UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO
UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

ANDRE LUIZ VIANA PEREIRA

PROJETO E IMPLEMENTAÇÃO DE UM MPSOC UTILIZANDO A IPNOSYS COMO UNIDADE DE PROCESSAMENTO

MOSSORÓ - RN
2014

ANDRE LUIZ VIANA PEREIRA

PROJETO E IMPLEMENTAÇÃO DE UM MPSOC UTILIZANDO A IPNOSYS COMO UNIDADE DE PROCESSAMENTO

Dissertação apresentada ao Mestrado de Ciência da Computação – associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semiárido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Silvio Roberto Fernandes de Araujo – UFERSA.

MOSSORÓ-RN

2014

ANDRE LUIZ VIANA PEREIRA

PROJETO E IMPLEMENTAÇÃO DE UM MPSOC UTILIZANDO A IPNOSYS COMO UNIDADE DE PROCESSAMENTO

Dissertação apresentada ao Mestrado em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: 19 / 02 / 2014.

BANCA EXAMINADORA

Sílvio Roberto Fernandes de Araújo

Dr. Sílvio Roberto Fernandes de Araújo – UFERSA
Presidente da banca e orientador

Ivan Saraiva Silva

Dr. Ivan Saraiva Silva – UFPI
Primeiro Membro – Externo

Dra. Karla Darlene Nepomuceno Ramos – UERN
Segundo Membro – Interno

AGRADECIMENTOS

Inicialmente, a minha família, por toda dedicação e incentivo, que me encorajaram nessa caminhada.

Ao professor orientador, Sílvio Roberto Fernandes, pela ajuda, atenção e tempo disponibilizado.

Aos professores Ivan Saraiva Silva e Karla Darlene Nepomuceno pelas importantes contribuições dadas na defesa deste trabalho.

Aos demais professores do PPgCC, tanto da UFERSA quanto da UERN.

A CAPES, que viabilizou a realização deste trabalho por meio da concessão de bolsa.

E por último, mas não menos importante, a Mickaelly Moreira por todo o amor, atenção e compreensão dedicados. Bem como a sua família, por tão bem me recepcionar.

RESUMO

A IPNoSys é uma arquitetura de propósito geral não convencional recentemente proposta, cuja principal vantagem é o alto desempenho computacional diante de aplicações paralelas. Mesmo assim, sua arquitetura permite uma quantidade limitada de quatro fluxos simultâneos de execução, o que impede um desempenho ainda maior. Este trabalho propõe uma nova arquitetura MPSoC baseada na IPNoSys, no qual o principal ganho em desempenho é proporcionado pela possibilidade de mais fluxos de execução paralelos. Tal MPSoC é composto por múltiplas IPNoSys interligadas por meio de uma rede em chip em topologia grelha. Para que fosse possível utilizar a IPNoSys como elemento de processamento, foram necessárias alterações em sua arquitetura, linguagem de programação e montador. A avaliação do MPSoC foi feita a partir dos resultados produzidos por seu simulador, desenvolvido em SystemC, executando uma série de experimentos com características distintas. Esses resultados mostraram que o desempenho da arquitetura proposta é aproximadamente três vezes superior ao da IPNoSys Original, ao custo de um pequeno consumo extra de energia.

Palavras-Chave: MPSoC, IPNoSys, Rede em Chip, Hierarquia de Redes em Chip.

ABSTRACT

The IPNoSys is a general purpose unconventional architecture recently proposed which has as main advantage its high computational performance facing parallel applications. Even so, its architecture allows a limited amount of four simultaneous execution flow which hinder even greater performance. This paper propose a new IPNoSys-based MPSoC architecture in which the major gain in performance is from the possibility of have more than four parallel execution flow. Such MPSoC is composed by multiple IPNoSys interconnected through a network on chip in mesh topology. The IPNoSys needed some modification in its architecture, program language and assembler, to be used as processing element. The system evaluation was done from the results obtained by its simulator, developed in SystemC, executing a serie of experiments with distinct characteristics. This results shown that the new architecture performance is nearly three times greater than Original IPnosys performance, with a little extra energy consumption.

Keywords: MPSoC, IPNoSys, Networ-on-chip, Hierarquical NoC.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

ALU	*Arithmetic Logic Unit*
ASIC	*Application Specific Integrated Circuit*
CC-NUMA	*cache-coherent non-uniform memory access*
CHNoC	*Cluster-based Hierarchical* NoC
CN	*Cluster Node*
CPU	*Central Processing Unit*
CR	*Core Router*
DCT	*Discrete Cosine Transform*
DSP	*Digital Signal Processor*
E/S	Entrada e saída
EP	Elemento de Processamento
ER	*Edge Router*
FCFS	*First-Come-First-Served*
FIFO	*First In First Out*
HCR-NoC	*Hybrid Communication Reconfigurable Network on Chip*
IOMAU	*I/O Memory Access Unit*
IPNoSys	*Integrated Processing* NoC *System*
JPEG	*Joint Photographic Experts Group*
LRS	*Least-Recently-Served*
MAU	*Memory Access Unit*
MIMD	*Multiple Instruction Multiple Data*
MISD	*Multiple Instruction Single Data*
MPSoC	*Multiprocessors* SoC
NI	*Network Interface*
NoCs	*Networks-on-Chip*
NORMA	*non-remote memory access*
NUMA	*non-uniform memory access*
PDL	*Package Description Language*
RAM	*Random Memory Access*
RLE	*Run-Length Encoding*

RPU	*Routing and Processing Unit*
SB	*Switch Box*
SIMD	*Single Instruction Multiple Data*
SISD	*Single Instruction Single Data*
SoC	*System-on-chip*
SU	*Synchronization Unit*
TDMA	*Time Division Multiple Access*
UMA	*uniform memory access*
VHDL	VHSIC *Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*

SUMÁRIO

1 INTRODUÇÃO

Nos anos de 50 e 60, a tecnologia de fabricação de máquinas monoprocessadas era suficiente para atender à demanda de processamento da época. A partir dos anos 70, a evolução dos monoprocessadores não foi capaz de acompanhar a necessidade crescente de capacidade de processamento. Isso levou à utilização de técnicas de concorrência, a fim de alcançar o processamento requerido (ROSE; NAVAUX, 2008). Uma das primeiras técnicas foi o uso de unidades específicas para tratar as operações de entrada e saída, enquanto a execução de instruções era realizada por outra unidade dedicada (FERNANDES, 2012). Posteriormente, surgiu o entrelaçamento de memória, que permitia o acesso simultâneo à memória pela divisão desta em bancos, a técnica de *pipeline*, possibilitando a execução concorrente de estágios distintos do ciclo de execução das instruções (ROSE; NAVAUX, 2008), e os processadores superescalares, que possuíam várias unidades de execução, capazes de processar múltiplas instruções de um mesmo programa em paralelo (STALLINGS, 2006).

À medida que a tecnologia evoluiu e o custo do hardware do computador tornou-se mais baixo, foi possível a integração de vários componentes, como processador, memória e controladores, em um único *chip*. Esses sistemas completos em uma única pastilha são denominados *System-on-chip* (SoCs) (ZEFERINO, 2003b). A colocação de mais de um processador por *chip* interligados por um subsistema de comunicação definiu um tipo especial de SoC, chamado de *Multiprocessors* SoC (MPSoC) (WOLF; JERRAYA, 2005). Estes permitiram a exploração do multiprocessamento, ou seja, a realização de múltiplas tarefas em paralelo. Uma das grandes vantagens de se utilizar MPSoCs é a possibilidade de dividir a carga de trabalho entre seus elementos de processamento, permitindo grandes ganhos de desempenho computacional e energético.

Os MPSoCs vêm sendo considerados como um bom método para o aumento do desempenho de processadores (ALVES *et al*., 2007), sendo este determinado tanto pela capacidade computacional dos núcleos de processamento quanto pelo subsistema de comunicação (FERNANDES, 2012). Existem diferentes soluções para a interconexão e comunicação em MPSoCs, entre elas: os barramentos, canais ponto-a-ponto e as Redes em Chip ou *Networks-on-Chip* (NoCs). Os barramentos fornecem um canal multiponto compartilhado entre todos os processadores. Essa solução possui um baixo custo e grande reusabilidade, porém, o aumento do número de núcleos conectados causa redução do desempenho do sistema. Os canais ponto-a-ponto proporcionam melhor desempenho e menor

latência, pois a comunicação é feita diretamente entre dois núcleos e independentemente dos demais. Nesse tipo de arquitetura, o aumento do número de núcleos está diretamente ligado ao aumento da complexidade e custo do projeto (MARTINI *et al*., 2009), a ponto de inviabilizar sua utilização. As NoCs, que são estruturas baseadas nos conceitos de redes de computadores, apresentam alta escalabilidade, comunicação paralela, estrutura reutilizável e interconexão ponto-a-ponto entre elementos adjacentes. Sua utilização, porém, pode resultar no aumento da área do chip e da potência dissipada. Mesmo assim, as NoCs têm sido consideradas a solução ideal para a interconexão e comunicação em MPSoC.

Baseado nos conceitos de NoCs, Fernandes (2012) propõem um novo modelo de arquitetura paralela denominado *Integrated Processing* NoC *System* (IPNoSys). Tal arquitetura utiliza uma NoC onde seus roteadores possuem, além da capacidade fundamental de roteamento, a capacidade de executar as instruções que formam as aplicações. Assim, a rede da IPNoSys deixa de ser apenas um subsistema de comunicação e passa a ser a responsável pelo processamento dos pacotes que nela circulam (FERNANDES *et al*., 2008). Resultados publicados em (FERNANDES *et al*., 2009a) demonstram que a IPNoSys possui grande poder computacional quando comparada à arquiteturas semelhantes, principalmente, no processando de aplicações com alta capacidade de paralelização.

No entanto, a IPNoSys está limitada a paralelização de aplicações em quatro de fluxos simultâneos de execução. Essa limitação é imposta por sua própria arquitetura: cada uma das quatro unidades de acesso à memória é capaz de produzir um fluxo de execução por vez. Isso impede uma maior exploração do paralelismo das aplicações e, consequentemente, um melhor desempenho da arquitetura.

Diante desse contexto, este trabalho visa desenvolver um sistema de maior poder computacional baseado na IPNoSys. O principal ganho em desempenho deste sistema é devido à possibilidade de haver mais que quatro fluxos de execução sendo processados simultaneamente.

1.1 OBJETIVOS

O objetivo geral deste trabalho é propor uma nova arquitetura MPSoC homogênea baseada na IPNoSys que permita maior exploração do paralelismo das aplicações. Nesse MPSoC, múltiplas IPNoSys serão utilizadas como Elementos de Processamento (EPs),

estando interligadas entre si por meio de uma rede em chip. Essa organização criará uma hierarquia de redes constituída por uma rede global, responsável pela comunicação entre EPs, que são, por sua vez, redes locais capazes de processar as aplicações. Cada EP segue o modelo arquitetural da IPNoSys, que funcionará como um processador independente capaz de executar até quatro fluxos de execução em paralelo. A quantidade de fluxos total do sistema será proporcional à quantidade de EPs utilizados. A interconexão por NoC permitirá que a arquitetura apresente as vantagens relacionadas a este conceito, como alta escalabilidade, flexibilidade e reaproveitamento de componentes.

Os objetivos específicos deste trabalho estão pontuados a seguir:

- Desenvolver uma estratégia de comunicação entre várias arquiteturas IPNoSys;
- Desenvolver uma interface de rede para troca de dados entre IPNoSys e NoC;
- Implementar a NoC da arquitetura;
- Adaptar o formato do pacote para o novo sistema;
- Adaptar a linguagem de programação da arquitetura original;
- Adaptar o montador da arquitetura à nova linguagem e ao novo formato dos pacotes;
- Desenvolver aplicações de teste com diferentes características, que permitam comparar o MPSoC a arquitetura original.

1.2 ORGANIZAÇÃO DA DISSERTAÇÃO

O trabalho está estruturado da seguinte forma:

- O capítulo 2 mostra alguns conceitos de arquiteturas paralelas, como classificações e redes de interconexão, uma rápida abordagem sobre MPSoC e alguns trabalhos relacionados encontrados na literatura.
- No capítulo 3, os conceitos sobre a arquitetura IPNoSys relevantes para este trabalho são apresentados.
- O capítulo 4 descreve a arquitetura MPSoC proposta neste trabalho.
- No capítulo 5, os experimentos e seus respectivos resultados são apresentados.
- O capítulo 6, por fim, apresenta as considerações finais e trabalhos futuros.

2 ARQUITETURAS COM MÚLTIPLOS PROCESSADORES

Uma maneira tradicional de aumentar o desempenho de sistemas computacionais é utilizar vários processadores capazes de processar em paralelo. As arquiteturas multiprocessadas vêm sendo consolidadas como um bom método para aumentar o desempenho de computação e, em alguns casos, diminuir a potência dissipada (ALVES *et al*., 2007).

As próximas subseções apresentam conceitos relacionados às arquiteturas paralelas que serviram de embasamento teórico para o desenvolvimento deste trabalho. Primeiramente, nas seções 2.1 e 2.2, são apresentados alguns critérios de classificação de máquinas paralelas, como as de Flynn e de compartilhamento de memória. Na seção 2.3, são mostrados alguns tipos de redes de interconexão. Depois disso, algumas características sobre MPSoCs são abordadas na seção 2.4. E, por fim, trabalhos relacionados são descritos na seção 2.5.

2.1 CLASSIFICAÇÃO DE FLYNN

A classificação de Flynn é a forma mais comum de classificar máquinas paralelas (STALLINGS, 2006). Ela considera que, apesar de sistemas computacionais executarem uma sequência de instruções sobre uma sequência de dados, a multiplicidade dos fluxos, tanto de instruções quanto de dados, pode ser diferenciada (ROSE; NAVAUX, 2008). Dependendo da combinação dessa multiplicidade, os sistemas de computação podem ser divididos em quatro classes: SISD (*Single Instruction Single Data*), MISD (*Multiple Instruction Single Data*), SIMD (*Single Instruction Multiple Data*) e MIMD (*Multiple Instruction Multiple Data*). A Figura 1 apresenta a representação de cada classe.

As máquinas SISD executam uma única sequência de instruções sobre uma única sequência de dados. Essas máquinas utilizam apenas um processador com acesso a somente uma memória. Nessa classe, são enquadradas as máquinas de Von Neumann tradicionais com apenas um processador.

Em máquinas MISD, vários fluxos de instruções atuam sobre um único fluxo de dados. Na prática, diferentes instruções operariam a mesma posição de memória ao mesmo

tempo (ROSE; NAVAUX, 2008). Como isso é tecnicamente impraticável, essa classe não possui nenhuma implementação.

Na classe SIMD, as máquinas executam uma instrução sobre um conjunto de dados de forma simultânea. Nesse caso, uma única unidade de controle alimenta vários processadores com uma mesma sequência de instruções. Cada processador recebe instruções semelhantes e a executa sobre um conjunto de dados distintos. Um exemplo de máquina SIMD são os computadores vetoriais.

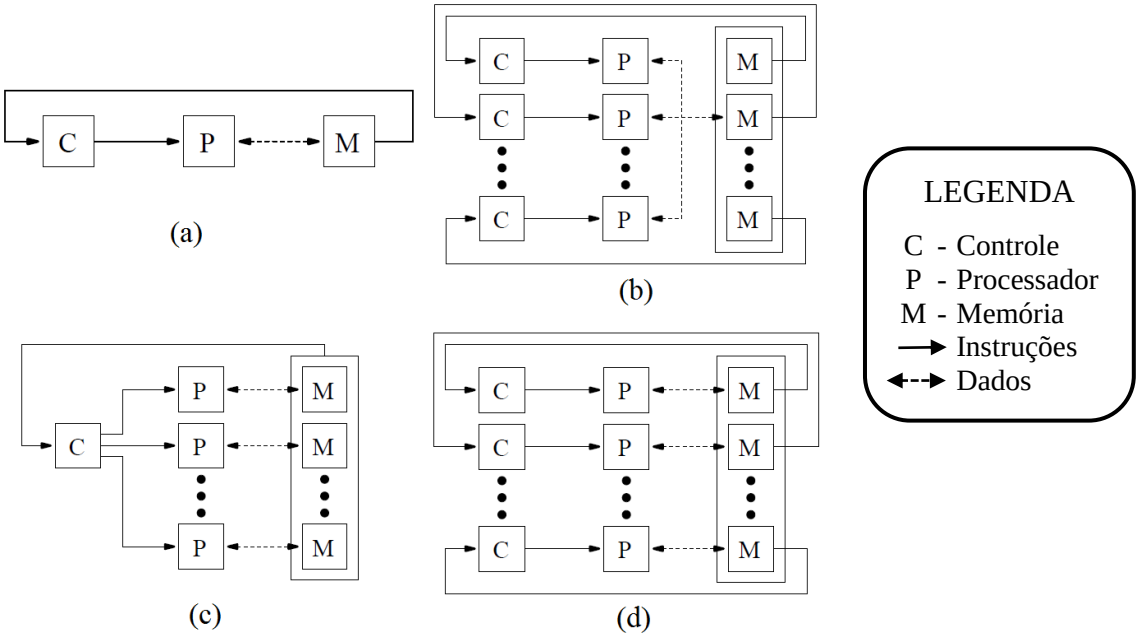


Figura 1: Classificação de Flynn: (a) SISD; (b) MISD; (c) SIMD; (d) MIMD. Fonte: (ROSE; NAVAUX, 2008).

Por fim, na classe MIMD vários fluxos de instruções são executados sobre vários fluxos de dados distintos. As máquinas dessa classe possuem um conjunto de processadores executando simultaneamente uma sequência diferente de instruções, sobre um conjunto de dados distintos (STALLINGS, 2006). Exemplos dessa classe são máquinas com múltiplos processadores ou com processadores multinúcleo, comuns atualmente em servidores e computadores pessoais, respectivamente.

As arquiteturas paralelas se concentram nessas duas últimas classes, sendo a última mais comum, pois qualquer grupo de máquinas trabalhando em conjunto pode ser considerado uma máquina MIMD (ROSE; NAVAUX, 2008). A arquitetura IPNoSys, utilizada no presente trabalho, se enquadra nesta classificação.

2.2 CLASSIFICAÇÃO SEGUNDO O COMPARTILHAMENTO DE MEMÓRIA

As máquinas MIMD podem ainda ser subdivididas em duas classes, pelo critério de compartilhamento de memória: multiprocessador, caso a memória seja compartilhada entre todos os processadores da máquina, ou multicomputador, caso a memória seja privada.

Segundo Rose e Navaux (2008), os multiprocessadores são caracterizados pela replicação do componente processador de uma arquitetura convencional, onde todos eles podem acessar uma memória compartilhada através de uma rede de interconexão. Nesse tipo de máquina, a comunicação entre processos é feita através da própria memória, com operações do tipo *load* e *store*. A memória possui um espaço de endereçamento único e, dependendo do tipo de seu acesso, os multiprocessadores podem ser classificados como (HESS, 2003):

- Acesso uniforme à memória (*uniform memory access* – UMA): quando utilizam memória centralizada. Isto é, quando a memória é formada por módulos igualmente distantes de todos os processadores do sistema, fazendo com que a latência de acesso seja a mesma para qualquer um. A Figura 2(a) apresenta esse sistema.
- Acesso não uniforme à memória (*non-uniform memory access* – NUMA): quando utiliza memória distribuída. Nesse caso, a memória é dividida em módulos associados a cada processador. Apesar de estarem conectados diretamente a processadores específicos, os módulos compõem um espaço de endereçamento único e podem ser acessado por todos. Porém, a latência para acessar a memória local é menor que a latência para acessar uma memória remota, sendo, por isso, chamado de acesso não uniforme. Esse sistema é mostrado na Figura 2(b). Existem ainda outras abordagens para os sistemas NUMA, que fazem uso de uma memória especial, mais rápida que a memória principal, para alcançar um maior desempenho, chamada memória *cache*. Um exemplo desses sistemas é o de acesso não uniforme à memória com coerência de *cache* (*cache-coherent non-uniform memory access* – CC-NUMA). Essas outras abordagens não serão apresentadas neste trabalho.

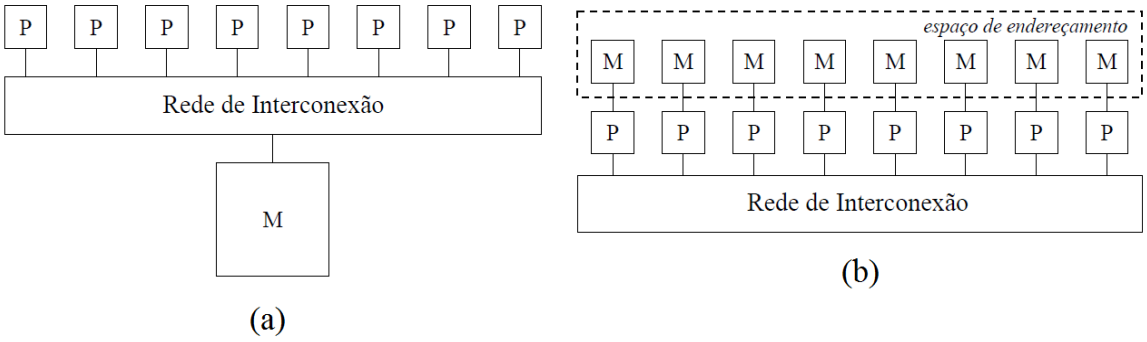


Figura 2: Multiprocessadores: (a) UMA; (b) NUMA. Fonte: (ROSE; NAVAUX, 2008).

A segunda classificação possível para as máquinas MIMD em relação ao compartilhamento de memória é chamada de multicomputador. Nessa classificação, as máquinas são construídas a partir da replicação de toda a arquitetura convencional, e não somente do processador. Dessa forma, cada processador possui sua própria memória local, que pode ser acessada exclusivamente por ele. Diferentemente da anterior, cada módulo de memória é um espaço de endereçamento distinto, onde não é possível o uso de variáveis compartilhadas. A troca de informações só pode ser efetuada por meio de mensagens enviadas através da rede de interconexão (HESS, 2003). Portanto, em relação ao tipo de acesso, os multicomputadores são classificados como sem acesso a variáveis remotas (*non-remote memory access* – NORMA).

2.3 REDE DE INTERCONEXÃO

A rede de interconexão é o meio pelo qual os processadores são ligados entre si e a outros componentes do sistema. A utilização de uma rede adequada permite que, nos multiprocessadores, os problemas de conflitos pelo acesso ao meio sejam amenizados e, nos multicomputadores, a troca de mensagens ocorra de forma mais eficiente (ROSE; NAVAUX, 2008). A avaliação de quanto uma rede é eficiente para um dado sistema é determinada pela avaliação de vários critérios, como escalabilidade, desempenho, custo e confiabilidade. Existem, basicamente, dois tipos de rede de interconexão: as redes estáticas e as redes dinâmicas.

As redes estáticas são aquelas que possuem suas ligações fixas, ou seja, não podem alterar sua topologia redirecionando suas conexões (MONTEZ, 1995). Esse tipo de rede

também é chamado de rede ponto-a-ponto, por existir, entre dois componentes, uma ligação direta e dedicada. A Figura 3 apresenta algumas organizações estruturais clássicas para redes estáticas, chamadas de topologias.

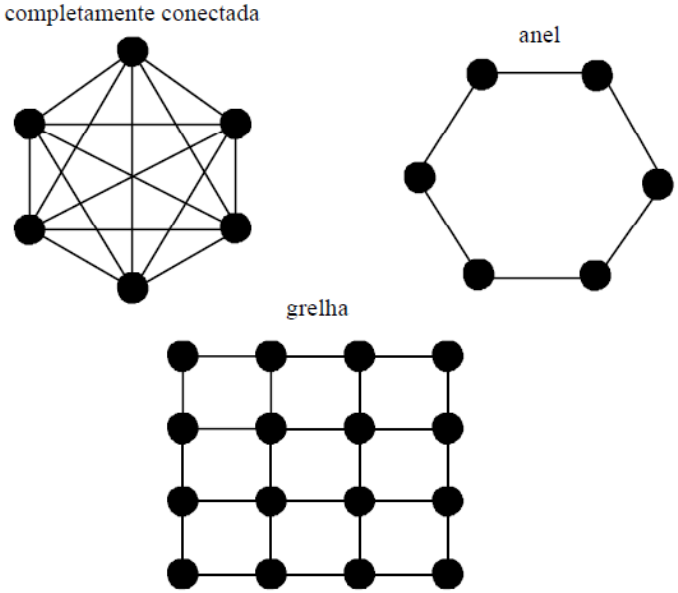


Figura 3: Algumas topologias estáticas. Fonte: (MONTEZ, 1995) modificada pelo autor.

A topologia da rede pode ser ajustada de modo que os componentes (nós) da rede sejam mais ou menos conectados, como podem ser vistas, na Figura 3, a rede completamente conectada e a anel, respectivamente. A topologia considerada ideal seria a completamente conectada, pois as mensagens trocadas entre quaisquer nós são feitas de forma direta (MONTEZ, 1995). Entretanto, o aumento na quantidade de ligações eleva, consideravelmente, o custo de implementação da rede, o que a torna inviável. Como solução, utiliza-se topologias parcialmente conectadas, ou seja, nem todos os nós estão conectados diretamente entre si. Neste caso, as mensagens devem ser retransmitidas por cada nó em direção ao seu destino, até que ele seja alcançado.

As redes de interconexão dinâmicas são aquelas que não possuem uma topologia fixa, não havendo um padrão de comunicação. Quando uma conexão entre dois pontos é necessária, esse tipo de rede adapta-se dinamicamente para permitir a transferência dos dados. As redes dinâmicas podem ser classificadas de três formas:

- bloqueantes, quando o estabelecimento de uma comunicação entre dois pontos impede que os demais se comuniquem.
- unilateral, quando a rede permite mais de uma transmissão por vez, mas de modo que um nó envolvido em uma comunicação esteja ou enviando ou recebendo dados.

- bilateral, quando a rede permite mais de uma transmissão por vez e os nós podem enviar e receber ao mesmo tempo.

Dentre as redes dinâmicas, uma alternativa de baixo custo bastante utilizada é o barramento (TANENBAUM, 2006). Um barramento é um caminho de comunicação compartilhado entre dois ou mais dispositivos. Tipicamente, ele é composto por vários caminhos, cada um capaz de transmitir sinais que representam um único dígito binário. Os dados transmitidos por um dispositivo através do barramento podem ser recebidos, ao mesmo tempo, por todos os outros que estejam conectados (STALLINGS, 2006). Por outro lado, se mais que um dispositivo transmitir ao mesmo tempo, os sinais colidirão e serão adulterados. Portanto, os barramentos se encaixam na classe de redes dinâmicas bloqueantes, sendo necessário um mecanismo que garanta o envio de sinais por apenas um dispositivo a cada instante. O componente responsável por essa coordenação é chamado de árbitro.

Sempre que um componente deseja iniciar uma transferência de dados através do barramento, ele primeiro deve fazer uma requisição ao árbitro. O árbitro segue um algoritmo simples de arbitragem para determinar quando o dispositivo requisitante pode utilizar o meio de transmissão. Existem vários algoritmos de arbitragem, usando abordagens de ceder o acesso aos dispositivos seguindo a ordem de chegada das requisições, de utilizar o compartilhamento circular de tempo (*round-robin*) ou de usar algum tipo de esquema de prioridade. É importante que os árbitros não realizem algoritmos complexos, ao ponto de necessitarem de vários ciclos para a tomada de decisão. Isso porque esse algoritmo se repete a cada requisição, podendo comprometer o desempenho do sistema.

O barramento é a arquitetura de comunicação tipicamente utilizada, pois possui como vantagem características como reusabilidade e baixo custo (ZEFERINO, 2003b). Porém, por ser um canal compartilhado entre vários componentes do sistema, o aumento da quantidade de dispositivos capazes de iniciar uma transmissão compromete seu desempenho. Para contornar esse problema, a maioria dos sistemas de computação utiliza hierarquia de múltiplos barramentos, isto é, dois ou mais barramento são colocados interconectando os dispositivos. Os barramentos da hierarquia se comunicam através de pontes entre si.

Um tipo mais complexo de rede de interconexão, denominado Rede em Chip, é apresentado na subseção a seguir. Seus conceitos são mais detalhados por ser a base da arquitetura IPNoSys e também o sistema de comunicação utilizado na arquitetura proposta neste trabalho, permitindo construir uma hierarquia de redes.

2.3.1 Redes em chip

Uma Rede em Chip ou *Network-on-Chip* (NoC) é uma plataforma baseada nos conceitos originados da área de redes de computadores e comunicação de dados (WOSZEZENKI, 2007). Elas são utilizadas para prover a comunicação entre elementos, ou núcleos, conectados à rede, que podem ser: processadores, memórias, dispositivos de E/S, componentes específicos de *hardware* ou até computadores completos (FERREIRA, 2009). Tipicamente, as NoCs são compostas por roteadores interligados entre si e aos núcleos de um sistema integrado por meio de enlaces ponto a ponto (ZEFERINO, 2003b), como exemplificada na Figura 4. A comunicação é feita, na maioria das vezes, através do envio e recebimento de mensagens organizadas dentro de pacotes.

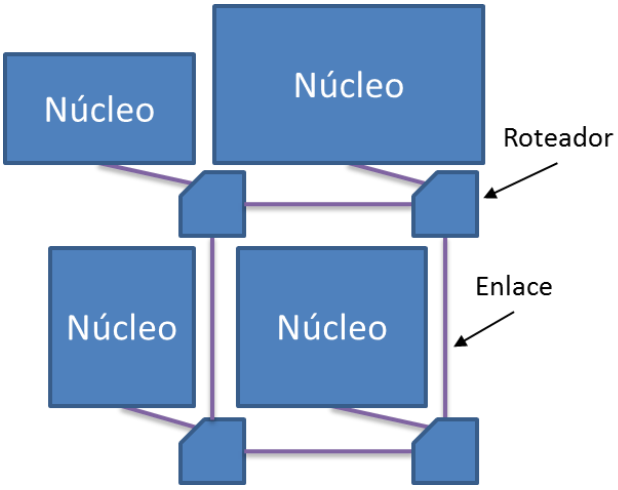


Figura 4: Exemplo de Rede em Chip. Fonte: (ZEFERINO; SUZIN, 2003) modificada pelo autor.

Segundo Zeferino (2003b), os roteadores são os componentes responsáveis por determinar o caminho pelo qual um pacote deve seguir pela rede para que chegue ao seu destino. Eles são constituídos por um conjunto de portas de entrada e de saída, unidades de chaveamento e uma lógica para controle de roteamento e arbitragem.

As portas de entrada e de saída são os canais pelos quais os roteadores se comunicam com outros roteadores ou com os núcleos do sistema. Associado a cada porta, pode haver uma pequena quantidade de memória, denominada *buffer*, utilizada para armazenar dados temporariamente. Além de transmitir os dados contidos nos pacotes, as portas são também utilizadas para transmitir informações de controle, a fim de regular o tráfego na rede.

As unidades de chaveamento são os elementos responsáveis por estabelecer a ligação interna entre uma porta de entrada e uma de saída do roteador, ou seja, elas permitem que os pacotes recebidos possam ser reenviados por qualquer porta de saída. A Figura 5 apresenta uma possível implementação de um roteador semelhante aos utilizados na Figura 4, destacando seus *buffers* e chaves. Nela, pode-se perceber a direção que os pacotes que chegam ao *buffer* de cada porta de entrada podem tomar. Em outras implementações, as chaves podem aparecer integradas em um único módulo, também chamado de *crossbar* (ZEFERINO, 2003b).

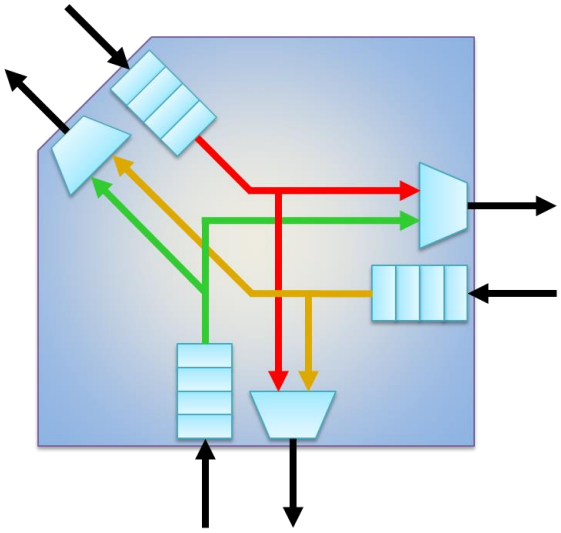


Figura 5: Visão interna do roteador. Fonte: (ZEFERINO, 2003a) modificada pelo autor.

O roteamento define o caminho a ser seguido por um pacote entre sua origem e destino (ZEFERINO, 2003a). Internamente, a lógica de roteamento é responsável por determinar por qual porta de saída um pacote recebido deve seguir para atingir o seu destino. A escolha da porta pode levar em consideração vários fatores, como topologia da rede, tráfego e custo do enlace. A lógica de arbitragem é utilizada para resolver conflitos internos, quando pacotes disputam pelo mesmo recurso (*buffers* ou canais de saída). Portanto, um árbitro tem função de controlar o uso dos recursos, estabelecendo quem pode utilizá-los em dado momento. O roteamento e a arbitragem ainda serão discutidos mais adiante.

Os enlaces são os canais físicos que fazem a ligação direta entre roteadores ou roteadores e núcleos da rede, podendo ser vistos em lilás na Figura 4. Cada canal é composto por um conjunto de fios, que transmitem informações de controle e dados dos pacotes na forma de palavras. De acordo com Zeferino (2003b), os enlaces das NoCs, geralmente, são *full-duplex*, ou seja, possuem um par de canais unidirecionais opostos capazes de transmitir sinais simultaneamente em ambas as direções.

