**ROSELI DA SILVEIRA UHLENDORF**

# MPSOC PARA AVALIAÇÃO DO DESEMPENHO DE UMA REDE-EM-CHIP EM FPGA

Itajaí (SC), agosto 2016


**UNIVERSIDADE DO VALE DO ITAJAÍ**

**CURSO DE MESTRADO ACADÊMICO EM COMPUTAÇÃO APLICADA**


# MPSOC PARA AVALIAÇÃO DO DESEMPENHO DE UMA REDE-EM-CHIP EM FPGA

por


Roseli da Silveira Uhlendorf


Dissertação de mestrado apresentada como requisito parcial à obtenção do grau de Mestre em Computação Aplicada.
Orientador: Cesar Albenes Zeferino, Dr.


Itajaí (SC), agosto 2016

# MPSOC PARA AVALIAÇÃO DO DESEMPENHO DE UMA REDE-EM-CHIP EM FPGA

Roseli da Silveira Uhlendorf

Agosto 2016

Orientador: Cesar Albenes Zeferino, Dr.

Área de Concentração: Computação Aplicada

Linha de Pesquisa: Sistemas Embarcados e Distribuídos

Palavras-chave: NoC, MPSoC, FPGA, Avaliação de Desempenho.

Número de páginas: 136

# RESUMO

Com o aumento do nível de integração de componentes em silício, se tornou possível a construção de sistemas com múltiplos elementos de processamento em uma única pastilha, os quais são denominados SoCs (System-on-Chip). Com o aumento da densidade de núcleos nos SoCs as arquiteturas de comunicação precisam ser reutilizáveis e oferecer paralelismo de comunicação e largura de banda escalável. Porém, essas características não são obtidas simultaneamente nos barramentos compartilhados tradicionalmente utilizados nesses sistemas. Para atender a esses requisitos, foi proposto o uso das arquiteturas de comunicação chaveada, denominadas Rede-em-Chip ou NoC (Network-on-Chip), as quais permitem o compartilhamento de seus canais por diferentes comunicações. No âmbito do projeto de uma NoC, uma das atividades principais é a avaliação do seu desempenho e das aplicações implementadas sobre essa rede. O Laboratório de Sistemas Embarcados e Distribuídos da Universidade do Vale do Itajaí tem realizado estudos visando disponibilizar uma plataforma para avaliação de desempenho da NoC SoCIN (System-on-Chip Interconnection Network) em FPGA (Field-Programmable Gate Array). Os primeiros trabalhos incluíram o desenvolvimento de geradores de tráfego para essa plataforma, os quais foram baseados em soluções em software embarcado e em hardware, mas apresentaram limitações quanto ao desempenho e à flexibilidade para geração de tráfego. Nesta dissertação, foi desenvolvida uma plataforma MPSoC (Multi-Processor System-on-Chip) para avaliação do desempenho da NoC SoCIN em FPGA com o uso de processadores programáveis de 32 bits e de uma biblioteca de comunicação baseada em troca de mensagens MPI (Message Passing Interface). A plataforma oferece flexibilidade para geração de tráfego e também para a implementação de aplicações paralelas. Essa plataforma foi sintetizada em FPGA e os resultados experimentais mostram que a plataforma é capaz de executar um experimento em tempo inferior ao de um simulador com precisão de ciclos rodando em um computador.

# MPSOC FOR PERFORMANCE EVALUATION OF A NETWORK-ON-CHIP IN FPGA

Roseli da Silveira Uhlendorf

Agosto 2016

Advisor: Cesar Albenes Zeferino, Dr.

Area of Concentration: Applied Computer Science

Research Line: Embedded and Distributed Systems

Keywords: NoC, MPSoC, FPGA, Performance Evaluation

Number of pages: 136

# ABSTRACT

The increasing in the level of integration of components on silicon became possible to construct systems with multiple processing elements on a single chip, which are known as SoCs (System-on-Chip). With the increasing in the density of cores in SoCs, the communication architectures used to interconnect them needs to be reusable and offer parallelism and scalable bandwidth. However, these features are not available simultaneously in traditional shared bus architectures. To meet these requirements, it was proposed the used of the Network-on-Chip (NoC) approach, which enables the sharing of its links by different communications. Within the design framework of a NoC, one of the main activities is the assessment of its performance and of the applications implemented on this network. The Laboratory of Embedded and Distributed Systems of the University of Vale do Itajaí has conducted studies aiming at providing a platform for performance evaluation of SoCIN (System-on-Chip Interconnection Network) NoC on FPGA (Field Programmable Gate Array). The first works included the development of traffic generators to that platform, which were based on embedded software and solutions in hardware, but presented limitations regarding performance and flexibility for traffic generation. In this work, we developed a MPSoC platform (Multi -Processor System-on- Chip) to evaluate the performance of NoC SoCIN in FPGA using programmable processors 32 bit and a communication library based on the exchange of messages MPI (Message Passing Interface). The platform offers flexibility for traffic generation and also for the implementation of parallel applications due to the use of MPI. The platform was synthesized in FPGA and the experimental results show that it is able to run an experiment faster than a cycle-accurate NoC simulator running on a computer.

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AceNoCs | Accelerated Emulation Platform for NoCs |
| AHB | Advanced High-performance Bus |
| ALUT | Adaptative Look-Up Table |
| AMBA | Advanced Microcontroller bus Architecture |
| ARM | Advanced RISC Machine |
| ASIC | Application Specific Integrated Circuit |
| BE | Best Effort |
| BT | Broadcast transmission |
| CBR | Constant Bit Rate |
| CCM | Communication and Control Module |
| CE | Communication Element |
| CI | Circuit Integrated |
| CNF | Chaos Normal Form |
| CPU | Central Processing Unit |
| CRC | Cyclic Redundancy Check |
| DSP | Digital Signal Process |
| DVFS | Dynamic Voltage Frequency Scaling |
| ECB | Emulation Control Block |
| EDK | Embedded Development Kit |
| EG | Event Generator |
| FCFS | First Come First Served |
| FFT | Fast Fourier Transform |
| FIFO | First-In, First-Out |
| FPGA | Field-Programmable Gate Array |
| GALS | Globally Asynchronous Locally Synchronous |
| GPP | General Propose Processor |
| GT | Guaranteed throughput |
| HardNoC | Hardware Network-on-Chip |
| HDL | Hardware Description Language |
| HW | Hardware |
| IB | Internal Bus |
| IP | Intellectual Property |
| JPEG | Joint Photographic Experts Group |
| JTAG | Joint Test Action Group |
| LEDS | Laboratory of Embedded and Distributed System |
| LPS | Local Processor System |
| LRS | Least Recently Served |
| MAC | Media Access Control |
| MAS | Monitoring service Access point |
| MIMD | Multiple Instruction Multiple Data |
| MISD | Multiple Instruction Single Data |
| MMS | Pipeline of Matrix Multiplications |
| MNI | Monitoring Network Interface |
| MP | Multi-Processor |
| MPI | Message Passing Interface |

| | |
|---|---|
| MPoPC | Multi-Processor on Programmable Chip of Soft Multiprocessor |
| MPSoC | Multiprocessor System-On-Chip |
| NI | Network Interface |
| NM | Node Monitoramento |
| NoC | Network-On-Chip |
| NP | Nodo Processamento |
| NS | Nodo Supervisor |
| NTNT | Node-to-Node Transmission |
| OCIN | On-Chip Interconnect Network |
| OCP | Open Core Protocol |
| OCTG | On-line Configurable Traffic Generator |
| OPB | On-chip Peripheral Bus |
| OTF | Open Trace Format |
| ParIS | Parameterizable Interconnect Switch |
| PAT | Payload abstraction technique |
| PE | Processing Element |
| PLB | Processor Local Bus |
| PLD | Programmable Logic Device |
| PPS | Processor Protection System |
| QoS | Quality of Service |
| RE | Resource Element |
| RISC | Reduced Instruction Set Computer |
| RNI | Resource Network Interface |
| RTL | Register Tranfer Level |
| SAF | Store and Forward |
| SIMD | Single Instruction, Multiple Data |
| SISD | Single Instruction, Single Data |
| SoC | System-on-Chip |
| SoCIN | System-on-Chip Interconnection Network |
| SP | Single Processor |
| SPP | Single Purpose Processor |
| SRAM | Static Random Access Memory |
| SW | Software |
| TG | Traffic Generator |
| TM | Traffic Meter |
| TSIM | Timing Simulation |
| UDP/IP | User Datagram Protocol/Internet Protocol |
| UNIVALI | Universidade do Vale do Itajaí |
| USB | Universal Serial Bus |
| UVM | Universal Verification Methodology |
| VC | Virtual Cut |
| VCT | Virtual Cut Through |
| VFI | Voltage Frequency Island |
| VHDL | Very High Speed Integrated Circuits Hardware Description Language |
| WH | Wormhole |
| XUP | Xilinx University Program |

# SUMÁRIO

# 1 INTRODUÇÃO

Ao longo das últimas três décadas, o avanço tecnológico para fabricação dos semicondutores tem permitido cada vez mais a redução de área de silício na construção de circuitos integrados e o aumento da densidade de componentes. Na década de 80, a maioria dos circuitos integrados (CIs) complexos era composta por dezenas de milhares de transistores, e nesta última década já é possível encontrar CIs com centenas de milhões de transistores, acompanhando o aumento do nível de integração definido pela Lei de Moore[1] (WANG; STROUD; TOUBA, 2008, p. 1; DUATO; YALAMANCHILI; NI, 2003, p. 2).

O aumento do nível de integração de componentes em silício tem permitido a construção de sistemas com múltiplos componentes, como processadores, controladores e memória, integrados em uma única pastilha, os quais são denominados sistemas integrados ou SoCs (Systems-on-Chip). A integração de um sistema em um único chip permite obter aumento de desempenho, redução de consumo de energia, redução de custo e aumento de confiabilidade (WANG; STROUD; TOUBA, 2008, p. 1; DUATO; YALAMANCHILI; NI, 2003, p. 2).

Dentro do contexto de SoCs, as arquiteturas multiprocessadas (MPSoCs – MultiProcessor SoCs) têm sido uma das alternativas para a realização de sistemas eletrônicos digitais de alta complexidade. Funcionalidades que antes eram executadas exclusivamente por hardware, passam a ser executadas em software (HUBNER; BECKER, 2011, p. v).

Para minimizar o tempo de projeto, os SoCs são construídos utilizando-se componentes de hardware reutilizável, os quais são denominados núcleos (do inglês, *cores*). Os núcleos de um SoC podem ser interconectados por meio de conexões exclusivas (canais ponto-a-ponto) ou compartilhadas (barramento). A primeira abordagem oferece melhor desempenho, mas baixa reusabilidade. Já a segunda abordagem possui desempenho inferior, mas uma reusabilidade maior, o que é prioritário para minimizar o tempo de projeto. Dada à alta competitividade, a redução da janela de mercado e a necessidade de atender a requisitos de *time-to-market* (tempo entre início do projeto até a disponibilidade ao mercado), o projeto de sistemas complexos tem priorizado soluções que favoreçam a reusabilidade de componentes de hardware ou de software. Devido à sua

1 Lei de Moore: Em 1965, Gordon Moore, um dos fundadores da Intel, previu que o número de transistores dos chips teria um aumento de 100%, pelo mesmo custo, a cada período de 18 meses (WANG; STROUD; TOUBA, 2008, p. 2).

reusabilidade, o barramento foi escolhido como arquitetura preferencial para interconexão de componentes em SoCs (SANTO; ZEFERINO; SUSIN, 2004).

No entanto, com o aumento da densidade de núcleos nos SoCs (para várias dezenas a centenas de unidades), esses sistemas precisam de arquiteturas de comunicação reutilizáveis (como o barramento), mas que ofereçam paralelismo de comunicação e largura de banda escalável, de modo que o seu desempenho aumente com o tamanho do sistema. Esses atributos não são oferecidos simultaneamente, nem pelos canais ponto-a-ponto e nem pelo barramento. Para atender a esses requisitos, na última década, foi proposto o uso de uma estrutura de comunicação formada por canais ponto-a-ponto chaveados por roteadores de modo a permitir o compartilhamento de seus canais por diferentes comunicações e atender aos requisitos dos futuros SoCs. Essa estrutura, foi denominada Rede-em-Chip ou NoC (Network-on-Chip) (JANTSCH; TENHUNEN, 2003, p. 9).

Um exemplo de NoC é a SoCIN (SoC Interconnection Network). Essa rede tem como característica principal possuir em sua estrutura um núcleo de roteador parametrizável, cuja largura dos canais, profundidade dos *buffers* e mecanismos de comunicação podem ser configurados em função dos requisitos do sistema. Isso possibilita ao desenvolvedor gerar uma instância da rede SoCIN que melhor atenda aos requisitos de custo e desempenho da aplicação alvo (SANTO; ZEFERINO; SUSIN, 2004; ZEFERINO; SUSIN, 2003). Essa rede é utilizada como NoC de referência neste trabalho.

## 1.1 PROBLEMA DE PESQUISA

O desenvolvimento de uma NoC é constituído por etapas de especificação, projeto arquitetural, implementação, validação (simulação e física) e avaliação (desempenho, custos de área e energia). A etapa de validação e avaliação, foco deste projeto, pode ser realizada por simulação computacional ou por emulação física, utilizando geradores de tráfego (TGs – Traffic Generators) para gerar fluxo de comunicação[2].

Para auxiliar o desenvolvimento do projeto na etapa de avaliação de NoCs, o Laboratório de Sistemas Embarcados e Distribuídos (LEDS – Laboratory of Embedded and Distributed Systems)

2 Um fluxo de comunicação representa uma comunicação realizada entre dois nodos (fonte e destino) e envolve a transferência de vários pacotes. Exemplos incluem fluxos que representem comunicações entre uma interface de E/S e um processador ou entre um processador e um coprocessador (JATSCH; TENHUNEN, 2003, p. 92).

da Universidade do Vale do Itajaí (Univali) tem realizado estudos visando disponibilizar uma plataforma para avaliação de desempenho da rede SoCIN em dispositivo lógico programável do tipo FPGA (Field-Programmable Logic Array– Matriz Lógica Programável em Campo).

A primeira pesquisa foi realizada por Pereira (2008) com o enfoque de solucionar o problema de baixa flexibilidade na modelagem do tráfego das emulações dos geradores de tráfego em hardware e alto custo computacional. O projeto consistiu no desenvolvimento de um gerador de tráfego e de um medidor de tráfego (TM – Traffic Meter) sintetizáveis para uma plataforma de validação e avaliação de Redes-em-Chip, ambos baseados em uma tecnologia de processador de propósito único (SPP – Single Purpose Processor[3]) não programável. A Figura 1 apresenta a visão geral da plataforma de validação e avaliação.

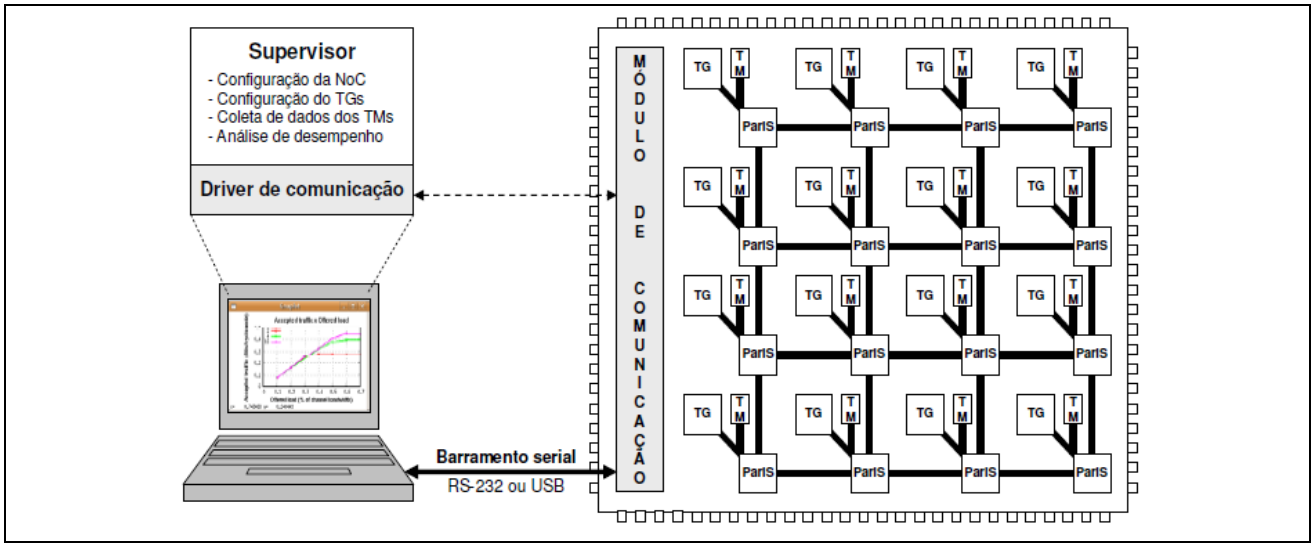


Figura 1. Plataforma de avaliação de desempenho baseada em SPP
Fonte: Pereira (2008).

O gerador de tráfego foi modelado em linguagem de descrição de hardware (HDL – Hardware Description Language) e sintetizado em FPGA, tendo como características: (*i*) baixa latência para geração de tráfego; (*ii*) pouca flexibilidade para modificações (qualquer alteração referente à topologia acarreta em modificação de hardware, necessitando de uma nova síntese do sistema) e; (*iii*) geração de fluxos de comunicação diferentes limitada a quatro (o que impede a

3 Um Single-Purpose Processor (SPP) é um processador de propósito único, desenvolvido para executar exatamente uma aplicação específica. Apresenta em sua arquitetura básica, lógica controle, registradores de estado, unidades aritméticas e memória de dados. Não é programável e, portanto, oferece pouca flexibilidade (VAHID; GIVARGIS, 2001, p. 1-8).

emulação de padrões de tráfego com muitos destinatários por nodo, como, por exemplo, o tráfego uniforme em sistemas com mais de cinco nodos). Constatou-se que, apesar da observabilidade limitada da implementação em hardware em relação a software, a possibilidade de realizar a validação física e o ganho de desempenho, que neste trabalho foi 400 vezes maior que em uma implementação em software para a rede avaliada, justificam o uso desta abordagem.

A segunda pesquisa foi realizada por Frantz (2008) com o objetivo de disponibilizar uma plataforma física para avaliação de NoCs baseada em geradores de tráfego flexíveis por meio do uso de hardware que permitisse a modelagem em alto nível e que produzisse, automaticamente, a configuração dos geradores com um baixo custo computacional. O projeto consistiu no desenvolvimento de um modelo sintetizável de gerador de tráfego para NoCs para emulação de tráfego em hardware. Também foi desenvolvido um medidor de tráfego sintetizável que permitiu extrair dados da resposta da rede ao tráfego injetado pelos geradores, conforme apresentado na Figura 2. O gerador e medidor de tráfego foram baseados em software embarcado, utilizando um processador de propósito geral (GPP – General Purpose Processor[4]), o PicoBlaze (fabricado pela Xilinx), o que resultou em uma maior flexibilidade, especialmente para a geração de fluxos com características diferentes ou para múltiplos destinatários.

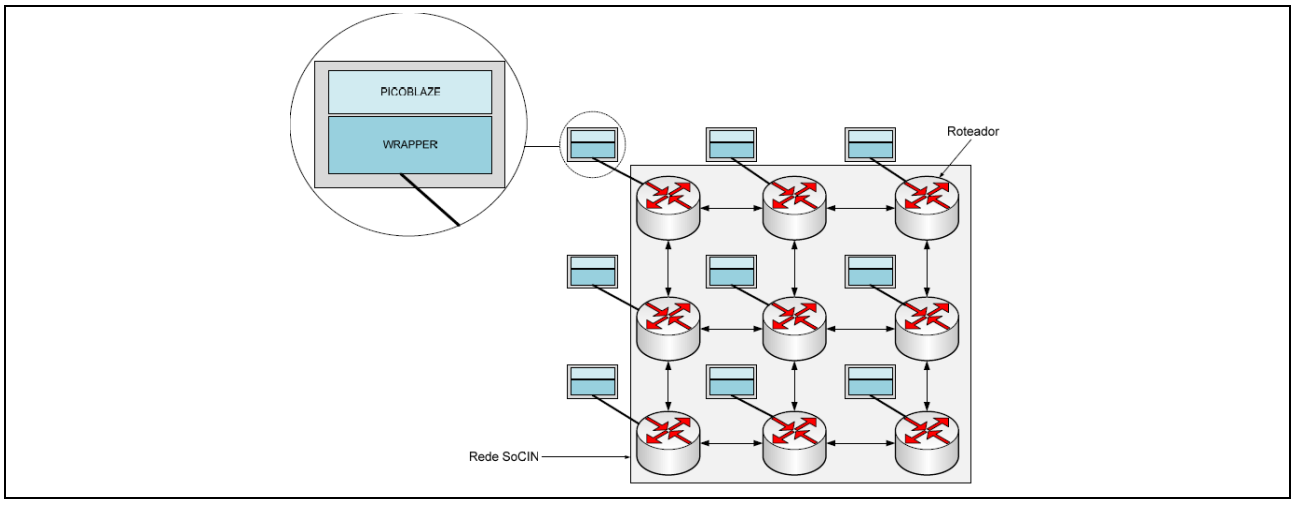


Figura 2. Plataforma de avaliação de desempenho baseada em um GPP
Fonte: Frantz (2008).

4Um General-Purpose Processor (GPP) é processador de propósito geral programável, também conhecido como CPU (Central Processing Unit) ou microprocessador. Apresenta em sua arquitetura básica ALU (Arithmetic-Logic Unit), memória de programa, memória de dados, registrador de instrução e contador de programa. Um GPP oferece alta flexibilidade para a implementação de novas funcionalidades (VAHID; GIVARGIS, 2001, p. 1-6).

Porém, como o PicoBlaze é um processador de 8 bits, a latência para a geração de tráfego é alta, especialmente se considerado o uso de uma NoC com canais de 32 bits, pois são necessários quatro ciclos de relógio para construir uma palavra do pacote a ser injetado na rede SoCIN. Para lidar com este problema é necessário que o processador PicoBlaze opere a uma frequência de relógio maior que a da rede.

Por último, Pizzoni (2010) realizou a integração do gerador de tráfego baseado em hardware de Pereira (2008) a uma rede, adicionando: (*i*) módulo de comunicação externa via RS-232 para configuração da plataforma em hardware; (*ii*) controle da execução de experimentos; e (*iii*) coleta e análise dos dados. O projeto consistiu no desenvolvimento dos módulos, Supervisor (módulo de configuração da NoC), CCM (Communication and Control Module – Módulo de Controle e Comunicação) e ECB (Emulation Control Block – Bloco de Controle de Emulação), conforme ilustrado na Figura 3.

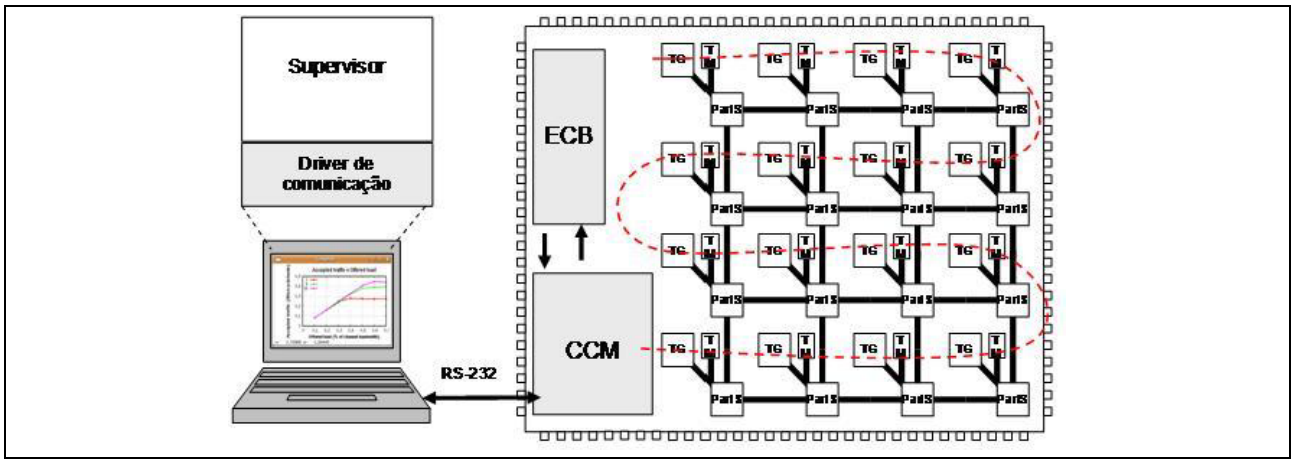


Figura 3. Módulos para comunicação com o computador
Fonte: Pizzoni (2010).

No entanto, Pizzoni encontrou dificuldades na validação da plataforma, na qual o CCM, quando integrado à plataforma de hardware, não funcionou conforme especificado, não permitindo a integração e validação dos módulos com Supervisor e ECB na plataforma. Assim, os experimentos previstos para avaliação do ganho obtido na emulação em hardware, quando comparada com a simulação em software, não puderam ser realizados.

Além desses trabalhos, foi desenvolvida uma ferramenta de avaliação do desempenho da rede SoCIN baseada na simulação de modelos descritos em SystemC RTL (Register Transfer

Level) (ZEFERINO et al., 2007; BRUCH; PIZZONI; ZEFERINO, 2009). Essa ferramenta, denominada BrownPepper, oferece observabilidade maior que as soluções baseadas na emulação em FPGA, porém consome mais tempo para realização de um experimento de avaliação de desempenho. Uma segunda versão dessa ferramenta, chamada RedScarf, foi desenvolvida por SILVA (2014) e esta versão tem como principais vantagens a execução *multithreaded* (múltiplos simuladores executados nos múltiplos núcleos do computador, simultaneamente), o que reduz o tempo de realização dos experimentos, e o suporte a múltiplas plataformas de sistema operacional (Windows, Linux e MacOS X).

O problema alvo desta dissertação consistiu em determinar como disponibilizar uma plataforma para emulação de tráfego e avaliação do desempenho da rede SoCIN em hardware que apresente alta flexibilidade para geração de tráfego e viabilize o desenvolvimento de aplicações MPSoC. Buscou-se encontrar uma solução para contornar as limitações dos trabalhos citados acima e para reduzir o tempo de realização de um experimento de avaliação de desempenho em relação à simulação em SystemC. Também se buscou disponibilizar uma plataforma MPSoC, baseada na SoCIN, com flexibilidade para futura implementação de aplicações baseadas em processamento paralelo.

Esta dissertação buscou responder as seguintes perguntas:

1. Quais são as vantagens e as limitações da plataforma de avaliação de desempenho proposta em relação às soluções similares em FPGA, em particular aos trabalhos desenvolvidos por Pereira (2008) e Frantz (2008)?
2. Qual é o ganho da plataforma em relação ao tempo de execução de um experimento quando comparado com a simulação em SystemC?
3. Como superar as limitações dos trabalhos desenvolvidos por Pereira (2008) e Frantz (2008)?

### 1.1.1 Solução Proposta

Para tratar o problema acima identificado, foi proposto o desenvolvimento de uma plataforma MPSoC em FPGA com gerador de tráfego baseado em um processador embarcado programável de 32 bits, no entendimento de que tal arquitetura gasta menos ciclos para geração de

tráfego do que a solução desenvolvida por Frantz (2008) e oferece flexibilidade superior à solução obtida por Pereira (2008). Além disso, a plataforma MPSoC em FPGA deve realizar experimentos em menor tempo que um simulador em SystemC.

A solução proposta baseia-se nas seguintes hipóteses:

1. Um processador de propósito geral de 32 bits requer menos ciclos que um processador de 8 bits para a geração de tráfego;
2. Um processador de propósito geral de 32 bits oferece maior flexibilidade para geração de fluxos que uma implementação baseada em processador de propósito específico;
3. Um processador de propósito geral de 32 bits consome maior área de silício que implementações baseadas em um processador de 8 bits ou em processador de propósito específico;
4. A emulação da NoC em FPGA baseada em um processador de propósito geral de 32 bits permite realizar experimento de avaliação de desempenho em um tempo menor que o obtido pela simulação em SystemC.

## 1.2 OBJETIVOS

### 1.2.1 Objetivo Geral

Desenvolver uma plataforma MPSoC baseada em processador programável para avaliação do desempenho da NoC SoCIN em FPGA.

### 1.2.2 Objetivos Específicos

1) Proporcionar flexibilidade para geração de tráfego na plataforma de avaliação da NoC SoCIN em FPGA;

2) Reduzir a latência para geração de tráfego na plataforma de avaliação da NoC SoCIN em FPGA; e

3) Reduzir o tempo para avaliação do desempenho da NoC SoCIN em relação à simulação em SystemC.

## 1.3 METODOLOGIA

### 1.3.1 Classificação

Esta pesquisa utiliza o método hipotético-dedutivo, assumindo-se as hipóteses de que a solução proposta reduz a latência e aumenta a flexibilidade para geração de tráfego na plataforma de avaliação de desempenho em NoCs quando comparada com as soluções anteriores, embora aumentando o custo de silício. Ela caracteriza-se como pesquisa aplicada, pois, a partir da compilação dos dados fornecidos pelo medidor de tráfego, é possível avaliar a solução proposta. Também pode ser classificada como pesquisa quantitativa porque é feita uma comparação quantitativa com as implementações de Pereira (2008) e Frantz (2008), de forma a apresentar a diferença quanto à latência para geração de tráfego (número de ciclos gastos) e custos de silício, além de uma comparação quantitativa com um simulador SystemC. Além disso, a pesquisa é classificada como exploratória, pois se baseia em um estudo preliminar da plataforma de avaliação de desempenho em NoCs, como também dos trabalhos de Pereira (2008), Frantz (2008) e Pizzoni (2010) de modo a obter uma maior compreensão da estrutura do sistema.

### 1.3.2 Procedimentos

Para atingir os objetivos propostos neste trabalho foram utilizados os procedimentos técnicos de pesquisa bibliográfica e pesquisa experimental. Por meio da pesquisa bibliográfica, foram obtidos conhecimentos para o melhor entendimento dos requisitos da plataforma proposta. A técnica de pesquisa experimental foi utilizada para extrair informações para avaliação do sistema para com isso disponibilizar uma plataforma física totalmente operacional.

## 1.4 ESTRUTURA DA DISSERTAÇÃO

Este documento está organizado em seis capítulos correlacionados. O Capítulo 1 apresentou por meio de sua contextualização o tema proposto neste trabalho. Foram estabelecidos os resultados esperados por meio da definição de seus objetivos e apresentadas às delimitações do trabalho permitindo uma visão clara do escopo proposto. O Capítulo 2 apresenta a fundamentação teórica

com uma síntese dos conceitos de sistemas integrados (SoCs), arquitetura de Redes-em-Chip (NoC), avaliação de desempenho, geração de tráfego e MPSoCs. Continuando, o Capítulo 3 apresenta uma pesquisa dos trabalhos relacionados ao tema desta dissertação, permitindo identificar informações sobre técnicas de geração, medição e monitoramento de tráfego, como também os trabalhos realizados na área de MPSoCs. Já o Capítulo 4 apresenta uma visão geral do trabalho proposto abordando os itens: projeto arquitetural, fluxo de projeto, requisitos funcionais e não-funcionais. A partir do projeto implementado, o Capítulo 5 apresenta as etapas de verificação, avaliação e experimentação como também os resultados obtidos. Por fim, o Capítulo 6 apresenta as conclusões do trabalho.

# 2 REFERENCIAL TEÓRICO

Neste capítulo, é apresentada uma revisão da literatura sobre os temas fundamentais para o entendimento deste trabalho. Primeiramente, na Seção 2.1, são apresentados os conceitos, características, vantagens e desvantagens de Sistemas-em-Chip (SoCs). Em seguida, na Seção 2.2, são apresentadas às características das arquiteturas de Redes-em-Chip (NoCs). Já na Seção 2.3, é abordado com mais detalhes a avaliação de desempenho de NoCs. Na Seção 2.4, são apresentadas as informações das arquiteturas MPSoCs. Por fim, a Seção 2.5 apresenta as considerações sobre os assuntos apresentados neste capítulo.

## 2.1 SISTEMAS-EM-CHIP

Em meados dos anos 90, os projetos antes desenvolvidos com filosofia em *chip-set* começaram a ser implementados em sistema embarcados baseados em SoCs, permitindo a integração de vários processadores, memórias, interface para periféricos e até blocos dedicados em um único chip. O SoC é uma implementação em ASIC (Application Specific Integrated Circuit– Circuito Integrado de Aplicação Específica), PLD (Programmable Logic Device – Dispositivo Lógico Programável) ou baseada no reuso de núcleos ou blocos IP (Intelectual Property – Propriedade Intelectual) como, por exemplo, sistemas microprocessados (RAJSUMAN, 2000, p. 3).

De acordo com a forma na qual os núcleos são disponibilizados ao integrador do SoC, eles podem ser classificados em:

- Soft-Core: Consiste na disponibilização dos blocos de núcleos na forma de uma descrição RTL (Register-Transfer Level– Nível de Transferência de Registrador) sintetizável ou em uma lista de ligações (*netlist*), ou seja, descritos em HDL que pode ser mapeada para diferentes processos de fabricação e requer síntese e leiaute. Apresenta alta reusabilidade, flexibilidade e portabilidade. Contudo, o tamanho e a velocidade dependem da tecnologia utilizada.
- Firm-Core: Consiste na disponibilização do núcleo com topologia e estrutura otimizada, disponibilizando melhor desempenho e melhor uso de área por meio do roteamento e posicionamento dos blocos do núcleo. Este tipo de núcleo pode ser parametrizável,

necessitando de leiaute. Apresenta tamanho e velocidade parcialmente previsível e média reusabilidade, flexibilidade e portabilidade**.**

- Hard-Core: Consiste da disponibilização na forma criptografada do núcleo já otimizado para desempenho, potência e tamanho. Mapeado para um processo específico, ou seja, implementado em nível de leiaute para uma tecnologia específica. Tamanho e velocidade são previamente determinados, mas apresenta baixas reusabilidade, flexibilidade e portabilidade (RAJSUMAN, 2000, p. 5).

O cenário para desenvolvimento de SoCs, pode ser classificado de três formas: (*i*) desenvolvimento ASIC com fabricante: este desenvolve todos os componentes do SoCs como também os fabrica; (*ii*) desenvolvimento integrado: ocorre quando o desenvolvimento dos componentes não pertence ao fabricante, deixando apenas a ele a responsabilidade de fabricar o SoCs; e (*iii*) desenvolvimento desktop: o desenvolvimento e a fabricação do SoC são totalmente independentes das fábricas (RAJSUMAN, 2000, p. 3).

A Figura 4 ilustra a taxa de crescimento de complexidade dos núcleos (IPs), software embarcado e dos próprios sistemas no período de 1995 a 2000 (RAJSUMAN, 2000, p. 6). Observa-se que com o aumento na integração de núcleos e o uso de sistema embarcados em SoCs, a complexidade no desenvolvimento também cresceu.

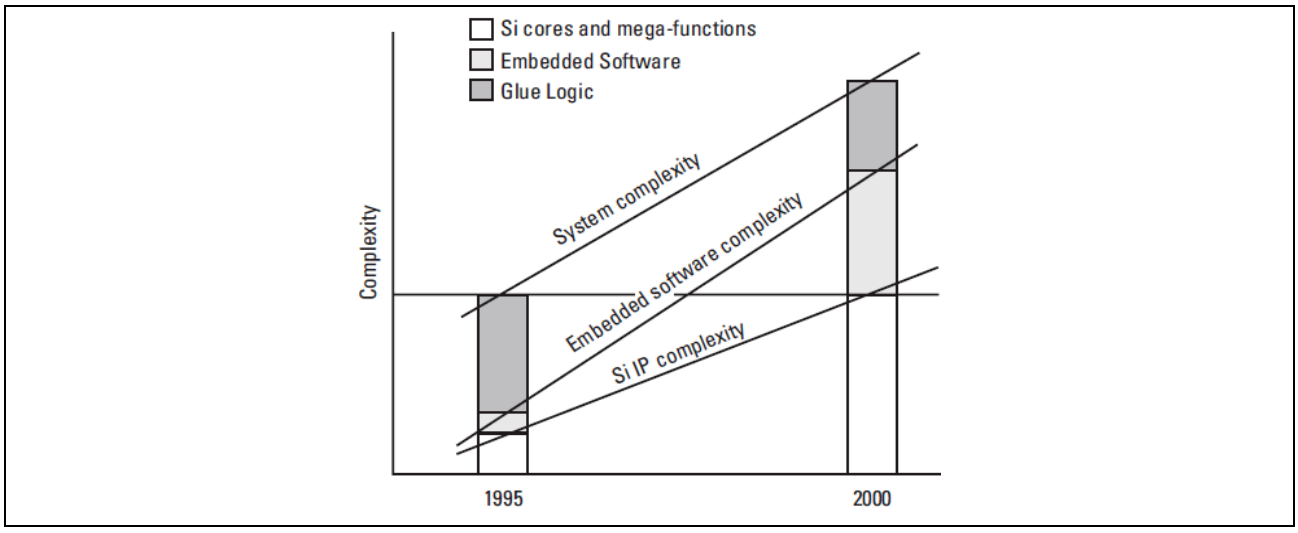


Figura 4. Taxa de crescimento da complexidade dos núcleos
Fonte: Rajsuman (2000).

A interconexão entre núcleos em um SoCs pode ser implementada por meio de canais ponto-a-ponto e/ou barramento (multiponto). Porém, na medida em que se aumenta a quantidade de núcleos em um SoC, essa interconexão se torna cada vez mais complexa (no caso do ponto-a-ponto) ou o desempenho do sistema diminui (no caso do barramento) (COPOLLA et al., 2009, p. xiii, p. 41).

A interconexão baseada em canais ponto-a-ponto, exemplificada na Figura 5(a), apresenta como vantagem um melhor desempenho, devido às comunicações entre diferentes fontes e destinos serem realizadas por canais exclusivos e por apresentar alto grau de paralelismo. No entanto, tal solução apresenta como desvantagens, baixa escalabilidade, reusabilidade e flexibilidade por necessitar de um número excessivo de fios para o caso de se interligar um SoC com uma grande quantidade de núcleos.

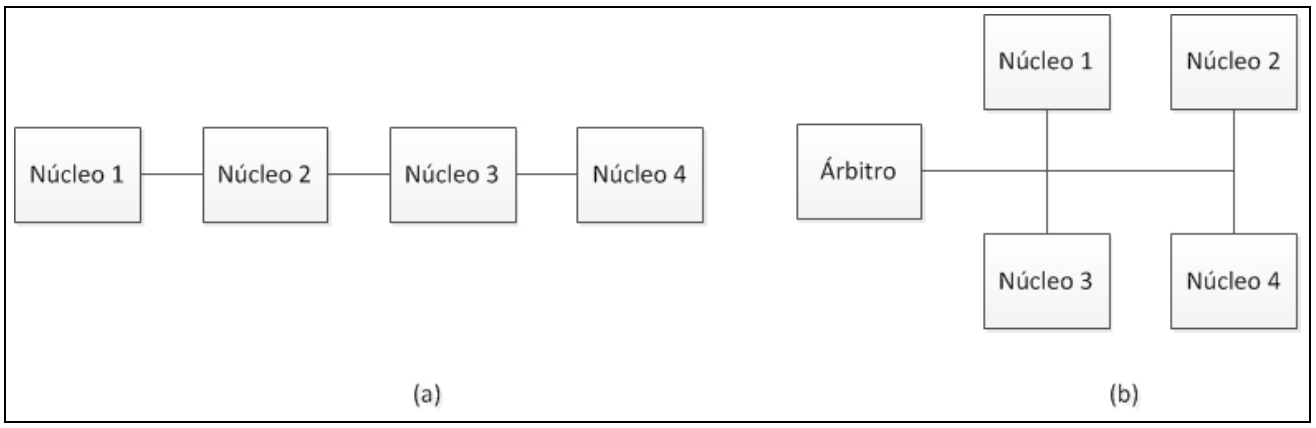


Figura 5. Interconexão SoC: (a) Ponto-a-Ponto; (b) Multiponto
Fonte: Adaptado de Zeferino (2003).

Já a interconexão barramento, exemplificada na Figura 5(b), apresenta como vantagens, baixo custo de implementação e alta reusabilidade, pois cada núcleo possui uma única porta para comunicação com todos os demais núcleos do sistema. Porém, os fios do canal compartilhado sofrem com o aumento da capacitância e da resistência parasitas a cada novo núcleo conectado ao barramento, degradando assim o seu desempenho (frequência máxima de operação) (COPOLLA et al., 2009, p. xiii, p. 41; ZEFERINO, 2003, p. 58). O grau de paralelismo nesta interconexão é baixo, pois apenas uma transação pode ser efetuada por vez.

Um comparativo entre as interconexões ponto-a-ponto e barramento é ilustrado no Quadro 1 (ZEFERINO, 2003, p. 58).

Quadro 1. Comparativo entre interconexões SoC

| **Característica** | **Ponto-a-Ponto** | **Barramento** |
|---|---|---|
| Carga capacitiva em cada fio | Menor | Maior |
| Consumo de energia em cada fio | Menor | Maior |
| Frequência de operação | Maior | Menor |
| Paralelismo em comunicação | Sim | Não |
| Largura de banda escalável | Sim | Não |
| Área | Depende do tamanho e da complexidade | Depende do tamanho e da complexidade |
| Reusabilidade | Não | Sim |

Fonte: Adaptado de Zeferino (2003, p. 58).

Para conseguir atender aos requisitos de desempenho e consumo de energia em SoCs, com um número cada vez maior de núcleos integrados, uma nova estrutura para comunicação foi proposta no início dos anos 2000: as Redes-em-Chip ou NoCs (Networks-on-Chip). As NoCs são escaláveis, reutilizáveis, apresentam eficiência na comunicação (multiplexando diferentes fluxos de comunicação) e interconexões curtas (JERGER; SHIUAN, 2009, p. 1).

## 2.2 CONCEITOS BÁSICOS SOBRE REDES-EM-CHIP

No futuro, os sistemas integrados em um chip irão conter bilhões de transistores, e dezenas de centenas de núcleos IPs. A eficiência das soluções que utilizam uma grande quantidade de núcleos IPs pode ser alcançada através de rede de interconexão on-chip.

As indústrias iniciaram diferentes desenvolvimentos baseados em NoCs tais como: Æthereal NoC (Philips), STNoC (STMicroelectronics) e 80-*core* NoC (Intel) (GEBALI; ELMILIGI; KHARASHI, 2009, p. 2).

O desenvolvimento de uma NoC consiste em realizar a especificação de cinco características: topologia, roteamento, controle de fluxo, arquitetura do roteador e arquitetura de enlace (JERGER; SHIUAN, 2009, p. 4). O tipo de núcleo, topologia e esquema de interconexão são informações importantes para determinar a eficiência da NoC em determinadas aplicações. As NoCs apresentam como características: (*i*) confiabilidade e eficiência no gerenciamento de energia; (*ii*) escalabilidade da largura de banda; (*iii*) reusabilidade; e (*iv*) decisões de roteamento distribuído (GEBALI; ELMILIGI; KHARASHI, 2009, p. 3).

### 2.2.1 Topologia

A topologia em uma NoC especifica a organização física da interconexão da rede como também determina a quantidade de roteadores que uma mensagem irá atravessar e a distância entre fonte e destino, informação necessária para o cálculo da latência da rede. O custo e a complexidade de implementação da topologia estão relacionados diretamente com dois fatores: (*i*) número de portas em cada nodo; e (*ii*) tipo da topologia (JERGER; SHIUAN, 2009, p. 29).

A topologia para uma NoC pode ser classificada em: direta, indireta e irregular (JERGER; SHIUAN, 2009, p. 32). A escolha da topologia da NoC pode ser realizada com o auxílio das métricas: (*i*) grau do nodo: refere-se ao número de portas em cada nodo; (*ii*) contagem de saltos: refere-se ao número de enlaces que a mensagem têm que atravessar até chegar ao seu destino; (*iii*) carga máxima do canal: refere-se ao número máximo de bits por segundos que pode ser injetado em cada nodo antes da saturação; e (*iv*) diversidade de caminho: refere-se à flexibilidade do algoritmo de roteamento em relação ao balanceamento de tráfego (JERGER; SHIUAN, 2009, p. 30).

A rede direta ou ponto-a-ponto é uma arquitetura que consegue resolver os problemas de escalabilidade de uma rede de médio compartilhamento, tendo como propriedade o aumento dos nodos (aumento na disponibilidade de largura de banda), *trade-off* (relação entre custo e conectividade) e alta conectividade (alto desempenho, alto consumo de energia e custo de área para implementação dos roteadores e enlaces). Para as redes diretas, a topologia ideal seria aquela em que cada nodo estaria conectado a todos os outros nodos, porém o custo e o tamanho da rede a tornaria inviável para implementação e as mensagens não passariam por nodos intermediários para encontrar o destino.

A arquitetura genérica do nodo de uma rede direta é constituída por processador, memória local, componentes funcionais e roteador, conforme ilustrado na Figura 6. Nesta arquitetura, o único componente que se mantém independente da aplicação é o roteador. Por esta razão, a rede direta também é conhecida como rede baseada em roteador (DUATO; YALAMANCHILI; NI, 2003, p. 13).

