados para a rede diretamente, de outro modo, quando os dados surgem da rede, o processador é informado por *pooling* (tabela

contendo os dados recém-chegados) ou interrupção por meio de uma via especial de controle. Outra maneira de conexão é pelas interfaces *Mater/Slave*. O comportamento no envio dos dados é o mesmo realizado pela interface do tipo *Slave*, a diferença está no recebimento dos dados da rede. Ao receber os dados provindos da rede a interface toma o controle do barramento e escreve em espaço de memória predefinido pelo processador, notificando-o que os dados encontram-se disponíveis na memória local.

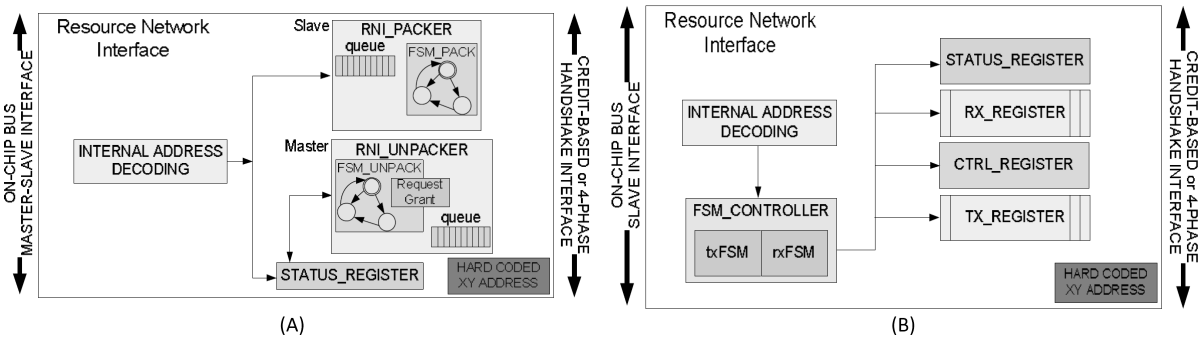


Figura 11 – Proposta de interface de rede de Joven et al. (2008)

Fonte: Adaptado de Joven et al. (2008)

A proposta sugerida por Joven et al. (2008) mostra-se interessante para solucionar problemas de interconexão entre o dispositivo final e a rede-em-chip. Esta abordagem possibilita a flexibilização do projeto para atingir os requisitos da aplicação. Entretanto, requer que o projetista lide com todas as premissas de um projeto de interconexão por rede-em-chip. Desde a especificação da rede-em-chip até a modelagem da API que é utilizada pelo código da aplicação do MPSoC.

Apesar de Joven et al. (2008) implementar transações do modelo MPI, eles utilizam a técnica de *mapping* (mapeamento em memória) para enviar dados pela rede, isto é, o dispositivo de origem dos dados precisa participar intensamente de empacotamento. É necessário que a origem escreva os dados a serem transmitidos em regiões específicas da memória para que estes sejam enviados pela interface de rede para o nó destino. Esta característica mostra vantagens quanto a agilidade na passagem de informações diretamente para a rede, porém é desvantajosa frente à técnica de filas de serviços pois requer que o processador lide com o processo da comunicação ao invés de dedicar-se a execução de computação.

3.6 RESUMO DO CAPÍTULO

Neste capítulo foi feito o levantamento bibliográfico das diferentes pesquisas aplicadas à área de interface de redes para redes-em-chip. Foram vistos as principais técnicas para interfaceamento de redes-em-chip e dispositivos IP. Dessas, avalia-se criticamente alguns aspectos quando adotado aplicações em MPSoC e identifica-se possíveis soluções a serem aplicadas em uma interface mais adaptada aos sistemas multiprocessados sobre redes-em-chip.

4 INTERFACE DE MULTIPROCESSAMENTO EM REDE: MPNI

A solução proposta nesta dissertação visa prover meios para processadores manterem comunicação não bloqueante através das redes-em-chip, porém permaneçam livres para executar trabalho útil (computação), deixando a cargo de uma interface de rede o gerenciamento de suas comunicações (tarefa secundária aos processadores). A interface de rede foi descrita em VHDL (*Very High Speed Circuits Description Language*) e dispõe de parametrização em tempo de projeto, sendo configurável de acordo com os requisitos de cada aplicação. Os processadores, ou Elementos Processantes (EPs) ligados a ela não sofreriam qualquer modificação de hardware e o protocolo de comunicação EP/NI é exequível pelo uso de operações *load* e *store*. A interface de rede, através de serviços, provê meios para utilização da NoC utilizando o paradigma de passagem de mensagens, liberando o processador para realização de computação útil.

A interface proposta, batizada de *Multiprocessor Network Interface* (MPNI), visa utilizar um subconjunto de operações padronizadas do MPI 3.0, comuns em sistemas que utilizam o paradigma de troca de mensagens. Por meio dessa ferramenta, será possível o reuso dos serviços dessa camada de hardware em projetos de MPSoC. O que se propõe neste trabalho serve às aplicações MPSoC homogêneas e heterogêneas, usando o modelo de memória distribuída, executando operações MPI. O esquema básico da estrutura de um MPSoC utilizando a interface MPNI está ilustrado na Figura 12.

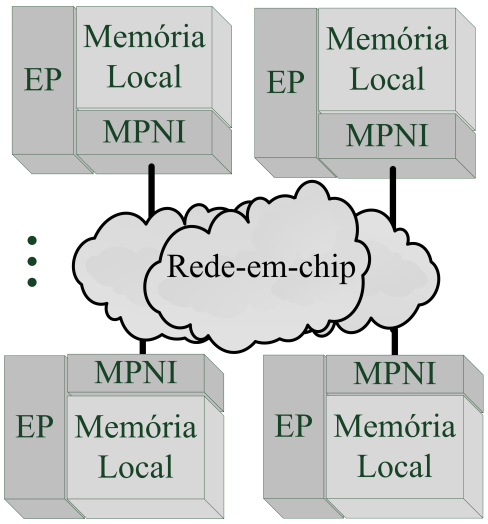


Figura 12 – Esquema de ligação dos nós (processador (EP), MPNI e memória local) em uma rede de topologia genérica

4.1 ARQUITETURA DA INTERFACE DE REDE

O módulo MPNI é um IP *core* projetado em *Register Transfer Level* (RTL). Este dispositivo trabalha com acesso direto a uma memória local compartilhada entre a interface e o processador por meio de um barramento. Neste esquema, após ser detectada escrita no endereço correspondente ao ponteiro da fila de envios, a interface solicita acesso à memória local e inicia o processo de comunicação diretamente com a rede. Assim, nenhuma ação adicional é requerida ao processador, que está livre para executar computação útil. Vale dizer que a memória local a cada processador e respectiva MPNI adota o esquema de memória distribuída, isto é, não utiliza um espaço de endereçamento global ao MPSoC, cada nó possui seu próprio espaço de endereços.

Na interface entre processador, memória e MPNI não é utilizado protocolo de barramento padrão específico. Para a MPNI a regra de troca de informações entre ela e o processador segue um modelo próprio. Neste caso, o módulo dispõe de um árbitro de barramento que funciona no esquema *Round Robin*. Este é consultado pelos dispositivos que querem ter acesso à memória local. Para isso, faz-se uso do protocolo *handshake*, o processador solicita o controle do barramento por meio do *req bit* e o árbitro sinaliza a concessão pelo *ack bit*. Após concluir o acesso, o barramento é liberado e alocado para o próximo dispositivo na fila. Esta técnica é uma solução a protocolos de barramento proprietários, permitindo uma flexibilização no uso deste *softcore* em dispositivos de outros fabricantes. Em um arranjo básico temos a interface de rede e o EP compartilhando esta memória, porém, opcionalmente, dispositivos adicionais podem ser conectados a este barramento, bastando parametrizar a interface para tal. Na Figura 13 é mostrada a arquitetura da interface de rede.

Visando minimizar o custo de área em *chip* e a simplificação do funcionamento da interface de rede, não se optou pela criação de *buffers* internos à MPNI para recebimento e envio simultâneos de pacotes. Por esta razão a interface funciona em *half-duplex*. Isto maximiza o uso dos *buffers* da própria rede e da memória local.

O único *buffer* utilizado pela MPNI é o *Buffer* de Operações, este guarda as informações da operação em curso. Em caso de envio de pacotes, por exemplo, a operação é copiada da memória local e mantida no *Buffer* de Operações até que o comando seja completado. Assim, todas as informações ficam à disposição da interface de rede, diminuindo o acesso à memória e, consequentemente, a disputa entre os dispositivos no barramento.

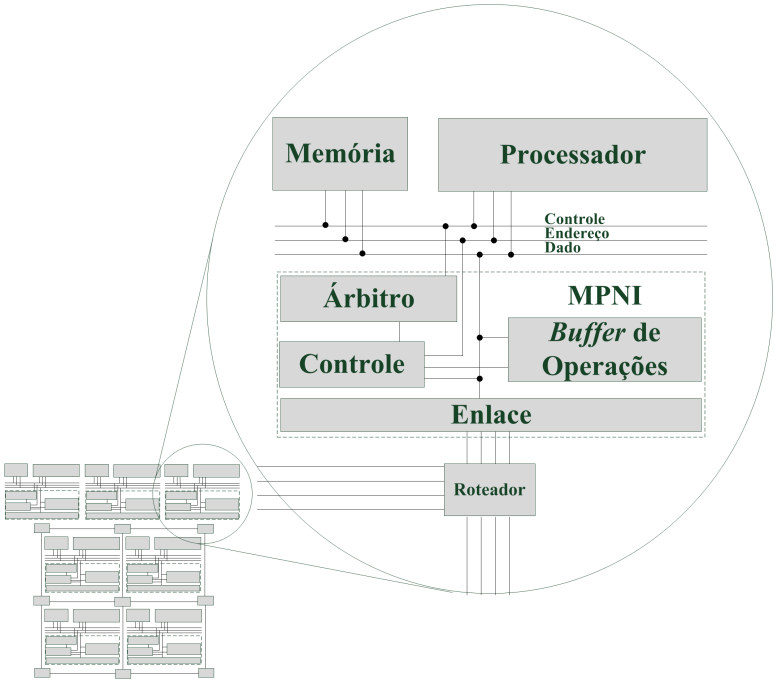


Figura 13 – Diagrama de ligações entre memória local compartilhada, processador e MPNI (bloco de controle, enlace, árbitro e *buffer* de operações) ligados a uma rede em topologia grelha 2-D

4.2 ESQUEMA DE MEMÓRIA

A interface de rede opera através de passagem de parâmetros via memória compartilhada entre esta e o respectivo processador. O esquema de memória usa listas encadeadas para ordenar os serviços de comunicação a serem executados pela interface de rede. As solicitações de comunicação são especialmente montadas de acordo com a rede utilizada. Nesta implementação, a interface de rede MPNI é compatível com a rede SoCIN*fp* (ZEFERINO; SANTO; SUSIN, 2004), entretanto, utilizando a técnica de filas de solicitações, pode ser adaptada ao uso de diferentes redes-em-chip disponíveis. Para isso, muda-se o formato das solicitações (tornando-a adaptada à rede-em-chip utilizada) e parte do comportamento da interface modelada pela máquina de estados da MPNI.

Na estrutura de dados montada na memória, cada serviço de envio é organizado em duas ou mais palavras, que, mais tarde, serão convertidas em *FLow control unITs* (Flit). As palavras devem conter, no mínimo, 24 *bits* para acondicionar todas as informações necessárias. Considerando a rede SoCIN*fp*, proposta por Zeferino, Santo e Susin (2004), a primeira palavra contém os *Status Bits* (SB), *bits* de HLP ou Protocolo de Alto Nível (PAN) , endereço de origem e endereço de destino na rede. A segunda informa o endereço de memória para início da transferência dos dados. Para certos comandos é necessário utilizar uma terceira palavra, sua função é específica de cada comando. Adiante estes comandos serão detalhados.

Caso seja utilizada a rede HERMES (MORAES et al., 2004), algumas diferenças devem ser respeitadas. A primeira palavra deve conter o endereço de origem e destino na rede. A segunda deve informar a quantidade total de palavras a serem transferidas, considerando os dados e as informações de protocolo. A terceira palavra determina o endereço local para início da transferência dos dados. Na quarta devem constar as informações de SB e PAN. Assim como no esquema da SoCIN*fp* (ZEFERINO; SANTO; SUSIN, 2004), para os demais comandos uma palavra adicional é requerida.

A partir de determinada posição da memória local compartilhada entre EP e MPNI, estão organizados em região contígua os ponteiros para as listas de envios, recebimentos e grupos. Estas são listas circulares encadeadas contendo informações das operações e grupos de comunicação. A lista de envios contém informações para execução dos comandos *initialize*, *finalize*, *turn on*, *turn off*, *put* e *send*. Os comandos *initialize* e *finalize* utilizam informações presentes na lista de grupos para serem executadas. A lista de grupos registra todos os grupos no qual o processador está inserido. A Figura 14 mostra um exemplo do esquema de memória utilizado pela MPNI, montado considerando a rede SoCIN*fp* (ZEFERINO; SANTO; SUSIN, 2004).

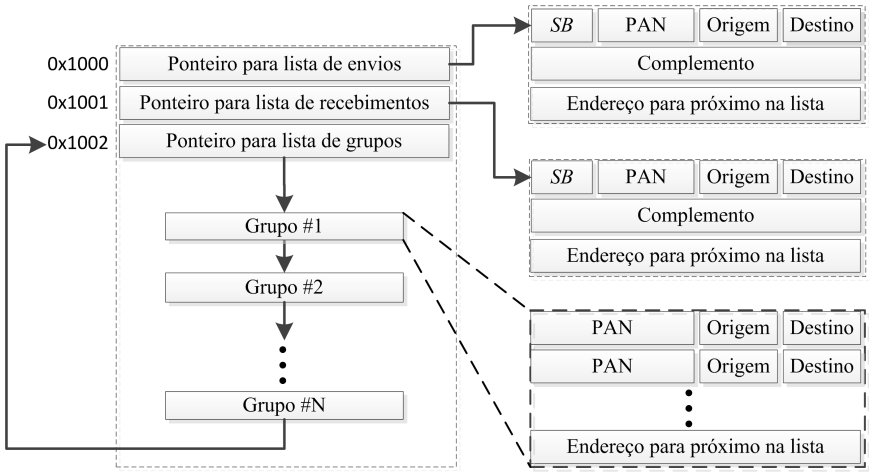


Figura 14 – Esquema de memória utilizando listas circulares encadeadas

Operações de recebimento devem ser cadastradas pelo EP na lista de recebimentos. Esta lista será consultada a cada recebimento cooperativo, isto é, aquele em que origem e destino concordam para que haja a transferência de dados. Assim como nas operações da lista de envios, ainda considerando a rede SoCIN*fp*, o recebimento deve conter as informações de SB, PAN, endereço de destino e origem na rede. O complemento informa o endereço inicial no qual os dados devem ser escritos no destino, enviados pelo uso de *send* a partir do nó de origem. Na ocorrência da chegada de um pacote a lista é consultada. Caso seja identificada, a operação é realizada e os SBs de recebimento são atualizados, confirmando o sucesso da comunicação, do contrário, o pacote é descartado. As funções de colocar e retirar solicitações de envio, recebimento e grupo são exclusivamente

do processador local. A interface de rede apenas atualiza os SBs de cada operação cadastrada nas listas.

4.3 SERVIÇOS DA INTERFACE DE REDE

Em uma análise preliminar do relatório MPI 3.0, foram identificadas algumas operações básicas do paradigma de passagem de mensagens. A priori, faz-se a distinção entre as categorias de comandos unilaterais e cooperativos. Comandos unilaterais são executados sem o conhecimento do destino. As operações cooperativas, entretanto, executam a passagem de informações de maneira complementar. Ou seja, para que a operação seja completada, origem e destino devem executar instruções cooperativamente. É o caso do comando *send* e *receive*, ambos devem ser executados pela origem e destino dos dados, respectivamente. Outra distinção ocorre entre comandos bloqueantes e não bloqueantes. Ao demandar um serviço à camada de interface de rede, o processador solicitante pode ser bloqueado até que toda a operação seja concluída, ou liberado imediatamente à passagem de parâmetros para a interface. As operações básicas identificadas para dar suporte às exigências de aplicações em ambientes multiprocessados foram organizadas na tabela 1.

Tipo	**Código Binário**	**Código Hexadecimal**	**Comando**
Cooperativo	0000	0x0	*Send* bloqueante
Cooperativo	0001	0x1	*Receive* bloqueante
Cooperativo	0010	0x2	*Send* não-bloqueante
Cooperativo	0011	0x3	*Receive* não-bloqueante
Unilateral	0100	0x4	*Put* bloqueante
Unilateral	0101	0x5	*Put* não bloqueante
Unilateral	0110	0x6	*Turn on*
Unilateral	0111	0x7	*Turn off*
Cooperativo	1000	0x8	*Initialize*
Cooperativo	1001	0x9	*Finalize*
—	1010 ... 1111	0x*A* ... 0x*F*	*Reservado*

Tabela 1 – Serviços disponíveis na MPNI: classificação por tipo e código de operação.

O par de comandos *initialize/finalize* define e desfaz, respectivamente, grupos de comunicação, possibilitando a realização de tarefas cooperativas. O comando *initialize* executa o envio da tabela do grupo de comunicação para todos os participantes inseridos neste. A tabela é formada por informações como número identificador do

grupo, quantidades de processadores envolvidos, *rank* (identificação ordinal) de cada EP e endereço deste na rede. Para excluir o grupo de comunicação executamos o comando *finalize*. Este emite um pacote à todos os participantes do referido grupo informando a exclusão deste da lista de grupos ativos.

Com a intenção de gerenciar a energia utilizada no MPSoC, os comandos *turn on* e *turn off* foram criados para proporcionar os efeitos de ligar e desligar os processadores em tempo de execução. Outro comando é o *put*, que oferece uma maneira de transferir palavras de dados para a memória de determinado EP de forma unilateral. Em outras palavras, a origem dos dados tem a permissão de escrita direta na memória do processador destino. Este comando é útil na passagem de instruções e dados para EPs que estão sendo inicializados.

Os comandos *send* e *receive* são usados na comunicação de propósito geral. Estes são essencialmente cooperativos. Isto é, para cada *send* de uma dada origem deve existir um *receive* correspondente no destino. Caso contrário, os pacotes recebidos pela interface de rede serão descartados no destino.

O comando *put* foi criado com o objetivo de iniciar a troca de dados entre os processadores conectados à rede. Analisando um exemplo, inicialmente um sistema pode dispor de apenas um processador com a memória de programa contendo instruções válidas. Para que os demais possam iniciar o recebimento de dados cooperativamente é preciso que a memória de programa dos demais dispositivos sejam inicializadas. Esta operação só poderá ser executada por meio do comando unilateral *put*. Assim, blocos de dados são enviados para o EP remoto e, a partir da interpretação destas instruções, os dispositivos iniciam a troca de mensagens cooperativamente por meio de *send/receive*.

É importante ressaltar que o padrão MPI 3.0 serviu apenas como guia para a formulação dos serviços disponíveis na interface. Comandos como *put*, *turn on* e *turn off* não compõe o padrão, estes foram criados com o intuito de adaptar o sistema às particularidades observadas em MPSoCs.