As mensagens são as informações trocadas durante a comunicação entre núcleos conectados à rede. Geralmente, elas são transmitidas sob a forma de um ou mais pacotes. Um pacote, por sua vez, é a menor unidade de informação que inclui detalhes sobre o roteamento e sequenciamento de uma mensagem. Ele é constituído por uma sequência de palavras com largura igual à largura do canal físico de transmissão de dados (ZEFERINO, 2003b). Cada pacote possui sua estrutura iniciada por um cabeçalho, seguido pela carga útil e o terminador. O cabeçalho pode ser formado por uma ou mais palavras e contém informações de roteamento e controle, utilizadas pelos roteadores para transportar o pacote da origem até o destino. A carga útil é a seção onde estão os dados propriamente ditos da mensagem e, por fim, o terminador é uma palavra predefinida usada para indicar o fim do pacote.

Uma NoC é caracterizada com base em alguns critérios, que são: topologia, roteamento, chaveamento, controle de fluxo, arbitragem e memorização. As próximas subseções, com exceção da última, apresentam uma breve visão sobre cada uma dessas características. A última subseção, 2.3.1.7, destaca as principais vantagens e desvantagens do uso de NoCs.

2.3.1.1 Topologia

A comunicação em Redes em Chip acontece de forma indireta, isto é, para trocar mensagens com outros núcleos da rede, é necessário que elas passem através de elementos intermediários, que são os roteadores. Cada roteador está conectado fisicamente a outros, sendo a estrutura formada pelas ligações entre eles a topologia da rede (REINBRECHT, 2012).

Conforme já mencionado na seção 2.3, quanto maior o número de enlaces entre os componentes da rede, maiores são seu custo e complexidade. Os tipos mais tradicionais de topologia para NoCs são apresentados na Figura 6. Segundo (REINBRECHT, 2012), a grelha (também conhecida por *mesh*) é a topologia mais comumente utilizada. Nela, os roteadores podem possuir até quatro vizinhos (norte, sul, leste e oeste), como pode ser visto na Figura 6 (a). A topologia toróide (chamada de *torus*) é semelhante à grelha, tendo como diferença a existência de ligações entre os elementos de bordas opostas. Desse modo, todos os roteadores estão conectados a outros quatro, o que torna o sistema um pouco mais complexo, mas permite caminhos menores entre origem e destino. A topologia anel (ou *ring*) possui

roteadores dispostos sequencialmente. Essa estrutura permite uma rede com roteadores simples, de apenas dois canais de comunicação, mas com baixa escalabilidade, pois o desempenho diminui conforme o número de elementos cresce. Por fim, a árvore (ou *tree*) possui topologia semelhante à mostrada na Figura 6 (d). Nesta, a largura dos canais diminui à medida que se aproxima das folhas, que é onde há maior tráfego na rede (REINBRECHT, 2012).

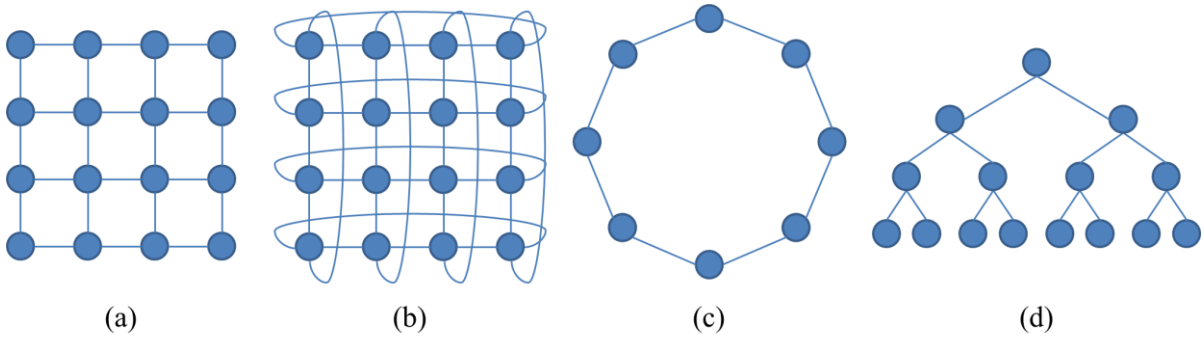


Figura 6: Exemplos de topologias: (a) Grelha, (b) Toróide, (c) Anel e (d) Árvore. Fonte: (REINBRECHT, 2012).

2.3.1.2 Roteamento

Analisando a Figura 6, é possível perceber que pode haver um ou mais caminhos a serem seguidos por um pacote entre dois roteadores. O método utilizado para definir esses caminhos é chamado de roteamento (REGO, 2006). Ao receber um pacote, o roteador escolhe, segundo um algoritmo de roteamento, a porta de saída pela qual o pacote deve ser encaminhado. Ele pode, então, ser entregue ao núcleo conectado ao roteador, quando chega ao seu destino, ou a outro roteador, reiniciando esse processo. A informação básica utilizada no roteamento é o endereço de destino do pacote, sendo fundamental que cada roteador possua um endereço único na rede.

Os algoritmos de roteamento são classificados de acordo com diversos critérios, sendo alguns deles: momento de realização do roteamento, local onde as decisões de roteamento são tomadas, adaptatividade e número de destinatários (ZEFERINO, 2003b).

Em relação ao primeiro critério citado, ele pode ser dinâmico, quando o roteamento é realizado no tempo de execução, ou estático, quando é realizado no tempo de compilação.

Os locais de tomada de decisão de roteamento podem ser de três tipos: centralizado, quando o caminho é definido por um controlador central na rede, fonte, quando o caminho é estabelecido pelo emissor do pacote antes de injetá-lo, e distribuído, no qual o caminho é definido pelos próprios roteadores enquanto o pacote atravessa a rede.

O critério de adaptatividade se refere à capacidade do algoritmo fornecer diferentes caminhos entre os mesmos elementos. Um algoritmo adaptativo leva em consideração informações como tráfego da rede ou estado dos canais para definir a rota do pacote. Já os algoritmos estáticos fornecem sempre o mesmo caminho entre origem e destino para o transporte dos pacotes.

Por fim, em relação ao número de destinatários do pacote, o algoritmo pode ser *unicast*, quando possui somente um destino, ou *multicast*, quando pode ser encaminhado para múltiplos destinatários.

Existem vários algoritmos de roteamento propostos (FERNANDES, 2012), com características ideais para diferentes propósitos. Um algoritmo bastante utilizado em topologias dimensionais é o roteamento XY (REGO, 2006). Nesse algoritmo, o pacote é transmitido em uma dimensão por vez. No caso da topologia grelha, o pacote percorre primeiramente o eixo X até a coordenada do destino e, então, o eixo Y, alcançando seu destino. As principais vantagens desse algoritmo são simplicidade e liberdade de um problema denominado *deadlock*. Esse problema ocorre quando há uma dependência cíclica entre pacotes, impossibilitando o funcionamento normal do sistema. Por exemplo, quando um pacote depende de um segundo para ser transmitido e, ao mesmo tempo, este segundo depende do primeiro. A grande desvantagem desse algoritmo é a redução do número de rotas possíveis, por seguir sempre o percurso de XY, o que reduz a taxa de utilização da rede e da largura de banda disponível (ZEFERINO, 2003b).

2.3.1.3 Chaveamento

Nas NoCs, as mensagens são transportadas pela rede através de seus roteadores. Ao chegar por uma porta de entrada de um roteador, elas são encaminhadas por uma de suas portas de saída, sendo levadas ao núcleo associado ao roteador ou a outro roteador adjacente. A forma como é feita a transferência dos dados de um canal de entrada para um canal de saída é definida pelo chaveamento (ZEFERINO, 2003b). Existem dois tipos principais de técnicas

de chaveamento: as baseadas no estabelecimento de um caminho completo entre a origem e o destino da mensagem ou na divisão das mensagens em pacotes que alocam seu caminho à medida que trafegam pela rede.

Na primeira técnica, também chamada de chaveamento por circuito, a comunicação só ocorre após o estabelecimento de um caminho completo entre os elementos de origem e destino. Inicialmente, um cabeçalho com o endereço do destinatário e informações de controle é injetado na rede pelo elemento que deseja iniciar a comunicação. Esse cabeçalho reserva os canais de comunicação à medida que avança através dos roteadores em direção ao seu destino. Caso algum canal necessário já esteja alocado, o cabeçalho aguarda até que ele se torne disponível. Uma vez que tenha alcançado o destino, uma mensagem de reconhecimento é enviada ao elemento de origem e a comunicação é efetivamente iniciada. Após o envio de toda a carga útil da mensagem, o terminador é injetado, sinalizando o fim da mensagem e a liberação do circuito preestabelecido. O chaveamento por circuito deve ser usado somente em sistemas nos quais as mensagens são longas e pouco frequentes (ZEFERINO, 2003b). Isso porque, caso as mensagens sejam curtas, o tempo gasto para estabelecer o caminho pode até superar o tempo de transferência dos dados. Além disso, o caminho estabelecido impede que determinados canais sejam utilizados por outros pacotes, o que seria um problema em redes com alta frequência de troca de mensagens.

Em sistemas onde há mensagens curtas e frequentes, a técnica adequada é a de divisão das mensagens em pacotes. Existem três métodos mais comuns: *store-and-forward*, *virtual cut-through* e *wormhole*.

No *store-and-forward* (SAF), cada pacote possui um tamanho fixo e carrega todas as informações necessárias para seu roteamento (REGO, 2006). Durante o tráfego pela rede, os pacotes são armazenados completamente em cada roteador que passam, sendo encaminhados somente após isso. Essa estratégia necessita de *buffers* com tamanhos suficientes para armazenar, no mínimo, um pacote. O fato de armazenar o pacote completamente em cada roteador prejudica o desempenho da rede.

A segunda técnica, *virtual cut-through* (VCT), foi proposta como uma alternativa ao *store-and-forward* (ZEFERINO, 2003b). A diferença básica é que, na VCT, existe a possibilidade de um pacote ser encaminhado diretamente para uma porta de saída, sem a necessidade de seu armazenamento completo no *buffer*. Para que isso aconteça, é preciso apenas que o canal de saída pretendido e o roteador seguinte estejam disponíveis (FERNANDES, 2012). Caso não estejam, o VCT se comporta de forma semelhante ao SAF.

Por fim, o *wormhole* foi proposto com o objetivo principal de reduzir o tamanho dos *buffers* utilizados no VCT (ZEFERINO, 2003b). Nessa técnica, os pacotes são divididos em unidades menores, chamadas de *flits*. Os *flits* são transmitidos em *pipeline* à medida que os *buffers* de entrada dos roteadores seguintes vão se tornando disponíveis. Geralmente, o primeiro *flit* possui as informações necessárias para o roteamento do pacote, sendo seguido pelos demais através dos mesmos canais. Esta característica é um ponto negativo dessa técnica, pois os canais ficam bloqueados para um segundo pacote até que todos os *flits* do primeiro tenham sido transmitidos.

2.3.1.4 Controle de fluxo

Os recursos existente em uma NoC, como *buffer* e canais de transmissão, são frequentemente disputados pelos pacotes que nela trafegam. Quando esses recursos não estão disponíveis para um pacote, por estar reservado a outro, por exemplo, as seguintes situações podem ocorrer: descarte do pacote, bloqueio no local onde se encontra, recebimento e armazenamento temporário ou desvio para outro caminho (ZEFERINO, 2003b).

Geralmente, associado a cada entrada dos roteadores existe um *buffer* para armazenamento temporário de pacotes. Porém, há casos em que os pacotes são maiores que o espaço disponível no *buffer* do receptor, sendo necessária a interrupção do fluxo de transmissão pelo emissor. Nesses casos, o pacote permanece bloqueando até que haja disponibilidade de espaço no receptor. O controle de fluxo é a estratégia de controle utilizada para sincronizar essa troca de dados entre transmissor e receptor. Segundo Reinbrecht (2012), estas três são as mais comuns: baseado em créditos, *on*/*off* e *ack*/*nack*.

No controle de fluxo baseado em créditos, o roteador transmissor consulta o espaço disponível no *buffer* do receptor antes de iniciar uma transmissão. O dado só é repassado caso haja espaço suficiente para seu armazenamento no roteador seguinte. A informação sobre o espaço disponível, geralmente, é indicada através de um canal específico entre os roteadores.

O controle de fluxo *on*/*off* se assemelha ao baseado em créditos, porém, o transmissor não tem acesso à quantidade de espaços disponíveis no *buffer* do receptor. No lugar disso, o transmissor tem acesso a apenas um sinal, responsável por informar se a transmissão pode ou não ser realizada. Geralmente, o receptor sinaliza "*on*" (pode transmitir) quando há espaço no *buffer* e "*off*" (não pode transmitir) quando ele está cheio.

No *ack*/*nack* o roteador transmissor não se preocupa com o espaço disponível no *buffer* do receptor. O primeiro simplesmente envia o dado e aguarda um sinal de recebimento vindo do segundo. Caso esse sinal seja um "*ack*", o transmissor reconhece o sucesso na transmissão e passa para o próximo dado. Caso o sinal seja um "*nack*", a interpretação é de que a transmissão foi mal sucedida, sendo reenviado o mesmo dado no próximo ciclo.

2.3.1.5 Arbitragem

Conforme mencionado no início da seção 2.3.1.4, os recursos são frequentemente disputados pelos pacotes na rede. Enquanto um pacote está sendo transmitido por determinado canal de saída, outros podem requerer o uso desse mesmo canal. A arbitragem é o mecanismo que resolve os conflitos entre dois ou mais pacotes (ou canais de entrada) que competem pelo mesmo recurso (ZEFERINO, 2003b). O arbitro é, portanto, responsável por determinar qual dos solicitantes pode utilizar o canal em dado momento.

A arbitragem pode ser classificada como centralizada, quando ela é realizada por um único módulo, ou distribuída, quando é feita por módulos independentes associados a cada recurso. Geralmente, o árbitro utiliza um ou mais critérios para seleção dos pacotes ou canais. Alguns deles são (HERVÉ, 2009):

- Prioridades estáticas e dinâmicas: nesses esquemas, os pacotes possuem diferentes níveis de prioridade, refletindo em maior ou menor preferência no momento da seleção. Na prioridade estática, cada pacote possui um nível fixo de prioridade, enquanto que na dinâmica, a prioridade pode variar de acordo com algumas circunstâncias, como, por exemplo, o tempo de permanência na fila de espera.
- Política de FCFS (*First-Come-First-Served*): os canais de entrada são selecionados por ordem de requisição, de modo que o primeiro a requisitar é o primeiro a ter acesso ao canal.
- Política de LRS (*Least-Recently-Served*): nesse método, o canal que foi menos vezes selecionado recentemente é o canal que será escolhido.
- Método *Round-Robin*: em uma possível implementação do *round-robin*, o árbitro disponibiliza a porta a qual está associado a cada canal de entrada por ciclo, de maneira circular e por intervalos de tempo idênticos. Caso um canal de entrada

queria iniciar uma transmissão, ele sinaliza para o árbitro no momento em que a porta lhe for oferecida. A partir desse momento, o canal de entrada tem acesso ao canal de saída até que conclua sua transmissão de dados. Após isto, o árbitro passa a vez para o próximo canal de entrada e o ciclo recomeça.

2.3.1.6 Memorização

O chaveamento por divisão de mensagem (visto na seção 2.3.1.3) exige que *buffers* sejam utilizados para o armazenamento temporário dos pacotes que trafegam pela rede. Isso possibilita que pacotes destinados a uma porta de saída já em uso sejam armazenados, desocupando canais anteriormente ocupados e tornando a rede mais disponível.

As técnicas de memorização são basicamente três (ZEFERINO, 2003b): memorização centralizada compartilhada, memorização na entrada e memorização na saída. Na primeira, há apenas um *buffer* central utilizado para armazenar pacotes bloqueados em qualquer canal de entrada. Tal *buffer* deve possuir largura de banda suficiente para atender todos os fluxos de entrada e de saída de dados simultaneamente.

A memorização na entrada é caracterizada pela existência de *buffers* independentes vinculados a cada canal de entrada. Segundo Rego (2006), esta é a técnica de memorização mais utilizada em NoCs. Ela possui diferentes implementações cujas modificações estão na forma de organização interna de seus buffers. Algumas das implementações, detalhadas em (ZEFERINO, 2003b), são: buffer FIFO, buffer SAFC, buffer SAMQ e buffer DAMQ.

Por fim, a memorização de saída possui *buffers* vinculados às portas de saída dos roteadores. Esses buffers devem suportar a demanda de todos os canais de entrada simultaneamente. Essa técnica exige também um controle de fluxo interno, entre canais de entrada e de saída.

2.3.1.7 Vantagens e desvantagens das NoCs

O uso de NoCs em ambientes multiprocessados tem sido considerado bastante adequado (FERNANDES, 2012; REGO, 2006). Seu estudo é motivado por vantagens que essas arquiteturas de interconexão apresentam em relação a outras, sendo as principais: escalabilidade da largura de banda, paralelismo na comunicação, eficiência energética e reusabilidade (WOSZEZENKI, 2007; ZEFERINO, 2003b).

Uma arquitetura de comunicação com largura de banda escalável é aquela na qual esta propriedade cresce à medida que o sistema aumenta (ZEFERINO, 2003b). Na comunicação através de barramentos, todos os elementos conectados compartilham o mesmo meio de transmissão e disputam por sua utilização. Isso resulta na divisão da largura de banda pelo número de elementos conectados, os tornando uma arquitetura com baixa escalabilidade. De modo oposto, em uma NoC, o número de canais de transmissão aumenta a cada novo núcleo adicionado ao sistema, devido utilizar enlaces ponto a ponto. Desse modo, um novo elemento inserido na rede não interfere na largura de banda dos outros já existentes, sendo possível o crescimento da rede sem a degradação do seu desempenho.

O paralelismo na comunicação e a eficiência energética também são vantagens proporcionadas pelos enlaces ponto a ponto. A primeira é devido à multiplicidade de canais de comunicação, o que permite a transferências de mensagens entre pares de elementos distintos de maneira simultânea (WOSZEZENKI, 2007). E a segunda é em consequência do uso de ligações ponto a ponto curtas para interligação dos elementos da rede. Essa característica, mais detalhada em (ZEFERINO, 2003b), reduz o consumo de energia.

A reusabilidade é uma característica existente também nas arquiteturas de interconexão baseadas em barramentos. Ela se refere à capacidade de reutilizar uma estrutura, sem modificações, em um mesmo projeto ou em projetos distintos. Nas NoCs, os roteadores e enlaces podem ser replicados e agregados ao sistema conforme o aumento do número de núcleos.

As Redes em Chip apresentam ainda algumas desvantagens, que, segundo Zeferino (2003b), são atenuadas pelos avanços tecnológicos e soluções arquiteturais, tais como maiores consumo de área de silício, latência na comunicação e complexidade de projeto (NOBRE, 2012). Em relação aos barramentos, a primeira e última desvantagem é bem visível. Já a latência na comunicação acontece por causa da contenção dos pacotes pelos roteadores para

pequenos processamentos, como o de roteamento. Isso acontece a cada salto, durante todo o percurso pela rede, o que gera atrasos na entrega das mensagens.

2.4 MPSOC

O aumento na capacidade de integração de transistores permitiu o surgimento de sistemas computacionais completos em um único chip, denominados Sistemas em chip, ou *System-on-chip* (SoC) (FERNANDES, 2012). Com isso, grandes computadores paralelos puderam se tornar pequenos chips, capazes de ser utilizados em computadores pessoais ou até sistemas embarcados (REGO, 2006). A característica fundamental de um SoC é a sua complexidade. Uma memória, por exemplo, possui um grande número de transistores integrados, mas sua estrutura regular a torna um componente, e não um sistema (WOLF; JERRAYA, 2005). Geralmente, os SoCs são formados por múltiplos componentes trabalhando em conjunto, como microcontroladores, processadores específicos, processadores de propósito geral, memórias, etc.

Um tipo especial de SoC, chamado *Multiprocessor System-on-Chip*, ou MPSoC, pode ser definido como um Sistema em Chip composto por múltiplos Elementos de Processamento (EPs) conectados por meio de uma estrutura de interconexão (REGO, 2006). Existem dois grupos distintos de MPSoC, conforme apresentado por Torres *et al*. (2011): heterogêneos e homogêneos. O primeiro grupo possui em sua composição EPs com diferentes funcionalidades, como um ou mais processadores de propósito geral, específicos (ASICs), de sinal digitais (Digital Signal Processors, DSPs), aceleradores, memórias, etc. No segundo grupo, os MPSoCs são compostos por múltiplos processadores, de propósito geral ou específicos, semelhantes e memórias. Em ambos os grupos, os EPs, geralmente, estão interconectados por uma NoC.

Basicamente, os MPSoCs heterogêneos são utilizados em aplicações específicas, com componentes adequados para tais aplicações. Isso permite melhores desempenho e eficiência energética, porém com maior dificuldade de programação. Os MPSoCs homogêneos, por sua vez, são mais utilizados em sistemas de processamento paralelo (WOLF *et al*., 2008), como consoles, *desktops*, servidores e supercomputadores (TORRES *et al*., 2011). Este grupo permite sistemas mais flexíveis, escaláveis e fáceis de programar, porém com menor eficiência energética.

2.5 TRABALHOS RELACIONADOS

A demanda por maiores poder computacional tem feito com que arquiteturas de processamento paralelo sejam cada vez mais utilizadas (AROCA; GONÇALVES, 2012). Os MPSoCs permitem, além dessas características, a reutilização de componentes de hardware, o que reduz o tempo, a complexidade e o custo de seu desenvolvimento. Um desafio existente em arquiteturas multiprocessadas atuais é o projeto de sistemas de comunicação eficientes entre seus elementos, que possuam ainda alta reusabilidade e escalabilidade. Isso reflete em várias pesquisas propondo MPSoCs e NoCs para esses sistemas com diferentes organizações, sendo algumas delas apresentadas nesta seção.

Uma abordagem, chamada de GigaNetIC, consiste em um MPSoC hierárquico escalável, introduzido por Puttmann *et al*. (2007). Sua arquitetura possui três níveis de hierarquia, chamados PE *level*, *cluster level* e SoC *level*. No nível mais baixo, o PE *level*, existe um subsistema chamado N-*Core*, responsável pela capacidade computacional do sistema. Ele é composto por um processador e uma memória cache de 32 KB para instruções e dados locais. No nível intermediário *cluster level*, quatro subsistemas N-*Core* estão conectados por um barramento local, que possibilita a troca de dados entre si. O controle desse barramento é realizado por um árbitro *round-robin* com distribuição uniforme de acesso. Conectado ao barramento, há também um elemento chamado S*witch Box* (SB), que liga o *cluster* de processadores a rede GigaNoC. No nível mais alto, chamado SoC *level*, cada *switch* está conectado a outros, possibilitando a comunicação entre *cluster*. As interconexões entre os *switches* podem estar organizadas em diferentes topologias. Por fim, a comunicação neste sistema é feita por meio de troca de mensagens organizadas em pacotes. A Figura 7 mostra o GigaNetIC com organização em grelha.

Segundo os autores, tal arquitetura foi implementada inicialmente em SystemC, com o objetivo de realizar testes de software e acertos de parâmetros de forma rápida, e, posteriormente, em VHDL e Verilog.

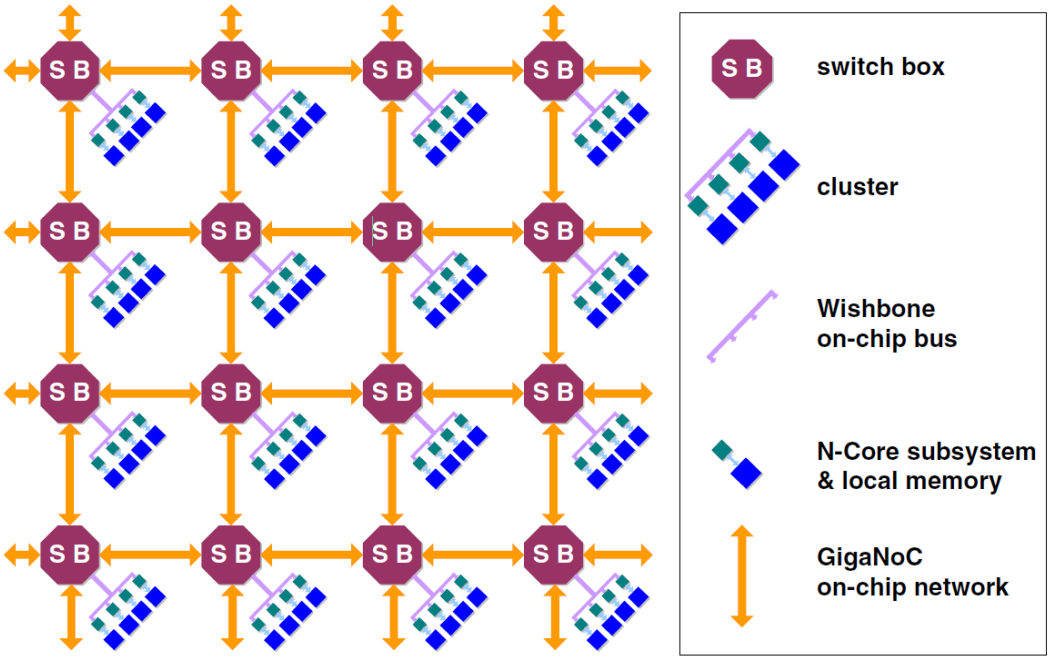


Figura 7: GigaNetIC em grelha. Fonte: (PUTTMANN *et al*., 2007).

Em outra abordagem MPSoC baseada em *cluster*, apresentada por Luo-Feng *et al*. (2010), 17 processadores Nios II foram integrados em um único sistema. Eles estão organizados em um processador central e quatro *clusters* de processamento, cada um com quatro processadores. A interconexão é feita por uma estrutura híbrida, onde uma hierarquia de barramentos fornece comunicação intra-*cluster* e uma NoC, comunicação inter-*cluster*, de maneira semelhante à abordagem GigaNetIC. A Figura 8 representa a arquitetura.

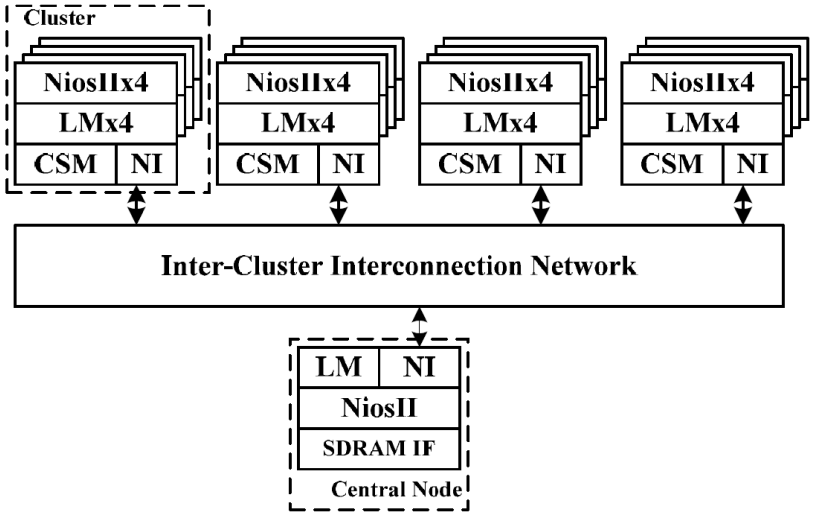


Figura 8: MPSoC homogêneo baseado em cluster. Fonte: (LUO-FENG *et al*., 2010).

A comunicação entre *clusters* é feita através da memória compartilhada, representada na figura pelo módulo CMS. Cada processador possui sua memória local, usada na execução

de tarefas, e tem acesso a NoC por meio da interface de rede NI. A fim de otimizar o desempenho do *cluster*, cada processador possui um barramento exclusivo, para acesso a sua memória local, e um compartilhado, para acesso a memória compartilhada. O processador central, destacado como *Central Node*, é a unidade principal que controla todo o sistema. Os clusters e o processador central estão conectados por uma NoC irregular, composta por cinco roteadores. Nessa NoC, a comunicação é realizada por meio de pacotes uniformes com tamanho ilimitado e 34 bits de largura.

O trabalho desenvolvido por Zheng *et al*. (2010), chamado de *Hybrid Communication Reconfigurable Network on Chip*, ou HCR-NoC, apresenta uma estrutura diferente das mostradas até o momento. De forma inversa, sua hierarquia de rede é formada por NoCs locais e um barramento interligando essas NoCs. A Figura 9 demonstra como esta arquitetura está organizada.

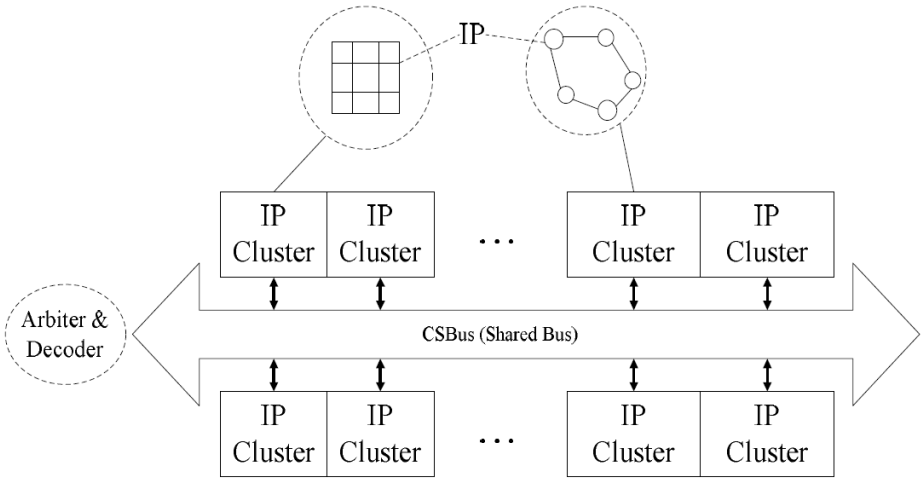


Figura 9: Arquitetura HCR-NoC. Fonte: (ZHENG *et al*., 2010).

Na figura, as NoCs que interligam os elementos de processamento estão representadas pelos módulos nomeados IP *Cluster*. Cada *cluster* está ligado ao barramento compartilhado CSBus. Segundo os autores, este barramento utiliza a técnica TDMA, na qual o tempo de uso do barramento é dividido entre todos os elementos conectados, permitindo múltiplas comunicações ao longo do tempo.

A quarta abordagem apresentada aqui é a arquitetura CHNoC, proposta por Leng *et al*. (2005). Nesta arquitetura, a comunicação é realizada por meio de uma hierarquia de NoCs, divididas em duas redes: *Edge Network* e *Core Network*. A primeira rede é composta por *Edge Routers* (ER) e *Cluster Nodes* (CN). A *Core Network* é formada apenas por roteadores, denominados *Core Routers* (CR). A Figura 10 mostra a organização dessas redes e componentes.

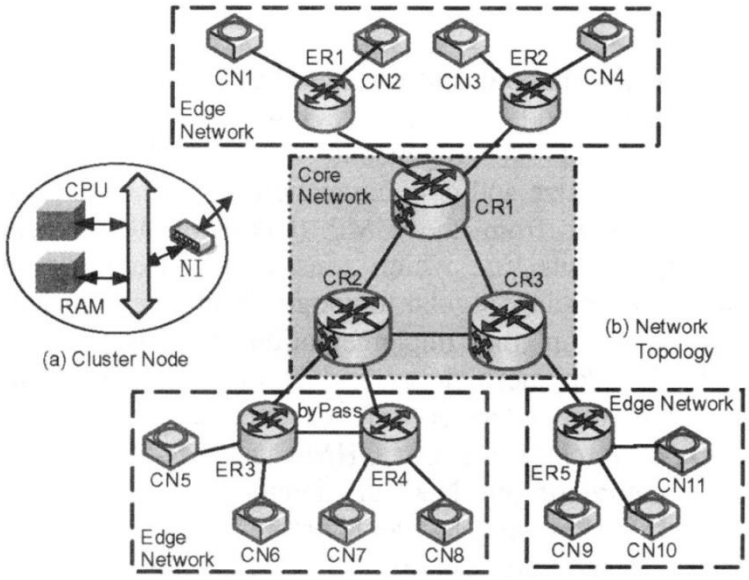


Figura 10: Arquitetura CHNoC. Fonte: (LENG *et al*., 2005).

O módulo CN é onde os EPs do *cluster* estão localizados. Na figura, o CN é exemplificado contendo um processador (CPU), uma memória (RAM) e uma interface de rede (NI) conectados via barramento. Em geral, o CN é composto por vários processadores e uma interface de rede, que é a responsável por realizar a comunicação entre *clusters*. Os *clusters* estão conectados entre si através de um ER e os ERs, por sua vez, pelos CRs. A comunicação é realizada por troca de pacotes.

Em outra abordagem, Seifi e Eshghi (2008) propõem um trabalho onde uma NoC com roteadores adaptados é utilizada para a comunicação entre seus EPs. Nessa rede em chip, chamada *Clustered* NoC, os roteadores foram desenvolvidos de modo a suportar até quatro EPs conectados. Para isso, eles contam com quatro portas locais além das quatro portas (Norte, Sul, Leste e Oeste) para a comunicação com outros roteadores, comuns em topologias do tipo grelha. A Figura 11 a seguir apresenta a arquitetura desse roteador.