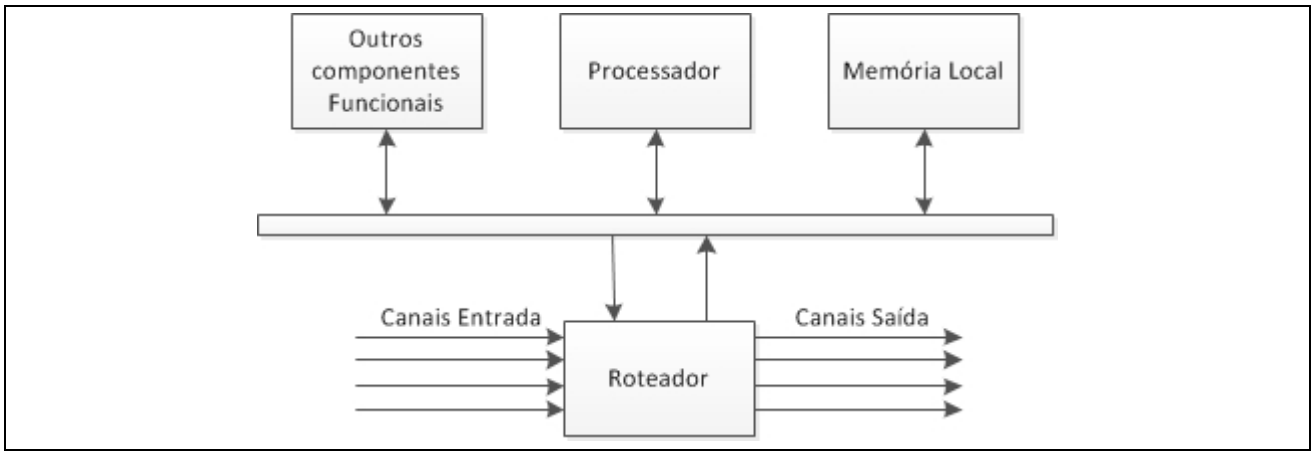


Figura 6. Arquitetura genérica do nodo de uma rede direta
Fonte: Adaptado Duato et al. (2003).

A topologia de rede direta pode ser classificada de acordo com sua estrutura, por exemplo: anel, malha 2D e toróide 2D, conforme ilustrado na Figura 7. A topologia em anel, apresentada na Figura 7(a), tem como limitações a escalabilidade e a redução de desempenho à medida que aumenta os nodos na rede, porém, por apresentar simetria nas bordas, consegue oferecer um melhor balanceamento de carga (PASRICHA; DUTT, 2008, p. 444). Já a topologia em malha 2D, ilustrada na Figura 7(b), por não apresentar simetria nos vértices, concentra alta demanda de tráfego no centro da rede. Nessa topologia, todos os enlaces tem o mesmo tamanho e cada nodo é conectado com outros quatro vizinhos exceto os nodos dos vértices (JERGER; SHIUAN, 2009, p. 33).

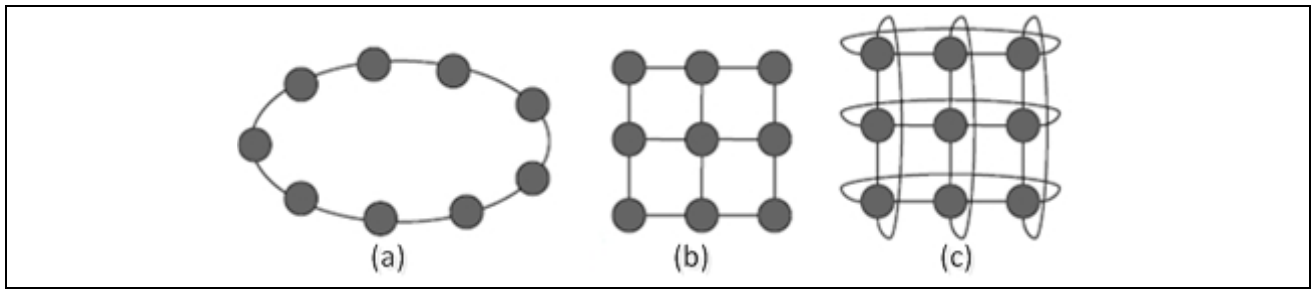


Figura 7. Redes diretas: (a) Anel; (b) Malha 2D; e (c) Toróide 2D
Fonte: Jerger et al. (2009).

A topologia toróide 2D, ilustrada na Figura 7(c), é semelhante a uma malha 2D com a diferença de que os nodos dos vértices são conectados aos seus opostos via conexões de fios. Nessa topologia, todos os nodos são conectados a outros quatro nodos. Uma conexão longa entre os nodos dos vértices pode acarretar em um atraso no sinal (PASRICHA; DUTT, 2008, p. 445).

Exemplos de NoCs que utilizam topologias diretas incluem: SoCIN (ZEFERINO; SUSIN, 2003), Spidergon (COPOLA et al., 2009, p. 93-100), Hermes, SoCBUS e Nostrum (PASRICHA; DUTT, 2008, p. 445).

As topologias de redes indiretas, ilustradas na Figura 8, utilizam as chaves para realizar um caminho de conexão entre dois nodos da rede, sendo que essas chaves não realizam processamento de informações. Os pacotes são chaveados indiretamente por meio de uma série de nodos intermediários de chaveamentos entre fonte e destino (PASRICHA; DUTT, 2008, p. 446).

A topologia de rede indireta pode ser classificada de acordo com sua estrutura, por exemplo: árvore gorda (*fat-tree*), borboleta (*butterfly*), Clos ou Benes. A topologia *fat-tree*, ilustrada na Figura 8(a), é uma rede indireta multiestágio em que as chaves são utilizadas para conectar um estágio de roteadores a outro. A topologia *butterfly*, ilustrada na Figura 8(b), oferece um bloqueio para a rede multiestágio, pois as informações podem ser temporariamente bloqueadas ou descartadas na rede se uma contenção ocorrer. A SPIN é um exemplo de rede NoC que utiliza topologia *fat-tree* (PASRICHA; DUTT, 2008, p. 446).

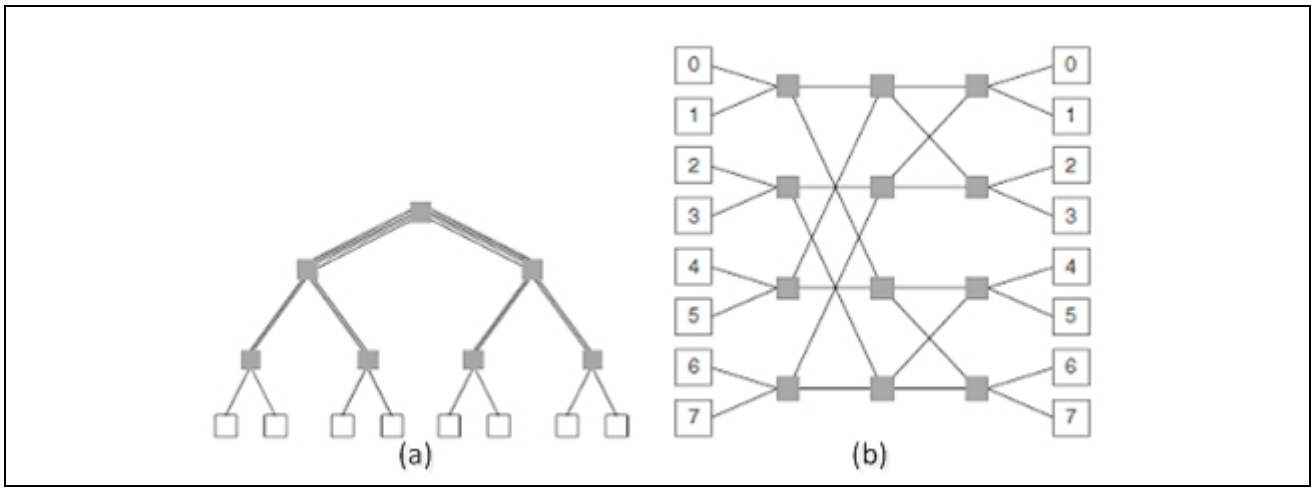


Figura 8. Redes indiretas: (a) *fat tree*; e (b) *butterfly*
Fonte: Pasricha et al. (2008).

A topologia Clos, ilustrada na Figura 9(a), consiste em uma rede de três estágios em que cada estágio é formado por chaves *crossbar* (são rápidos por permitir conexões simultâneas entre todas as suas entradas e saídas). A topologia Clos é um exemplo de rede sem bloqueios, porém com custo alto devido a vários *crossbar* completos, mas consequentemente oferece todo o suporte para

alto desempenho devido à alta largura de banda. A topologia Benes, ilustrada na Figura 9(b), é um exemplo de rede que tem caminhos de conexão que podem rearranjar para fornecer uma conexão, requerendo um controlador apropriado (PASRICHA; DUTT, 2008, p. 447).

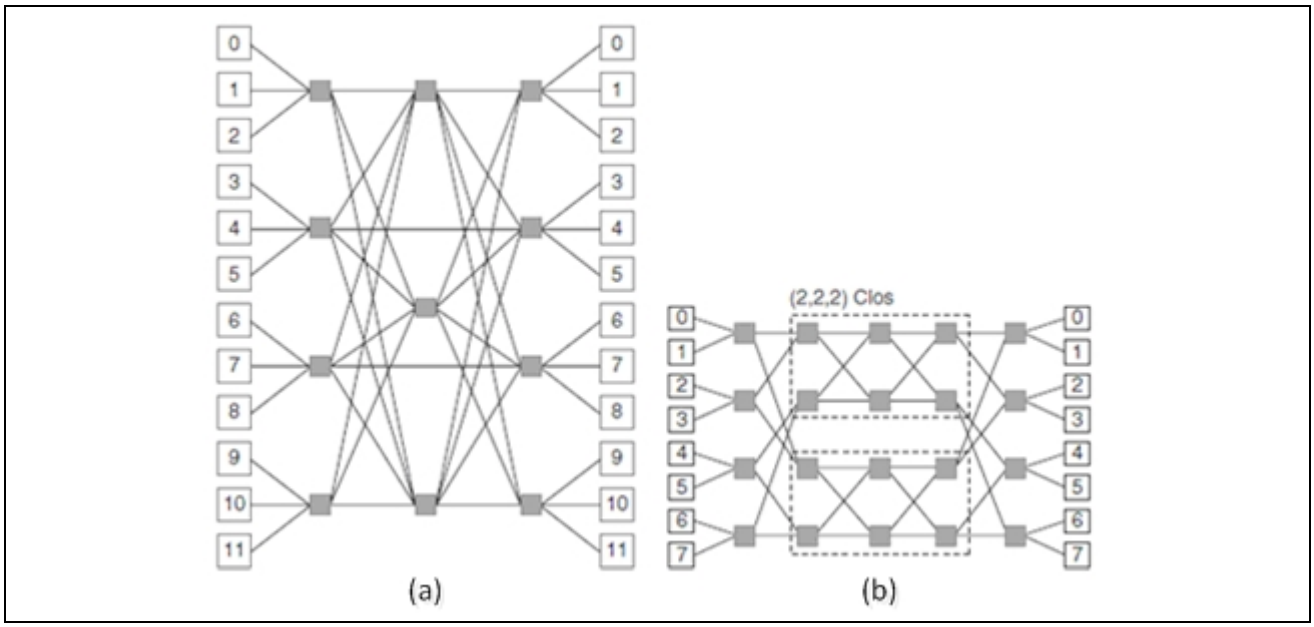


Figura 9. Redes indiretas: (a) Clos; e (b) Benes
Fonte: Pasricha et al. (2008).

As topologias de redes irregulares ou *ad hoc* são geralmente a integração do barramento compartilhado com rede direta e rede indireta. O principal objetivo desta rede é o aumento da disponibilidade da largura de banda quando comparada com barramento compartilhado, e redução de distância entre os nodos quando comparado com as redes diretas e indiretas. A topologia de rede irregular é tipicamente customizada para uma aplicação. Um exemplo de topologia de rede irregular e uma redução da malha 2D em que os roteadores e enlaces não utilizados são removidos, conforme apresentado na Figura 10(a). Um cluster é baseado em rede híbrida, onde cada cluster tem qualquer combinação com barramento compartilhado, rede direta e indireta. A Figura 10(b), ilustra um exemplo de topologia hibrida baseada em cluster que combina topologia de malha 2D e anel. As NoCs *x*Pipes e Æthereal são dois exemplos de arquitetura NoC que permitem topologia irregular (PASRICHA; DUTT, 2008, p. 448).

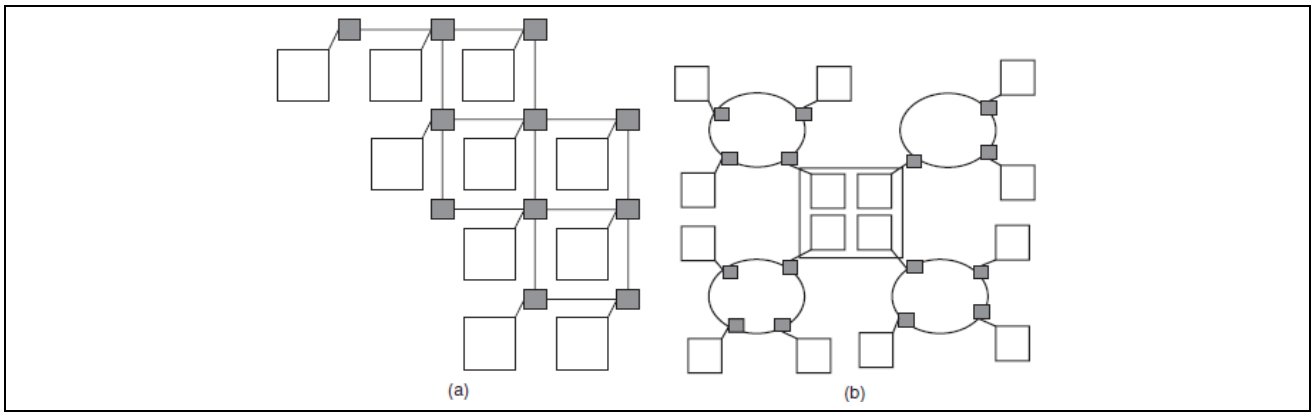


Figura 10. Redes irregulares: (a) Malha otimizada; e (b) Topologia híbrida
Fonte: Pasricha et al. (2008).

### 2.2.2 Roteamento

O roteamento estabelece o caminho que cada pacote ou mensagem deve seguir. Algumas propriedades das redes de interconexão sofrem influência direta do algoritmo de roteamento utilizado. Entre estas propriedades podemos citar as seguintes:

- Conectividade: Capacidade de rotear pacote de qualquer nodo fonte para qualquer nodo destino;
- Adaptatividade: Capacidade de rotear pacotes através de caminhos alternativos na presença de congestionamento ou falha de componentes;
- Liberdade de Deadlock e Livelock: Capacidade de garantir que os pacotes não serão bloqueados ou ficarão circulando pela rede sem atingir o nodo destino; e
- Tolerância à faltas: Capacidade de rotear pacotes na presença de faltas em componentes. A tolerância à faltas pode ser alcançada pelo encaminhamento de pacote em duas ou mais fases, armazenando-o em alguns nodos intermediários (DUATO;YALAMANCHILI; NI, 2003, p. 139).

De acordo com Duato, Yalamanchili e Ni (2003, p. 139-141), a taxonomia para o algoritmo de roteamento segue os critérios de classificação abaixo:

- Número de destinos: O algoritmo de roteamento pode ser primeiro classificado de acordo com o número de destinos, se os pacotes serão destinados a um destino ou a múltiplos destinos;
- Decisão de roteamento: O algoritmo de roteamento também pode ser classificado de acordo com o local da decisão de roteamento: (*i*) roteamento centralizado: caminho estabelecido por um controle centralizado; (*ii*) roteamento fonte: caminho definido pelo nodo origem; (*iii*) roteamento distribuído: determina de maneira distribuída o caminho que o pacote irá percorrer na rede; e (*iv*) roteamento *multicast*: os pacotes podem ser entregues para todos os nodos destinos ou apenas para o último nodo destino;
- Implementação: O algoritmo de roteamento pode ser implementado de diferentes maneiras, como, por exemplo, baseado em tabela de roteamento *(table lookup*) ou executando um algoritmo de roteamento em software ou em hardware;
- Adaptatividade: O algoritmo de roteamento pode ser determinístico, quando fornece sempre o mesmo caminho entre um nodo fonte e destino, ou adaptativo, quando utiliza as informações da rede sobre o tráfego, tais como: *status* do canal e congestionamento;
- Progressividade: O algoritmo de roteamento adaptativo pode ser classificado em relação à progressividade, ou seja, quando o cabeçalho do pacote consegue avançar pela rede, reservando um novo canal a cada operação de roteamento, ou *backtracking*, quando o cabeçalho do pacote retornar pela rede, liberando os canais previamente reservados;
- Minimalidade: O algoritmo de roteamento adaptativo pode ser classificado de acordo com sua minimalidade em dois tipos: (*i*) *profitable*: seleciona canais que levem a caminhos mais curtos até o destino; e (*ii*) *misrouting*: seleciona canais que levem a caminhos mais afastados do seu destino; e
- Número de caminhos: O algoritmo de roteamento adaptativo pode ser classificado de acordo com: caminho completo que utiliza todos os caminhos disponíveis, ou parciais, que utiliza apenas um conjunto de caminhos disponíveis.

### 2.2.3 Chaveamento

A estratégia de chaveamento determina como o fluxo de dados irá seguir através dos roteadores na rede, definindo também a granularidade da transferência de dados e a técnica de chaveamento utilizada. A Figura 11 ilustra a estrutura de uma mensagem. As mensagens geradas pelos nodos são particionadas em vários pacotes. Esses pacotes podem ser divididos em múltiplos *flits* (unidade de controle de fluxo). Um *flit* é uma unidade básica sobre a qual é realizado o controle de fluxo e é essencial para a sincronização entre roteadores. Um *flit* é composto por um ou mais *phits* (unidade física). Diferentes arquiteturas de NoCs utilizam diferentes tamanhos de *phit*, *flit*, pacotes e mensagens, pois o tamanho tem um impacto direto no custo, desempenho e potência das NoCs (PASRICHA; DUTT, 2008, p. 448-450).

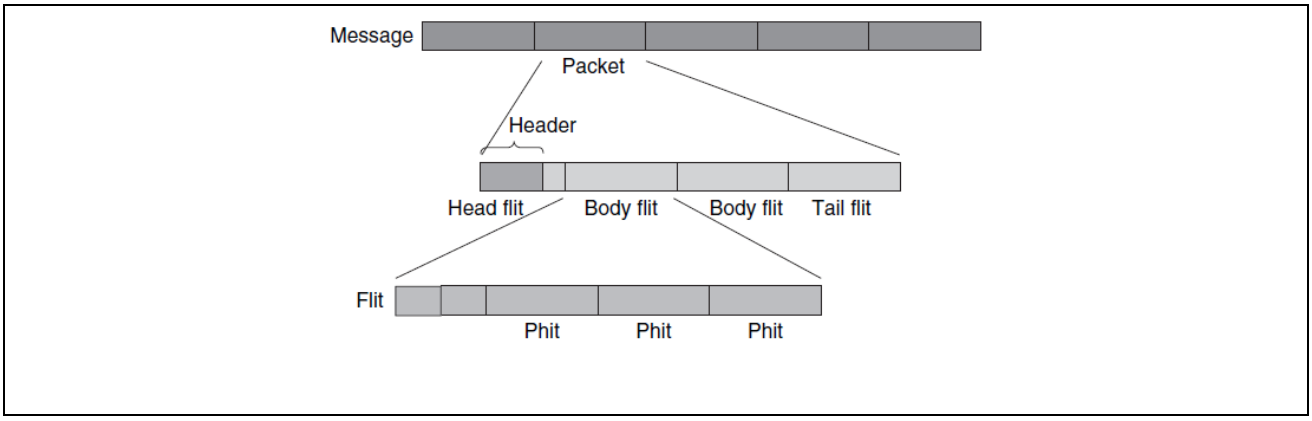


Figura 11. Estrutura mensagem: pacote, *flits* e *phits*
Fonte: Pasricha et al. (2008).

As duas principais técnicas de chaveamento utilizadas em NoCs são: chaveamento por circuito e chaveamento por pacote, as quais são descritas a seguir.

#### 2.2.3.1 Chaveamento por circuito

O chaveamento por circuito acontece quando o caminho físico entre o nodo fonte e destino é reservado antes da transmissão dos dados. O caminho físico é construído a partir de uma série de enlaces e roteadores. O *flit* de cabeçalho da mensagem atravessa a rede da origem até o destino, reservando assim os enlaces ao longo do caminho. Se o cabeçalho encontrar o destino sem nenhum conflito, todos os enlaces do caminho são reservados, disponibilizados e reconhecidos.

A transmissão dos dados só é iniciada após o recebimento da informação de disponibilidade e reconhecimento do caminho. Se o enlace apresentar reserva de circuito para outro nodo fonte, será encaminhada uma mensagem com a informação de indisponibilidade. O caminho é mantido reservado até que todos os dados da mensagem sejam transmitidos. A vantagem desta abordagem é que, no momento em que está reservado o caminho, a largura de banda é toda disponibilizada para essa transmissão, resultando assim em uma baixa latência. Um exemplo de arquitetura de rede NoC que implementa o chaveamento puro por circuito é a SoCBUS. Entretanto, um circuito de chaveamento puro não apresenta escalabilidade com o crescimento da NoC, considerando que vários enlaces estarão ocupados para a transmissão de dados, novas mensagens que desejarem ser transmitidas poderão ser descartadas antes mesmo da liberação do caminho (PASRICHA; DUTT, 2008, p. 449-450).

#### 2.2.3.2 Chaveamento por pacote

No chaveamento por pacote, em vez de estabelecer o caminho antes de enviar qualquer dado através do circuito de chaveamento, os pacotes são transmitidos da origem independente do caminho do destinatário, permitindo assim roteamento e atrasos diferentes. Enquanto o chaveamento por circuito tem um tempo de espera para iniciar seguido por uma latência mínima dos roteadores, o chaveamento por pacote não tem tempo de espera para iniciar a transmissão, mas apresenta atrasos durante a transmissão devido ao congestionamento nos roteadores. Por não reservar o caminho, vários pacotes de diferentes fontes podem solicitar ao roteador a utilização de um enlace. As técnicas de chaveamento de pacote mais utilizadas são: (*i*) Store and Forward (SAF): o roteador deve possuir espaço para armazenar um pacote inteiro e o seu roteamento é realizado apenas quando todo o pacote é recebido; (*ii*) Virtual Cut Through (VCT): o roteador também deve possuir espaço para armazenar um pacote inteiro, mas o seu roteamento pode ser iniciado tão logo o cabeçalho for recebido, reduzindo a latência de comunicação; e (*iii*) Wormhole (WH): o roteador deve ser capaz de armazenar apenas alguns *flits* do pacote (os demais são mantidos nos roteadores anteriores do caminho), o que reduz o custo de armazenamento, e o roteamento pode ser iniciado assim que o cabeçalho for recebido (PASRICHA; DUTT, 2008, p. 450-451).

### 2.2.4 Controle de fluxo

O controle de fluxo pode ser considerado como um protocolo de sincronização entre transmissão e recepção de dados, utilizando para isso as seguintes técnicas: (*i*) Handshake: o receptor retorna um sinal de confirmação quando o *phit* é recebido; (*ii*) Stop-and-Go: o receptor retorna um sinal solicitando a interrupção ou o estabelecimento do fluxo de um *phit* em função de sua disponibilidade para recebê-lo; (*iii*) Baseado em créditos: o receptor retorna um sinal de crédito quando o *phit* é consumido; e (*iv*) Canais virtuais: os canais físicos são compartilhados por canais lógicos (DUATO; YALAMANCHILI; NI, 2003, p. 422-424).

O controle de fluxo também pode ser uma alternativa para resolver o problema de contenção dos pacotes quando atravessam a rede. Em particular, em nível de enlace de dados, quando ocorre erro na transmissão, dependendo do erro, é possível realizar a recuperação por meio dos mecanismos de controle disponibilizados. Por exemplo, se um pacote for corrompido e necessitar de retransmissão, o fluxo é paralisado e é enviado um sinal de requisição para realocar *buffer* e recurso de banda. A maioria das técnicas de controle de fluxo pode gerenciar o congestionamento de enlace, mas não gerenciam as realocações necessárias para a retransmissão (DALLY; TOWLES, 2004, p. 225).

### 2.2.5 Arbitragem

Enquanto o caminho de dados dos roteadores é composto de *buffers* e *switches*, a parte de controle dos roteadores é composta de árbitros e alocadores. Os árbitros são utilizados para resolver múltiplas solicitações para um simples recurso. Sempre que um recurso, tal como *buffer*, canal ou uma porta de *switch* é compartilhada, um árbitro é necessário para permitir acesso de um agente por vez para cada recurso. As principais técnicas de arbitragem são: (*i*) prioridades estáticas: cada canal de entrada possui uma prioridade fixa ao competir pelo canal de saída; (*ii*) prioridades dinâmicas: as prioridades dos canais de entrada são atribuidas a cada arbitragem por uma fila circular (*round-robin*) de prioridade rotativa; (*iii*) *deadline* ou escalonamento por idade: a mensagem que esta aguardando há mais tempo pelo uso do canal de saída é selecionada pelo árbitro; (*iv*) LRS (Least Recently Served): o canal de entrada menos servido recentemente é selecionado; (*v*) FCFS (First Come First Served): o primeiro canal de entrada a solicitar o canal de saída é selecionado pelo árbitro; e (*vi*) multinível: implementa em mais de um nível diferentes esquemas de arbitragem,

como prioridade dinâmica em conjunto com prioridade estática (DUATO; YALAMANCHILI; NI, 2003, p. 422-424).

## 2.3 ANÁLISE DE DESEMPENHO

Existem várias formas de se obter o desempenho de uma rede e uma delas é por meio da interconexão de um terminal a uma das portas da rede, conforme ilustrado na Figura 12. O terminal de instrumentação é constituído por um gerador de pacotes e um isolador de tráfego da rede, permitindo obter as métricas de vazão (*throughput*), latência e tolerância à faltas. O sistema pode ser configurado para realizar medidas na forma: (*i*) *open-loop*: nesta forma os parâmetros são controlados independentemente da rede, ou seja, os pacotes são gerados e injetados na rede mesmo quando o *buffer* de entrada estiver cheio; (*ii*) *closed-loop*: nesta forma de medida, é considerada a avaliação de um sistema completo em que a rede influencia no tráfego; e (*iii*) *stead state*: o sistema roda um experimento (ou simulação) em três fases: (1) aquecimento: não é realizado nenhum registro de tempo ou quantidade de pacotes; (2) medição: todos os pacotes que chegam neste intervalo são contados: e (3) drenagem: aguarda a chegada de todos os pacotes gerados na fase anterior, para realizar a medida de tempo final.

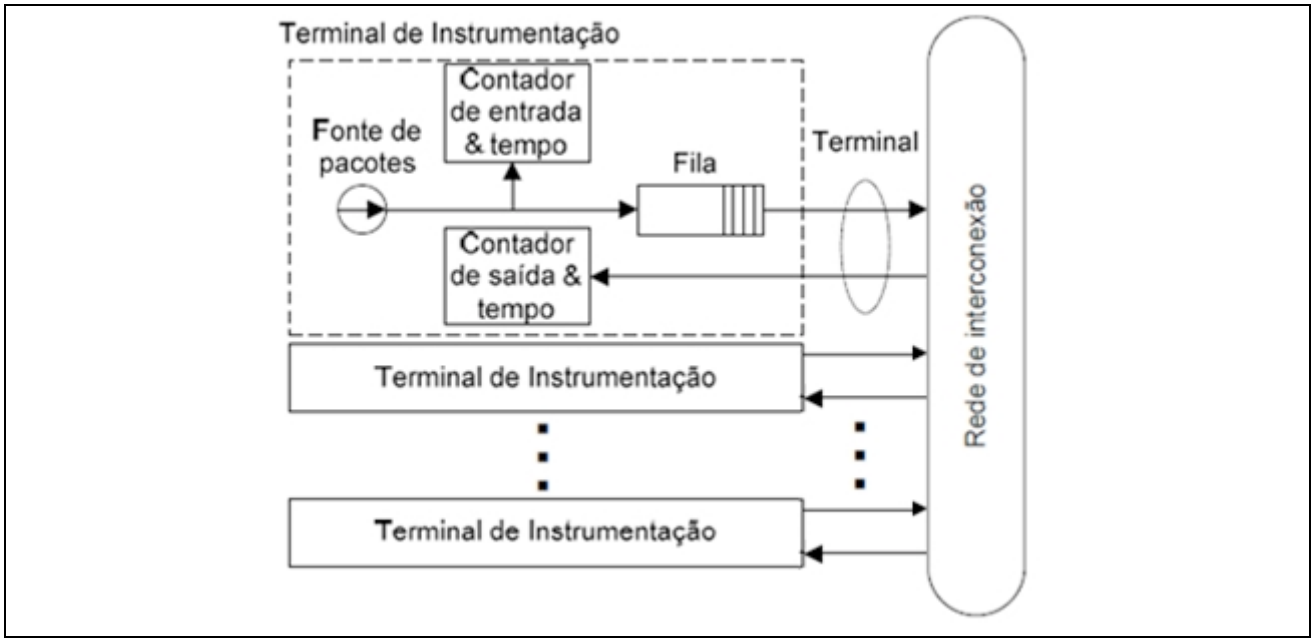


Figura 12. Estrutura do terminal de instrumentação
Fonte: Adaptado de Dally e Towles (2004).

A métrica vazão é obtida na fase de medição, por meio da informação da quantidade de pacotes recebidos. A latência é obtida por intermédio do intervalo de tempo entre início e fim de transferência de todos os pacotes recebidos. Durante as fases de aquecimento e drenagem, o gerador de pacotes continua funcionando, porém, os pacotes gerados não são analisados (DALLY; TOWLES, 2004, p. 449-468).

### 2.3.1 Vazão

A vazão (ou *throughput*) pode ser definida como a taxa em que os pacotes são entregues à rede para um padrão de tráfego específico. Esta medida conta a quantidade de pacotes que chega ao destino em um intervalo de tempo para cada fluxo considerando um padrão de tráfego. Já o tráfego oferecido ou demanda é a taxa em que os pacotes são gerados pelo gerador de pacotes. A Figura 13, apresenta o gráfico da vazão (ou tráfego aceito) em função da demanda. Observa-se que antes de chegar à fase de saturação, a vazão mantém uma relação diretamente proporcional. Aumentando-se o tráfego oferecido, é alcançada a zona de saturação, onde a vazão se mantém constante com o aumento da demanda (DALLY; TOWLES, 2004, p. 452).

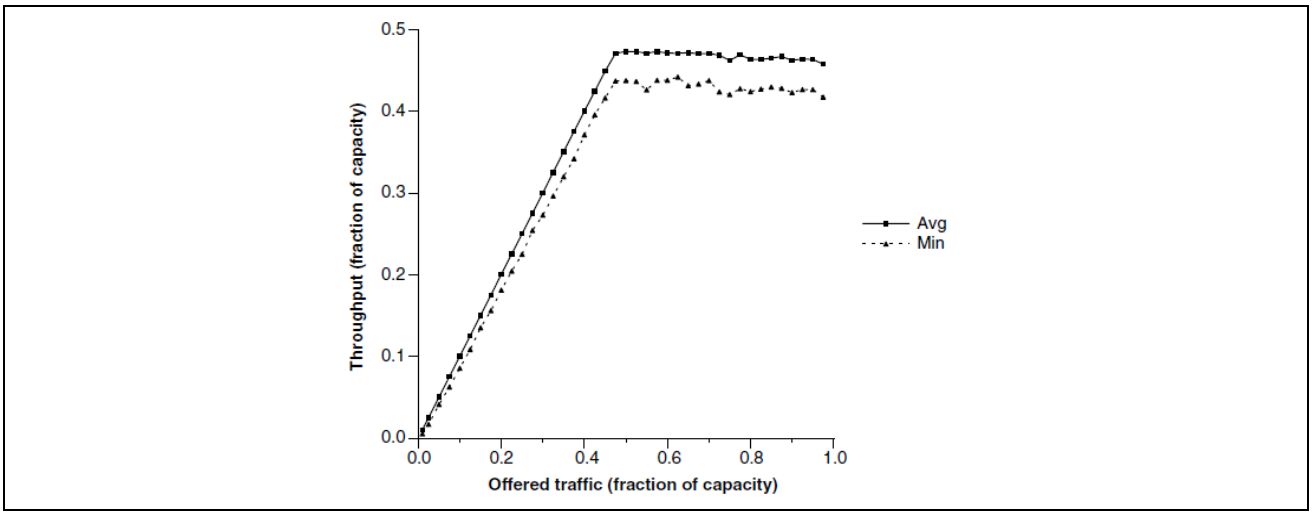


Figura 13. Relação entre vazão e tráfego oferecido
Fonte: Dally e Towles (2004).

### 2.3.2 Latência

Latência é o intervalo de tempo que um pacote leva para atravessar a rede e chegar ao seu destino, ignorando a latência gerada pela contenção interna dos pacotes devido ao

compartilhamento de recursos. Uma vez incluído o tempo de contenção por meio de modelos ou de simulação, a latência se torna dependente do tráfego oferecido, conforme ilustrado na Figura 14 (DALLY; TOWLES, 2004, p. 455).

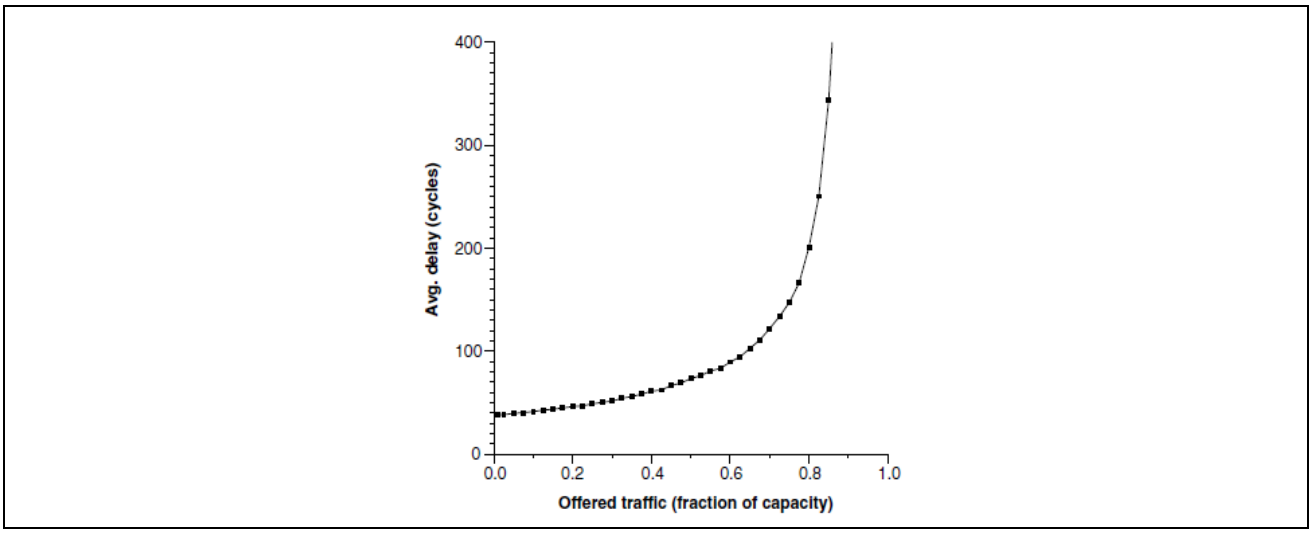


Figura 14. Relação entre latência e tráfego oferecido
Fonte: Dally e Towles (2004).

### 2.3.3 Tolerância a Faltas

Tolerância a faltas apresenta a capacidade em que uma rede tem em se manter em operação na presença de uma ou mais faltas. Uma rede é considerada tolerante a faltas em um simples ponto se esta continuar operando na presença de qualquer nodo ou canal com falta, ou seja, consegue entregar pacotes entre os terminais restantes. Para muitos sistemas, considerar tolerância a faltas em um simples ponto é suficiente se a probabilidade de ocorrência de uma ou múltiplas faltas for extremamente baixa (DALLY; TOWLES, 2004, p. 508).

### 2.3.4 Geração de tráfego

A geração de tráfego é uma das formas utilizadas para realizar a avaliação das redes de interconexão, de forma a extrair informações de desempenho, visto que o tráfego gerado influencia diretamente o resultado do desempenho. Esta avaliação requer a definição e a representação dos modelos de tráfego (DUATO; YALAMANCHILI; NI, 2003, p. 479).

De acordo com Tedesco (2005, p. 9), a modelagem de tráfego especifica a estrutura de transmissão dos dados do nodo fonte para o destino. Os modelos de tráfego são utilizados quando não se tem muitos detalhes das aplicações do sistema. A maioria dos resultados de avaliação de desempenho tem considerado somente a distribuição uniforme. Contudo, pesquisadores têm considerado cargas sintéticas, simulando o comportamento de aplicações reais. Poucos pesquisadores têm avaliado a rede de interconexão utilizando padrões de tráfego produzidos por aplicações reais (DUATO; YALAMANCHILI; NI, 2003, p. 475).

Conforme apresentado por Duato, Yalamanchili e Ni (2003, p. 480-481), o modelo de tráfego pode ser definido pelos parâmetros: distribuição de tráfego, taxa de injeção e tamanho da mensagem. A localidade da comunicação pode ser classificada como: (*i*) espacial: quando a distância entre os nodos é menor que a distribuição uniforme, onde todos os nodos têm a mesma probabilidade de serem destinos (como resultado, cada mensagem consome poucos recursos, o que também reduz a contenção das mensagens); ou (*ii*) temporal: quando a probabilidade de enviar mensagens para nodo que recentemente foi utilizado é maior que para outro nodo.

No padrão de tráfego com distribuição uniforme, todos os nodos têm a mesma probabilidade de receberem mensagens, pois não há um destino específico. Já no padrão de tráfego com distribuição não uniforme, a probabilidade de enviar mensagens para um nodo, diminui com o aumento da distância entre fonte e destino, ou seja, são enviadas com mais frequência para os nodos vizinhos (DUATO; YALAMANCHILI; NI, 2003, p. 481).

O Quadro 2, apresenta os padrões de tráfego comumente usados na avaliação de redes de interconexão para computadores paralelos e que também se aplicam a NoCs. No padrão de tráfego randômico, cada nodo fonte tem a mesma probabilidade para enviar pacotes. Por realizar uma distribuição de tráfego uniforme se torna um dos mais utilizados. Tal padrão consegue realizar um balanceamento de carga mesmo para topologias e algoritmos de roteamento que normalmente apresentam baixo balanceamento de carga (DALLY; TOWLES, 2004, p. 51).

Quadro 2. Padrões de tráfego

| **Tipo de Tráfego** | **Padrão** |
|---|---|
| Random | $\lambda sd = 1/N$ |
| Permutação | $d = \pi(s)$ |
| Permutação Bit | $di = s_{f(i)} \oplus g_{(i)}$ |
| • Complemento Bit | $di = \overline{s_i}$ |
| • Reversão Bit | $di = s_{b-i-1}$ |
| • Rotação Bit | $di = s_{i+1} \bmod b$ |
| • Shuffle | $di = s_{i-1} \bmod b$ |
| • Transposição | $di = s_{i+b/2} \bmod b$ |
| Permutação Byte | $dx = f(s_{g(x)})$ |
| • Tornado | $dx = sx + (\lceil k/2 \rceil - 1) \bmod k$ |
| • Neighbor | $dx = sx + 1 \bmod k$ |

Fonte: Adaptado de Dally et al. (2004).

Já no padrão de tráfego permutação, é gerado um fluxo de tráfego de cada nodo fonte para um simples nodo destino, de forma a verificar as limitações da topologia ou algoritmo de roteamento. O padrão de tráfego permutação de bit é um subnível da permutação, no qual cada endereço de destino é calculado a partir da permutação dos bits do endereço fonte. O padrão de tráfego permutação de byte (ou de dígito) é um subnível da permutação, no qual os dígitos do endereço são calculados a partir do endereço fonte. Tal permutação se aplica apenas a redes em que os endereços possam ser expressos por dígitos como a *k-ary n-cube* (toróide) e a *k-ary n-fly* (multiestágio *butterfly*) (DALLY; TOWLES, 2004, p. 51).

## 2.4 MPSoC

Na última década, novas arquiteturas foram desenvolvidas utilizando soluções MPSoC (Multiprocessor System-on-Chip) baseadas em multiprocessador implementados em FPGA, permitindo uma rápida prototipagem, como também a pesquisa de novas arquiteturas e técnicas de comunicação.

Essas arquiteturas de sistemas em MPSoC são, na maioria das vezes, constituídas por múltiplos processadores programáveis como componentes do sistema. Esses sistemas não são

simplesmente processadores tradicionais integrados em um único *chip*, mas processadores específicos para aplicações embarcadas (WOLF; JERRAYA; MARTIN, 2008).

Os sistemas MPSoC baseados em FPGA apresentam vantagens que compensam sua aplicação: (*i*) flexibilidade e reconfiguração: o número de processadores *soft-core* que podem ser incluídos é limitado apenas pela capacidade do FPGA e também é possível reconfigurar cada processador independentemente; (*ii*) *time-to-market*: o desenvolvimento do processador não inclui a fabricação do CI, resultando em uma redução de tempo de desenvolvimento; (*iii*) baixo custo: o custo do processo de desenvolvimento e fabricação é considerado barato, devido à flexibilidade de alteração do projeto sem a necessidade de alterar a fabricação do CI; e (*iv*) escalabilidade: sistemas MPSoCs baseados em FPGA podem aumentar o número de processadores ou periféricos se existirem recursos disponíveis no FPGA (DORTA et al., 2010, p.2).

Ao contrário dos SoCs tradicionais, MPSoCs são formados por dois ou mais processadores que gerenciam o processo da aplicação, alcançando assim alto desempenho. A escalabilidade associada com alta adaptabilidade faz dos sistemas MPSoC uma solução que combina flexibilidade de software com aceleração (HUBNER; BECKER, 2011, p. 2).

Atualmente os principais fabricantes de FPGAs para sistemas MPSoCs são: Altera e Xilinx. A Xilinx é largamente utilizada em sistemas MPoPC (Multiprocessor-on-Programmable Chip or Soft Multiprocessor) e disponibiliza três linhas de processadores: (*i*) *Soft-core* Microblaze (MB): processador RISC (Reduced Instruction Set Computer) de 32 bits, baseado em uma arquitetura Harvard; (*ii*) *Soft-core* PicoBlaze: processador RISC de 8 bits; e (*iii*) *Hard-core* Power PC: processador RISC de 32 bits. A Altera disponibiliza na linha de processadores *soft-core* o Nios II com barramento Avalon 32 bits com ferramentas de integração entre software e hardware que permite a adição de sistemas multiprocessados. Já na linha de processadores *Hard-core* está disponível as famílias dos devices Cyclone e Arria com processador ARMv7 de 32 Bits. Os *open-source soft-core* mais comuns são o OpenRisc da OpenCores e o Leon 3 da Gaisler, conforme apresentado no Quadro 3 (DORTA et al., 2010, p.3).

Quadro 3. Principais modelos de processadores baseados em FPGA

| Processador | Fabricante | Tipo arquitetura | Classificação | Licença |
|---|---|---|---|---|
| MicroBlaze | Xilinx | RISC – 32 Bits | *Soft-core* | Proprietário |
| PowerPC | Xilinx | RISC – 32 Bits | *Hard-core* | Proprietário |
| PicoBlaze | Xilinx | RISC – 8 Bits | *Soft-core* | Proprietário, sem custo |
| ARM Córtex-M1 | ARM | RISC – 32 Bits | *Soft-core* | Proprietário |
| Nios II | Altera | RISC – 32 Bits | *Soft-core* | Proprietário |
| Cyclone V | Altera | ARMv7-A- 32 Bits | *Hard-core* | Proprietário |
| Arria V/ Arria 10 | Altera | ARMv7-A- 32 Bits | *Hard-core* | Proprietário |
| Leon 3 | Gaisler | RISC – 32 Bits | *Soft-core* | *Open-source* |
| OpenRisc 1200 | OpenCore | RISC – 32 Bits | *Soft-core* | *Open-source* |
| LatticeMico32 | Lattice | RISC – 32 bits | *Soft-core* | *Open-source* |

Fonte: Adaptado: Dorta et al. (2010), Balwaik et al. (2013), Altera (2014)

A Figura 15 ilustra um comparativo em relação à quantidade de processadores que podem ser inseridos em sete modelos de FPGAs do fabricante Altera considerando os processadores *soft-core* e *open-core* (Nios II/f, ARM Cortex-M1, Leon 3). Pode-se observar que o device EPS5SEBB permite a inserção de aproximadamente duzentos processadores *soft-core* Nios II/*f*.