4.4 FILA DE SOLICITAÇÕES

As solicitações de comunicação à interface MPNI devem ser previamente cadastradas nas filas de serviços presentes na memória localmente a cada processador. Estas estruturas de dados estão disponíveis com o intuito de facilitar a troca de informações entre o processador e a interface. Nelas é possível cadastrar solicitações que envolvam envios e recebimento de pacotes, além de listar os grupos de comunicação onde cada processador está inserido. Estas são filas circulares simplesmente encadeadas onde, a partir de um endereço inicial escolhido em tempo de projeto, os dados podem ser

organizados para permitir que a interface de rede tenha acesso às requisições de serviço de comunicação cadastradas pelo processador em tempo de execução.

Na fila circular simplesmente encadeada de envio, ou apenas fila de envios, quando em constante acesso pelo processador, podemos encontrar requisições de serviços à interface de rede que envolvam o envio de dados. Os comandos que requerem envio de dados são o *SEND*, *PUT*, *INITIALIZE*, *FINALIZE*, *TURN ON* e *TURN OFF*. Na Figura 15 está exemplificado três solicitações distintas que remetem dados pela rede: (a) *PUT*, (b) *INITIALIZE* e (c) *FINALIZE*.

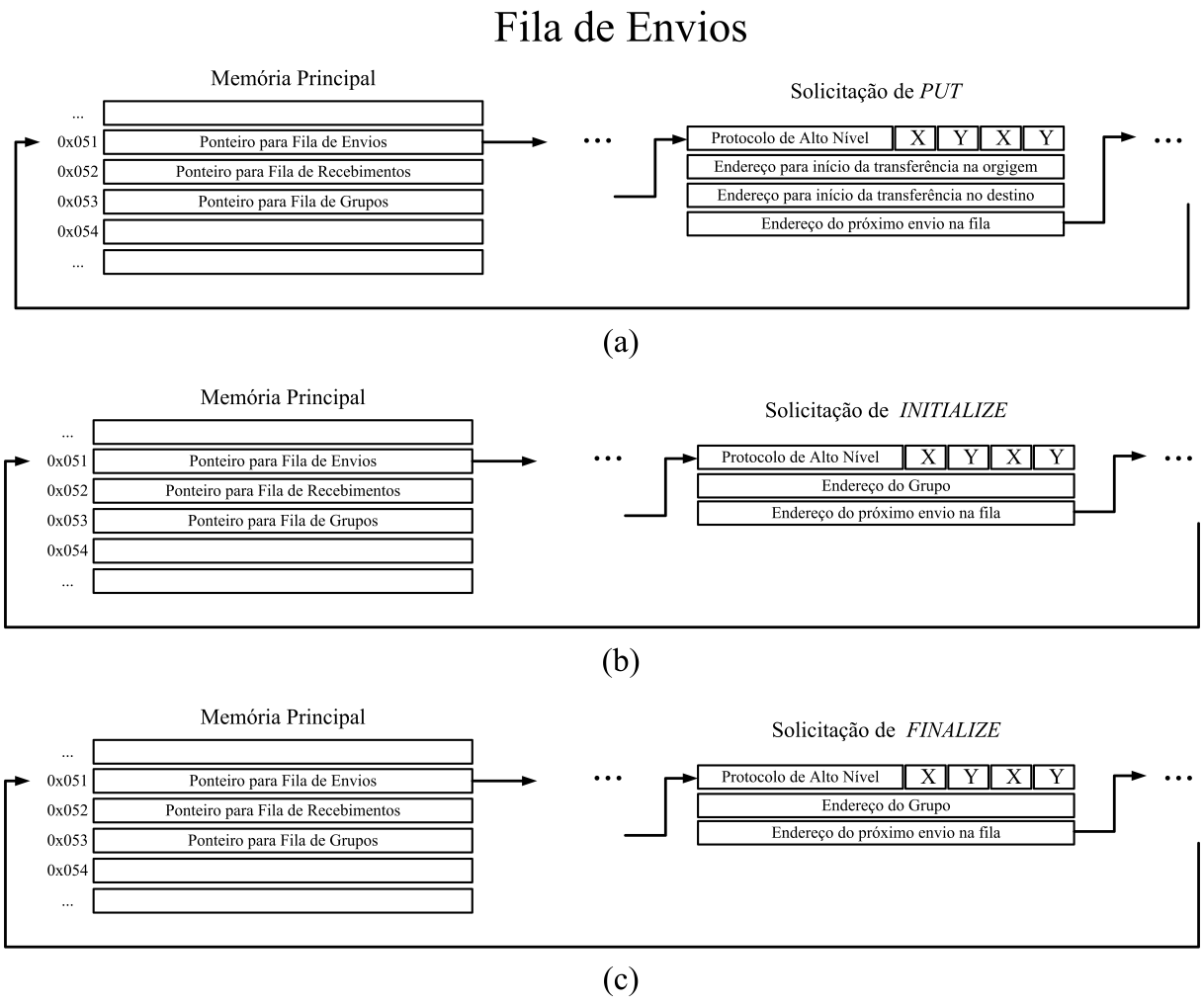


Figura 15 – Esquema representativo da organização das listas circulares para serviços que requerem envio de dados: (a) *PUT*, (b) *INITIALIZE*, (c) *FINALIZE*.

É possível que tenhamos várias solicitações cadastradas na fila de envios, muitas do mesmo tipo inclusive, e em diferentes ordenações. Assim como em uma fila, a ordem de prioridade na execução de cada solicitação é dada pela posição na fila, isto é, as requisições cadastradas próximas ao início da fila serão executadas antes que as demais.

Na fila circular simplesmente encadeada de recebimentos, ou apenas fila de rece-

bimentos, podemos encontrar as requisições por serviços que envolvam o recebimento de dados. Quando um *SEND* é executado por processador remoto, é necessário que uma solicitação de *RECEIVE* esteja previamente cadastrada no destino dos dados para que a transferência seja concluída com sucesso. A Figura 16 mostra o diagrama que representa uma requisição *N* eventualmente cadastrada. As solicitações de serviços de recebimento de dados não possuem prioridades na execução. Ao receber um pacote originário de um *SEND* remoto, a interface de rede pesquisa a lista de recebimento com o objetivo de identificar se a operação pode ser completada e os dados transferidos para a memória local. Caso o recebimento não esteja cadastrado na fila, os pacotes são retirados da rede-em-chip e descartados, não gerando nenhuma confirmação de recebimento dos dados pelo destino.

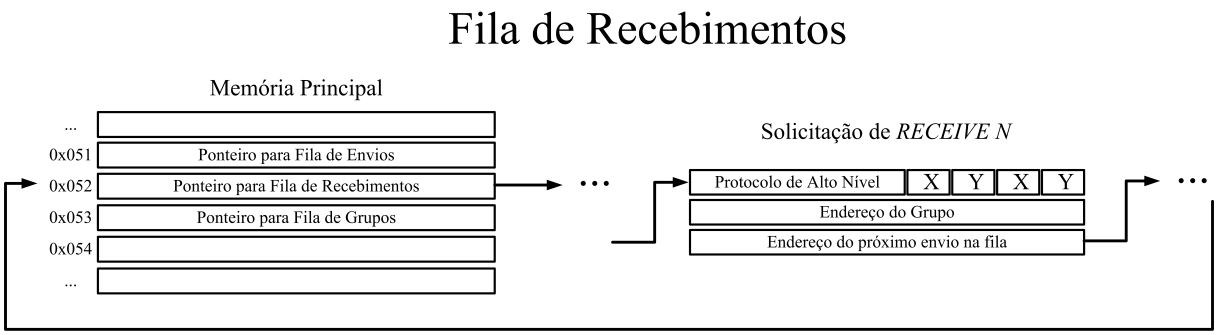


Figura 16 – Esquema representativo da organização das listas circulares para serviços que requerem confirmação no recebimento dos dados: *RECEIVE*.

Observe que várias solicitações de *RECEIVE* podem ser cadastradas e que a execução de cada uma delas não está ligada à posição em que se encontram na fila. Vale ressaltar que, por questões de desempenho, as requisições de recebimento devem estar, na medida do possível, organizadas na sequência em que se espera a chegada dos pacotes. Isto é decorrência de que a interface de rede pesquisará sequencialmente a fila de recebimentos até encontrar a referida requisição antes de permitir a escrita dos dados recém-chegados na memória local. Assim, para requisições em posições finais da fila, o tempo gasto na procura da requisição de recebimento poderá ampliar significativamente a latência final da comunicação. Ainda considerando solicitações de *RECEIVE*, a interface não faz distinção de prioridade na execução do recebimento dos pacotes, assim, serão copiados na memória o conteúdo de qualquer pacote que seja recebido pela interface de rede e que esteja devidamente cadastrado na fila de recebimentos, não importando a ordem.

Por fim, tem-se a lista de grupos. Esta lista contém informações de dispositivos que são alcançáveis por uma dada origem de dados. A partir destas informações é possível executar solicitações de serviço de envios de dados que envolvam grupos de dispositivos conectados à rede. Na Figura 17 está representada a fila de envios. Adiante

será detalhada a formação da lista de grupos. Por ora basta compreender que estas listas contêm informações organizadas com o propósito de agilizar o envio de dados pela rede para os diferentes dispositivos presentes no grupo.

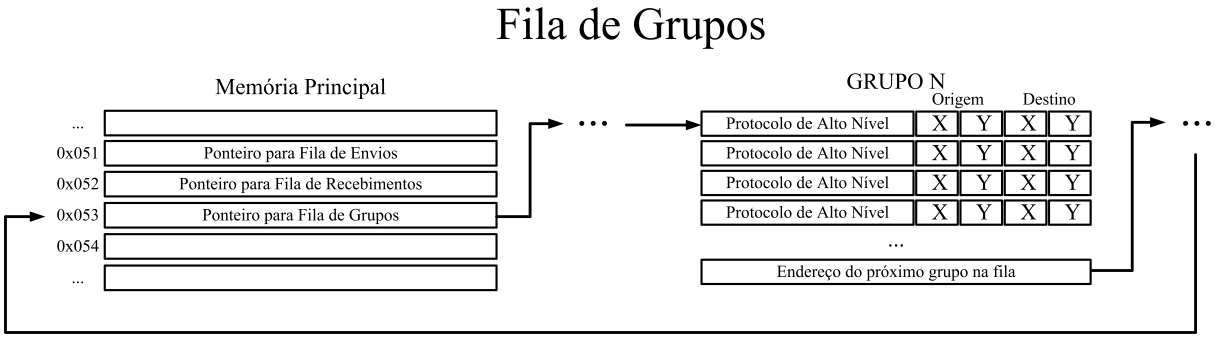


Figura 17 – Esquema representativo da organização das listas circulares de grupos de comunicação presentes na memória.

Assim como a fila de recebimentos, a posição do grupo na fila não denota nenhuma prioridade entre estes. Assim, o grupo pode estar cadastrado em qualquer posição. Em contrapartida, diferentemente do que ocorre na fila de recebimentos, o cadastro de grupos e posições finais da fila não resulta em acréscimo no atraso da comunicação, ou impacto em qualquer outra característica de desempenho. Isto é fato decorrente da maneira em que as listas são acessadas. Estas são endereçadas diretamente, não requerendo pesquisa por registros a ser realizada pela interface de rede.

4.4.1 Requisição de Serviços

A interface de rede opera através de passagem de parâmetros via memória compartilhada entre esta e o respectivo processador. O esquema de memória usa listas circulares simplesmente encadeadas para ordenar os serviços de comunicação a serem executados pela interface de rede. As solicitações de comunicação são especialmente montadas de acordo com a rede utilizada.

Na memória local compartilhada entre EP e MPNI, estão organizados em região contígua os ponteiros para as listas de envios, recebimentos e grupos. Elas contêm informações de operações e grupos de comunicação. A fila de envios contém informações para execução dos comandos *INITIALIZE*, *FINALIZE*, *TURN ON*, *TURN OFF*, *PUT* e *SEND*. Os comandos *INITIALIZE* e *FINALIZE* utilizam informações presentes na lista de grupos para serem executadas. A lista de grupos registra todos os grupos no qual o processador está inserido. A Figura 18 mostra o formato básico de uma solicitação utilizado pela MPNI, montado considerando a rede SoCIN*fp*.

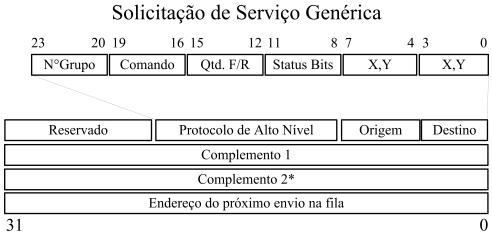


Figura 18 – Formato de uma solicitação de serviço genérica. *Esta palavra complementar é requerida apenas para solicitação de *PUT*.

As solicitações, ao serem processadas, são atualizadas com as informações de *status bits*. Estes códigos, conforme é mostrado na Tabela 2, servem para identificar ao dispositivo solicitante a completude da solicitação. Solicitações processadas pela MPNI possuem, basicamente, os seguintes *status*: *WAIT*, *ON_GOING*, *INACTIVE_GROUP*, *OK_GROUP*, *ERROR* e *OK_REQUEST*. A situação *WAIT* identifica aquela solicitação que aguarda processamento pela interface. O estado de *ON_GOING* designa que a solicitação está sendo processada e ainda não foi concluída. Este ocorre em resposta a solicitações de *INITIALIZE/FINALIZE* e informa que o envio da solicitação foi concluído. Em complemento, o *status INACTIVE_GROUP* e *OK_GROUP* presentes na primeira linha de cada lista na fila de grupos informa a conclusão das operações *FINALIZE* e *INITIALIZE*, respectivamente. Os *status ERROR* e *OK_REQUEST*, informam que a solicitação já foi processada. A primeira designa que ocorreu um erro e a solicitação não pode ser concluída, a segunda informa que a solicitação foi processada corretamente e gerou a ação esperada. Contudo, é importante salientar que cada *status* possui uma localização específica e segue o esquema da Tabela 2. As solicitações de *INITIALIZE/FINALIZE* na fila de requisições de envio são confirmadas por meio dos *bits* de *status* da primeira entrada da lista de grupo, não utilizam, portanto, o *status OK_REQUEST* na requisição presente na fila de envios.

É importante ressaltar que as funções de colocar e retirar solicitações de envio, recebimento e lista de grupo são exclusivamente do processador local. A interface de rede apenas atualiza os SBs de cada operação cadastrada nas listas envio e recebimento, informando o *status* de atendimento conforme a Tabela 2.

A Figura 19 mostra, de maneira resumida, a interação entre a interface de rede e o processador solicitante de um serviço de envio. A máquina de estados representa que, inicialmente, o processador identifica a necessidade de comunicação, acrescentando uma requisição à lista apropriada. Depois de inseri-la, o processador deve sobrescrever o conteúdo de memória do ponteiro para a referida lista (endereço 0x1000, conforme exemplo da Figura 14). A escrita no endereço de memória específico é detectada pela MPNI, que inicia automaticamente a busca pela solicitação recentemente incluída. Todas as requisições de envios não processadas serão atendidas. Não encontrando solicitações

Código binário (Hexadecimal)	***Status* da Solicitação**	**Localização**
0000 (0x0)	*WAIT*: Solicitação de envio pendente de processamento pendente de processamento.	Requisição na fila de envios.
0001 (0x1)	*ON_GOING*: Solicitação *INITIALIZE/FINALIZE* em processamento.	Requisição na fila de envios.
0010 (0x2) 1011 (0xB)	Reservado.	Reservado.
1100 (0xC)	*INACTIVE_GROUP*: Grupo inativado, solicitação *FINALIZE* completa.	Na primeira entrada da lista de grupos.
1101 (0xD)	*OK_GROUP*: Grupo ativo, solicitação *INITIALIZE* completa.	Na primeira entrada da lista de grupos.
1110 (0xE)	*ERROR*: Solicitação não processada por inconsistência na requisição.	Requisição na fila de envios.
1111 (0xF)	*OK_REQUEST*: Solicitação de envio processada com êxito.	Requisição na fila de envios.

Tabela 2 – Definição dos códigos de *status bits*.

pendentes, a MPNI retorna ao modo *idle*.

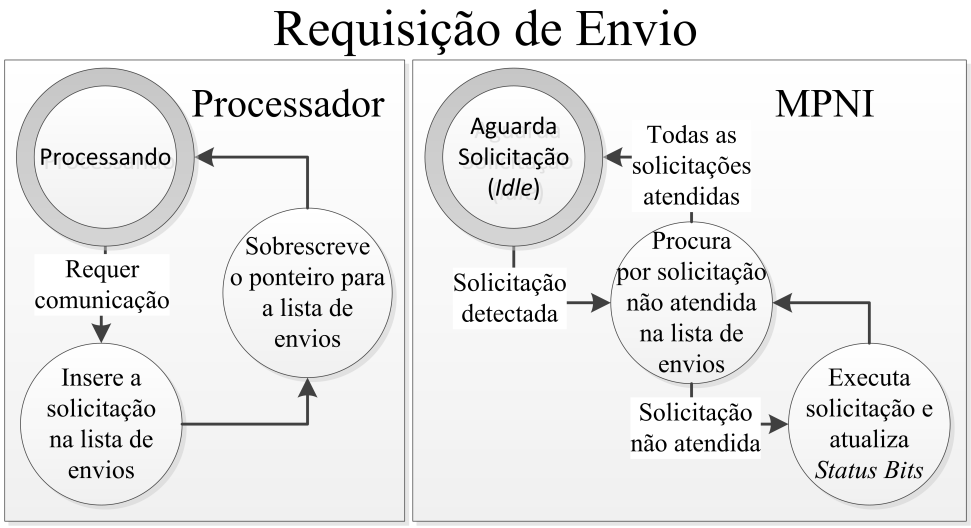


Figura 19 – Máquina de estados resumida mostrando interações entre MPNI e processador para uma solicitação de envio.

Em complemento a operações de envio cooperativo, para que o recebimento de dados seja concluído com sucesso no destino, a MPNI processa o pacote conforme a Figura 20. Previamente é necessário que o processador destino programe o recebimento de dados na fila de recebimentos. A partir deste momento, chegado um pacote pela interface de entrada da interface de rede, a MPNI executará a busca pela requisição previamente cadastrada. Caso a encontre, os dados de recebimento são conferidos e a

transferência de dados para a memória local é permitida. Do contrário, os dados são simplesmente descartados para que os *buffers* da rede-em-chip sejam liberados para que novas mensagens sejam recebidas.