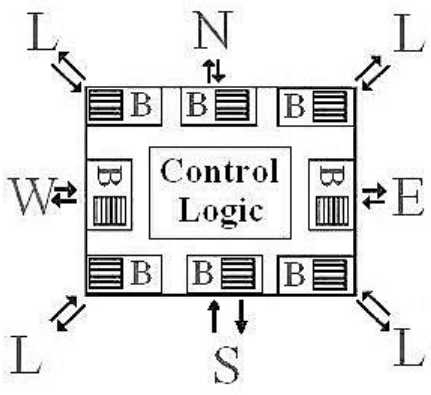


Figura 11: Arquitetura do roteador. Fonte: (SEIFI; ESHGHI, 2008).

A comunicação entre EPs é feita por meio da troca de mensagens divididas em pacotes. Cada pacote é composto por uma quantidade fixa de *flits* de tamanho variável, onde o primeiro corresponde ao endereço do roteador de destino e o segundo, ao tamanho da carga útil do pacote. Os roteadores são endereçados por sua posição na rede XY, cujo valor de X é a posição horizontal e Y a posição vertical. Diferentemente dos trabalhos anteriores, este utiliza apenas uma NoC para a comunicação entre EPs e entre os *clusters* formados a partir do agrupamento de quatro desses elementos. Apesar da simples estrutura para a criação de uma rede de *clusters*, o tamanho desses *clusters* está limitado a quatro EPs. A Figura 12 apresenta uma visão geral desse sistema.

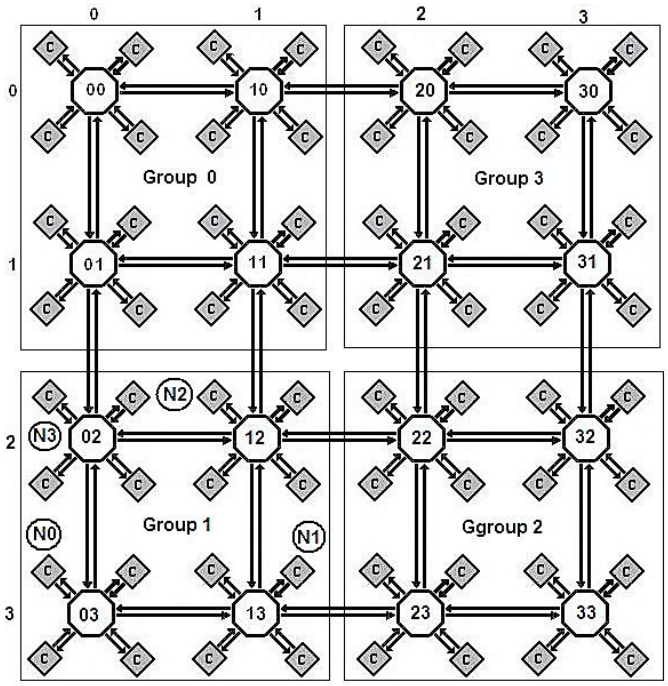


Figura 12: Visão geral da arquitetura. Fonte: (SEIFI; ESHGHI, 2008).

A principal semelhança entre os trabalhos apresentados nesta seção é a existência de redes hierárquicas, isto é, redes locais de EPs, com capacidade de comunicação entre seus elementos, e uma rede global, que permite a comunicação entre redes, também chamadas de *clusters*. O presente trabalho propõem um sistema MPSoC que utiliza a arquitetura IPNoSys como elemento de processamento, interligadas por uma NoC simples, através da qual a comunicação ocorre. Essa nova arquitetura será detalhada no capítulo 4. Para comparação entre os sistemas apresentados e o desenvolvido neste trabalho, deve-se considerar a IPNoSys como uma rede de processamento, formada a partir de suas RPUs. Desse modo, pode-se perceber a hierarquia de redes existente, onde cada IPNoSys do sistema corresponde a uma rede local, formada pelos núcleos de processamento RPU, e a NoC para comunicação entre

IPNoSys corresponde à rede global. A comunicação entre as redes locais do MPSoC, ou seja, a comunicação inter-*cluster*, é feita através das MAUs de cada IPNoSys. Estas podem ser comparadas aos módulos NI, encontrados em algumas abordagens descritas.

Entre as principais diferenças acerca dos trabalhos apresentados e do MPSoC IPNoSys, a primeira está na estrutura de interconexão. Nas abordagens relacionadas, ela geralmente é feita por uma rede híbrida, formada por barramentos e NoCs. Neste sistema é utilizado apenas NoCs em topologia grelha, tanto nas redes locais quanto globais. A segunda diferença está na característica de separabilidade entre computação e comunicação. Como pode ser visto nos trabalhos apresentados, todos eles possuem roteadores ou barramentos, responsáveis pela comunicação, e EPs, encarregados de realizar a computação. No MPSoC IPNoSys, a computação e comunicação entre os elementos das redes locais são realizadas pela mesma unidade: a RPU.

3 ARQUITETURA IPNOSYS

As características a respeito da arquitetura IPNoSys relevantes para o desenvolvimento deste trabalho são apresentadas neste capítulo. Caso haja o interesse, informações mais aprofundadas sobre a arquitetura podem ser conseguidas em (FERNANDES, 2012). A seção 3.1 expõe uma visão geral, com as principais unidades e modelo de processamento. Na seção 3.2 é explicado o algoritmo de roteamento desenvolvido para a arquitetura. A seção 3.3 mostra o formato dos pacotes que trafegam pela rede da IPNoSys. As seções 3.4 e 3.5 apresentam a Unidade de Roteamento e Processamento e a Unidade de Acesso à Memória, respectivamente. Por fim, a seção 3.6 introduz o modelo de programação da arquitetura.

3.1 VISÃO GERAL

A *Integrated Processing NoC System* (IPNoSys) é uma arquitetura baseada em NoC, proposta por Fernandes (2012), onde os roteadores foram modificados para permitir tanto a transmissão quanto o processamento dos dados. Para que isso fosse possível, foram adicionadas aos roteadores uma *Arithmetic Logic Unit* (ALU) e uma *Synchronization Unit* (SU). Esse conjunto passou a ser chamado de Unidade de Processamento e Roteamento ou RPU (*Routing and Processing Unit*) (FERNANDES *et al*., 2009a). A IPNoSys não utiliza processadores convencionais, sendo rede de RPUs as responsáveis por executar os pacotes de instruções que compõem os programas. A quantidade de RPUs pode ser configurada, mas sempre com a mesma quantidade de linhas e colunas, organizadas em grelha 2D, como mostrado na Figura 13.

Está presente na IPNoSys um espaço de memória dividido em quatro módulos, localizados em cada canto da rede. Neles, os dados e aplicações são armazenados no formato de pacotes, podendo ser acessados somente por componentes associados a cada módulo de memória, chamados de Unidade de Acesso à Memoria ou MAU (*Memory Access Unit*). Apesar de serem módulos distribuídos, o espaço de endereçamento é único e igualmente compartilhado para toda a arquitetura (FERNANDES, 2012). A unidade denominada IOMAU é diferenciada das demais por compartilhar a memória associada com o sistema de entrada e saída (E/S). Nesse sistema, o IONode, que possui acesso direto à memória assim como as

MAUs, controla as operações de E/S (FERNANDES *et al*., 2011). Para que não haja conflito no momento de acessar a memória, o MemArbiter é usado para coordenar sua utilização.

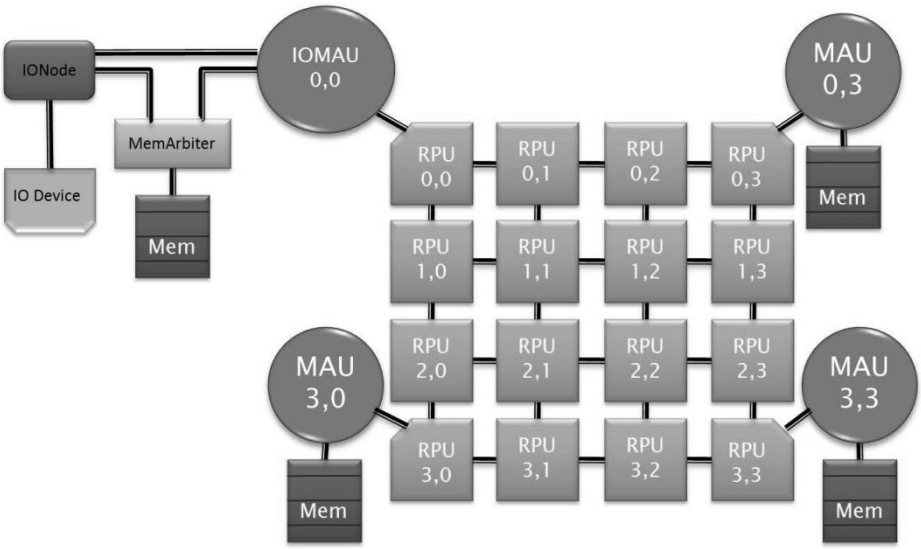


Figura 13: Arquitetura IPNoSys. Fonte: (FERNANDES, 2012).

As MAUs são as responsáveis por injetar os pacotes armazenados na memória para o processamento na rede de RPUs, podendo existir até quatro sendo executados em paralelo. Esses pacotes funcionam como uma sequência de instruções, onde a operação corrente é aquela no início do pacote. Os resultados produzidos pelas operações podem ser inseridos em posições posteriores do mesmo pacote, para uso futuro, ou enviados para a memória, sendo apenas armazenados ou colocados em outros pacotes já existentes na memória (FERNANDES *et al*., 2008). Cada RPU alcançada pelo pacote executa uma quantidade de instruções, determinada no momento de compilação, e encaminha o restante para a próxima, seguindo de acordo com o algoritmo de roteamento. Dessa forma, o pacote vai diminuindo até que todas as instruções tenham sido executadas. A IPNoSys utiliza um modelo diferente de execução, no qual os dados e instruções estão, normalmente, juntos. Isso reduz a quantidade de acessos à memória em busca de dados.

A Figura 14 apresenta um exemplo onde a instrução ADD é executada e o resultado produzido é inserido numa posição posterior desse mesmo pacote. A Figura 14 (a) mostra o estado do pacote antes da execução e a Figura 14 (b) mostra o pacote após a execução. Como pode ser visto, a instrução executada e os dois operandos deixam de fazer parte do pacote e o resultado da operação é inserido na posição 8, como determinado na instrução. Informações sobre o conjunto de instruções da IPNoSys são apresentadas na seção 3.6.

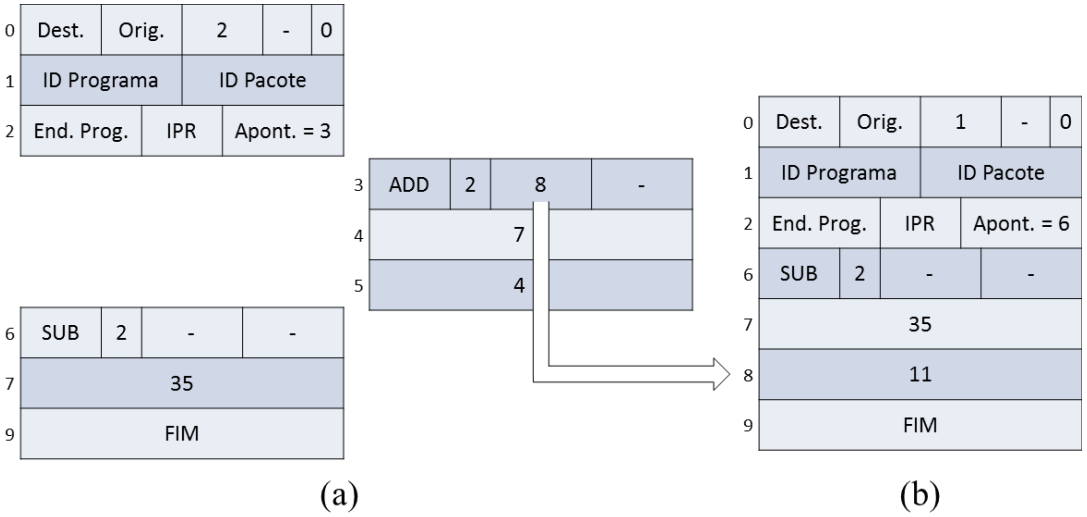


Figura 14: Exemplo de execução. Fonte: (FERNANDES, 2012) modificada pelo autor.

Para que um pacote seja completamente processado, é necessário que o algoritmo de roteamento o mantenha na rede até que todas as instruções tenham sido executadas. Isso exige que a distância entre a origem e o destino do pacote seja suficientemente grande. Porém, é possível haver mais instruções que o número de RPUs da rede, o que ocasionaria a execução incompleta dos pacotes. Como solução, os pacotes são roteados novamente para novos destinos enquanto possuírem instruções, seguindo o algoritmo denominado *spiral complement*, que foi projetado para esse fim.

3.2 ALGORITMO DE ROTEAMENTO

O algoritmo de roteamento da IPNoSys, denominado *spiral complement*, tem como objetivo traçar uma sequência de RPUs suficientemente grande para que todas as instruções do pacote sejam executadas. Isso é possível porque o algoritmo encontra uma nova rota sempre que um pacote chega ao seu destino possuindo instruções pendentes (FERNANDES *et al.*, 2009a). Supondo que um pacote seja injetado para processamento na rede pela IOMAU 0,0, a espiral formada durante o percurso do pacote é a apresentada na Figura 15.

Inicialmente, o pacote é injetado na rede pela IOMAU 0,0. Seu primeiro destino será a RPU mais distante, representada pelo complemento da origem, ou seja, a RPU 3,3 no canto oposto da rede. O pacote segue através de todas as RPUs entre a origem e o destino de acordo com o algoritmo de roteamento XY. Nesse algoritmo, o percurso acontece primeiramente no

sentido horizontal, até atingir a coluna do destinatário, e depois no vertical, alcançando seu destino. Caso o pacote seja entregue contendo instruções não executadas, ele é novamente roteado para o destino mais distante possível, que seria, mais uma vez, o canto oposto da rede (RPU 0,0). Porém, para que o pacote também atinja RPUs mais internas, uma linha ou uma coluna da rede é desconsiderada a cada novo roteamento. Dessa forma, o novo destino ainda será o canto oposto, mas da rede reduzida. Na Figura 15, a linha 0 foi desconsiderada, sendo a RPU 1,0 o novo destino do pacote.

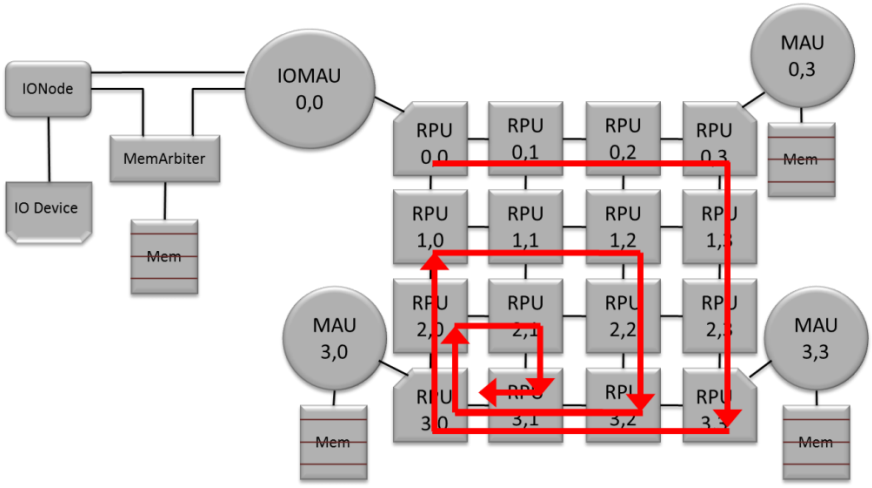


Figura 15: Percurso no algoritmo spiral complement. Fonte: (FERNANDES, 2012) modificada pelo autor.

Caso toda a espiral mostrada na Figura 15 seja percorrida e ainda restarem instruções no pacote, o algoritmo reinicia uma nova espiral tendo como ponto de partida a RPU onde a espiral anterior se encerrou. Esse algoritmo é capaz de criar caminhos que permitem o processamento de pacotes de qualquer tamanho em redes de qualquer dimensão.

Uma situação inconveniente que pode ocorrer na execução desse algoritmo é o *deadlock*. Esse problema é causado quando há uma dependência cíclica entre núcleos de uma rede (FERNANDES, 2012). Por exemplo, no caso da IPNoSys, o *deadlock* pode ocorrer quando um pacote for longo o suficiente para que o roteamento leve-o novamente a uma RPU, para execução da próxima instrução, onde o restante do pacote ainda se encontra sendo transmitido. Isso levaria a uma dependência cíclica para liberação dos *buffers* dos canais de transmissão do caminho do pacote. Para resolver o *deadlock*, foi adotada uma estratégia denominada execução localizada, que consiste em continuar executando instruções do pacote até que surja espaço na próxima RPU e a transmissão possa acontecer (FERNANDES *et al*., 2011).

3.3 FORMATO DO PACOTE

O pacote da arquitetura IPNoSys é formado por uma quantidade variável de palavras de 32 bits, existindo três partes distintas: as três primeiras palavras formam o cabeçalho, a última possui o terminador, que representa o fim do pacote, e o que há entre essas duas partes é o conjunto de instruções e operandos (FERNANDES, 2012). Existem ainda mais quatro bits de controle que informam o tipo de cada palavra transmitida: cabeçalho, fim de pacote, instrução ou operando, representados pelas letras C, F, I e O, respectivamente, na Figura 16.

A seguir é apresentada uma breve descrição a respeito dos campos de cada palavra do pacote e seus respectivos tamanhos em bits (FERNANDES, 2012).

- Cabeçalho – bit de controle C ativado.
    - Destino (8 bits): endereço da RPU de destino do roteamento atual.
    - Origem (8 bits): endereço da RPU que iniciou o roteamento atual.
    - # Instruções (8 bits): quantidade de instruções presentes no pacote.
    - Re-rotear (6 bits): utilizado para o cálculo do próximo endereço de destino.

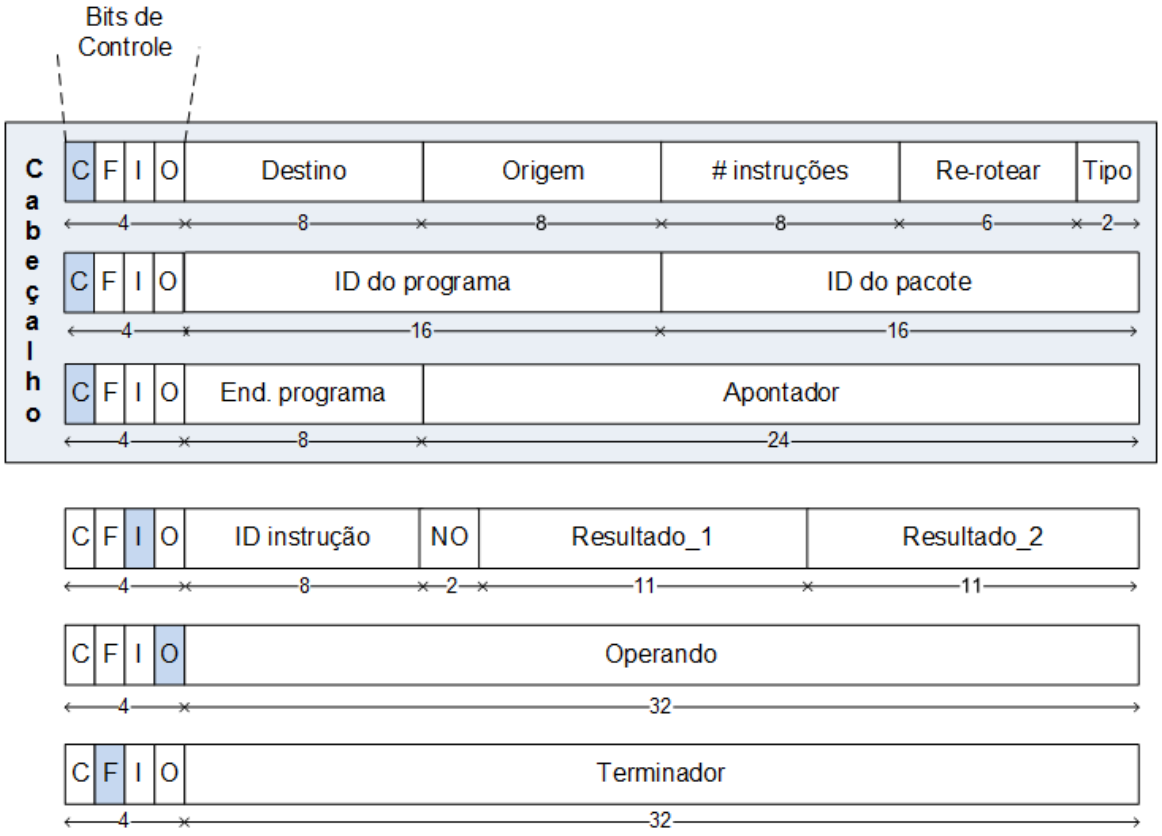


Figura 16: Representação do pacote da arquitetura IPNoSys. Fonte: (FERNANDES, 2012).

  - Tipo (2 bits): indica o tipo do pacote ("00_b" = pacote de controle; "01_b" = pacote regular; "10_b" = pacote interrompido; "11_b" = pacote chamador de função).
  - ID do programa (16 bits): identificador de uma determinada aplicação.
  - ID do pacote (16 bits): identificador do pacote de um programa.
  - Endereço do programa (8 bits): endereço da MAU que iniciou o programa.
  - IPR (8 bits): informa a quantidade de instruções do pacote que devem ser executadas em cada RPU.
  - Apontador (16 bits): informa qual a posição da próxima instrução que deve ser executada no pacote.
- Instrução – bit de controle I ativado.
  - ID da Instrução (8 bits): identificador da instrução.
  - NO (2 bits): número de operandos envolvidos na execução da instrução ("00_b" = nenhum operando; "01_b" = um operando; "10_b" = dois operandos; "11_b" = número de operandos informado pelo campo Resultado_2).
  - Resultado_1 (11 bits): em pacotes de controle, este campo é utilizado para indicar o endereço da MAU que deverá executar a instrução a qual está associado. Nos demais pacotes, ele é utilizado para indicar a posição onde o resultado da operação deve ser inserido no mesmo pacote.
  - Resultado_2 (11 bits): quando o campo "NO" possui valor "11_b", este campo indica a quantidade de operandos menos um envolvidos na instrução. Quando "NO" possui outros valores, este campo indica uma segunda posição no pacote onde o resultado da instrução também deve ser inserido.
- Operando – bit de controle O ativado.
  - Operando (32 bits): dado utilizado como operando da instrução.
- Fim de pacote – bit de controle F ativado.
  - Terminador (32 bits): palavra contendo 32 zeros que indica o fim do pacote.

O cabeçalho e o terminador são elementos únicos em um pacote, enquanto que as instruções e operandos têm sua quantidade variável, dependendo da aplicação. Existem quatro tipos de pacotes: regular, controle, interrompido e *caller* (FERNANDES, 2012). Os pacotes regulares carregam um conjunto de instruções que são executadas nas RPUs da arquitetura. Os pacotes de controle se diferenciam dos regulares por carregarem apenas uma instrução e estarem endereçados para uma MAU específica, responsável por executar tal instrução. Os

pacotes interrompidos são os regulares que tiveram sua execução impossibilitada por estarem esperando uma operação de E/S. E os pacotes *caller* são pacotes regulares que realizaram uma chamada de função, que é executada por meio de outro pacote. Os pacotes interrompido e *caller* retornam para a memória e permanecem até que seu processamento possa continuar.

3.4 UNIDADE DE ROTEAMENTO E PROCESSAMENTO

As Unidades de Roteamento e Processamento, ou RPU (*Routing and Processing Unit*) substituem os roteadores na NoC da IPNoSys. A esse componente, foi adicionada a capacidade de realizar operações lógico-aritméticas e instruções de salto (FERNANDES *et al*., 2008). A RPU possui em sua arquitetura uma ALU, uma SU, buffers de entrada, um *crossbar* e árbitros nas saídas. Devido a arquitetura utilizar a topologia grelha 2D, as RPUs interiores possuem quatro portas de comunicação e as mais exteriores possuem apenas três (Figura 13). Cada porta é ligada a uma RPU adjacente, com exceção das RPUs nos cantos da rede, onde uma das portas é ligada a uma MAU (FERNANDES *et al*., 2009b).

Ao receber um pacote, a RPU o armazena em um *buffer* de entrada e checa o seu tipo. Caso seja de controle, interrompido ou *caller*, ele é transmitido por um canal virtual em direção ao seu destino, utilizando o algoritmo de roteamento XY tradicional. Caso seja um pacote regular, a RPU verifica se é o destinatário, a fim de decidir se deve calcular um novo destino de acordo com o algoritmo *spiral complement*. Se a RPU for o destino do pacote, ela calcula um novo e o roteia, se não, apenas o roteia. O canal virtual utilizado por pacotes regulares é diferente do canal utilizado pelos demais.

Em cada porta de saída, existe um árbitro para resolver os conflitos entre pacotes que solicitam a porta. Antes de iniciar a transmissão de um pacote regular, esse árbitro solicita que uma quantidade de instruções predeterminada no momento da compilação da aplicação seja executada pela ALU, caso sejam instruções logico-aritmética ou de salto, ou pela SU, caso sejam instruções de acesso à memória ou sincronização (FERNANDES *et al*., 2009b). A SU é responsável por criar pacotes de controle destinados a MAU determinada pela instrução. Após receber a confirmação da execução das instruções, o árbitro as remove, juntamente com os operandos, adiciona a quantidade removida ao apontador do pacote e inicia a transmissão. Se for gerado algum resultado, eles são armazenados em um *buffer* de resultados e inseridos no

pacote no momento de sua transmissão. Se um salto for realizado, as instruções desviadas são descartadas.

A execução localizada, comentada na seção 3.2, é realizada quando uma próxima RPU não tem buffer suficiente para receber o cabeçalho de um pacote ou quando ocorre uma situação de iminência de *deadlock*. Nesses casos, o árbitro continua a solicitar que instruções sejam executadas pela ALU ou pela SU da RPU em que o pacote se encontra.

3.5 UNIDADE DE ACESSO À MEMÓRIA

Na IPNoSys, existem quatro Unidades de Acesso à Memória, ou MAUs (*Memory Access Unit*), localizadas, cada uma, em um canto da rede (Figura 13). Na arquitetura, as aplicações são representadas por pacotes que estão armazenados na memória, sendo as MAUs responsáveis por lê-los e injetá-los na NoC para a execução (FERNANDES *et al*., 2009b). Dessa forma, é possível haver até quatro pacotes regulares sendo injetados para processamento na rede de RPUs ao mesmo tempo, isto é, até quatro fluxos de execução em paralelo. Elas também realizam as operações de leitura e escrita de informações, por meio da execução de instruções de acesso à memória e sincronização.

A MAU possui três componentes: Gerenciador de Memória, Gerenciador de Pacotes e Unidade de Controle (FERNANDES, 2012). O primeiro permite a localização de valores ou pacotes na memória através do endereço de onde estão armazenados. O Gerenciador de Pacotes é a unidade responsável por executar instruções que são enviadas para as MAUs por meio de pacotes de controle. Essas instruções solicitam leitura ou escrita de dados, inserção de valores em pacotes armazenados na memória e injeção de pacotes na rede de RPUs. Por fim, a Unidade de Controle coordena as tarefas entre os dois gerenciadores e faz o controle de entrada e saída de pacotes nas MAUs. A arquitetura da MAU é resumida pela Figura 17.

As instruções executadas pelas MAUs chegam até elas por meio de pacotes de controle enviados a partir das RPUs. Esses pacotes são transmitidos por canais virtuais livres de *deadlocks*, pois além dos pacotes serem curtos, eles não percorrem rotas circulares (FERNANDES *et al*., 2009b). As instruções que não são executadas nas RPUs, são, portanto, executadas nas MAUs.

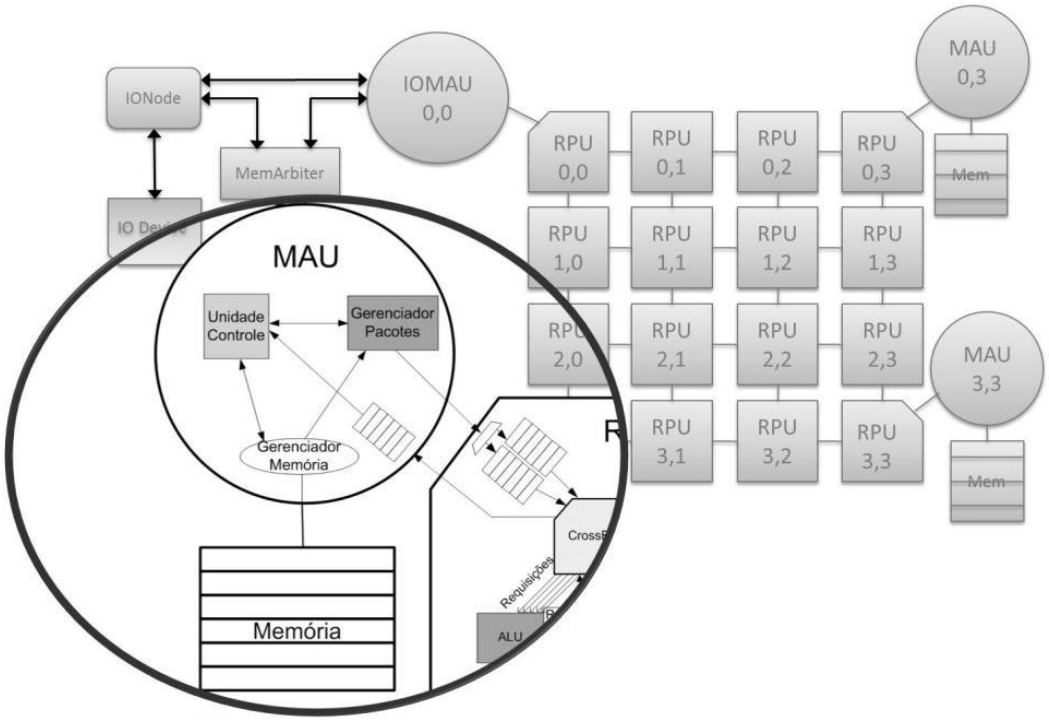


Figura 17: Arquitetura da MAU. Fonte: (FERNANDES, 2012).

3.6 PROGRAMABILIDADE

A IPNoSys possui um conjunto de instruções composto por 32 instruções, resumidas na Tabela 1.

Tabela 1: Conjunto de instruções da IPNoSys. Fonte (FERNANDES, 2012)

Codop	Instrução	Tipo	# Operandos	Descrição
0	ADD	Aritmética	2	Soma 2 inteiros
1	SUB	Aritmética	2	Subtrai 2 inteiros
2	MUL	Aritmética	2	Multiplica 2 inteiros
3	DIV	Aritmética	2	Divide 2 inteiros
4	NOT	Lógica	1	Negação de um valor
5	AND	Lógica	2	Conjunção de dois valores
6	OR	Lógica	2	Disjunção de dois valores
7	XOR	Lógica	2	Ou-exclusivo de dois valores
8	RSHIFT	Deslocamento	2	Desloca *n* bits de um valor à direita
9	LSHIFT	Deslocamento	2	Desloca *n* bits de um valor à esquerda
10	LOAD	Acesso à memória	1	Solicita um valor da memória

11	STORE	Acesso à memória	Vários	Armazena um valor na memória
12	EXEC	Sincronização	1	Ordena injeção imediata de um pacote
13	SYNEXEC	Sincronização	Vários	Ordena a injeção de um pacote após sincronização
14	SYNC	Sincronização	1	Sinal de sincronização para um pacote
15	RELOAD	Acesso à memória	1	Retorna um valor carregado na memória
16	BE	Condicional	2	Desvia se igual
17	BNE	Condicional	2	Desvia se diferente
18	BL	Condicional	2	Desvia se menor
19	BG	Condicional	2	Desvia se maior
20	BLE	Condicional	2	Desvia se menor ou igual
21	BGE	Condicional	2	Desvia se maior ou igual
22	JUMP	Incondicional	0	Desvia incondicionalmente
23	COPY	Auxiliar	1	Copia um valor para outra instrução no mesmo pacote
24	NOP	Auxiliar	0	Sem operação
25	SEND	Sincronização	2	Envia um valor para ser inserido em um pacote
26	IN	Entrada/Saída	3	Recebe bytes do controlador de E/S
27	OUT	Entrada/Saída	3	Envia bytes ao controlador de E/S
28	WAKEUP	Sistema	1	Ordena a reinjeção de um pacote antes interrompido
29	NOTIFY	Sistema	1	Notivida estado de um pacote
30	CALL	Procedimento	Vários	Faz a chamada de uma função/pacote
31	RETURN	Procedimento	2	Retorna o resultado de uma função para o chamador

Conforme já apresentado na Figura 16, os pacotes possuem um cabeçalho, um terminador e um conjunto de instruções e operandos. Neste conjunto, cada instrução possui um formato fixo, incluindo uma palavra do tipo instrução com uma ou mais do tipo operando vinculadas (excetuando-se as instruções JUMP e NOP, que não possuem operandos). A Figura 18 mostra o formato geral das instruções nos pacotes. A primeira linha representa a palavra de instrução e as demais, as palavras de operando.