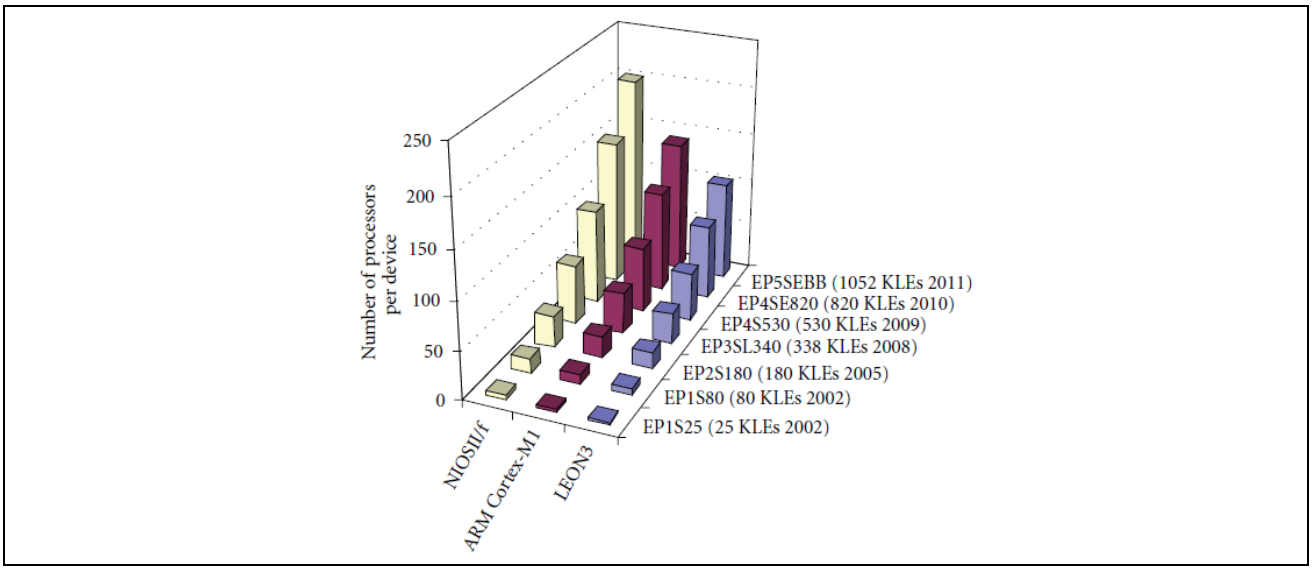


Figura 15. Quantidade de processadores por dispositivo da Altera

Fonte: Rufas et al. (2012).

### 2.4.1 Estrutura genérica de um MPSoC

Normalmente, são as aplicações que determinam a arquitetura do MPSoC baseado em FPGA. O sistema pode ser configurado em: (*i*) *Master-Slave*: neste sistema pode-se ter um ou mais processadores atuando como processador *master*, controlando o comportamento dos outros processadores *slave*; (*ii*) Pipeline: o sistema é composto por uma cadeia de processadores, cada processador atuando em um estágio do *pipeline,* sendo que as tarefas são particionadas no tempo resultando em um melhor desempenho; (*iii*) *Net*: neste sistema multiprocessado, não existe hierarquia entre processos, todos os processadores podem iniciar a comunicação entre si quando necessário (DORTA et al., 2010, p. 3).

Outra importante característica em um sistema MPSoC é o tipo da implementação física da arquitetura de comunicação. Existem três abordagens: (*i*) ponto-a-ponto; (*ii*) barramento compartilhado; e (*iii*) NoC.

Os métodos de troca de informações utilizados entre os processadores podem ser classificados em: memória compartilhada ou passagem de mensagem (memória distribuída). O Quadro 4, apresenta um resumo das características de uma arquitetura MPSoC (DORTA et al., 2010, p. 4).

Quadro 4. Arquitetura MPSoC

| **Arquitetura Sistema** | **Arquitetura comunicação** | **Método comunicação** |
|---|---|---|
| Master-Slave | Ponto-a-Ponto | Passagem de mensagem |
| Pipeline | Barramento compartilhado | Memória compartilhada |
| Net | NoC | Passagem de mensagem ou Memória compartilhada |

Fonte: Adaptado de Dorta et al. (2010).

A estrutura genérica de um MPSoC, ilustrada na Figura 16, é apresentada como sendo uma estrutura composta de vários elementos de processamento (PE – Processing Element), podendo ser classificada como homogênea ou heterogênea.

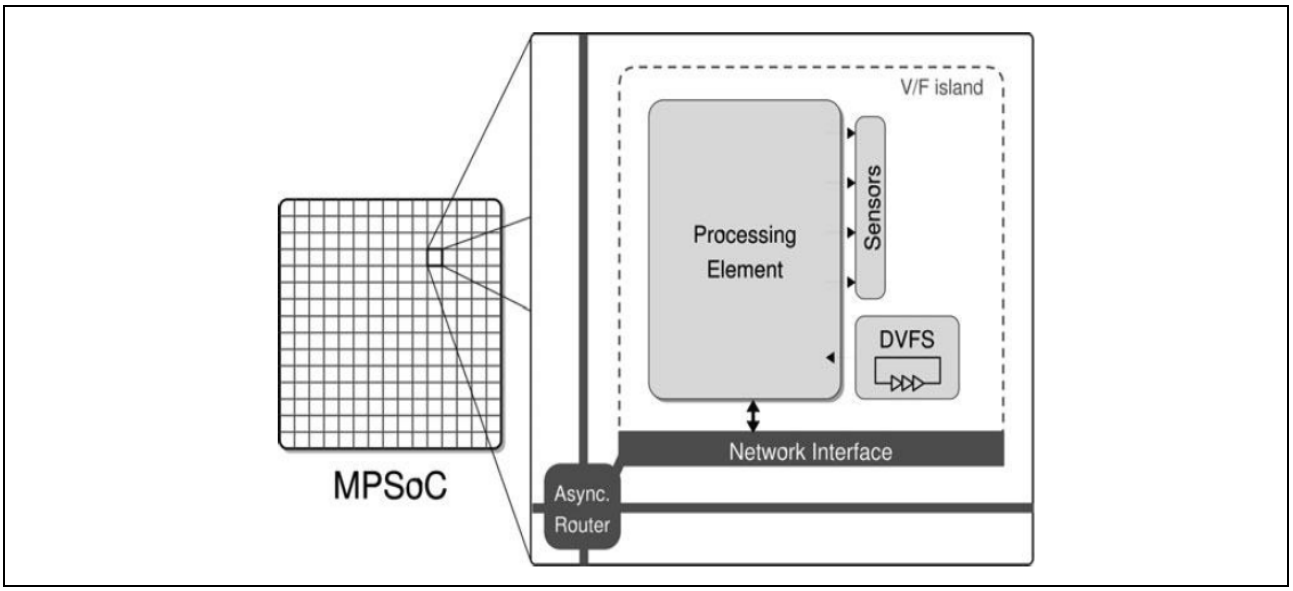


Figura 16. Estrutura genérica MPSoC
Fonte: Hubner e Becker (2011).

Um sistema MPSoC heterogêneo é formado por diferentes tipos de PEs, por exemplo: processadores de propósito geral, processador de sinal digital (DSP – Digital Signal Processor), hardware de aceleração, periféricos e uma infraestrutura de interconexão como uma NoC. Os MPSoCs heterogêneos conseguem oferecer uma melhor relação desempenho/potência e, por isso, são utilizados em sistemas embarcados. Porém eles sofrem com limitações de flexibilidade e escalabilidade (HUBNER; BECKER, 2011, p. 7).

A Figura 17 ilustra uma estrutura genérica de um MPSoC heterogêneo composto por processador de propósito geral (CPU) em P1, vários aceleradores (vídeo, áudio, etc.) em P2, elementos de memória em P3, periféricos em P4 e uma rede de interconexão (DORTA et al., 2010, p. 4).

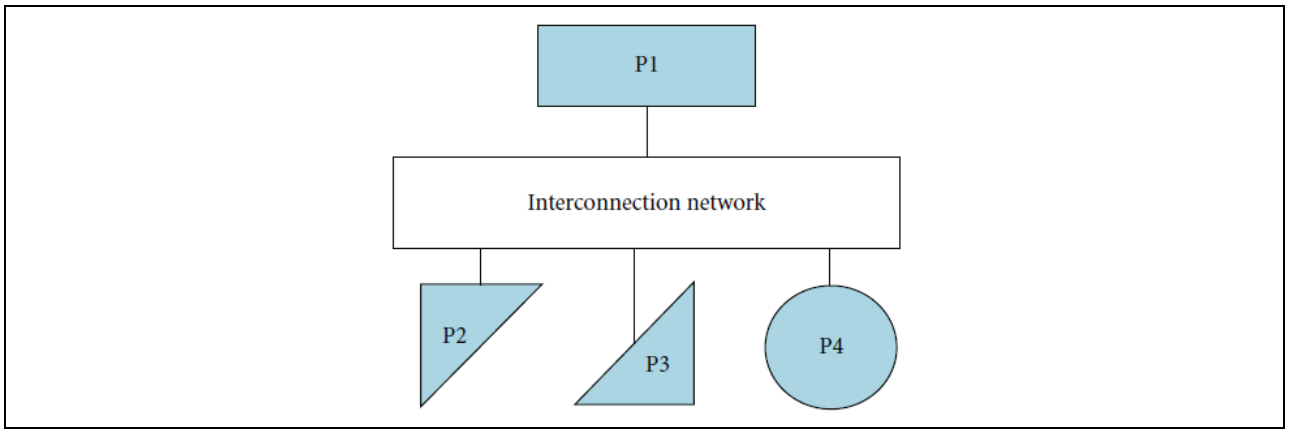


Figura 17. Estrutura de sistema MPSoC heterogêneo
Fonte: Dorta et al. (2010).

Já os sistemas MPSoCs com estrutura homogênea são formados por um mesmo tipo de PE, conforme ilustrado na Figura 18. Este PE é instanciado várias vezes e todas as instâncias são interconectadas por meio de uma infraestrutura de comunicação dedicada. Neste sistema, tanto o hardware como o software são os mesmos. Ele apresenta maior flexibilidade, escalabilidade, processamento e menor eficiência de potência, bem como permite a reusabilidade do PE. São comumente usados para consoles de *videogame*, computadores *desktop*, servidores e supercomputadores. Este modelo de estrutura é frequentemente referenciado na literatura como um modelo de arquitetura paralela (HUBNER; BECKER, 2011, p. 9-11).

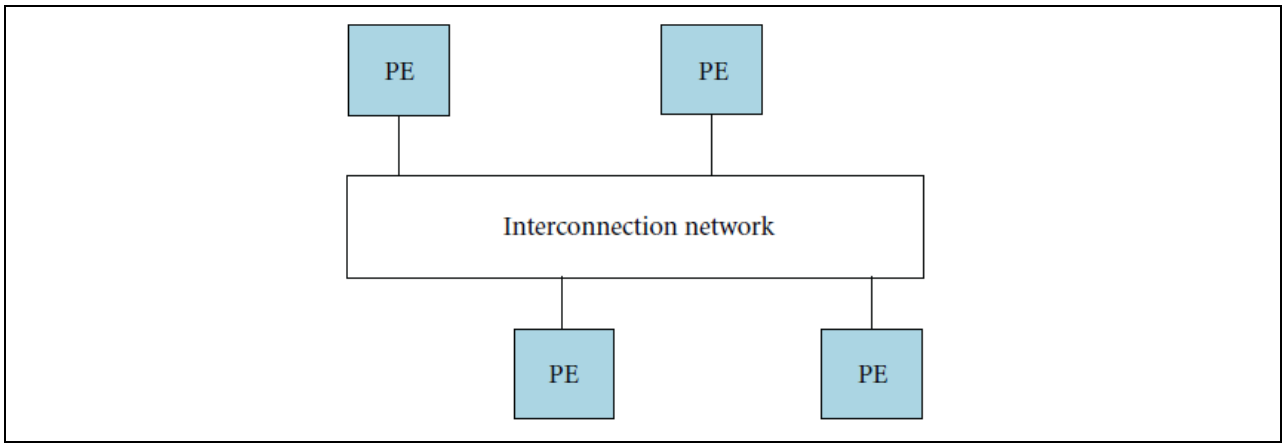


Figura 18. Estrutura de sistema MPSoC homogêneo
Fonte: Dorta et al. (2010).

A Figura 19 apresenta duas arquiteturas de sistemas, uma com a organização de memória compartilhada e outra com memória distribuída. Na primeira, a troca de informações pode ser realizada por meio de variáveis compartilhadas, o que requer cuidados com sincronismo e proteção de memória. Já na arquitetura de memória distribuída, a comunicação deve ser baseada na passagem de mensagens, ou seja, no envio e no recebimento de mensagens por meio de um protocolo de comunicação. O desenvolvedor do software é responsável por realizar a sincronização das tarefas (HUBNER; BECKER, 2011, p. 9-11).

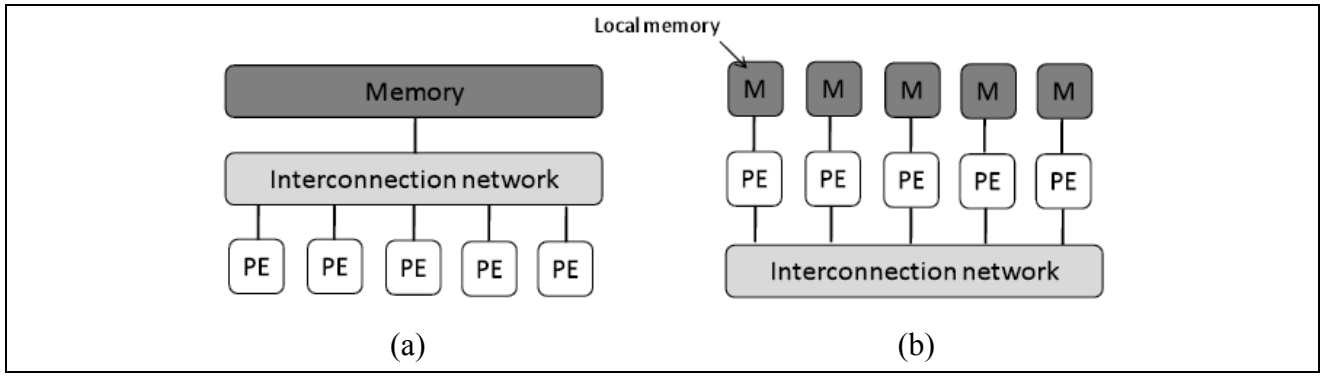


Figura 19. Organização da Memória: (a) Compartilhada; e (*b*) Distribuída
Fonte: Adaptado de Hubner et al. (2011).

### 2.4.2 Interconexão

Os elementos de processamento de um MPSoC são interconectados por meio de uma NoC, a qual é formada por interfaces de rede (NI – Network Interface), roteadores e enlaces. A NI é a interface entre a NoC e o PE. Já o roteador é responsável pelo roteamento dos pacotes entre PEs fonte e destino. A vazão na NI deve ser suficiente para o funcionamento dos aplicativos. As NoCs possibilitam a implementação de sistemas globalmente assíncronos e localmente síncronos (GALS – Globally Asynchronous Locally Synchronous) com a implementação de uma interface síncrono-assíncrono na NI.

### 2.4.3 Métodos de Comunicação

Em um sistema MPSoC, o método de comunicação entre os processadores pode ser realizada por: (*i*) passagem de mensagem; e (*ii*) memória compartilhada (HWANG; XU, 1998, p. 653-659). A comunicação por meio de memória compartilhada ocorre por meio de um espaço de

endereço de memória acessado por todos os processadores do sistema, permitindo assim ao sistema garantir a realização do processo de coerência de dados e sincronização (HWANG; XU, 1998, p. 653-659). Já a comunicação por meio de passagem de mensagem (também chamada de troca de mensagem) é realizada pela utilização de primitivas de comunicação de envio e recebimento de mensagens (ex: MPI_SEND e MPI_RECEIVE).

O método de comunicação por passagem de mensagem MPI (Message Passing Interface) foi criado em 1992, por um comitê que padronizou a comunicação para ambientes de memória distribuída de forma a oferecer: facilidade de utilização, portabilidade para outros sistemas, segurança no estabelecimento da comunicação e escalabilidade. Neste comitê, havia representantes de fabricantes, universidades e laboratórios governamentais envolvidos nos assuntos de computação paralela. No comitê, foram discutidos os assuntos referentes à sintaxe, à semântica e ao conjunto de rotinas destinadas à troca de mensagens. A primeira versão do padrão de passagem de mensagem MPI (1.0) foi apresentada em 1994 e disponibilizada pela Universidade do Tennessee. Em meados de 2015, foi publicada a versão MPI 3.1. Desde a primeira versão até agora, foram realizadas várias melhorias no documento, acrescentadas novas funcionalidades, feitas correções na documentação e incluídas novas rotinas (MESSAGE PASSING INTERFACE FÓRUM, 2015, p. ii-iv).

A biblioteca MPI disponibiliza primitiva especificas para gerenciamento de processos, para a comunicação ponto-a-ponto (síncrona ou assíncrona, bloqueante ou não bloqueante) e para comunicação em grupos (*broadcast* e sincronização de processos). Essas primitivas permitem que um processador envie dados para outro processador sem a necessidade de realizar nenhum processo de sincronização de dados (MESSAGE PASSING INTERFACE FÓRUM, 2015, p. ii-iv).

## 2.5 CONSIDERAÇÕES

Este capítulo apresentou uma breve revisão sobre conceitos de sistemas em chip, NoCs, geração de tráfego, avaliação de desempenho de NoCs e MPSoCs. Os SoCs permitem a construção e a integração de múltiplos componentes em um único chip por meio de interconexões ponto-a-ponto e/ou barramento. Nas interconexões ponto-a-ponto, a complexidade aumenta na medida em que cresce a quantidade de núcleos. Já na interconexão em barramento compartilhado, o desempenho diminui com o aumento do sistema. Para atender aos requisitos de desempenho e consumo de energia em SoCs, uma nova estrutura para comunicação foi proposta: a NoC. As NoCs são escaláveis, reutilizáveis e apresentam eficiência na comunicação, multiplexando diferentes

fluxos de comunicação em interconexões curtas. Para avaliar uma NoC, são utilizados geradores e medidores de forma a extrair informações para calcular o seu desempenho. Os sistemas MPSoCs são formados por dois ou mais processadores mestre que executam os processos da aplicação, alcançando assim alto desempenho. O método de comunicação nesses sistemas pode ser por passagem de mensagem ou memória compartilhada.

Neste trabalho, apresenta-se um MPSoC para avaliação de desempenho da NoC SoCIN baseado em processadores de propósito geral utilizando o método de comunicação por passagem de mensagem MPI para a comunicação entre os nodos. O Capítulo 4 apresenta com mais detalhes a arquitetura da plataforma MPSoC proposta. Já o capítulo a seguir (Capítulo 3) apresenta uma análise de trabalhos relacionados e posiciona esta dissertação em relação a esses trabalhos.

# 3 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos relacionados e o estado da arte de pesquisas sobre os tema: geração, medição e monitoramento de tráfego em NoCs. A Seção 3.1 aborda soluções baseadas em emulação para geração/medição de tráfego. Já a Seção 3.2 tem como enfoque as soluções baseadas em simulação para geração/medição de tráfego. Na Seção 3.3, são apresentadas algumas soluções para monitoramento das NoCs. Por último, na Seção 3.4, será apresentado o posicionamento desta dissertação em relação aos trabalhos relacionados e o estado da arte, obtidos por meio do protocolo de busca apresentado no Apêndice A.

## 3.1 EMULAÇÃO

### 3.1.1 Genko et al. (2005)

Genko et al. (2005) desenvolveram um ambiente de emulação flexível implementado em FPGA com o objetivo de explorar, avaliar e comparar as possíveis configurações de soluções NoC em nível físico. Esse ambiente foi implementado utilizando o kit de desenvolvimento Xilinx Virtex II Pro-V20 com o processador Power PC embarcado. A Figura 20 ilustra a arquitetura da plataforma de emulação da NoC.

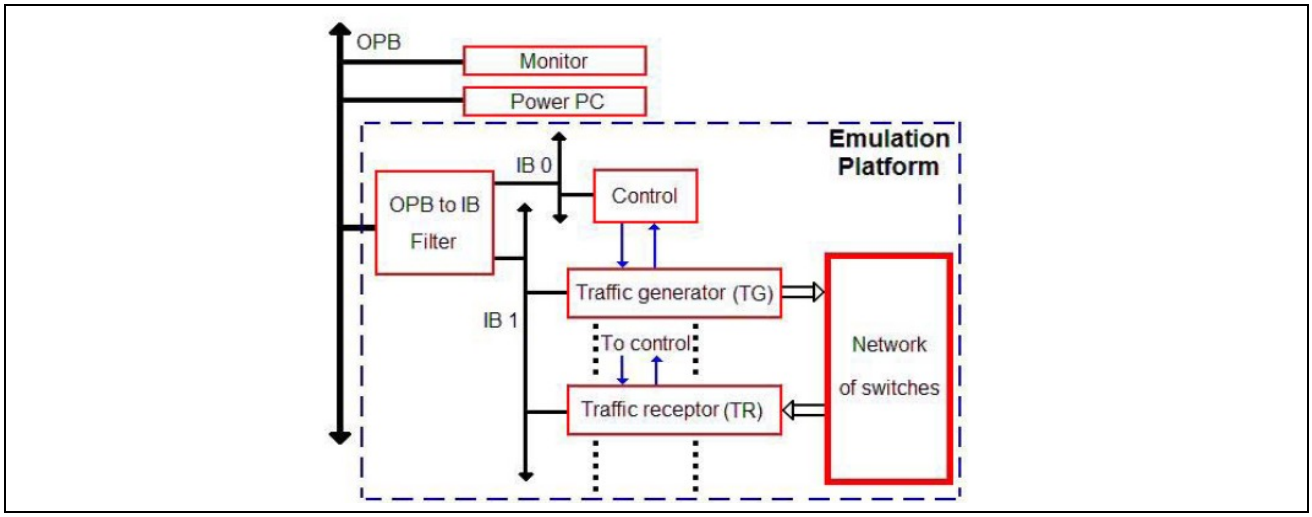


Figura 20. Plataforma emulação NoC de Genko et al.
Fonte: Genko et al. (2005).

Essa arquitetura apresenta como principais modulos: monitor, power PC e plataforma de Emulação. O módulo Monitor fornece uma interface para comunicação com o *Host Computer* por uma porta serial. Já o módulo Power PC é utilizado para gerenciar os processos de emulação. O módulo principal, que é a Plataforma de Emulação, é formado pelos seguintes submódulos: Gerador de Tráfego (TG – Traffic Generator), Medidor de Tráfego (TR – Traffic Receptor) e a NoC. A geração de tráfego pode ser do tipo estocástico ou *trace-driven* de aplicações reais. Foram implementadas duas diferentes formas de TRs nesta plataforma: (*i*) *traffic-activity-analysis*: realiza uma verificação automática dos *flits* recebidos por meio da verificação do CRC (Cyclic Redundancy Check) para garantir que os dados recebidos provenientes do TG não foram alterados; e (*ii*) *independent-packet-trace-analysis*: realiza a verificação manual do conteúdo, sendo que a informação dos *flits* é encaminhada pela interface serial.

A plataforma foi submetida a experimentos em dois ambientes: simulador HDL (ModelSim) e simulador SystemC, considerando duas configurações: (*i*) NoC 3x2 e geração de tráfego estocástica (distribuição uniforme e rajada); e (*ii*) NoC 2x2 e geração de tráfego *Trace-driven*. O modelo proposto apresentou uma aceleração (*speedup*) com magnitude quatro vezes maior em relação ao simulador HDL e tempo inferior de execução em relação ao systemC.

### 3.1.2 Wolkotte et al. (2007)

Wolkotte et al. (2007) desenvolveram uma plataforma de emulação para geração de tráfego utilizando duas placas para a: (*i*) implementação do SoC; e (*ii*) implementação da NoC no FPGA (Virtex-II 8000). A Placa SoC, ilustrada na Figura 21, contém dois processadores de propósito geral (ARM9), memória SRAM (de 1 MB), periféricos e conectores. A interconexão entre as placas de FPGA e SoC é realizada por meio de uma interface de memória de 32 bits de dados e 17 bits de endereçamento. O gerador de tráfego é implementado na placa SoC por software (processador ARM9). Na placa FPGA, foi modelada uma NoC com mecanismo de transferência dos dados por chaveamento de pacotes, com topologia malha e roteamento XY.

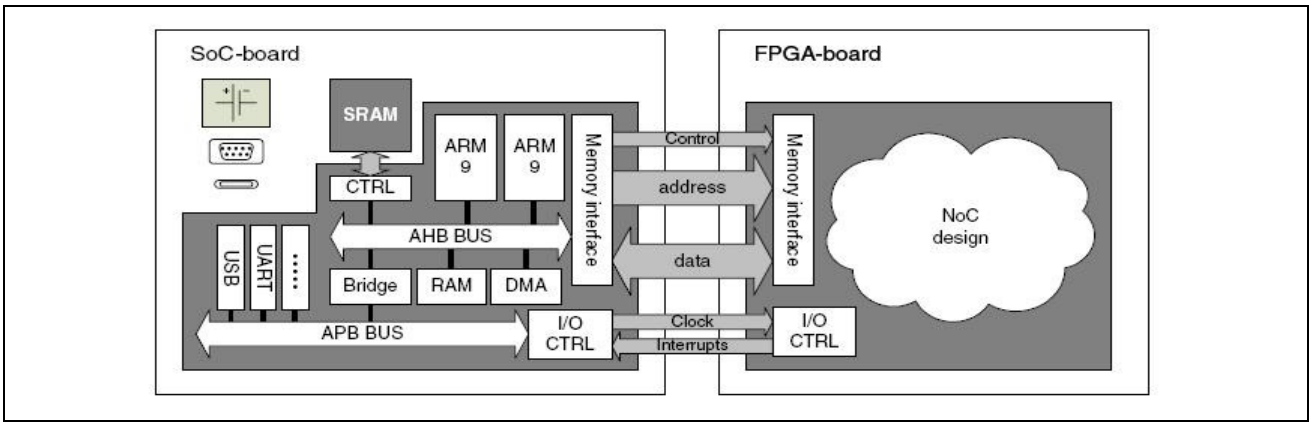


Figura 21. Plataforma de emulação de Wolkotte et al.
Fonte: Wolkotte et al. (2007).

Foi realizada uma comparação entre a NoC modelada/simulada em SystemC com a NoC emulada em FPGA. Os autores adotaram como métrica a latência média por pacote. O sistema emulado em FPGA mostrou ser 80-300 vezes mais rápido que o modelo simulado em SystemC. Por sintetizar e carregar um roteador por vez no FPGA, este permite a emulação de uma rede NoC com grandes dimensões, porém com o aumento no tempo para a execução do aplicativo.

### 3.1.3 Luo-Feng et al. (2008)

Luo-Feng et al. (2008) desenvolveram uma plataforma de emulação utilizando um kit de prototipação com o FPGA Stratix II EP2S180 do fabricante Altera. Foi realizado a implementação de um sistema baseado em NoC com o objetivo de comparar o desempenho de três topologias de interconexões, tais como: malha 2D, barramento compartilhado e ponto-a-ponto em relação ao desempenho. Esta plataforma é formada por um sistema de processamento local (LPS – Local Processing System) composto por quatro processadores de propósito geral, compatíveis com a arquitetura ARM, interconectados com a NoC por meio de interfaces de rede (NI), conforme apresentado na Figura 22.

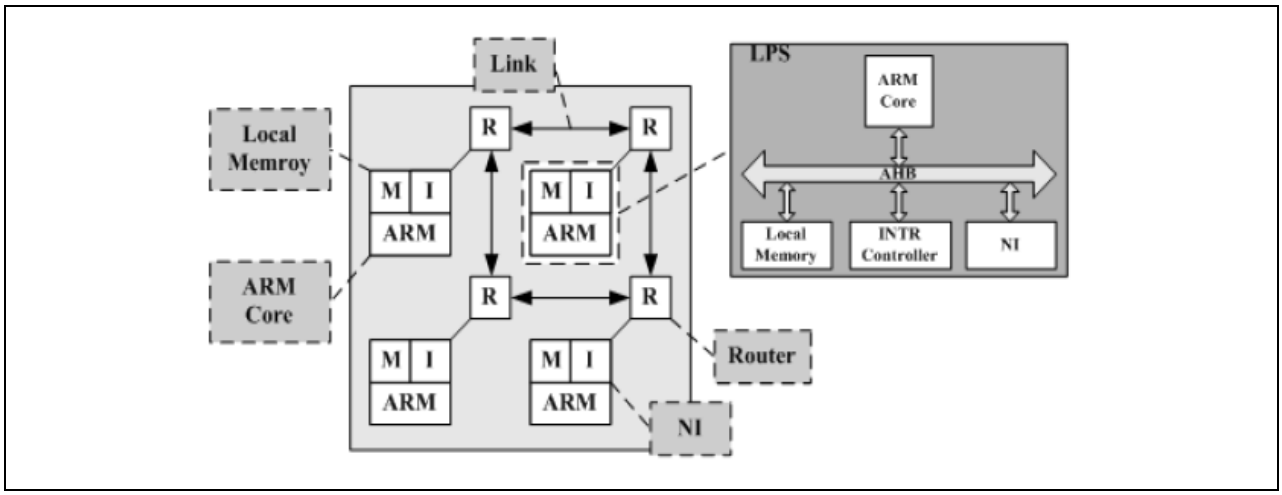


Figura 22. Plataforma de emulação NoC de Lou-Feng et al.
Fonte: Lou-Feng et al. (2008).

Foram desenvolvidas duas aplicações paralelas com diferentes padrões de tráfego, a MMP (*pipeline of matrix multiplications*– Pipeline de Multiplicação de Matrizes) e a FFT (*Fast Fourier Transform*– Transformada Rápida de Fourier) para comparação do tempo de execução nas três topologias. Os autores adotaram como métrica do sistema a aceleração (*speedup*) e o tempo de execução do sistema. Os resultados mostraram que a arquitetura da plataforma de emulação na NoC com topologia malha 2D oferece um melhor desempenho na comunicação da rede, bem como uma melhor utilização da área do FPGA devido à disponibilização de alta largura de banda e melhor paralelismo. Ressalta-se que o tamanho da rede NoC é limitada pelo FPGA.

### 3.1.4 Zeferino e Pereira (2008)

Zeferino e Pereira (2008) desenvolveu uma plataforma de emulação com a rede SoCIN baseada nos núcleos do roteador ParIS (Parameterizable Intecornnect Switch) com módulos geradores de tráfego (TG – Traffic Generator) e medidores de tráfego (TM – Traffic Meter) sintetizáveis para validação e avaliação de desempenho de uma NoC em FPGA, conforme é ilustrado na Figura 23.

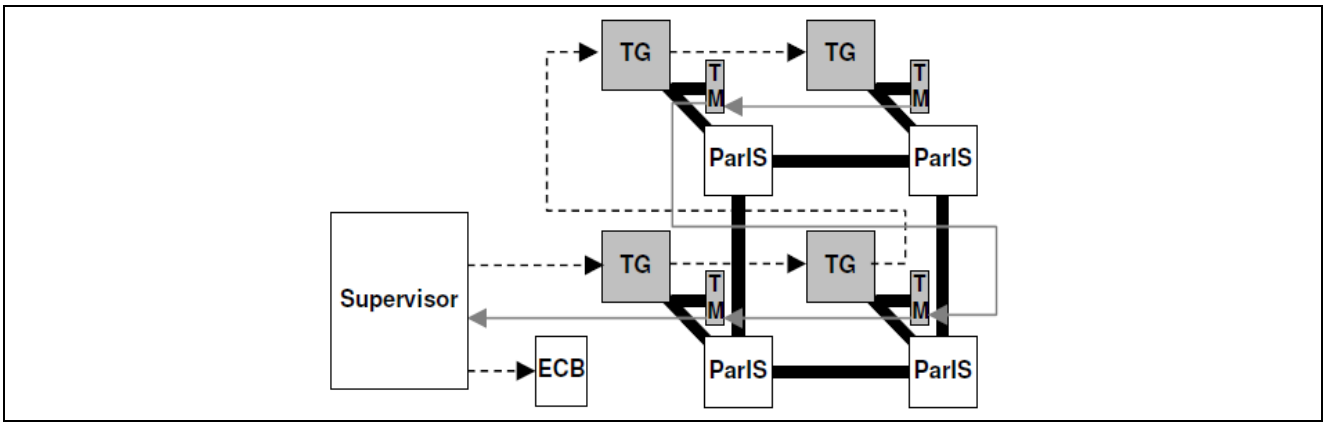


Figura 23. Plataforma de avaliação de desempenho de Zeferino e Pereira
Fonte: Adaptado de Zeferino e Pereira (2008).

Esta plataforma foi baseada na arquitetura original da plataforma em SystemC para avaliação da rede SoCIN (ZEFERINO et al., 2007). Foi gerado um sistema 2x2 composto de nodos formados por instâncias do TG, do TM e do roteador ParIS, sintetizado para o FPGA Cyclone II EP2C35F672C da Altera. Os módulos geradores e medidores de tráfego foram implementados como processador de propósito específico (não programáveis) e descritos em VHDL. Cada módulo TG é capaz de gerar quatro fluxos de comunicação diferentes. Os módulos TG e TM possuem uma interface serial com o bloco supervisor com a função de realizar a configuração dos TGs e a coleta dos dados pelos TMs.

Foram adotadas como métricas de avaliação: a latência mínima, a latência máxima, a latência média e o tráfego aceito (vazão). Os módulos TM e TG foram implementados e validados por simulação, sendo totalmente parametrizáveis. Esta plataforma apresenta algumas limitações. O tamanho máximo da rede depende da capacidade do FPGA. Os TGs geram apenas quatro fluxos diferentes de pacotes e o tempo para configuração dos geradores aumenta com a quantidade de TGs presentes na NoC.

### 3.1.5 Wen et al. (2009)

Wen et al. (2009) desenvolveram um gerador de tráfego configurável em tempo real (OCTG: On-Line Configurable Traffic Generator, módulo IP) para avaliação de desempenho da comunicação na NoC, de forma a permitir o ajuste direto das configurações do ambiente de geração de tráfego sem a necessidade de reiniciar ou parar a execução da plataforma de emulação. Os

parâmetros disponíveis para a configuração do OCTG são: quantidade de pacotes, tamanho dos pacotes, taxa de injeção dos pacotes, modo de tráfego, modo de configuração e nodo destino. A configuração em tempo real do OCTG é realizada através da interface JTAG (Joint Test Action Group). A configuração do OCTG pode ser realizada de duas formas: (*i*) configurar todos os OCTGs; e (*ii*) configurar apenas um OCTG. O TG proposto pode gerar dois modos de tráfego: BT (Broadcast Transmission) e o NTNT (Node-to-Node Transmission). No BT, o nodo fonte envia pacotes para todos os outros nodos da rede. Já no NTNT, o nodo fonte pode enviar pacotes para somente um destino. O experimento foi baseado em uma NoC com topologia em malha 2D 3x3, composta por 9 OCTGs e um bloco de configuração (Configuration Engine) implementado no FPGA da família Startix II da Altera, o qual é responsável por converter os dados provenientes da interface JTAG para o módulo de configuração de tempo real, conforme ilustrado na Figura 24.

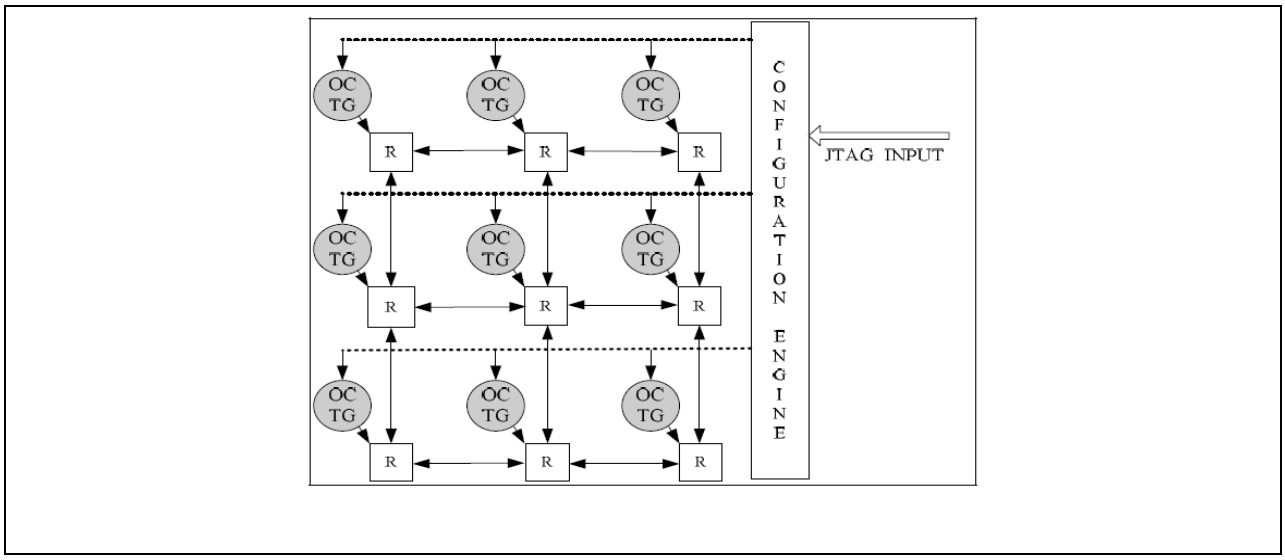


Figura 24. Estrutura do sistema de avaliação de desempenho de Wen et al.
Fonte: Wen et al. (2009).

O bloco Configuration Engine, ilustrado na Figura 25, é formado por processador embarcado Nios II, ROM e JTAG_UART. O módulo de configuração em tempo real, quando recebe os dados correspondentes ao parâmetro, atualiza imediatamente os sinais de configurações. O OCTG suporta diferentes modos de tráfego e permite selecionar diferentes algoritmos de roteamento, sendo ambos alterados durante a execução do processo.

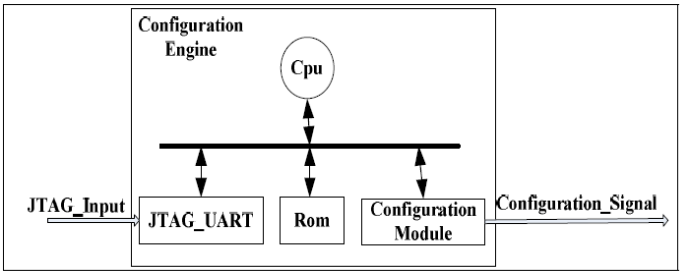


Figura 25. Estrutura interna do bloco de configuração de Wen et al.
Fonte: Wen et al. (2009).

### 3.1.6 Lotlikar et al. (2011)

Lotlikar et al. (2011) desenvolveram uma plataforma para emulação de NoC em FPGA, denominada ACeNoCs (Accelerated Emulation Plataform for NoCs – Plataforma de Emulação Acelerada para NoCs), capaz de realizar emulação síncrona e assíncrona baseada em GALS (Globally Asynchronous Locally Synchronous – Globalmente Assíncrono e Localmente Síncrono). A NoC é modelada usando os recursos de hardware FPGA, enquanto que os modelos de tráfego e as análises estatísticas foram implementados por *software* (*soft-core* MicroBlaze).

O hardware do ACeNoCs suporta a exploração das características dos roteadores, como também das configurações da rede. A parte de software permite a reconfiguração dos parâmetros de emulação e disponibiliza uma interface *Plug and Play* para integração de diferentes arquiteturas de roteadores e topologias de NoCs. A plataforma ACeNoCs, ilustrada na Figura 26, teve o hardware prototipado no kit de desenvolvimento XUPV5 (Xilinx University Program), o qual contém o FPGA Virtex-5 (VLX110T). No software executado no MicroBlaze, foram implementados os geradores e medidores de tráfego, controle dinâmico de emulação e gerador de relógio. Os geradores de tráfegos suportam: (*i*) Carga sintética, tais como: *uniform random, bit complement, bit reversal, matrix transpose, bit shuffle, bit rotation e hotpot*; e (*ii*) Carga real gerada por *trace-driven* de aplicações reais.

Os TRs (Traffic Receptors) realizam o cálculo da latência média e a latência para cada pacote recebido. A interface entre a parte de hardware e software é baseada em registradores conectados ao barramento PLB (Processor Local Bus). A emulação da plataforma foi realizada utilizando o EDK Xilinx (Embedded Development Kit).

A configuração da plataforma é feita por dois arquivos de configuração, um responsável pela configuração da NoC (dimensão da rede, topologia, tamanho dos dados do roteador, esquema de arbitragem, etc.) e outro pela modelagem de tráfego (quantidade de *flit*, padrão de tráfego, quantidade de pacotes, distribuição *clock*, etc.).

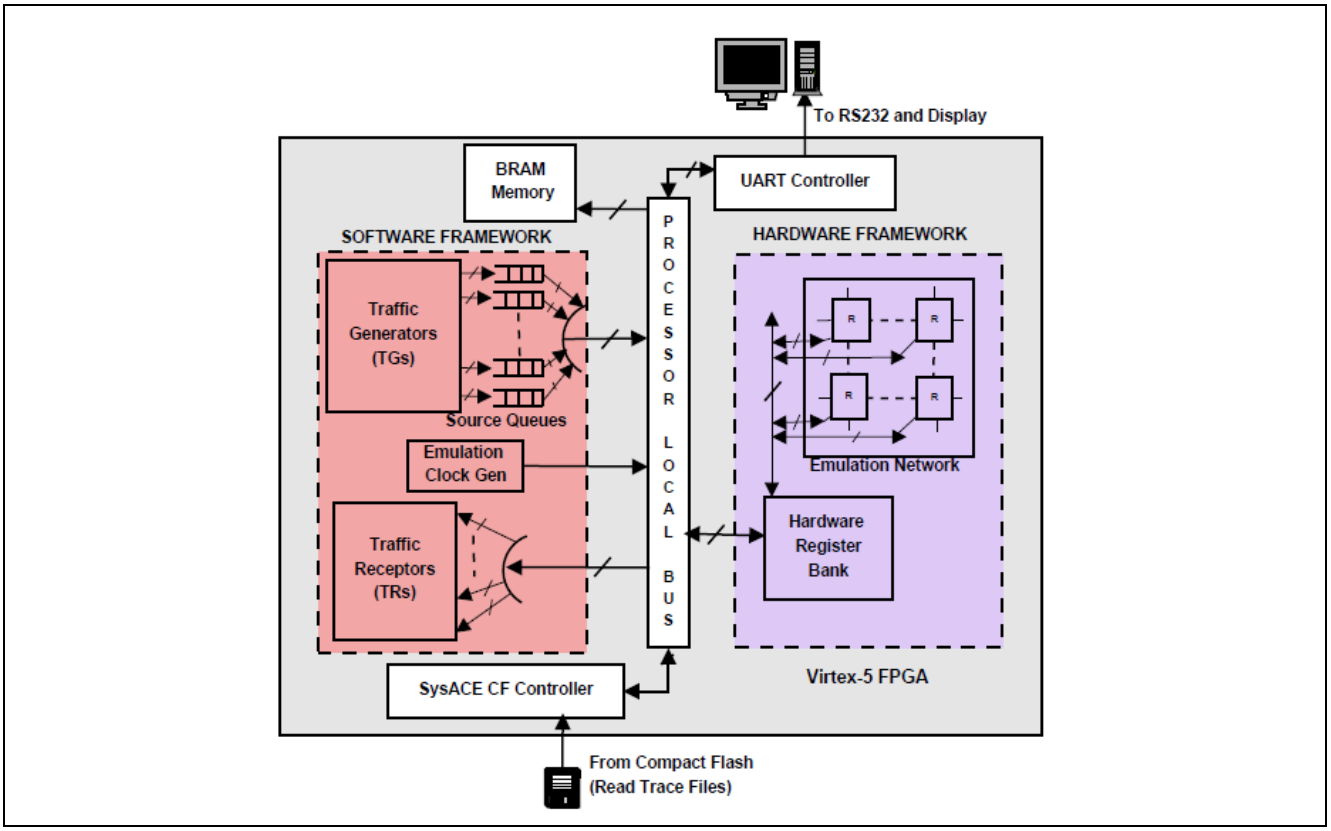


Figura 26. Plataforma de Emulação NoC de Lotlikar et al.
Fonte: Lotlikar et al. (2011).

O processo de emulação é finalizado apenas quando todos os pacotes gerados são recebidos ou quando é encontrado algum erro. Após o término são apresentadas as informações: quantidade de pacotes gerados e recebidos, ciclos de emulação, latência média dos pacotes, vazão e tempo total de emulação.

A avaliação da AcENoCs foi realizada por meio de comparação com o simulador de *software* OCIN_TSIM (On Chip Interconnect Network Timing Simulator) e o simulador HDL (ModelSim). A configuração utilizada na plataforma foi a seguinte: topologia em malha 2D 5x5, algoritmo de roteamento XY-DOR, dois canais virtuais por porta, buffers com profundidade de 8 *flits*. A rede opera de forma síncrona a um relógio comum. Os geradores de tráfegos geram fluxo do

tipo de complemento de bit, chegando até um milhão de pacotes, considerando o tamanho do pacote de cinco *flits*. A plataforma AcENoCs permitiu atingir uma aceleração de 10000 a 12000 vezes quando comparada ao simulador.

### 3.1.7 Zhou et al. (2011)

Zhou et al. (2011) propuseram um método de decodificação de imagem JPEG em uma arquitetura composta por processadores Nios II baseado no barramento Avalon. A arquitetura do sistema, conforme ilustra a Figura 27, apresenta nodos de processamento, interface de comunicação e módulos funcionais, sendo responsáveis por armazenar, decodificar e apresentar as imagens JPEG. Esta proposta foi desenvolvida com o kit de desenvolvimento com o FPGA da família Stratix II - EP2S180F1020C4 do fabricante Altera. O PE (elemento de processamento) é composto por um Core (Nios II) e uma memória interna (P_mem). A memória interna é utilizada para o armazenamento do programa de software. A memória interna, Image_RAM, é dividida em quatro regiões de tal forma que cada PE armazene suas imagens decodificadas.