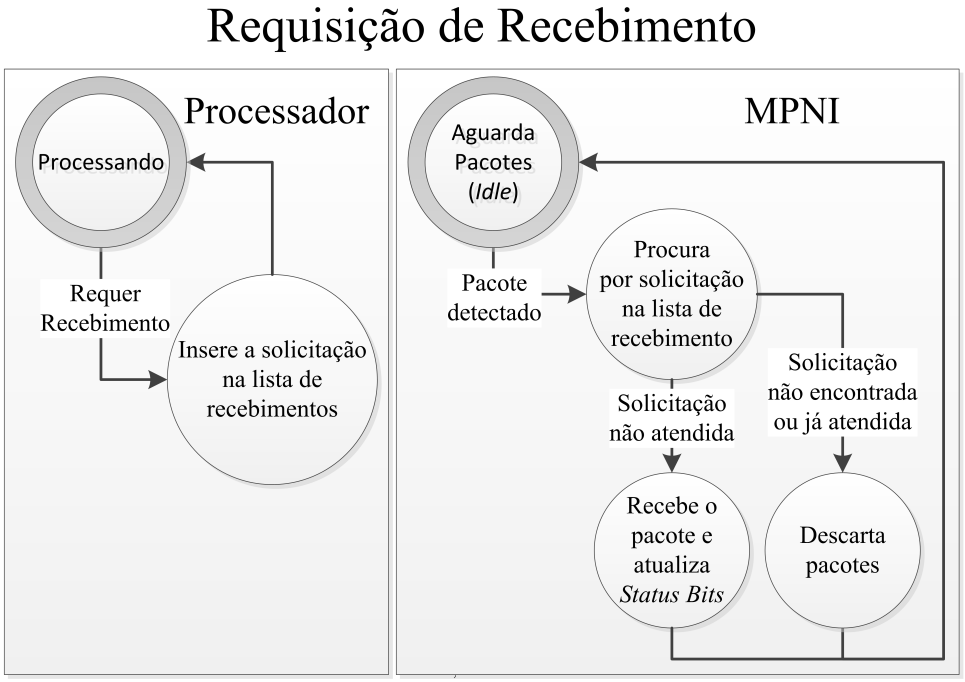


Figura 20 – Máquina de estados resumida mostrando interações entre MPNI e processador para uma solicitação de recebimento.

Tanto para o envio quanto para o recebimento de dados, a MPNI opera em modo de escuta. Isto é, a interface mantém permanentemente um circuito de detecção de atividade ligado ao barramento processador/memória e à interface de saída do roteador local. Em caso de detecção de atividade, a interface de rede providenciará o devido tratamento da ocorrência. É importante frisar que não há prioridade entre as interfaces de barramento e rede. A primeira atividade detectada será tratada, a segunda ocorrência aguardará até a conclusão da operação já iniciada.

4.4.2 Serviço Initialize/Finalize

Para requerer a inicialização de um grupo de comunicação, o processador precisará acrescentar duas entradas às filas de serviços na memória local. Primeiramente deve-se incluir na fila de grupos a lista contendo todos os dispositivos pertencentes ao grupo que se pretende inicializar. Em seguida, a requisição de *FINALIZE* ou *INITIALIZE* deverá ser cadastrada caso se queira inicializar ou desfazer um grupo de comunicação, respectivamente.

4.4.2.1 Formato da Solicitação de Initialize/Finalize

Para a execução do serviço de inicialização de grupo de comunicação, a MPNI precisa das seguintes informações: número do grupo, identificação do comando, quantidade de dispositivos inclusos no grupo, os *bits* com as informações de *status* da solicitação e o endereço da memória onde a lista de grupo está registrada. A Figura 21 demonstra o formato da solicitação de *INITIALIZE* ou *FINALIZE*. Note que a diferença entre criar ou desfazer o grupo reside na identificação do comando a ser realizado.

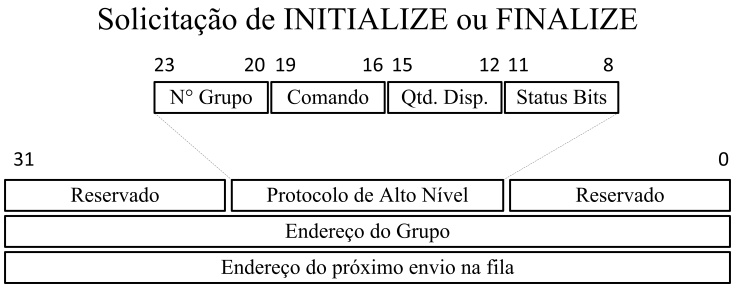


Figura 21 – Formato de uma solicitação de serviço *INITIALIZE/FINALIZE*.

Na execução do serviço de *INITIALIZE/FINALIZE*, a MPNI processa os dados presentes na solicitação e identifica qual grupo na fila de grupos a requisição se refere. Em seguida, a lista de dispositivos é acessada e o envio do pacote de inicialização do grupo é realizado. A Figura 22 mostra a representação do referido pacote a ser destinado pela rede.

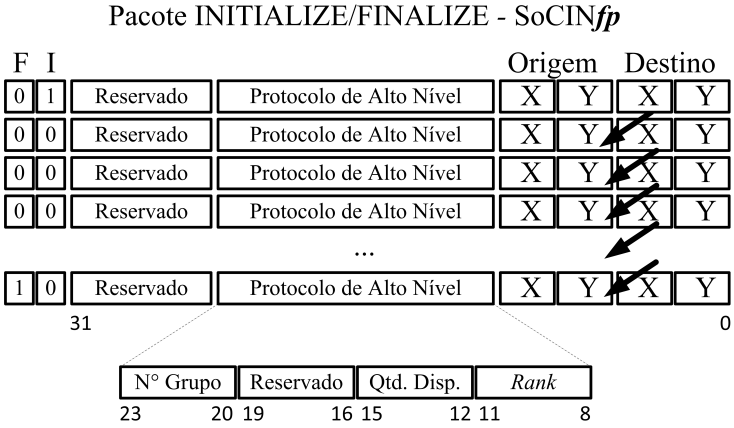


Figura 22 – Formato do pacote de *INITIALIZE/FINALIZE*.

O pacote de inicialização ou finalização de grupo é enviado para todos os dispositivos inclusos no grupo a ser criado/finalizado. Compete à interface de rede de cada dispositivo incluso no grupo receber a referida lista e adicioná-la à fila de grupos local. A lista empacotada é repassada para cada dispositivo seguindo a mesma ordem

dos dispositivos cadastrados e, ao final do processo, todos os participantes do grupo terão a lista completa dos dispositivos daquele grupo de comunicação em sua respectiva fila de grupos. A Figura 23 esquematiza como é realizada a passagem da lista de grupos entre as interfaces de rede dos dispositivos participantes do grupo de comunicação que está sendo criado. Neste exemplo é mostrado um grupo de comunicação hipotético formado pelos nós ([0,0];[2,0];[3,1] e [3,3]) de uma rede-em-chip organizada na topologia grelha 2D. Os outros nós da rede foram suprimidos para simplificar a visualização.

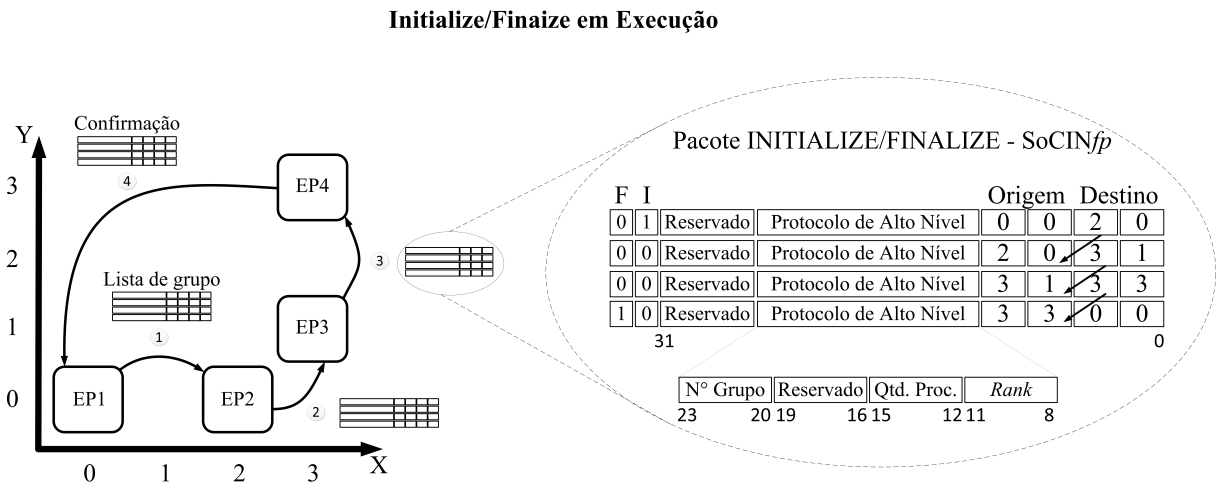


Figura 23 – Execução do comando *INITIALIZE/FINALIZE*.

O pacote de inicialização/finalização é injetado na rede pela interface ligada ao dispositivo inicializador do grupo, conforme é visto em (1). Os dispositivos sequencialmente na lista vão propagando e montando/desmontando os grupos em suas respectivas memórias locais acessíveis a cada dispositivo do grupo, passos (2) e (3). Observe que o pacote é repassado para cada dispositivo na rede seguindo a ordem em que foram cadastrados na lista de grupos. Ao atingir a última interface de rede listada, o pacote então retorna para o dispositivo inicializador/finalizador do grupo, conforme em (4). A completude desta ação confirma que o comando foi executado corretamente. Na recepção do pacote pelo dispositivo inicializador/finalizador, a MPNI atualiza os bits de *status* da solicitação na fila de envios, confirmando a execução do comando com sucesso.

4.4.3 Serviço Turn On e Turn Off

Adicionalmente aos comandos de MPI básicos relacionados anteriormente, foi avaliado a utilização de um comando próprio que permitisse ao programador do sistema

ligar e desligar processadores em tempo de execução. Esta característica possibilita ao usuário criar programas energeticamente eficientes.

Apesar da existência das operações de ligar e desligar núcleos, a tecnologia FPGA (*Field Programmable Gate Array*) utilizada neste trabalho não permite efetivamente desligar partes do circuito eletrônico em tempo de execução. Porém, em multiprocessamento, a espera ociosa, isto é, o aguardo por instruções e dados para execução causa um consumo permanente de energia dinâmica, aquela provocada por variação do estado dos bits dos registradores do sistema e/ou memórias. Pois o processador permanentemente acessa a memória e procede com a execução de rotinas de espera. Este custo energético é poupado com a utilização do mecanismo de bloqueio de permissão do acesso à memória local. Assim, a interface de rede paralisa a execução do processamento devido a não permissão do acesso de outros dispositivos ao barramento local. Isto provoca a espera por parte do processador pela liberação do barramento da memória, reduzindo chaveamentos internos ao dispositivo e, consequentemente, economizando energia.

Este comando torna-se verdadeiramente útil ao programador no que tange ao gerenciamento de energia gasta pelo multiprocessador, especialmente em máquinas com vários dispositivos ligados à rede. Em sistemas dessa magnitude, sua disponibilidade possibilita uma redução de consumo total do sistema. A MPNI possui, em tempo de projeto, o parâmetro "ligado". Esta opção define se a interface criada estará em modo ligado ou desligado ao energizar o FPGA.

4.4.3.1 Formato da Solicitação de Turn On e Turn Off

A solicitação de *TURN ON/OFF* deve ser montada na fila de envios da memória local ao dispositivo solicitante. O formato da instrução montada na fila segue o exposto na Figura 24. Para que a solicitação seja realizada são necessárias as seguintes informações: número do grupo, código do comando a ser executado (*TURN ON/OFF*), *rank* (identificação do dispositivo a ser ligado/desligado participante do grupo) e o endereço da origem e do destino da solicitação.

Na execução do comando, a MPNI acessa a fila de envios, identifica a requisição e executa diretamente o envio das informações organizadas na memória. A disposição dos campos na solicitação corresponde aos utilizados diretamente pela rede-em-chip em uso. Assim, o envio da solicitação é agilizado, há uma redução de impacto no tempo total de comunicação, pois o enviado da mensagem é realizado imediatamente. A Figura 25 mostra o pacote injetado na rede pela interface de rede.

O pacote de *TURN ON/OFF* é bem resumido, contém apenas dois *Flits*. O *Flit* terminador, por sua vez, não contém informação útil. Este existe apenas para preencher

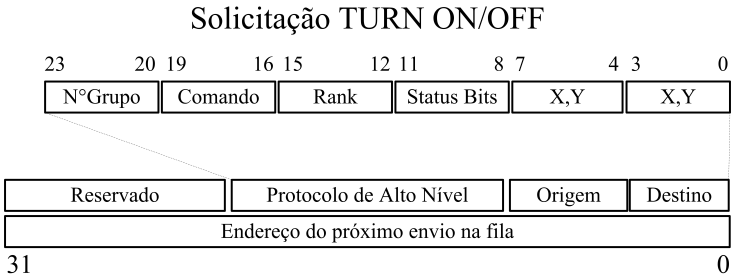


Figura 24 – Formato de uma solicitação do serviço *TURN ON/OFF*.

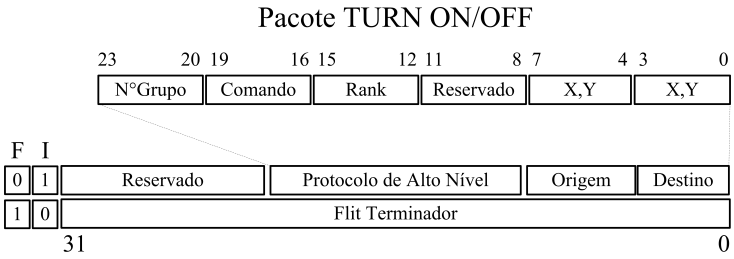


Figura 25 – Formato do pacote *TURN ON/OFF*.

o pacote com o mínimo de informações necessárias para que seja permitido trafegar o dado pela rede SoCIN-*fp*. Outro aspecto importante do comando é que qualquer processador inserido em um grupo pode desligar outros processadores, não havendo privilégios entre estes.

4.4.4 Serviço Put

O serviço de *PUT*, assim como o *TURN ON/OFF* não compõe a lista de opções disponíveis no padrão MPI. Este comando também foi proposto após análise de características específicas de MPSoCs genéricos. Neste caso, o multiprocessador, ao iniciar, necessita passar informações para os dispositivos escravos conectados à rede. Este serviço admite que os programadores realizem a passagem de instruções e dados para a memória dos dispositivos escravos sem que estes interfiram no processo. Esta operação de inicialização é classificada como unilateral, ou seja, não é necessária contrapartida do dispositivo remoto para que os dados sejam transferidos.

4.4.4.1 Formato da Solicitação de Put

A solicitação de *PUT* requer o envio de dados pela rede. Isto sugere que as solicitações deste serviço devam ser inseridas na fila de envios presente na memória localmente ao dispositivo requisitante. O formato segue o mesmo padrão que os anteriores, com a diferença de que este comando requer a especificação de um endereço de memória onde os dados de origem estão e um outro endereço que indica onde serão escritos no destino. Outro aspecto a ser considerado é a quantidade de *Flits* que serão transferidos. O esquema contendo todos os campos a serem preenchidos para a solicitação de *PUT* está mostrado na Figura 26.

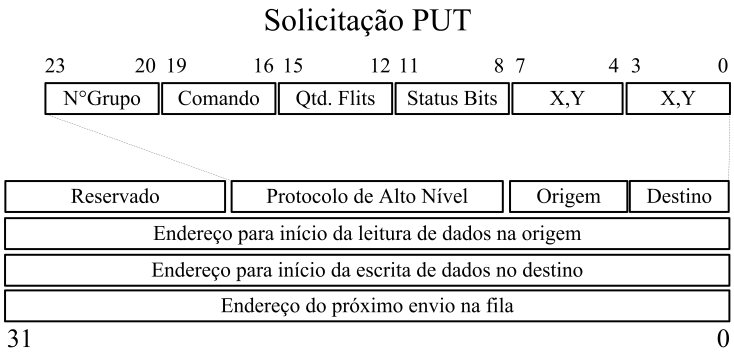


Figura 26 – Formato de uma solicitação do serviço *PUT*.

O pacote enviado pela rede contém as informações necessárias para encontrar o destino e escrever em endereço especificado na origem os dados repassados. A Figura 27 demonstra o exemplo de pacote gerado pela solicitação *PUT*. Neste caso, todos os *Flits*, além dos dois primeiros que compõe o cabeçalho, contém os dados para transferência.

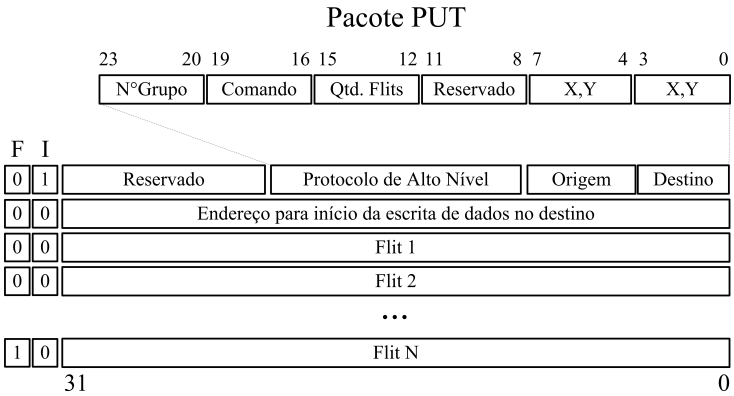


Figura 27 – Formato do pacote *PUT*.

O pacote *PUT* é recebido pela interface de rede no destino que trata de escrever os dados disponíveis no corpo do pacote no endereço repassado no cabeçalho. Esta

operação não gera perda de dados, ou seja, sempre ocorrerá a conclusão da transferência pois trata-se de um comando unilateral, não requerendo ação agendada no destino para ser concluído. Esta característica é essencial na inicialização/configuração da memória de dispositivos remotos. Como exemplo pode-se citar dispositivos espalhados na rede dedicados à execução de entrada e saída.

4.4.5 Serviço Send

Um dos serviços de MPI mais utilizados no paradigma de troca de mensagens é o *SEND*. Este é responsável pelo envio de dados pela rede para destinos que estejam preparados para receber e tratar estas informações. O comando MPI *SEND* é de propósito geral e visa criar mensagens de envio de dados para destinos de interesse do agente comunicador. Este serviço disponível na MPNI é classificado como cooperativo, isto é, é realizado em dois momentos, uma na origem e outra no destino. Ao final da operação de troca de mensagens por meio de *SEND*, os dados estarão disponíveis para acesso no determinado destino da rede.

4.4.5.1 Formato da Solicitação de Send

A solicitação *SEND* deve ser cadastrada na fila de envios. É importante salientar que a ordem a qual as requisições estão na fila de envios será a mesma que a MPNI seguirá executando até não sobrar nenhuma requisição pendente. Para o cadastramento de uma solicitação *SEND* são: número do grupo de comunicação, código do comando *SEND* (0000 ou 0010, conforme Tabela 1) quantidade de *Flits* a serem transferidos, endereço de origem (x,y) e destino (x,y) dos dados na rede-em-chip. Além destes, é necessário que a origem especifique o endereço inicial de onde os dados serão copiados para compor o pacote a ser remetido pela rede.

Nesta solicitação é possível identificar o campo de *status bits*. Este campo é preenchido automaticamente pela interface de rede após a conclusão do envio da informação, conforme visto na Tabela 2. Estes *bits* devem ser preenchidos com "0000"(Tabela 2. Este *status* identifica que a solicitação aguarda por processamento a ser realizado pela interface de rede.