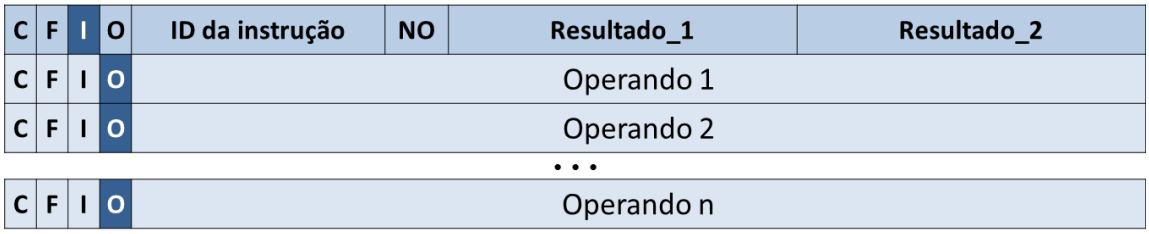
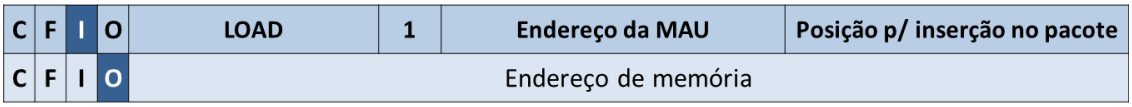


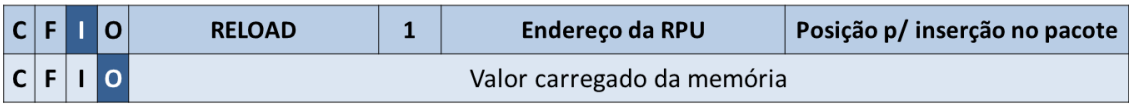
Figura 18: Formato da instrução. Fonte: (FERNANDES, 2012) modificada pelo autor.

Nos parágrafos seguintes, serão detalhados os formatos das seguintes instruções: LOAD, STORE, EXEC, SEND, SYNC, RELOAD e SYNEXEC. O formato das demais, mostradas na Tabela 1, não será apresentado, pois é relevante para este trabalho o conhecimento mais detalhado de apenas as sete instruções citadas neste parágrafo. Caso haja interesse em conhecer o formato das demais, Fernandes (2012) detalha todas as instruções.

A instrução LOAD é utilizada para solicitar à MAU o carregamento de um valor armazenado na memória para uma posição específica no pacote que está sendo executado. Ao receber essa instrução, a MAU produz um pacote de controle endereçado para a RPU que enviou o LOAD, com uma instrução RELOAD e o valor solicitado. Esta instrução é reconhecida na RPU como resposta ao LOAD. A organização das informações necessárias para a execução dessas instruções é mostradas na Figura 19.



(a)



(b)

Figura 19: Formato da instrução LOAD (a) e RELOAD (b). Fonte: (FERNANDES, 2012) modificada pelo autor.

A instrução STORE é usada para armazenar um determinado valor na memória. Na IPNoSys, é possível armazenar um valor em múltiplos endereços de memória. Por essa razão, o campo NO possui o valor 3 e o campo Resultado_2, a quantidade de operandos que a instrução possui. No campo Resultado_1, está o endereço da MAU que executará o STORE. O primeiro operando possui o valor que será armazenado e os demais possuem os endereços de memória. O formato da STORE é mostrado na Figura 20.

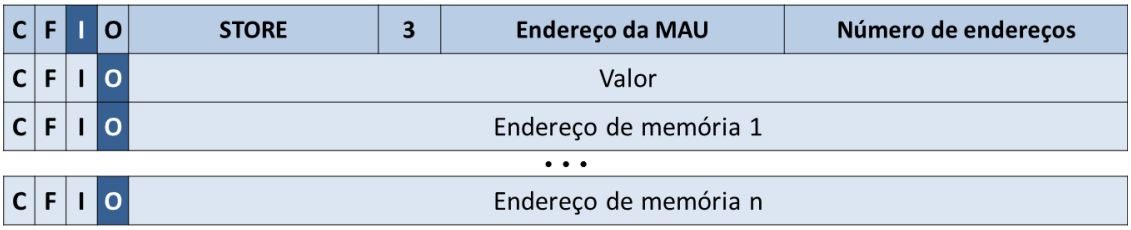


Figura 20: Formato da instrução STORE. Fonte: (FERNANDES, 2012) modificada pelo autor.

As instruções EXEC e SYNEXEC são utilizadas para solicitar que MAUs injetem pacotes, localizados na memória, na rede de RPUs. Em ambas as instruções, um pacote de controle é gerado pela RPU e enviado para uma MAU específica, carregando os identificadores do pacote que deve ser inserido na rede. Dois identificadores, de programa e de pacote, são necessários. Isso porque a IPNoSys permite a execução simultânea de vários programas, compostos por um ou mais pacotes.

A diferença entre as duas instruções é que, na SYNEXEC, a injeção do pacote só deve acontecer após o recebimento de sinais de sincronização. Os sinais esperados devem ser especificados junto com a instrução SYNEXEC. Eles chegam até a MAU por meio da instrução SYNC, que será apresentada a seguir. O formato das instruções EXEC e SYNEXEC é mostrado na Figura 21.

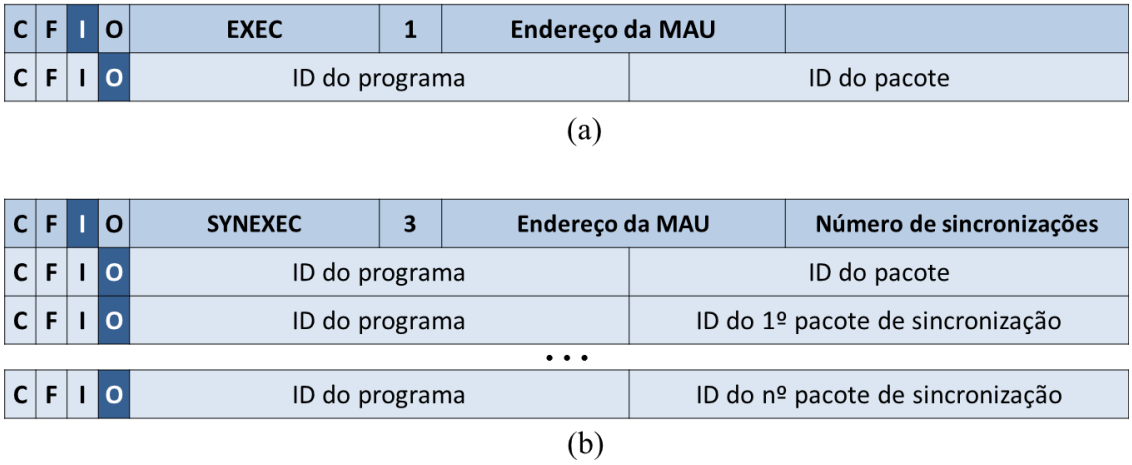


Figura 21: Formato das instruções EXEC (a) e SYNEXEC (b). Fonte: (FERNANDES, 2012) modificada pelo autor.

A instrução SYNC é utilizada para enviar sinais de sincronização para as MAUs. Nesta instrução, é especificada qual MAU deve receber o sinal de SYNC e qual pacote aguarda este sinal, tendo formato conforme apresentado na Figura 22 (a). A RPU, ao encontrar uma instrução desse tipo, cria um pacote de controle endereçado para a MAU especificada contendo dois operandos. O primeiro operando armazena os identificadores do pacote regular que espera o SYNC e o segundo, os identificadores do pacote que enviou tal sinal. Desse modo, a instrução no pacote de controle apresenta formato como mostrado pela Figura 22 (b).

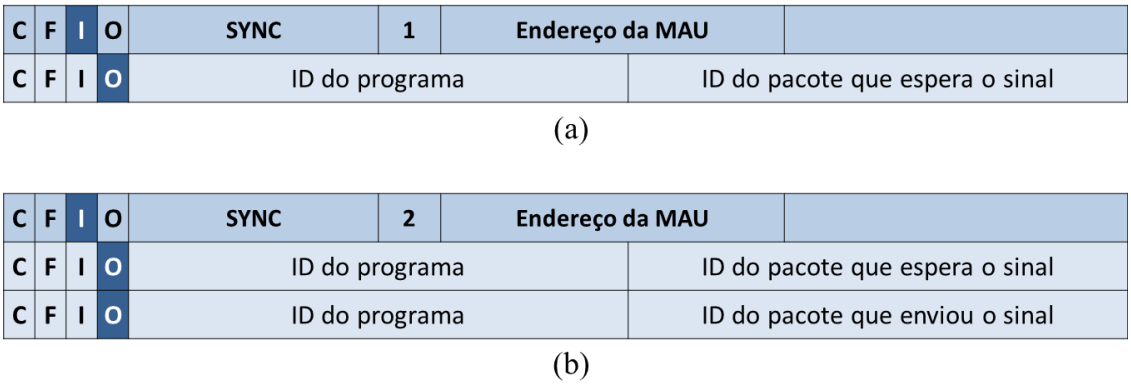


Figura 22: Formato da instrução SYNC. Fonte: (FERNANDES, 2012) modificada pelo autor.

Por fim, a instrução SEND é utilizada para enviar um dado valor para um pacote regular ainda armazenado na memória, sendo bastante utilizada em laços de repetição, por exemplo. Nessa instrução, são necessárias a posição de inserção do valor dentro do pacote, os identificadores deste pacote, formado pelo par ID do programa e ID do pacote, e o endereço de MAU onde tal pacote se encontra armazenado. Essas informações estão organizadas como mostrado na Figura 23.

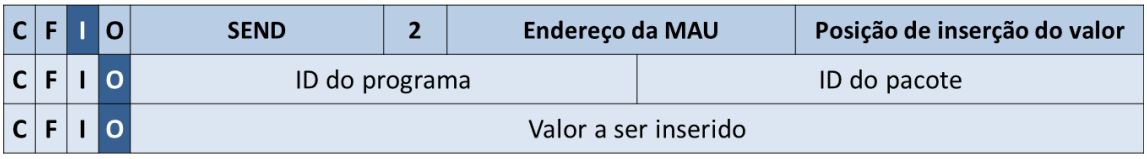


Figura 23: Formato da instrução SEND. Fonte: (FERNANDES, 2012) modificada pelo autor.

Utilizando o conjunto de instruções apresentado na Tabela 1, é possível desenvolver programas, baseados em pacotes, executáveis na arquitetura IPNoSys. Os programas podem estar divididos em quantos pacotes for necessário, cada um contendo suas instruções e dados. Para facilitar o desenvolvimento de aplicações, foi desenvolvida uma Linguagem de Descrição de Pacotes (PDL – *Package Description Language*), possuindo um nível intermediário entre linguagens de alto nível e o formato dos pacotes. No Apêndice 1 de (FERNANDES, 2012), é apresentado um guia de programação para a linguagem PDL.

A linguagem PDL deve ser processada por um montador, semelhante à linguagem *assembly*. O montador traduz o código PDL em código objeto da arquitetura, ou seja, em pacotes binários, que é utilizado como entrada pelo simulador da arquitetura. A Figura 24 representa esse processo.

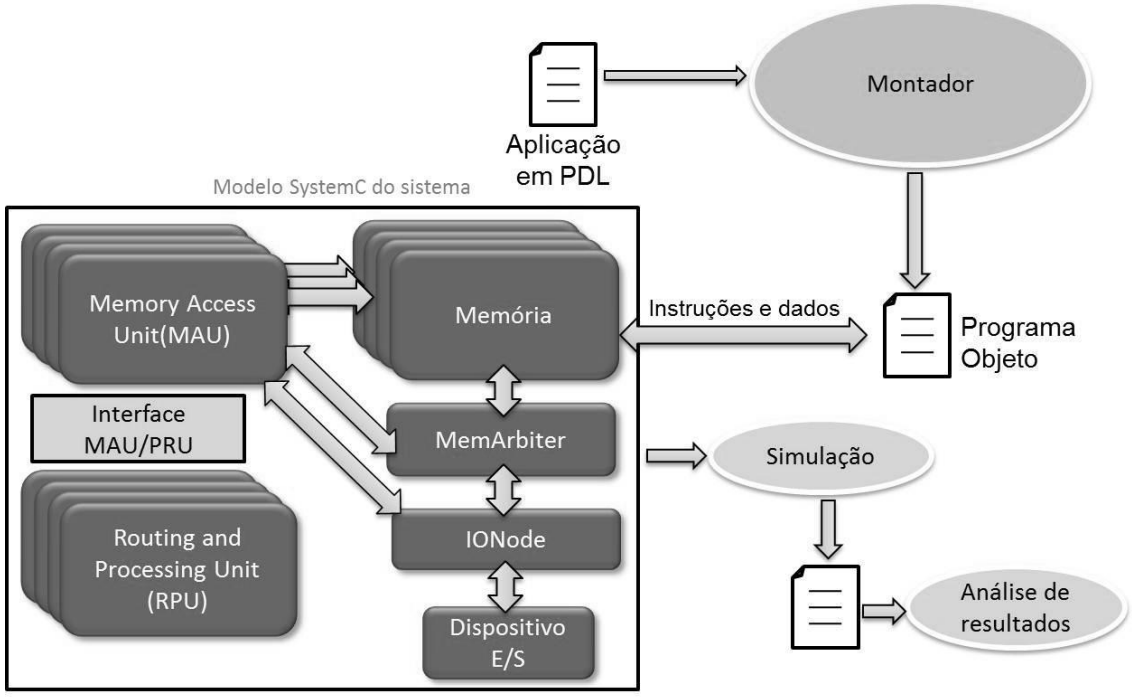


Figura 24: Ambiente de programação e simulação IPNoSys. Fonte: (FERNANDES, 2012).

Após o desenvolvimento de uma aplicação utilizando a linguagem PDL, o arquivo contendo o código fonte deve ser passado como entrada para o montador. Este realiza a análise léxica, sintática e semântica do código fonte, identificando e indicando possíveis erros (FERNANDES, 2012). Outra função do montador é preencher campos do pacote automaticamente com informações que podem ser calculadas a partir do código da aplicação. Por exemplo, o campo NO, que armazena o número de operandos das instruções, é preenchido automaticamente, sem que o programador o informe explicitamente. O campo Resultado_2, quando utilizado para esta mesma função, também é preenchido de forma automática. O montador ainda cria o pacote utilizado para indicar o fim do programa, chamado de pacote zero.

Para iniciar uma simulação, é necessário passar como parâmetro o arquivo contendo o código objeto da aplicação. O simulador, então, lê o arquivo e inicializa as memórias com os dados lidos. Durante a simulação, vários resultados são produzidos, sendo armazenados em um arquivo de texto para avaliação posterior.

4 MPSOC IPNOSYS

A IPNoSys explora o paralelismo das aplicações através da injeção de múltiplos pacotes em sua rede de RPUs. Desse modo, o desempenho da arquitetura, normalmente, está ligado ao número de pacotes regulares que podem ser processados simultaneamente. Os módulos responsáveis por injetar esses pacotes na NoC da IPNoSys são as quatro MAUs, localizadas nos cantos da rede. Esta característica limita a arquitetura a quatro fluxos simultâneos de execução, não sendo possível explorar um grau de paralelismo superior a este.

Este trabalho tem como principal objetivo o desenvolvimento de um sistema que possibilite a execução de mais que quatro pacotes simultaneamente. A estratégia adotada foi a utilização de múltiplas IPNoSys interconectadas trabalhando em conjunto. Desse modo, a quantidade de fluxos de execução paralelos possível é igual a quatro vezes a quantidade de IPNoSys existentes no MPSoC, o que permite a exploração de um maior grau de paralelismo e, consequentemente, obtenção de um sistema de maior desempenho baseado na arquitetura original. As próximas seções apresentam tal sistema em detalhes. A seção 4.1 dá uma visão geral a respeito dessa nova arquitetura. A seção 4.2 mostra as modificações realizadas nos pacotes. Na seção 4.3, as alterações arquiteturais e novos componentes incluídos na IPNoSys são apresentados. A seção 4.4 mostra características sobre a programabilidade no sistema proposto e, na seção 4.5, as rede de interconexão do MPSoC é descrita.

4.1 VISÃO GERAL

O MPSoC IPNoSys é uma plataforma MPSoC homogênea que utiliza a arquitetura IPNoSys como Elemento de Processamento (EP). Cada EP possui seus fluxos internos de execução, como na IPNoSys original, e pode se comunicar com os demais por meio da troca de mensagens. Essa organização cria uma espécie de hierarquia de rede, na qual as redes mais internas, ou locais, são responsáveis pelo processamento das informações e a rede mais externa, ou global, pela comunicação entre as internas.

A rede global é uma NoC, onde cada EP do sistema está conectado diretamente a um roteador e os roteadores, por sua vez, conectados entre si. A Figura 25 apresenta uma configuração desse sistema com quatro EPs. Nela, os roteadores estão representados pelos

elementos identificados pela letra “R”, os EPs aparecem nomeados por “IPNoSys” e os canais de comunicação são as linhas entre os roteadores e os roteadores e EPs.

O par de números que aparece nos EPs e roteadores são os endereços utilizados para identificação desses elementos na rede. Eles são necessários devido à possibilidade de endereçar mensagens para RPUs e MAUs tanto locais quanto remotas, isto é, aquelas que estão em outros EPs. As mensagens com endereço local são processadas dentro da própria IPNoSys, enquanto que as mensagens com endereços diferentes do local são entregues aos roteadores, para que eles as transportem em direção ao seu destino.

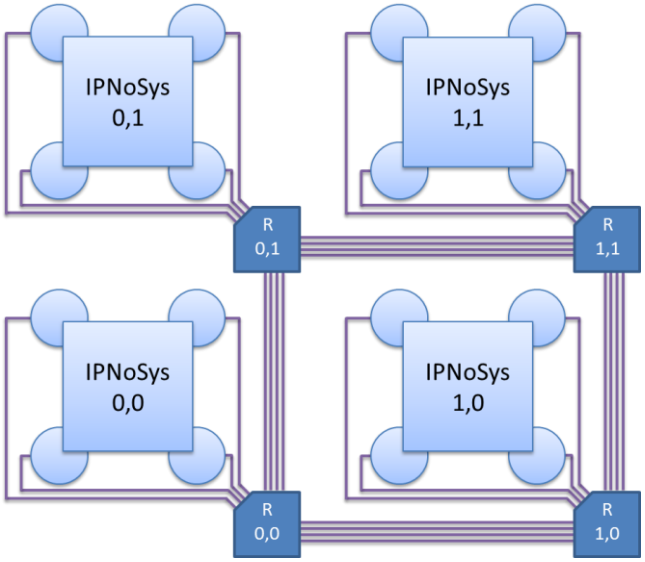


Figura 25: MPSoC IPNoSys 2x2. Fonte: Autoria própria.

As IPNoSys estão conectadas aos roteadores através de suas MAUs, representadas pelos círculos na Figura 25, de modo que toda troca de mensagens acontece através delas. Cada MAU possui um canal *full-duplex* direto com o roteador, possibilitando que todas enviem e recebam mensagens simultaneamente. A transmissão dessas mensagens entre roteadores também pode acontecer de forma paralela, pois há um canal físico exclusivo correspondente a cada MAU. Essa característica cria quatro vias de comunicação independentes entre os EPs, mas obriga que mensagens sejam trocadas sempre entre MAUs de endereços semelhantes. Por exemplo: mensagens enviadas a partir da MAU_1 da IPNoSys 0,0 endereçadas para a IPNoSys 1,0 (ou para qualquer outra) serão recebidas sempre pela MAU_1 deste último EP.

As seções a seguir descrevem as modificações na arquitetura IPNoSys original e apresenta a NoC utilizada como subsistema de interconexão.

4.2 FORMATO DO PACOTE

As mensagens entre EPs são trocadas por meio dos pacotes de controle da IPNoSys Original. O formato dos pacotes necessitou de algumas modificações, a fim de possibilitar seu endereçamento adequado. Dois novos campos foram adicionados à primeira palavra do cabeçalho: “IPN Destino” e “IPN Origem”. O primeiro é responsável por identificar o EP para o qual uma mensagem se destina, enquanto que o segundo armazena o EP que originou dada mensagem. A Figura 26 mostra o novo formato do pacote.

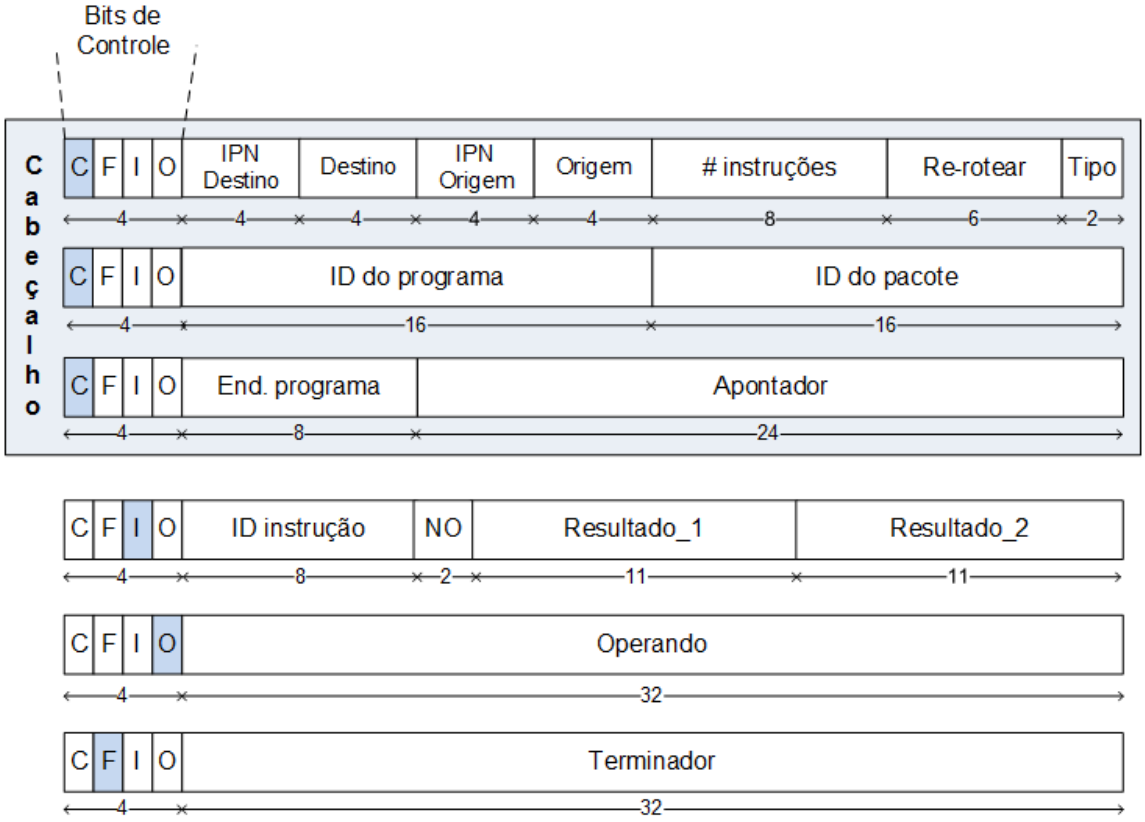
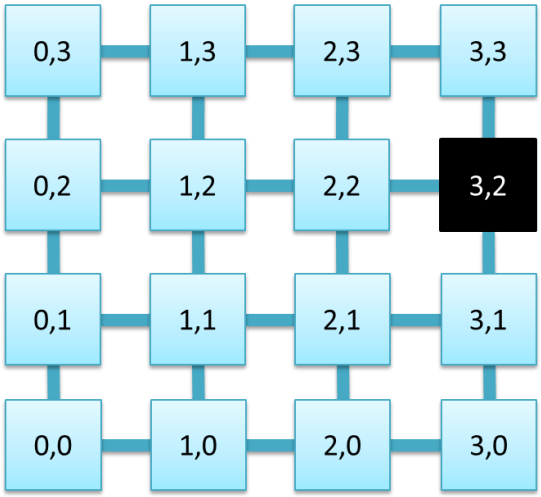


Figura 26: Novo formato do pacote. Fonte: Autoria própria.

A inclusão desses dois novos campos ao cabeçalho foi possível após a redução dos campos “Destino” e “Origem”, existentes no cabeçalho do pacote original. Ambos os campos possuíam oito bits de largura e, na arquitetura proposta neste trabalho, passaram a ter quatro bits cada um. Os quatro bits restantes de cada campo foi utilizado como identificador de EP (“IPN Destino” e “IPN Origem”). Com essas larguras, é possível endereçar até dezesseis EPs, organizados em grelha 2D 4x4, cada um com as dimensões máximas de 4x4 RPUs. Na Figura 26, percebe-se que o campo “IPN Destino” precede o campo “Destino” assim como o “IPN

Origem" precede o "Origem". Dessa forma, os bits mais significativos, do destino ou da origem, são usados para determinar o EP, e os bits menos significativos para especificar uma MAU ou RPU dentro da EP.

O endereço dos EPs, assim como o endereço das RPUs e MAUs, é formado por um par de valores, como descrito na seção anterior. No pacote, cada valor desse par é definido por dois bits, totalizando os quatro bits disponíveis nos campos "IPN Origem" e "IPN Destino". Como exemplo, um pacote endereçado para o EP 1,0 teria, no campo "IPN Destino", o valor binário 0100_b, cujo primeiro par de bits corresponde ao valor 1 e o segundo, ao valor 0 do endereço do EP 1,0. O mesmo valor (0100_b) existiria no campo "Destino" caso o pacote fosse endereçado para a RPU 1,0. Em outro exemplo, um pacote originado pelo EP 3,2 possuiria o valor binário 1110_b no campo "IPN Origem". Nesse caso, os dois bits mais significativos se referem ao valor 3 e os dois menos significativos, ao valor 2 do endereço do EP 3,2. A Figura 27 apresenta os valores referentes a este segundo exemplo resumidos em uma tabela.



EP 3,2	
X	Y
3	2
11_b	10_b
1110_b	

Figura 27: Exemplo de endereçamento de EP. Fonte: Autoria própria.

4.3 MODIFICAÇÕES NA ARQUITETURA IPNOSYS

Além das mudanças no formato do pacote, foram necessárias adaptações nos módulos RPU e MAU da IPNoSys. Todas as operações executadas nesses módulos, que realizam leituras ou alterações nos endereços de origem e destino do pacote, tiveram que ser adaptadas às novas larguras e posicionamentos desses endereços. Essas operações, basicamente, são: criação de pacotes de controle e roteamento de pacotes. Além disso, os novos campos, "IPN

Destino” e “IPN Origem”, tiveram que ser considerados na realização dessas operações. Foi necessário ainda adicionar a capacidade de rotear pacotes às MAUs, pois estas unidades foram utilizadas como interface entre as redes locais e a global do MPSoC.

A criação de pacotes de controle pode acontecer tanto nas RPUs quanto nas MAUs. As instruções nas RPUs que provocam a criação de pacotes de controle são: LOAD, STORE, EXEC, SEND, SYNC e SYNEXEC. Já nas MAUs, a única instrução é a RELOAD. No primeiro grupo, os pacotes criados são endereçados para alguma MAU, especificada juntamente com a instrução. No segundo, a instrução RELOAD é a resposta para uma LOAD, sendo, portanto, endereçada para a origem do pacote LOAD, identificada pelos campos do cabeçalho.

No MPSoC IPNoSys, os endereços especificados junto às instruções podem se referir a qualquer MAU do sistema, incluindo as remotas. É por meio dessas instruções que os EPs se comunicam uns com os outros.

A Figura 28 (a) mostra um exemplo de pacote de controle criados em uma RPU a partir da instrução LOAD. No exemplo, esse pacote foi criado pela RPU 0,2 do EP 0,0, tendo nos campos “Origem” e “IPN Origem”, localizados na palavra 0, os valores 2 (0010_b) e 0 (0000_b), respectivamente. O destino desse pacote é a MAU_3 do EP 1,1, determinados pelos valores 15 (1111_b) e 5 (0101_b) nos campos “Destino” e “IPN Destino”, respectivamente. O campo mais a direita desta mesma palavra, contendo o valor 0, indica que o pacote é de controle. O valor 0, na palavra 2, se refere a posição de memória do valor solicitado, conforme descrito na seção 3.6.

A Figura 28 (b) representa o pacote de controle de resposta, criado na MAU e endereçado à RPU, com a instrução RELOAD. Como pode ser visto, os campos de endereço são os mesmos, porém invertidos, ou seja, o destino deste pacote é a RPU 0,2 do EP 0,0 e a origem é a MAU_3 do EP 1,1. O número 21, na palavra 2, representa o valor existente na posição 0 da memória no momento da execução do LOAD.

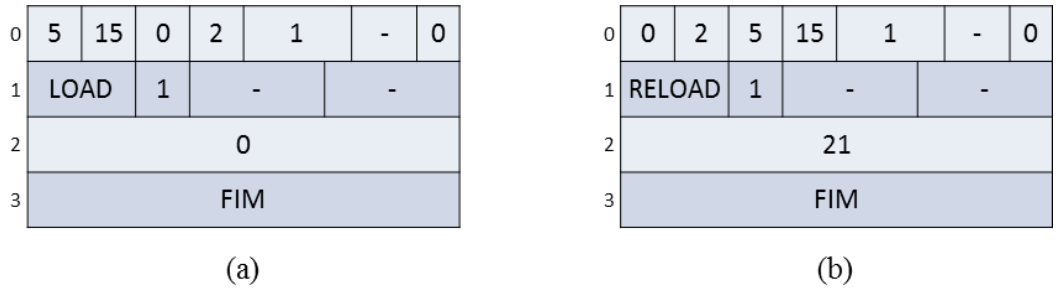


Figura 28: Exemplos de pacotes de controle: (a) LOAD e (b) RELOAD. Fonte: Autoria própria.

O roteamento de pacotes, na IPNoSys original, é feito apenas pelas RPUs. Os pacotes regulares são roteados de acordo com o algoritmo *spiral complement*. O uso desse algoritmo permite que os pacotes se mantenham na rede até que todas as suas instruções sejam executadas. Outra operação, relacionada ao algoritmo *spiral complement*, que também envolve os endereços do cabeçalho, é a redefinição dos endereços de origem e destino sempre que o pacote chega a RPU de destino ainda contendo instruções. Isso permite a continuação da execução das instruções restantes. Os pacotes de controle, por sua vez, são roteados pelas RPUs segundo o algoritmo de roteamento XY tradicional.

No MPSoC IPNoSys, o roteamento de pacotes regulares de acordo com o algoritmo *spiral complement* se manteve o mesmo. A operação de redefinição dos endereços realizada pelo algoritmo considera somente os campos “Destino” e “Origem”, ignorando os valores existentes nos novos campos “IPN Destino” e “IPN Origem”. Foram realizadas alterações apenas na largura e posicionamento dos endereços, conforme dito no início desta seção. O algoritmo não foi alterado, pois, assim como na IPNoSys Original, seu objetivo é de manter o pacote regular na rede de RPUs até que suas instruções sejam completamente executadas. Os pacotes deste tipo nunca são transportados para outros EPs. No momento da operação de redefinição dos endereços de origem e destino pelo algoritmo *spiral complement*, os valores existentes nos novos campos são simplesmente mantidos, permanecendo os mesmo durante toda a existência do pacote regular na rede de RPUs.

Os pacotes de controle, diferentemente dos regulares, podem ser transportados para outros EPs do MPSoC. Os pacotes criados pelas RPUs são roteados, primeiramente, dentro do EP e, posteriormente, na rede de EPs. No primeiro passo, as RPUs consideram apenas o endereço da MAU de destino, desconsiderando o endereço do EP de destino, assim como acontece com os pacotes regulares. Desse modo, o algoritmo de roteamento XY é executado normalmente, levando o pacote da RPU até a MAU determinada pelo campo “Destino”. Ao chegar à MAU, esta verifica se o endereço contido no campo “IPN Destino” corresponde ao do EP atual ou não. Em caso positivo, a instrução é executada normalmente. Em caso negativo, o pacote é entregue ao roteador conectado, para que ele seja transportado ao EP especificado. Neste segundo caso, o pacote atravessa os roteadores da rede, chegando, finalmente, a MAU do EP endereçado pelo pacote. Os detalhes sobre a rede utilizada para interconexão das IPNoSys serão apresentados na seção 4.5. A Figura 29 mostra a rota seguida pelo pacote apresentado na Figura 28 (a).

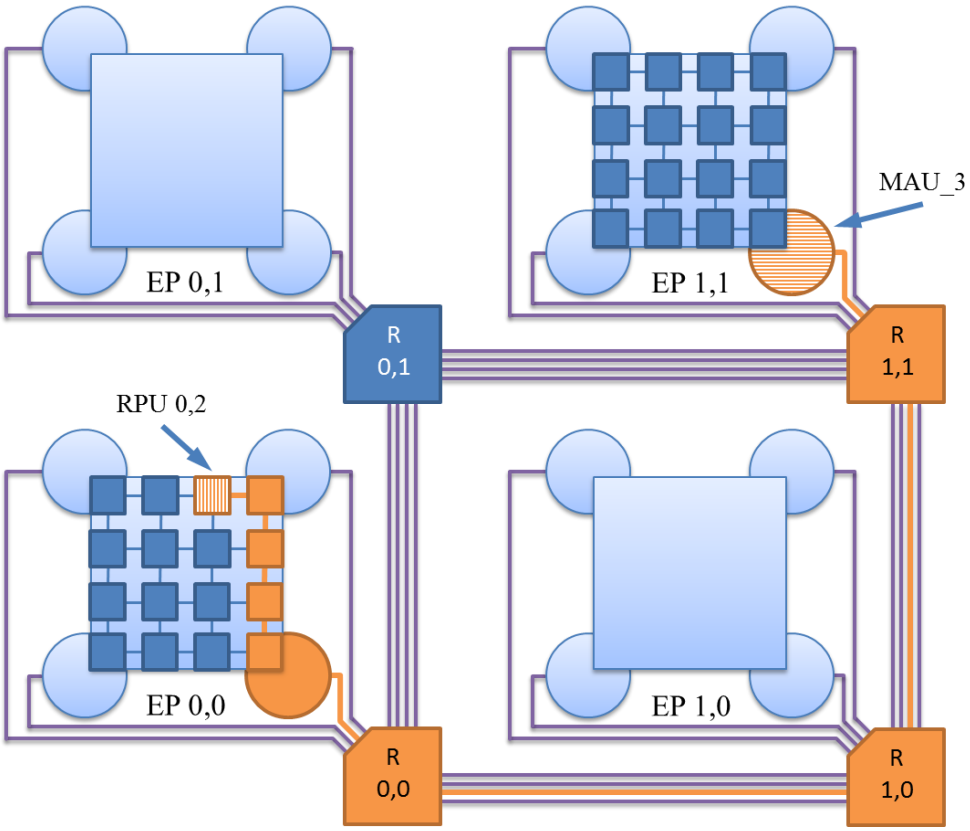


Figura 29: Exemplo de rota de um pacote de controle LOAD. Fonte: Autoria própria.