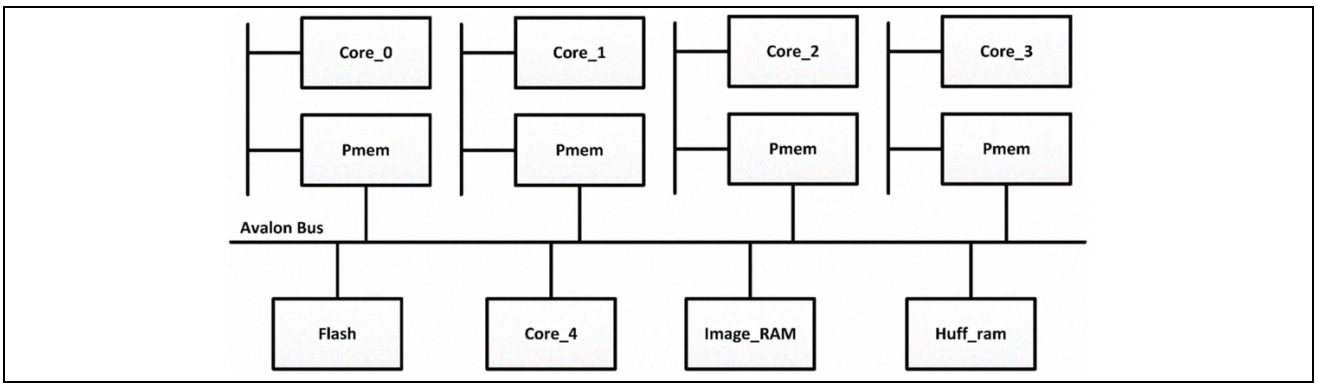


Figura 27. Plataforma de emulação HardNoC de Zhou et al.
Fonte: Zhou et al. (2011).

Para garantir que apenas um PE tenha o acesso de leitura ou escrita na memória Imagem_RAM, foi inserido na estrutura do sistema o componente Mutex que tem como função liberar ou bloquear o acesso a memória. Para avaliar o desempenho da estrutura proposta, foi realizada uma comparação com os sistemas: (*i*) estrutura de decodificação JPEG baseado no barramento AHB com 4 core; e (*ii*) estrutura de decodificação JPEG baseado em barramento Avalon com apenas um core. Utilizando como métricas: (*1)* quantidade de memória utilizada; (*2*) quantidade da ALUT (Adaptative Look-Up Table); e (*3*) tempo de decodificação.

A solução proposta, quando comparada com barramento AHB, apresentou uma redução de 12% na quantidade de memória, redução de 49% na quantidade de ALUT. Comparando com a estrutura de um Core, o tempo de decodificação foi 1/3 menor. Este sistema apresentou melhor resultado em relação à quantidade de recursos para executar a mesma tarefa, se tornando uma boa alternativa para decodificação de imagem JPEG.

### 3.1.8 Heck et al. (2012)

Heck et al. (2012) desenvolveram uma plataforma de emulação baseada em módulos geradores e medidores de tráfegos para NoCs com o objetivo de acelerar o processo de validação por meio de prototipação em FPGA. A arquitetura da plataforma, denominada HardNoC, é ilustrada na Figura 28. Este sistema é formado por: bloco de injeção e coleta de pacotes (*injector/colector*), bloco de comunicação com o PC (Serial) e NoC. O primeiro é formado por módulos: geradores, receptores, memória local e contador de ciclos de *clock*. O módulo gerador de tráfego permite gerar fluxos baseados em dois tipos de taxa de injeção: CBR (Constant Bit Rate – Taxa de Bit Constante) e Pareto On-Off. Os autores adotaram como métricas a latência (mínima, média e máxima) e quantidade de pacotes recebidos. Essa plataforma foi implementada em dois *kits* de desenvolvimento (Xilinx Virtex-II Pro Development System XUP-V2PRO e Xilinx ML505 Virtex-5) com o objetivo de verificar o funcionamento e os recursos utilizados pelo FPGA nas NoCs Hermes-2VC (sistema 2x3) e Hermes-GLP (sistema 3x3).

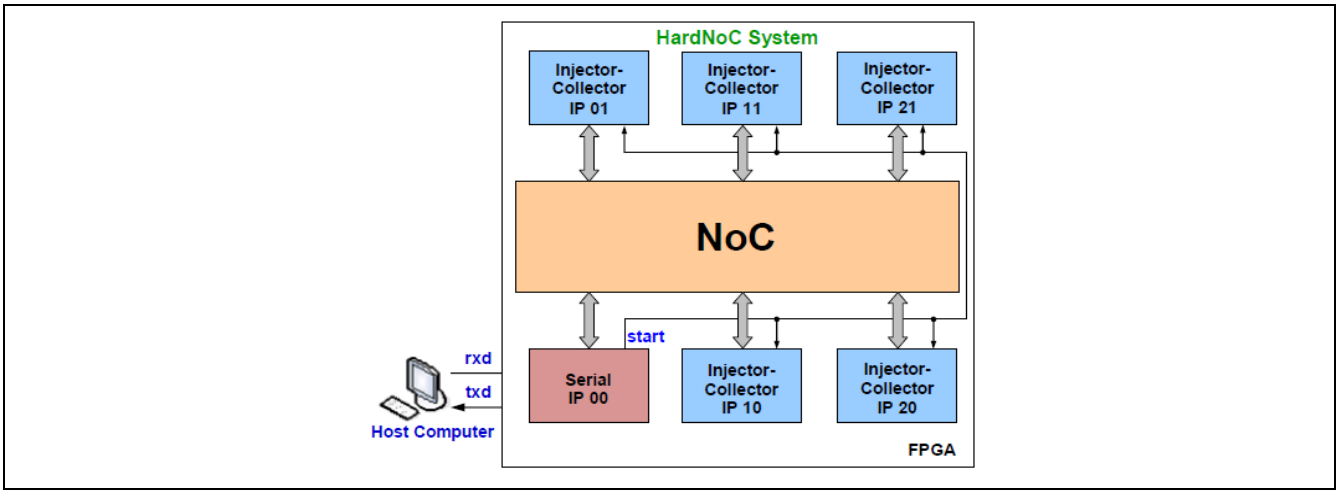


Figura 28. Plataforma de emulação HardNoC Heck et al.
Fonte: Heck et al. (2012).

Os resultados mostraram que a plataforma HardNoC, por não utilizar *soft processor* para geração e medição de tráfego, permite um aumento na dimensão das NoCs que podem ser emuladas. A plataforma não apresenta dispositivo para finalizar a execução do sistema, sendo esta uma limitação do sistema.

### 3.1.9 Fernandez-Alonso e Castells-Rufas (2010)

Fernandez-Alonso e Castells-Rufas (2010) propuseram uma metodologia para auxiliar no desenvolvimento e construção de sistemas baseados em NoC, bem como aplicações para desenvolvimento e análise de desempenho. Nessa metodologia, várias ferramentas foram combinadas tais como: SoPC Builder da Altera, NoCMAKER da UAB (Universitat Autonoma Barcelona) e Vampir do fabricante GWT_TUD. A Altera oferece ao desenvolvedor a ferramenta SoPC (System-on-a-Programmable-Chip) Builder, a qual permite a construção de sistemas utilizando elementos de processamentos (processador Nios II) e periféricos interconectados pelo barramento Avalon. O acesso aos dispositivos é feito por meio de mapeamento em memória. A NocMAKER é uma ferramenta para desenvolvimento de NoC que oferece um ambiente de simulação e permite avaliação de desempenho, área e consumo. O Vampir é responsável por realizar a avaliação da comunicação paralela por meio dos arquivos traces OTF (Open Trace Format) gerados pelo sistema. A plataforma MPSoC, ilustrada na Figura 29, consistiu em um elemento de processamento Master (responsável pelo controle dos escravos) e quinze elementos de processamentos escravo conectados por uma NoC 4x4 2D-Mesh. O elemento de processamento Master tem acesso à memória SDRAM externa por meio de um controlador de memória conectado ao barramento Avalon. Cada elemento de processamento é composto por um processador Nios II que contém cache de dados de 2KB, cache de instruções de 4KB, com FPU, interface JTAG e serial. A conexão entre o PE e a NoC é realizada por meio da interface de rede (NI). A NoC implementada possui as seguintes características: 4x4 2D-Mesh, chaveamento *wormhole*, controle de fluxo *phase handshake* e roteamento distribuído (algoritmo de roteamento XY). O fabricante Altera oferece ao desenvolvedor que utiliza o processador Nios II uma IDE (Integrate Development Environment – Ambiente de Desenvolvimento Integrado) para desenvolvimento de software. Foram testados e implementados os dois principais modelos de programação paralela: (*i*) OpenMP (para arquitetura de memória compartilhada); e (*ii*) ocMPI (para arquitetura de memória distribuída). Para auxiliar na análise e no processo de diagnóstico de algum problema de desempenho, são utilizadas

as interfaces com o usuário, como por exemplo: console por meio da interface UART e console utilizando a infraestrutura da rede.

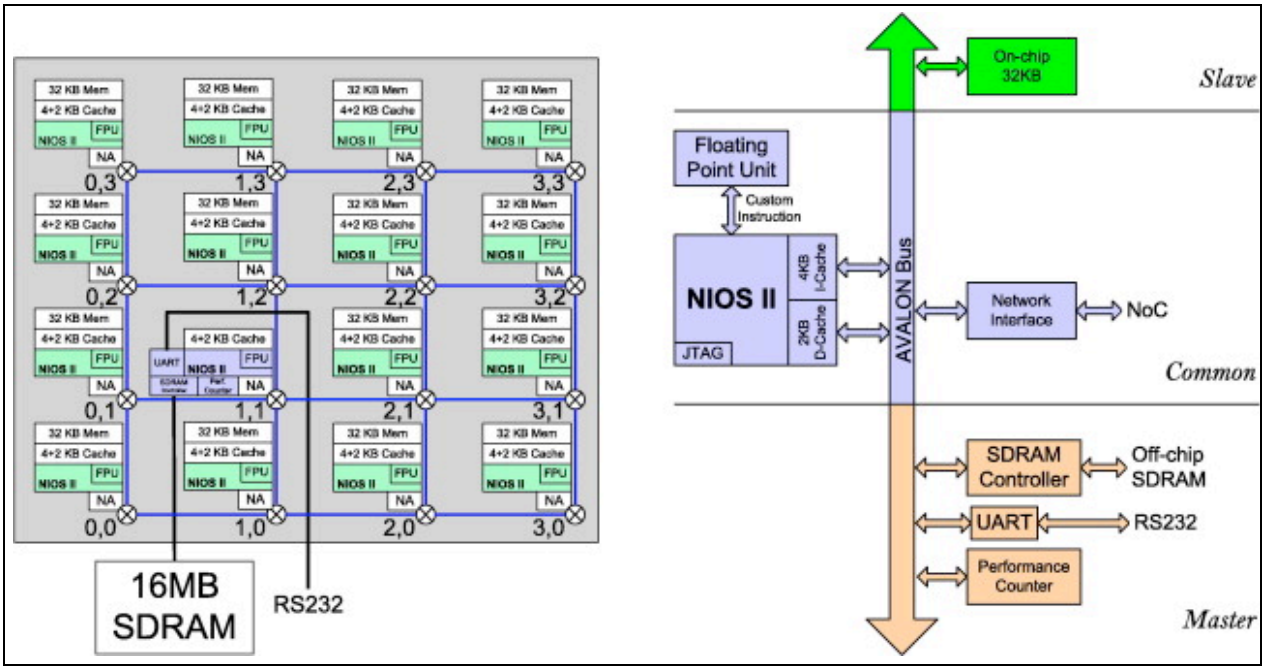


Figura 29. Plataforma MPSoC e estrutura interna do PE de Fernandez-Alonso et al.
Fonte: Fernandez-Alonso et al. (2010).

Um método alternativo para que não ocorra à sobrecarga de mensagens e gargalos nos canais de saída, consiste no uso de uma plataforma virtual baseada em simulador de conjunto de instrução (ISS – Instructio-Set Simulator). Este método intercepta as chamadas de funções durante a execução e produz *traces*, o qual é integrado ao NOCMAKER. Para realizar a simulação completa de um sistema MPSoC, é necessário instancializar o ISS entre o PE e a interface de rede. Foram criadas algumas funções para geração do arquivo OTF. Três aplicativos foram executados para avaliar a plataforma, tais como: (*i*) multiplicação de matriz que subdivide a matriz original em submatrizes de 30x30 elementos; (*ii*) aplicação igual à anterior, porém com submatrizes de 50x50 elementos; e (*iii*) a transformada de Hough para detectar círculos com raio 10 em imagens monocromáticas. A metodologia proposta permitiu analisar e diagnosticar problemas de desempenho por meio das ferramentas inseridas no sistema MPSoC.

### 3.1.10 Fernandez-Alonso et al. (2012)

Fernandez Alonso et al. (2012) implementaram um interface de comunicação por passagem de mensagem (ocMPI) a partir do padrão MPI (Message Passing Interface) destinado para a comunicação em sistemas com memória distribuídas. O ocMPI tem como objetivo oferecer um modelo de programação e uma linguagem para o programador de forma a prover um aumento na produtividade como também uma interface de comunicação entre elementos de processo. Além disso, OcMPI disponibiliza ao programador várias facilidades de otimização da aplicação e geração de *trace* para análise detalhada. O ocMPI foi desenvolvido em ANSI C a fim de minimizar o impacto das bibliotecas quando compiladas por outros processadores que suportem gcc, como por exemplo: Nios II, Microblaze e ARM. A biblioteca ocMPI é formada por onze primitivas, sendo apresentadas no Quadro *5*, apenas sete.

Quadro 5. Primitivas ocMPI

| **Primitivas** | **Descrição** |
|---|---|
| ocMPI_Init | Inicializa o ambiente de execução ocMPI |
| ocMPI_Finalize | Finaliza o ambiente de execução ocMPI |
| ocMPI_Comm_rank | Determina o identificador das chamadas do processo no comunicador |
| ocMPI_Comm_size | Determina o tamanho do grupo associado ao comunicador |
| ocMPI_Send | Realiza o envio básico (blocking send) |
| ocMPI_Recv | Realiza a recepção básica (blocking receive) |
| ocMPI_Wtime | Retorna o tempo da chamada do processo |

Fonte: Adaptado de Fernandez-Alonso et al (2012).

Em uma arquitetura com memória distribuída, as primitivas ocMPI são utilizadas para enviar e receber as mensagens. Contudo, é necessário utilizar um protocolo para sincronizar o envio e o recebimento das informações. Este protocolo pode ser, por exemplo: *eager* (a mensagem é enviada independentemente do estado do receptor) ou *rendezvous* (antes do envio da mensagem é realizado o processo de *handshake*).

O *Quadro 6* apresenta um comparativo entre as bibliotecas de passagem de mensagem disponíveis no mercado em relação à disponibilidade, ao tamanho e à quantidade de funções disponíveis. O MPICH e o OpenMPI não são bibliotecas proprietárias e suportam até 300 primitivas MPI com tamanho de 7MB e 25MB. Já os TDM-MPI, SoCMPI e RAMPSoC-MPI são soluções mais leves para sistemas embarcados. Estas soluções são similares ao ocMPI em relação ao tamanho da biblioteca e à quantidade de primitivas. Entretanto, são bibliotecas proprietárias para sistemas específicos e seu porte não é trivial. O ocMPI não é uma solução proprietária e pode ser utilizável em qualquer plataforma e arquitetura.

Quadro 6. Bibliotecas MPI

| **Biblioteca MPI** | **Disponibilidade** | **Tamanho da Biblioteca** | **Quantidades de primitivas** |
|---|---|---|---|
| openMPI | Não proprietária | *25MB* | *300* |
| MPICH | Não proprietária | *7MB* | *300* |
| TDM-MPI | Proprietária | *9KB* | *11* |
| SoCMPI | Proprietária | *11-16KB* | *6-18* |
| RAMSoC-MPI | Proprietária | *37KB* | *18* |
| ocMPI | Não proprietária | *8-14KB* | *7-11* |

Fonte: Adaptado de Fernandez-Alonso et al (2012).

### 3.1.11 Hartono et al. (2013)

Hartono et al. (2013) propuseram um sistema MPSoC implementado com NoC para interconexão entre os elementos de processamento e substituindo a tradicional matriz de interconexão AMBA-AHB. A arquitetura de interconexão NoC para o sistema embarcado MPSoC é baseada no ARM Cortex-M0 de 32 bits. O primeiro sistema proposto consistiu em uma plataforma MPSoC baseado em NoC, conforme ilustrado na Figura 30a. O nodo de processamento, conforme ilustrado na Figura 30b, consiste de um processador ARM Cortex-M0, uma memória de programa, uma memória de dados, interconexão interna por meio do barramento AHB-Lite e uma interface de rede AHB2NoC.

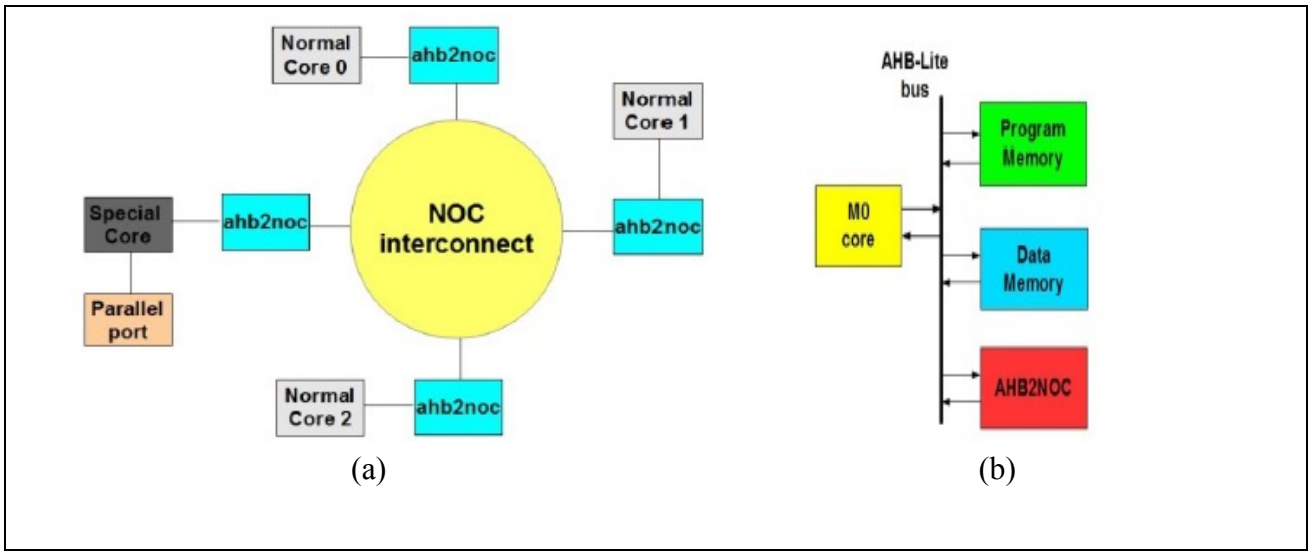


Figura 30. Arquitetura: (a) sistema MPSoC em NoC; e (b) núcleo de Hartono et al.
Fonte: Hartono et al. (2013).

A configuração e a geração da NoC foram obtidas por meio da ferramenta CONNECT NoC. Esta ferramenta permite a síntese da NoC (apenas em Verilog) como também realiza a configuração de alguns itens, tais como topologia, quantidade de roteadores e tamanho de dados do *flit*, entre outros. Para medir o desempenho do sistema, foi alterada a interconexão do sistema de NoC para matriz de interconexão AMBA-AHB, conforme ilustrado na Figura 31. Neste sistema, os processadores compartilham recursos por meio de matriz de interconexão.

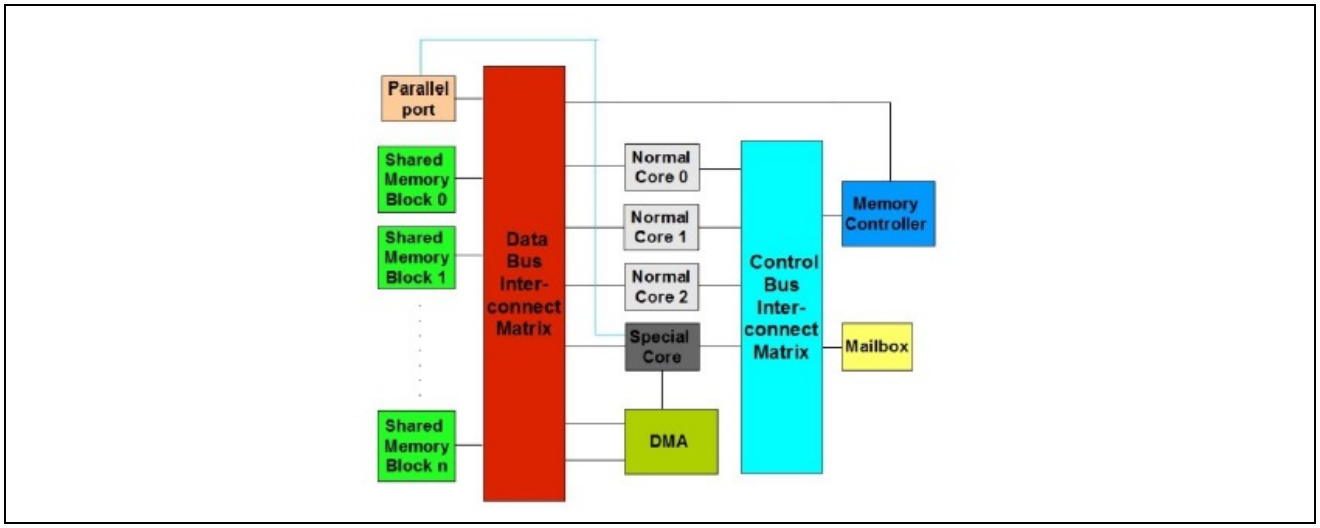


Figura 31. Arquitetura: sistema MPSoC em barramento de Hartono et al.
Fonte: Hartono et al. (2013).

Ambos os sistemas foram verificados funcionalmente utilizando a Metodologia de Verificação Universal (UVM – Universal Verification Methodology) baseado em *testbenches*. A aplicação executada nos dois sistemas MPSoCs (baseados em NoC e em barramento) realizava a criptografia *coarse-grain* AES dos dados a partir dos arquivos binários inseridos na memória de programa de cada processador. A *Tabela 1* apresenta a latência média, a vazão e a quantidade de dados enviados.

Tabela 1. Resultados da plataforma MPSoC de Hartono et al.

| | **Números dados (Kbytes)** | **Latência média (Ciclos)** | **Vazão (bits/ciclos)** |
|---|---|---|---|
| Matriz Interconexão AMBA-AHB | 1 | 5074,56 | 0,427 |
| | 2 | 5373,97 | 0,434 |
| | 4 | 5569,27 | 0,432 |
| | 8 | 5663,43 | 0,431 |
| NoC | 1 | 1233,29 | 1,336 |
| | 2 | 1259,53 | 1,329 |
| | 4 | 1272,65 | 1,325 |
| | 8 | 1279,22 | 1,323 |

Fonte: Adaptado de Hartono et al. (2013).

A partir dos resultados obtidos, o sistema baseado em interconexões em NoC apresentou melhor desempenho em relação à latência e à vazão quando comparado com o sistema baseado em matriz de interconexão AMBA-AHB.

## 3.2 SIMULAÇÃO

### 3.2.1 Mahadevan et al. (2005)

Mahadevan et al. (2005) propuseram um modelo de gerador TG integrado à plataforma de simulação MPSoC baseada em SystemC MPARM, em nível de bit e *cycle-true*, conforme ilustrado na Figura 32. A plataforma MPARM suporta: o barramento de interconexão AMBA (Advanced Microcontroller Bus Architecture), o barramento serial STBus e várias arquiteturas de comunicação NoC como, por exemplo, a *x*pipes.

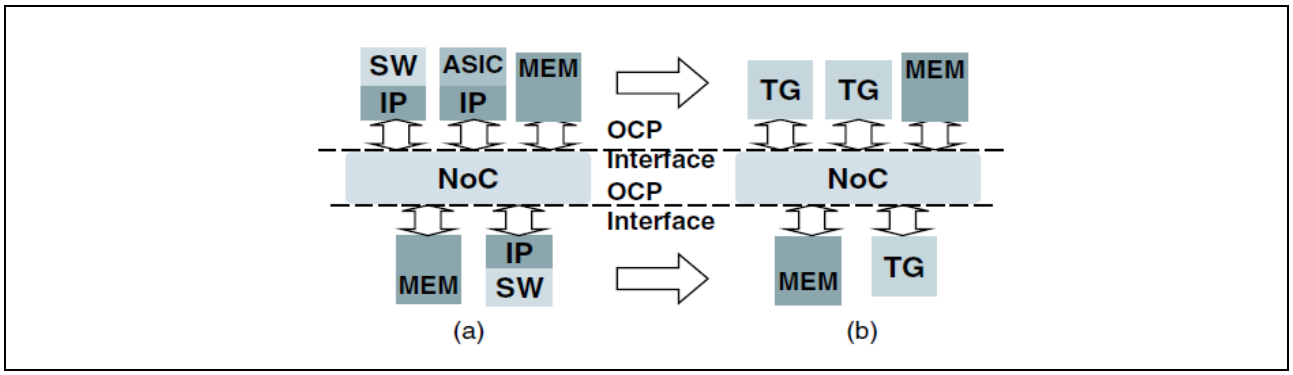


Figura 32. Plataforma de Simulação: (a) IP-*Core*; e (b) Modelo TGs de Mahadevan et al.
Fonte: Mahadevan et al. (2005).

A utilização do protocolo OCP (Open Core Protocol) entre os IP *cores* e a rede de interconexão permite reduzir o tempo de desenvolvimento, como também aumenta o reuso dos nodos. Antes da utilização dos geradores de tráfegos, o usuário pode utilizar um simulador com modelos IP no nível de *bit-true* e *cycle-true*. Durante a execução do simulador, as informações do *trace* são coletadas das interfaces OCP e armazenadas em arquivo *.trc*. O passo seguinte é realizar a conversão do arquivo *.trc* para um arquivo TG *.tgp*. Finalizado o processo de conversão é utilizado um montador (*assembler*) para converter o arquivo *.tgp* para binário *.bin*. Esse arquivo será carregado e executado na memória de instrução do TG. O processador do TG é um processador multiciclo com memória de instruções, mas sem memória de dados. Para avaliar o desempenho do TG, foram utilizados quatro *benchmarks* com aplicações de manipulação de matrizes: (*i*) aplicação em SPP; (*ii*) Cacheloop; (*iii*) Aplicação em MP (MultiProcessor); e (*iv*) aplicação em MP com codificação/decodificação no ambiente de simulação. O experimento permitiu comparar a velocidade fornecida pelo ARM com o modelo TG, quando submetidos a um sistema com até 12 processadores, considerando uma interconexão de barramento AMBA AHB. A métrica utilizada no experimento foi à aceleração (*speedup)*. Os resultados mostram a viabilidade da abordagem do modelo baseado TG com desacoplamento da simulação IP *core*. O modelo de TG proposto no barramento AMBA conseguiu aumentar a velocidade de simulação de 2x até 4x quando comparado com o IP *core*.

### 3.2.2 Tedesco et al. (2005)

Tedesco et al. (2005) propuseram um método genérico para avaliação de desempenho e geração de tráfego para uma rede NoC HERMES com topologia em malha 2D. Os modelos de

tráfego são definidos por três parâmetros: (*i*) distribuição espacial dos pacotes; (*ii*) taxa de injeção dos pacotes; e (*iii*) tamanho dos pacotes. A Figura 33 ilustra dois tipos de método para avaliação de desempenho em NoC: (*i*) avaliação externa, no qual a rede é considerada como uma caixa preta e os resultados dos tráfegos são obtidos por uma interface externa; e (*ii*) avaliação interna, no qual o desempenho é obtido em cada canal da rede.

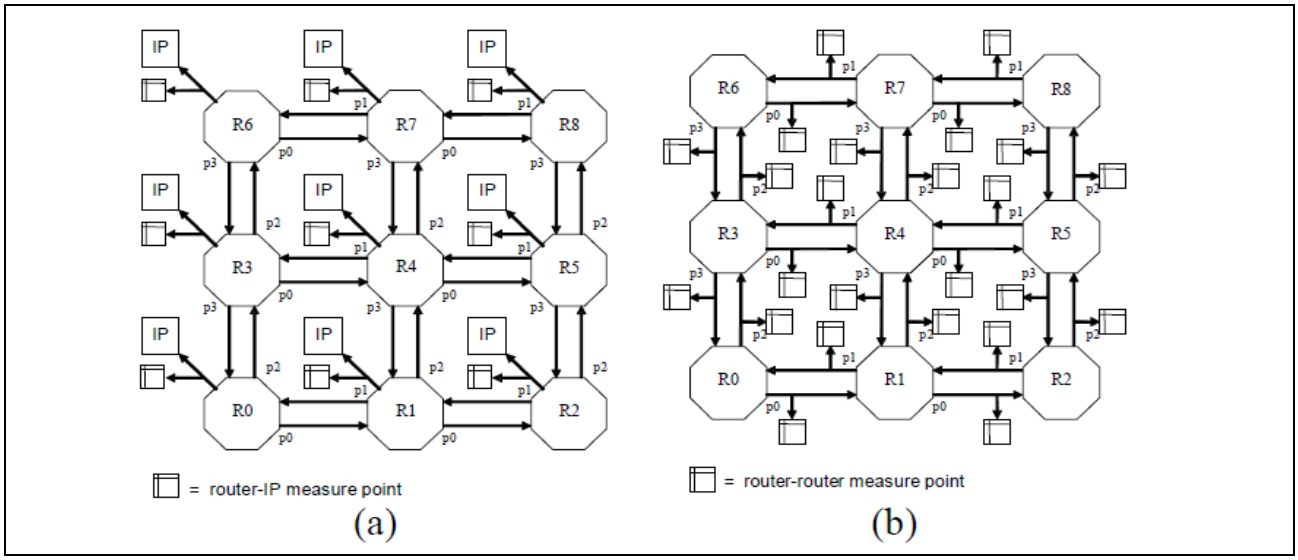


Figura 33. Avaliação de desempenho: (a) NoC Externa; e (b) Interna de Tedesco et al.
Fonte: Tedesco et al. (2005).

A avaliação de desempenho interna permite verificar o desempenho de cada canal na NoC de acordo com os parâmetros de estrutura e geração de tráfego. Duas métricas permitem avaliar o uso médio do canal: (*i*) largura de banda média do canal: apresenta o percentual utilizado pelo canal durante a transmissão dos pacotes; e (*ii*) vazão: informa a quantidade de bits transmitidos por unidade de tempo (bits/s). Já a métrica, tempo médio para transmissão de *flit*, permite avaliar o número médio de ciclos utilizados para transmitir um *flit*. A avaliação de desempenho externa utiliza para análise das informações o gráfico Chaos Normal Form (CNF) e a análise de distribuição de latência. O gráfico CNF apresenta informações de vazão (no primeiro gráfico) e latência da rede (no segundo gráfico). A distribuição de latência também é apresentada em relação ao número de pacotes. O experimento utilizou uma NoC HERMES com topologia em malha 2D 8x8, chaveamento por pacotes, controle de fluxo baseado em créditos, *flits* de 16 bits, *buffers* com capacidade de 8 *flits*, algoritmos de roteamento determinístico e parcialmente adaptativo (West-First), com e sem canais virtuais. A NoC foi descrita em VHDL e foram obtidas as métricas de uso

médio do canal, tempo médio para transmissão de *flits* e o gráfico CNF para avaliação da NoC interna. Essas métricas permitem ao desenvolvedor aperfeiçoar sua NoC para diferentes cenários de tráfego.

## 3.3 MONITORAMENTO

### 3.3.1 Ciordas et al. (2004)

Mesmo não tendo o enfoque no tema relacionado a geradores e medidores de tráfego se tornou interessante por apresentar uma alternativa de extração de dados em NoCs não intrusivo. Ciordas et al. (2004) propuseram um serviço de monitoramento baseado em pontos de monitoramento, *probes* de hardware, anexados aos componentes das NoCs (roteadores ou interface de rede), com as seguintes características: escalabilidade, flexibilidade, monitoramento não intrusivo, monitoramento em tempo real, monitoramento baseado modelos de eventos (*identifier, timestamp, producer, attributes*) e *setup* do serviço de monitoramento durante a execução. Para validação do serviço foi considerada a NoC Æthereal. A Figura 34 ilustra a arquitetura interna de um *probe* e o seu fluxo de configuração.

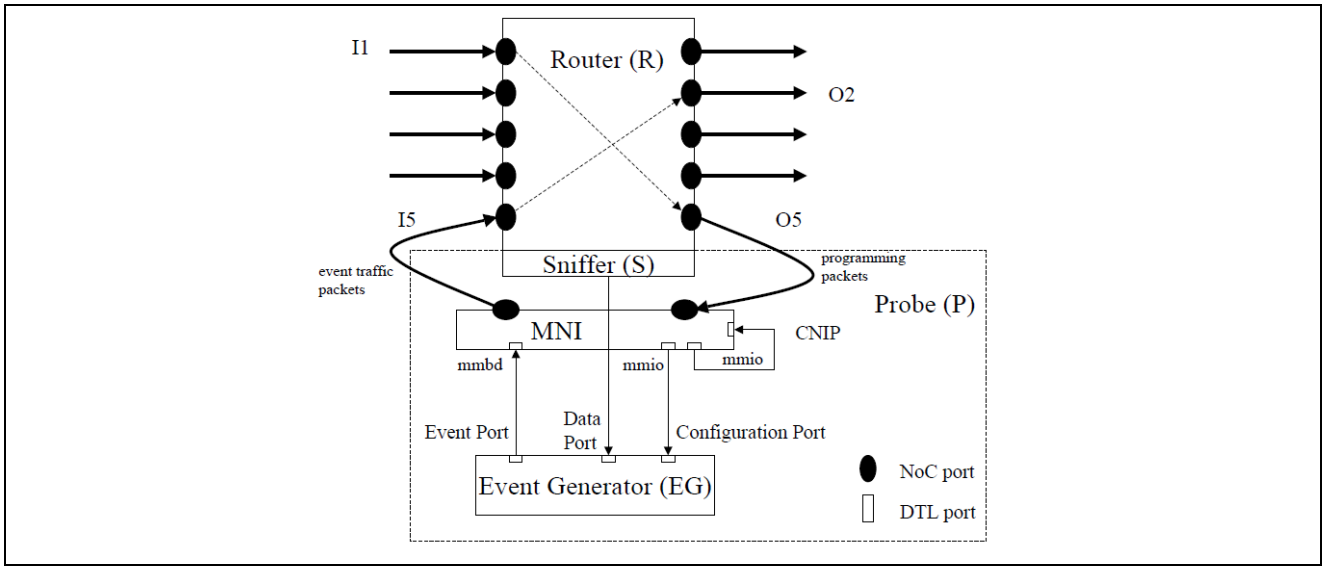


Figura 34. Arquitetura do ponto de monitoramento (*probe*) de Ciordas et al.
Fonte: Ciordas et al. (2004).

A arquitetura de um *probe* de hardware é composta pelos seguintes componentes: *sniffer* (S), gerador de eventos (EG – Event Generator) e interface de monitoramento de rede (MNI –

Monitoring Network Interface). A função do componente S é capturar as informações do rotador (R) e encaminhá-los à porta de dados do EG. Já o EG tem como função gerar os eventos *time stamped* com base nos dados recebidos do componente S. O MNI tem como função codificar os eventos gerados pelo EG e encaminhá-los para a NoC. Os serviços de monitoramento são classificados por: (*i*) modelo evento (*timestamped*); (*ii*) ponto de monitoramento (*probe*); (*iii*) modelo de programação (MAS – Monitoring Service Access Point); (*iv*) gerenciamento de tráfego (GT – Guaranteed Throughput ou BE – Best Effort); e (*v*) monitoramento de serviço de NoCs (distribuído ou centralizado). O tráfego gerado para realizar a configuração é de 4,8 KB/s, que representa uma magnitude seis vezes menor que 2 GB/s por enlace gerados pela rede NoC Æthereal. O gerenciamento de tráfego foi obtido através do reuso dos serviços.

### 3.3.2 Hou et al. (2009)

Hou et al. (2009) apresentaram um sistema de monitoramento de rede em tempo real utilizando canais virtuais com aplicações JPEG, que monitora o desempenho da NoC, por meio do cálculo da latência média e vazão. A plataforma consiste em uma NoC com topologia em malha 2D 2x3, com canais virtuais e 6 processadores ARM implementada no FPGA EP2S180 Stratix II da Altera. A arquitetura da plataforma é composta pelos seguintes módulos: (*i*) Elemento de Recurso (RE - Resource Element); (*ii*) Elemento de Comunicação (CE - Comunication Element), e (*iii*) Roteador. O RE é formado por vários IPs, tais como processadores, memórias e controle de interrupção que se comunicam por meio de um barramento compartilhado. Os REs têm como funções: roteamento (algoritmo XY), controle de fluxo, chaveamento, arbitragem e memorização. A interface de rede (RNI – Resource Network Interface) estabelece a conexão entre o RE e o CE, sendo responsável por empacotar e desempacotar os dados transferidos na rede. Quando um RE deseja se comunica com outro RE, o acesso à rede é realizado via RNI. A Figura 35a ilustra a integração dos REs com os CEs.

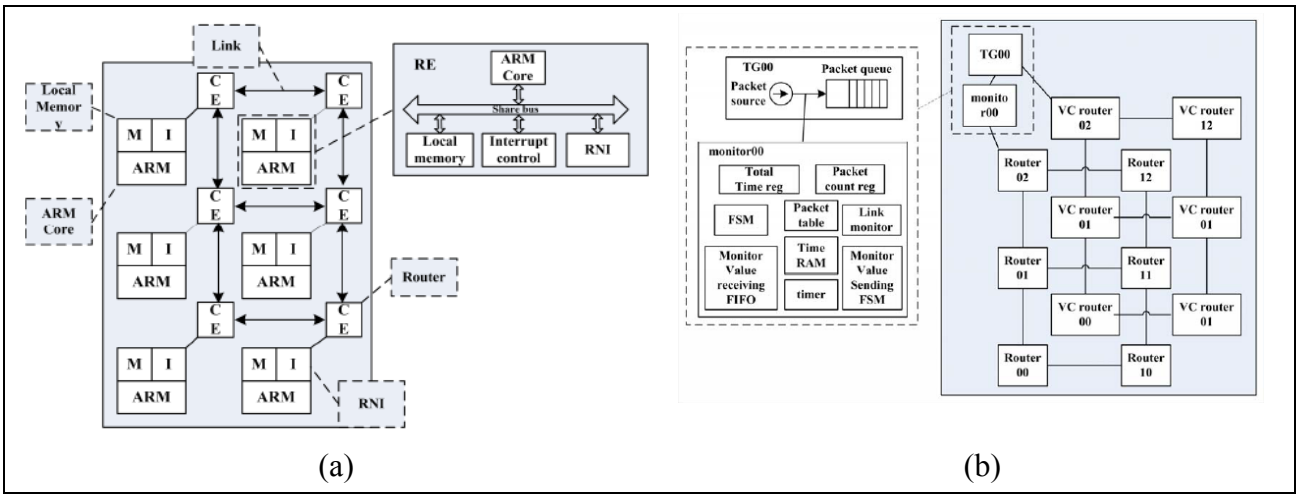


Figura 35. Sistema monitoramento: (a) Arquitetura; e (b) Monitor na rede de Hou et al.
Fonte: Hou et al. (2009).

O canal virtual é utilizado nessa proposta para evitar *deadlock*, reduzir a latência e aumentar a vazão. No roteamento com canal virtual (VC), cada canal físico é associado a várias filas pequenas, ao invés de uma única fila maior. A arquitetura do monitor de rede é ilustrada na Figura 35b. O gerador de tráfego foi implementado no processador ARM para geração de tráfego uniforme, com 16 *flits* por pacote, quatro canais virtuais por porta de entrada e executando aplicações reais de JPEG. O monitor apresenta uma interface JTAG com o computador para encaminhar os dados coletados. Foi avaliado o desempenho da plataforma com e sem a utilização dos canais virtuais. A plataforma com canais virtuais melhorou a vazão em 62% e diminui a latência média dos pacotes em relação à plataforma sem canais virtuais. Este sistema apresenta limitações em relação ao tamanho da NoC, o que o torna inviável para emulação de grandes NoCs.

### 3.3.3 Alhonen et al. (2010)

Alhonen et al. (2010) direcionaram sua pesquisa para o tema de monitoramento de NoCs. Para realizar a implementação da plataforma de monitoramento de NoCs, foram utilizadas duas placas: (*i*) Altera DE2, na qua

l foram implementados a NoC e os blocos monitor, controle UDP/IP e controle Ethernet; e (*ii*) Altera Stratix II S180, na qual foram implementados o chip Ethernet – PHY e MAC, conforme apresentado na Figura 36.

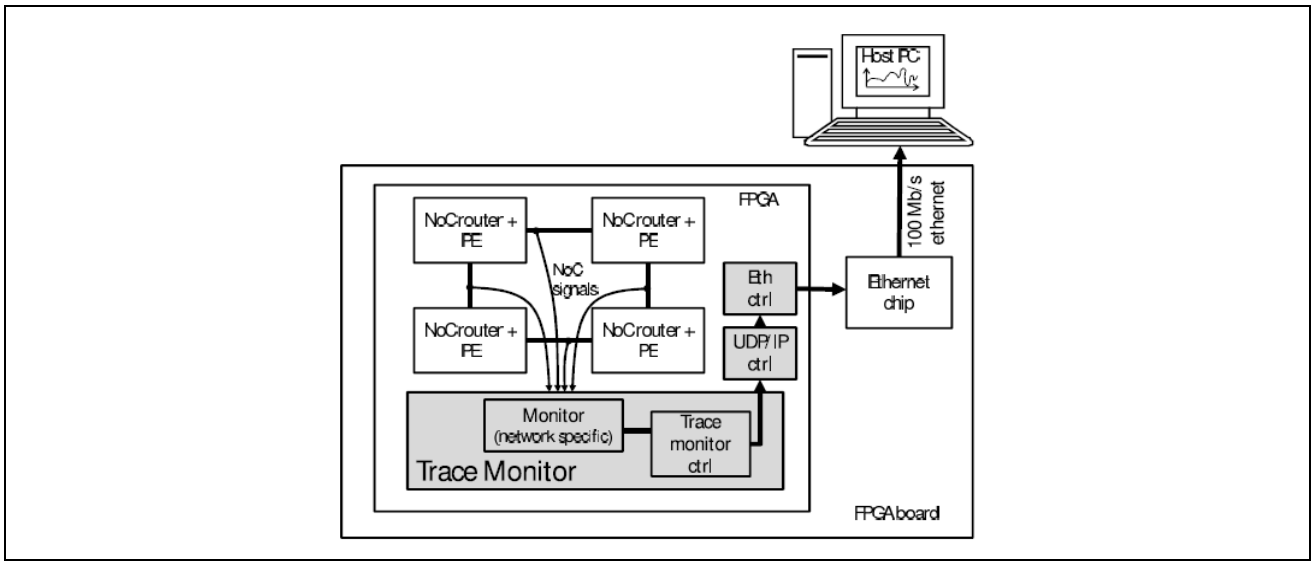


Figura 36. Plataforma de monitoramento de Alhonen et al.
Fonte: Alhonen et al. (2010).

O bloco Trace Monitor foi desenvolvido e sintetizado em VHDL com a função de capturar e processar as seguintes informações: (*i*) dados transferidos com sucesso pela rede; e (*ii*) dados disponíveis, mas não transferidos. Já o bloco Trace Monitor Ctrl envia os dados coletados do contador para a FIFO do bloco externo (Ethernet *chip*). O bloco UDP/IP Ctrl é responsável por gerar o cabeçalho e o processo de inicialização lógica entre o bloco Trace Monitor Ctrl e o componente externo Ethernet *chip*, sendo sintetizado em VHDL e controlado pelo bloco Eth Ctrl. Esta ferramenta pode realizar a avaliação das medidas e a análise da NoC conseguindo atender ao mesmo tempo, ao requisito de precisão e escala. O Trace Monitor mostrou ser uma ferramenta capaz e versátil para realizar análise de sistema em plataforma real.

## 3.4 ANÁLISE COMPARATIVA

O Quadro *7* apresenta um resumo das informações a respeito dos trabalhos sobre emulação de NoC em FPGA, destacando as seguintes informações: tecnologia de processador, modelo do FPGA, arquitetura MPSoC, tipo de geração de tráfego, NoC, topologia, métricas e avaliação de área. Todas as NoCs utilizadas adotaram topologia em malha 2D. A maioria dos trabalhos apresenta uma solução de plataforma para avaliação de desempenho em NoC para gerar modelos de tráfego considerando qualquer tipo de arquitetura de NoC.