A operação de *SEND*, quando executada, injeta o pacote conforme é visto na Figura 29. Como foi visto anteriormente, todas as solicitações seguem o padrão adotado para a rede SoCIN-*fp*. Um aspecto importante a ser ressaltado é que há um limite para o

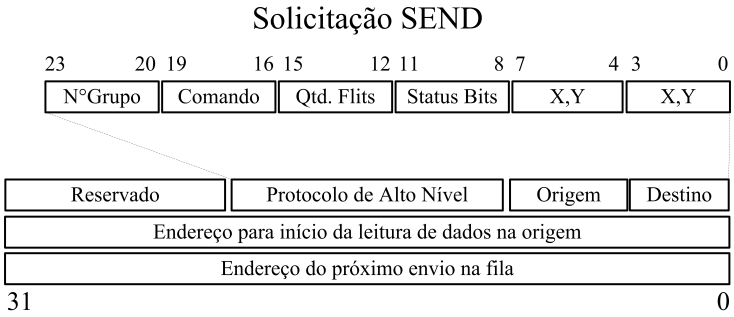


Figura 28 – Formato de uma solicitação do serviço *SEND*.

tamanho do pacote enviado pela interface de rede por meio de uma solicitação *SEND*. A origem só pode enviar até 16 entradas da memória local por pacote. Esta aparente limitação visa limitar o tamanho do pacote que trafega na rede. Assim, para necessidade de pacotes maiores, a origem deve segmentar o envio das informações em mais de um pacote objetivando a passagem de mais informação pela rede. Estes segmentos podem ser rastreados e protocolos de verificação podem ser implementados em camadas acima para garantir a integridade e a ordem dos pacotes recebidos pela rede.

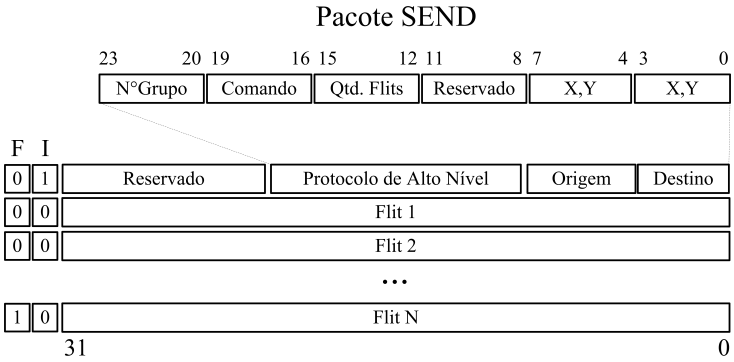


Figura 29 – Formato do pacote *SEND*.

4.4.6 Serviço Receive

Operações de *RECEIVE* devem ser cadastradas pelo EP na lista de recebimentos. Esta lista será consultada a cada recebimento cooperativo que a interface detectar vindo da rede. Assim como nas operações da lista de envios, ainda considerando a rede SoCIN*fp*, o recebimento deve conter as informações de SB, PAN, endereço de destino e origem na rede. O complemento informa o endereço inicial no qual os dados devem ser escritos no destino, enviados pelo uso de *SEND* a partir do nó de origem. Na ocorrência da chegada de um pacote a lista é consultada. Caso seja identificada, a

operação é realizada e os SBs de recebimento são atualizados, confirmando o sucesso da comunicação, do contrário, o pacote é descartado. As funções de colocar e retirar solicitações de envio, recebimento e grupo são exclusivamente do processador local. A interface de rede apenas atualiza os SBs de cada operação cadastrada nas listas.

4.4.6.1 Formato da Solicitação de Receive

Assim como nas operações da lista de envios, ainda considerando a rede SoCIN*fp*, o recebimento deve conter as informações de SB, PAN, endereço de destino e origem na rede. A informação do campo complemento 1, assim como a Figura 18 sugere, informa o endereço inicial no qual os dados devem ser escritos no destino, enviados pelo uso de *SEND* a partir do nó de origem. Na ocorrência da chegada de um pacote a lista é consultada. Caso seja identificada, a operação é realizada e os SBs da solicitação de recebimento são atualizados, confirmando o sucesso da comunicação, do contrário, o pacote é descartado.

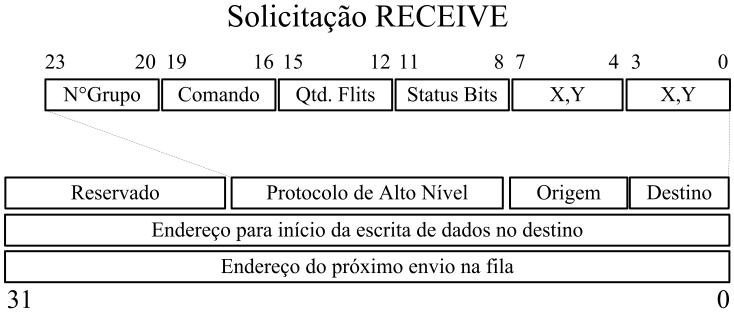


Figura 30 – Formato de uma solicitação do serviço *RECEIVE*.

As solicitações de *RECEIVE*, diferentemente das outras vistas até o momento, não geram pacotes a serem transmitidos pela rede. A solicitação cria uma expectativa de que dados provenientes de outros pontos do sistema serão recebidos em algum instante futuro. As requisições de *RECEIVE* constituem uma fila de pedidos separada das outras requisições para proporcionar agilidade na busca e verificação de permissão para concluir o recebimento de pacotes gerados por requisições *SEND* executadas remotamente.

4.5 EXEMPLO DE ENVIO - VISÃO DO SISTEMA OPERACIONAL

Por questões de segurança e manutenção da estabilidade de um sistema *hardware/software*, é necessário que o Sistema Operacional (SO) faça o intermédio entre o *hardware* e os programas de usuário. Desta maneira, a interface de rede e as transações de comunicação foram projetadas para serem manipuladas diretamente pelo SO. Assim, o programa de usuário solicita as ações de comunicação e o Sistema Operacional gerencia o processo em conjunto com o *hardware* da MPNI. A Figura 31 demonstra a dinâmica de solicitações de serviços de envio de dados pela rede. O programa de usuário solicita o serviço do sistema operacional que cadastra a requisição na fila de envios presente na memória. Após o envio ter sido concluído, a MPNI atualiza os *Status Bits* (SB). O sistema operacional verifica a finalização do processo, avisando ao programa de usuário e excluindo requisição da fila de envios.

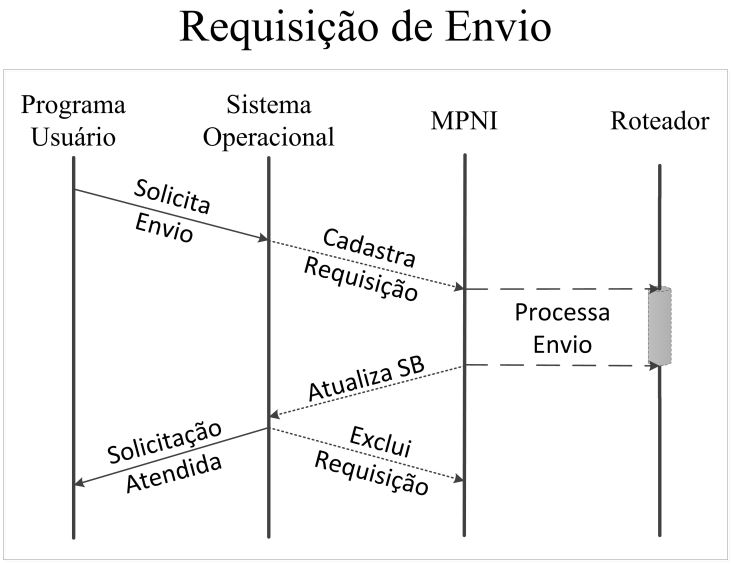


Figura 31 – Diagrama de interações entre programa de usuário, sistema operacional, MPNI e rede - envio.

Para o processo de recebimento de dados no destino as responsabilidades são semelhantes, sobretudo considerando as operações de recepção de dados. A Figura 32 demonstra a dinâmica de solicitações de serviços de recebimento de dados vindos da rede. O programa de usuário solicita o serviço do sistema operacional que cadastra a requisição na fila de recebimentos presente na memória. Após a MPNI detectar informações vindas da rede e executar a recepção dos dados, os *Status Bits* (SB) são atualizados. O sistema operacional então verifica a finalização do processo, avisando ao programa de usuário e excluindo requisição da fila de recebimentos.

Portanto, para que a interface de rede opere corretamente do ponto de vista do SO é preciso uma pequena rotina de *software* para cadastrar e ativar o serviço.

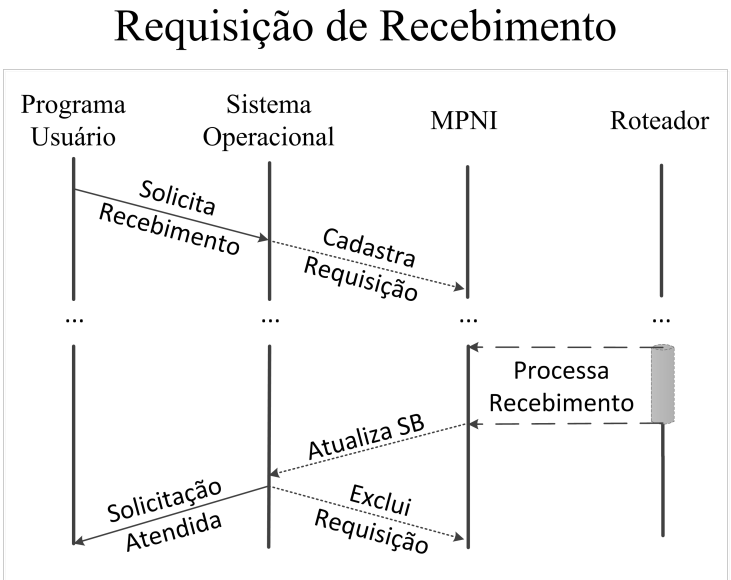


Figura 32 – Diagrama de interações entre programa de usuário, sistema operacional, MPNI e rede - recebimento.

Em um exemplo de rotina simplificado, temos o código do Quadro 4.5. Define-se inicialmente um ponteiro para fila de envios e inicializa-o com o endereço de memória onde situa-se o ponteiro para início da fila de envios (no exemplo, 0x0051). O trecho de código *Solicita_envio()* é uma função recursiva, nesta é realizada a busca do final da fila de envios. No caso do fim da fila ser encontrado, o sistema operacional cadastra as informações de *PAN_ORIGEM_DESTINO* (define aspectos do protocolo de envio e endereços de destino e origem na rede), o *END_DADOS* (endereço na memória local de onde os dados serão transferidos) e finaliza-se a fila com o ponteiro de próximo na fila apontando para o início da fila de solicitações de envio. Para que a interface detecte tal intervenção é preciso que o ponteiro de início de fila seja sobrescrito com o próprio valor (**end_fila_envio=*end_fila_envio;*) (Figura 33-1).

Quadro 4.5 - Rotina do SO manipulando a Fila de Envios - SEND

```
// Globais.
int *end_fila_envio;
#define int end_inicial = 0x0051;

end_fila_envio=end_inicial;

void Solicita_envio(void){
if(*end_fila_envio==end_inicial)
{ // Fim da fila de envios.

        //Aloca o novo pedido na fila.
```

```
        *end_fila_envio=alocar_memoria(3);
        end_fila_envio=*end_fila_envio;

        //Inclui o requerimento de envio na fila.
        *end_fila_envio=PAN_ORIGEM_DESTINO;
        end_fila_envio++;
        *end_fila_envio=END_DADOS;
        end_fila_envio++;
        *end_fila_envio=end_inicial;
        end_fila_envio=end_inicial;

        // Sobrescrita do ponteiro da fila de envios.
        *end_fila_envio=*end_fila_envio;
}
else
{  // Busca o fim da fila de envios.
        while(*end_fila_envio/=end_inicial)
        {
                end_fila_envio+=3;
                end_fila_envio=*end_fila_envio;
        }
   // Solicia cadastro na fila.
        Solicita_envio();
}
}
```

Considerando que a fila de envios esteja corretamente configurada, ação realizada pela rotina do sistema operacional descrito no Quadro 4.5, a Figura 33 mostra detalhadamente o papel de cada componente do sistema realizando a tarefa de envio dos dados pela rede-em-chip. No detalhe, os passos 2 (acesso à fila de envios) e 3 (encaminhamento do pacote para a rede) são de responsabilidade da MPNI.

A operação de envio realizada pela interface de rede, após configurada a solicitação na memória local, ocorre de forma independente dos dispositivos no barramento. O processo completo é idêntico para o envio de dados executado pelas operações *Send*, *Turn on/off*, *Put*, *Initialize* e *Finalize* e didaticamente pode ser dividido em três etapas. A primeira, compreende a configuração no serviço na memória compartilhada localmente entre o dispositivo e a MPNI, feita pelo dispositivo requisitante da operação. A segunda etapa, decorrente da detecção da atividade de escrita em endereços de memória específicos, se segue à primeira. Esta compreende a busca do serviço requisitado e configuração

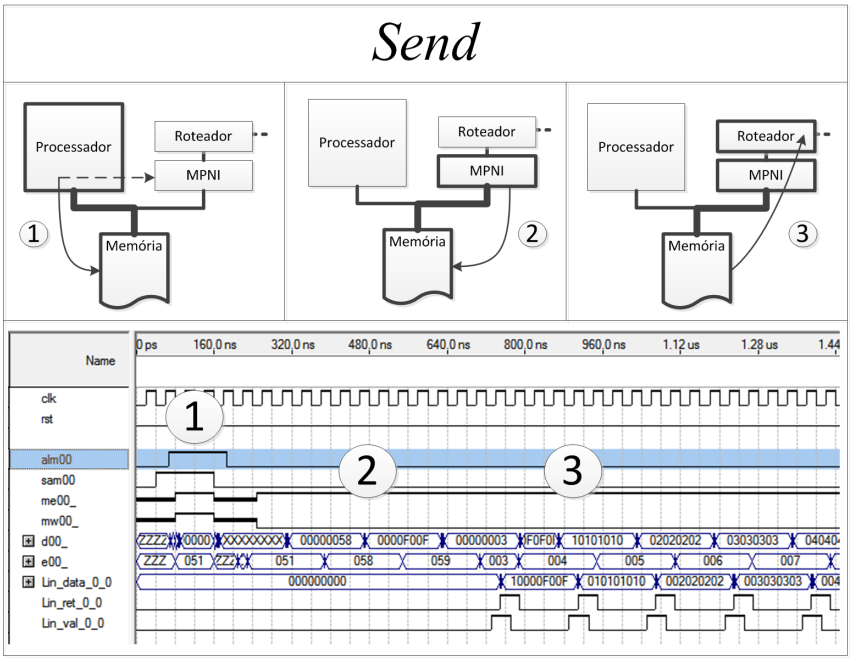


Figura 33 – Interações entre os dispositivos de um nó da rede durante um envio de mensagem.

da interface de rede, que se prepara para a transferência dos dados. A terceira e última etapa é o envio dos dados para o roteador ligado à interface local. A figura 34 demonstra graficamente o que foi descrito acima.

O dispositivo que requer o serviço de comunicação deve acessar os endereços específicos na memória e configurar adequadamente a solicitação seguindo a forma padrão já mencionada na seção 4.4.1. Observe que, dependendo do serviço, ocorrerá diferenças na forma de configuração da requisição. Na Figura 34 pode-se identificar este primeiro acesso à memória na Etapa 1.

No processo de escrita na memória para configuração do serviço pelo dispositivo requisitante, a interface de rede detecta a ação e pressupõe o cadastramento de uma solicitação de comunicação. Em seguida, segue-se a Etapa 2, onde a interface acessa o barramento e busca pelo serviço cadastrado na lista de envios. Observe que nem todos os serviços cadastrados necessariamente serão envios, tem-se recebimentos ou grupos. Isto significa que a interface de rede efetuará a busca e poderá não encontrar um envio ativo, neste caso a interface retorna ao estado de aguardo por solicitações vindas da rede ou do barramento.

Em caso de a interface encontrar uma solicitação ativa, esta é imediatamente atendida, gerando o envio de dados pela rede. A Etapa 3 visível na Figura 34 representa esta parte do processo. Os dados são lidos da memória local e enviados para o roteador que encaminhará a mensagem para o destino na rede, onde terá seu devido tratamento pela interface destino.

Existem, porém, duas formas da Etapa 3 ser realizada. Em uma primeira análise,

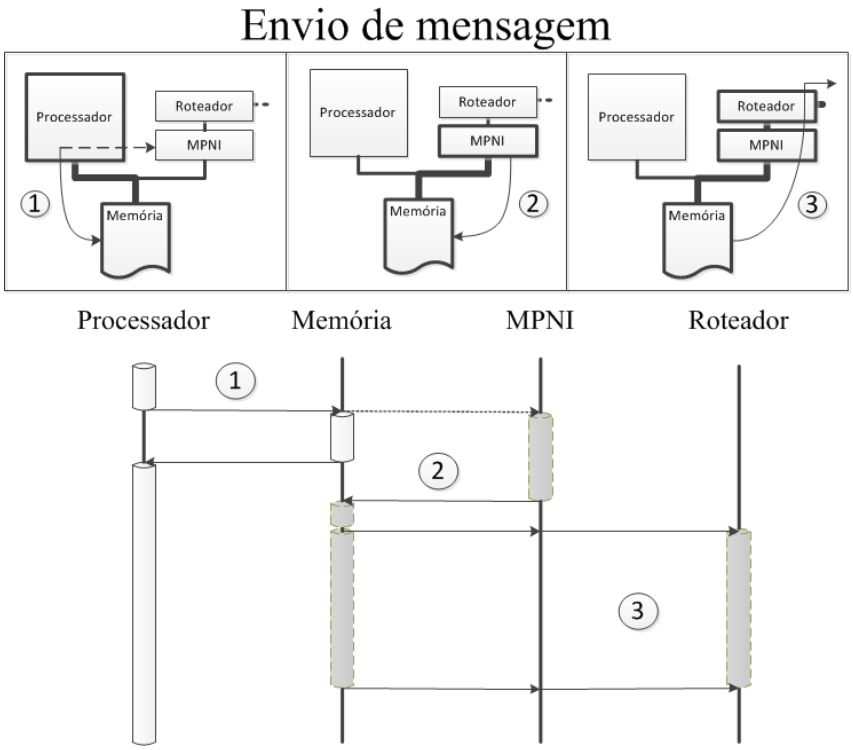


Figura 34 – Interações entre os dispositivos de um nó da rede durante um envio de mensagem.

pode-se ter um volume grande de dados sendo comunicados, assim os dispositivos ficariam impedidos de acessar a memória para realizar outros processos locais. Entretanto, há duas maneiras de a transferência de dados acontecer. Esta questão divide as operações de transferência de dados disponíveis em duas classificações: bloqueantes e não-bloqueantes. A Figura 35 organiza estas duas formas de operação em dois esquemas.