Na figura, a RPU 0,2, hachurada verticalmente, representa a origem do pacote, enquanto que a MAU_3, hachurada horizontalmente, representa o seu destino. Os outros módulos, destacados em tom mais claro, indicam o caminho entre origem e destino seguido pelo pacote. Como pode ser visto, primeiramente o pacote segue em direção a MAU endereçada pelo campo “Destino”, de acordo com o algoritmo XY. Ao receber o pacote, a MAU constata que ele está endereçado para outro EP e o encaminha para o roteador. A rede, então, o transporta para o EP 1,1, determinado pelo campo “IPN Destino”, onde ele é recebido e tem sua instrução executada pela MAU_3 deste EP.

Seguindo a mesma lógica, porém de maneira contrária, os pacotes de controle produzidos pelas MAUs têm o campo “IPN Destino” considerado no início do roteamento. Primeiramente, o endereço contido neste campo é comparado ao endereço do EP ao qual a MAU pertence. Caso seja igual, o pacote é injetado na própria IPNoSys, sendo entregue à RPU conectada. Caso o endereço seja distinto, o pacote é transferido para o roteador, sendo, em seguida, transportado pela rede em direção ao EP de destino. Chegando ao roteador deste EP, o pacote é entregue a MAU de mesmo endereço daquela que o criou. Essa MAU, então, o injeta na rede de RPUs, por meio da qual ele é levado à RPU de destino, seguindo o algoritmo

de roteamento XY. A Figura 30 mostra o caminho seguido pelo pacote apresentado na Figura 28 (b).

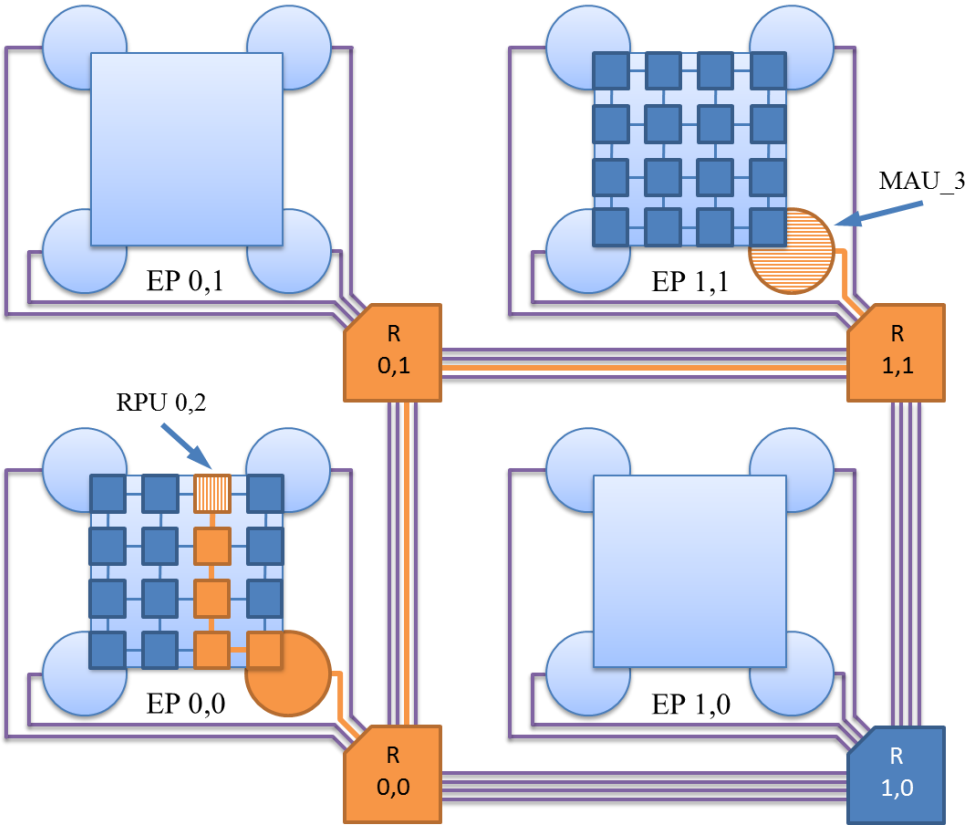


Figura 30: Exemplo de rota de um pacote de controle RELOAD. Fonte: Autoria própria.

A MAU_3 do EP 1,1 hachurada horizontalmente representa o local onde o pacote foi criado, enquanto que a RPU 0,2 do EP 0,0 hachurada verticalmente representa o módulo destinatário do pacote. Os elementos em tom mais claro mostram o caminho seguido pelo pacote. Da mesma forma que o pacote com LOAD, o pacote com RELOAD segue através das redes de acordo com o algoritmo de roteamento XY. Nesse caso, inicialmente através dos roteadores e, posteriormente, através da rede de RPUs.

4.3.1 Detalhamento da MAU

No MPSoC IPNoSys, as MAUs são utilizadas para comunicação do EP com a rede de interconexão, ou seja, ela funciona como uma ponte entre a rede interna de RPUs e a rede externa de roteadores. Para que essa nova funcionalidade fosse possível, sua arquitetura sofreu algumas alterações. A nova arquitetura da MAU é apresentada na Figura 31 a seguir.

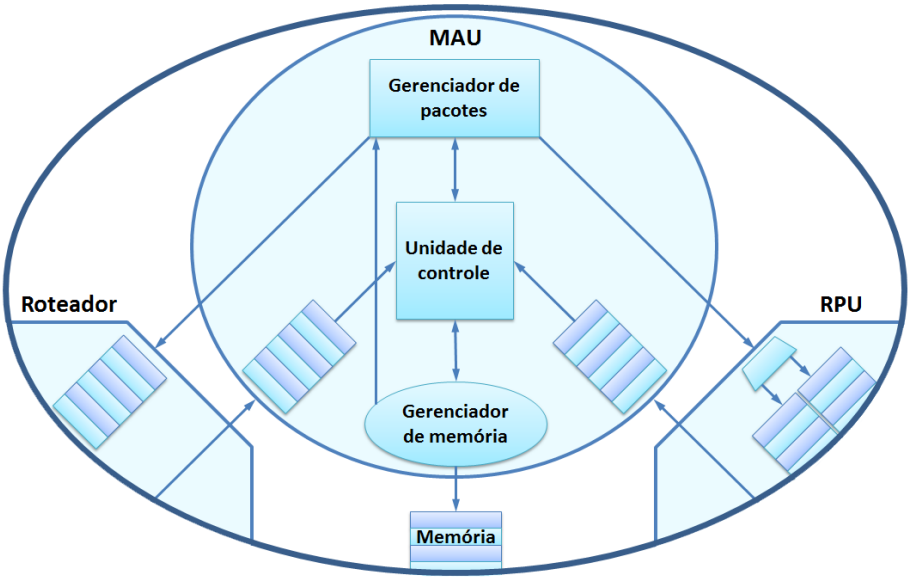


Figura 31: Arquitetura da MAU no MPSoC. Fonte: Autoria própria.

Como pode ser visto, as MAUs receberam um novo conjunto de portas de entrada e saída e um *buffer* de entrada. Além disso, o módulo Gerenciador de Pacotes recebeu a função de roteador, isto é, na nova arquitetura, esse módulo é responsável por determinar se o pacote deve ser repassado ou não da rede interna para a externa ou vice-versa. O conjunto de portas e a técnica de memorização são os mesmos utilizados nos roteadores da NoC. Por esse motivo, esses dois pontos serão detalhados na seção 4.5.1, onde são apresentados os roteadores do MPSoC.

Quando o pacote chega à MAU por meio da RPU, o Gerenciador de Pacotes verifica o endereço contido no campo “IPN Destino” antes de desempacotar a instrução contida no pacote de controle. Caso não corresponda ao endereço do EP em que se encontra, ele apenas repassa o pacote para o *buffer* de entrada do roteador, o injetando na rede externa. Caso corresponda ao endereço do EP em que se encontra, a instrução é executada normalmente. Por outro lado, quando o pacote de controle é entregue à MAU pela rede externa, ou seja, vindo do roteador conectado, essa verificação do campo “IPN Destino” não é realizada, pois os roteadores sempre entregam os pacotes no EP de destino correto. No lugar disso, o Gerenciador de Pacotes verifica qual a instrução contida nesse pacote. Se for uma RELOAD, ele repassa o pacote de controle para a rede de RPUs, pois essa instrução é executada em uma RPU. Caso contrário, o Gerenciador de Pacotes a executa normalmente. A Figura 32 resume em fluxogramas esses novos processos adicionados ao Gerenciador de Pacotes original. A Figura 32 (a) se refere ao processo do pacote vindo da RPU e a Figura 32 (b), ao processo do pacote vindo do roteador.

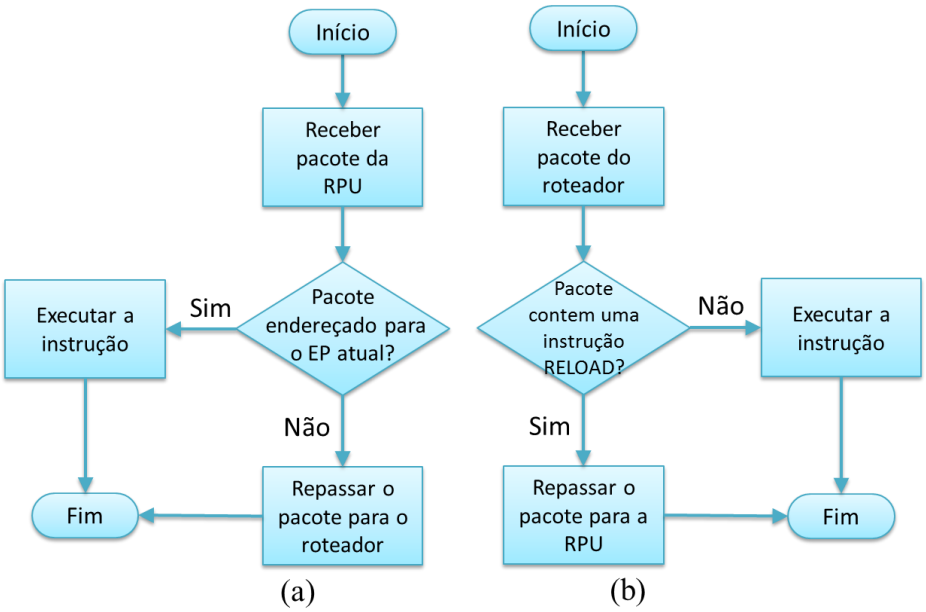


Figura 32: Fluxograma dos novos processos do Gerenciador de Pacotes. Fonte: Autoria própria.

Os *buffers* de entrada das MAUs são tratados pelo Gerenciador de Pacotes de forma alternada. A cada alternância, um pacote armazenado é completamente processado, seguindo a ordem de chegada ao *buffer*. Adotou-se uma estratégia sequencial deste gerenciador optando por simplicidade arquitetural. Entretanto, o mesmo poderia ter sido implementado com capacidade de processar pacotes de ambos os *buffers* em paralelo.

A Unidade de Controle foi modificada para permitir o recebimento de pacotes de ambas as redes de forma simultânea. Por fim, o Gerenciador de Memória não sofreu alterações.

4.4 PROGRAMABILIDADE

O conjunto de instruções do MPSoC IPNoSys se manteve o mesmo existente na IPNoSys Original, conforme mostrado na Tabela 1. Isso foi possível porque, na nova arquitetura, a comunicação entre EPs acontece por meio do endereçamento de pacotes para MAUs remotas, não necessitando de instruções especiais para esse fim. Apesar disso, foram necessárias modificações na forma como é feita referência às MAUs na linguagem PDL. Essas modificações afetaram as instruções de acesso à memória e sincronização, já citadas em seções anteriores: LOAD, STORE, EXEC, SYNEXEC, SYNC, RELOAD e SEND.

Todas essas instruções, com exceção da RELOAD, são executadas por MAUs. Na linguagem PDL, é de responsabilidade do programador a declaração explicita do endereço de MAU onde as instruções devem ser executadas. Quando uma RPU encontra alguma dessas instruções, durante o processamento dos pacotes regulares, ela empacota tal instrução em um pacote de controle e o envia em direção à MAU especificada na instrução. Na IPNoSys original, a referência à MAU é feita por meio das palavras reservadas: MAU_0, MAU_1, MAU_2 e MAU_3. A Tabela 2 mostra alguns exemplos de instruções de acesso à memória e sincronização na linguagem PDL original e suas respectivas descrições.

Tabela 2: Exemplos de comandos PDL de acesso à memória e sincronização da IPNoSys Original. Fonte: Autoria própria.

Instrução	Descrição
EXEC MAU_2; prog_0, pac_2;	Solicita que a MAU_2 injete o pacote regular identificado por pac_2, do programa 0, na rede de RPUs para sua execução.
LOAD MAU_3 var1; m0;	Solicita o carregamento do valor armazenado na posição de memória m0 da MAU_3 para a variável var1.
STORE MAU_0; var2 m1;	Solicita o armazenamento do valor contido na variável var2 na posição m1 da memória da MAU_0.

Na linguagem PDL modificada para o MPSoC IPNoSys, a referência às MAUs precisou de um novo identificador, utilizado para definir em que EP uma dada MAU está. Desse modo, o endereço das MAUs é formado pelo par MAU, EP. O primeiro elemento é semelhante ao da PDL original, ou seja, através das palavras reservadas, conforme discutido no parágrafo anterior. O segundo elemento, o identificador do EP, corresponde ao termo "EP" seguido dos valores relativos ao endereço do EP separados por *underline* (_). Por exemplo, caso se deseje fazer referência à MAU 0 do EP 3,1, o termo utilizado seria "MAU_0, EP_3_1". A Tabela 3 a seguir apresenta as mesmas instruções da Tabela 2, mas na nova PDL para o MPSoC IPNoSys.

Tabela 3: Exemplo de instruções na nova PDL. Fonte: Autoria própria.

Instrução	Descrição
EXEC MAU_2, EP_0_1; prog_0, pac_2;	Solicita que a MAU_2 do EP 0,1 injete o pacote regular identificado por pac_2, do programa 0, na rede de RPUs para a execução.

LOAD MAU_3, EP_1_1 var1; m0;	Solicita o carregamento do valor armazenado na posição de memória m0 da MAU_3 do EP 1,1 para a variável var1.
STORE MAU_0, EP_3_2; var2 m1;	Solicita o armazenamento do valor contido na variável var2 na posição m1 da MAU_0 do EP 3,2.

Na linguagem PDL, o programador também é responsável por determinar explicitamente em que módulo de memória cada pacote vai estar durante a execução da aplicação. Essa característica permite que os múltiplos pacotes que compõem uma aplicação sejam distribuídos entre várias MAUs, da forma que for conveniente ao programador. Quanto melhor estiver distribuída a aplicação, maior será o paralelismo na execução. A localização de cada pacote é definida em sua declaração e, tanto a PDL original quanto a modificada, seguem o mesmo formato da referência a MAU nas instruções: na PDL original, pelo identificador de MAU e, na nova PDL, pelo par MAU, EP. A Tabela 4 compara a declaração dos pacotes nas duas versões da PDL.

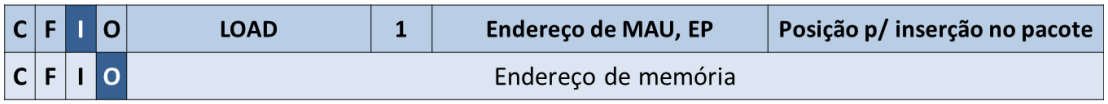
Tabela 4: Declaração dos pacotes na PDL original e modificada. Fonte: Autoria própria.

PDL Original	PDL Modificada
PACKAGE pac_1 ADDRESS MAU_0 . . (Instruções) . END	PACKAGE pac_1 ADDRESS MAU_0, EP_1_3 . . (Instruções) . END

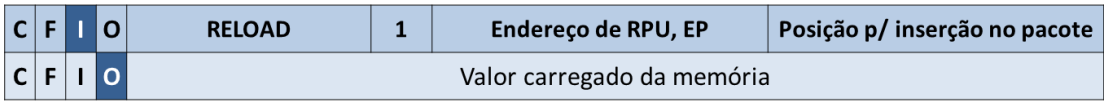
A Tabela 4 apresenta, em ambos os casos, um pacote denominado pac_1. Pode-se perceber que, no primeiro caso, é feita a declaração apenas de qual MAU (MAU_0) o pacote vai estar localizado. No segundo caso, além da MAU, é definido o EP em que essa MAU está, neste caso, a MAU_0 é do EP 1,3 (EP_1_3).

Dentro do pacote, essas instruções se mantiveram com o mesmo formato ao daquele apresentado na Figura 18, porém, o campo Resultado_1 teve seu conteúdo alterado. Com exceção da instrução RELOAD, todas as instruções discutidas na seção 3.6 possuem, originalmente, o endereço da MAU onde deve ser executada no campo Resultado_1. Na nova PDL, a localização de uma MAU é feita a partir dos identificadores de MAU e EP. Portanto, o campo Resultado_1, no MPSoC IPNoSys, armazena esses dois valores, ambos com quatro

bits de largura. Do mesmo modo, na instrução RELOAD, esse campo armazena quatro bits referentes ao endereço da RPU e quatro referentes ao endereço do EP. A Figura 33 mostra exemplos do formato da instrução na PDL modificada. As mudanças, em relação ao formato original (Figura 19), aparecem no campo Resultado_1. Apesar de a figura mostrar apenas a instrução LOAD e RELOAD, as demais seguem o mesmo princípio, com conteúdo semelhante no campo Resultado_1.



(a)



(b)

Figura 33: Instruções (a) LOAD e (b) RELOAD na nova PDL. Fonte: Autoria própria.

Todas as modificações apresentadas na seção 4.2 e nesta tiveram impacto direto no montador da linguagem PDL. Este sofreu alterações para que se adequasse ao novo formato dos pacotes, ao novo conteúdo do campo Resultado_1 e à nova forma de referenciar as MAUs e RPUs no código PDL, tanto nas instruções de acesso à memória e sincronização quanto na declaração de pacotes.

4.5 REDE EM CHIP

A plataforma de interconexão utilizada para permitir a troca de informações entre os EPs do MPSoC IPNoSys é uma Rede em Chip. A razão da escolha desta tecnologia foi em virtude das várias vantagens oferecidas por ela aos sistemas MPSoC, como apresentado na seção 2.3.1.7 deste trabalho.

Tal NoC consiste em uma grelha de roteadores ligados, cada um, a um EP do MPSoC. Como discutido na seção 4.2, o endereço de cada roteador é determinado por um par de valores. Esse endereço é definido com base na posição que o EP ocupa na rede, sendo o primeiro valor correspondente à coordenada X e o último, à coordenada Y. O tamanho máximo da rede está limitado a uma grelha 4x4, devido à largura e organização dos campos “IPN Destino” e “IPN Origem”. Ambos os campos possuem largura de quatro bits, cujos dois

primeiros endereçam a coordenada X e os dois últimos, a coordenada Y. Portanto, X e Y podem assumir valores somente entre zero e três cuja representação é possível com dois bits. Configurações de redes menores podem ser utilizadas, inclusive não quadradas. A Figura 34 a seguir apresenta algumas configurações de grelhas possíveis: (a) 4x4, (b) 2x3, (c) 1x3 e (d) 4x3.

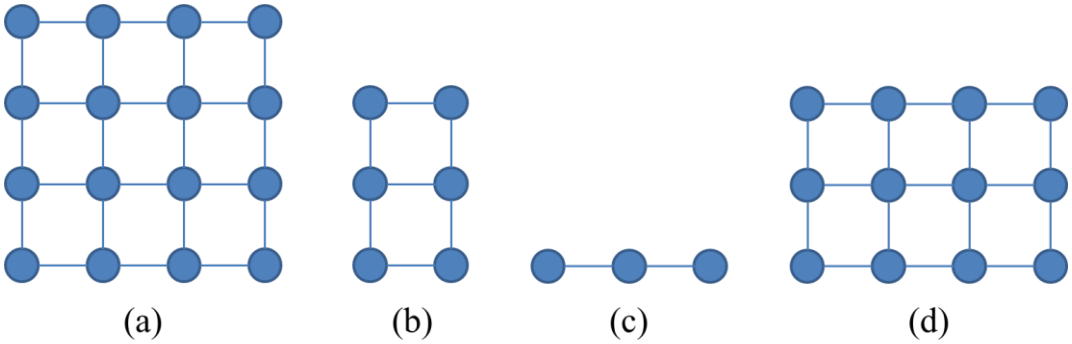


Figura 34: Possíveis configurações da NoC. Fonte: Autoria própria.

As mensagens trocadas entre roteadores são os mesmos pacotes que circulam pela NoC interna de cada IPNoSys, sem qualquer encapsulamento adicional. Eles utilizam, inclusive, os bits de controle existentes para identificação do fim do pacote.

Cada roteador pode possuir de três a cinco conjuntos de portas, dependendo de sua localização na rede: Norte, Sul, Leste, Oeste e Local. Os primeiros quatro conjuntos são utilizados para a conexão entre roteadores, enquanto que o último é para a conexão com o EP associado. Analisando a Figura 34 (a), pode-se perceber que os roteadores das bordas, com exceção dos cantos, possuem quatro conjuntos de portas (incluindo o conjunto de portas Local, não visível na figura). Os roteadores dos cantos da rede, por sua vez, possuem apenas três. Cada roteador está associado a um EP, possuindo sempre o conjunto Local.

O roteamento é definido pelo algoritmo determinístico XY, assim como acontece nas RPUs da IPNoSys. As razões para uso desse algoritmo são sua simplicidade e ausência de *deadlocks*.

A troca de dados entre EPs acontece por meio de pacotes de controle, ou seja, pacotes curtos que contém apenas uma palavra de cabeçalho, uma de instrução, algumas de operandos (em média, duas) e uma de fim de pacote. Para esse tipo de pacote, o chaveamento por circuito não é o mais adequado, como explicado na seção 2.3.1.3. Entre os chaveamentos por pacote, aquele que geralmente apresenta melhor desempenho é o *wormhole*. Por esse motivo, este foi o chaveamento adotado.

Os pacotes de controle que circulam dentro dos EPs ou entre eles são, geralmente, resultantes de instruções anteriormente contidas em pacotes regulares. Ao produzir um pacote de controle, a instrução correspondente é eliminada do pacote regular, sem necessidade de qualquer confirmação de que o pacote de controle chegou ao seu destino. Por esse motivo, em hipótese nenhuma, pacotes de controle podem ser perdidos, pois isso afetaria a correta execução dos programas. Levando isso em consideração, o controle de fluxo utilizado é o baseado em créditos, que não permite o descarte de pacotes.

A próxima subseção dá uma visão mais detalhada dos roteadores, apresentando ainda características como arbitragem e memorização.

4.5.1 Roteadores

Os roteadores usados no MPSoC IPNoSys foram desenvolvidos com o objetivo de permitir o tráfego de dados a partir das quatro MAUs dos EPs de forma simultânea. Devido a possibilidade de haver quatro fluxos de execução paralelos na IPNoSys, optou-se por essa estratégia como forma de evitar possíveis gargalos na comunicação entre EPs. Além disso, as RPUs não tomam conhecimento da existência de outros EPs, uma vez que sua comunicação sempre acontece com as MAUs locais, que por sua vez, se comunicam, quando necessário, com MAUs de mesmo endereço de um outro EP. Para isso, foi preciso implementar roteadores com quatro vias de comunicação com os EPs e entre si. O roteador se tornou, portanto, o encapsulamento de quatro roteadores semelhantes, um para cada MAU do EP. Cada roteador interno é isolado dos demais, de forma que um pacote entregue a um deles nunca é repassado para roteadores de outras vias. A Figura 35 apresenta a visão interna do roteador do MPSoC. O roteador interno R0 está conectado à MAU_0, o roteador R1 está conectado à MAU_2, o R2 está conectado à MAU_3 e o R3, à MAU_1.

A escolha por utilizar vias de comunicação independentes e isoladas entre MAUs foi em virtude da simplicidade tanto dos roteadores quanto do roteamento de pacotes nas RPUs. Em relação ao primeiro ponto, caso os roteadores pudessem se comunicar entre si, mais canais entre eles seriam necessários, o que os tornaria mais complexos. Em relação ao segundo, o roteamento de pacotes nas RPUs se manteve o mesmo, sem qualquer operação extra que prejudique o desempenho da comunicação interna da IPNoSys. Assim, a existência dos outros EPs é transparente às RPUs, sendo conhecida apenas pelas MAUs.

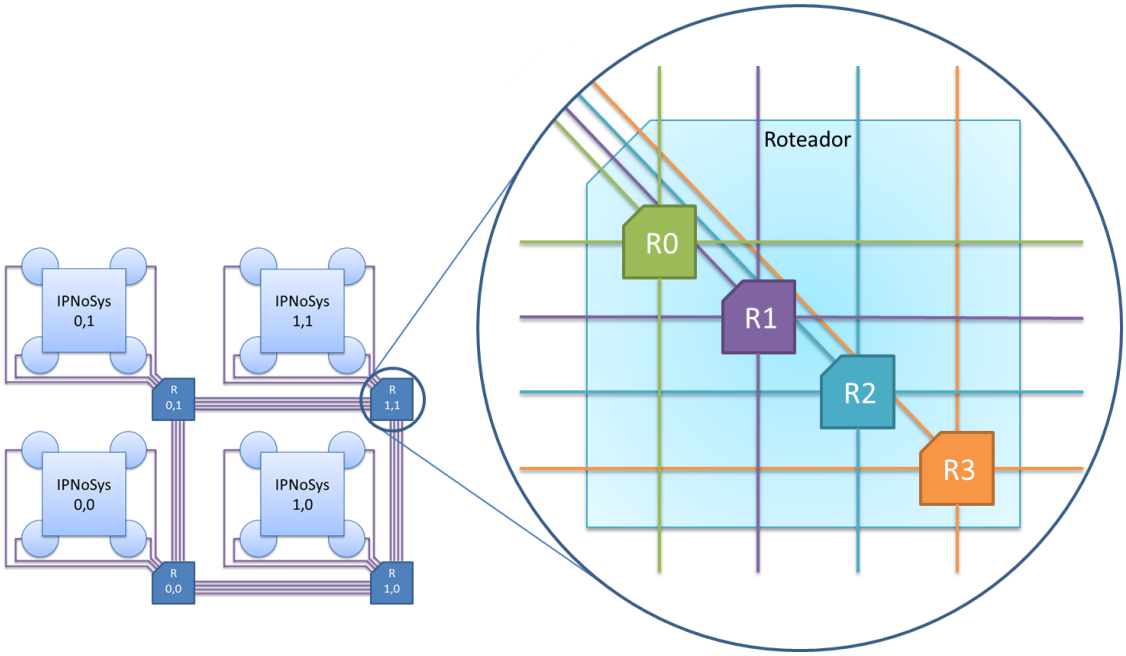


Figura 35: Visão interna do roteador. Fonte: Autoria própria.

Os roteadores internos também podem possuir até cinco conjuntos de portas, que seguem o mesmo raciocínio apresentado na seção anterior. Cada conjunto possui um canal de entrada, um de saída e seus respectivos sinais para controle de fluxo, possibilitando comunicação *full-duplex*, ou seja, com envio e recebimento simultâneos de dados. A Figura 36 demonstra o conjunto Leste de portas existente em um roteador interno. Essas mesmas portas existem nos demais conjuntos de cada roteador interno. Elas são também as mesmas existentes nas MAUs para a comunicação com roteadores, como apresentado na seção 4.3.1.

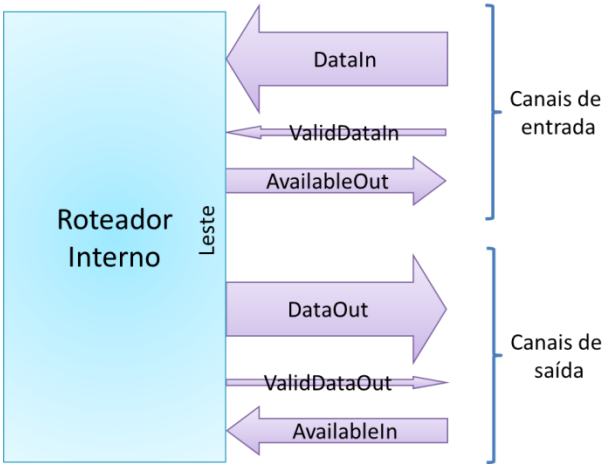


Figura 36: Conjunto de portas Leste do roteador interno. Fonte: Autoria própria.

As seta denominadas *DataOut* e *DataIn* representam as portas por onde as palavras dos pacotes são transmitidas, tendo largura de 36 bits. A primeira porta é para saída de dados e a segunda, para entrada. A porta representada pela seta nomeada *ValidDataOut* informa ao

roteador adjacente quando há uma palavra válida na porta *DataOut*. De forma contrária, a porta *ValidDataIn* é usada pelo roteador para ser informado quando há uma palavra de dado válida na sua porta de entrada *DataIn*. Essas duas portas possuem largura de apenas um bit. A porta *AvailableOut* informa ao vizinho o espaço disponível no *buffer* de entrada do roteador, ou seja, quantas palavras de dados ainda podem ser recebidas. Já a porta *AvaliableIn* é usada pelo roteador para identificar quanto espaço disponível há no *buffer* de entrada do seu vizinho, isto é, quantas palavras podem ser enviadas. A largura dessas portas depende do tamanho do espaço disponível para armazenamento. Na rede utilizada, o buffer possui espaço para dez palavras, exigindo um canal com largura de, no mínimo, quatro bits. O controle de fluxo é feito a partir dessas duas últimas portas apresentadas, implementando a técnica de controle de fluxo baseado em créditos. A Figura 37 mostra o fluxograma referente ao funcionamento do canal de entrada e saída dos roteadores durante o recebimento (a) e transmissão (b) de dados.

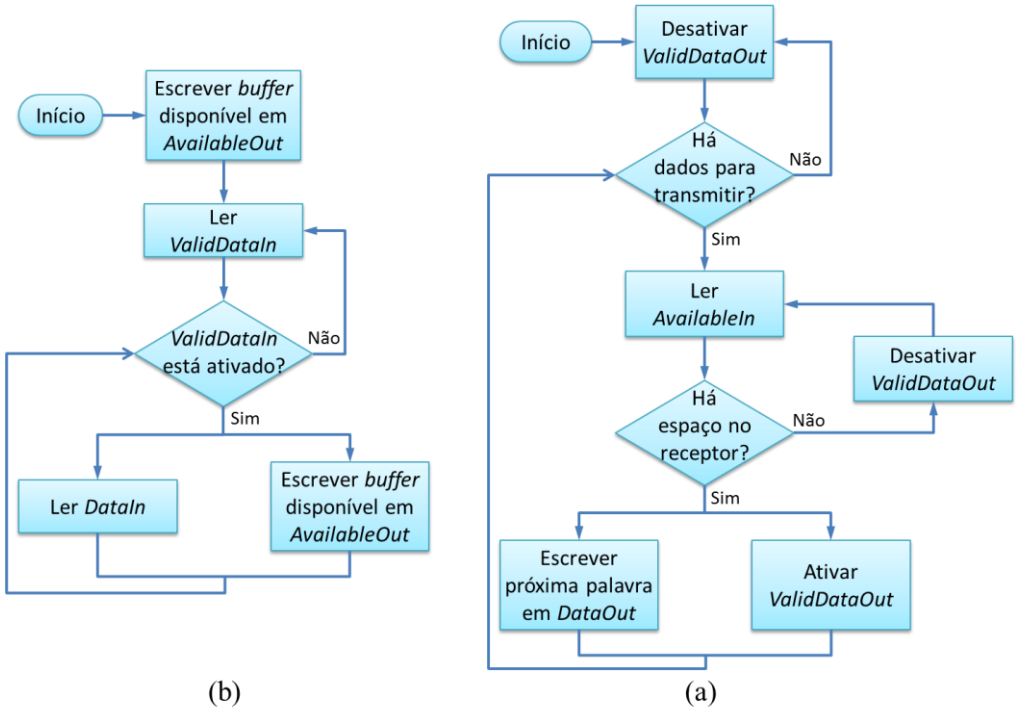


Figura 37: Funcionamento dos canais de entrada (a) e saída (b). Fonte: Autoria própria.