Quadro 7. Análise comparativa

| Trabalhos Relacionados / Emulação | Tecnologia do Processador | FPGA | MPSoC | Tipo de Geração Tráfego | NoC | Topologia | Métricas | Avaliação de área |
|---|---|---|---|---|---|---|---|---|
| Genko et al. (2005) | GPP/ PowerPC | Virtex II | NÃO | Estocástica / Trace-Driven | *n.i.* | Malha 2D | Aceleração | SIM |
| Wolkotte, Holzenspies, Smit (2007) | GPP / ARM 9 | Stratix II | NÃO | *n.i.* | *n.i.* | Malha 2D | Latência e Vazão | SIM |
| Pereira e Zeferino (2008) | SPP / NI | Cyclone II | NÃO | Estocástica | SoCIN | Malha 2D | Latência e Vazão | SIM |
| Luo-Feng et al. (2008) | GPP / ARM | Stratix II | NÃO | Aplicação Real | *n.i.* | Malha 2D | Aceleração | SIM |
| Alohonen et al. (2010) | GPP / Nios II | Stratix II | SIM | Aplicação Real | *n.i.* | Malha 2D | Dados transferidos e não transferidos | SIM |
| Fernandez-Alonso et al. (2010) | GPP / Nios II | Cyclone II | SIM | Aplicação Real | *n.i.* | Malha 2D | Aceleração | SIM |
| Lotlikar, Pai, Gratz (2011). | GPP / Microblaze | Virtex 5 | SIM | Estocástica / Trace-Driven | *n.i.* | Malha 2D | Aceleração | SIM |
| Heck et al. (2012) | SPP / NI | Virtex-II, Virtex 5 | NÃO | Estocástica | Hermes 2VC, Hermes GLP | Malha 2D | Latência, quantidade de pacotes recebidos. | SIM |
| Hartono et al. (2013) | GPP / ARM | Córtex-M0 | SIM | Aplicação Real | *n.i.* | Malha 2D | Dados transferidos, latência e vazão. | NÃO |
| Esta dissertação | GPP / Nios II | Cyclone IV | SIM | Aplicação Real, Estocástica | SoCIN | Malha 2D | Aceleração, latência e vazão | SIM |

Legenda: *n.i.* – não informado pelos autores.

Por meio da análise dos trabalhos encontrados, foram identificadas as propostas de plataforma para avaliação de desempenho, geradores e medidores de tráfego, como também as principais métricas utilizadas. Foi possível observar que a maioria dos trabalhos realizou avaliação de desempenho de maneira externa e os geradores de tráfego foram implementados em processadores de propósito geral, utilizando como métricas latência e vazão. A emulação do sistema em hardware permite que o tempo de avaliação do sistema seja rápido comparado com simulação que, em muitos casos, se torna inviável a utilização.

Esta dissertação se propôs a desenvolver uma plataforma MPSoC baseada em processadores programáveis para avaliação do desempenho da NoC SoCIN em FPGA utilizando o processador Nios II. A arquitetura do sistema MPSoC proposto tem estrutura homogênea e a comunicação entre os processadores é feita por meio do método de passagem de mensagem ocMPI.

# 4 ARQUITETURA DE MPSoC PARA AVALIAÇÃO DO DESEMPENHO DE NOC EM FPGA

Este capítulo apresenta o projeto de uma plataforma MPSoC de arquitetura homogênea baseada em um processador de propósito geral com função de geração e medição de tráfego de avaliação do desempenho da NoC SoCIN em FPGA. O capítulo apresenta uma visão geral da plataforma, a análise de requisitos e o projeto arquitetural.

## 4.1 VISÃO GERAL

A plataforma MPSoC para avaliação do desempenho da NoC SoCIN em FPGA, baseia-se na arquitetura desenvolvida por Zeferino et al. (2007) e modelada em SystemC. A solução proposta foi formada pela rede SoCIN, nodos de processamento (NP), nodo supervisor (NS), computador Supervisor (S) e nodo de monitoramento (NM). Foram baseados em um subsistema constituído de um processador embarcado (Nios II), memória local e periféricos conectados à SoCIN por meio de interfaces de rede.

Os nodos de processamento (NPs) realizam a geração do tráfego à NoC SoCIN. A funcionalidade de geração de tráfego teve como referência o método genérico de injeção de tráfego proposto por Tedesco (2005). O nodo supervisor realiza a configuração dos geradores de tráfego dos nodos de processamento (NPs). A plataforma proposta foi implementada na placa de desenvolvimento Mercúrio IV (Development and Education Board) do fabricante Macnica, a qual utiliza um FPGA da Família Cyclone IV da Altera, podendo também ser portada para outros kits de desenvolvimento baseados em FPGAs da Altera. Essa plataforma busca ser mais flexível que a desenvolvida por Pereira (2008) e Pizzoni (2010), que eram baseadas em processadores não programáveis de propósito específicos, como também ter melhor desempenho que a desenvolvida por Frantz (2008), que era baseada em um processador programável de 8 bits (o microcontrolador PicoBlaze). O nodo de medição de tráfego foi baseado em um subsistema constituído de um processador embarcado (Nios II), memória local, blocos de aquisições de medidas e periféricos, conectados à porta local dos roteadores da NoC SoCIN. Este nodo é responsável por coletar as informações necessárias para o cálculo das métricas de desempenho.

## 4.2 ANÁLISE DE REQUISITOS

Esta seção apresenta os requisitos funcionais e não-funcionais definidos para a plataforma proposta na fase de projeto.

### 4.2.1 Requisitos Funcionais

- RF01: A configuração dos fluxos nos NPs deve utilizar a própria rede SoCIN como meio de comunicação entre o NS e os NPs. Os arquivos de configuração de tráfego devem ser enviados pelo Supervisor ao NS utilizando uma porta de comunicação serial. Cada arquivo de configuração deve ser transferido pela própria rede aos NPs, na forma de pacotes. Os pacotes recebidos pelos NPs serão armazenados na memória para posterior utilização;
- RF02: A geração de pacotes deve ser iniciada a partir de um comando recebido do NS;
- RF03: Ao criar e enviar um pacote, o NP deve incluir no cabeçalho os endereços de rede do nodo emissor (fonte) e do nodo receptor (destinatário);
- RF04: O NP deve ser capaz de enviar pacotes baseados nos descritores de fluxos armazenados na memória local;
- RF05: O NP deve ser capaz de gerar fluxos de comunicação para diferentes destinatários na rede;
- RF06: O NP deve ser capaz de gerar diferentes fluxos de comunicação para um mesmo destinatário na rede;
- RF07: Cada pacote enviado deve conter informações que permitam a identificação do pacote para posterior análise de desempenho;
- RF08: O NP deve escalonar o envio de pacotes de cada fluxo de forma balanceada, evitando que um determinado fluxo seja postergado indefinidamente;
- RF09: O NP deve ser capaz de receber pacotes a ele destinados, retirando-os da rede; e
- RF10: O NM deve ser capaz de coletar e armazenar as informações sobre os pacotes injetados e ejetados da rede na memória compartilhada com o NS.

### 4.2.2 Requisitos Não-Funcionais

- RNF01: O NS, os NPs e o NM devem possuir interface de comunicação compatível com o enlace da rede SoCIN;
- RNF02: O NS, os NPs e o NM devem ser baseados no processador Nios II da Altera;
- RNF03: Os NPs e o NM devem possuir uma memória local;
- RNF04: O NS deve possuir uma memória externa;
- RNF05: A rede SoCIN deve ser baseada no modelo sintetizável do roteador ParIS;
- RNF06: A plataforma proposta deve ter um custo em área de silício que permita a integração de sistemas com pelo menos quatros NPs em kits de prototipação educacionais (Ex. Terasic DE2, Macnica/DHW Mercurio IV, Terasic DE1-SoC);
- RNF07: A configuração de tráfego deve ser feita pela ferramenta *gtr* que permite que tal configuração seja realizada utilizando o modelo de injeção proposto por Tedesco (2005); e
- RNF08: Cada fluxo deve ser definido por um descritor de fluxo. Os campos que compõem o descritor de fluxo são: quantidade de pacotes, tamanho do pacote, taxa de injeção requerida e endereço do destinatário.

## 4.3 FLUXO DE PROJETO

A Figura 37 apresenta as etapas do fluxo de projeto no desenvolvimento da plataforma proposta. Foram utilizadas as ferramentas da Altera para apoio ao projeto com seus FPGAs, incluindo: Quartus II, Qsys e Nios II (IDE para o desenvolvimento de aplicações para o processador Nios II). O fluxo é composto de três partes principais: Sistema, Hardware e Software.

- Sistema: A partir da especificação da plataforma, é possível realizar a criação, a integração e a interconexão dos componentes no sistema proposto por meio da ferramenta Qsys, que está inclusa no software Quartus II. Os componentes que constituem os nodos da plataforma (NS, NP e NM), incluindo o processador Nios II, a memória e os periféricos, estão disponíveis na biblioteca de componentes da ferramenta

Qsys (ou seja, são IPs[5] da própria Altera). O Qsys permite a criação de IPs proprietários, como a interface de rede e o bloco de aquisição de medidas necessária à plataforma. Após gerar o sistema da plataforma, é realizada a integração com a parte de hardware (ferramenta Quartus II) e de software (ferramenta Nios II);

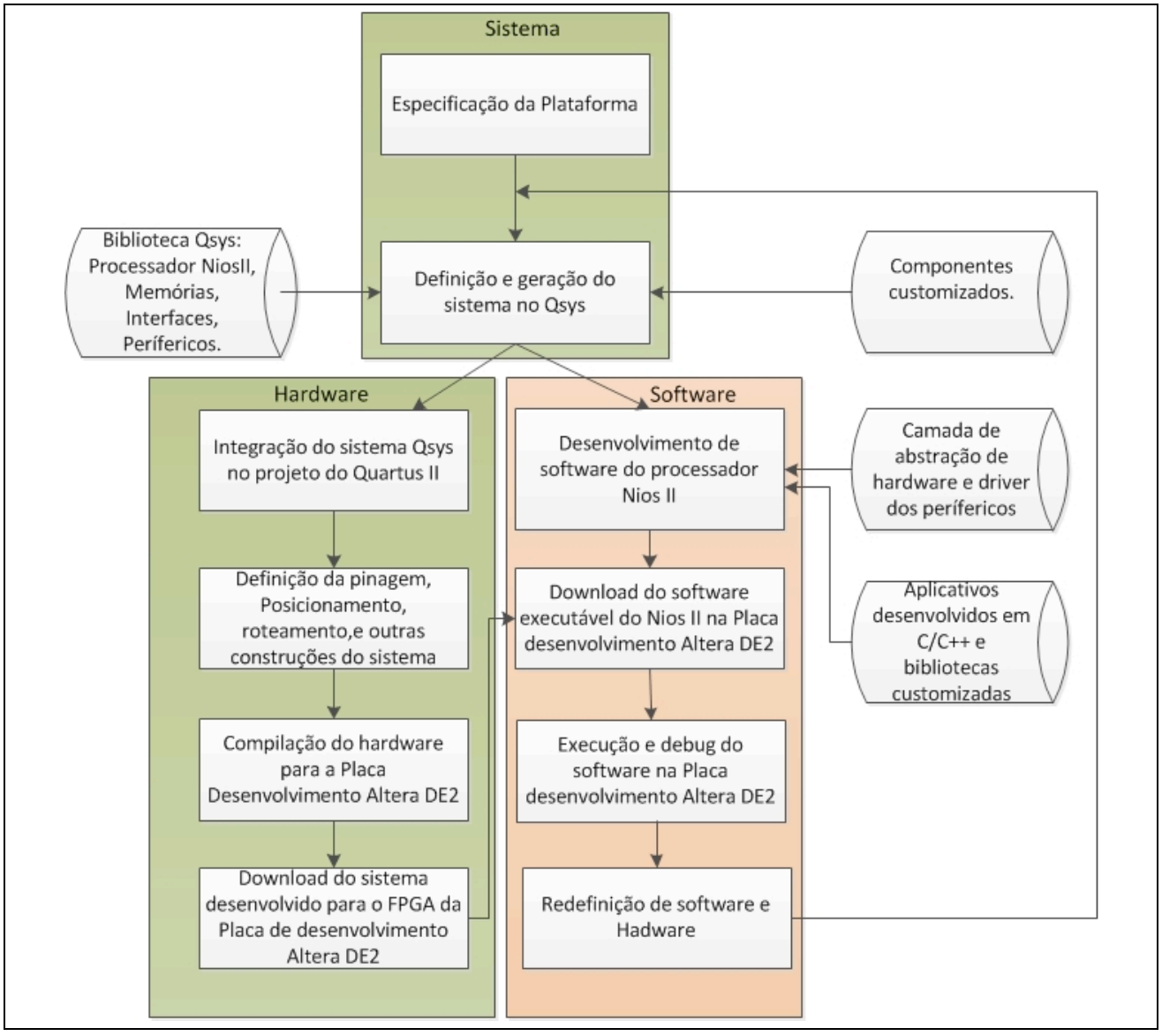


Figura 37. Fluxograma de integração: Qsys, software Quartus II e Nios II

- Hardware: A ferramenta Quartus II é utilizada no desenvolvimento dos IPs (interface de rede e bloco de aquisição) e da plataforma MPSoC. Após a descrição do modelo da plataforma, é realizada a síntese para o dispositivo alvo e a transferência do arquivo de configuração para o FPGA da placa de desenvolvimento; e

[5] IP – Intellectual Property *blocks*: modelos sintetizáveis de components de hardware.

- Software: Finalizadas as etapas de geração da plataforma (Sistema e Hardware), o software Nios II é usado na implementação das funcionalidades dos nodos NS, NP e NM, as quais são codificadas em linguagem C. A biblioteca ocMPI é inserida no projeto de software neste momento. Com o software finalizado, são realizadas a compilação e a transferência do software para a placa de desenvolvimento. A partir deste momento, a plataforma MPSoC está pronta para a realização dos experimentos.

Deve-se destacar que, uma vez definida uma plataforma de tamanho específico para integração na placa de desenvolvimento, ela pode ser transferida e mantida permanentemente. Após, diferentes experimentos podem ser realizados sobre a mesma plataforma, necessitando apenas que o Supervisor (S) envie ao NS a configuração do experimento. Portanto, o fluxo de projeto ilustrado na Figura 37, só é executado na fase de desenvolvimento da plataforma e em eventuais manutenções/atualizações.

## 4.4 ARQUITETURA DA PLATAFORMA

A plataforma proposta, representada na Figura 38, é formada por: (*i*) rede SoCIN (NoC com topologia em malha 2D e tamanho 3x2); (*ii*) computador Supervisor (S); (*iii*) nodo supervisor (NS); (*iv*) nodo de monitoramento (NM); e (*v*) um conjunto de nodos de processamento (NP).

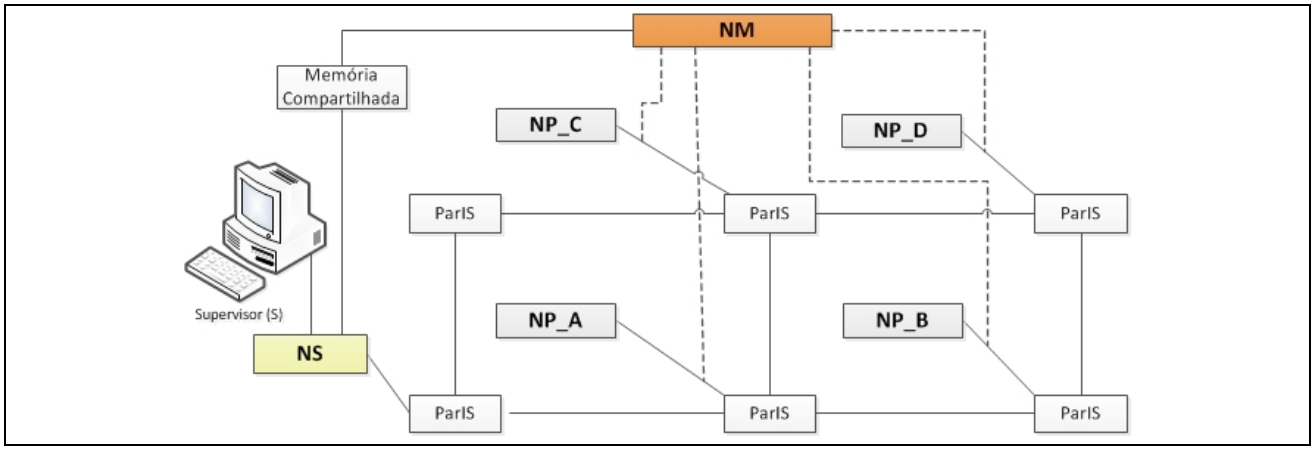


Figura 38. Estrutura da plataforma MPSoC proposta

O NS possui uma interface de comunicação serial com o Supervisor, pela qual recebe a configuração dos geradores de tráfego dos NPs. Esse nodo é responsável pela configuração dos NPs, controle do experimento (configuração e habilitação), como também pelo envio dos dados coletados pelo NM ao Supervisor para cálculo de desempenho. Os dados coletados pelo NM na rede

SoCIN são armazenados na memória compartilhada com o NS (ilustrada na figura). Os NPs são responsáveis pela geração dos pacotes na rede SoCIN utilizando o método de injeção de tráfego proposto por Tedesco (2005), como também pelo envio da informação de término do experimento ao NS. Para um experimento de avaliação do desempenho de uma NoC 2x2, é preciso implementar uma rede 3x2 para conectar o NS e suportar os caminhos necessários entre este e os demais nodos, conforme as restrições do algoritmo de roteamento. Por exemplo, no algoritmo XY, todo pacote deve percorrer primeiramente os enlaces da direção X antes de utilizar os enlaces da direção Y. Dessa forma, o roteador do canto superior esquerdo cumpre o papel de estabelecer o único caminho permitido para envio de pacotes dos nodos de processamento C e D para o nodo supervisor (NS).

As configurações dos geradores de tráfego do sistema são repassadas ao Supervisor por meio da ferramenta de simulação RedScarf com o uso do fluxo de simulação e avaliação de desempenho ilustrado na Figura 39, o qual foi originalmente desenvolvido para o ambiente BrownPepper (ZEFERINO et al., 2007). O RedScarf é formado pelo conjunto de ferramentas: (*i*) *gnoc*: responsável pela geração das redes SoCIN (SoCIN.h); (*ii*) *gsys*: responsável pela geração da descrição do sistema (main.cpp); (*iii*) *gtr*: responsável pela geração do arquivo de configuração de tráfego (traffic.cfg); (*iv*) *gcc*: compilador C responsável pela geração do sistema de simulação a partir do arquivo main.cpp (sistem.x); e (*v*) *atr*: responsável pela análise dos dados coletados.

O Supervisor utiliza apenas o arquivo .cfg produzido pela ferramenta *gtr* do RedScarf, a qual permite que o desenvolvedor configure os parâmetros do gerador de tráfego, tais como: tipo de distribuição espacial, tipo de injeção e parâmetros específicos do modelo de injeção selecionado.

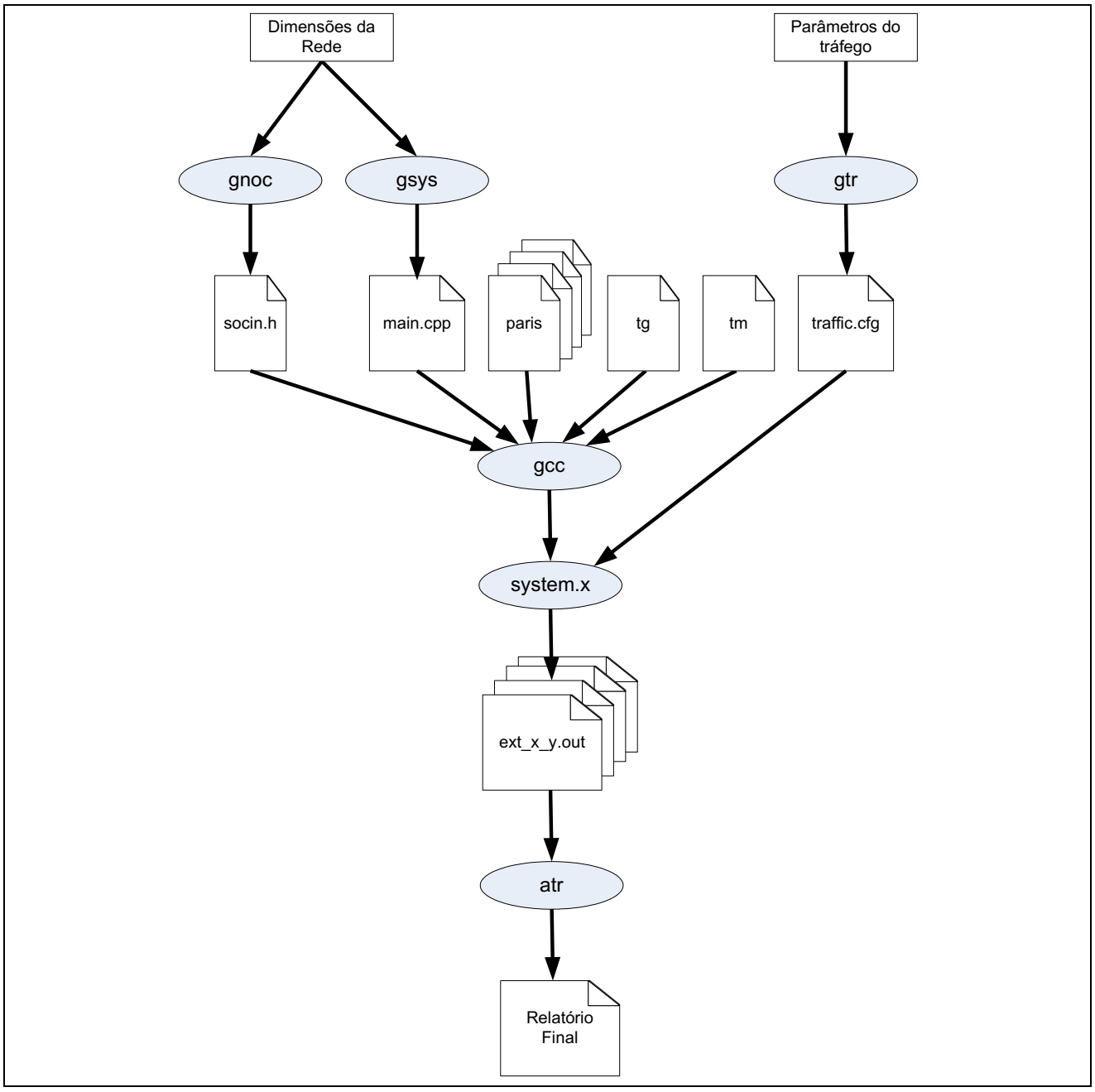


Figura 39. Estrutura da ferramenta de avaliação de desenvolvimento RedScarf
Fonte: Zeferino et al. (2007)

Conforme ilustrado na Figura 40, a plataforma MPSoC proposta apresenta o fluxo do processo de configuração, habilitação, coleta de dados e finalização de um experimento de emulação. Abaixo da figura são descritos os processos com detalhes.

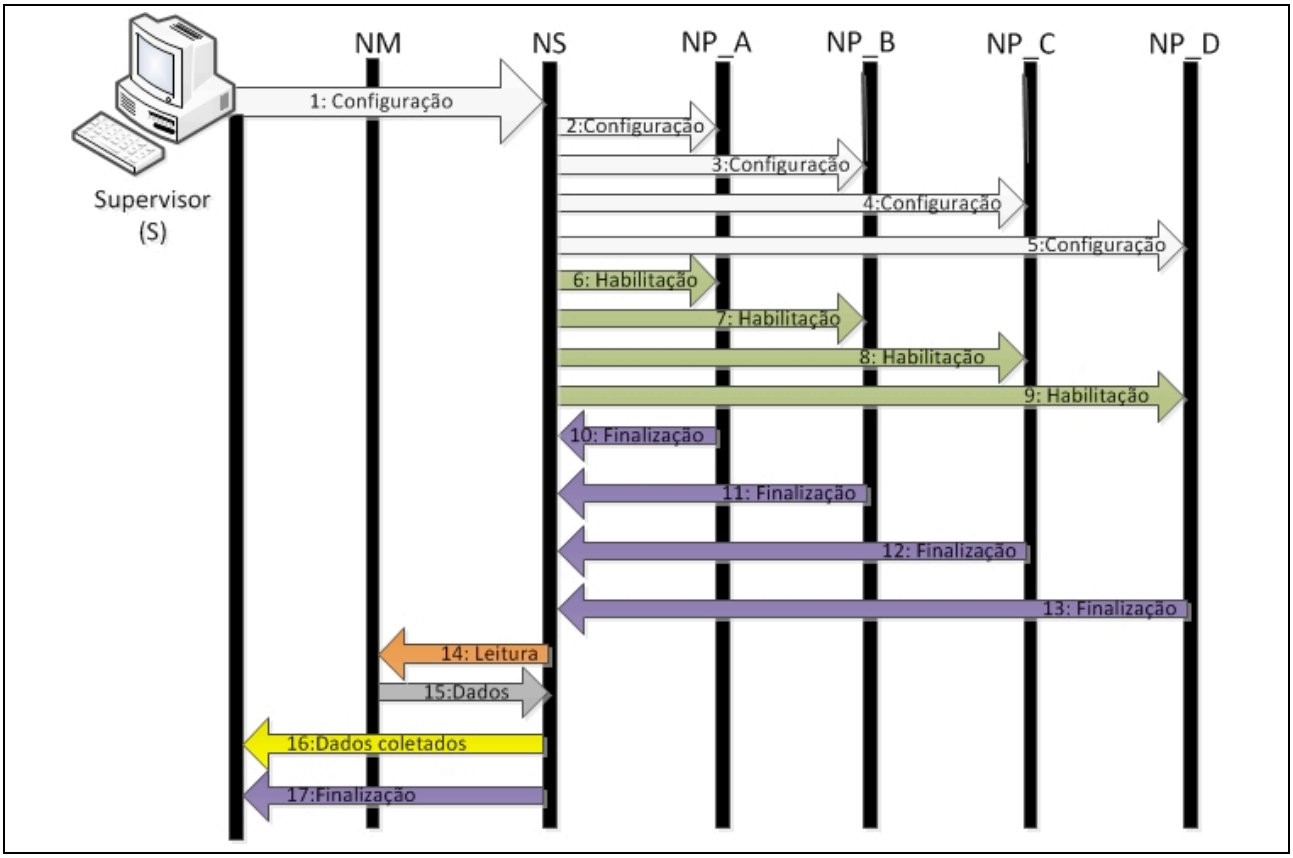


Figura 40. Fluxos dos processos: configuração, habilitação, finalização e leitura

- Processo de configuração: Na fase de configuração, o Supervisor encaminha uma solicitação ao NS para que este configure a geração de tráfego na rede (passo: 1). Após receber a solicitação de configuração de tráfego, o NS encaminha para todos os NPs o descritor do fluxo que deverá ser gerado. Observa-se que é encaminhada uma solicitação para cada NP (passos: 2, 3, 4, 5);

- Processo de habilitação: Após o último descritor de fluxo ser enviado, o NS inicia a fase de habilitação dos geradores de tráfego nos NPs. Todas as habilitações são encaminhadas sequencialmente (passos: 6, 7, 8, 9), pois a rede não suporta *multicasting*. Uma vez que os NPs estejam habilitados, é realizada a geração de tráfego na rede, como também a extração das informações dos pacotes recebidos e gerados pelo NM, ou seja, o experimento de avaliação do desempenho da NoC;

- Processo de finalização: Após finalizar a geração de tráfego na rede, cada NP encaminha ao NS a mensagem de finalização com o objetivo de informar o término do experimento de emulação (passos: 10, 11, 12, 13). Isso só é possível porque os geradores são configurados para gerar uma quantia de pacotes pré-determinados. O NS informa o

término do experimento (passo: 17) ao Supervisor apenas após o envio dos dados coletados pelo NM (passo: 16); e

- Processo de leitura dos dados: Ao término da fase de finalização dos NPs, o NS realiza a leitura na memória compartilhada onde foram armazenados os dados coletados da rede pelo NM (passos: 14, 15). Os dados armazenados na memória compartilhada são encaminhados ao Supervisor, para posterior cálculo do desempenho(passo: 16).

### 4.4.1 Projeto dos nodos

O fabricante Altera disponibiliza o processador embarcado Nios II em três versões (*economic*, *standard* e *full*), oferecendo ao desenvolvedor maior flexibilidade na configuração da arquitetura (em relação à CPU, aos periféricos e à memória), alto desempenho, baixo custo e baixo risco de tornar-se obsoleto (ALTERA, 2011, p. 5-14).

Neste trabalho, é utilizada a versão *standard* do processador Nios II, a qual é chamada Nios II/s. Segundo Altera (2011, p. 5-14), essa versão é destinada para soluções que tenham pouco processamento, requeiram baixo custo e utilizem aplicações de médio desempenho. O processador Nios II/s tem como principais características: (*i*) arquitetura RISC de 32 bits; (*ii*) cache de instrução; (*iii*) endereçamento externo de até 2 GB; (*iv*) *pipeline* de 5-estágios; (*v*) hardware para operações aritméticas de multiplicação e divisão; (*vi*) opção de deslocamento por hardware; (*vii*) suporte para até 256 instruções personalizadas; e (*viii*) módulo de depuração baseado na interface JTAG.

A Tabela 2 apresenta as características do dispositivo EP4CE30F23C7 FPGA da família Cyclone IV da Altera, utilizado na placa de desenvolvimento Mercúrio IV do fabricante Macnica.

Tabela 2. Recursos do dispositivo EP4CE30F23C7 - Família Cyclone IV

| **Recursos** | **EP4C30F23C7** |
|---|---|
| Elementos lógicos (LEs) | 28,848 |
| Memória interna (Kbits) | 594 |
| Multiplicadores Internos 18x18 | 66 |
| PLLs | 4 |
| Rede Clock Global | 20 |
| Blocos I/O | 8 |
| Máximo I/O | 532 |

Fonte: Adaptado de Altera (2016).

A plataforma proposta utiliza uma arquitetura MPSoC homogênea em que os nodos de processamento são constituídos de um processador (Nios II), memória local, interface de rede, além de periféricos, conforme apresentado na Figura 41. O bloco correspondente ao nodo processamento (NP) é responsável pelo processo de geração de tráfego.

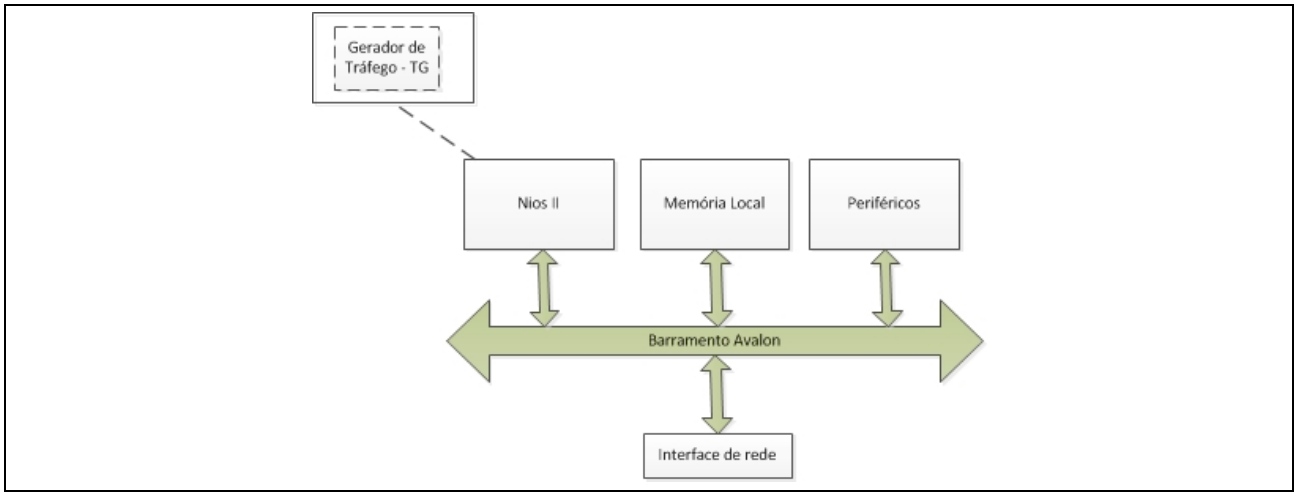


Figura 41. Estrutura interna do nodo de processamento (NP) proposto

O nodo de monitoramento é constituído por processador (Nios II), memória local, mutex, blocos de aquisição de medidas, além de periféricos. A Figura 42 apresenta a estrutura interna do nodo de monitoramento considerando uma rede SoCIN 3x2.

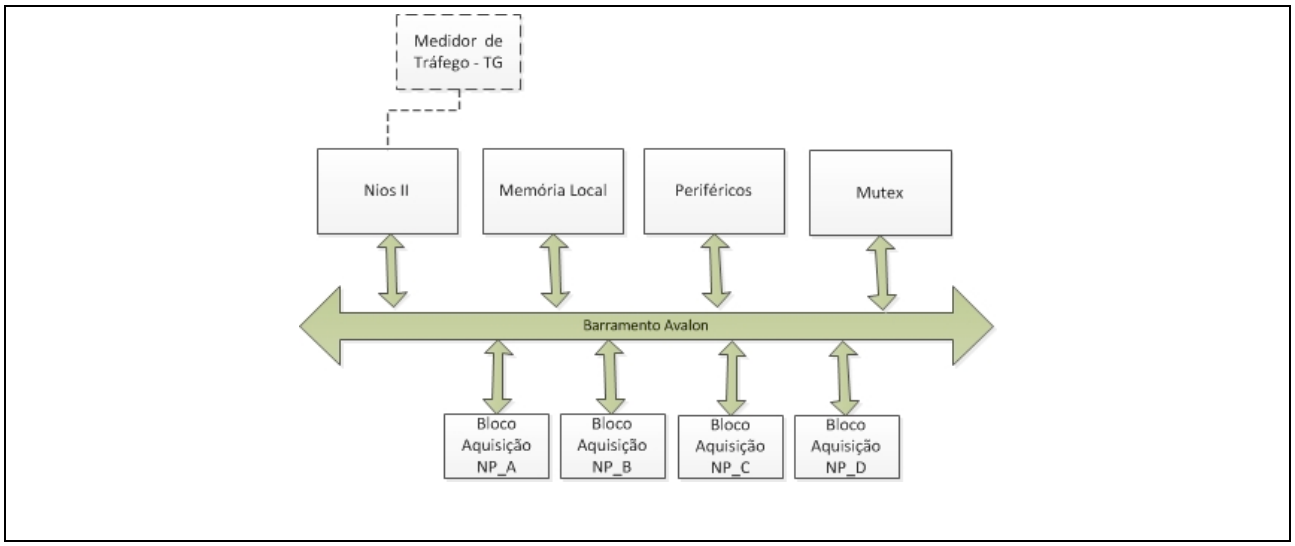


Figura 42. Estrutura interna do nodo de monitoramento (NM) proposto

Cada bloco de aquisição de medida é acoplado na interface local de cada roteador da rede de forma a realizar a captura dos pacotes recebidos e transmitidos. A arquitetura proposta para os

nodos (NS, NP e NM), apresentada na Figura 43, é composta dos blocos identificados na figura e descritos logo a seguir. Os blocos hardware gerados a partir da biblioteca de componentes do *framework* da Altera são identificados como IP Altera.

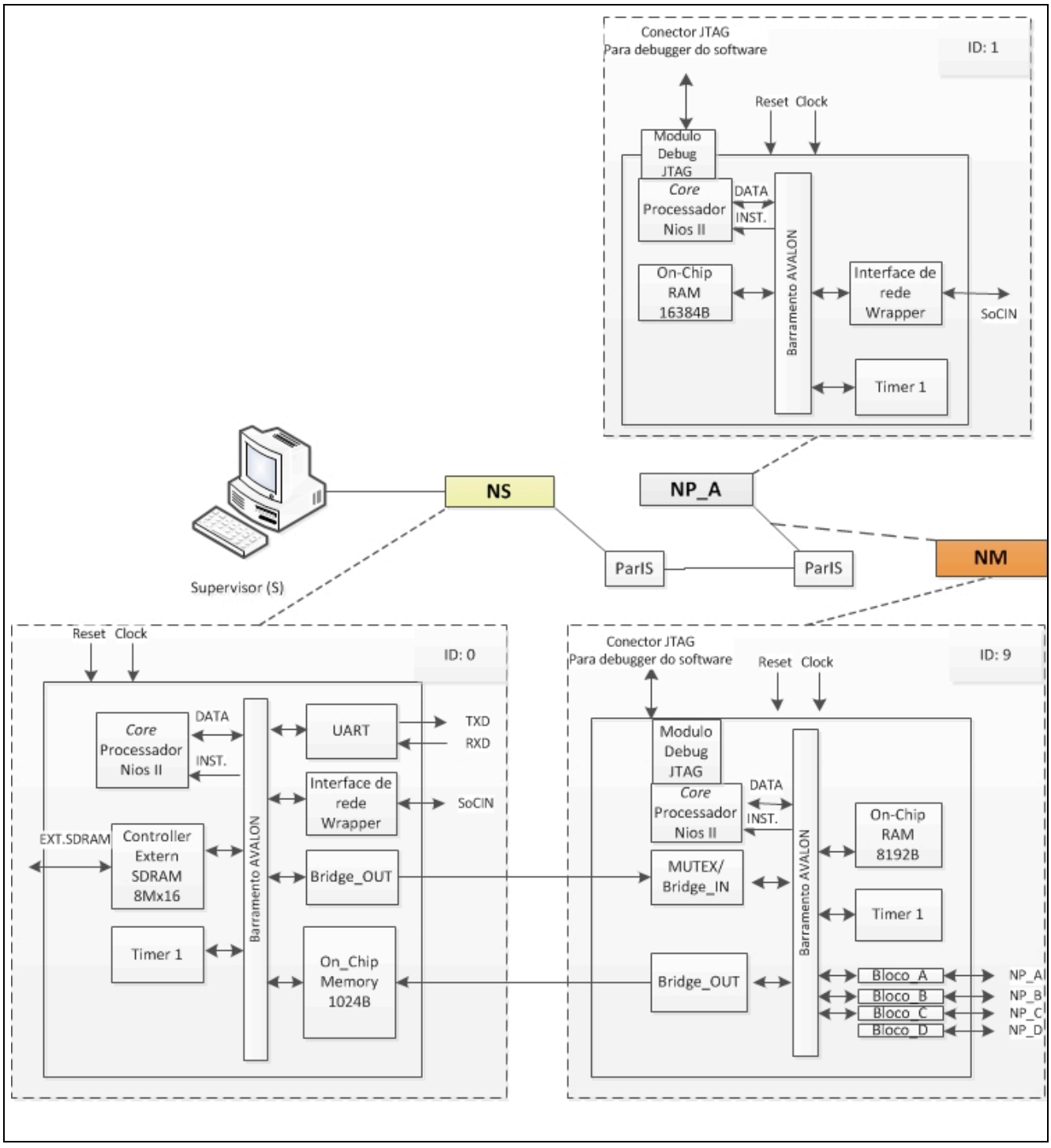


Figura 43. Arquitetura interna do sistema proposto para o NS, NP e NM

- Interface de comunicação UART (IP Altera): interface de comunicação entre o Supervisor e NS;
- Interface de rede: oferece os serviços de adaptação entre o barramento Avalon e a rede SoCIN, enquadramento do pacote, injeção e extração de pacotes da rede;
- Temporizador – Timer (IP Altera): responsável pela temporização do sistema;
- On-chip-RAM (IP Altera): utilizada para armazenar os dados do programa executado nos NPs e no NM. O NS possui uma memória compartilhada com o NM;
- Controller Extern SDRAM (IP Altera): utilizado no NS para controlar o acesso e a comunicação do sistema com barramento Avalon à SDRAM externa;
- Mutex (IP Altera): inserido no NM para gerenciar o acesso à memória compartilhada;
- Nios II (IP Altera): processador de propósito geral que executa as funcionalidades dos nodos (NS, NP e NM). Considerando o NS, o software do processador Nios II é responsável pela extração dos dados do arquivo de configuração *.tcf,* envio do descritor de fluxo para cada nodo de processamento da rede como também o controle do experimento (configuração, habilitação e finalização). Já para o NP, o software do Nios II é responsável por executar o experimento no sistema gerando tráfego conforme o descritor de fluxo recebido, injetando e extraindo pacotes na rede. O software do Nios II para o NM é responsável pelo processo de extração dos pacotes da rede para cálculo das métricas de desempenho;
- Bloco de aquisição de Medidas: inserido no nodo de monitoramento (NM) com a função de capturar os pacotes injetados e recebidos no nodo correspondente. Os dados capturados são armazenados primeiramente na FIFO local e encaminhados à memória compartilhada com o NS pelo processador Nios II. Na Figura 43 os blocos de aquisição de medida são identificados como: Bloco_A, Bloco_B, Bloco_C e Bloco_D; e
- Módulo JTAG – Joint Test Action Group (IP Altera): permite a conexão do console da ferramenta de desenvolvimento com a placa de desenvolvimento Mercúrio IV.

### 4.4.2 Projeto da interface de rede

A interface de rede permite a comunicação entre o núcleo e a rede SoCIN como também a integração com a biblioteca ocMPI. Os códigos fontes em linguagem C dessa biblioteca foram disponibilizados pelos seus desenvolvedores para uso neste projeto (Seção 3.1.10 ). A biblioteca contém arquivos com a API (Application Program Interface), funções da camada de transporte e o *device driver* de comunicação. O *device driver* realiza o empacotamento e desempacotamento de dados transferidos por meio da interface de rede e, por isso, precisa ser readequado para ser compatibilizados com o protocolo de comunicação da SoCIN.

Para o envio de uma mensagem, o *device driver* deve construir o pacote incluindo o seu cabeçalho e os dados da mensagem. O pacote então deve ser escrito na interface de rede, a qual deve acrescentar os bits de enquadramento bop (*begin-of-packet*) e eop (*end-of-packet*) que sinalizam o início e o final do pacote. A Figura 44 ilustra a estrutura do pacote gerado nesse processo.

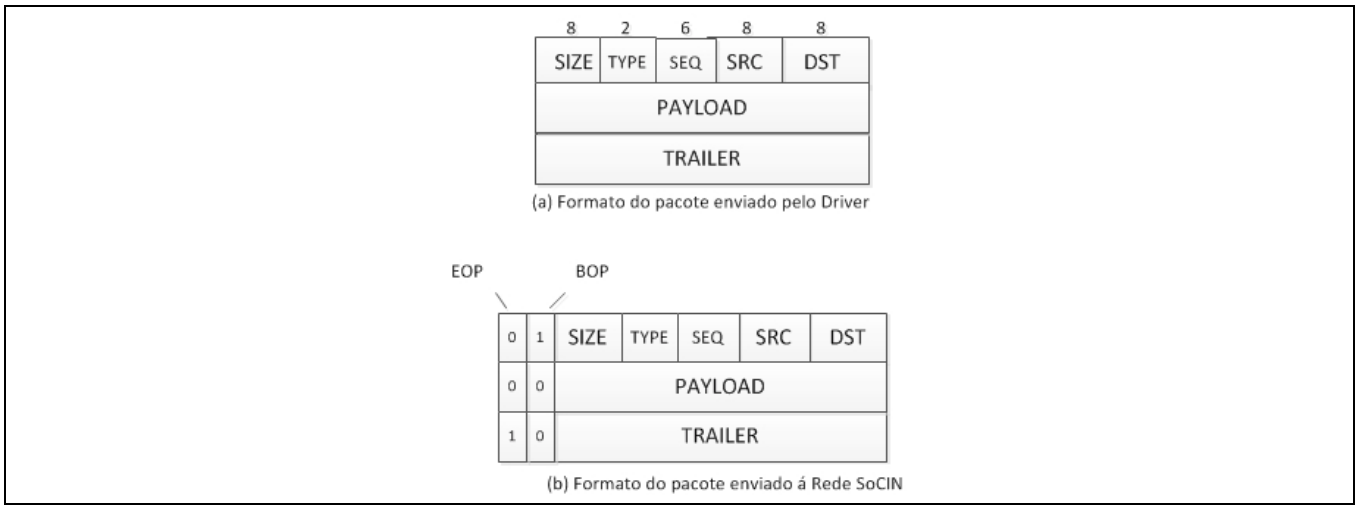


Figura 44. Formato do pacote enviado: (a) pelo *device Driver*; e (b) á rede SoCIN

O cabeçalho é composto de cinco campos. Os campos SRC (*source*) e DST (*dest*), de 8 bits cada, informam as coordenadas dos nodos origem e destino, respectivamente. O campo TYPE, de 2 bits, define o tipo de pacote conforme segue. Um pacote isolado, que não integra uma mensagem maior, é identificado por TYPE igual a 11. Na plataforma proposta, um pacote tem um tamanho máximo que define a profundidade do *buffer* de saída da interface de rede. No caso do envio de uma mensagem maior que um pacote, a mensagem pode ser fragmentada em vários pacotes. Estes pacotes podem ter o campo TYPE igual a 01 (primeiro pacote da mensagem), 10 (último pacote da

mensagem) ou 00 (pacote do meio da mensagem). O campo SEQ (Sequence Number), de 6 bits, identifica a ordem do pacote na mensagem e o campo SIZE, de 8 bits, define o tamanho do pacote em Bytes.