As operações bloqueantes (Figura 35-A) são aquelas em que a Etapa 2, isto é, o bloqueio do barramento, é mantido até a finalização do processo de envio da mensagem. Esta modalidade pode ser interessante na medida em que o processador local necessite de agilidade na entrega dos dados ao destino. Em contrapartida, na modalidade não-bloqueante apresentada no esquema da Figura35-B, o envio de dados é realizado em pequenas etapas, permitindo que os dispositivos locais acessem o barramento antes que o envio dos dados seja concluído. Esta técnica ganha em flexibilidade, porém perde em eficiência em termos de tempo para ser completada. As operações bloqueantes são sensivelmente mais rápidas que as não-bloqueantes. Contudo, existem ocasiões em que o processamento local pode ser protelado em detrimento a conclusão do envio dos dados. Nestes casos, o envio bloqueante estará mais adaptado às condições destas aplicações.

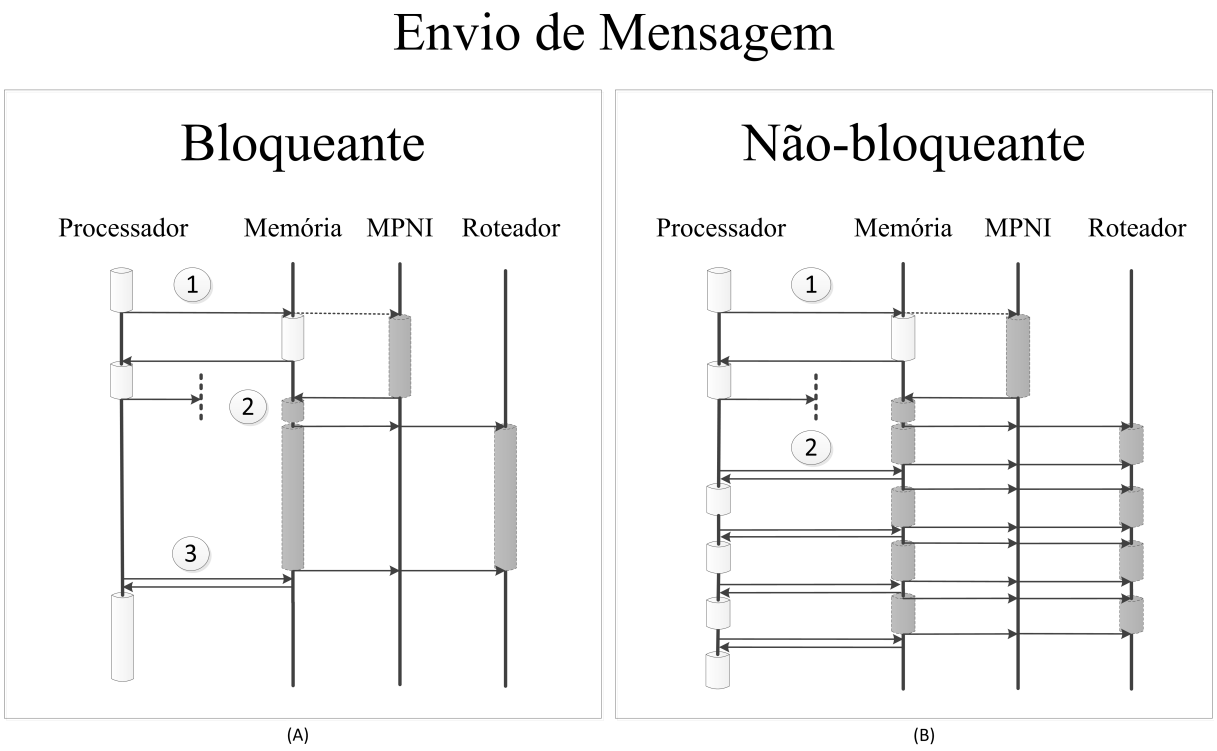


Figura 35 – Exemplo de interações bloqueantes e não-bloqueantes para um nó da rede durante um envio de mensagem.

4.6 EXEMPLO DE RECEBIMENTO - VISÃO DO SISTEMA OPERACIONAL

Análogo ao fragmento de código criado para cadastrar o envio de dados pela rede, a solicitação de *RECEIVE* deve ser cadastrada a fim de que haja a validação e recepção dos dados no destino. O Quadro 4.6 descreve um possível programa para cadastrar a solicitação de recebimento na referida fila (no exemplo, 0x0052). Observe que, assim como para o exemplo do Quadro 4.5, o código abaixo também é uma função recursiva onde o fim da fila é encontrado e, na sequência, uma nova solicitação é cadastrada. Neste exemplo, a constante PAN_ORIGEM_DESTINO corresponde aos *bits* de protocolo de alto nível e END_DADOS ao endereço onde os dados recebidos pela interface de rede devem ser copiados na memória local.

Quadro 4.6 - Rotina do SO manipulando a Fila de Recebimentos - RECEIVE

```
// Globais.
int *end_fila_recebimentos;
#define int end_inicial = 0x0052;


end_fila_recebimentos=end_inicial;
```

```
void Solicita_recebimento(void){
if(*end_fila_recebimentos==end_inicial)
{ // Fim da fila de recebimentos.

                //Aloca o novo pedido na fila.
                *end_fila_recebimentos=alocar_memoria(3);
                end_fila_recebimentos=*end_fila_recebimentos;

                //Inclui o requerimento de envio na fila.
                *end_fila_recebimentos=PAN_ORIGEM_DESTINO;
                end_fila_recebimentos++;
                *end_fila_recebimentos=END_DADOS;
                end_fila_recebimentos++;
                *end_fila_recebimentos=end_inicial;

}
else
{  // Busca o fim da fila de recebimentos.
        while(*end_fila_recebimento/=end_inicial)
        {
                end_fila_recebimento+=3;
                end_fila_recebimento=*end_fila_recebimento;
        }
   // Solicia cadastro na fila.
        Solicita_recebimento();
}
}
```

Assim, considerando que a fila de recebimentos foi previamente configurada pelo processador local ao executar a rotina descrita no Quadro 4.6, a Figura 36 mostra detalhadamente o papel de cada componente do sistema realizando a tarefa de recebimento dos dados gerados no exemplo da Seção 4.5. No detalhe, os passos 1 (chegada dos dados na interface de saída da rede) e 2 (escrita dos dados na memória local) são de responsabilidade da MPNI. No passo 3, o processador no destino pode acessar os dados.

Como visto nas operações de envio de mensagens, para o recebimento, podemos dividir o processo em três etapas distintas. Desde que a fila de serviço esteja previamente configurada a primeira etapa ocorre quando chegam dados pela interface do roteador do

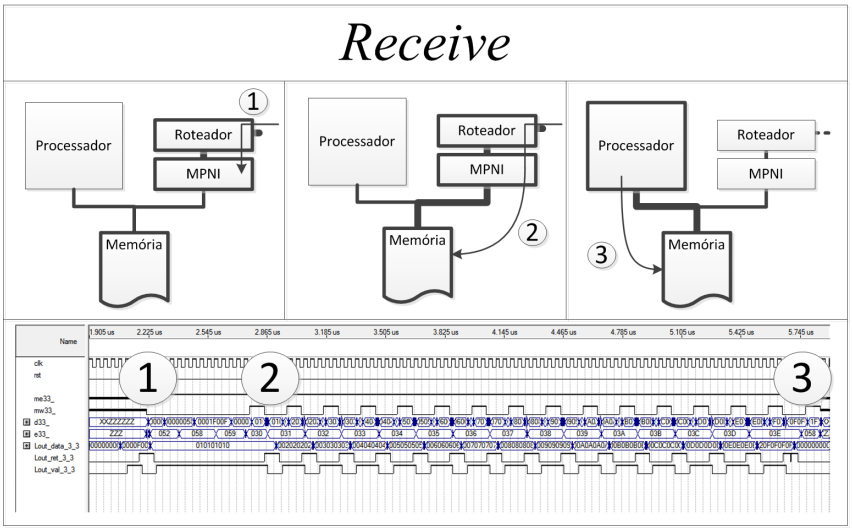


Figura 36 – Interações entre os dispositivos de um nó da rede durante um recebimento de mensagem cooperativo.

destino. A segunda etapa compreende de a MPNI realizar o acesso à fila de recebimentos para permitir a transferência de dados e, em um terceiro momento/etapa, onde os dados são aceitos e transferidos para a memória destino. A Figura 37 esquematiza o processo de recebimento de dados realizado no destino.

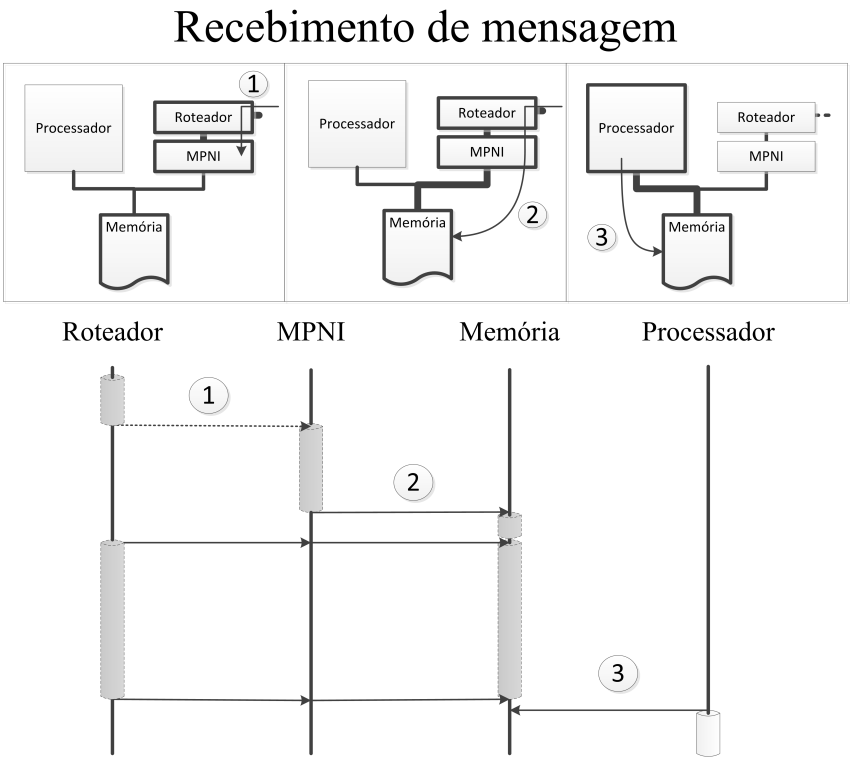


Figura 37 – Interações entre os dispositivos de um nó da rede durante um recebimento de mensagem.

A Etapa 1, conforme pode ser visto no esquema anterior, compreende o acesso ao barramento realizado pelo dispositivo requisitante de recebimento de dados. Esta etapa não gera, necessariamente, ações imediatas a serem executadas pela interface destino. Eventualmente, dados válidos chegam pelo roteador localmente ligado à MPNI e produz a consulta da memória local para certificação de que o recebimento é genuíno, isto é, que está previsto e que deva ser completado. Assim, a Etapa 2 ocorre. Esta compreende a busca pelas informações para tratar o recebimento dos dados. Caso o recebimento não esteja previsto, os dados são simplesmente descartados, liberando os *buffers* do roteador para novas entregas de dados. A Etapa 3 é caracterizada pela liberação do barramento para que o processador possa acessar as informações recém-chegadas.

Conforme foi visto para operações de envio de dados, no recebimento, os dados também podem ser entregues em duas formas. A primeira, conforme é possível ver na Figura 38 da esquerda, o processador é impedido de acessar a memória até que toda a operação de recebimento seja concluída. Para processos não-bloqueantes, os acessos à memória para recebimento dos dados vindos da rede são intercalados com os acessos do processador local. Esta medida tenta não atrasar o processamento já em curso quando da chegada de novas informações vindas de outro processador através da rede.

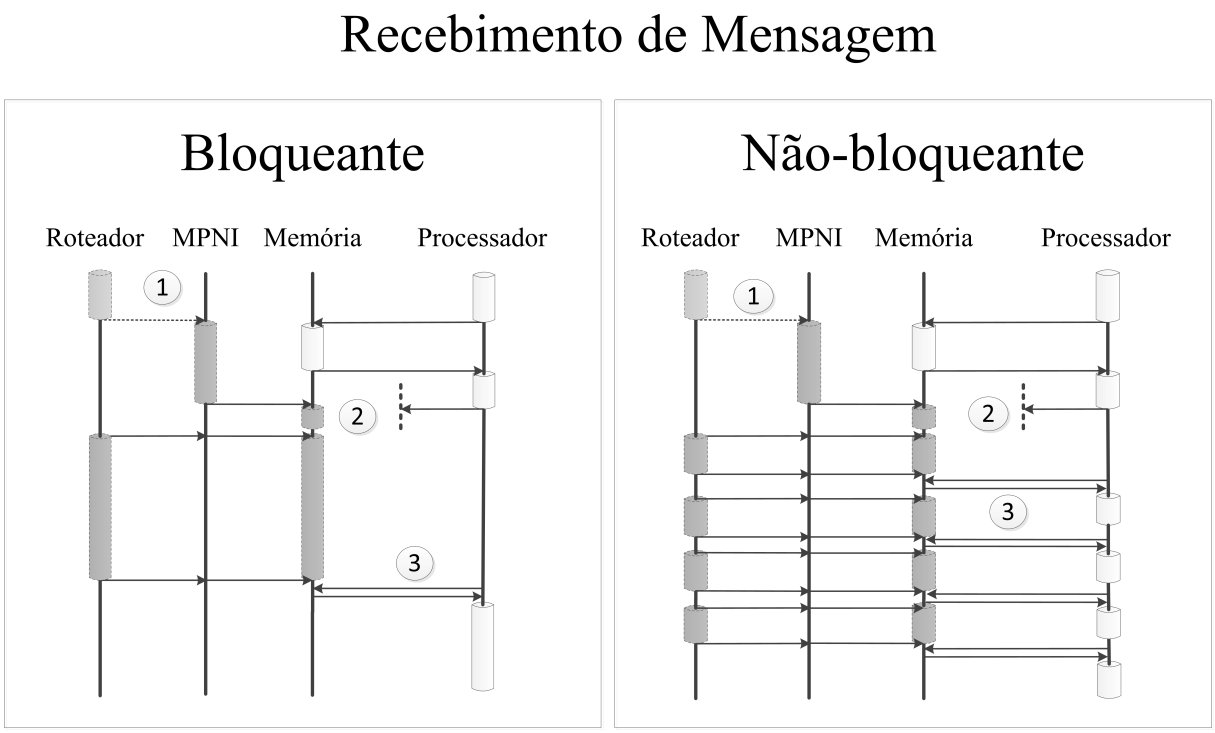


Figura 38 – Exemplo de interações bloqueantes e não-bloqueantes para um nó da rede durante um recebimento de mensagem.

Naturalmente ambos os recebimentos de informação proveniente de algum nó remoto conectado à rede são eficazes, entretanto a eficiência em termos de tempo para

conclusão da tarefa é superior em recebimentos bloqueantes quando comparados com os não-bloqueantes. Aqui se repete o observado nas operações de envio, porém, conforme já discutido, há aplicações para ambos os métodos em sistemas de processamento paralelo.

4.7 APLICAÇÃO SEND/RECEIVE - VISÃO DO PROGRAMA DO USUÁRIO

Conforme visto, a interface de rede MPNI permite ao sistema operacional disponibilizar alguns serviços de comunicação entre processadores aos programas de usuário. A região de memória onde encontra-se a fila de serviços é parametrizada em tempo de projeto do MPSoC de maneira a permitir que a fila de serviços esteja em região de memória endereçada apenas pelo sistema operacional. Esta opção mantém a segurança e a estabilidade do sistema, visto que a disponibilidade das filas de serviços diretamente para os programas de usuário não garante que as comunicações sigam qualquer padronização, levando a instabilidade ou, em certos casos, permitindo a ação de rotinas maliciosas. Assim, os programas de usuários apenas solicitam aos serviços por meio de funções do sistema operacional, que é responsável pelo acesso direto ao sistema de comunicação disponível por interface de rede e rede-em-chip.

Para que seja possível visualizar como o conjunto processador/interface de rede trabalha apresenta-se um exemplo de comunicação, desta vez percebida pelo programa de usuário. Neste exemplo, supõe-se que um sistema multiprocessado está funcionando com apenas um processador enquanto que os demais estão em modo *STOP* (resultado da execução do comando *TURN OFF* previamente). Nestas condições, um programa de usuário em execução no endereço de uma rede na topologia grelha-2D [0,0] pretende compartilhar informações a serem processadas por 3 outros processadores (nos endereços [2,0], [3,1] e [3,3]), perfazendo um grupo de comunicação com 4 participantes.

Inicialmente o programa do usuário deve solicitar ao sistema operacional a criação de um grupo com 4 participantes. Atendendo esta solicitação o sistema operacional deverá inscrever um grupo contendo 4 processadores na fila de grupos e, em seguida, cadastrar o comando *INITIALIZE* na fila de envios. Esta ação seguida da sobrescrita do endereço de início da fila de envios produz a execução do comando conforme Figura 39-A. Em sequência, o sistema operacional inicializa a memória dos processadores remotos com (1) o programa de usuário e (2) as requisições de *RECEIVE* na fila de recebimento dos dispositivos remotos. Neste caso, o comando para a passagem de informações a ser utilizado é o *PUT* (Figura 39-B), pois os demais processadores não possuem dados nas suas filas de serviços, não conseguindo realizar nenhuma transferência cooperativa com o processador criador do grupo. Ao concluir a inicialização das filas

de serviços e a passagem do programa de usuário para os processadores remotamente, os processadores podem trocar informações por meio de *SEND/RECEIVE*. Entretanto, para que a interface local a cada processador permita a realização de computação, é necessário que o comando *TURN ON* seja executado a partir do processador principal no grupo, Figura 39-C. O Quadro 4.7 demonstra o código do programa de usuário solicitando sequencialmente os serviços por meio de funções disponíveis pelo sistema operacional. Na Figura 39 esta esquematizado graficamente a troca de informações bem como os serviços da interface MPNI utilizados em cada passo.

Quadro 4.7 - Programa solicitando medidas do SO

```
int Id_grupo;

// SO: retorna a id do grupo criado.
// Requer: quantidade de processadores no grupo.
Id_grupo=Solicitacao_grupo(4);

// SO: inicializa o grupo identificado pelo id.
// Requer: Id do grupo.
Initialize(Id_grupo);

// SO: inicializa processadores o programa a executar.
// Requer: Id do grupo, endereco inicial do programa
// e quantidade de Bytes do programa.
Put(Id_grupo, endereco_codigo, qtd_bytes);

// SO: Habilita os processadores do grupo para
// processamento do programa inicializado.
// Requer: Id do grupo.
Turn_On(Id_grupo);

// SO: Envia dados para o programa compartilhado
// entre os processadores.
// Requer: Id do grupo, endereco inicial dos dados
// e quantidade de Bytes de dados.
Send(Id_grupo, endereco_dados, qtd_bytes);

// SO: Aguarda dados de processadores remotos.
// Requer: Id do grupo, endereco inicial para
// transferir dados e quantidade de Bytes de dados.
```

```
Receive(Id_grupo, endereco_dados, qtd_bytes);

// Aguarda a concluir recebimento dos dados processados.
while(!Receive_Status_bits(Id_grupo)){}

// SO: Desabilita os processadores do grupo.
// Requer: Id do grupo.
Turn_Off(Id_grupo);

// SO: Finaliza o grupo identificado com id.
// Requer: Id do grupo.
Finalize(Id_grupo);
}
```

A partir deste ponto da execução das tarefas de comunicação, o caminho está livre para a troca de informações entre os programas de usuário replicados nos demais processadores e o programa principal no processador em [0,0]. Assim, os dados podem ser enviados por meio de solicitações *SEND/RECEIVE*. Isto significa que os programas de usuário devem solicitar ao sistema operacional novas transferências de dados para as memórias dos outros processadores contidos no grupo, ações ilustradas em Figura 39-D e Figura 39-E.