No recebimento de dados, representado pelo fluxograma da Figura 37 (a), o primeiro passo é informar ao roteador adjacente, através do sinal *AvailableOut*, a quantidade de espaços disponível no *buffer* de entrada. Inicialmente, esse valor corresponde ao tamanho total do *buffer*. Depois disso, o sinal *ValidDataIn* é verificado continuamente. Quando este sinal é ativado, significa que existe uma palavra de dado válida em *DataIn*. Então, esta palavra é lida, armazenada em *buffer* e o valor de *AvailableOut* é atualizado, sendo decrementado em uma unidade. O valor de *AvailableOut* também é atualizado à medida que o

pacote vai sendo retirado do *buffer* de entrada da MAU durante seu processamento. Esse ciclo é repetido continuamente durante todo o funcionamento do sistema.

Na operação de transmissão de dados, mostrada na Figura 37 (b), o sinal *ValidDataOut* é, inicialmente, desativado, indicando que não há nenhum dado sendo transmitido. Esta situação é mantida até que se deseje transmitir algum pacote. Quando isso ocorre, o sinal *AvailableIn* é, primeiramente, lido, a fim de verificar se o roteador adjacente possui espaço disponível em seu *buffer* de entrada. Caso não haja, a verificação continua até que algum espaço seja liberado. Quando houver pelo menos um espaço disponível, a palavra é colocada na saída *DataOut* e *ValidDataOut* é ativado, informando ao receptor que há uma palavra sendo transmitida. Esse ciclo se repete até que todo o pacote seja transmitido e se inicia sempre que houver pacotes para a transmissão.

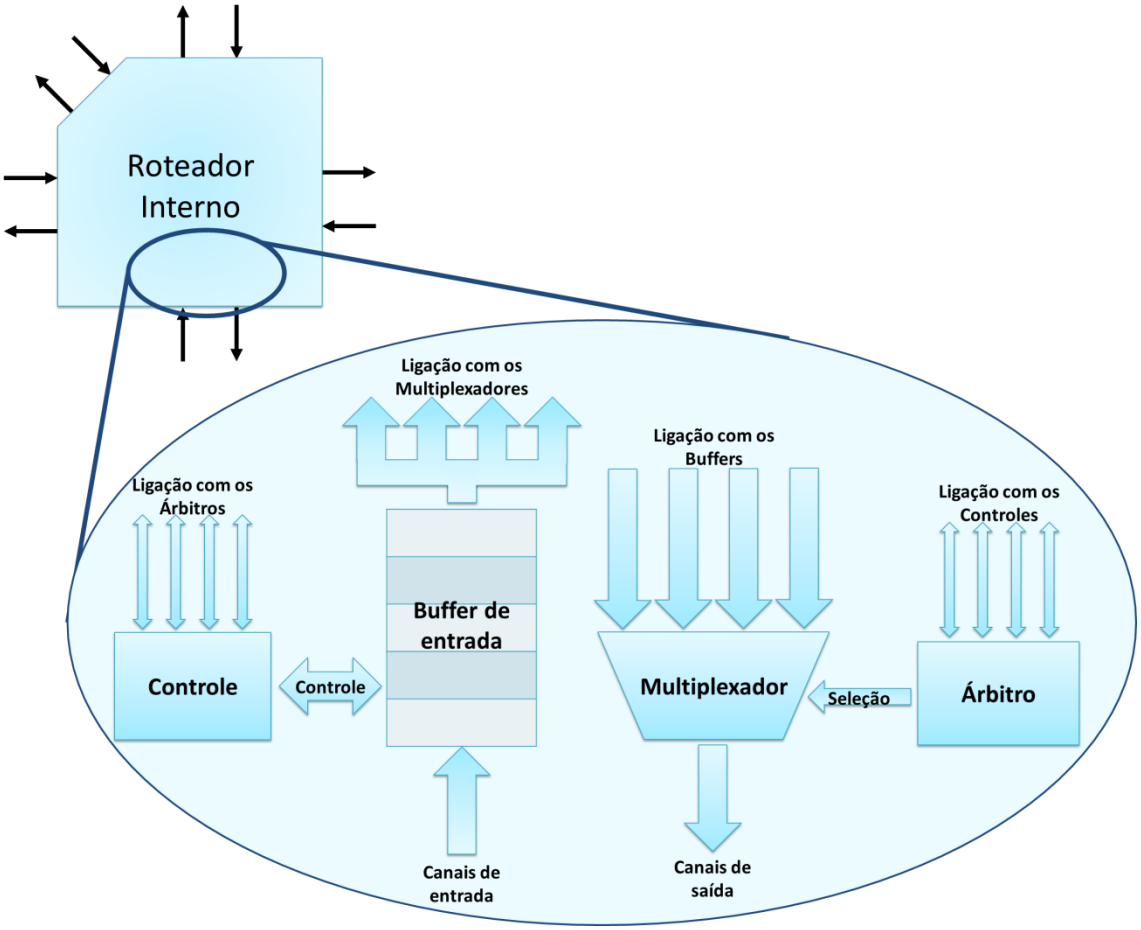


Figura 38: Módulos e sinais. Fonte: Autoria própria.

Os roteadores internos são simples, compostos por poucos módulos. A Figura 38 destaca, resumidamente, os módulos e sinais internos existentes em cada conjunto de portas desses roteadores. As legendas próximas às setas descrevem quais outros módulos, dos demais conjuntos de portas, estão ligados a um determinado módulo. Cada seta representa a

ligação entre o respectivo módulo e um de outro conjunto de portas. Por exemplo: considerando que os módulos da figura correspondem à porta Sul, cada seta do módulo Controle indica que ele está conectado ao Árbitro das portas Oeste, Local, Norte e Leste. Esta lógica serve para qualquer módulo da figura. A ligação entre Buffer de entrada e Multiplexador possui representação distinta porque cada buffer possui uma única saída conectada a entrada de todos os multiplexadores pelo mesmo sinal.

Como pode ser visto na Figura 38, os *buffers* estão localizados junto aos canais de entrada, o que caracteriza a técnica de memorização na entrada. Eles são do tipo FIFO (*First In First Out*). Nesse tipo de *buffer*, os pacotes armazenados temporariamente são tratados na ordem em que chegam.

A arbitragem é do tipo descentralizada com política de FCFS. Dessa maneira, há um árbitro associado a cada porta de saída do roteador, responsável por decidir qual *buffer* de entrada pode dispor de sua porta de saída em dado momento. A decisão dos árbitros é baseada na ordem em que as requisições são recebidas. Uma vez dada autorização de uso, o *buffer* terá a porta de saída a sua disposição até que todo o pacote referente à requisição tenha sido transmitido. Esse é um ponto negativo existente em chaveamentos *wormhole*.

A ligação entre buffer de entrada e porta de saída é feita através de multiplexadores. Existe um total de cinco multiplexadores por roteador, estando cada um em uma de suas portas de saída. Cada multiplexador recebe, como entrada, a saída de todos os *buffers* de entrada e um sinal do árbitro, usado para determinar qual dos cinco *buffers* deve ser ligado à porta de saída do roteador em dado momento.

Junto a cada *buffer* de entrada, há uma unidade, chamada de Controle, responsável por tratar os pacotes que chegam. Essa unidade, ao perceber que existe algum dado no seu *buffer*, lê a primeira palavra armazenada, que equivale ao cabeçalho do pacote, identifica por qual porta de saída este pacote deve ser encaminhado, por meio algoritmo de roteamento XY, e solicita tal porta ao árbitro correspondente. Ao receber a permissão do árbitro, a unidade de controle sinaliza para seu buffer que a transmissão pode ser efetuada. Enquanto o *buffer* de entrada transmite os dados pela saída do roteador, o Controle verifica se o fim do pacote foi transmitido. Quando isso acontece, a transmissão é finalizada e o Controle sinaliza isto ao árbitro. O árbitro, então, pode passar a vez para o próximo *buffer* que tenha solicitado sua porta de saída, caso exista. A unidade de controle permanece ociosa até que outro pacote chegue ao *buffer* e o ciclo recomece.

5 EXPERIMENTOS E RESULTADOS

Esta seção apresenta alguns experimentos, e seus respectivos resultados, desenvolvidos com o objetivo de avaliar o desempenho da arquitetura proposta neste trabalho. Os resultados foram obtidos pela execução de cada experimento no simulador do MPSoC IPNoSys, desenvolvido em SystemC com precisão de ciclos a partir do simulador da IPNoSys Original. O MPSoC utilizado nos testes foi configurado com uma rede de EPs de tamanho 2x2, totalizando quatro IPNoSys conectadas.

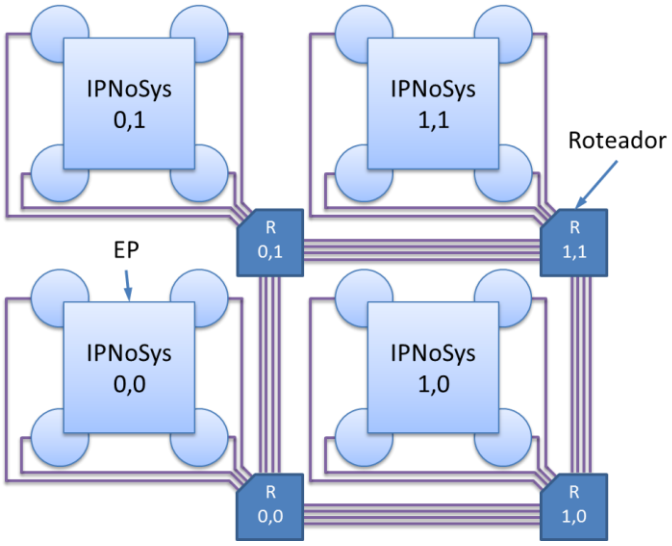


Figura 39: Configuração do MPSoC utilizado nos experimentos. Fonte: Autoria própria.

Cada experimento teve uma versão desenvolvida também para a IPNoSys Original, com resultados obtidos pelo simulador desenvolvido por Fernandes (2012), também em SystemC com precisão de ciclos. Apesar de algumas aplicações serem semelhantes às utilizadas por Fernandes (2012), a implementação destas não é a mesma. Isso explica possíveis discordâncias entre os resultados apresentados por Fernandes (2012) e os mostrados nas próximas seções. A avaliação do MPSoC foi feita a partir da comparação de seus resultados aos obtidos pelo simulador da arquitetura original.

Todos os experimentos realizados na IPNoSys Original possuem quatro fluxos de execução, pois este é o máximo de paralelismo que tal arquitetura suporta. Já os experimentos utilizados no MPSoC IPNoSys possuem dezesseis fluxos simultâneos, que corresponde também a quantidade máxima de fluxos simultâneos na configuração 2x2. Apesar de serem utilizadas aplicações individuais com alta capacidade de paralelização, o comportamento dos sistemas pode ser comparado ao dos mesmos executando múltiplas aplicações com baixa

capacidade de paralelização, algo que normalmente acontece em sistemas reais. Em cada versão dos experimentos, a quantidade de dados processados foi variada e, mesmo com diferentes quantidades de fluxos de execução, a quantidade de instruções e dados em cada comparação entre os sistemas é sempre a mesma.

O EP 0,0 do MPSoC IPNoSys foi utilizado como ponto de partida para a execução dos pacotes em todos os experimentos. Isso significa que os pacotes que iniciam a aplicação e os dados necessários para o processamento estão localizados nas memórias deste EP. Os dados resultantes do processamento também são todos armazenados na memória do EP 0,0, exceto no Experimento 1, onde há um caso em que o armazenamento também foi feito no EP 1,1.

Todos os experimentos foram implementados na linguagem PDL. Aqueles desenvolvidos para a arquitetura original utilizam a linguagem PDL original, enquanto que os experimentos para o MPSoC utiliza a PDL modificada, apresentada na seção 4.4. O código objeto utilizado nos simuladores são obtidos pelos respectivos montadores, compatíveis com cada sistema e linguagem.

Buscou-se produzir aplicações com características distintas, que explorassem diferentes tipos de instruções da IPNoSys. As seções seguintes apresentam seis aplicações, sendo elas: cópia de dados na memória, soma de valores na memoria, contador, multiplicação de matrizes, descompressão RLE e DCT-2D. Em cada seção, é feita uma breve descrição sobre o experimento, seu propósito, a organização de sua implementação e os resultados das simulações em ambos os sistemas. Como resultados, são apresentados, quando relevantes, a quantidade de ciclos de execução, a memória necessária, o uso da rede externa, a energia consumida pelo sistema e o tempo médio gasto com carregamento de dados (LOADs).

As medidas de consumo de energia apresentadas nas seções seguintes equivalem a energia consumida pelos buffers de entrada das RPUs, das MAUs e dos roteadores no MPSoC IPNoSys e das RPUs e MAUs na IPNoSys Original. Os procedimentos para cálculo desses resultados foram os mesmos adotados em (FERNANDES, 2012). Para maiores detalhes, tal trabalho deve ser consultado.

5.1 CÓPIA DE DADOS NA MEMÓRIA

O primeiro experimento, apresentado nesta seção, consiste em uma aplicação simples para copiar valores armazenados na memória de uma região para outra. Ela avalia o

desempenho da arquitetura executando, exclusivamente, operações de acesso à memória e sincronização, não possuindo qualquer outro tipo de instrução.

A aplicação é composta por um pacote iniciador, vários pacotes copiadores, que realizam a operação de copia de fato, e um pacote finalizador. O primeiro pacote contém múltiplas instruções EXEC, uma para cada pacote copiador, e uma SYNEXEC, para injeção do pacote finalizador após o processamento dos demais. Os pacotes copiadores possuem várias instruções LOAD e STORE, responsáveis por carregar valores armazenados da memória e os armazenar novamente em outro local, respectivamente. Devido ao limite de 256 instruções por pacote regular, a quantidade desse tipo de pacote depende do número de instruções da aplicação. Para cada valor inteiro a ser movido na memória, um LOAD e um STORE são necessários. Cada pacote solicita a injeção do seguinte, por meio de um EXEC. Os últimos pacotes processados enviam SYNCs, para que o pacote finalizador seja, enfim, injetado. A Figura 40 a seguir apresenta a organização desta aplicação.

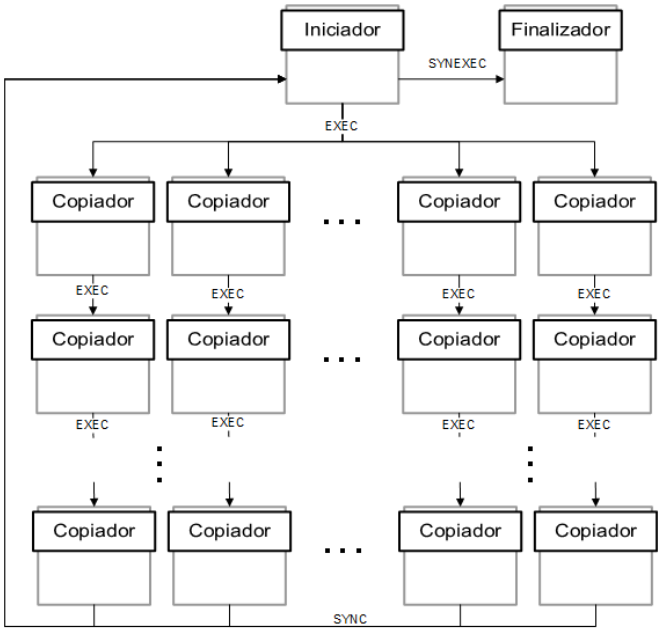


Figura 40: Organização do Experimento 1. Fonte: Autoria própria.

No MPSoC, esse experimento teve duas implementações: uma movimentando os dados dentro do EP 0,0 (identificado nos gráficos de resultado como “EP0,0 p/ EP0,0”) e outra movendo os dados do EP 0,0 para a memória do EP 1,1 (identificado nos gráficos como “EP0,0 p/ EP1,1”). Isso significa que, na primeira implementação, as instruções LOAD e STORE são todas enviadas, pelos demais EPs, para as MAUs do EP 0,0, onde são executadas. Já na segunda, os LOADs são enviados para o EP 0,0 e os STOREs para o EP 1,1. Na IPNoSys Original, a movimentação dos dados acontece em suas próprias memórias. Os

valores a seguir apresentam os resultados obtidos a partir da movimentação de diferentes quantidades de bytes nas arquiteturas, sendo elas: 8 KB, 16 KB, 32 KB e 64 KB.

O gráfico da Figura 41 apresenta a quantidade de ciclos necessária para mover cada uma das quantidades de bytes descritas anteriormente.

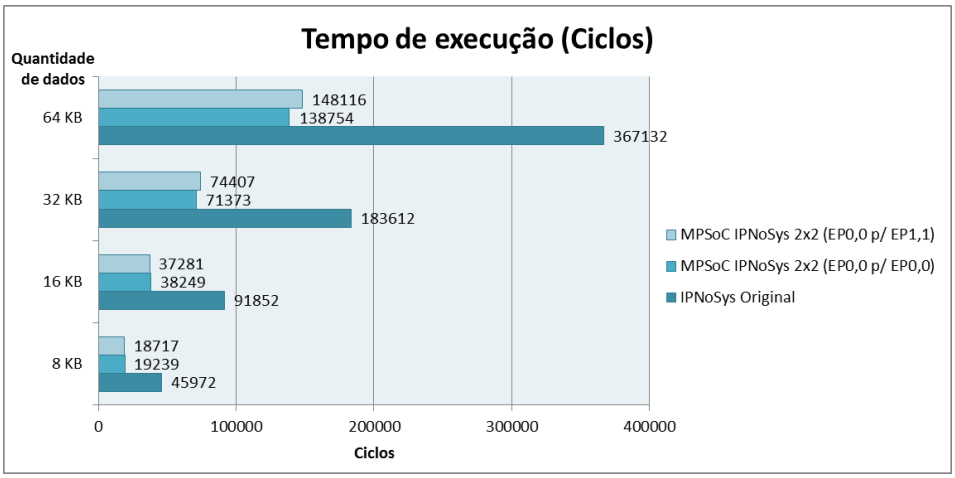


Figura 41: Tempo de execução no Experimento 1. Fonte: Autoria própria.

Nesse experimento, o MPSoC IPNoSys se mostrou, em média, 2,5 vezes superior a arquitetura IPNoSys Original executando a aplicação que move os dados dentro das memórias locais. Para a movimentação dos dados do EP 0,0 para o EP 1,1, o desempenho foi inferior, sendo, em média, 2,46 vezes mais rápido que na arquitetura original.

O próximo gráfico apresenta a energia consumida pelas arquiteturas durante a execução dessas aplicações descritas.

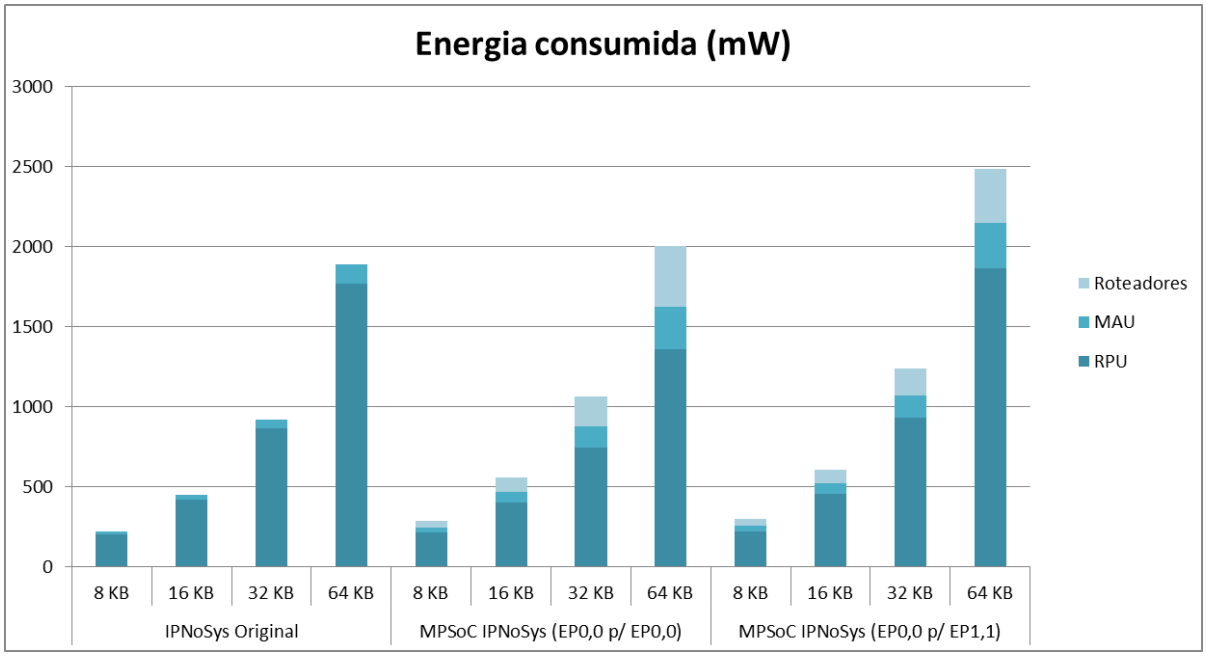


Figura 42: Energia consumida no Experimento 1. Fonte: Autoria própria.

Comparando o resultado da IPNoSys Original ao do MPSoC IPNoSys (EP0,0 p/ EP0,0) mostrados na Figura 42, pode-se perceber que a energia consumida pelo segundo foi ligeiramente maior que pelo primeiro. O alto consumo das MAUs e roteadores pode ser explicado pela localização dos dados e grande quantidade de instruções de acesso à memória desse experimento. Como todos os valores são carregados e armazenados no EP 0,0, as instruções LOAD e STORE são todas enviadas para ele, resultando em um alto tráfego pela rede de roteadores. Nesse caso, o consumo do MPSoC foi, em média, 1,20 vez maior que a da IPNoSys Original. O mesmo acontece com os resultados do MPSoC IPNoSys (EP0,0 p/ EP1,1), porém o consumo foi, em média, 1,34 vez maior.

Por fim, o gráfico da Figura 43 mostra o tempo médio da execução de LOADs nos dois sistemas.

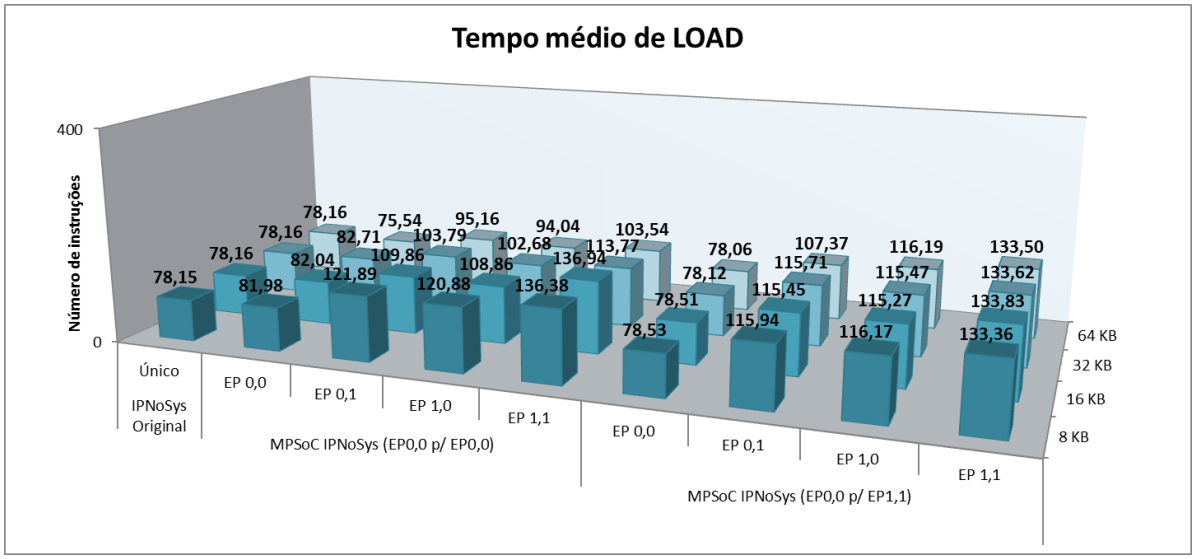


Figura 43: Tempo médio de LOAD no Experimento 1. Fonte: Autoria própria.

Os valores da Figura 43 se referem ao tempo médio, em ciclos, que um pacote com LOAD leva para ir dos respectivos EPs para o EP 0,0, onde o dado é carregado, e retornar para a origem com o RELOAD, seguindo uma lógica semelhante à apresentada no exemplo das Figura 29 e Figura 30. De acordo com o gráfico, em geral, o tempo aumenta à medida que a distância entre os EPs da rede também aumenta. Os tempos relativos ao EP 0,0 são de LOADs locais, os demais são de LOADs remotos.

Somente o tempo referente à instrução LOAD é apresentado neste trabalho devido ela ser a única bloqueante, isto é, a RPU que origina o LOAD só continua com a execução após o recebimento do RELOAD. As demais instruções de acesso à memória e sincronização também têm seu tempo de execução afetado pela distância entre EPs, porém, por não serem

bloqueantes, a influência delas sobre o tempo de execução total da aplicação não é muito relevante.

Um estudo realizado por Viana e Fernandes (2013) mostra que, no MPSoC IPNoSys, o tempo de execução dessas instruções em um EP adjacente consome 15 ciclos a mais que a mesma sendo executada localmente. À medida que a distância entre EPs aumenta, ou seja, a cada salto do pacote de um roteador para outro, 6 ciclos são adicionados ao tempo final de execução da instrução. O tempo da instrução LOAD apresenta esses valores dobrados, pois equivale a soma do tempo gasto com o LOAD e o RELOAD. A Figura 44 apresenta as fórmulas para cálculo da latência na comunicação deduzidas por Viana e Fernandes (2013), onde (a) é a latência no LOAD e (b) a latência nas demais instruções. A variável "s" representa o número de saltos pela rede.

(a) $$L_{LOAD} = 12s + 30$$

(b) $$L_{OTHERS} = 6s + 15$$

Figura 44: Fórmulas para cálculo da latência de comunicação. Fonte: (VIANA; FERNANDES, 2013).

Estes resultados são bem inferiores aos mostrados na Figura 43. A explicação para isso é que os resultados apresentados por Viana e Fernandes (2013) foram obtidos a partir da execução exclusiva de cada instrução no MPSoC. Já os da Figura 43, foram colhidos durante a execução de uma aplicação, com uma grande carga de instruções no sistema. Portanto, além da distância entre EPs, o tempo de execução das instruções de acesso à memoria é influenciado pela carga desse tipo de instrução nos EPs da rede em um dado momento.

5.2 SOMA DE VALORES NA MEMÓRIA

O experimento desta seção realiza a soma de todos os valores existentes em uma determinada região da memória. O objetivo principal desse experimento é avaliar processamento de aplicações com uma quantidade balanceada de instruções de acesso à memória e aritméticas, executadas nas MAUs e RPUs, respectivamente.

A aplicação é composta por um pacote iniciador, um pacote finalizador e vários pacotes somadores, que realizam, além da soma, o carregamento dos dados localizados na

memória. O pacote iniciador possui a mesma constituição e propósito daquele apresentado na seção anterior. Os pacotes somadores possuem uma instrução LOAD e uma ADD para cada valor da memória a ser somado. A quantidade desses pacotes também depende da quantidade de instruções, em razão do limite de 256 por pacote. Eles estão organizados de modo que o pacote em processamento passa o valor somado até o momento para o pacote seguinte, por meio de um SEND, e solicita a injeção deste por meio de um EXEC. Esse processo se repete até que todos os pacotes sejam processados. Os últimos são responsáveis por enviar SYNCs, para que a injeção do pacote finalizador aconteça. Além de concluir a aplicação, o pacote finalizador ainda tem a função de somar todos os valores calculados pelos pacotes somadores e armazenar o resultado na memória, por meio de um STORE. Esses valores calculados chegam até ele por meio de instruções SEND existente nos últimos pacotes somadores. A Figura 45 mostra a organização e algumas dessas mensagens trocadas entre os pacotes deste segundo experimento.

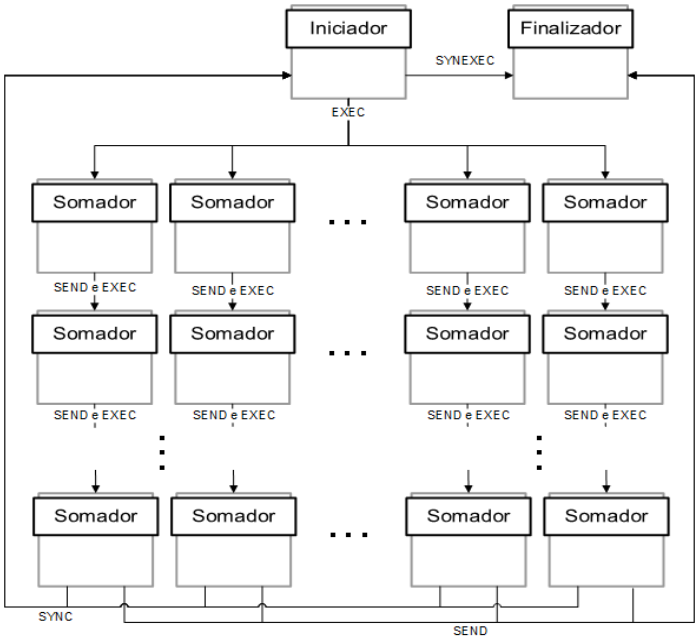


Figura 45: Organização do experimento 2. Fonte: Autoria própria.

Neste experimento, foi realizada a soma das seguintes quantidades de valores armazenados na memória: 2.000, 4.000, 8.000 e 16.000 inteiros. No MPSoC IPNoSys, todos os valores que devem ser somados estão armazenados na memória do EP 0,0. Tanto este EP quanto os demais carregam esses valores, realizam as adições e entregam os resultados para o pacote finalizador, também localizado do EP 0,0. Já na IPNoSys Original, os dados estão na memória local. O gráfico a seguir mostra o tempo gasto em ciclos para a execução deste experimento.

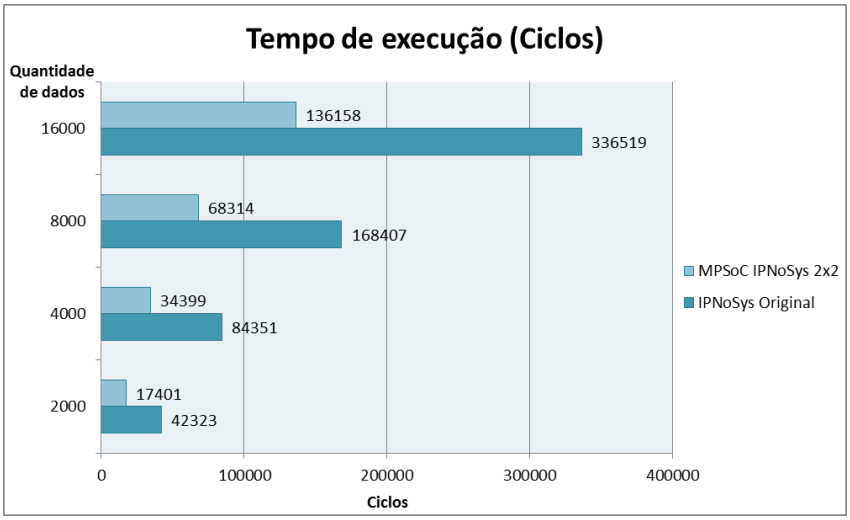


Figura 46: Tempo de execução do Experimento 2. Fonte: Autoria própria.

O desempenho das arquiteturas neste experimento foi semelhante ao do anterior, com o MPSoC IPNoSys se mostrando, em média, 2,45 vezes mais rápido que a IPNoSys Original. Pode-se perceber também que, à medida que a quantidade de dados cresce, o desempenho do MPSoC se torna cada vez melhor em relação à arquitetura original.

A seguir são apresentados os tempos médios de LOAD obtidos no Experimento 2.

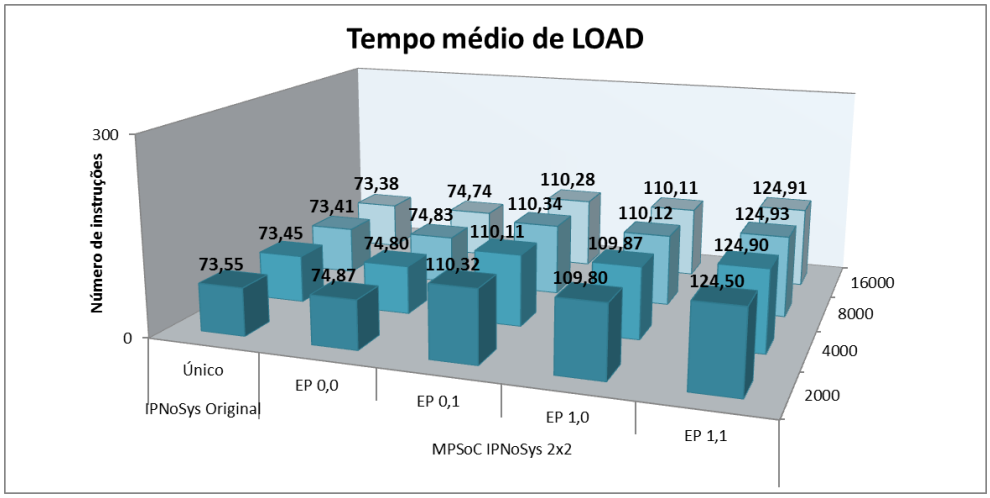


Figura 47: Tempo médio de LOAD do Experimento 2. Fonte: Autoria própria.