A estrutura do cabeçalho permite que sejam endereçados até 256 nodos diferentes, pois as coordenadas utilizam 4 bits para a abcissa (X) e 4 bits para a ordenada (Y). Também permite que uma mensagem seja quebrada em até 64 pacotes ($2^6$) e que cada pacote possua até 256 bytes ($2^8$) ou 64 *flits*. Se necessário, as larguras dos campos SIZE e SEQ podem ser redimensionadas para permitir um maior número de pacotes por mensagem, com a redução do tamanho máximo de cada pacote. Porém, isso exige alterações no *device driver* e na interface de rede.

Destaca-se que o campo SEQ permite a implementação de um *buffer* de reordenamento na camada de transporte para lidar com a possibilidade de ocorrer entrega fora de ordem no caso de se usar algoritmos de roteamento parcialmente adaptativo. No entanto, essa funcionalidade não é considerada neste trabalho, pois o foco está na medição do desempenho da rede e não na aplicação.

#### 4.4.2.1 Arquitetura da Interface de Rede

A arquitetura da interface de rede é composta de duas camadas: específica e rede, conforme apresentado na Figura 45a. A camada específica realiza a comunicação direta com o núcleo de processamento por meio do barramento Avalon. Já a camada de rede realiza o armazenamento temporário de pacotes em FIFOs como também a inserção e extração dos bits de enquadramento. A estrutura interna da interface de rede é divida em dois blocos, conforme apresentado na Figura 45b: transmissão (TX) e recepção (RX).

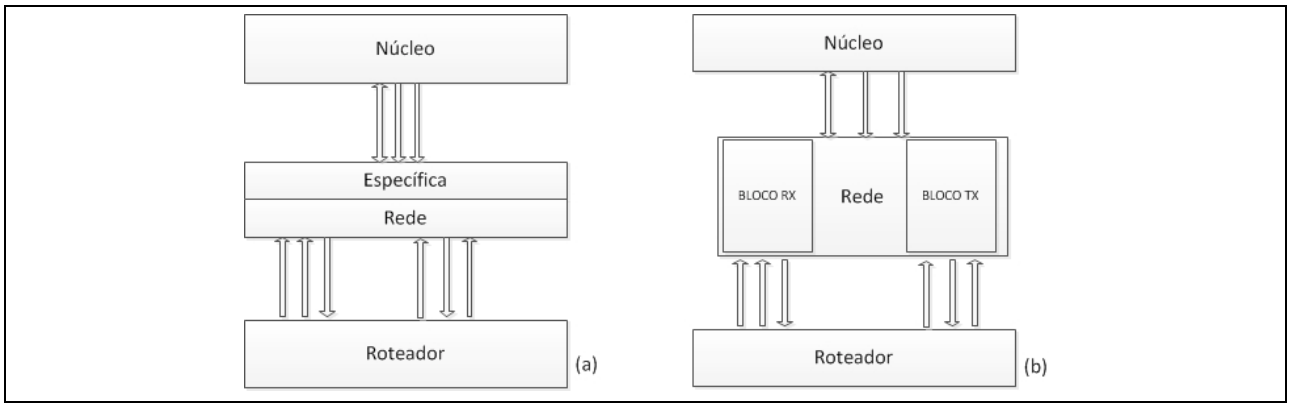


Figura 45. Interface de rede: (a) organização em camadas; e (b) bloco TX e RX

O bloco TX é responsável pelo tratamento das informações recebidas do núcleo para encaminhamento à rede. Já o bloco RX é responsável pelo tratamento das informações recebidas da rede SoCIN para encaminhamento ao núcleo. Cada bloco implementa uma parte das funcionalidades das duas camadas que integram a interface de rede.

A interface de rede é configurada pelo conjunto de parâmetros apresentados na Figura 46, os quais também são parâmetros de configuração da SoCIN: (*i*) profundidade das FIFOs (`c_LOG2_DEPTH`, `c_DEPTH`); (*ii*) técnica de controle de fluxo em nível de enlace (`c_FC_TYPE`, `c_CREDIT`); (*iii*) largura do campo de endereço do cabeçalho (`c_ADDR_WIDTH`); e (*iv*) largura do canal de dados (`c_DATA_WIDTH`).

```
Generic (
c_FIFO_TYPE           : string  := "RING"; options: NONE, SHIFT, RING and ALTERA
c_LOG2_DEPTH          : integer := 3;
c_DEPTH               : integer := 8;
c_FC_TYPE             : string  := "CREDIT";  options: CREDIT or HANDSHAKE
c_CREDIT              : integer := 4;   maximum number of credits
c_DATA_WIDTH          : natural := 32;
c_ADDR_WIDTH          : natural := 8
;
```

Figura 46. Parâmetros ajustáveis da interface de rede

#### 4.4.2.2 Projeto do bloco de transmissão

O bloco TX é responsável por realizar o armazenamento e o enquadramento do pacote a ser injetado na rede SoCIN. A sua estrutura interna, conforme apresentado na Figura 47, é constituída por: (*i*) lógica de adaptação; (*ii*) registrador Status; (*iii*) FIFO; (*iv*) controle de transmissão; (*v*) bloco de inserção dos bits de enquadramento; e (*vi*) controle do fluxo de saída. Esses blocos são detalhados logo a seguir.

O processo de transmissão do núcleo para a interface de rede ocorre da seguinte forma: o núcleo realiza, primeiramente, um processo de leitura no registrador de estado da interface de rede de forma a verificar se está pronta para receber e encaminhar um pacote. Se estiver, o *device driver* inicia o processo de escrita do pacote na FIFO. Após finalizar o processo de escrita na FIFO, ele envia um comando para iniciar a transmissão para a rede SoCIN. Neste momento, são inseridos bits de enquadramento do pacote nos *flits*, na medida em que estes são injetados na rede.

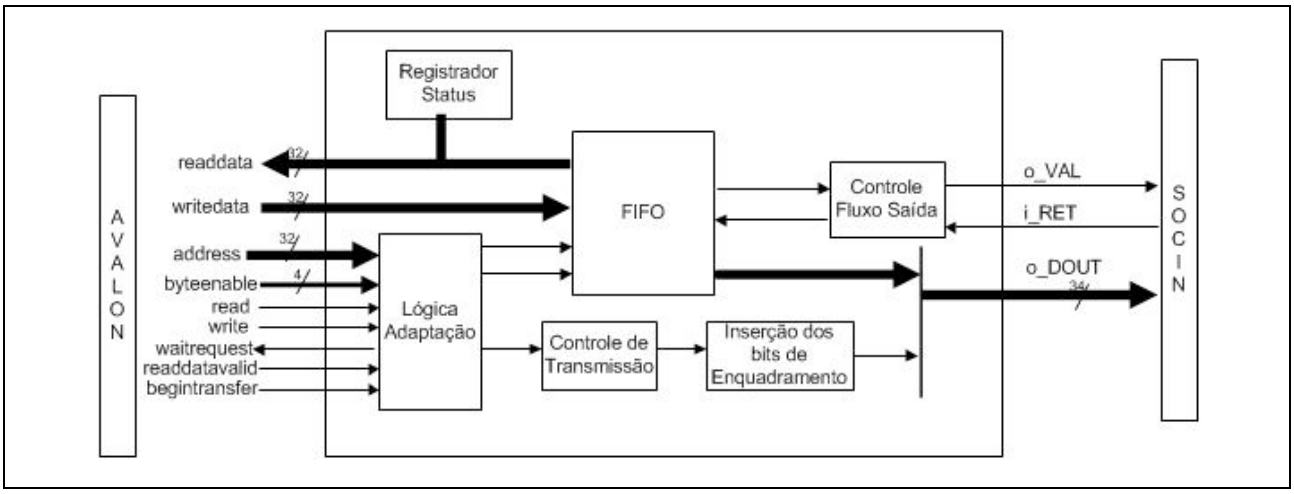


Figura 47. Diagrama de blocos da interface rede: bloco de transmissão

Na interface de rede, a lógica de adaptação é responsável por liberar o acesso dos processos de leitura e escrita do núcleo na interface de rede por meio de lógica combinacional, enquanto o registrador Status é responsável por informar ao núcleo se a interface está liberada para receber um novo pacote. O bloco FIFO é utilizado para o armazenamento temporário do pacote enviado pelo núcleo a rede.

O bloco de controle de transmissão é responsável por encaminhar ao registrador Status a informação do estado da interface de rede (liberada para receber um novo pacote) e também por liberar a leitura na FIFO.

O comando do *device driver* para envio dos pacotes armazenados na FIFO à rede habilita o bloco de inserção de bits de enquadramento. Este bloco é constituído por um contador e uma máquina de estados. O contador é incrementado cada escrita na FIFO, registrando o tamanho do pacote a ser enviado. Essa informação é necessária para determinar o momento de inserir os bits de sinalização de final do pacote. Já o processo de leitura na FIFO habilita o contador para contagem descrecente. O contador é carregado com o tamanho do pacote, contido no cabeçalho. Quando o contador chega ao valor igual a um, será repassado um sinal à máquina de estado para indicar que um pacote foi recebido.

A máquina de estados é responsável por injetar na rede um pacote previamente armazenado no FIFO. Ela é formada por quatro estados, conforme é ilustrado na Figura 48: s_IDLE, s_SEND_HEADER, s_SEND_PAYLOAD e s_SEND_TRAILER. A máquina inicia no estado S_IDLE e aguarda até que i_TX_START apresente nível lógico 1. Esse sinal é gerado pela lógica

de adaptação a partir do recebimento de um comando emitido pelo *device driver*. Quando isso ocorre, a máquina muda de estado para s_SENDER_HEADER, no qual ela envia o cabeçalho, o que é confirmado pelo sinal i_RD em 1. Se o pacote contém apenas dois *flits* a máquina muda de estado para s_TRAILER. No estado s_PAYLOAD, são enviados os demais *flits* do cabeçalho, com exceção do último *flit* que é enviado no estado s_TRAILER. Após o envio do pacote, a máquina volta para o estado s_IDLE.

Já a unidade de controle de fluxo utiliza os sinais o_VAL e i_RET para regular o fluxo de envio de cada *flit* para a rede, conforme a técnica selecionada (Handhake ou Credit-based).

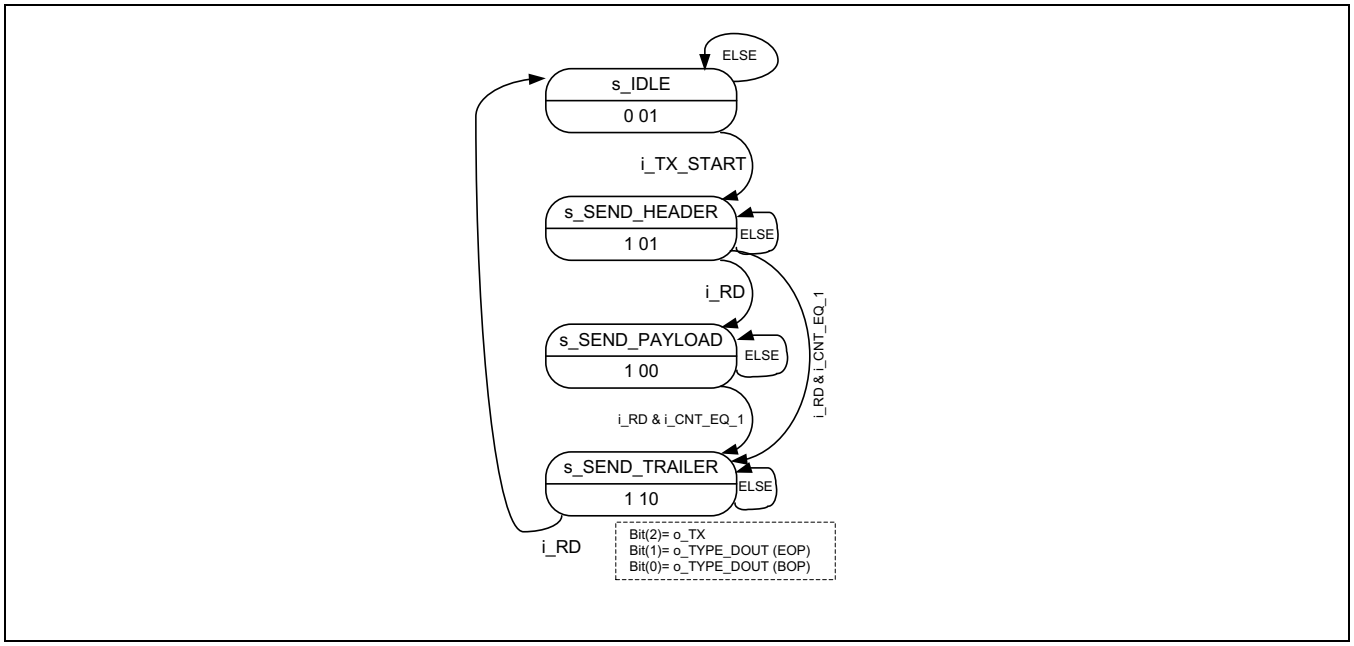


Figura 48. Máquina de estados do bloco TX

#### 4.4.2.3 Projeto do bloco de recepção

O bloco de recepção (RX) é responsável por realizar o armazenamento e a extração da informação do tamanho do pacote enviado pela rede SoCIN. A estrutura interna do bloco de recepção, conforme apresentado na Figura 49, é constituída por: (*i*) registrador Status; (*ii*) lógica de adaptação; (*iii*) FIFO; (i*v*) controle fluxo de entrada; e (*v*) controle de recepção.

O processo de recepção dos dados pelo núcleo ocorre da seguinte forma: o *device driver* realiza, primeiramente, um processo de leitura no registrador de Status da interface de rede para verificar se há dados para serem lidos. A interface de rede, por sua vez, só irá liberar a leitura na

FIFO após receber um pacote inteiro. O primeiro *flit* recebido, cabeçalho, contém as informações de destino, origem, tipo, sequência e tamanho do pacote. A informação do tamanho do pacote será utilizada para controlar a quantidade de *flits* a serem recebidos e liberar a leitura da FIFO pelo *device driver*. O *device driver* só irá liberar a interface de rede para receber um novo pacote, apenas após a recepção completa do pacote (esta restrição pode ser removida se o *buffer* tiver capacidade para armazenar mais pacotes).

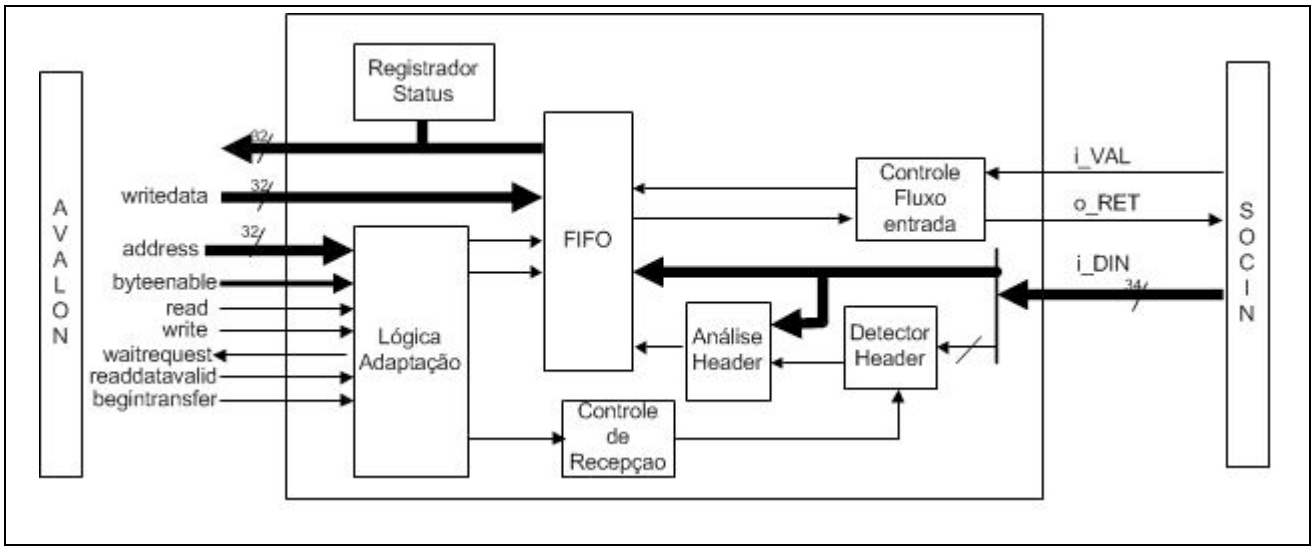


Figura 49. Diagrama de blocos da interface de rede: bloco de recepção

A lógica de adaptação é responsável por liberar o acesso dos processos de leitura e escrita do núcleo na interface de rede por meio de lógica combinacional. O registrador de Status da recepção é responsável por informar ao núcleo se há pacote pronto para ser lido. O bloco controle de recepção libera o armazenamento de um novo pacote na FIFO. Enquanto não for extraido o pacote da FIFO pelo núcleo, não é liberado um novo processo de escrita. O bloco detector header é responsável por detectar e extrair do cabecalho as informações da quantidade de *flits* do pacote. Essas informações são enviadas ao bloco de análise header. O bloco de análise header é formado por dois componentes: contador e comparador. O contador realiza a contagem dos *flits* recebidos. Já o comparador é responsável por comparar a informação extraída do cabeçalho referente ao tamanho do pacote com o valor do contador. Quando essas duas entradas forem iguais, significa que o pacote foi recebido por completo. O bloco FIFO é utilizado para o armazenamento temporário do pacote recebido pela rede, enquanto a unidade de controle de fluxo de entrada utiliza os sinais i_VAL e o_RET para regular o fluxo dos *flits* que chegam da rede.

### 4.4.3 Método de comunicação ocMPI

A interface de comunicação entre os núcleos é realizada pelo padrão MPI (Message Passing Interface), porém em uma versão otimizada chamada ocMPI desenvolvida por Fernandez-Alonso et al. (2012) (Seção 3.1.10 ). Essa versão permite a inserção da biblioteca ocMPI no ambiente de desenvolvimento Quartus II possibilitando a utilização deste padrão de comunicação. Os desenvolvedores disponibilizaram o código fonte da biblioteca ocMPI, camada de transporte e *device driver* para realizar a comunicação com a interface de rede por mapeamento de memória. A versão ocMPI possui onze funções de comunicação, além de declarações de constantes e variáveis. A Figura 50 apresenta o fluxo da função ocMPI_Send desde a camada de aplicação até o envio para NoC.

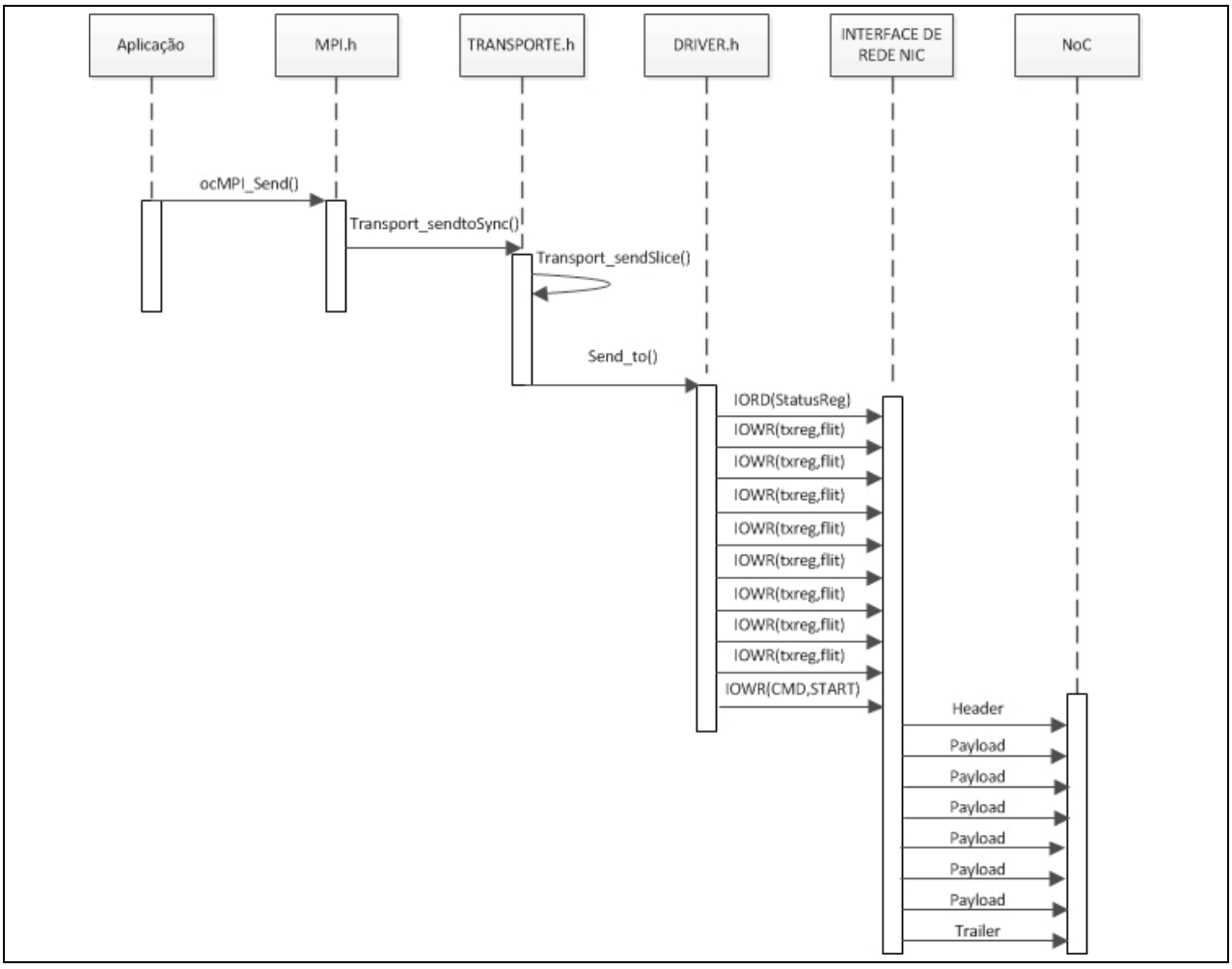


Figura 50. Diagrama da primitiva ocMPI_Send

A camada de Aplicação repassa para a primitiva ocMPI_Send os atributos referentes ao *buffer* e informações para o empacotamento. Essas informações são enviadas a camada de

Transporte que cria primeiramente um *buffer* com todos os dados a serem enviados e, posteriormente, realiza o processo de encapsulamento dos dados. Os dados encapsulados são armazenados em um novo *buffer* e encaminhados ao Driver.

O Driver realizará um processo de leitura no registrador de estado da interface para verificar se poderá enviar um novo pacote. A interface de rede armazenará o pacote recebido na sua FIFO. O pacote só será escrito na NoC após o Driver liberar a transmissão. O exemplo da Figura 50 ilustra a escrita de um pacote com oito *flits*, incluindo um *flit* de cabeçalho e sete *flits* de dados.

O estabelecimento da comunicação ocMPI ponto-a-ponto é realizado em três etapas: (*i*) inicialização do ambiente do ambiente ocMPI (ocMPI_init, ocMPI_comm_size, oCMPI_comm_rank); (*ii*) comunicação entre os processadores (ocMPI_Send, ocMPI_Recv); e (*iii*) finalização do ambiente ocMPI (ocMPI_Finalize).

O processo de inicialização ocMPI_Init (`int *argc, char ***argv`) verifica primeiramente se as primitivas ocMPI_Finalize e ocMPI_Init foram chamadas anteriormente. Após esta verificação, duas mensagens são enviadas a cada processador, uma com a informação de seu identificador e outra com a informação da quantidade de processadores que constituem a rede. A primitiva ocMPI_Comm_Rank (`ocMPI_Comm comm, int *rank`) determina o identificador para a comunicação do processo e a primitiva ocMPI_Comm_Size (`ocMPI_Comm comm, int *size`) determina a quantidade existente de processadores na comunicação. Estas duas funções fazem parte do processo de inicialização.

A comunicação entre os processadores ocorre por meio das primitivas ocMPI_Send (`void *buf, int count, ocMPI_Datatype datatype, int dest, int tag, ocMPI_Comm comm`) e ocMPI_Recv (`void *buf, int count, ocMPI_Datatype datatype, int source, int tag, ocMPI_Comm comm, ocMPI_Status *status`).

As primitivas ocMPI_Send e ocMPI_Recv enviam e recebem as mensagens utilizando bloqueantes. Ou seja, o processo só retorna à execução do programa após a mensagem ter sido completamente enviada ou recebida, liberando assim o *buffer* para ser reutilizado pela aplicação.

Na primitiva ocMPI_Send (`void *buf, int count, ocMPI_Datatype datatype, int dest, int tag, ocMPI_Comm comm`), os atributos `buf` (endereço dos dados a serem enviados), `count` (número de elementos a serem enviados) e `datatype` (tipo dos dados) são todas informações

referentes ao *buffer*. Os outros três atributos `dest` (destino do pacote), `tag` (identificação da mensagem) e `Comm` (identificador da comunicação) são informações utilizadas para o empacotamento da mensagem. A Figura 51, ilustra um pacote com oito *flits*, formado por um *flit* de cabeçalho, 5 *flits* para a configuração da comunicação MPI e 2 *flits* de dados.

| B3 | B2 | B1 | B0 | |
|---|---|---|---|---|
| SIZE | TYPE | SRC | DST | 1 *flit* - Cabeçalho |
| OcMPI Global Rank | | | | 2 *flit* – Configuração comunicação ocMPI |
| Destino | | | | 3 *flit* – Configuração comunicação ocMPI |
| Tag | | | | 4 *flit* – Configuração comunicação ocMPI |
| DataType | | | | 5 *flit* – Configuração comunicação ocMPI |
| Count | | | | 6 *flit* – Configuração comunicação ocMPI |
| Payload | | | | 7 *flit* – Dados |
| Payload | | | | 8 *flit* – Dados |

Figura 51. Cabeçalho e *flits* de configuração do ocMPI

Na primitiva ocMPI_Recv (`void *buf, int count, ocMPI_Datatype datatype, int source, int tag, ocMPI_Comm comm, ocMPI_Status *status`), os atributos `buf` (endereço do dado a ser recebido), `count` (número de elementos a serem recebidos) e `datatype` (tipo dos dados) são informações referentes ao *buffer*. Os outros quatros atributos `source` (origem do pacote), `tag` (identificador da mensagem), `Comm` (identificador da comunicação) e `status` (verifica se as informações recebidas referentes a `source` e `tag` são validas) são para o processo de encapsulamento.

A primitiva ocMPI_Finalize é chamada após o término da comunicação, ou seja, garantindo que o estabelecimento da comunicação ocorreu com sucesso. Já a primitiva ocMPI_Wtime, é utilizada para obter o tempo da chamada de um processo.

### 4.4.4 Projeto do bloco de aquisição

O bloco de aquisição de medidas é um componente do nodo de monitoramento. Cada bloco de aquisição inserido no sistema está conectado fisicamente à porta local de um roteador ParIS. Este bloco é responsável por realizar a extração e o armazenamento do cabeçalho do pacote com seu respectivo tempo de injeção ou extração na rede SoCIN conforme a sua localição (canal de entrada

ou canal de saída da porta do roteador). O bloco de aquisição é dividido em dois sub-blocos: (a) canal de entrada, captura dados de pacotes injetados na rede através do canal de entrada da porta Local to reotador; e (b) canal de saída, captura dados de pacotes ejetados da rede através do canal de saída da porta Local to reotador

O sub-bloco canal de entrada, apresentado na Figura 52, é responsável por realizar o armazenamento das informações do pacote injetado, ou seja, o cabeçalho e o tempo de injeção na rede SoCIN. A estrutura interna desse bloco, conforme apresentado na Figura 52, é constituída de: (*i*) detecção do cabeçalho (ENA_Header); (*ii*) controle de entrada (CTRL_Input); (*iii*) FIFOs (FIFO_Header, FIFO_Timer_Low, FIFO_Timer_High); (*iv*) registrador Status (Status_FIFO); (*v*) Timer; e (*vi*) controle do fluxo de entrada (IFC). Esses blocos são discutidos logo a seguir.

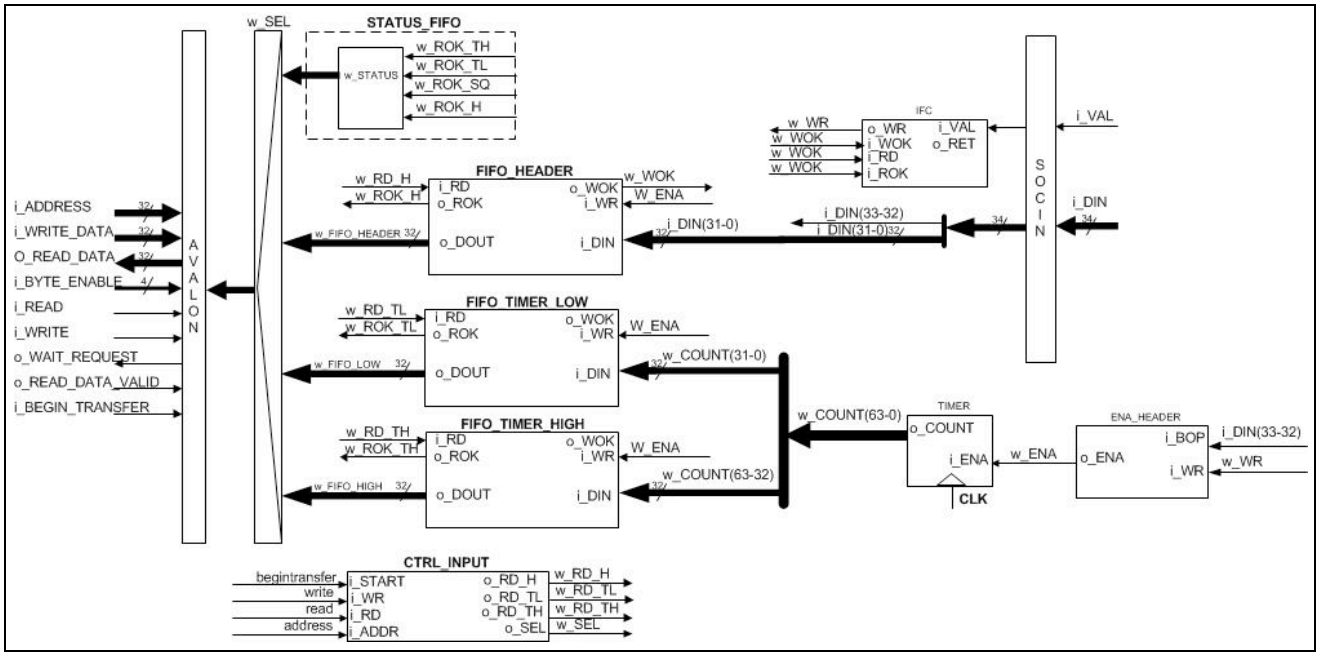


Figura 52. Diagrama de blocos do componente captura_transmissão na rede SoCIN

O processo de aquisição dos dados transmitidos pelo núcleo para a rede SoCIN ocorre da seguinte forma. O nodo de monitoramento realiza, primeiramente, um processo de leitura no registrador de Status (Status_FIFO) de forma a verificar se há dados para serem lidos. Se houver, o bloco de controle entrada (CTRL_Input) irá liberar o processo de leitura do núcleo monitoramento nas FIFOs do bloco de aquisição por meio de lógica combinacional. O bloco detecção de cabeçalho (ENA_Header) é responsável por detectar quando um pacote é inserido na rede SoCIN por meio dos bits de enquadramento de pacote (BOP). Após a detecção do pacote, seu cabeçalho é registrado, bem como o tempo (obtido pelo Timer) em que foi feita a detecção, o qual corresponde ao tempo

inicial da transmissão desse pacote. Essas informações são armazenadas em três FIFOs: o cabeçalho é armazenado na FIFO_Header e o tempo na FIFO_Timer_Low e na FIFO_Timer_High. A unidade de controle de fluxo de entrada utiliza os sinais i_VAL e o_RET para regular o fluxo dos *flits* que chegam da rede.

O sub-bloco canal de saída, apresentado na Figura 53, é responsável por realizar o armazenamento das informações relativas aos pacote entregues pela rede, ou seja, o cabeçalho e o tempo de ejeção completa do pacote da rede SoCIN, dado pelo ciclo em que o terminador do pacote é entregue ao destinatário. A diferença entre esse tempo e o tempo inicial de transmissão define a latência da comunicação. A estrutura interna deste sub-bloco, conforme apresentado na Figura 53, é constituída de: (*i*) detecção do cabeçalho (ENA_Header_Out); (*ii*) controle de entrada (CTRL_Out); (*iii*) FIFOs (FIFO_Header_Out, FIFO_Timer_Low_Out, FIFO_Timer_High_Out); (*iv*) registrador Status (Status_FIFO_Out); (*v*) Timer_Out; e (*vi*) controle do fluxo de entrada (IFC). Esses blocos são discutidos logo a seguir.

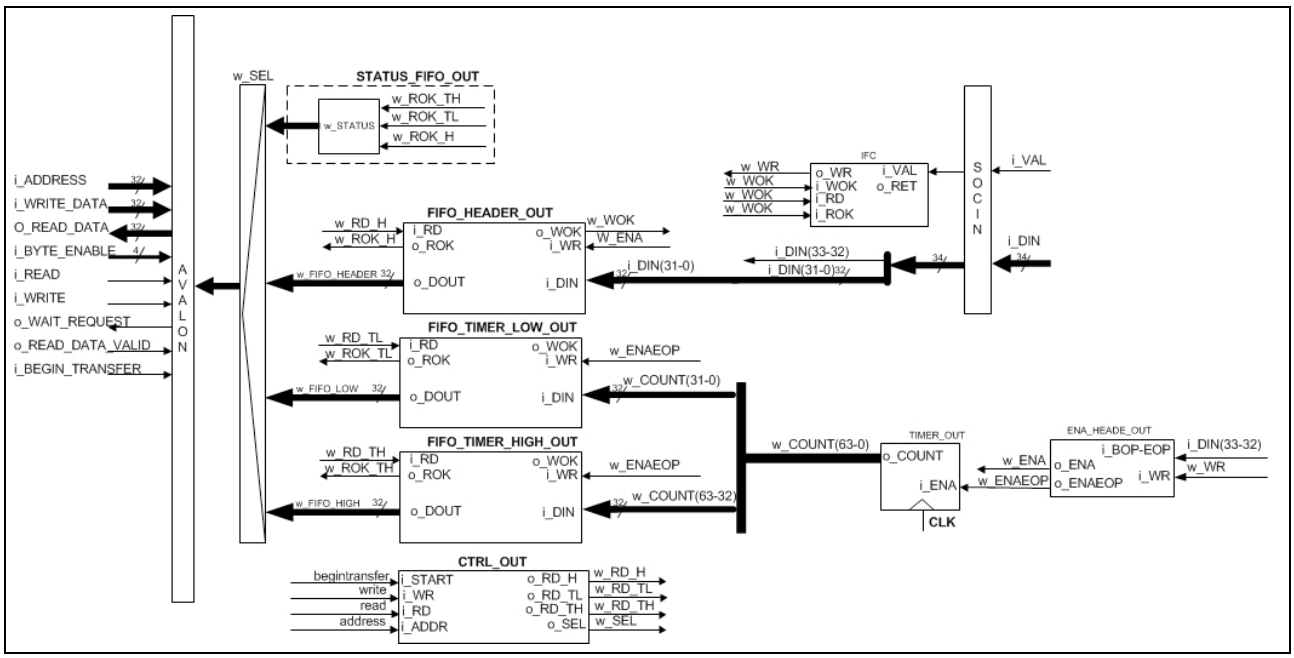


Figura 53. Diagrama de blocos do componente captura_recepção na rede SoCIN

O processo de aquisição dos dados recebidos pela rede para o núcleo ocorre da seguinte forma. O núcleo de monitoramento realiza, primeiramente, um processo de leitura no registrador de Status (Status_FIFO_Out) de forma a verificar se há dados para serem lidos. Se houver, o bloco de controle entrada (CTRL_Out) irá liberar o processo de leitura do núcleo de monitoramento nas FIFOs do bloco de aquisição por meio de lógica combinacional. O bloco detecção de cabeçalho

(ENA_Header_Out) é responsável por detectar o ínicio e, também, o término do pacote recebido na rede SoCIN por meio dos bits de enquadramento de pacote (BOP e EOP). As informações do cabeçalho são registrado quando este é detectado. Após, quando o terminador do pacote é recebido, é feito o registro do tempo corrente do Timer. Essas informações são armazenadas em três FIFOs: o cabeçalho é armazenado na FIFO_Header e o tempo nas FIFO_Timer_Low e FIFO_Timer_High. A unidade de controle de fluxo de entrada utiliza os sinais i_VAL e o_RET para regular o fluxo dos *flits* que chegam da rede.

# 5 RESULTADOS EXPERIMENTAIS

Este capítulo apresenta as etapas de verificação como também os cenários de teste e os resultados obtidos por simulação e emulação na plataforma proposta. A fase de verificação do sistema proposto foi realizada em quatro etapas:

- Primeira Etapa – Implementação e verificação da comunicação com a interface de rede: Nesta etapa, foi realizado o desenvolvido em VHDL dos módulos TX e RX da interface de rede em VHDL e a verificação por meio de simulação utilizando a ferramenta ModelSim. Nas simulações nesta etapa e nas demais, foi utilizado um sinal de relógio com uma frequência de 100 MHz (ou seja, com período de 10 ns);

- Segunda Etapa – Integração da interface de rede a rede SoCIN: Os módulos TX e RX, bem como a SoCIN, foram integrados à plataforma. A construção dos arquivos *testbenches* para o ModelSim permitiu a implementação das aplicações para teste da plataforma para verificação e medição de desempenho;

- Terceira Etapa – Adaptação do *device driver*: Consistiu em inserir a biblioteca ocMPI no projeto de software do ambiente de desenvolvimento Nios II Software Build Tools for Eclipse e adaptar o *device driver* ocMPI para o protocolo da SoCIN; e

- Quarta Etapa – Construção e integração da plataforma proposta: Utilizando a ferramenta Qsys, foi possível modelar a plataforma proposta com os núcleos: nodo supervisor, nodo monitor e nodos de processamentos. A integração do sistema foi realizada em parte até chegar à plataforma proposta completa, ou seja, supervisor, nodo supervisor, nodo processamento e nodo de monitoramento. Com a plataforma modelada e integrada, validamos o funcionamento da plataforma.

Os resultados das etapas citadas anteriormente são apresentados ns subseções que seguem.

## 5.1 VERIFICAÇÃO

### 5.1.1 Comunicação com a interface de rede

A primeira etapa consistiu na implementação e verificação da comunicação entre o NP e a interface de rede, conforme apresentado na Figura 54. Este sistema foi formado por núcleo de processamento e duas inteface de rede (blocos internos de transmissão e recepção interligados, ou seja, LoopBack). O núcleo é responsável por enviar o pacote de dados pela interface de rede (NIC_A) e receber o mesmo pacote de dados por outra interface de rede (NIC_B).

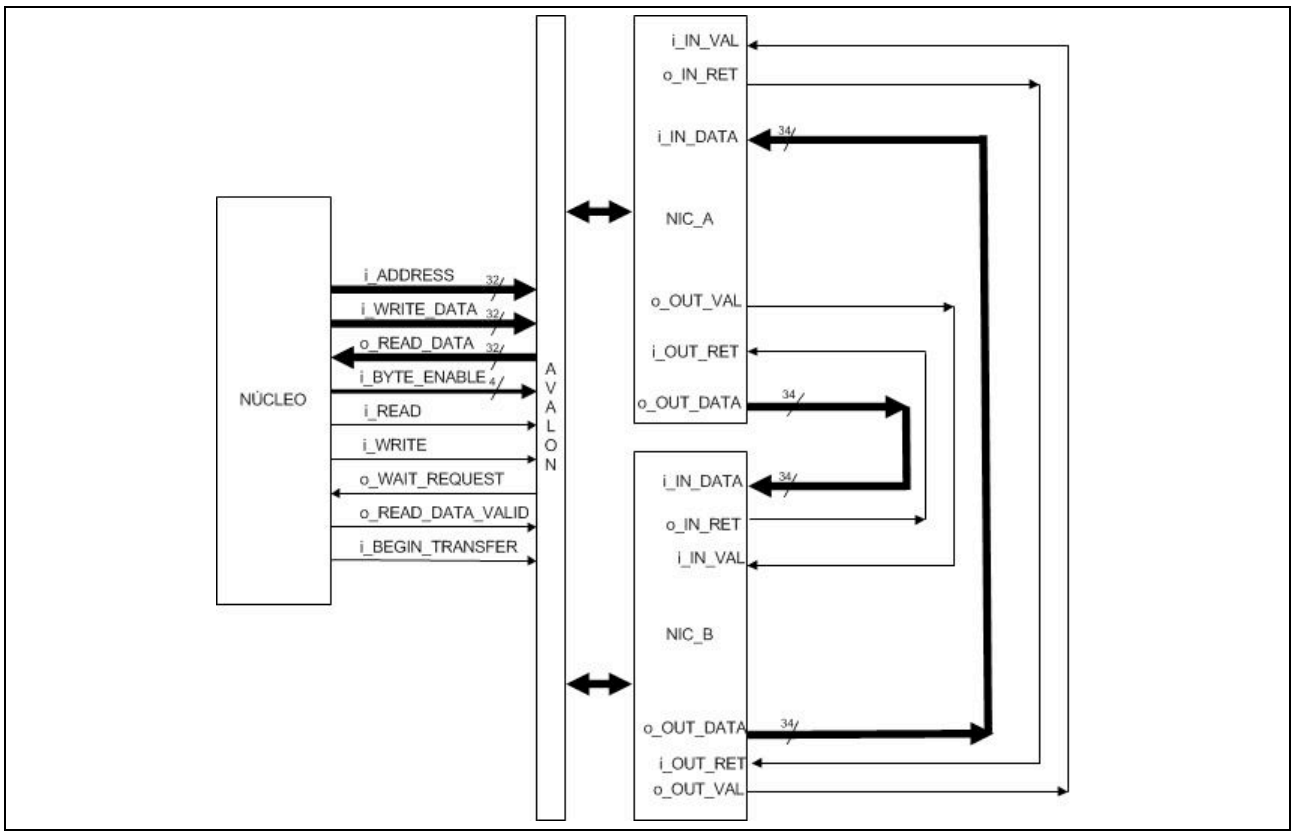


Figura 54. Sistema para validação da comunicação entre núcleo e interface de rede

#### 5.1.1.1 Leitura do registrador Status

Antes de transmitir um pacote, o processador verifica o registrador de Status do bloco de transmissão da interface de rede (NIC_A). Esse registrador informa se a interface está disponível para transmitir um pacote. A Figura 55 apresenta o diagrama de forma de onda de simulação do processo de leitura no registrador Status, destacado pelo pela moldura retangular. Observa-se que o

processo de leitura ocorre no primeiro ciclo de relógio(*clk*), no qual os sinais *read* e *begintransfer* são ativados pelo mestre do barramento Avalon em conjunto com o *address* (0x01). No segundo ciclo de relógio, o registrador disponibiliza a informação referente ao estado da FIFO transmissão (terceiro bit menos significativo) para o sinal *readdata* (0x00000000). No exemplo, o estado da FIFO é igual à zero, indicando que está pronta para receber um novo pacote do núcleo.

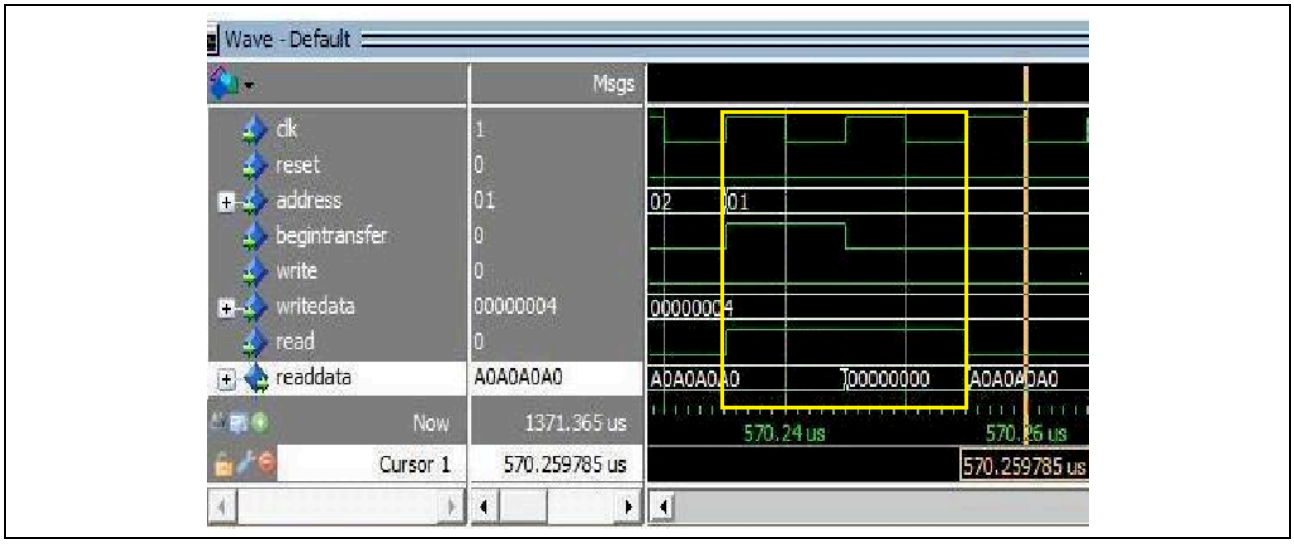


Figura 55. Processo de leitura do registrador Status da interface de rede

### 5.1.1.2 Escrita na FIFO

A transmissão do pacote é iniciada após o bloco de transmissão da interface de rede informar ao núcleo por meio do registrador Status que está pronta para transmitir. A Figura 56 apresenta os diagramas de formas de onda de simulação para o processo de escrita na FIFO transmissão da interface de rede, destacado por molduras. O processo de escrita ocorre no primeiro ciclo de relógio, marcador A. Quando os sinais *write* e *begintransfer* são ativados pelo mestre do barramento Avalon, em conjunto com o *address* (0x00), a informação presente no sinal *writedata* é armazenada na FIFO.