O fragmento de programa de usuário repassado para os processadores remotos tratarão os dados encaminhados pela rede e executarão o envio dos dados já computados ao processador principal do grupo, Figura 39-E. Por fim, no término da execução do programa de usuário, o sistema operacional conclui a seção do grupo de comunicação por meio do comando *FINALIZE* (Figura 39-G) e *TUNR OFF* (Figura 39-F).

Em resumo, o programa de usuário solicita recursos ao sistema operacional para execução de uma rotina específica. Este se encarrega em reservar os processadores necessários (*INITIALIZE*); configurá-los para que seja realizada a transferência de instruções (rotina do programa de usuário) e dados a serem processados (comando *PUT*); liberar o sistema para executar a computação (*TURN ON*); distribuir e agrupar efetivamente as informações necessárias ao programa de usuário (comando *SEND/RECEIVE*); suspensão dos processadores (*TURN OFF*) e liberação de recurso (*FINALIZE*) solicitado pelo programa de usuário.

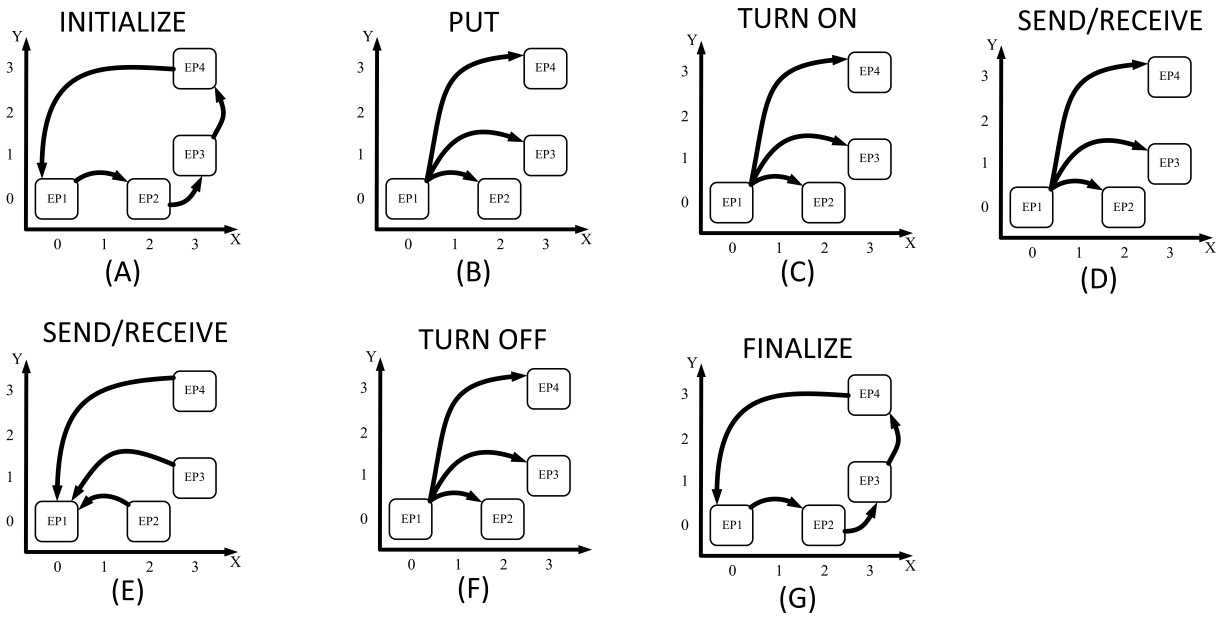


Figura 39 – Esquema em etapas para distribuir o programa de usuário realizado pelo Sistema Operacional.

4.8 RESUMO DO CAPÍTULO

Neste capítulo a interface de rede foi analisada em seus diferentes aspectos: arquitetura interna, esquema de memória e filas de serviços. Adicionalmente um exemplo de envio e de recebimento foi mostrado do ponto de vista das ações a serem tomadas pelo sistema operacional na execução da comunicação solicitada pelos programas de usuário. Aspectos como formato das filas de serviços e formato do pacote a ser comunicado (considerando a rede SoCIN-*fp*) também foram elencados. Por fim foi caracterizado a diferença entre os comandos bloqueantes e não-bloqueantes do barramento de comunicação entre processador/memória/MPNI. Além de mostrado os passo que o sistema operacional executa para a correta distribuição de processamento em um grupo com 4 processadores.

5 RESULTADOS

A interface de rede apresentada nesta dissertação foi implementada utilizando linguagem de descrição de *hardware*. Os resultados de síntese, bem como a simulação do circuito final, foram organizados de modo a permitir focar a análise crítica da latência produzida pela inserção da interface MPNI em projetos de multiprocessadores que utilizem redes-em-chip como meio de interconexão. Para isso, confronta-se o impacto da latência inserida pela interface proposta em comparação a não utilização desta técnica de interfaceamento.

5.1 SÍNTESE DA MPNI

A interface foi implementada utilizando linguagem de descrição de *hardware* (VHDL) e os resultados da simulação foram obtidos pelo uso do *software* Quartus II 9.1 do fabricante Altera. O dispositivo empregado na simulação foi o FPGA EP2C35F672C6 da família Ciclone II, fabricado na tecnologia de 90nm. A estimativa de uso de área em *chip* foi extraída da totalização do número de elementos lógicos (parcial ou completamente utilizados) na simulação. A potência dissipada foi obtida pela ferramenta *Powerplay Power Analyzer* disponível no Quartus II. Para a análise, foi definida uma taxa de chaveamento de 50% operando a um *clock* de 49,995MHz. A taxa de chaveamento nos informa que os sinais da arquitetura alternaram entre os níveis lógicos 0 e 1 durante 50% do tempo da simulação. Este nível de atividade é considerado elevado, mas foi adotado com o objetivo de definir a máxima potência requerida pelo módulo em toda a sua operação. A Tabela 3 apresenta um resumo da área em chip, potência dissipada e frequência máxima apenas para o circuito da interface de rede MPNI.

Elementos Lógicos (LEs)	700
Potência Máxima Dissipada	27,28mW
Clock Máximo	111,59MHz

Tabela 3 – Resultados de Síntese da interface de rede MPNI

Em uma análise comparativa de área, onde organiza-se o circuito de um nó em um arranjo geométrico hipotético, a MPNI sugere um uso em área de aproximadamente o dobro de um roteador ParIS e um terço do tamanho do processador Nios II (versão *fast*). A Figura 40 demonstra uma comparação entre áreas dos diferentes componentes

de um nó de um multiprocessador hipotético composto por roteador (PaRIS - 360 EL), interface de rede (MPNI - 700 EL) e (Nios II *fast* -2200 EL).

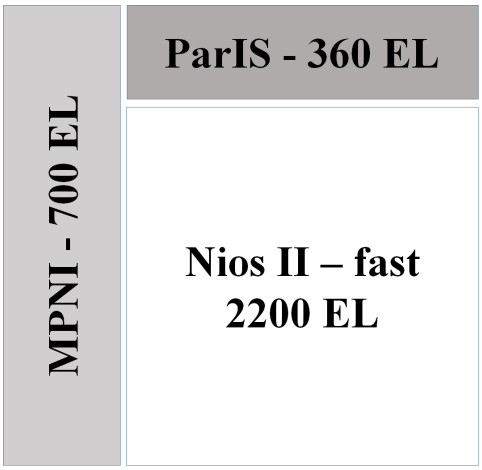


Figura 40 – Quadro comparativo entre áreas em chip para cada elemento de um nó composto por roteador (PaRIS - 360 EL), interface de rede (MPNI - 700 EL) e (Nios II *fast* -2200 EL).

Em uma análise comparativa, para avaliarmos a potencialidade do uso de redes-em-chip e interfaces de rede na conexão de vários elementos processantes, foram levantados dados de síntese dos circuitos separadamente. Somando-se a quantidade de *Logic Elements* do processador Nios II, versão *Standard* (1030 LEs) (ALTERA, 2011), em conjunto com a interface MPNI (700 LEs), utilizando roteadores ParIS (360 LEs) (ZEFERINO; SANTO; SUSIN, 2004), é possível calcular quantos nós podem ser utilizados em um projeto hipotético de multiprocessador. Estimando-se a área em *chip*, considerando apenas quantidade de elementos lógicos, chega-se à Tabela 4.

FPGA	Elementos Lógicos Disponíveis	Processadores Integrados (EP+MPNI+ParIS)
EP1C20 (130nm)	20.060	18
EP2C70 (90nm)	68.416	62
EP3C120 (65nm)	119.088	109
EP4C6X150 (60nm)	149.760	137
5CEFA9 (28nm)	301.000	275

Tabela 4 – Projeção da quantidade de processadores integráveis em único FPGA

Para o quadro comparativo são considerados apenas FPGAs das famílias *Cyclone* I, II, III, IV e V. Estes são dispositivos de baixos custo e consumo de energia comercializados pelo fabricante Altera. Em outro aspeto do experimento não se contabiliza a área gasta com memória local, considera-se a utilização de um *chip* externo de memória dedicado.

5.2 AVALIANDO A LATÊNCIA DOS SERVIÇOS DA INTERFACE

Comparando duas aplicações de redes-em-chip, uma sem interface de rede e outra com a interface de rede MPNI, pode-se avaliar o impacto dos processos da interface proposta na latência total observada na comunicação ponto a ponto. Para a obtenção dos resultados de simulação foram montados os circuitos referentes a rede-em-chip SoCIN-*fp* e interface de rede MPNI. O processador não foi instanciado, sendo os sinais de dados, endereço e controle provindos do processador apenas simulados.

Para cada serviço, as medições seguiram um padrão de comparação onde avalia-se a diferença percentual do tempo requerido apenas pela rede-em-chip na tarefa de propagação da mensagem e o tempo acrescido pela interface de rede MPNI no mesmo processo. Cada serviço possui um comportamento peculiar e foi analisado separadamente levando-se em conta alguns parâmetros que pudéssemos comparar entre uma dada aplicação que faça uso da interface de rede proposta e outra que não a utilize.

5.2.1 Análise de Tempo para o Serviço INITIALIZE/FINALIZE

O tempo necessário para a inicialização ou extinção de um dado grupo de comunicação depende da (i) quantidade de processadores listados e (ii) da distância (contada em *hops*) entre estes na rede. A distância varia com a topologia, entretanto, para que a operação seja concluída o mais rápido possível é mandatório que a quantidade de *hops* na execução da operação seja a menor possível. Esta característica é obtida quando o número de processadores se assemelha a quantidade de *hops* no caminho do pacote até atingir o processador inicial.

Para a obtenção dos resultados foram analisados os dados conseguidos a partir da simulação de uma rede em grelha 2-D, 4x4, onde cada roteador ParIS está conectado a uma interface de rede MPNI. Cada interface de rede está conectada a um barramento onde se tem acesso à memória local. As solicitações de criação/extinção de grupo são disparadas a partir do nó (0,0). O pacote é injetado na rede e segue o caminho conforme mostrado na Figura 41. Os resultados da simulação estão organizados na Tabela 5.

A simulação foi realizada na ferramenta *Simulator Tool* disponível no *software* Quartus II 9.1 *Web Edition* do fabricante Altera. Foram realizados 4 experimentos para aquisição dos dados necessários à análise. A Figura 42 mostra os resultados observados para o teste com 2 processadores. Nela é possível distinguir cinco momentos que marcam o processo realizado pela interface na execução do comando. Em 1 o processador na

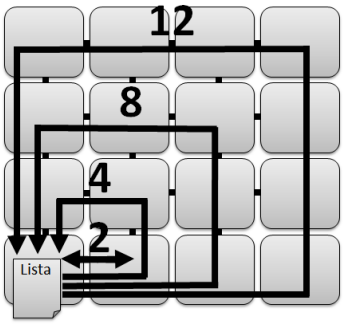


Figura 41 – Esquema de experimentação para os serviços *INITIALIZE/FINALIZE*.

coordenada (0,0) solicita acesso à MPNI local pelo canal de *sam00* (solicitação de acesso à memória), levando o sinal de nível lógico baixo para alto. O sinal de liberação de acesso é dado em *alm00* (acesso liberado à memória local) seguindo o mesmo comportamento. Neste instante o processador acessa a região de memória que direciona para a fila de envios e sobrescreve a região de memória com o mesmo valor encontrado. Esta ação ativa a interface de rede que opera automaticamente. Em 2 o pacote com as informações para a execução do comando são enviadas pela rede. Os *bits Lin_val*_00 (dado válido na entrada da rede) e *Lin_ret*_00 (dado recebido pela rede com sucesso) demonstram o envio de dois *flits*. Em 3 pode-se perceber a escrita na memória localmente ao processador em (0,1), é nesta oportunidade que a lista do grupo a ser inicializado é salvo na memória e os dados são reenviados para a origem para confirmação da criação de grupo. Em 4 o processador em (0,0) recebe a lista novamente e atualiza o *status* da lista recém criada. Em 5 o processo de inicialização é totalmente concluído pela interface. É importante ressaltar que o processo inverso, isto é, a finalização do grupo, segue a mesma dinâmica.

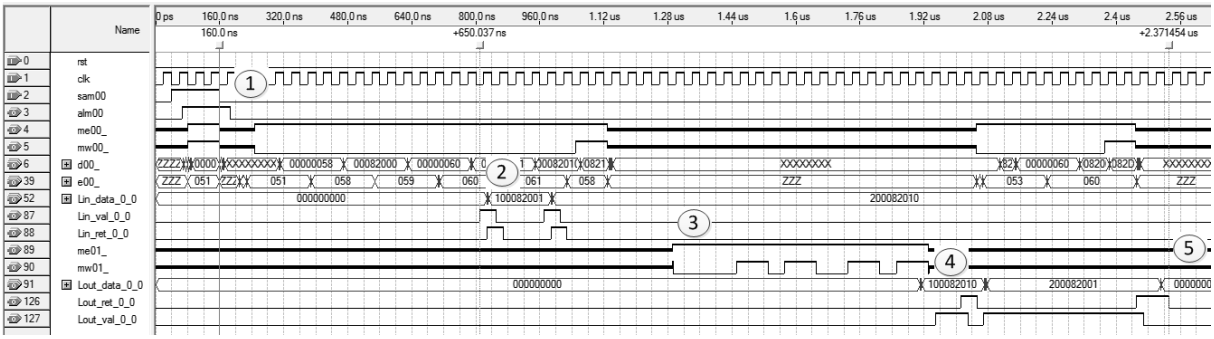


Figura 42 – Resultado da simulação no *software* Quartus II 9.1 *Web Edition* para o serviço *INITIALIZE/FINALIZE* com 2 processadores.

Os experimentos seguiram o mesmo já descrito apenas se ampliando o número de processadores listados no grupo. Todos os dados foram normalizados e sintetizados na Tabela 5. A simulação foi gerada utilizando o*clock* de 25,000 MHz, valor padrão adotado na experimentação de todos os serviços da interface.

Quantidade de Processadores	Tempo Utilizado (µs)	Ciclos de *Clock*
2	2,371454	119
4	4,131454	207
8	7,813889	391
12	11,651805	583

Tabela 5 – Resultados da simulação para o comando *INITIALIZE/FINALIZE* em um sistema com o *clock* de 25,000 MHz.

Para avaliar o tempo necessário na inicialização ou finalização de um grupo contendo uma quantidade variável de processadores, plotou-se o gráfico da Figura 43. Tem-se o tempo total utilizado na execução do comando *INITIALIZE/FINALIZE* no eixo das coordenadas e a quantidade de processadores contidos no grupo no eixo das abcissas. O tempo requerido para a execução do comando depende diretamente da quantidade e da localização dos dispositivos incluídos no grupo a ser inicializado ou finalizado.

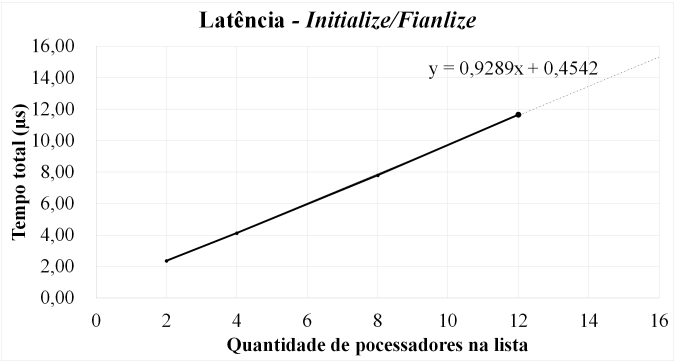


Figura 43 – Tempo para execução do comando *INITIALIZE/FINALIZE* versus quantidade de processadores na lista de grupo.

5.2.2 Análise de Tempo para o Serviço TURN ON/OFF

Analisando o tempo necessário para concluir a operação de ligar ou desligar núcleos em tempo de execução, distingue-se apenas um fator determinante para a variação de resultados. Não considerando o atraso produzido por excesso de tráfego na rede, a execução com comando *TURN ON/OFF* é dependente exclusivamente da distância em *hops* entre origem e destino. Este comando utiliza-se de apenas dois *flits*. Isto decorre de que este é o comprimento mínimo da mensagem que trafega pela rede-em-chip SoCIN-*fp*. Para experimentar este atraso, foram realizados testes onde um processador origem localizado no endereço (0,0) da rede executa o comando de ligar e

desligar os processadores situados em nós distantes 2, 3, 5 e 7 *hops* do processador que originou as solicitações. A Figura 44 demonstra esquematicamente os testes realizados.

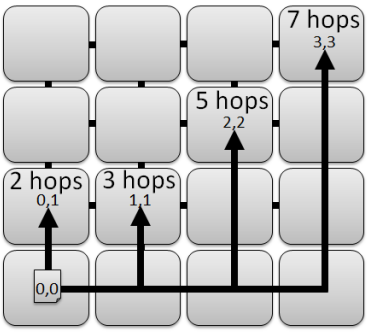


Figura 44 – Rota dos pacotes seguindo o roteamento XY na simulação dos comandos *TURN ON/OFF* para 2, 3, 5 e 7 *hops*.

Na Figura 45 é mostrada a simulação no Quartus II *Web Edition*. Pela análise da atividade dos sinais é possível distinguir três momentos importantes. Em 1 o processador em (0,0) sinaliza o disparo da solicitação para a MPNI local. O processamento da requisição ocorre instantaneamente e resulta no envio da solicitação pela rede. Em 2 os dados são remetidos pela rede. Em 3 a informação chega ao destino em (1,1) e produz o efeito desejado. É importante observar que o estado da interface de rede no destino é "000D"(desligada), após o recebimento dos dados a interface muda o estado para 000E (executando).