Na Figura 47, os resultados do EP 0,0 do MPSoC IPNoSys foram bem próximos aos da IPNoSys Original. Isso demonstra que o desempenho das MAUs é suficiente para atender tanto as requisições locais por dados quanto as requisições feitas simultaneamente pelos outros três EPs do MPSoC. Em outras palavras, mesmo com uma carga de instrução LOAD quatro vezes maior, as MAUs não foram sobrecarregadas, apresentando um desempenho comparável ao do sistema original. Os resultados dos demais EPs foram piores devido ao alto tráfego de pacotes pela rede e os saltos entre roteadores.

O gráfico a seguir mostra o consumo de energia de ambos os sistemas.

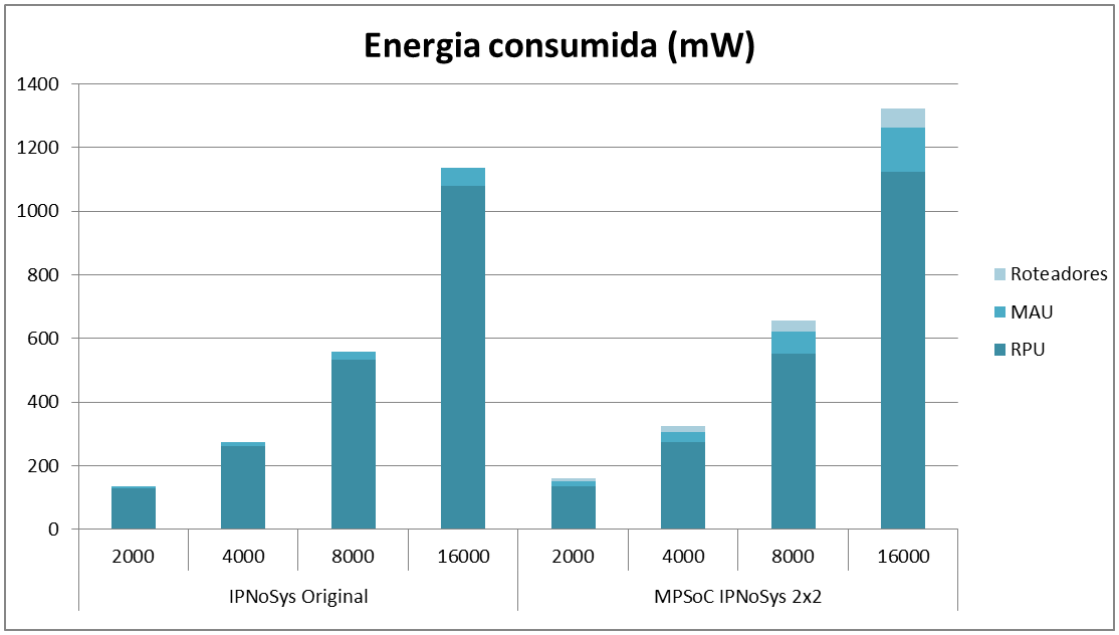


Figura 48: Consumo de energia do Experimento 2. Fonte: Autoria própria.

O MPSoC IPNoSys apresentou um consumo, em média, 1,18 vez superior. Este valor é justificado também pelo alto tráfego de pacotes pela rede, o que faz com que as MAUs e roteadores gastem mais energia. Por fim, o último gráfico desta seção mostra a quantidade de pacotes enviados dos EPs para os roteadores durante a execução.

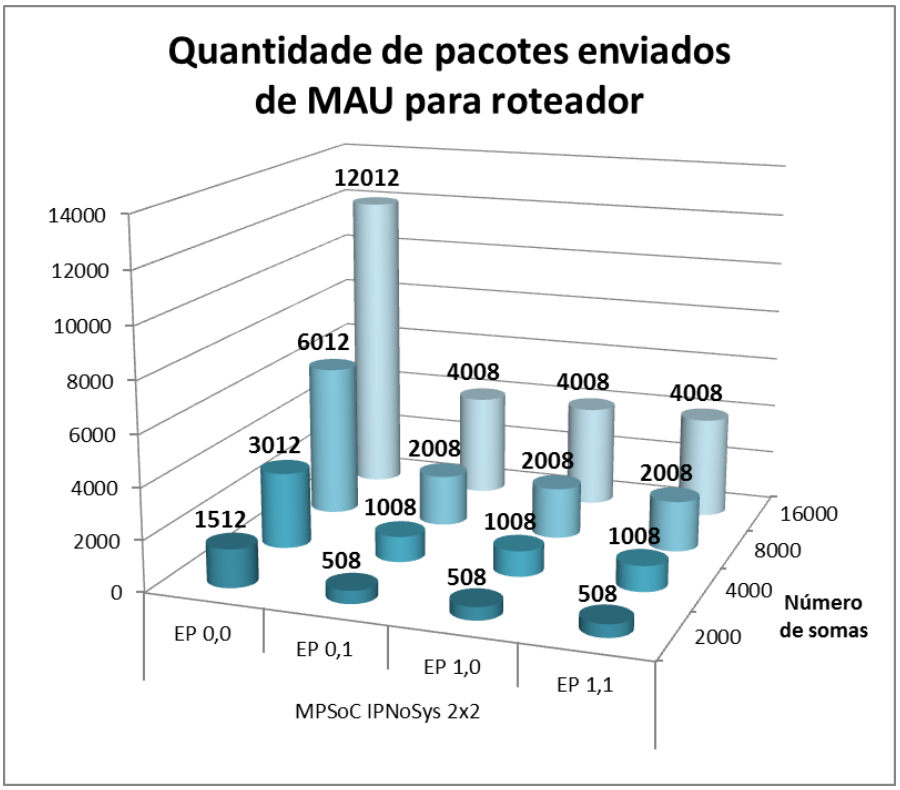


Figura 49: Quantidade de pacotes entregues a rede. Fonte: Autoria própria.

A grande maioria dos pacotes enviados a partir do EP 0,0 possui instruções RELOAD. Já os pacotes enviados pelos demais EPs possuem em sua maioria instruções LOAD. O EP 0,0 também envia alguns pacotes com EXEC e os demais, alguns com instruções SEND e SYNC.

5.3 CONTADOR

A aplicação do terceiro experimento faz, simplesmente, a contagem de zero até um valor especificado. Seu principal objetivo foi avaliar o sistema MPSoC executando aplicações com muitas instruções aritméticas e poucos acessos à memória e sincronização.

Assim como as aplicações anteriores, esta é composta por um pacote iniciador, vários pacotes contadores e um pacote finalizador. O primeiro e o último são semelhantes aos pacotes do Experimento 2. Isso significa que o iniciador solicita a injeção dos pacotes e o finalizador soma os resultados enviados pelos pacotes contadores, armazena na memória e finaliza a aplicação. Os pacotes contadores são formados por duzentas instruções ADD e algumas condicionais e de sincronização. Cada pacote cria um laço de repetição, solicitando a sua própria reinjeção na rede de RPUs até que o valor a ser contado seja alcançado. Quando isso acontece, o pacote finalizador é injetado e a simulação, encerrada. A Figura 50 mostra a organização deste experimento.

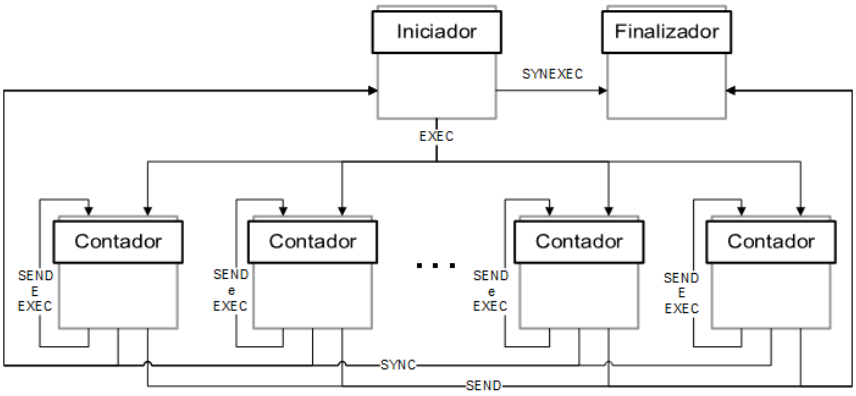


Figura 50: Organização do Experimento 3. Fonte: Autoria própria.

No experimento, foi realizada a contagem de 0 até 128 mil, 256 mil, 512 mil e 1,024 milhão. Essa contagem foi dividida pela quantidade de fluxos de execução, de modo que cada fluxo realizasse uma parte. Após a execução de cada fluxo, os resultados das contagens são

somados e o valor final é obtido. O primeiro gráfico, da Figura 51, mostra o tempo de execução deste experimento.

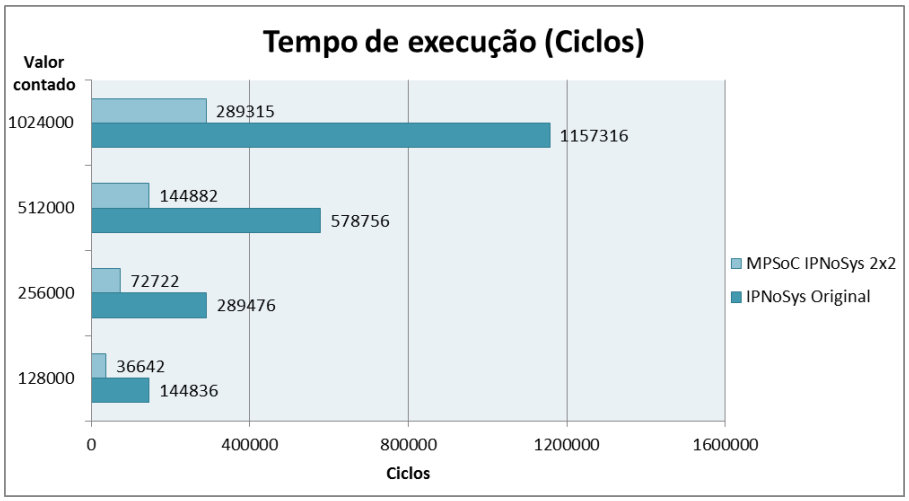


Figura 51: Tempo de execução do Experimento 3. Fonte: Autoria própria.

A comunicação entre EPs neste experimento é mínima: apenas poucos EXECs são enviados do EP 0,0 para os demais e poucos SENDs e SYNCs são enviados dos demais EPs para o EP 0,0. A Figura 52 mostra a quantidade de pacotes enviados por cada EP. Desse modo, praticamente todas as instruções são executadas localmente, com o mínimo de atraso por conta de comunicação ou carga de processamento extra, permitindo o desempenho máximo do MPSoC IPNoSys. Os tempos de execução apresentados na Figura 51 mostram que o desempenho do MPSoC foi, em média, 3,98 vezes superior ao da IPNoSys.

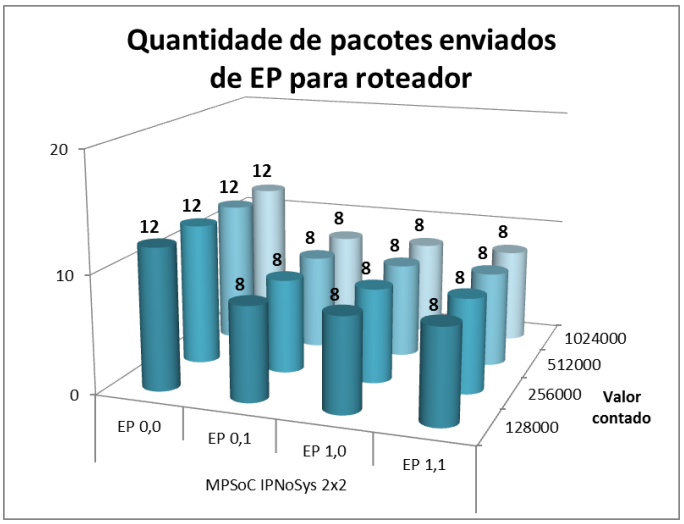


Figura 52: Quantidade de pacotes enviados pelos EPs no Experimento 3. Fonte: Autoria própria.

Em relação à energia consumida, o gráfico da Figura 53 mostra que o MPSoC consumiu um pouco menos que a IPNoSys Original. Como pode ser visto, a energia gasta

pelas MAUs foi bem pequena e praticamente a mesma em ambos os sistemas. Os roteadores do MPSoC também foram pouco utilizados.

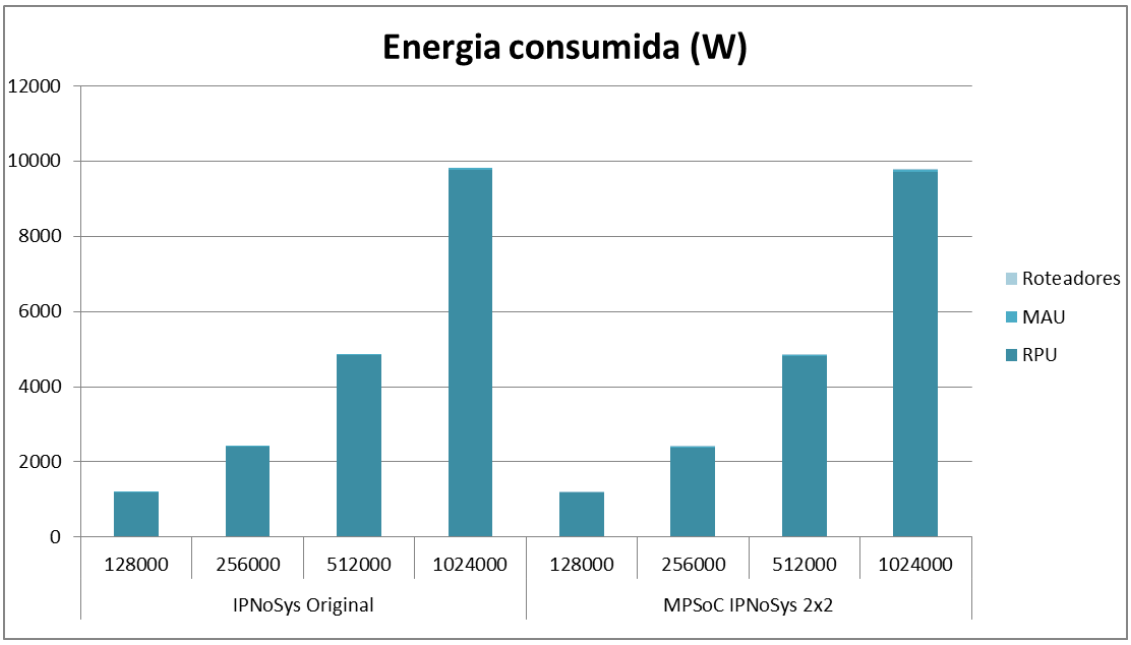


Figura 53: Consumo de energia no Experimento 3. Fonte: Autoria própria.

5.4 MULTIPLICAÇÃO DE MATRIZES

A multiplicação de matrizes recebe como entrada duas matrizes A e B e produz, como resultado, uma matriz C. A Figura 54 apresenta um exemplo de multiplicação de matrizes 2x2.

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \times \begin{vmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{vmatrix} = \begin{vmatrix} a_{11} * b_{11} + a_{12} * b_{21} & a_{11} * b_{12} + a_{12} * b_{22} \\ a_{21} * b_{11} + a_{22} * b_{21} & a_{21} * b_{12} + a_{22} * b_{22} \end{vmatrix}$$

Figura 54: Multiplicação de matrizes. Fonte: Autoria própria.

Apesar de a figura apresentar a multiplicação de simples matrizes 2x2, a lógica utilizada pode ser seguida na multiplicação de matrizes de qualquer dimensão, respeitando a condição de que o número de colunas de A deve ser igual ao número de linhas de B (LIMA, 2012). Cada elemento de C é obtido pela soma resultantes das multiplicações entre elementos de uma linha de A e uma coluna de B, um a um. A matriz C produzida tem o mesmo número de linhas de A e colunas de B.

Essa operação foi utilizada como experimento por se tratar de uma aplicação real simples, bastante comum e que exige um alto esforço computacional. Ela possui ainda, como característica, uma alta quantidade de carregamentos (por meio de LOADs) e uma baixa carga de armazenamento (por meio de STOREs). Além de tudo, sua execução é muito paralelizável.

No experimento, os valores correspondentes às duas matrizes A e B estão, inicialmente, localizados na memória do EP 0,0. O primeiro passo é carregar todos os elementos correspondentes a uma linha de A e todos correspondentes a uma coluna de B. Um a um, eles são multiplicados, os resultados dessas multiplicações são somados e o resultado final é, então, armazenado na memória do EP 0,0. Esse ciclo é repetido para cada elemento de C.

O pacote iniciador é o responsável por criar os fluxos de execução que calculam os elementos de C. Cada fluxo é encarregado de calcular uma parcela do total de elementos da matriz resultante. Eles são constituídos por um pacote carregador A e um carregador B, que carregam elementos de uma linha A e uma coluna B, respectivamente, e um pacote calculador, incumbido de realizar as operações aritméticas necessárias sobre os valores carregados e armazená-los na memória. Quando todos os valores de C são obtidos, o pacote terminador encerra a execução. Essa organização é mostrada pela Figura 55.

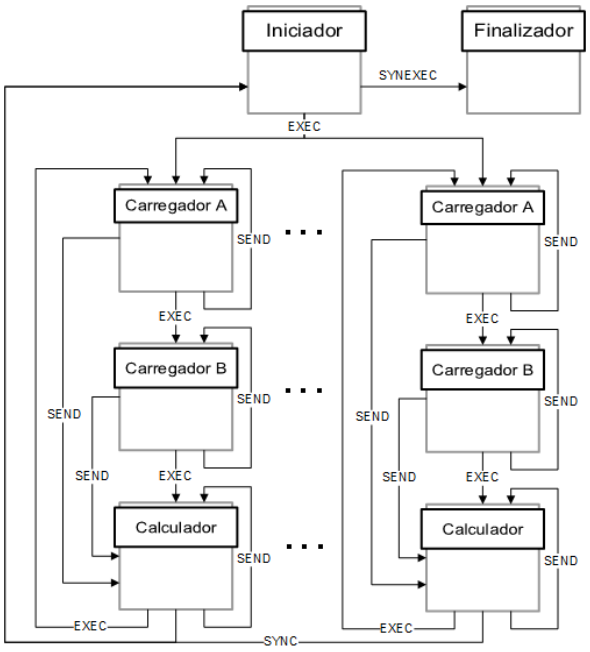


Figura 55: Organização do Experimento 4. Fonte: Autoria própria.

Os resultados apresentados a seguir foram obtidos a partir da multiplicação de matrizes com diversos tamanhos, sendo eles: 15x15, 21x21, 27x27 e 33x33. O primeiro aspecto, apresentado pela Figura 56, é a memória requerida para a execução de cada

multiplicação. O total de espaço na memória é resultado da soma da quantidade de dados e código da aplicação.

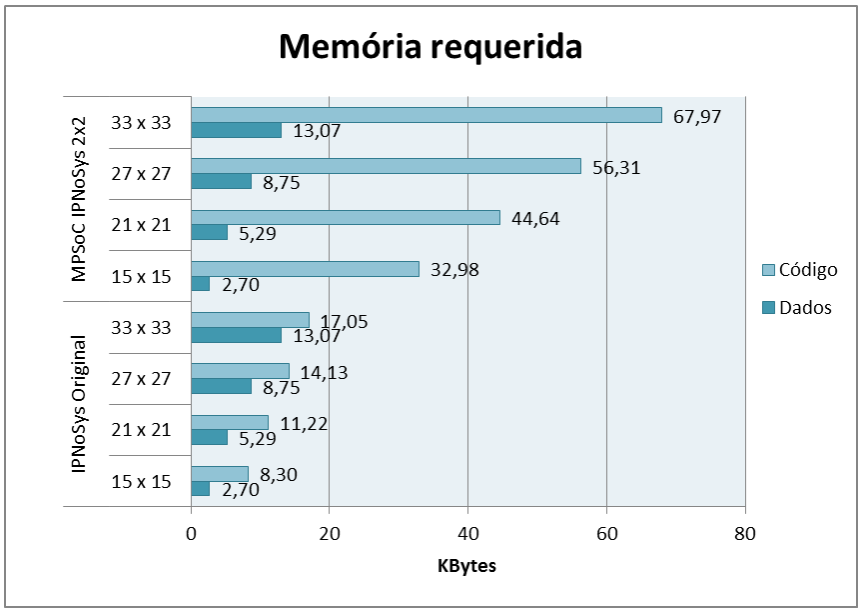


Figura 56: Memória requerida para o Experimento 4. Fonte: Autoria própria.

Em todos os resultados apresentados na figura anterior, a quantidade de dados foi a mesma em ambos os sistemas, o suficiente armazenar três matrizes: duas de entrada para a multiplicação e uma de saída, como resultado. A quantidade de código, por sua vez, aumentou bastante, por causa da quantidade de fluxos de execução de cada sistema. Como o MPSoC permite quatro vezes mais fluxos, o tamanho do também código pode ser até quatro vezes maior.

O próximo gráfico corresponde ao desempenho dos sistemas realizando a multiplicação de matrizes.

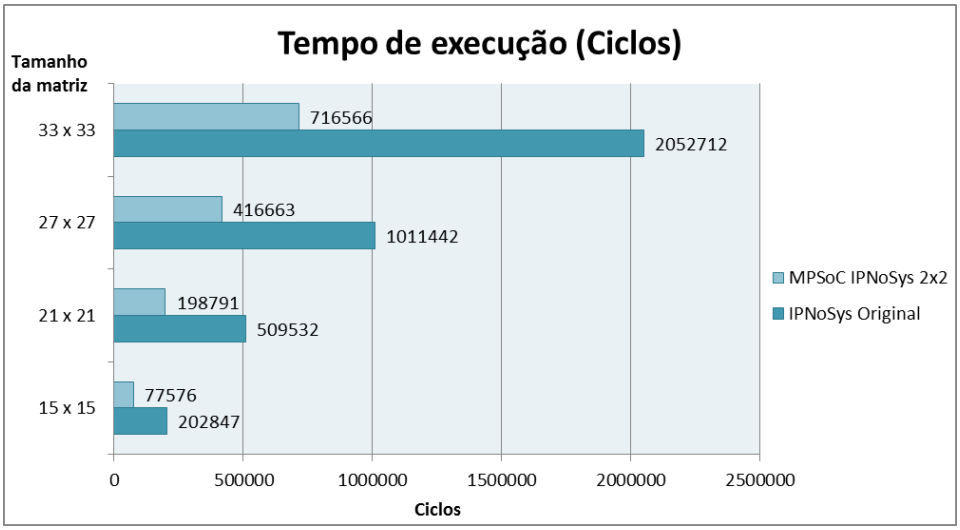


Figura 57: Tempo de execução do Experimento 4. Fonte: Autoria própria.

Os resultados mostram que o desempenho do MPSoC na multiplicação de matrizes é, em média, 2,62 vezes superior ao da IPNoSys Original. Apesar de uma grande quantidade de operações aritméticas, o número de LOAD também é alto, o que diminui um pouco o desempenho do sistema. Já em relação à energia consumida, o MPSoC se saiu 1,09 vez pior. A Figura 58 detalha este resultado.

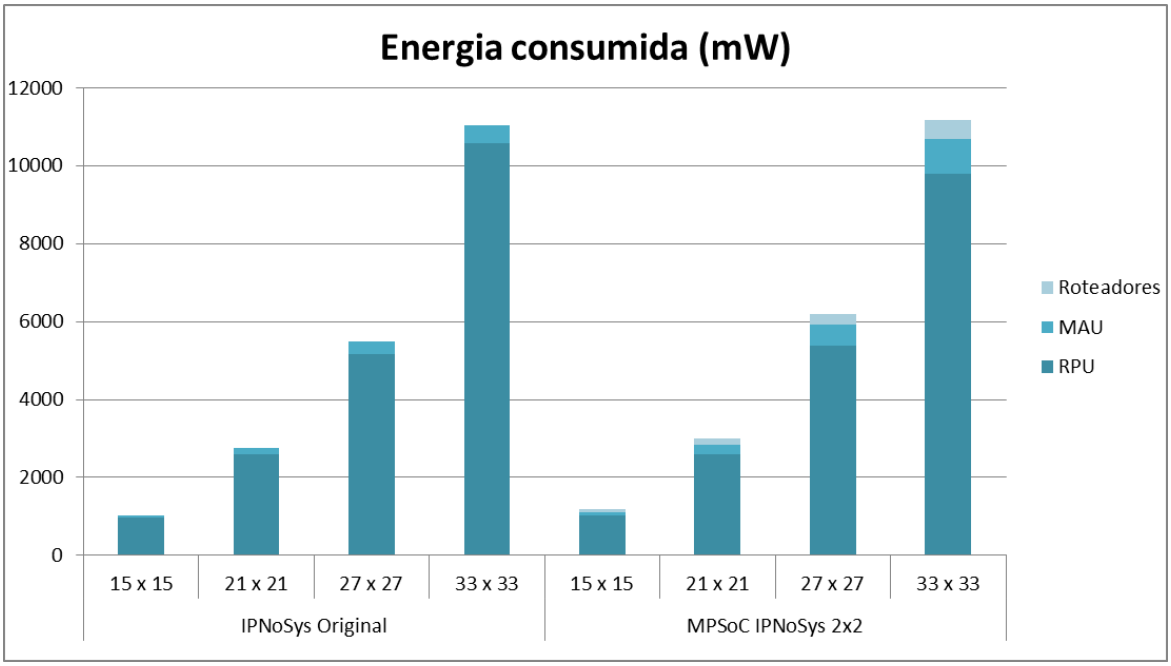


Figura 58: Energia consumida no Experimento 4. Fonte: Autoria própria.

Por fim, o tempo médio de LOAD obtido na multiplicação de matrizes é apresentado na Figura 59.

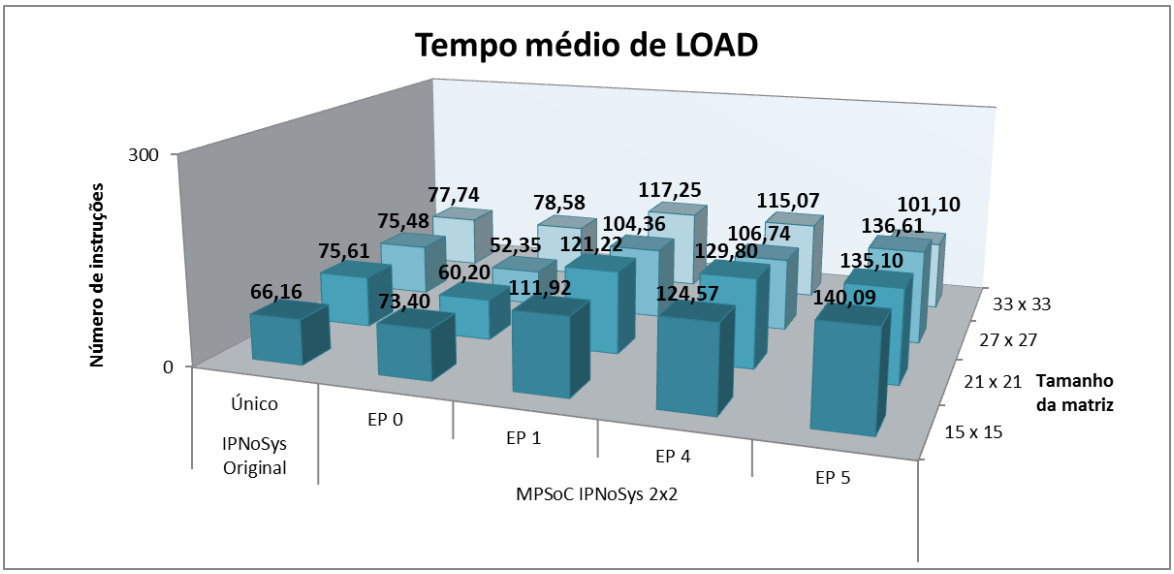


Figura 59: Tempo médio de LOAD no Experimento 4. Fonte: Autoria própria.

5.5 DESCOMPRESSÃO RLE

O *Run-Length Encoding* (RLE) é um algoritmo de compressão de dados, cujo princípio básico é a substituição de uma sequência de símbolos repetidos pela quantidade de vezes que eles se repetem e o próprio símbolo (AGOSTINI, 2002). Uma simplificação do RLE, otimizada para a codificação JPEG, se resume a substituir as sequências de zeros pela quantidade de vezes que este símbolo se repete antes de cada símbolo distinto dele. A descompressão RLE é o processo inverso à compressão. Em um conjunto de dados comprimidos, cada par de símbolos tem o primeiro elemento indicando a quantidade de zeros na sequência e o segundo, o símbolo após esta sequência. A Figura 60 exemplifica esses dois processos.

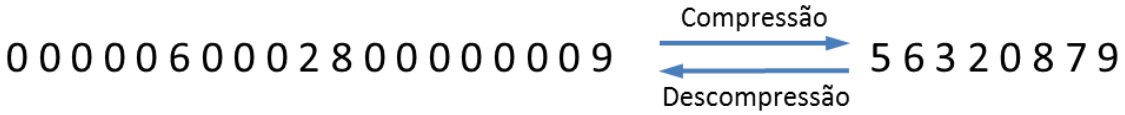


Figura 60: Exemplo de codificação RLE. Fonte: Autoria própria.

O descompressor RLE foi utilizado como experimento por realizar uma grande quantidade de armazenamento (STORE) e uma pequena quantidade de carregamento (LOAD) de dados. Essa característica contrasta com a da multiplicação de matrizes, onde ocorre o inverso. Figura 61 a seguir apresenta a organização dessa aplicação.

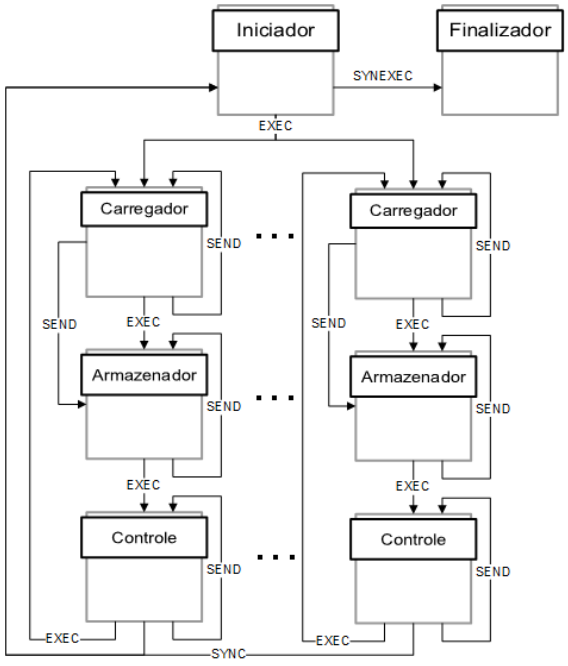


Figura 61: Organização do Experimento 5. Fonte: Autoria própria.

O pacote iniciador, como nos demais experimentos, cria os vários fluxos de execução da aplicação. Nela, cada fluxo é composto por um pacote carregador, um pacote armazenador e um pacote de controle. O primeiro é encarregado de carregar um par de valores armazenados na memória. O segundo armazena na memória a sequencia de zeros e o valor determinados pelo carregamento realizado no pacote anterior. O terceiro pacote controla os ciclos de repetição. Enquanto todos os dados não forem processados, o pacote carregador é reinjetado. Quando tudo estiver terminado, o sinal para injeção do pacote finalizador é enviado, concluindo a execução. Inicialmente, os dados de entrada estão na memória do EP 0,0. Após a descompactação, os valores produzidos também se encontram neste mesmo EP.

Assim como nos demais experimentos, diferentes valores também foram utilizados neste. A Tabela 5 a seguir faz a correspondência entre a quantidade de números inteiros utilizados como entrada e a respectiva quantidade de inteiros produzidos após a descompactação para cada um. Os valores utilizados nos testes foram criados aleatoriamente.

Tabela 5: Entrada e saída do descompressor RLE. Fonte: Autoria própria.

ID do teste	Quantidade de valores compactados	Quantidade de valores produzidos
T1	1170	8192
T2	2544	16384
T3	4750	32768
T4	7406	49152

O primeiro gráfico apresenta a quantidade de memória requerida por este experimento em cada teste.

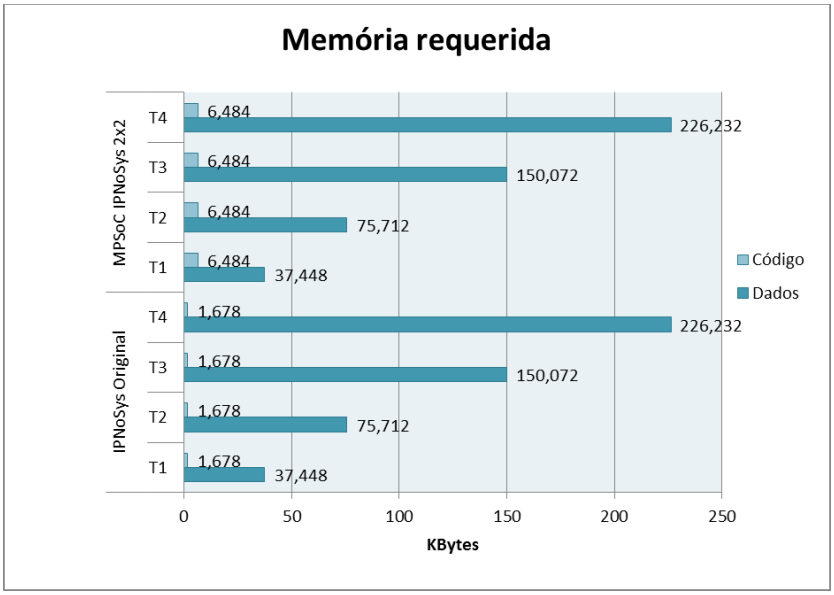


Figura 62: Memória requerida no Experimento 5. Fonte: Autoria própria.