Figura 56. Processo de escrita na FIFO

### 5.1.1.3 Escrita na unidade de controle e envio à rede SoCIN

Após o armazenamento do pacote no bloco de transmissão da interface de rede (NIC_A), o núcleo envia um comando liberando a transmitir do pacote à rede SoCIN. A Figura 57 apresenta o diagrama de forma de onda de simulação do processo de escrita na interface de rede como também o envio do pacote à rede SoCIN, destacados por molduras nas figuras. O marcador A apresenta o processo de escrita na unidade de controle onde os sinais *write* e *begintransfer* são ativados pelo

mestre do barramento Avalon em conjunto aos sinais *address* (0x02) e *writedata* (0x00000001) para a liberação do pacote a rede SoCIN.

A unidade de controle da interface de rede (NIC_A) libera a transmissão do pacote para a rede SoCIN por meio dos sinais: o_VAL, i_RET e DOUT, conforme marcador B. Observa-se que os dados na saída o_DOUT apresentam os bits de enquadramento nos *flits* do pacote. O marcador C apresenta os dados na entrada do bloco de recepção da interface de rede (NIC_B).

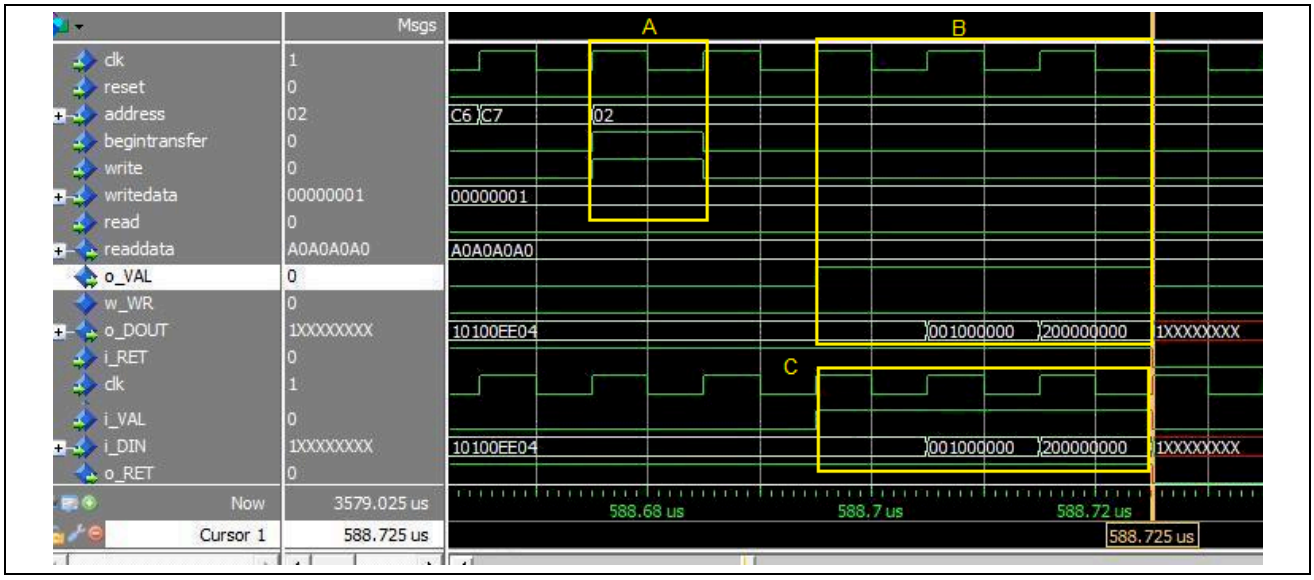


Figura 57. Processo de envio do pacote á rede SoCIN

Considerando o processo de transmissão pela interface de rede (NIC_A), desde a leitura do registrador Status até o envio à rede SoCIN, o tempo para todo esse processo foi de aproximadamente 18,47 µs (ou seja, 1847 ciclos).

#### 5.1.1.4 Processo de leitura dos dados

Para leitura dos dados, o processador verifica no registrador de Status do bloco de recepção da interface de rede (NIC_B) se existe pacote para ser lido. A Figura 58 apresenta o diagrama de forma de ondas de simulação para o processo de leitura dos dados na interface de rede, destacados pelas molduras. O marcador A apresenta o processo de leitura quando os sinais *read* e *begintransfer* são ativados pelo mestre do barramento Avalon em conjunto com *address* (0x03).

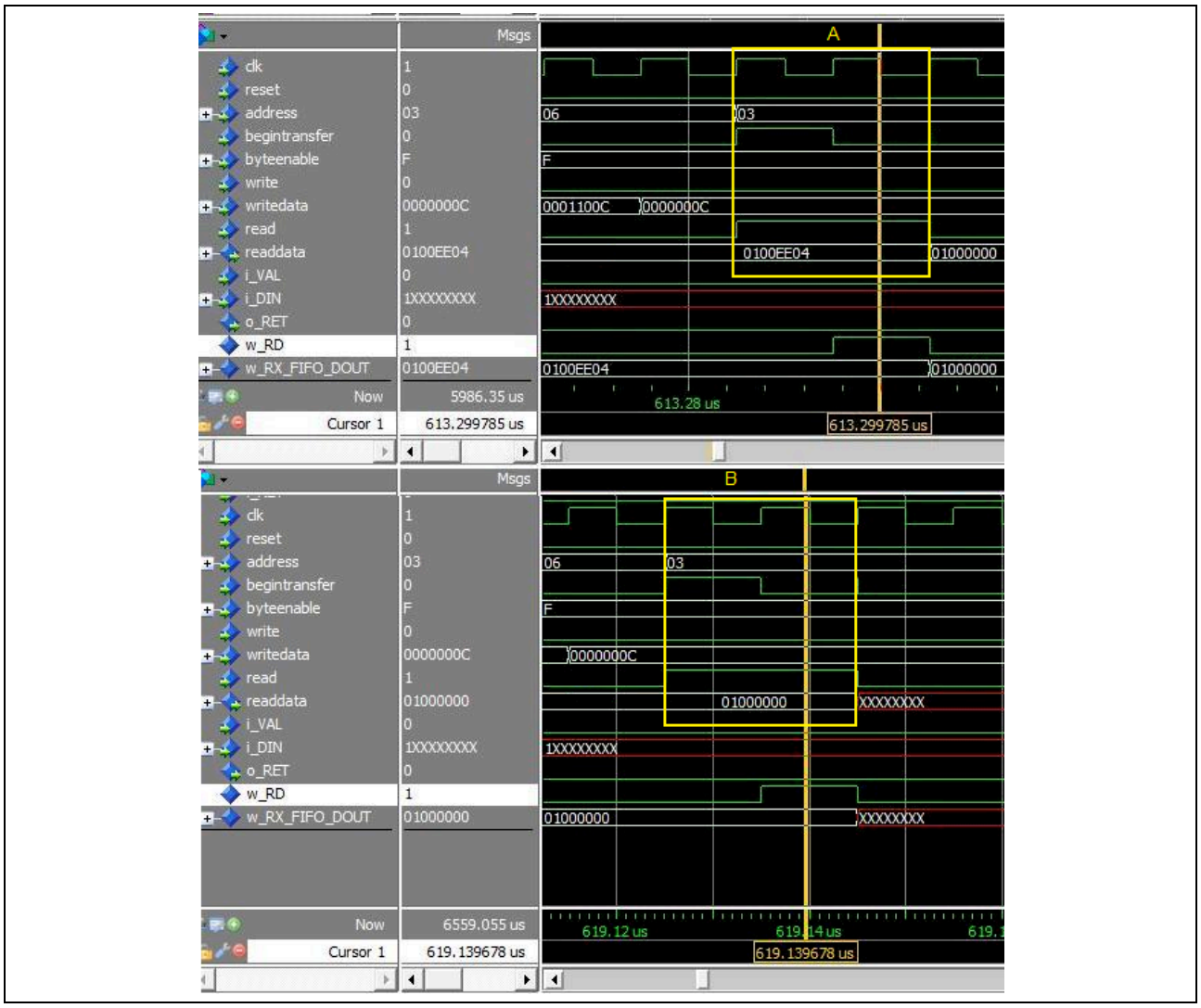


Figura 58. Processo de recepção dos dados pela interface de rede

Desta forma, a FIFO é acessada e o primeiro *flit* (0x0100EE04) do pacote sem os bits de enquadramento é destinado ao sinal *readdata*. O marcador B indica o acesso ao segundo *flit* (0x01000000) do pacote na FIFO. Considerando o processo de recepção pela interface de rede NIC_B, desde a recepção dos dados da SoCIN até a leitura da FIFO, o tempo para realizar todo esse processo foi de aproximadamente 30,45 µs (ou seja 3045 ciclos).

### 5.1.2 Integração da interface de rede a rede SoCIN

A segunda etapa consistiu na integração do núcleo de processamento com interface de rede e NoC SoCIN, conforme apresentado na Figura 59.

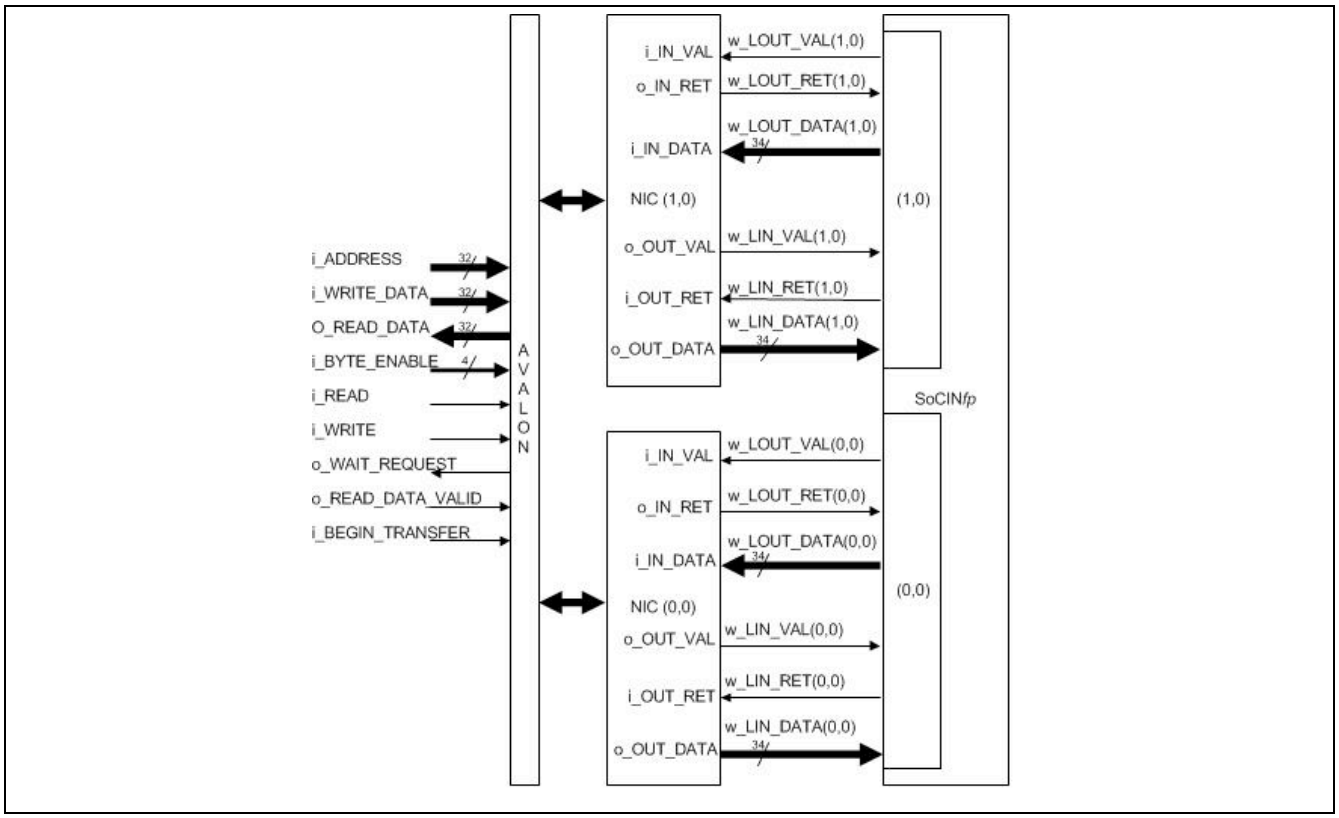


Figura 59. Sistema para validação da comunicação entre interface de rede e SoCIN

Para realizar esta etapa, foi criado um arquivo de simulação do barramento Avalon para o simulador ModelSim de tal forma que gera-se os processos de escrita e leitura na interface de rede. Isso foi necessário, pois a ferramenta Nios II Software Build for Eclipse não permitiu gerar o arquivo de simulação do ModelSim com o IP da rede SoCIN*fp* inserido no sistema. A Figura 60 apresenta o diagrama de formas de onda de simulação para o processo transmissão e recepção na rede SoCIN. Foram inseridos dois marcadores (molduras) no diagrama para indicar o início do processo de transmissão e recepção na rede SoCIN. O marcador A indica quando o sinal o_OUT_VAL é ativado pela interface de rede sinalizando o envio do pacote para a rede SoCIN. O marcador B indica quando o sinal i_IN_VAL é ativado pela rede SoCIN sinalizando o envio do pacote para a interface de rede.

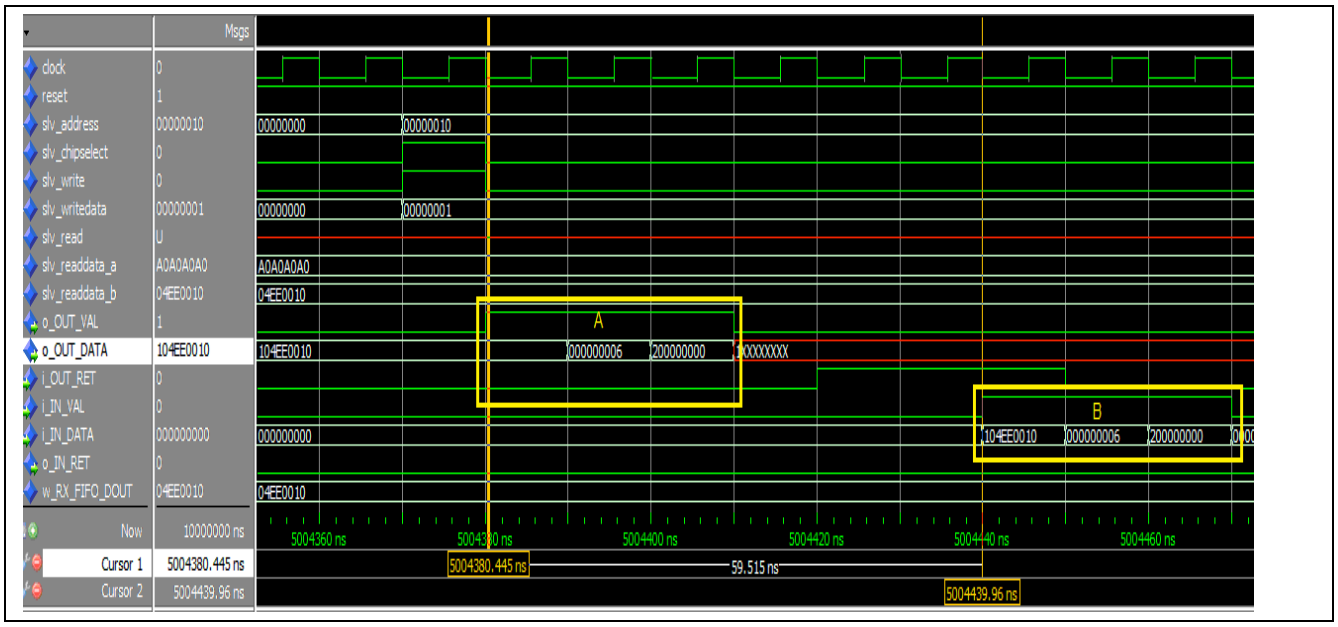


Figura 60. Processo de transmissão e recepção na rede SoCIN

Considerando que a rede SoCIN foi configurada com dois roteadores e que cada roteador consome três ciclos de relógio para encaminhar o cabeçalho de um pacote, na ausência de contenção, o tempo total gasto para este processo foi de 60 ns (ou seja, 6 ciclos). A transferência do restante do pacote implica em 10 ns (1 ciclo de relógio) para cada *flit* adicional.

### 5.1.3 Adaptação do device driver

Na terceira etapa, foi realizada a inserção e a adaptação da biblioteca ocMPI no projeto de software à rede NoC SoCIN. Para o estabelecimento da comunicação ocMPI entre os nodos de processamento por meio da rede SoCIN foi necessário realizar alterações na primitiva ocMPI, como também nas camadas de transporte e *device driver* da biblioteca ocMPI. A camada de aplicação repassa à primitiva Send_ocMPI o identificador do destino para realizar o processo de conversão em coordenadas, conforme (1) e (2). Nesse processo, é atribuído primeiramente o identificador aos enlaces da direção Y, conforme a Figura 61.

$$Index\_to\_address\_x(i) = \left( {}^{i}/_{Mesh\_H} \right) \tag{1}$$

$$Index\_to\_address\_y(i) = \left( i\%Mesh\_H \right) \tag{2}$$

onde:

- *x*: corresponde à coordenada no eixo *x*;
- *y*: corresponde à coordenada no eixo *y*;
- *Mesh_H*: corresponde à dimensão da rede NoC no eixo *y*.

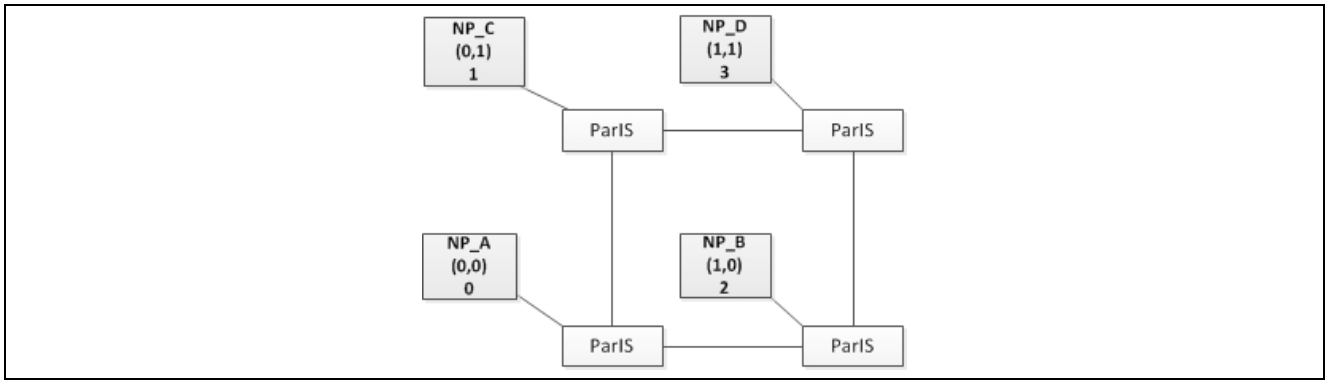


Figura 61. Atribuição do identificador aos enlaces direção YX, versão original

Devido à plataforma proposta atribuir primeiramente o identificador aos enlaces na direção X, foi necessário alterar a conversão das coordenadas conforme (3) e (4). Observe que o identificador percorre o eixo X primeiramente para depois alterar a posição em Y, conforme Figura 62.

$$Index\_to\_address\_x(i) = (i\%Mesh\_W) \tag{3}$$

$$Index\_to\_address\_y(i) = \left( {}^{i}\!/_{Mesh\_W} \right) \tag{4}$$

onde:

- *x*: corresponde à coordenada no eixo *x*;
- *y*: corresponde à coordenada no eixo *y*;
- *Mesh_W*: corresponde à dimensão da rede NoC no eixo *x*.

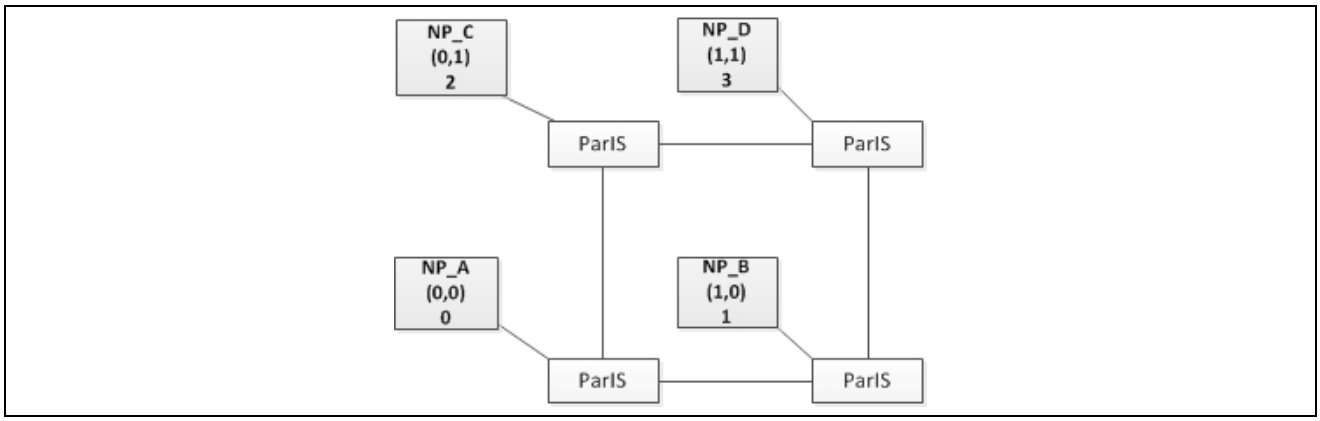


Figura 62. Atribuição do identificador aos enlaces direção XY, versão alterada

A camada de transporte não permite o envio de pacotes com tamanho superior a 8 *flits*, conforme a Figura 63a. Por isso, ela foi alterada de tal forma que mensagens maiores sejam quebradas em pacotes de até 8 *flits* cada, conforme ilustrado na Figura 63b em que uma mensagem de 24 *flits* é quebrada em três pacotes.

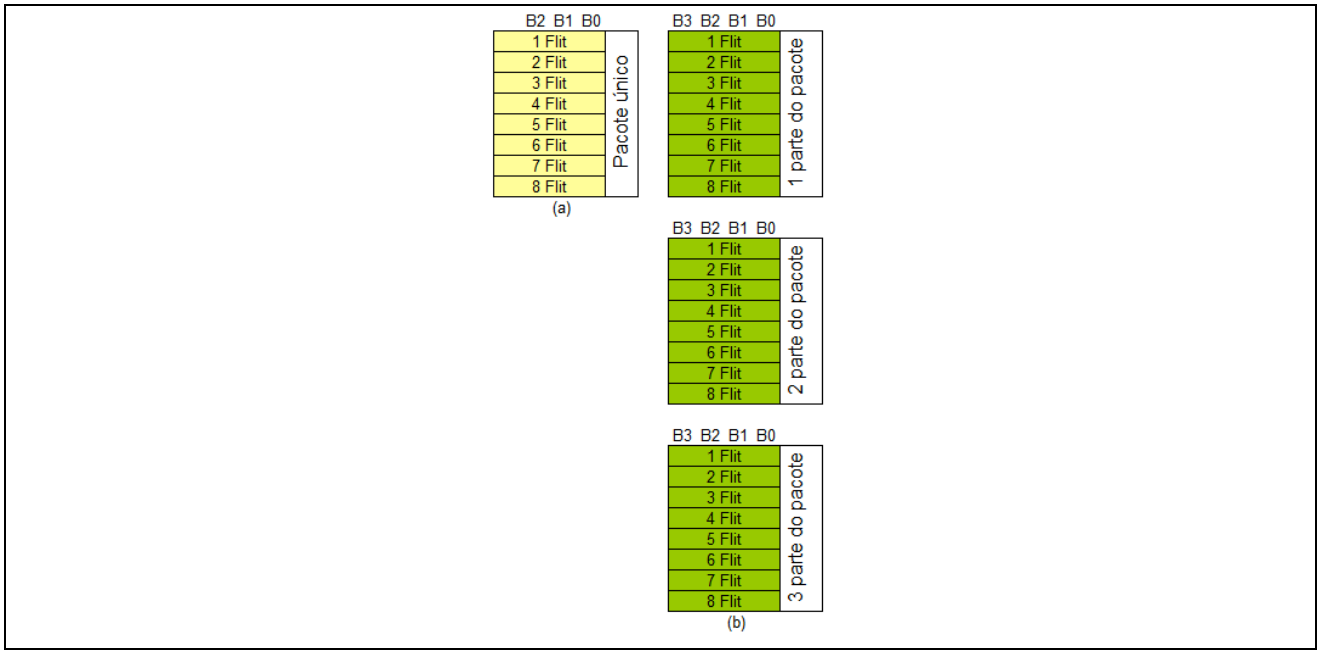


Figura 63. Tamanho do pacote versão: (a) Original; e (b) Alterada

A Figura 64 (a) apresenta o cabeçalho montado pelo *device driver* original e destinado a uma rede específica. Para atender à especificação do cabeçalho da SoCIN, ilustrado na Figura 64 (b), foi necessário alterar a formação do cabeçalho.

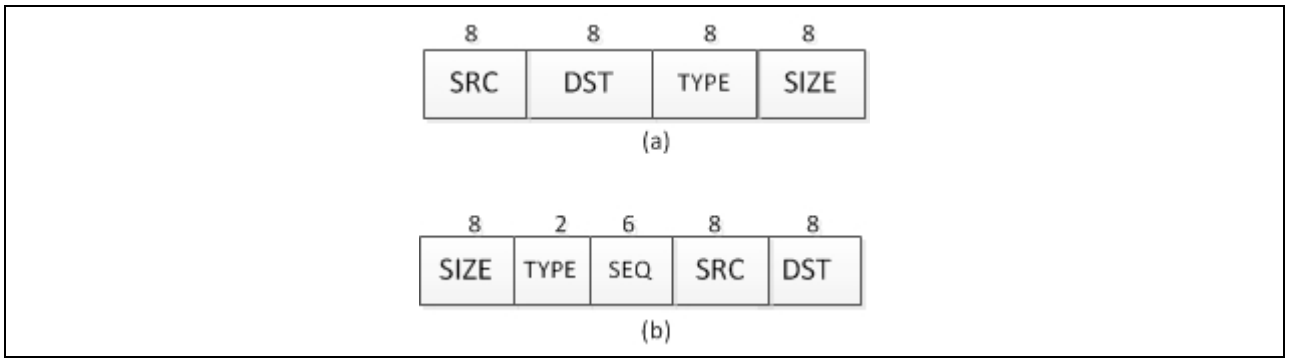


Figura 64. Formato do cabeçalho versão: (a) Original; e (b) Alterada para a SoCIN

## 5.1.4 Plataforma proposta

### 5.1.4.1 Verificação da comunicação entre Supervisor e NS

A quarta etapa consistiu na integração dos núcleos desenvolvidos com a rede NoC SoCIN. Iniciamos pela integração do Supervisor com NS e rede SoCIN, conforme ilustrado Figura 65, gerando o arquivo de configuração dos geradores de tráfego, *traffic.tcf* (anexo A), por meio da ferramenta RedScarf. Esse arquivo é enviado pelo S ao NS pela interface serial.

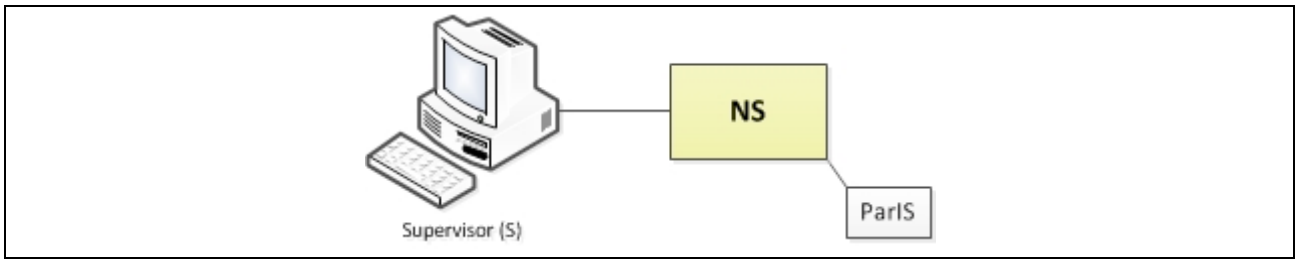


Figura 65. Arquitetura de validação da comunicação entre Supervisor e NS

Foi utilizada a ferramenta de comunicação serial Serial_App (ferramenta de comunicação via protocolo de comunicação RS-232), desenvolvida no laboratório de sistemas embarcados e distribuídos (LEDs), que permite: (*i*) estabelecer a conexão serial com o nodo NS; (*ii*) enviar do arquivo de configuração ao NS no momento desejado; e (*iii*) registrar o intervalo de tempo entre o envio do pacote de configuração ao NS até o retorno dos dados pelo NS. A Figura 66 apresenta a tela do console com impressão de mensagens de inicialização e com os dados do arquivo recebido pelo NS.

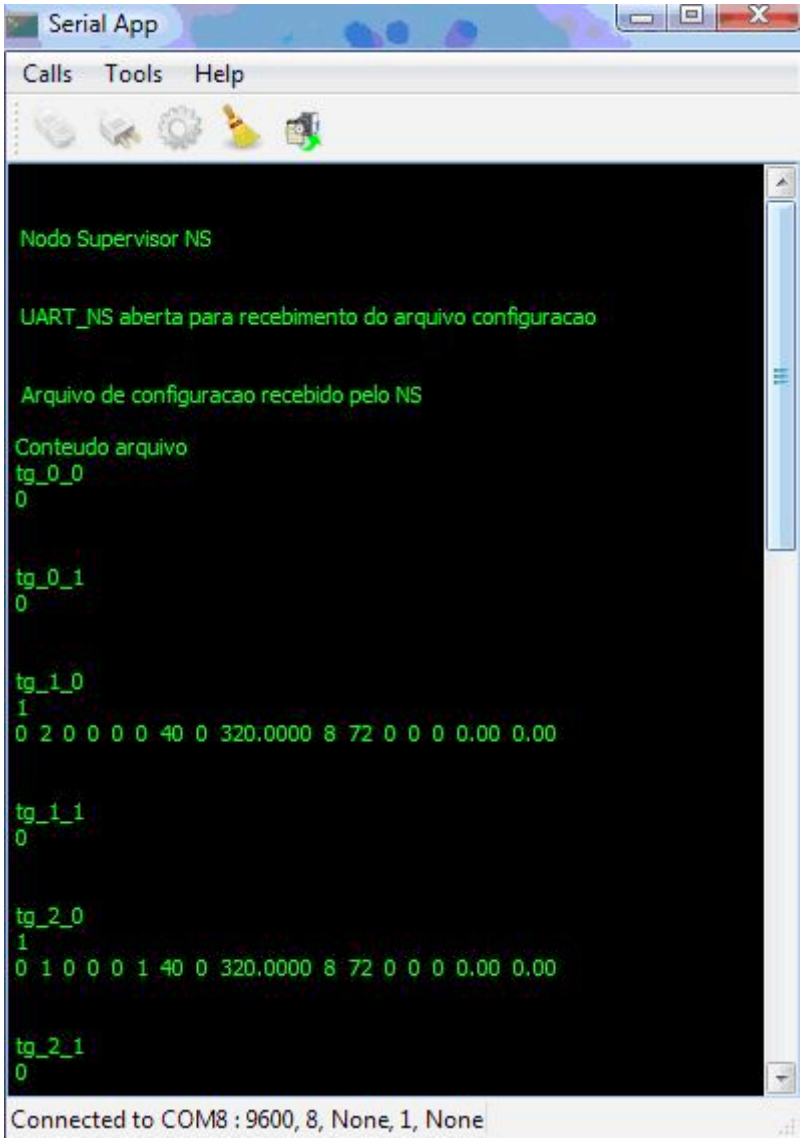


Figura 66. Ferramenta para comunicação serial Serial_App

Após receber o arquivo de configuração o NS extrai as informações necessárias para a montagem do pacote de configuração de cada nodo de processamento. A Figura 67 apresenta a formação do pacote de configuração para um NP com dois tipos de fluxos de dados a serem gerados. O primeiro *flit* corresponde ao cabeçalho onde contém os campos de EOP, BOP, SIZE, TYPE, SEQ, SRC e DST ( Seção 4.4.2 ). Os próximos cinco *flits* são informações do protocolo de comunicação ocMPI ( Seção 4.4.3 ). O sétimo *flit*, Number of Flows, corresponde à informação da quantidade de fluxo que o NP irá gerar.

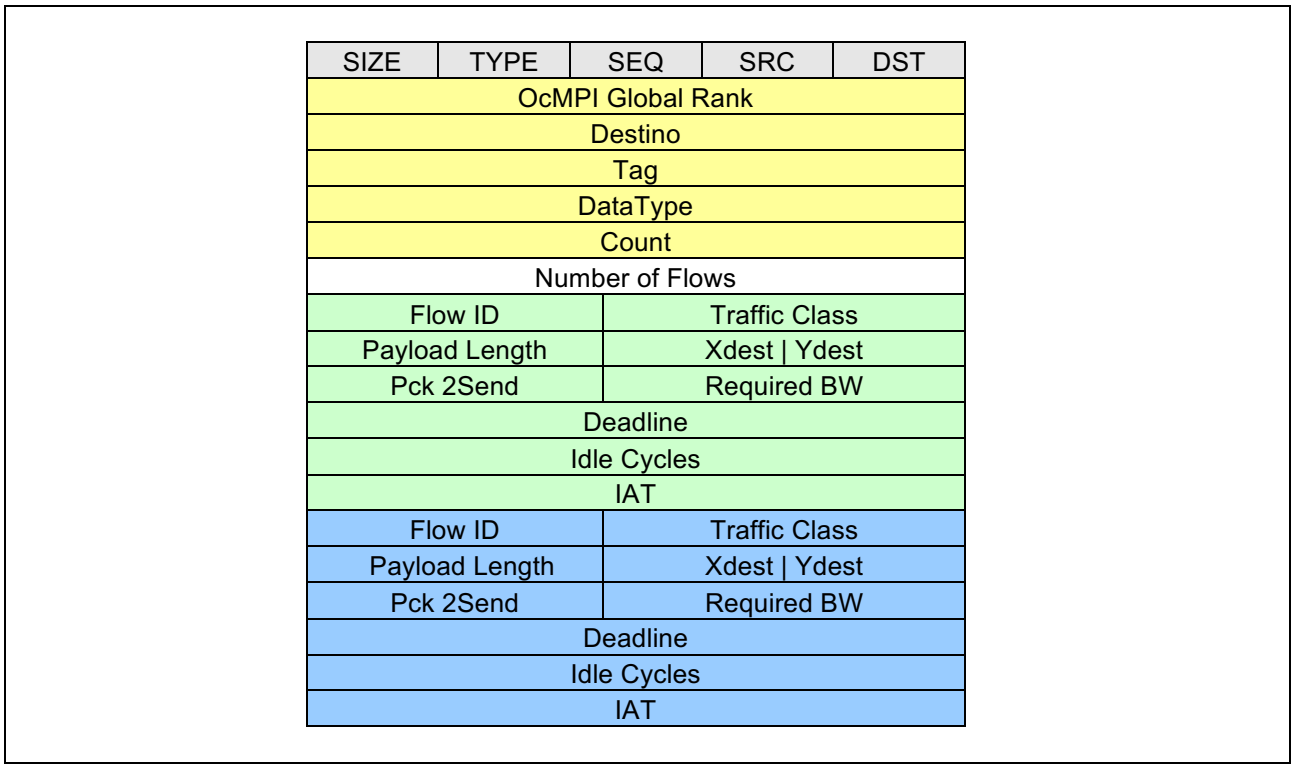


Figura 67. Pacote de configuração para um NP

A especificação do fluxo gerado é obtida pelos campos:

- *Flow ID*: Informa que tipo de fluxo será gerado;
- *Traffic Class*: Informa o tipo de tráfego a ser gerado. Podendo ser RT0 (Sinalização), RT1(Áudio/Vídeo), nRT0 (Leitura/Escrita na memória), nRT1(Transferência de grandes blocos de dado);
- *Payload Length*: Quantidade de *flits* por pacote;
- *Xdest, Ydest*: Coordenadas do nodo de processamento destino;
- *Pck2Send*: Quantidade de pacotes para injetar na rede;
- *Required BW*: Largura de banda requerida;
- *Deadline*: Intervalo de tempo entre o início de um pacote e o início do próximo pacote;

- *Idle Cycles*: Intervalo de tempo entre pacotes (tempo entre o terminador de um pacote e o cabeçalho do próximo cabeçalho – representa o processamento entre o fim de uma comunicação e o início de outra);

- *IAT (Inter-arrival Time)*: Intervalo de tempo para geração de pacotes (tempo entre os cabeçalhos de dois pacotes sucessivos – representa o período de criação de pacotes).

#### 5.1.4.2 Verficação da comunicação entre os NPs

O sistema de veriricação da comunicação entre os NPs foi composto de dois nodos de processamento (NP) e duas interfaces de rede conectadas (inseridas no NP) a uma rede SoCIN 2x1. A validação da comunicação entre os NPs, ilustrada na Figura 68, foi realizada da seguinte forma. Utilizando o protocolo de comunicação ocMPI, o NP_A enviou um pacote pela rede SoCIN ao NP_B, o qual extraiu as informações e respondeu à solicitação gerada pelo NP_A.

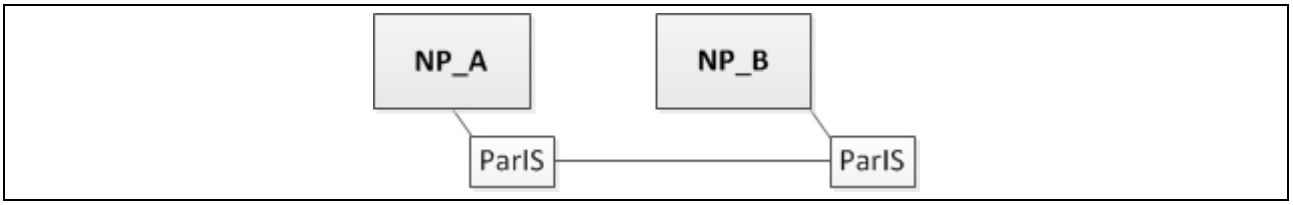


Figura 68. Sistema para validação da comunicação entre os NPs

Foi desenvolvido um aplicativo que realizou um *ping* entre os NPs. O pacote utilizado nessa comunicação contem sete *flits*: um *flit* de cabeçabelho, cinco *flits* de configuração do protocolo ocMPI e um *flit* de dado. O NP_A, conforme a Figura 69, iníciou o processo enviando o pacote citado anteriormente com a informação do dado igual a 1 ao NP_B. Esse processo se repetiu por cinco vezes, o que confirmou o envio e recebimento correto de pacotes corretamente por meio de passagem de mensagem ocMPI.

```
Primeiro FLIT do Pacote enviado = 18C10010          Dado enviado NP_A: 1
Dado recebido NP_A: 2
Primeiro FLIT do Pacote enviado = 18C20010          Dado enviado NP_A: 3
Dado recebido NP_A: 4
Primeiro FLIT do Pacote enviado = 18C30010          Dado enviado NP_A: 5
Dado recebido NP_A: 6
Primeiro FLIT do Pacote enviado = 18C40010          Dado enviado NP_A: 7
Dado recebido NP_A: 8
Primeiro FLIT do Pacote enviado = 18C50010          Dado enviado NP_A: 9
Dado recebido NP_A: 10
```

Figura 69. Console Nios II Build Tool for Eclipse NP_A: cabeçalho e dados

O NP_B, Figura 70, por sua vez após receber e extrair a informação do pacote recebido, o dado é incrementado em um e reenviado ao NP_A.

```
Dado recebido NP_B: 1
Primeiro FLIT do Pacote enviado = 18C11000            Dado enviado NP_B: 2
Dado recebido NP_B: 3
Primeiro FLIT do Pacote enviado = 18C21000            Dado enviado NP_B: 4
Dado recebido NP_B: 5
Primeiro FLIT do Pacote enviado = 18C31000            Dado enviado NP_B: 6
Dado recebido NP_B: 7
Primeiro FLIT do Pacote enviado = 18C41000            Dado enviado NP_B: 8
Dado recebido NP_B: 9
Primeiro FLIT do Pacote enviado = 18C51000            Dado enviado NP_B: 10
```

Figura 70. Console Nios II Build Tool for Eclipse NP_B: cabeçalho e dados

Um componente denominado Performance Counter foi acrescentado a arquitetura do nodo de processamento NP_A para realizar a medida de *overhead* da aplicação. O Performance Counter é um bloco contador em hardware que mede o tempo de execução do código escolhido. A Figura 71 apresenta o resultado obtido pelo Performance Counter para as funções ocMPI_send e ocMPI_Receiver.

```
--Performance Counter Report--
Total Time : 169 usec (16962 clock-cycles)
+---------------+-----+------------+---------------+------------+
| Section       |  %  | Time (usec)|  Time (clocks)|Occurrences |
+---------------+-----+------------+---------------+------------+
|     ocMPI Send|  27 |         47 |          4734 |          1 |
+---------------+-----+------------+---------------+------------+
| ocMPI Receiver|  22 |         38 |          3885 |          1 |
+---------------+-----+------------+---------------+------------+
```

Figura 71. Console Nios II Build Tool for Eclipse: Performance Counter

Observa-se que a função ocMPI_Send gastou 47 µs para montar todo o pacote e envia-lo para a interface de rede. Já a função ocMPI_Receiver gastou aproximadamente 38 µs para receber o pacote e desmontá-lo. Considerando o envio de um pacote, pode-se chegar ao cálculo do tempo de transmissão e recepção, conforme apresentado na Figura 72. O sistema de validação de comunicação entre núcleo, interface de rede e SoCIN, gastou 85,06 µs para transmitir o pacote do NP_A e recebê-lo no NP_B, sendo que a rede em si gasta apenas 0,06 µs para encaminhar o cabeçalho do pacote.

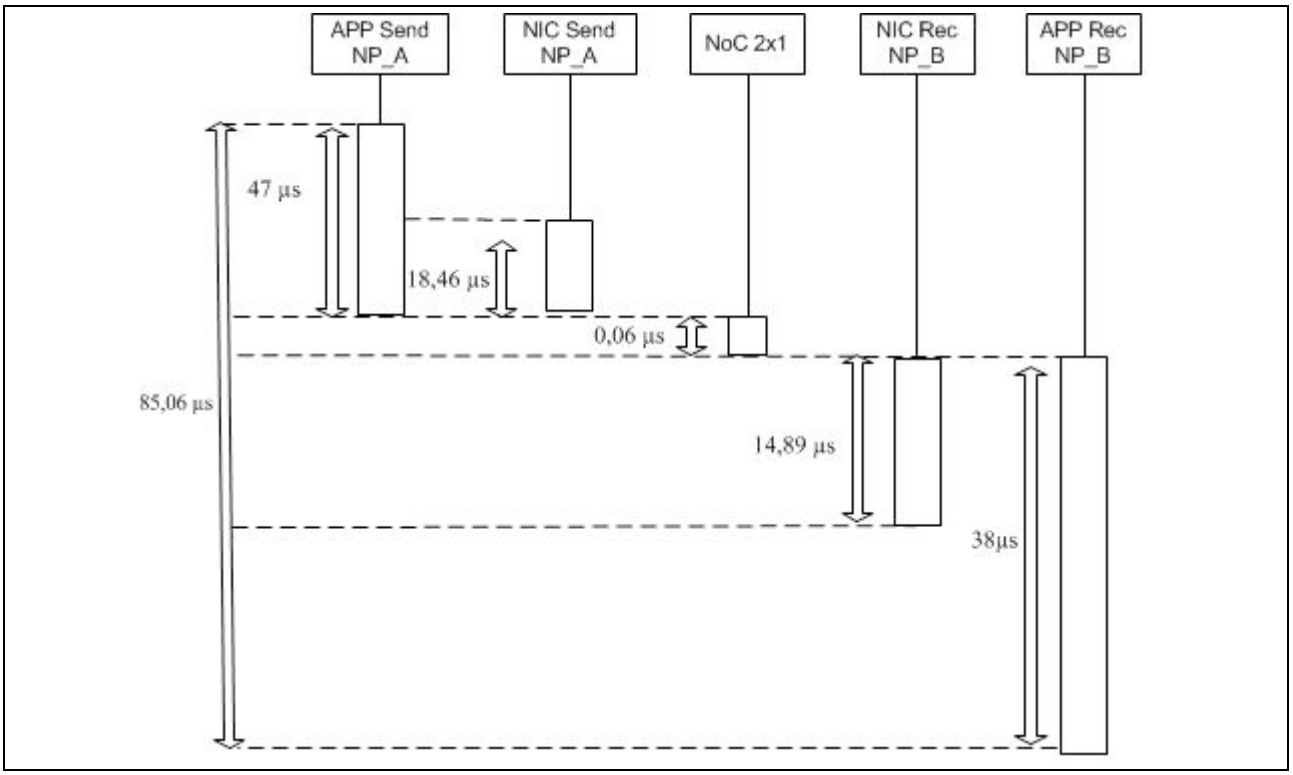


Figura 72. Diagrama de sequência do sistema de validação entre NPs e rede SoCIN.