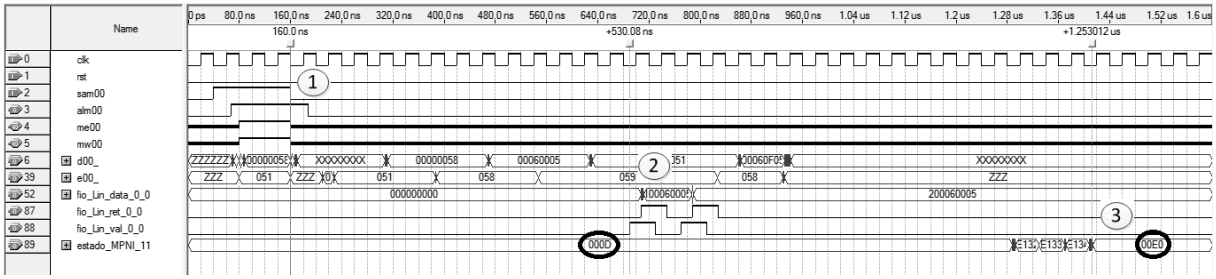


Figura 45 – Resultado da simulação no *software* Quartus II *Web Edition* para o serviço *TURN ON* com 3 *hops* entre origem e destino da solicitação.

A análise de tempo considerou o *clock* de 25,000 MHz e engloba o processo desde iniciada a execução na origem até a conclusão do comando no nó alvo. Os dados de simulação foram coletados e tabulados na Tabela 6.

Para o comando *TURN OFF*, os resultados foram ligeiramente diferentes conforme sugere a Tabela 7. Isto é decorrente da semelhança de tratamento dado pela interface para ambos comandos.

Na Figura 46 foram organizados os tempos médios observados na execução dos comandos *TURN ON/OFF* levando em conta a quantidade de *hops* no caminho do pacote.

Distância entre os Processadores (*hops*)	Tempo Utilizado (µs)	Ciclos de *Clock*
2	1,053167	27
3	1,253012	32
5	1,654157	42
7	2,054347	52

Tabela 6 – Resultados da simulação para o comando *TURN ON* em um sistema com o *clock* de 25,000 MHz.

Distância entre os Processadores (*hops*)	Tempo Utilizado (µs)	Ciclos de *Clock*
2	1,134269	29
3	1,334685	34
5	1,734048	44
7	2,133603	54

Tabela 7 – Resultados da simulação para o comando *TURN OFF* em um sistema com o *clock* de 25,000 MHz.

Como não há serviço homólogo disponível na rede-em-chip utilizada nesta dissertação, não gerou-se comparação percentual entre a abordagem utilizando a interface de rede e a sem utilizá-la.

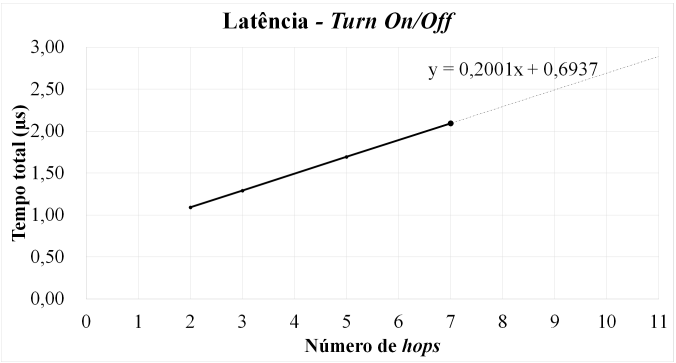


Figura 46 – Tempo médio na execução do comando *TURN ON/OFF* versus quantidade de *hops*.

5.2.3 Análise de Tempo para o Serviço SEND/RECEIVE

Para a análise dos testes do par de comandos *SEND* e *RECEIVE* foi montado o experimento conforme a Figura 44 mostrada anteriormente. Neste esquema, o processador conectado ao roteador (0,0) da rede-em-chip SoCIN-*fp* enviará certa quantidade de dados para os nós (0,1), (1,1), (2,2) e (3,3), distantes 2, 3, 5 e 7 *hops*, respectivamente.

Em uma análise da composição dos tempos gastos pela interface e rede de interconexão no envio e recebimento de um pacote, chega-se à Figura 47. A interface de rede e o roteador operam de forma complementar. Inicialmente a MPNI da origem dos dados identifica um envio ativo cadastrado na lista local. Em seguida a interface de rede acessa as informações e inicia-se a transmissão do pacote para a rede-em-chip. Nesta etapa da comunicação, SoCIN*fp* (ZEFERINO; SANTO; SUSIN, 2004) e MPNI de origem operam cooperativamente na transmissão do pacote. Por fim, os dados são recebidos pela MPNI de destino, tratados e, as informações úteis, são gravadas na memória, tornando-se disponível ao processador no destino.

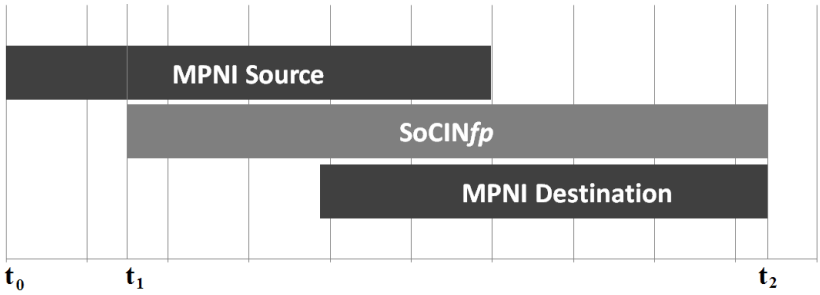


Figura 47 – Representação dos componentes de latência observados para todos os comandos quando analisado em pares de comunicação origem-destino.

A simulação mostrada na Figura 48 foi obtida com o auxílio da ferramenta *Simulator Tool* disponível no *software* Quartus II *Web Edition*. Nela, assim como nas simulações anteriores, distingue-se quatro momentos. Em 1 ocorre o disparo da solicitação. Em 2 inicia-se a transferência dos dados pela rede. Aqui verifica-se 9 *flits*, porém o primeiro é de cabeçalho e os demais de carga útil. Em 3 o destino recebe os dados e inicia a gravação na memória local. Por fim, em 4, a transferência é completada.

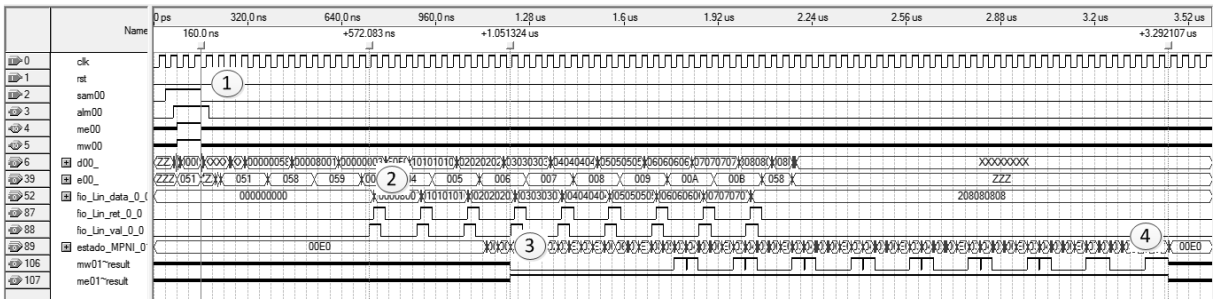


Figura 48 – Resultado da simulação no *software* Quartus II *Web Edition* para o serviço *SEND/RECEIVE* com 2 *hops* entre origem e destino da solicitação e envolvendo 8 *flits* de *payload*.

Analisando o tempo gasto por cada agente no processo, tem-se que as atividades pré-empacotamento dos dados, isto é, acesso à fila de serviços e interpretação do

comando a ser realizado, demandam o mesmo tempo para todos os envios realizados pelo uso do comando *SEND*. Esta é uma característica da técnica de filas de serviços, onde o processador cadastra solicitações e dispara a execução dos comandos cadastrados. O tempo t_1, para as simulações à frequência de 25,000 MHz, foi de 0,572083 µs. Este tempo de preparação do pacote é contabilizado como a latência inserida no processo completo de envio e recebimento, pois o restante é inerente ao processo de envio dos dados pela rede-em-chip. O tempo total decorrido na execução de envio pela origem e recebimento dos dados no destino foram contabilizados e organizados de acordo com a quantidade de *flits* enviados e a distância entre os pontos comunicantes (contada em *hops*). A Tabela 8 sintetiza os resultados da contagem de tempo total gasto no processo de transferência dos dados.

Quantidade de *flits* x *hops*	2 *hops*	3 *hops*	5 *hops*	7 *hops*
2 *flits*	2,092107 µs	2,291910 µs	2,692035 µs	3,090172 µs
4 *flits*	2,492107 µs	2,691910 µs	3,092035 µs	3,490172 µs
8 *flits*	3,292107 µs	3,491910 µs	3,892035 µs	4,250172 µs
16 *flits*	4,692107 µs	4,891910 µs	5,291275 µs	5,690172 µs

Tabela 8 – Resultados dos tempos ($t_2 - t_0$) observados na realização das trocas de mensagens pelo uso do comando *SEND/RECEIVE* em um sistema com o *clock* de 25,000 MHz.

Os dados de *clock* que correspondem aos mesmos tempos organizados na Tabela 8 foram plotados na Figura 49). É possível observar o comportamento linear do conjunto interface de rede e rede-em-chip quando variamos a quantidade de *Flits*, dado um número de *hops* constante.

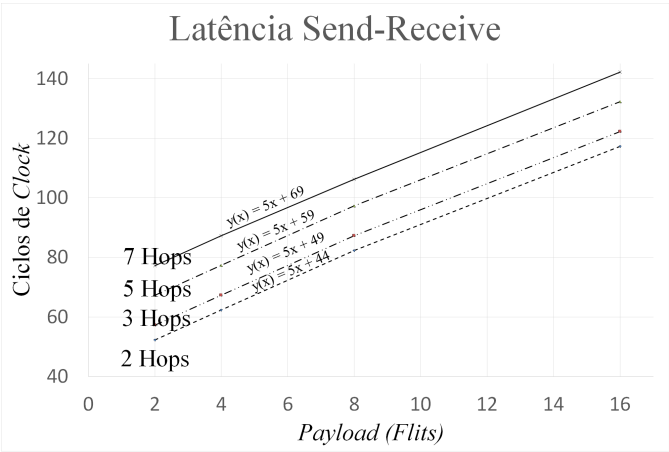


Figura 49 – Relação entre pulsos de *clock* necessários e quantidade de *Flits* enviados pelo uso dos comandos cooperativos *SEND/RECEIVE*.

Por meio dos dados da Tabela 8, é possível calcular o incremento percentual na

latência total requerida para a interface concluir a operação analisada. A Equação 5.1 mostra a relação.

$$Percentual = \frac{t_1 - t_0}{t_2 - t_0} \tag{5.1}$$

Tomando-se o intervalo de tempo gasto pela MPNI antes do envio da informação pela rede-em-chip (t_1-t_0) como sendo 0,572083 µs e o tempo total (t_2-t_0) usado para comunicar o pacote, a latência acrescida pela interface representou um percentual entre 28% e 10%. Porém, para pacotes contendo *payloads* acima de 16 *flits*, a porção de tempo adicional provocada pelo uso da interface MPNI cai para menos de 13% da latência total, independentemente da quantidade de *hops*. Os dados foram representados graficamente na Figura 50.

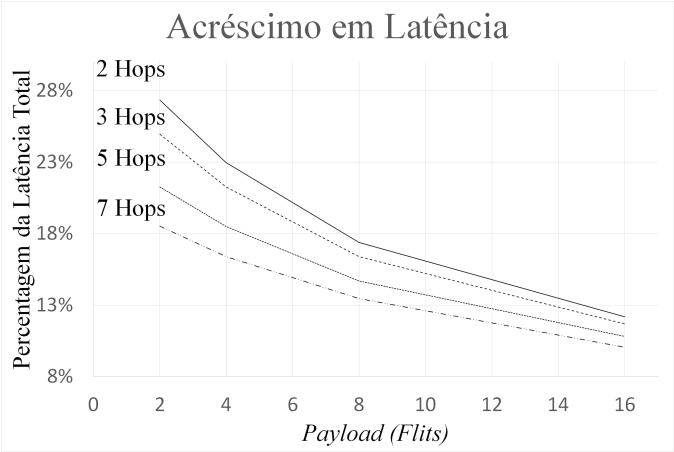


Figura 50 – Relação entre quantidade de latência acrescida por *Flits* enviados pelo uso dos comandos cooperativos *SEND/RECEIVE*.

5.2.4 Análise de Tempo para o Serviço PUT

Assim como o realizado nas avaliações anteriores, o esquema de teste do serviço *PUT* seguiu o mostrado na Figura 44. Semelhante ao experimento da avaliação dos comandos *SEND/RECEIVE*, serão realizados envios por meio do comando *PUT*, variando-se a quantidade de *flits* enviados e o número de *hops* no caminho do pacote. A Figura 51 demonstra um dos resultados do experimento em questão. Nela identifica-se os quatro momentos principais. Em 1 ocorre o disparo da solicitação. Em 2 o envio de dados pela rede. Em 3 o início do recebimento dos dados no destino. Por fim, em 4, a conclusão da escrita dos dados na memória localmente ao processador no nó (3,3), distante 7 *hops* do nó remetente.

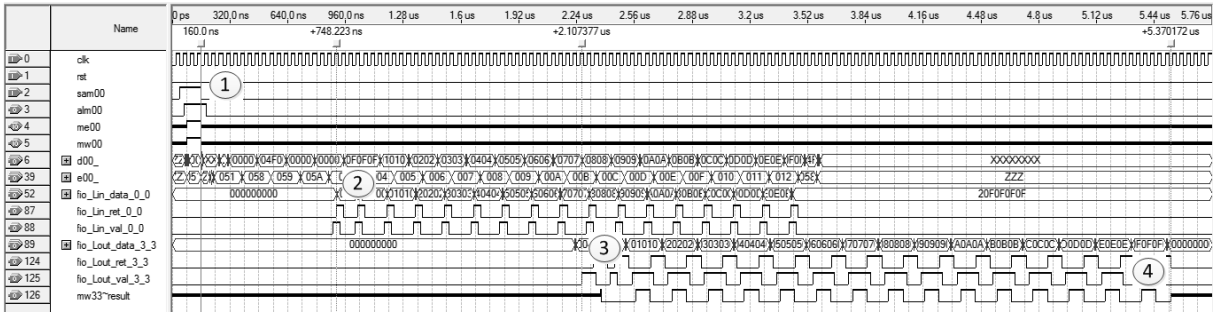


Figura 51 – Resultado da simulação no *software* Quartus II *Web Edition* para o serviço *PUT* com 7 *hops* entre origem e destino da solicitação e envolvendo 16 *flits* de *payload*.

Os tempos obtidos a partir do experimentos variaram conforme sintetizado na Tabela 9.

Quantidade de *flits* x *hops*	2 *hops*	3 *hops*	5 *hops*	7 *hops*
2 *flits*	1,772107 µs	1,971910 µs	2,372035 µs	2,770172 µs
4 *flits*	2,172107 µs	2,371910 µs	2,772035 µs	3,170172 µs
8 *flits*	2,972107 µs	3,171910 µs	3,572035 µs	3,970172 µs
16 *flits*	4,372107 µs	4,571910 µs	4,972035 µs	5,370172 µs

Tabela 9 – Resultados dos tempos ($t_2 - t_0$) observados na execução do comando *PUT* em um sistema com o *clock* de 25,000 MHz.

Os dados de *clock* que correspondem aos mesmos tempos organizados na Tabela 9 foram plotados na Figura 52). É possível observar o comportamento linear do conjunto interface de rede e rede-em-chip, assim como percebido para o comando *SEND/RECEIVE*, quando variamos a quantidade de *Flits*, dado um número de *hops* constante. Entretanto a quantidade de *clocks* previstos para operações *SEND/RECEIVE* são, em média, menores que os tempos necessários para o comando *PUT*.

Utilizando novamente a Equação 5.1, pode-se avaliar o incremento percentual em latência gerada pela interface. Como esperado, a latência da comunicação cresce linearmente a medida em que mais *flits* são adicionados à mensagem. Ao considerar distâncias maiores, o gráfico é deslocado para cima proporcionalmente ao número de *hops* utilizado. A Figura 53 mostra a relação existente entre tempo e quantidade de palavras comunicadas para diferentes quantidades de *hops*.

Conforme já discutido e mostrado na Figura 47 anteriormente, o intervalo de tempo entre t_0 e t_1 é utilizado pela interface de rede para atender à solicitação de comunicação cadastrada na lista de envios pelo processador local. Tomando-se o intervalo de tempo gasto pela MPNI antes do envio da informação pela rede-em-chip (t_1-t_0) e o tempo total (t_2-t_0) usado para comunicar o pacote, a latência acrescida pela

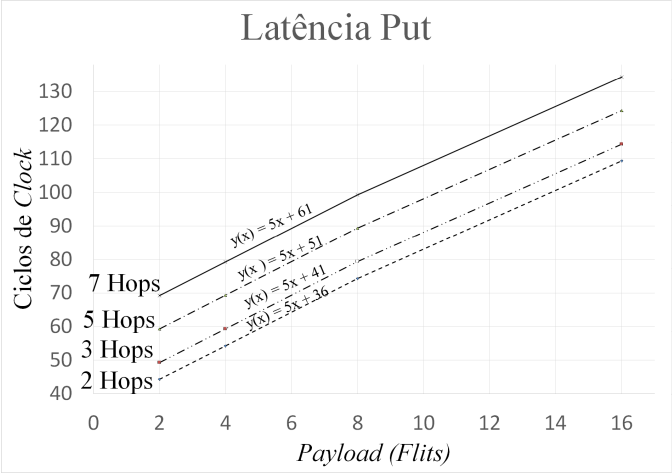


Figura 52 – Relação entre pulsos de *clock* necessários e quantidade de *Flits* enviados pelo uso do comando unilateral *PUT*.

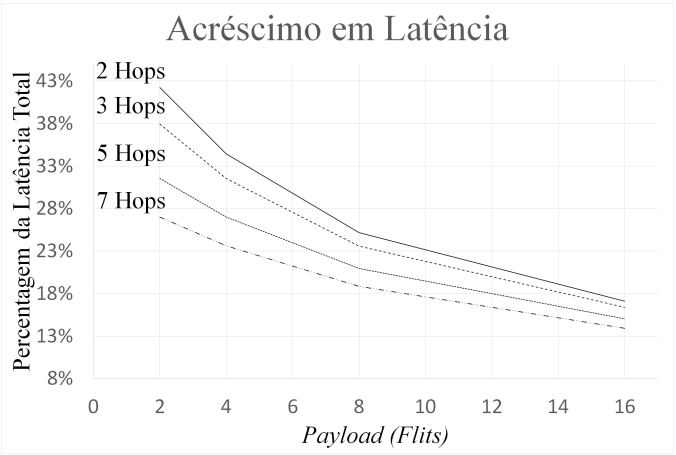


Figura 53 – Relação entre quantidade de *flits* enviados e latência da comunicação na simulação do comando *PUT*.

interface representou um percentual entre 42% e 16%. Porém, para pacotes contendo *payloads* acima de 16 *flits*, a porção de tempo adicional provocada pelo uso da interface MPNI cai para menos de 19% da latência total, independentemente da quantidade de *hops*. A Figura 53 demonstra o comportamento da latência adicional em percentual do tempo total da comunicação para *payloads* contendo 2, 4, 8 e 16 *flits*, para diferentes quantidade de *hops*.