Devido à simplicidade do algoritmo RLE, seu código é bem pequeno. Em razão do maior número de fluxos, a memória requerida para código no MPSoC IPNoSys é 3,86 vezes maior que na IPNoSys Original. De forma inversa à multiplicação de matrizes, a quantidade de código neste experimento é bem menor que a quantidade de dados. Esta, por sua vez, é igual à soma da quantidade dos valores de entrada e de saída apresentados na Tabela 5. No gráfico, a memória requerida aparece em KB.

O gráfico a seguir apresenta o tempo de execução de cada teste. Em média, o MPSoC IPNoSys foi 3,47 vezes superior à arquitetura original.

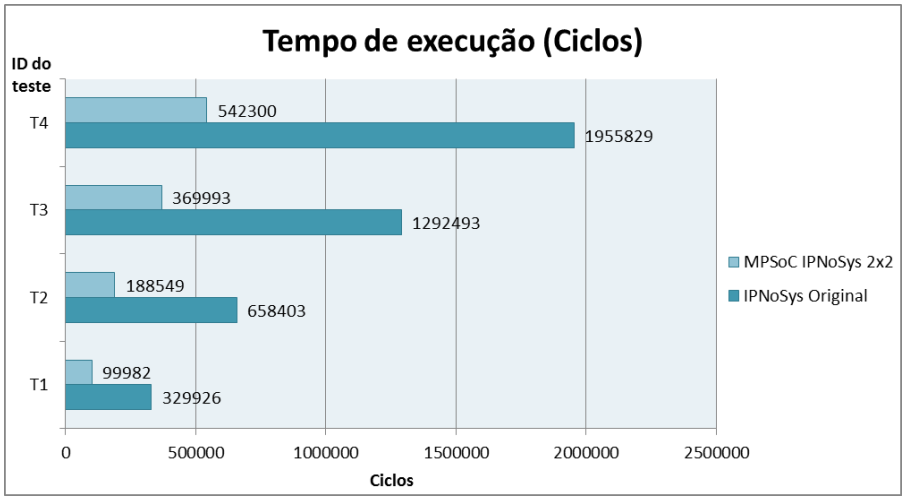


Figura 63: Tempo de execução do Experimento 5. Fonte: Autoria própria.

O algoritmo RLE possui uma grande quantidade de instruções de acesso à memória executadas no EP 0,0. A Figura 64 comprova isso demonstrando a quantidade de pacotes enviados a partir dos roteadores para cada EP.

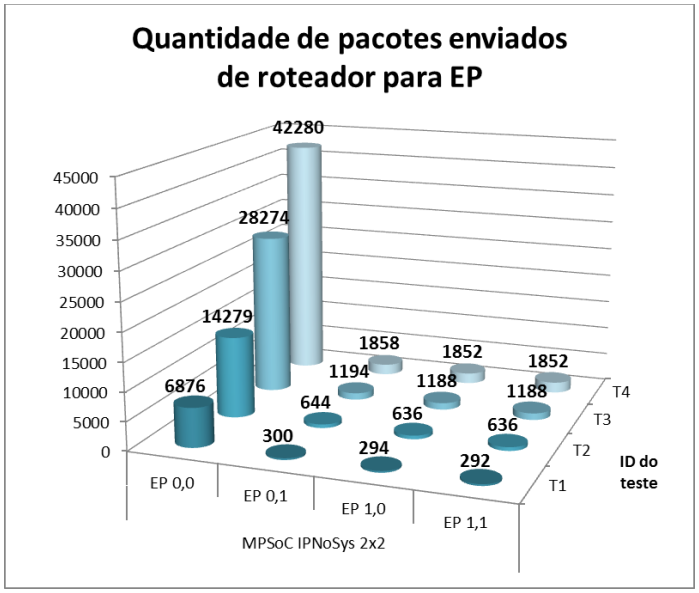


Figura 64: Pacotes recebidos pelos EPs no Experimento 5. Fonte: Autoria própria.

Apesar da grande quantidade de instruções de acessos à memória executadas no EP 0,0, a maioria são instruções STORE, do tipo não bloqueante. Isso faz com que o desempenho da arquitetura MPSoC neste experimento seja bem superior ao de outros com uma grande quantidade de instruções bloqueantes, como o experimento da seção 5.2.

Por fim, o consumo de energia foi bem constante em todos os testes, com o MPSoC consumindo, em média, 1,17 vez mais que a IPNoSys Original, mesmo com a alta taxa de utilização da rede. A Figura 65 a seguir mostra os resultados relacionados a este ponto.

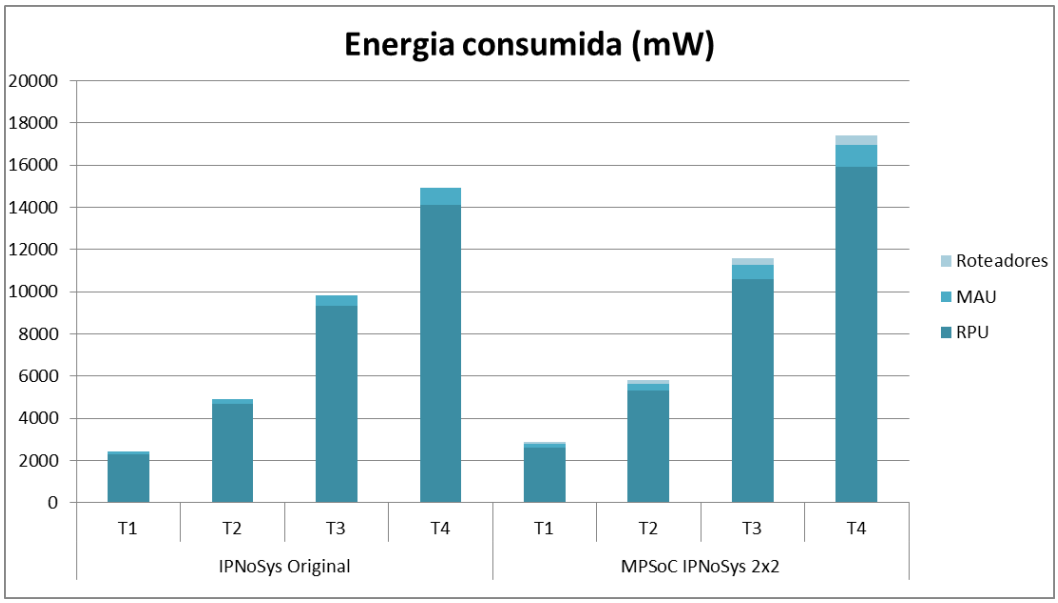


Figura 65: Consumo de energia no Experimento 5. Fonte: Autoria própria.

5.6 DCT-2D

A Transformada Discreta do Cosseno em duas dimensões, ou *Discrete Cosine Transform* 2D (DCT-2D), em inglês, é um cálculo matemático muito utilizado em processamento digital de imagens e compressão de dado. Existem diferentes abordagens a respeito do cálculo da DCT-2D, conforme discutido por Agostini (2002). O algoritmo considerado aqui foi aquele, apresentado nesse mesmo trabalho, cuja transformada é calculada a partir da aplicação da DCT-1D duas vezes sobre um mesmo bloco de dados de tamanho 8x8. A primeira DCT-1D é realizada linha a linha do bloco e a segunda, coluna a coluna, produzindo um bloco de tamanho semelhante como resultado. O cálculo da DCT-1D é feito por meio de 29 adições e 5 multiplicações, organizadas em seis passos. A Figura 66 apresenta esta operação.

No cálculo da Figura 66, os elementos representados pela letra “a” são os dados de entradas, ou seja, os oito valores correspondentes a uma linha ou coluna do bloco. Os elementos representados pela letra “S” são os dados de saída. As demais letras representam variáveis auxiliares utilizadas no processo.

Passo 1

$b0 = a0 + a7$	$b1 = a1 + a6$	$b2 = a3 - a4$
$b3 = a1 - a6$	$b4 = a2 + a5$	$b5 = a3 + a4$
$b6 = a2 - a5$	$b7 = a0 - a7$	

Passo 2

$c0 = b0 + b5$	$c1 = b1 - b4$	$c2 = b2 + b6$
$c3 = b1 + b4$	$c4 = b0 - b5$	$c5 = b3 + b7$
$c6 = b3 + b6$	$c7 = b7$	

Passo 3

$d0 = c0 + c3$	$d1 = c0 - c3$	$d2 = c2$
$d3 = c1 + c4$	$d4 = c2 - c5$	$d5 = c4$
$d6 = c5$	$d7 = c6$	$d8 = c7$

Passo 4

$e0 = d0$	$e1 = d1$	$e2 = m3 \times d2$
$e3 = m1 \times d7$	$e4 = m4 \times d6$	$e5 = d5$
$e6 = m1 \times d3$	$e7 = m2 \times d4$	$e8 = d8$

Passo 5

$f0 = e0$	$f1 = e1$	$f2 = e5 + e6$
$f3 = e5 - e6$	$f4 = e3 + e8$	$f5 = e8 - e3$
$f6 = e2 + e7$	$f7 = e4 + e7$	

Passo 6

$S0 = f0$	$S1 = f4 + f7$	$S2 = f2$
$S3 = f5 - f6$	$S4 = f1$	$S5 = f5 + f6$
$S6 = f3$	$S7 = f4 - f7$	

Onde:

$$m1 = \cos\left(\frac{4\pi}{16}\right)$$

$$m2 = \cos\left(\frac{6\pi}{16}\right)$$

$$m3 = \cos\left(\frac{2\pi}{16}\right) - \cos\left(\frac{6\pi}{16}\right)$$

$$m4 = \cos\left(\frac{2\pi}{16}\right) + \cos\left(\frac{6\pi}{16}\right)$$

Figura 66: Cálculo da DCT-1D. Fonte: (AGOSTINI, 2002).

A implementação em PDL segue o algoritmo exatamente como apresentado na Figura 66. O pacote iniciador solicita a injeção dos pacotes DCT-1D, que fazem o carregamento das linhas de dados, os cálculos e o armazenamento dos resultados na memória. Quando cada pacote DCT-1D conclui seus cálculos, eles sinalizam para a injeção de um segundo pacote iniciador, responsável por solicitar a injeção dos pacotes DCT-2D. Estes são semelhantes aos pacotes DCT-1D, tendo como diferença a posição de memória para as operações de carregamento e armazenamento de dados. Quando o processamento dos pacotes DCT-2D é concluído, o pacote finalizador é injetado, encerrando a execução. A Figura 67 mostra a organização deste experimento.

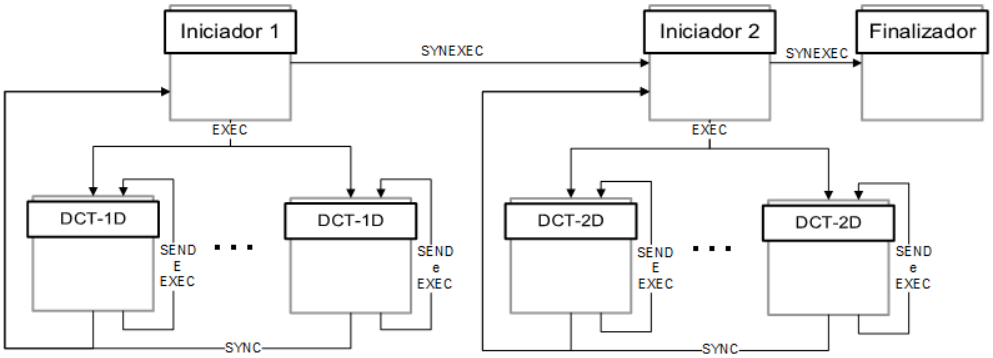


Figura 67: Organização do Experimento 6. Fonte: Autoria própria.

Neste experimento, a quantidade de dados utilizada nos testes está organizada em blocos de tamanho 8x8. Foram executados testes com 128, 192, 256 e 320 blocos. A quantidade de memória requerida para dados é igual a três vezes a quantidade de valores dos blocos. O primeiro terço da memória de dados é onde os valores dos blocos de entrada estão armazenados. O segundo terço é utilizado para armazenar os resultados da DCT-1D. O terceiro, e último, terço é onde o resultado final da DCT-2D vai estar armazenado após a execução de toda a aplicação. Os valores iniciais estão armazenados na memória do EP 0,0, os intermediários, obtidos pela DCT-1D, são armazenados nas memórias locais de cada EP e os valores finais, da DCT-2D, são armazenados, por fim, na memória do EP 0,0.

O gráfico da Figura 68 mostra o desempenho das arquiteturas neste último experimento.

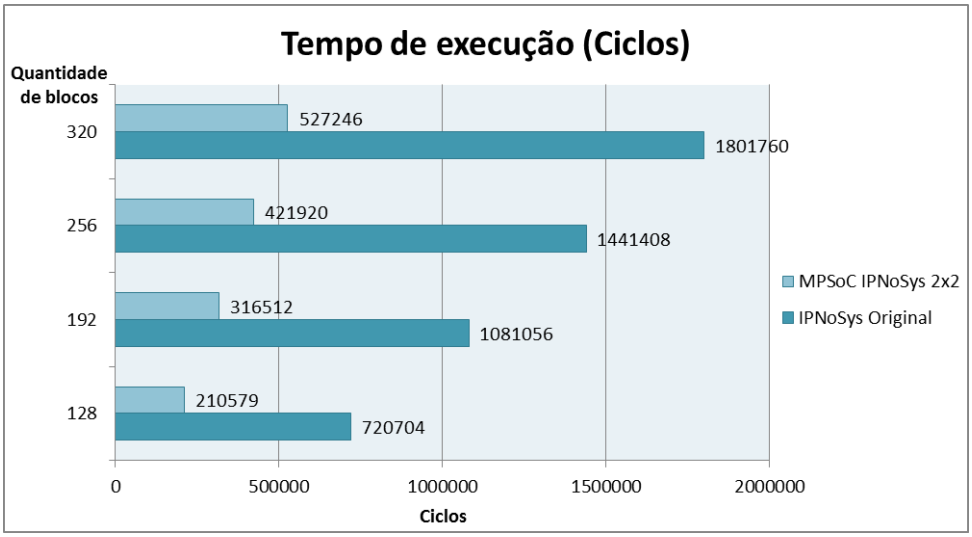


Figura 68: Tempo de execução no Experimento 6. Fonte: Autoria própria.

O ganho no tempo de execução foi praticamente o mesmo em todos os testes, com o MPSoC apresentando desempenho 3,42 vezes superior. O consumo de energia é mostrado no próximo gráfico.

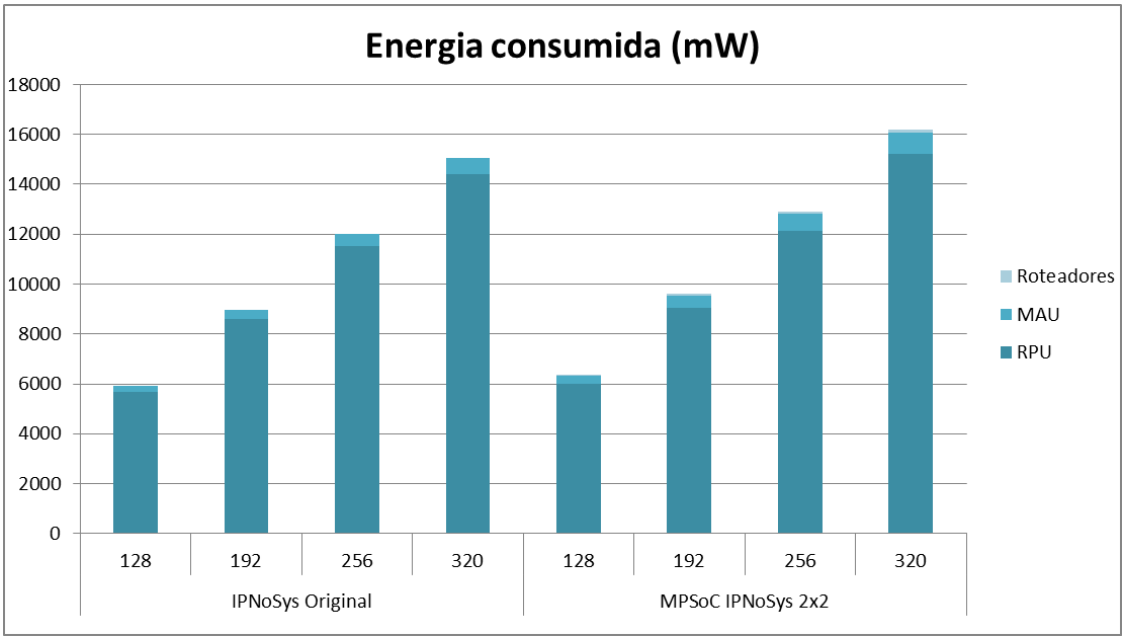


Figura 69: Consumo de energia no Experimento 6. Fonte: Autoria própria.

O consumo de energia no MPSoC IPNoSys também foi bem regular, em torno de 1,07 vez maior que na IPNoSys Original. Os baixo consumo dos roteadores é justificado pela estratégia de armazenar os valores da primeira DCT localmente. Isso evita que vários STOREs e LOADs trafeguem pela rede, além de melhorar o desempenho do sistema.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O uso de arquiteturas paralelas para a obtenção de maior poder computacional tem se tornado cada vez maior nos últimos anos. Isso pode ser percebido no mercado de eletrônicos, onde computadores, tablets, smartphones, câmeras fotográfica, entre outros, são facilmente encontrados com processadores multinúcleo. Além do alto desempenho, esses sistemas proporcionam maiores eficiência energética, escalabilidade, flexibilidade e um reduzido tempo de projeto e custos.

A IPNoSys é um sistema nativamente paralelo, que apresenta uma arquitetura não convencional e um modelo de execução de instruções diferentes dos tradicionais, onde os programas estão organizados dentro de pacotes de instruções e dados. Apesar de seu alto desempenho no processamento de aplicações paralelas, suas limitações arquiteturais impedem um rendimento ainda maior.

O presente trabalho apresentou uma proposta de arquitetura MPSoC baseada na IPNoSys que possibilita maior quantidade de fluxos de execução simultâneos e, por consequência, maior desempenho. Neste MPSoC, múltiplas IPNoSys estão conectadas através de uma NoC, de modo que as RPUs da IPNoSys se tornaram uma rede local de processamento de dados e a NoC, uma rede global de comunicação entre EPs. A interface entre essas duas redes foi feita através das MAUs, que receberam um novo canal de entrada e saída para a comunicação com os roteadores da NoC. A comunicação entre EPs acontece por meio da troca de pacotes de controle. Estes receberam alguns novos campos, com o objetivo de possibilitar o endereçamento dos componentes de cada IPNoSys do sistema. A linguagem PDL e o montador da arquitetura também tiveram que ser alterados.

Com o objetivo de avaliar a nova arquitetura proposta, o simulador do MPSoC IPNoSys foi desenvolvido em SystemC com precisão de ciclos. Vários experimentos, com diferentes características, foram implementados e colocados em execução neste simulador. Foram implementadas também versões dos mesmos experimentos para a IPNoSys Original, colocados em execução no simulador de tal arquitetura, desenvolvido por Fernandes (2012). A diferença básica entre as duas versões dos experimentos foi a quantidade de fluxos de execução, sendo dezesseis para o MPSoC IPNoSys e quatro para a arquitetura original.

A comparação entre os sistemas, levando em conta o resultado de todos os experimentos, mostrou que o MPSoC, em média, possui desempenho 2,98 vezes superior ao da IPNoSys Original. Se forem considerados apenas os resultados dos três últimos

experimentos apresentados, que consistem em aplicações reais, a arquitetura proposta apresenta desempenho 3,17 vezes superior a original. Conforme discutido, as aplicações com alta taxa de comunicação proporcionam um desempenho inferior, devido à latência na comunicação. Essa latência, por sua vez, apresenta valores variáveis, dependendo do tráfego na rede e da distância entre os EPs.

Em relação ao consumo de energia, a IPNoSys Original se mostrou melhor, com o MPSoC consumindo, em média, 1,15 vez mais. Novamente, considerando apenas os três últimos experimentos, este valor cai para 1,11 vez. O maior consumo de energia é causado pela comunicação entre EPs, pois, nesta operação, os pacotes precisam atravessar a rede de roteadores e uma MAU extra, o que leva ao uso de um maior número *buffers*, se comparado à arquitetura original. No caso da instrução LOAD, essa quantidade de componentes atravessados é ainda maior, pois um pacote com RELOAD ainda é gerado e devolvido à origem do primeiro. No Experimento 3, cuja a taxa de comunicação é muito baixa, o consumo de energia pelas duas arquiteturas foi semelhante.

Outra observação importante foi com relação ao desempenho das MAUs. Mesmo recebendo quatro vezes mais requisições, elas responderam em tempo semelhante ao da IPNoSys Original. Isso pode ser comprovado pela análise dos tempos de LOAD, apresentados em alguns experimentos. Outras observações e suas respectivas justificativas foram discutidos no próprio Capítulo 5, de experimentos e resultados. Por meio dos resultados obtidos, acredita-se que a arquitetura proposta neste trabalho é válida, apresentando um grande ganho em poder computacional ao custo de um pequeno consumo extra de energia.

O principal trabalho futuro pretendido é a implementação da arquitetura apresentada aqui em linguagem de descrição de hardware, tal como VHDL. Isso permitiria que mais resultados fossem colhidos, como área em chip, energia consumida pelo sistema e frequência máxima de operação, além dos demais apresentados, mas a partir de um protótipo real. Atualmente, uma pesquisa em andamento, por Neto e Fernandes (2013), está desenvolvendo a IPNoSys Original em VHDL. Tal trabalho futuro só poderá ser completamente executado após a conclusão desta pesquisa em execução, uma vez que a IPNoSys é um de seus principais elementos.

Outros trabalhos futuros são referentes à implementação de possíveis melhorias identificadas a partir de dificuldades encontradas durante os testes. Uma delas é o aumento da quantidade de bits do campo “# Instruções”, no cabeçalho dos pacotes. Atualmente, este campo permite pacotes com apenas 256 instruções, mas, em alguns casos, tamanhos maiores que este são necessários. Quando isto acontece, é preciso enviar os dados para outros pacotes,

a fim de continuar com o processamento, por meio de SENDs e EXECs extras. O tempo gasto com a execução dessas instruções poderia ser evitado com pacotes de tamanhos maiores.

Pretende-se ainda avaliar o desempenho do MPSoC com números maiores de EPs e quantidades menores de RPUs por EP. Configurações deste tipo podem permitir maior quantidade de fluxos de execução paralelos e economia da área total ocupada em chip.

O desenvolvimento de interfaces de rede e capacidade de comunicação para arquitetura IPNoSys possibilita o desenvolvimento de outros sistemas multiprocessados, como por exemplo uma IPNoSys 3D. Este sistema também consiste em uma rede de IPNoSys empilhadas, onde a comunicação entre EPs ocorre pela troca direta de mensagens entre MAUs adjacentes, sem o uso de roteadores. A Figura 70 representa uma visão geral deste sistema, que se encontra em fase desenvolvimento.

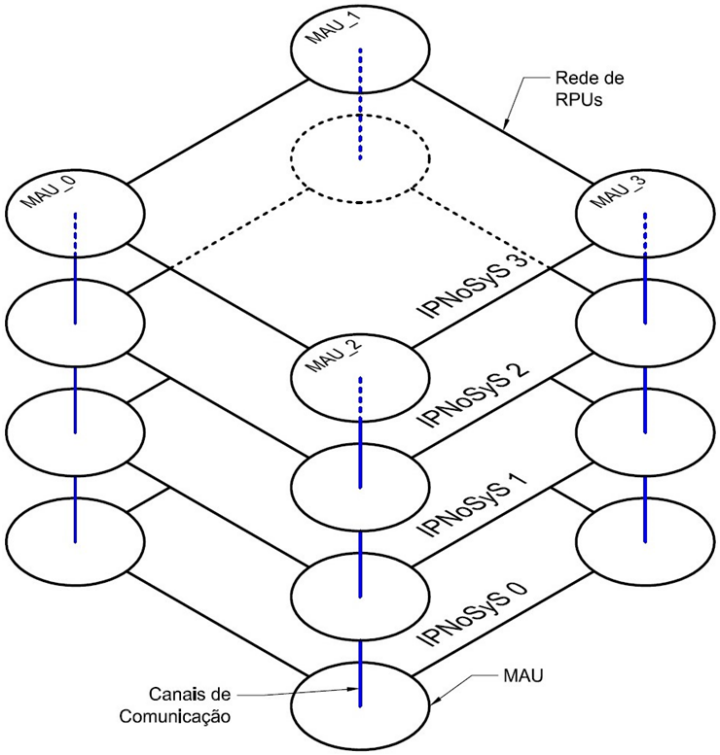


Figura 70: Visão Geral da IPNoSys 3D. Fonte: Autoria própria.

Portanto, este trabalho abriu caminho para o desenvolvimento de novas plataformas multiprocessadas baseadas em IPNoSys a partir de modificações nas interfaces de comunicação das MAUs, no formado dos pacotes, na PDL e no montador propostos neste trabalho.

7 REFERÊNCIAS

AGOSTINI, L. V. **Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG**. Dissertação (Mestrado em Sistemas e Computação) – Instituto de Informática. Universidade Federal do Rio Grande do Sul. Porto Alegre. 2002

ALVES, M. A. Z.; FREITAS, H. C.; WAGNER, F. R.; NAVAUX, P. O. A. Influência do Compartilhamento de *Cache* L2 em um *Chip* Multiprocessado sob Cargas de Trabalho com Conjuntos de Dados Contíguos e Não Contíguos. In: VIII Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD), Gramado. **Anais**... 2007. p. 27-34.

AROCA, R. V.; GONÇALVES, L. M. G. *Towards green data centers: A comparison of x86 and ARM architectures power efficiency*. ***Journal of Parallel and Distributed Computing* (*Print*)**, v. 72. 2012. p. 1770-1780.

FERNANDES, S. R. **Projeto de Sistemas Integrados de Propósito Geral Baseados em Redes em *Chip*** – Expandindo as Funcionalidades dos Roteadores para Execução de Operações: A plataforma IPNoSys. 2012. 191 f. Tese (Doutorado em Sistemas e Computação) – Departamento de Informática e Matemática Aplicada. Universidade Federal do Rio Grande do Norte. Natal. 2012.

FERNANDES, S. R.; OLIVEIRA, B. C.; COSTA, M.; SILVA, I. S. IPNoSys: uma nova arquitetura paralela baseada em Redes em Chip. In: IX Simpósio em Sistemas Computacionais (WSCAD-SSC), Campo Grande. **Anais**... 2008. p. 53-60.

FERNANDES, S. R.; OLIVEIRA, B. C.; COSTA, M.; SILVA, I. S. *Processing while routing: a network-on-chip-based parallel system*. In: IET *Computers* & *Digital Techniques*. **Anais**... 2009a. p. 525-538.

FERNANDES, S. R.; OLIVEIRA, B. C.; SILVA, I. S. *Using NoC routers as processing elements*. In: Simpósio de Circuitos Integrados e Design de Sistemas (SBCCI), Natal. **Anais**... 2009b.

FERNANDES, S. R.; SILVA, I. S.; VERAS, R. *I/O Management and Task Scheduler on Packet-Drive General Purpose Architecture*. In: XXXVII Conferencia Latinoamericana de Informática (CLEI), Quito. **Anais**... 2011. p. 1284-1295.

FERREIRA, R. E. **Implementação de algoritmos genéticos paralelos em uma arquitetura MPSoC**. 2009. 196 f. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Engenharia. Universidade do Estado do Rio de Janeiro. Rio de Janeiro. 2009.

HERVÉ, M. B. **Métodos de Teste de Redes-em-Chip (NoCs)**. 2009. 120 f. Dissertação (Mestrado em Microeletrônica) – Instituto de Informática, Instituto de Física, Instituto de Química e Escola de Engenharia. Universidade Federal do Rio Grande do Sul. Porto Alegre. 2009.

HESS, C. R. **MDX-cc**: Ambiente de Programação Paralela Aplicado a *Cluster* de *Clusters*. 2003. 132 f. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Informática. Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre. 2003.

LENG, Xianglun; XU, Ningyi; DONG, Feng; ZHOU, Zucheng, *Implementation and simulation of a cluster-based hierarchical NoC architecture for multi-processor SoC*. In: *IEEE International Symposium, Communications and Information Technology*. **Proceedings**... 2005. p.1203-1206. 2005.

LIMA, C. G. **Álgebra de Matrizes**. Departamento de Ciências Exatas / ESALQ - USP. São Paulo. 2012. Disponível em: <http://verde.esalq.usp.br/~jorge/cursos/cesa r/Apostila de Matrizes (2012).pdf > Acesso em: 20 de jan. 2014.

LUO-FENG, Geng; DUO-LI, Zhang; MING-LUN, Gao, *Performance evaluation of cluster-based homogeneous multiprocessor system-on-chip using FPGA device*. In: *2nd International Conference, Computer Engineering and Technology (ICCET).* **Proceedings**... 2010. p.V4-144,V4-147.

MARTINI, J. A; GONÇALVES JUNIOR, N. A.; GONÇALVES, R. A. L. Análise de Desempenho de Topologias para Redes em Chip. In: X Simpósio em Sistemas Computacionais (WSCAD-SSC), São Paulo. **Anais**... 2009.

MONTEZ, C. B. **Um Sistema Operacional com Micronúcleo Distribuído e um Simulador Multiprogramado de Multicomputador**. 1995. 95 f. Dissertação (Mestrado em Ciência da Computação). Universidade Federal de Santa Catarina. Florianópolis. 1995.

NETO, J. D. O.; FERNANDES, S. R. Metodologia de projeto para descrição da arquitetura IPNoSys em VHDL. In: VI Escola Potiguar de Computação e suas Aplicações. **Anais**... 2013. pp. 91-96.

NOBRE, C. A. **Avaliação da Execução de Aplicações Orientadas a Dados na Arquitetura de Redes em Chip IPNoSys.** 2012. 63 f. Dissertação (Mestrado em Sistemas e Computação)

– Departamento de Informática e Matemática Aplicada. Universidade Federal do Rio Grande do Norte. Natal. 2012.

PUTTMANN, C.; NIEMANN, J.-C.; PORRMANN, M.; RUCKERT, U., GigaNoC - *A Hierarchical Network-on-Chip for Scalable Chip-Multiprocessors*, *Digital System Design Architectures, Methods and Tools.* In: *10th Euromicro Conference.* **Proceedings**... 2007. p. 495-502.

REGO, R. S. L. S. **Projeto e Implantação de uma Plataforma MP-SoC usando SystemC**. 2006. 142 f. Dissertação (Mestrado em Sistemas e Computação) – Departamento de Informática e Matemática Aplicada. Universidade Federal do Rio Grande do Norte. Natal. 2006.

REINBRECHT, C. R. W. **Desenvolvimento e avaliação de redes-em-chip hierárquicas e reconfiguráveis para MPSoCs**. 2012. 119 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática. Universidade Federal do Rio Grande do Sul. Porto Alegre. 2012.

ROSE, C. A. F. de; NAVAUX, P. O. A. **Arquiteturas Paralelas**. Porto Alegre: Bookman, 2008. 151 p.

SEIFI, M. R.; ESHGHI, M., *A clustered NOC in group communication.* In: *IEEE Region 10 Conference, TENCON*. **Proceedings**... 2008. p.1-5.

STALLINGS, W. **Arquitetura e Organização de Computadores**. Quinta edição. São Paulo: Pearson, 2006. 786 p.

TANENBAUM, A. S. **Organização Estruturada de Computadores**. Quinta edição. Pearson, 2006. 464 p.

TORRES L.; BENOIT, P.; SASSATELLI, G.; ROBERT, M.; CLERMIDY, F.; PUSCHINI, D. *An introduction to multi-core system on chip trends and challenges*. In: *Multiprocessor system-on-chip: hardware design and tool integration.* **Proceedings**... 2011. p 1-21.

VIANA, A. L. ; FERNANDES, S. R. . *MPSoC with IPNoSys as Processing Element*. *In: 3th Workshop on Circuits and Systems Design*, Curitiba. **Anais**... 2013.

WOLF, W.; JERRAYA, A. ***Multiprocessor System-on-Chip***. Elsevier, 2005. 608 p.

WOLF, W.; JERRAYA, A.; MARTIN, G. *Multiprocessor system-on-chip (mpsoc) technology*. IEEE *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **Proceedings**... 2008. p. 1701-1713.

WOSZEZENKI, C. R. **Alocação de Tarefas e Comunicação Entre Tarefas em MPSoCs**. 2007. 121 f. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Informática. Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre. 2007.

ZEFERINO, C. A. Introdução às redes-em-chip. In: V Escola de Microeletrônica Sul (livro texto). Porto Alegre: **SBC**, 2003a, p. 93-104.

ZEFERINO, C. A. **Redes-em-*Chip***: Arquiteturas e Modelos para Avaliação de Área e Desempenho. 2003. 242 f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática. Universidade Federal do Rio Grande do Sul. Porto Alegre. 2003b.

ZEFERINO, C.A.; SUZIN, A. A. SoCIN: A Parametric and Scalable Network-on-Chip. In: Proceedings of the 16th symposium on Integrated circuits and systems design, **IEEE Computer Society**. 2003. pp. 169-174.

ZHENG, Liu; JUEPING, Cai; MING, Du; LEI, Yao; ZAN, Li. *Hybrid Communication Reconfigurable Network on Chip for MPSoC*. In: *24th IEEE International Conference, Advanced Information Networking and Applications* (AINA). **Proceedings**... 2010. p.356-361.