Em linhas gerais, uma das características identificadas pela análise dos dados de latência observados, ressalta-se que a parcela de tempo acrescida pela interface de rede na latência total da mensagem decresce com o tamanho da mensagem ou o número de *hops*, isto é, a distância entre transmissor e receptor. Como o tempo de preparação para o envio de dados pela MPNI é constante, quanto maior o *payload* ou o número de *hops* entre origem e destino, menos representativa torna-se a parcela da latência associada apenas à MPNI. É importante salientar que a interface de rede no destino não

acrescenta parcela considerável na latência total, pois opera em complemento à rede de interconexão, apenas gravando os dados recebidos na memória local.

5.2.5 Análise de Tempo - Quadro Resumo

Foi organizado, na Tabela 10, os tempos mínimos, bem como a quantidade de pulsos de *clock* necessários para que cada tipo de operação seja realizada. O tempo total depende do comando, da quantidade de nós envolvidos ou número de *hops*, com já discutido anteriormente.

Comando	Tempo Mínimo (μs)	Ciclos de *Clock*
INITIALIZE/ FINALIZE	0,4542	12
SEND/ RECEIVE	0,5721	15
PUT	0,7582	19
TURN ON/ TURN OFF	0,6937	18

Tabela 10 – Síntese dos tempos mínimos requeridos para a MPNI executar o respectivo comando em um sistema com o *clock* de 25,000 MHz.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Nesta dissertação abordou-se o tema interface de rede para redes-em-chip, realizando um breve histórico desde o surgimento das redes intra-chip até as atuais pesquisas em interfaces de rede. No referencial teórico revisou-se conceitos de redes intra-chip como topologias, métodos de roteamento, controle de fluxo, memorização, interfaces de rede. A especificação MPI 3.0 (FORUM, 2012) foi apresentada com o objetivo de dar ao leitor uma perspectiva das principais operações escolhidas para compor os serviços básicos da interface de rede MPNI.

A abordagem *hardware/software* apresentada é uma solução intermediária às estudadas por Bhojwani et. al. (BHOJWANI; MAHAPATRA, 2006). Uma desvantagem de sistemas de interface é a inclusão de latência na comunicação. Contudo, há ganhos no tocante ao desacoplamento entre computação e comunicação, favorecendo assim, o desempenho das aplicações distribuídas (LEE et al., 2008), resultando na sensível redução da carga de trabalho sobre o processador.

Foi apresentada, a título de prova do conceito, uma aplicação da interface de rede MPNI no desenvolvimento da solução de interconexão de um multiprocessador contendo 16 processadores. A rede utilizada foi a rede-em-chip SoCIN*fp* (ZEFERINO; SANTO; SUSIN, 2004) conjugada a interface de rede MPNI. Este esquema mostra-se interessante pois visa reduzir o tempo total do projeto do MPSoC, ao passo que simplifica as transações de comunicação entre os processadores e não requer modificação de nenhum dos componentes empregados.

A solução proposta concentrou-se em aspectos da comunicação em MPSoCs através do uso de redes-em-chip, permitindo o gerenciamento da comunicação a partir da interface de rede MPNI. Esta proposta está de acordo com o modelo de camadas de serviços, restringindo o escopo às operações de multiprogramação baseadas em MPI.

A interface MPNI dispõe de parametrização em tempo de projeto, sendo configurável de acordo com os requisitos de cada aplicação multiprocessada e a rede utilizada (SoCIN*fp*). Os elementos processantes ligados à interface não sofreram quaisquer modificações em seu hardware, reflexo do protocolo de comunicação utilizado entre o processador e a interface de rede, exequível através de operações de *load* e *store* em um *cache* compartilhado entre ambos. Esse fator mostrou-se como uma forte vantagem do esquema, diminuindo a necessidade de modificação das partes do sistema durante a fase de projeto, agilizando sua conclusão.

Como principal contribuição do trabalho proposto está a criação de uma interface de rede sintetizável em nível de transferência entre registradores (RTL, *Register Transfer Level*) que oferece uma camada de serviço para troca de mensagens através de transações

eficientes baseadas nos métodos descritos no padrão MPI 3.0 (FORUM, 2012). A interface desenvolvida é parametrizável em tempo de projeto e compatível inicialmente com a rede SoCIN*fp*, podendo ser expandida para qualquer outra rede *softcore*. Um protótipo foi descrito em linguagem de descrição de *hardware* e distribuído para livre utilização por pesquisadores e projetistas. Abaixo enumera-se as contribuições da interface de rede MPNI proposta neste trabalho de dissertação.

1. Oferece a capacidade de conectar-se uma NoC a qualquer EP/IP que execute operações *load/store* em memória, sem necessidade de modificar o projeto dos dispositivos envolvidos;

2. Disponibilização de operações de rede para computação paralela através de uma abordagem instrucional por meio da programabilidade da interface;

3. Apesar de incrementar a latência total do sistema, a técnica de filas prevê uma mitigação do impacto da latência produzida, tendo em vista que a preparação do pacote é realizada *off-line*, ou seja, a rede é acionada apenas quando da comunicação do cabeçalho e dados;

4. Operações realizadas sem intervenção do EP/IP envolvido – baseadas em *Direct Memory Access* (DMA) - permitindo que o dispositivo fique livre para executar computação.

5. Permite a criação de multiprocessadores heterogêneos, isto é, com diferentes tipos de processadores interconectados pela rede-em-chip. O programa executado por cada processador é independente dos demais.

6. A interface disponibiliza operações que permitem o Sistema Operacional gerenciar a energia gasta em tempo de execução utilizando comandos *TURN ON/OFF* presentes na implementação da interface.

Uma das desvantagens principais na utilização deste sistema é o acesso ao barramento. Condições de disputa pela memória podem ocorrer entre processador e interface de rede, causando atrasos na transmissão ou na computação. A interface de rede empregada produz um acréscimo na latência da comunicação na ordem de (40%) para mensagens contendo poucos *flits* de *payload*, entretanto esta parcela de latência adicional é reduzida pela metade quando comparada ao tempo total da transmissão de uma quantidade maior de dados ou a maior número de *hops* observados no caminho do pacote. Contudo, outras técnicas podem ser desenvolvidas contemplando meios de reduzir a latência adicional para pequenas mensagens ou transmissões curtas (poucos *hops*), acelerando estas trocas de mensagens. Uma solução poderia surgir da utilização de um sistema de interface de rede híbrido, unindo as soluções propostas nesta dissertação

e em (GARCIA, 2009). Mensagens pequenas seriam enviadas por meio de escritas em endereços fictícios de memória, sendo encaminhadas diretamente pela rede para o destino específico. Para mensagens maiores a solicitação seria cadastrada e executada dedicadamente pela interface. Esta segunda técnica favoreceria a transferências de dados em rajadas, onde o tempo de transferência médio por palavra é reduzido.

Este IP *core* possibilita a rápida integração de processadores em projetos de MPSoCs, aproveitando as principais características das redes-em-chip (escalabilidade, tolerância a falhas, comunicação simultânea, etc.). Além disto, por meio deste, sistemas multiprocessados podem ser projetados adaptando-se às necessidades da aplicação, atingindo capacidades de processamento elevadas com baixos custo e consumo de energia, ideais para aplicações embarcadas. Versões posteriores desta interface podem ampliar as opções de comandos da especificação MPI (*gather*, *scatter*, *reduce*, *broadcast*, *barrier*, etc.)(FORUM, 2012). Disponibilizá-los por meio da escolha de subconjuntos dos comandos disponíveis em tempo de projeto permite um melhor dimensionamento da interface, reduzindo custos com área e potência, oferecendo adaptabilidade aos requisitos de projeto.

Por fim, outros aspectos do sistema poderiam ser avaliados mais profundamente. É o caso do desempenho de um multiprocessador executando uma aplicação paralela utilizando a proposta de solução de interconexão. Neste cenário, além de experimentar diferentes redes e topologias, dados do impacto da latência diretamente sobre as aplicações poderiam ser obtidos, subsidiando análises para aperfeiçoar sistemas multiprocessados baseados em NICs e NoCs. Trabalhos futuros também incluem a comparação da solução *hardware/software* proposta nesta dissertação com uma respectiva abordagem em bibliotecas de *software* que utilize o padrão MPI, comparando algumas métricas de desempenho como latência da comunicação, potência consumida pela aplicação, etc. Outro ponto a ser avaliado em novas pesquisas é a implementação de comunicação *full-duplex* para transações realizadas pela interface de rede. Isto provavelmente traz melhoria ao desempenho da comunicação, porém, a custos de área em chip e potência dissipada.

REFERÊNCIAS

ALTERA. *Nios II Processor Referecence*. [S.l.], 2011. Citado na página 80.

BENINI, L.; MICHELI, G. D. Networks on chips: A new soc paradigm. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 35, n. 1, p. 70–78, jan. 2002. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/2.976921>. Citado 2 vezes nas páginas 17 e 27.

BEREJUCK, M. D.; ZEFERINO, C. A. Adding mechanisms for qos to a network-on-chip. In: SILVA, I. S.; RIBAS, R. P.; PLETT, C. (Ed.). *SBCCI*. ACM, 2009. ISBN 978-1-60558-705-9. Disponível em: <http://dblp.uni-trier.de/db/conf/sbcci/sbcci2009.html#BerejuckZ09>. Citado na página 39.

BHOJWANI, P.; MAHAPATRA, R. Interfacing cores with on-chip packet-switched networks. In: *VLSI Design, 2003. Proceedings. 16th International Conference on*. [S.l.: s.n.], 2003. p. 382–387. ISSN 1063-9667. Citado 5 vezes nas páginas 18, 36, 37, 38 e 39.

BHOJWANI, P.; MAHAPATRA, R. Core network interface architecture and latency constrained on-chip communication. In: *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on*. [S.l.: s.n.], 2006. p. 6 pp.–363. Citado 6 vezes nas páginas 9, 18, 19, 35, 37 e 92.

CARARA, E. A. *Serviços de Comunicação diferenciados em Sistemas Multiprocessados em Chip Baseados em Redes Intra-chip*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2011. Citado 2 vezes nas páginas 18 e 19.

DALLY, W.; SEITZ, C. The torus routing chip. *Distributed Computing*, Springer-Verlag, v. 1, n. 4, p. 187–196, 1986. ISSN 0178-2770. Disponível em: <http://dx.doi.org/10.1007/BF01660031>. Citado na página 33.

FORUM, M. *MPI: A Message-Passing Interface Standard Version 3.0*. 2012. Disponível em: <http://www.mpi-forum.org> (Jun. 2013). Citado 7 vezes nas páginas 18, 19, 24, 26, 92, 93 e 94.

GARCIA, F. V. *Disseny d'un adaptador d'IPs per a noc tolerants a falles*. [S.l.], 2009. Citado 3 vezes nas páginas 18, 19 e 94.

GIL, A. *Como elaborar projetos de pesquisa*. Atlas, 2010. ISBN 9788522458233. Disponível em: <http://books.google.com.br/books?id=HSGHRAAACAAJ>. Citado na página 22.

GUERRIER, P.; GREINER, A. A generic architecture for on-chip packet-switched interconnections. In: *Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM, 2000. (DATE '00), p. 250–256. ISBN 1-58113-244-1. Disponível em: <http://doi.acm.org/10.1145/343647.343776>. Citado na página 27.

JOVEN, J. et al. xenoc - an experimental network-on-chip environment for parallel distributed computing on noc-based mpsoc architectures. In: *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*. [S.l.: s.n.], 2008. p. 141–148. ISSN 1066-6192. Citado 6 vezes nas páginas 9, 18, 35, 42, 43 e 44.

KERMANI, P.; KLEINROCK, L. Virtual cut-through: a new computer communication switching technique. *Computer Networks*, v. 3, p. 267–286, 1979. Citado na página 33.

LEE, S. E. et al. A generic network interface architecture for a networked processor array (nepa). In: *Proceedings of the 21st international conference on Architecture of computing systems*. Berlin, Heidelberg: Springer-Verlag, 2008. (ARCS'08), p. 247–260. ISBN 3-540-78152-8, 978-3-540-78152-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1787770.1787797>. Citado 7 vezes nas páginas 9, 17, 18, 35, 41, 42 e 92.

MATOS, D. da S. M. *Interfaces Parametrizáveis para Aplicações Interconectadas por uma Rede-em-Chip*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação., 2010. Citado 7 vezes nas páginas 9, 18, 19, 27, 28, 35 e 36.

MELO, D. R. de. *Interface de Comunicação Extensível para a Rede-em-chip SOCIN*. Dissertação (Mestrado) — Universidade do Vale do Itajaí- Programa de Pós-Graduação em Computação, 2012. Citado 6 vezes nas páginas 9, 18, 19, 35, 39 e 40.

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, n. 1, p. 69–93, out. 2004. ISSN 0167-9260. Disponível em: <http://dx.doi.org/10.1016/j.vlsi.2004.03.003>. Citado na página 49.

RADULESCU, A. et al. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration. In: *Proceedings of the conference on Design, automation and test in Europe - Volume 2*. Washington, DC, USA: IEEE Computer Society, 2004. (DATE '04), p. 20878–. ISBN 0-7695-2085-5. Disponível em: <http://dl.acm.org/citation.cfm?id=968879.969205>. Citado 2 vezes nas páginas 18 e 19.

SILVA, E. L. d.; MENEZES, E. M. *Metodologia da pesquisa e elaboração de dissertação*. 3. ed.. ed. Florianópolis: UFSC, 2001. 121 páginas p. Citado na página 21.

STEFAN, R.; WINDT, J. de; GOOSSENS, K. G. W. On-chip network interfaces supporting automatic burst write creation, posted writes and read prefetch. In: KURDAHI, F. J.; TAKALA, J. (Ed.). *Proceedings of the 2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS 2010), Samos, Greece, July 19-22, 2010*. [S.l.]: IEEE, 2010. p. 185–192. ISBN 978-1-4244-7937-5. Citado 2 vezes nas páginas 18 e 19.

WALKER, D. W.; DONGARRA, J. J. MPI: a standard Message Passing Interface. v. 12, n. 1, p. 56–68, jan. 1996. ISSN 0168-7875. Citado na página 43.

ZEFERINO, C. A. *Redes-em-Chip : arquiteturas e modelos para avaliação de área e desempenho*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação., 2003. Disponível em: <http://hdl.handle.net/10183/4179>. Citado 8 vezes nas páginas 17, 27, 28, 29, 30, 31, 32 e 33.

ZEFERINO, C. A.; SANTO, F. G. M. E.; SUSIN, A. A. Paris: a parameterizable interconnect switch for networks-on-chip. In: BARROS, E. N. da S. et al. (Ed.). *SBCCI*. [S.l.]: ACM, 2004. p. 204–209. Citado 7 vezes nas páginas 33, 34, 48, 49, 80, 86 e 92.

APÊNDICE A – ARQUITETURA DA INTERFACE DE REDE MPNI

A arquitetura da interface de rede MPNI está esquematizada na Figura 54. Nesta perspectiva tem-se simbolicamente a estrutura em RTL da interface de rede discutida nos capítulos desta dissertação.

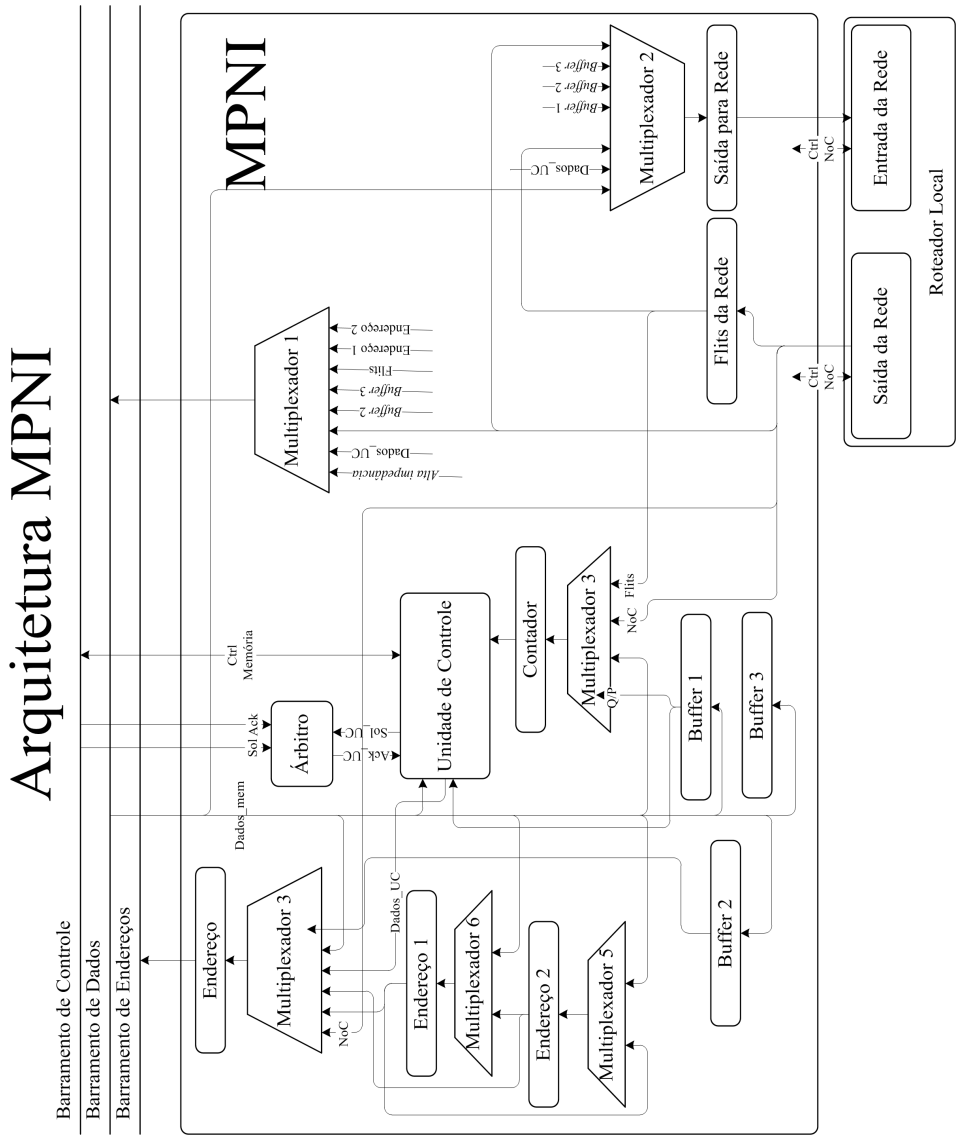


Figura 54 – Detalhes da arquitetura interna da Interface de Rede MPNI em RTL.