A transferência do corpo do pacote pela rede não é ilustrada. O tempo associado a essa transfência está incorporado no tempo de processamento da interface de rede destino, pois ocorre em paralelo. Nota-se, portanto, que a biblioteca de comunicação acrescenta um alto sobrecusto na latência da mensagem, o qual é devido às rotinas de comunicação da API, da camada de transporte e do *device driver*. Esse sobrecusto, porém, se justifica por facilitar a implementação da aplicação.

#### 5.1.4.3 Verificação da comunicação entre Supervisor, NS e NP

A Figura 73 ilustra o sistema utilizado para verificar: (*i*) a comunicação entre Supervisor, NS e NP; e (*ii*) o envio pelo NS dos pacotes de configuração do gerador de tráfego para o NP via rede SoCIN. Nesta fase, o sistema executou o processo de configuração e habilitação (Figura 40) do experimento de emulação. O Supervisor enviou pela interface serial o arquivo *traffic.tcf* para o NS, o qual extraiu as seguintes informações para a formação do pacotes de configuração do gerador de tráfego: (*i*) quantidade de fluxos; (*i*) identificador de fluxo; (*ii*) classe de tráfego; (*iii*) destino, (*iv*) quantidade de *flits*; (*v*) quantidade de pacotes; (*vi*) *Deadline*; (*vii*) *IdleCycles*; e (*viii*) *IAT*.

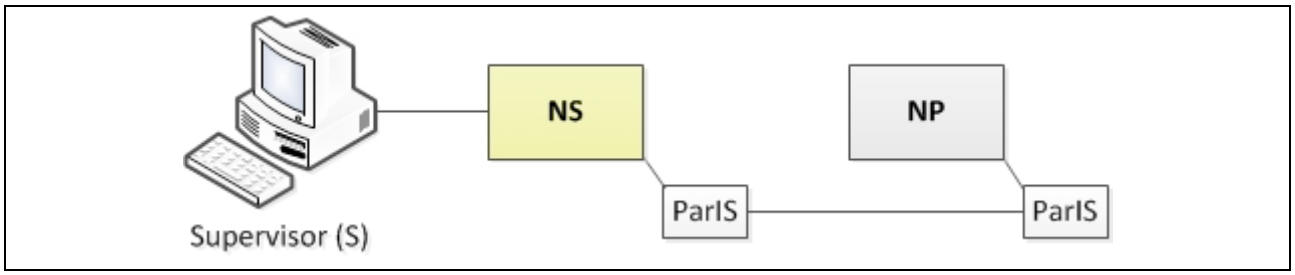


Figura 73. Arquitetura de validação da comunicação entre S, NS e NP

O NS montou um novo pacote de configuração do gerador de tráfego e o encaminhou ao NP especificado pela rede SoCIN utilizando o protocolo de comunicação ocMPI.

Foi necessário desenvolver dois aplicativos, um para o NS e outro para o NP. O aplicativo do NS realiza a leitura do arquivo da serial, extrai os dados para geração de tráfego e forma um novo pacote de configuração de tráfego para ser enviado ao NP correspondente (ver molduras A e B na Figura 74). Após o termino desse processo, é encaminhado um pacote com as informações de início de processo de geração de tráfego (ver moldura C na Figura 74).

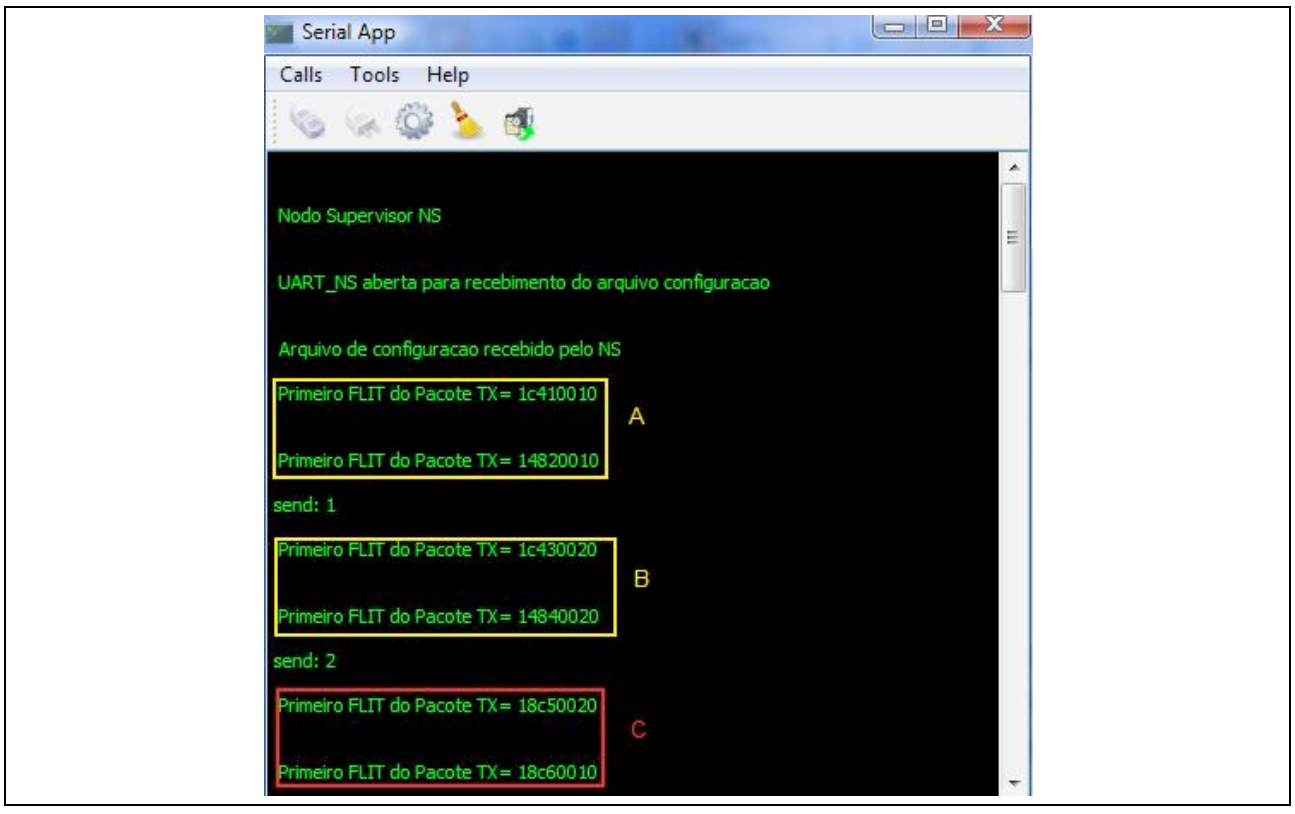


Figura 74. Cabeçalho dos pacotes de configuração e inicialização do gerador de tráfego

O pacote de configuração enviado pelo NS ao NP é formado por 14 *flits*. Esse pacote é fragmento em duas parte, a primeira parte é formada por um *flit* de cabeçalho, cinco *flits* de configuração ocMPI e dois *flits* de dados. Já a segunda parte é formada por um *flit* de cabeçalho e cinco *flits* de dados. A verificação da comunicação foi feita por meio de impressões no console da ferramenta Serial_App, conforme ilustrado na Figura 75. O aplicativo do NP extrai os dados do arquivo de configuração recebido e configura o tráfego a ser gerado na rede.

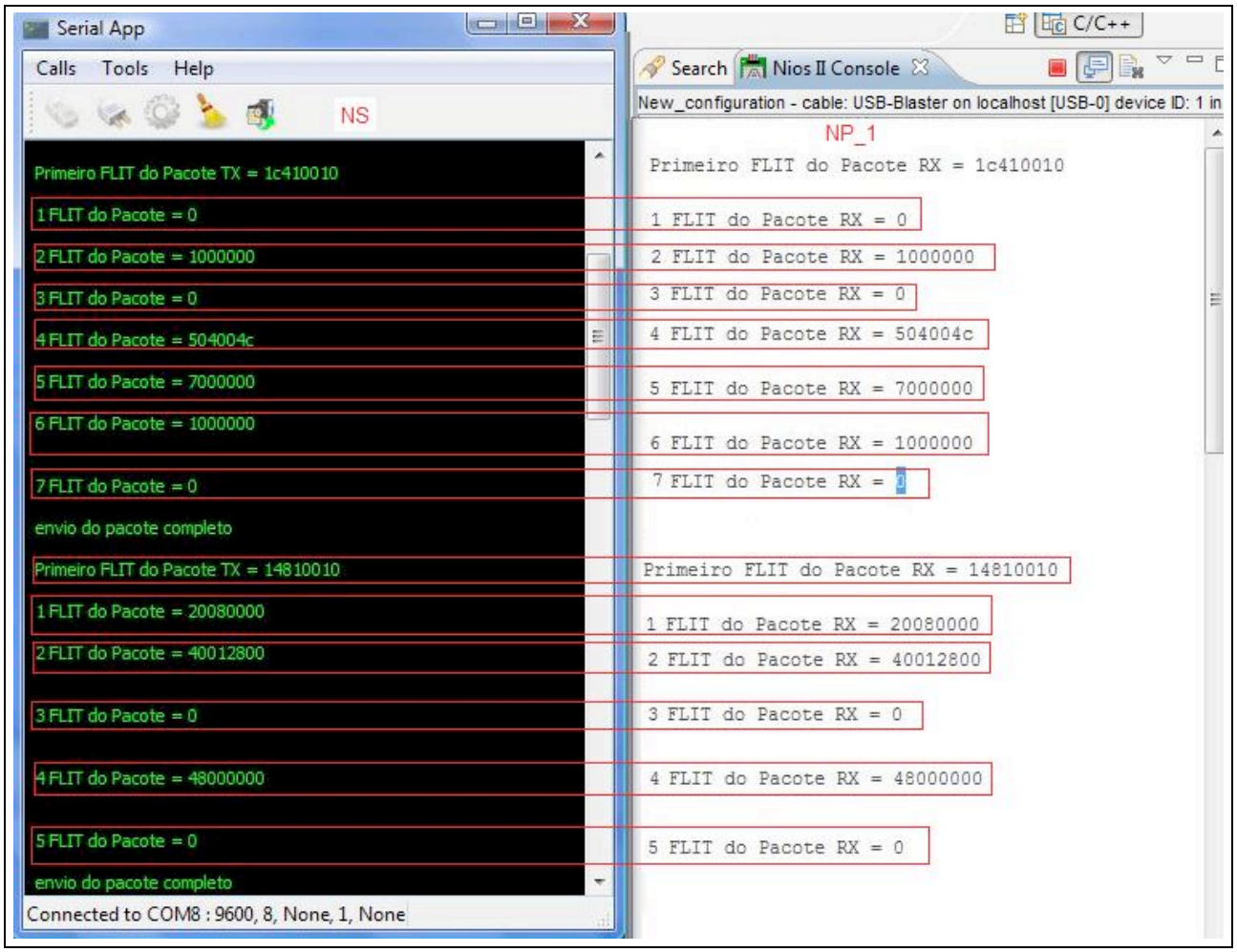


Figura 75. Mensagens apresentadas no console da ferramenta Serial_App

Para verificar o funcionamento de geração de tráfego, foi adicionado um nodo de processamento ao sistema formando um sistema 3x1, conforme ilustrado na Figura 76.

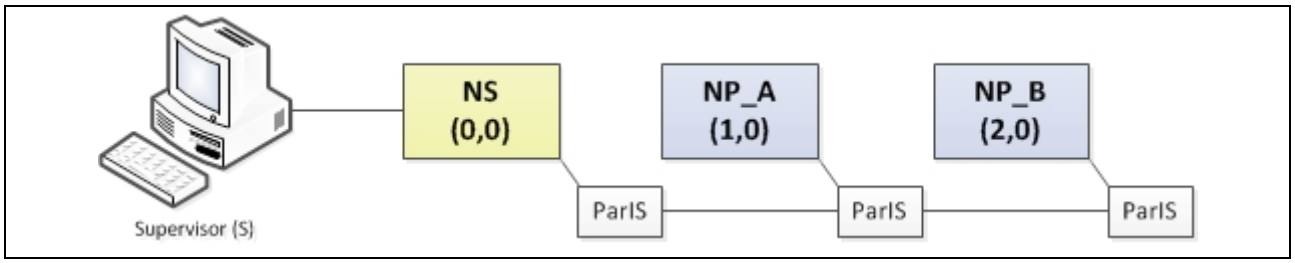


Figura 76. Arquitetura de validação da comunicação entre Supervisor, NS e 2 NPs

O NS é posicionado no roteador de endereço (0,0) e os nodos de processamento nos roteadores de endereço (1,0) e (2,0). Após a configuração, cada NP recebeu do NS os parâmetros para geração de fluxo, incluindo:

- *Payload Length*: 8 *flits*;
- *Xdest, Ydest*: (1,0) ou (2,0) - ou seja, o endereço do outro NP do sistema; e
- *Pck2Send*: 40 pacotes.

A partir dessas informações, cada NP gera 40 pacotes com 8 *flits* (cabeçalho seguido de 224 bits de dado) destinados ao outro nodo de processamento. A verificação da comunicação foi feita por meio de impressões no console da ferramenta Nios II Sotfware Build Tool for Eclipse, conforme ilustrado na Figura 77 que mostra os cabeçalhos dos pacotes injetados na rede pelo nodo (1,0) destinado ao nodo (2,0). Além das coordenadas do remetente e do destinatário (bytes 0x10 e 0x20), destaca-se o número de sequência que é incrementado a cada pacote (bytes 0xc1, 0xc2, etc, onde os 6 bits menos significativos representam o número de sequência) e.o tamanho do corpo do pacote em bytes (identificado pelo byte mais significativo, 0x1c, que é igual a 28). Após gerar todos os pacotes para a rede, o NP encaminha um pacote para o NS para notificar que enviou todos os seus pacotes (conforme definido na Figura 40).

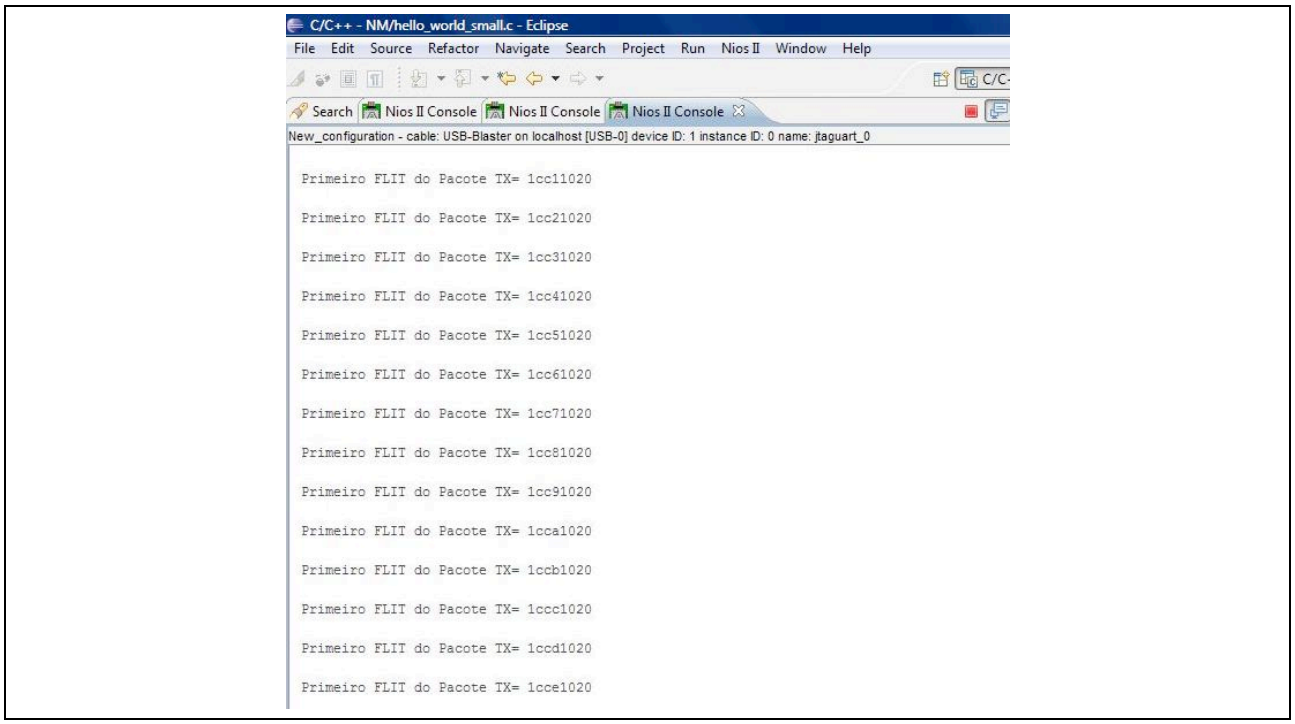


Figura 77. Console Nios II Build Tool for Eclipse – NP: pacotes transmitidos

#### 5.1.4.4 Verificação da comunicação entre Supervisor e NM

O sistema ilustrado na Figura 78 foi utilizado para verificar: (*i*) a extração dos dados dos pacotes injetados e ejetados na rede SoCIN pelo NM; e (*ii*) o acesso à memória compartilhada entre NS e NM.

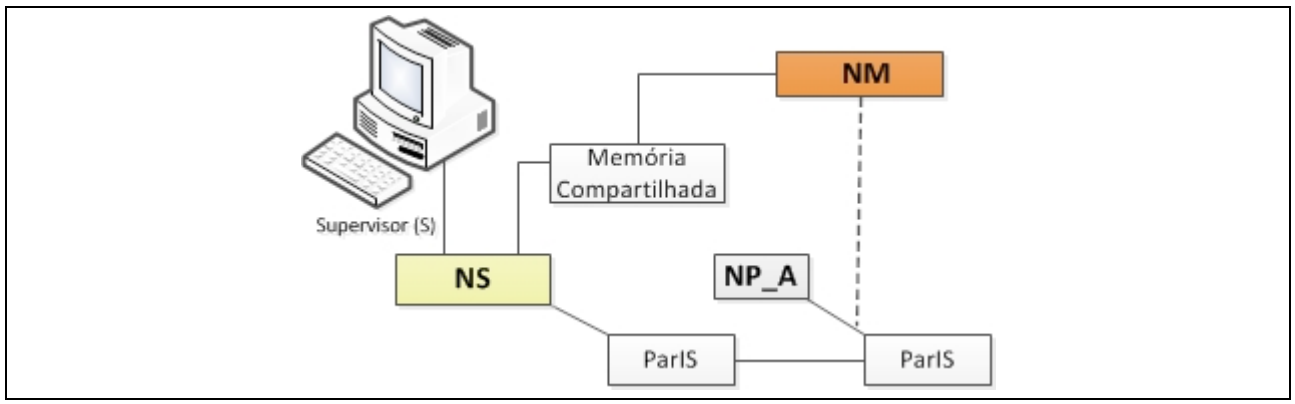


Figura 78. Arquitetura de validação da comunicação entre NS e NM

O NM captura todos os cabeçalhos de ínicio dos pacotes enviados e recebidos dos NPs com seus tempos de injeção ou extração na rede SoCIN. A Figura 79, apresenta os cabeçalhos extraídos. Por exemplo, a moldura A apresenta o pacote com cabeçalho 0x1CC12010 com seu respectivo

tempo de injeção (19644255**193**) na rede SoCIN e o tempo de recebimento no destinatário (1964255**207)**. Observa-se que a latência do pacote foi de 14 ciclos de relógio (207-193), ou seja, 6 *flits* para transferência do cabeçalho pelos roteadoes da rede, 1 ciclo para recebimento do cabeçalho na interface de rede e 7 ciclos para recebimento do restante do pacote. Como não há concorrência com outros fluxos, todos os demais pacotes experimentaram essa mesma latência.

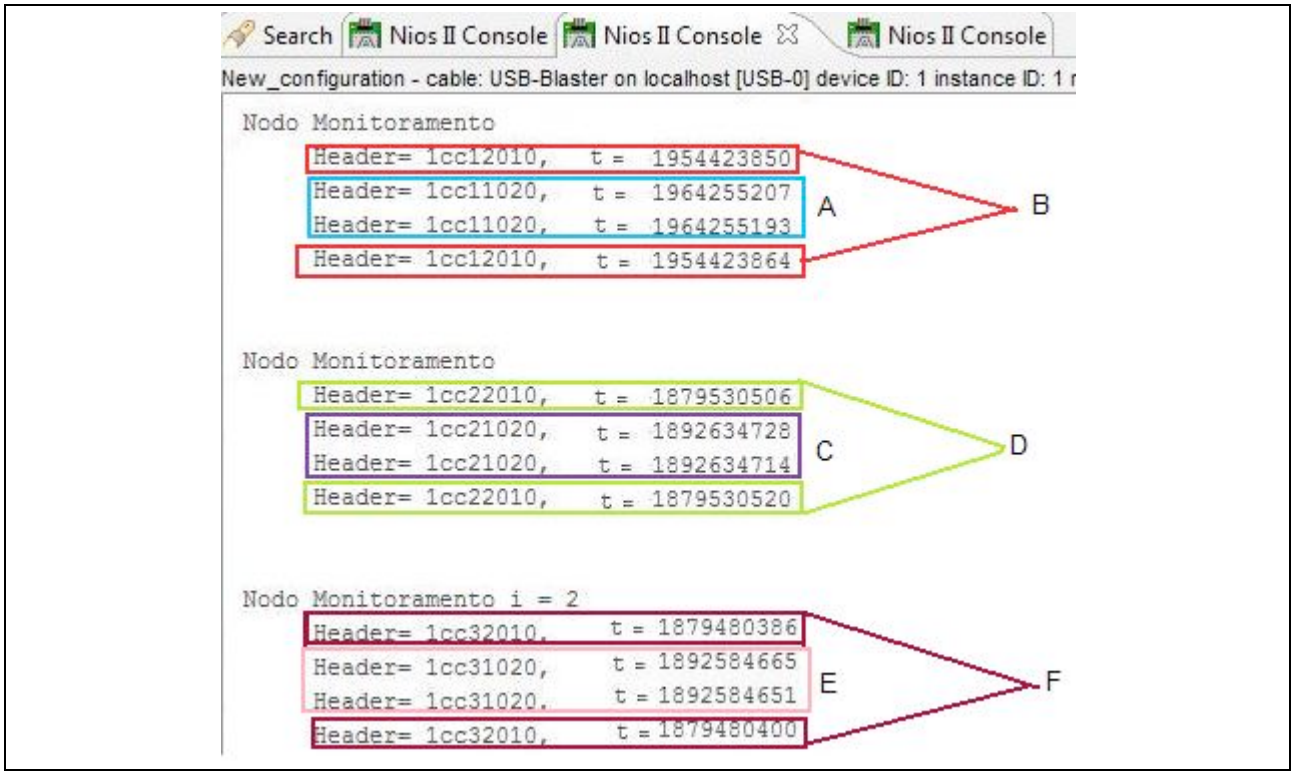


Figura 79. Console da Nios II Build Tool for Eclipse – NM: extração dos tempos

O componente Mutex irá infomar ao NS ou NM se a memória compartilhada está liberada ou bloqueada para utilização. O Mutex irá liberar o acesso ao NS apenas após o armazenamento dos dados capturados pelo NM, conforme a Figura 80. A partir desse momento, o NS realiza o processo de leitura na memória compartilhada e encaminha os da dos pela interface serial ao Supervisor na mesma ordem em que foram capturados originalmente. Cabe ao software do Supervisor identificar corretamente, para cada pacote, qual informação refere-se ao tempo de injeção na rede e qual se refere ao tempo de recebimento no destinatário, dado que este último será maior que o primeiro.

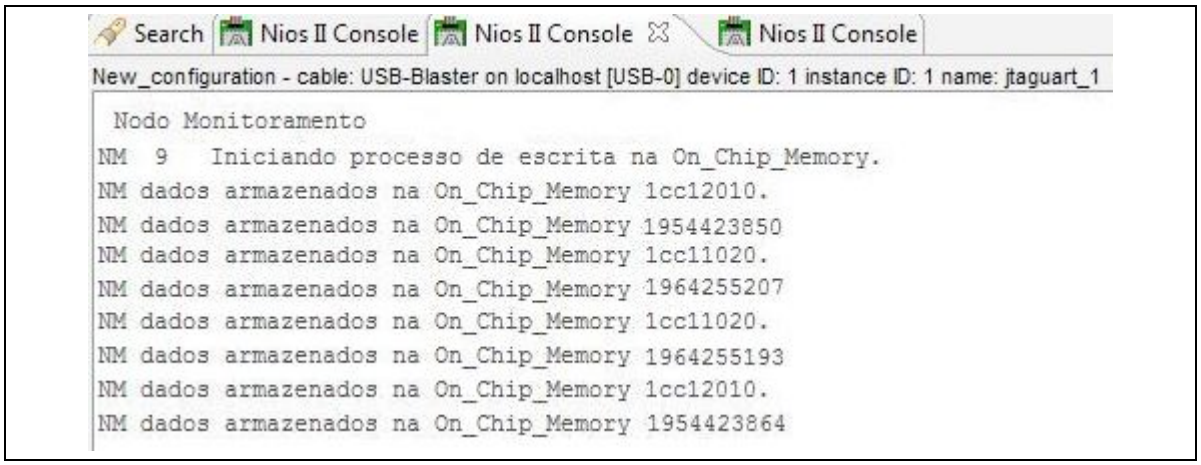


Figura 80. Console da Nios II Build Tool for Eclipse – NM: dados coletados

#### 5.1.4.5 Verificação da comunicação da plataforma completa

O último sistema foi utilizado para verificar a comunicação da plataforma completa, conforme a Figura 81.

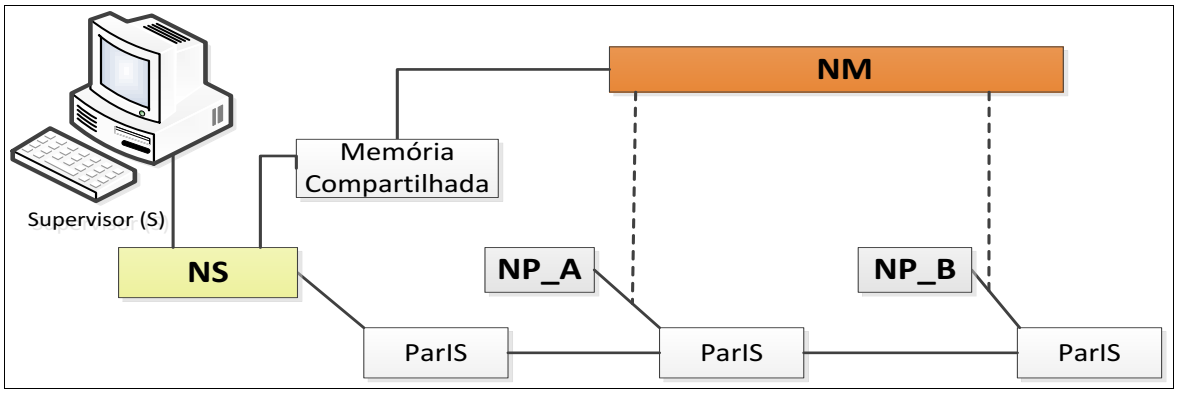


Figura 81. Arquitetura de validação da plataforma implementada

Foram verificados todos os processos ilustrados previamente na Figura 40, incluindo:

- Validação do processo de configuração: configuração do experimento com envio de descritores de fluxo pelo Supervisor ao NS e deste aos NPs;
- Validação do processo de habilitação: envio de mensagens do NS aos NPs para habilitar o início do experimento;
- Validação do processo de finalização: envio de mensagens dos NPs ao NS para finalizar o experimento;

- Validação do processo leitura da memória compartilhada: leitura dos dados armazenados na memória compartilhada pelo NS;
- Validação do processo de envio dos dados capturados pelo NM: Enviou dos dados coletados pelo NS ao Supervisor.

## 5.2 AVALIAÇÃO

A plataforma de avaliação de desempenho proposta foi avaliada em um sistema com dois nodos de processamento (NP), um nodo supervisor (NS) e um nodo de monitoramento (NM), conforme ilustrado na Figura 81. A partir dessa configuração, foi possível extrair os dados a seguir.

### 5.2.1 Custo de silício

A plataforma implementada foi sintetizada para o dispositivo de FPGA EP4CE30F23C7 da família Cyclone IV da Altera com o uso da ferramenta Quartus II – versão 13.1 de 32 bits. A Tabela 3 apresenta o consumo dos recursos do FPGA expressos pelas quantidades de elementos lógicos (LEs – Logic Elements), Look-Up Tables (LUTs), Flip-Flops (FFs) e blocos de memória *on-chip* de 9 Kbits (M9K) para os principais componentes do sistema. Não são mostrados os custos de periféricos utilizados, como a UART e um temporizador. As LUTs referem-se ao custo com lógica combinacinal, enquanto os FFs e os bits de memória representam o custo com lógica sequencial. Cada elemento lógico é composto de uma LUT e um FF, sendo que é possível que apenas um de cada desses recursos seja de fato utilizado em um LE. Por isso a quantidade de LUTs e FFs ocupados é geralmente menor que a de LEs.

Tabela 3. Recursos utilizados no FPGA para cada componente

| **Componente** | **LEs** | **LUTs** | **FFs** | **M9K** |
|---|---|---|---|---|
| Nodo supervisor (NS) | 1.141 | 1.065 | 665 | 3 |
| Interface de rede | 1.368 | 886 | 1.139 | 0 |
| Nodo de processamento (NP) | 3.084 | 2.455 | 2.086 | 21 |
| Nodo de monitoramento (NM) | 6.290 | 4.675 | 4.599 | 13 |
| Bloco de Aquisição Medidas (TX e RX) | 4.003 | 2.638 | 3.226 | 0 |
| SoCIN*fp* 3x1 | 5.068 | 3.246 | 4.065 | 0 |

Quanto à memória, é importante destacar que mesmo que um componente precise de 10 Kbits, por exemplo, poderão ser alocados dois ou mais blocos M9K para esse componente, dependendo de como ele utiliza sua memória. Os dados da tabela mostram que o componente de

maior custo em termos de LEs é o nodo monitoramento, enquanto o nodo de processamento é o que consome mais blocos de memória.

A Tabela 4, apresenta os recursos disponíveis no FPGA utilizado e os custos para sintetizar o sistema implementado. Também são apresentadas as taxas de ocupação que evidenciam que o sistema implementado ocupa mais de 90% dos elementos lógicos e dos blocos de memória do dispositivo, o que limitou o tamanho do sistema sintetizado.

Tabela 4. Recursos utilizados do FPGA considerando um sistema 3x1

| | **LEs** | **LUTs** | **FFs** | **M9K** |
|---|---|---|---|---|
| Dispositivo EP4C30F23C7 | 28.848 | 28.848 | 28.848 | 66 |
| Recursos ocupados pela plataforma | 26.281 | 16.377 | 21.691 | 61 |
| Taxa de ocupação | 91,1% | 56,8% | 75,2% | 92,5% |

### 5.2.2 Desempenho

Uma das métricas de desempenho para avaliar a plataforma proposta é a frequência máxima de operação do sistema implementado no FPGA. Essa informação é obtida com o uso da ferramenta TimeQuest Timing Analyzer do Quartus II, a qual determina o tamanho do caminho crítico (circuito com maior tempo de propagação) que limita o período máximo do sinal de relógio. Para o sistema implementado, a frequência máxima obtida é de 108,5 MHz. Ou seja, a plataforma física não pode operar a frequências maiores que este valor.

Uma segunda métrica de desempenho é o tempo para execução de um experimento no FPGA que é função da frequência de relógio utilizada (respeitando-se o limite máximo de operação). Para isso, foi utilizada uma frequência de 100 MHz e executado um experimento em que cada nodo de processamento é configurado para enviar 40 pacotes de 8 *flits* (32 bits de cabeçalho mais 224 bits de dado) para o outro nodo de processamento. O intervalo de tempo entre dois pacotes sucessivos é definido pelo tempo gasto pelos *device driver* para construir o pacote e injetá-lo na rede, ou seja, 47 µs (conforme mostrado previamente na Figura 72). Com isso, a taxa de injeção de dados é limitada a 4,76 Mbps (224 bits de dados / 47µs).

Para realizar um experimento é necessário que o Supervisor envie o arquivo de confirguração ao NS pela conexão serial e que este configure cada NP. Após, os NPs enviam pacotes um para o outro, o NM coleta e armazena informações sobre os pacotes na memória

compartilhada, a qual é lida pelo NS ao fim do experimento para envio dos dados ao Supervisor. Considerando todas essas etapas, a execução do experimento consumiu 3,2 segundos.

Embora não tenha sido feita a medição do custo de cada uma das etapas, estima-se que o envio de 40 pacotes de um nodo a outro da rede, sem intervalos adicionais ao tempo da camada de software, gaste o equivalente a 1,88 ms, ou seja, 40 x 47µs por pacote (não está contabilizada a latência da rede, 0,06 µs, por ser muito menor que a latência do *device driver*). Como os fluxos enviados pelos dois NPs não concorrem um com o outro por recursos da rede (canais e *buffers*), os dois fluxos de pacotes são iniciados e finalizados ao mesmo tempo. Dessa forma, cerca de 99,97% do tempo de execução deve-se às etapas inicial (configuração do experimento) e final (envio dos dados), e apenas 0,03% ao experimento em si. Para pagar o custo da configuração e coleta, seria necessário simular um volume muito maior de pacotes. No entanto, a solução implementada utiliza memória embutida no FPGA para armazenar os dados coletados, e o FPGA utilizado não permite aumentar a quantidade de pacotes. Esse problema pode ser contornado das seguintes formas: (*i*) utilização de um FPGA com maior capacidade de memória *on-chip*, mas que mesmo assim teria um limite de pacotes; (*ii*) executar um experimento em sub etapas para descarregar a memória quando a mesma estivesse cheia, utilizando uma memória FIFO, o que exige uma parada do experimento de tempos em tempos; e (*iii*) utilizar memória externa do kit, com um controlador de memória no FPGA, o que diminuiria bastante as restrições das demais soluções. Nenhuma dessas alternativas foi experimentada no projeto.

Uma terceira métrica de desempenho é aceleração do experimento em hardware em comparação com a sua execução em um simulador. O mesmo experimento foi executado no RedScarf considerando dois nodos enviando 40 pacotes de 224 bits de dados (mais 32 bits de cabeçalho) um para o outro. Para capturar o custo das camadas de software da plataforma de hardware, a taxa de injeção de dados foi definida em 4,76 Mbps (224 bits de dados / 47µs para injeção de um pacote, para um relógio de 100 MHz). O experimento foi executado em um computador laptop com processador Intel Core 2 Duo de 2.1 GHz e 4 GB de memória e levou aproximadamente 10 segundos. Ou seja, nesse caso, a aceleração foi cerca 3,1 vezes (10s / 3,2s), o que se considera pouco em relação ao reportado na literatura.

Desconsiderando-se as etapas de configuração e coleta de dados do experimento em hardware, seria obtida uma aceleração da ordem de 5.319 vezes (10s / 1,88ms), o que já está dentro do esperado. Logo, para que seja observado um benefício real na abordagem em hardware, é

necessário contornar o problema já exposto a respeito da memória e que limita o desempenho dessa solução. Outro fator limitante do desempenho é o meio de comunicação utilizado entre o Supervisor e o NS, uma conexão serial baseada na RS-232 com *baud rate* de 9.600 bps. Seria necessário utilizar um *baud rate* maior ou outro padrão de comunicação (ex. USB, Ethernet), o que ainda não foi experimentado.

Um dos objetivos da plataforma proposta é identificar métricas de desempenho do experimento (ex. latência média das mensagens transferidas e tráfego aceito pela rede). No experimento realizado, observou-se que a latência média de comunicação na rede foi de 14 ciclos de relógio, mesmo valor identificado no simulador. Quanto ao tráfego aceito (vazão), dadas as limitações do FPGA, não foi possível implementar um sistema que permitisse realizar experimentos de forma a identificar variações na vazão em função da taxa de injeção. Isso porque, considerando o alto custo das camadas de software, a taxa de injeção de dados é bastante baixa (máximo 4,7 Mbps) se comparada com a largura de banda do canal (3.200 Mbps = 32 bits x 100 MHz). Uma solução para contornar esse problema seria conectar a rede a um relógio que operasse a uma frequência muito mais baixa (ex. 1 MHz) que a do sinal de relógio dos nodos de processamento (ex. 100MHz). Com isso, o custo do empacotamento representaria muito menos ciclos da rede que no experimento realizado. Por exemplo, ao invés da injeção representar 0,14% da banda do canal (4,7 Mbps / 3200 Mbps), ela poderia significar 14% da banda (4,7 Mbps / 32 Mbps). Se a rede operasse a uma frequência ainda menor (ex. 100 KHz), seria possível atingir o seu ponto de saturação.

## 5.3 CONSIDERAÇÕES

### 5.3.1 Atendimento dos requisitos

A plataforma de avaliação de desempenho proposta neste trabalho foi submetida aos procedimentos de verificação apresentados neste capítulo. A partir dos resultados obtidos, foi possível averiguar o atendimento dos requisitos funcionais e não funcionais. Dos dez requisitos funcionais (RF) e oito requisitos não-funcionais (RNF) apresentados na Seção 4.2.2 um não foi atendido integralmente (RNF06) devido às limitações dos FPGAs dos kits utilizados, dois não puderam ser verificados devido ao tamanho do sistema (RF05 e RF08) e três tiveram que atualizados por restrições de implementação (RF10, RNF03 e RNF04), conforme discutido a seguir.

O RNF06 estabelecia que a plataforma deveria ter um custo que permitisse a integração de sistemas com pelo menos quatros NPs em kits de prototipação educacionais (ex. Terasic DE2, Macnica/DHW Mercurio IV e Terasic DE1-SoC). Foi feito um projeto inicial com o kit DE2 que se mostrou insuficiente. Após, o projeto foi migrado e desenvolvido no kit Mercúrio IV, porém, ao final da integração, este também mostrou limitado para a implementação de um sistema com quatro NPs. Ainda não foi possível verificar a integração no kit DE1-SoC, porém estima-se que esse suporte um sistema com as dimensões previstas originalmente. Isso porque o dispositivo utilizado nesse kit (Cyclone V, modelo 5CSEMA5F31C6N) possui 85.000 elementos lógicos e 397 blocos M10K (com 10 Kbits de memória *on-chip* em cada bloco). Vale lembrar que o sistema implementado consumiu 26.281 LEs e 61 blocos M9K. Considerando os custos relativos a cada nodo de processamento adicionado ao sistema (ou seja, interface de rede, roteador, bloco de aquisição no NM), estima-se que seja possível integrar um sistema com sete NPs e um NS interconectados por uma rede 4x2 (ou por uma rede 3x3).

O RF05 estabelecia que a plataforma deveria ser capaz de gerar fluxos de comunicação para diferentes destinatários na rede, enquanto o RF08 estabelecia que cada NP deveria ser capaz de escalonar o envio de pacotes de cada fluxo de forma balanceada, evitando que um determinado fluxo seja postergado indefinidamente. Esses dois requisitos não foram vericados, pois o sistema implementado contem apenas dois NPs e seria necessário pelo menos um NP adicional.

Na primeira versão da plataforma proposta, o NS não contemplava em sua estrutura interna o componente para controlar a SDRAM externa, pois se considerou que a memória *on-chip* conseguiria armazenar todo o programa deste nodo (NS). Porém, quando foi carregada a biblioteca para a análise dos arquivos recebidos pela interface serial, não foi possível armazenar o programa na memória *on-chip*. Consequentemente, foi retirado o acesso à memória externa pelo NM para uso pelo NS. Os dados coletados pelo NM na nova versão são armazenados na memória *on-chip* compartilhada com o NS. Em função dessas alterações, alguns dos requisitos originais do projeto tiveram que ser alterados conforme mostrado a seguir. Os requisitos originais estão marcados com texto tachado e ao lado são apresentadas as modificações realizadas.

- RF10: ~~O NM deve ser capaz de coletar informações sobre os pacotes injetados e ejetados da rede e enviar essas informações ao supervisor por meio de uma interface serial, enviando um stream de bits.~~ O NM deve ser capaz de coletar informações sobre os

pacotes injetados e ejetados da rede e enviar essas informações ao NS por meio de uma memória compartilhada e este ao Supervisor por meio de uma interface serial.

- RNF03: ~~O NS e os NPs devem possuir uma memória local.~~ O NS, o NM e os NPs devem possuir uma memória local.
- RNF04: ~~O NM deve possuir uma memória externa.~~ O NS deve possuir uma memória externa.

### 5.3.2 Respostas às perguntas de pesquisa

Esta dissertação buscou responder a quatro perguntas de pesquisa, conforme discutido a seguir.

1. *Quais são as vantagens e as limitações da plataforma de avaliação de desempenho proposta em relação às soluções similares em FPGA, em particular aos trabalhos desenvolvidos por Pereira (2008) e Frantz (2008)?*

   A plataforma proposta permite, a partir do sistema de hardware definido, gerar qualquer tipo de fluxo sem a necessidade de alterar software ou hardware. É necessário apenas o envio de um novo arquivo de configuração. A plataforma apresenta uma limitada memória compartilhada para armazenamento dos dados coletados na rede e não permite comunicação direta entre o Supervisor e o NM, necessitando assim que o NS envie os dados para o Supervisor. Devido à memória limitada, a implementação atual possui restrições quanto à quantidade de pacotes coletados e requer de solução alternativa para contornar esse problema.

2. *Qual é o ganho da plataforma em relação ao tempo de execução de um experimento quando comparado com a simulação em SystemC?*

   Para o sistema implementado e considerando os experimentos realizados, a aceleração foi cerca de 3,1 vezes. Porém, desconsiderando-se os tempos para de configuração e coleta de dados do experimento em hardware, estima-se que a aceleração da execução do experimento é da ordem de 5.319 vezes.

3. *Como superar as limitações dos trabalhos desenvolvidos por Pereira (2008) e Frantz (2008)?*

   A limitação de flexibilidade de configuração do sistema apresentado no trabalho desenvolvido por Pereira (2008) foi superado, porém o tempo para geração do pacote que era um dos limitantes do trabalho de Frantz não foi contornado. Isso ocorreu devido ao uso de uma API de programação paralela baseada na biblioteca ocMPI, dado que o envio de um pacote consome 4.700 ciclos de relógio do processador, enquanto o recebimento gasta 3.800 ciclos de relógio. Apesar disso, entende-se que o uso da biblioteca traz suas vantagens por facilitar o densenvolvimento de aplicações baseadas em processamento paralelo.

### 5.3.3 Verificação das hipóteses de pesquisa

Na proposta desta dissertação foram definidas quatro hipóteses de pesquisa que são discutidas a seguir:

1. *Um processador de propósito geral de 32 bits requer menos ciclos que um processador de 8 bits para a geração de tráfego.*

   O processador Nios II permite gerar o tráfego com menos ciclos de relógio que o processador de 8 bits, porém, o uso da biblioteca ocMPI acrescenta um sobrecusto de tempo importante o que não permitiu confirmar essa hipótese na prática.

2. *Um processador de propósito geral de 32 bits oferece maior flexibilidade para geração de fluxos que uma implementação baseada em processador de propósito específico*.

   A plataforma baseada em software executando em um processador de propósito geral pode ser facilmente reconfigurada pelo envio de um arquivo à mesma pela interface serial. Conforme demonstrado, a carga, a execução e a coleta de dados de um experimento são feitas em poucos segundos. Já a implementação baseada em processador de propósito específico implementada por Pereira (2008) requer a recompilação da plataforma e a reconfiguração do FPGA a cada nova configuração.

Embora o autor não informe o tempo necessário para isso, sabe-se que é muito maior que o tempo para configurar, executar e coletar os dados de um experimento na plataforma desenvolvida neste trabalho.

3. *Um processador de propósito geral de 32 bits consome maior área de silício que implementações baseadas em um processador de 8 bits ou em processador de propósito específico*.

   A plataforma implementada por Pereira (2008) foi baseada em SPPs e verificada por meio da síntese de um sistema com quatro nodos de geração de tráfego (equivalentes aos NPs) interconectados por uma rede 2x2. O sistema foi sintetizado em um FPGA da família Cyclone II da Altera e ocupou 5.619 LUTs e 4.747 FFs, segundo informado pelo autor. Já a plataforma implementada neste trabalho, com dois NPs, consumiu 16.377 LUTs e 21.691 FFs, demandando, portanto, muito mais recursos. Já a plataforma implementada por Frantz (2008), baseada em um processador de propósito geral de 8 bits, não foi sintetizada em FPGA e o autor não apresenta métricas de custo que permitam a comparação. Entratanto, por ser um sistema de complexidade menor, considera-se que seu custo também seja menor.

4. *A emulação da NoC em FPGA baseada em um processador de propósito geral de 32 bits permite realizar experimento de avaliação de desempenho em um tempo menor que o obtido pela simulação em SystemC.*

   Conforme mostrado, a plataforma em hardware é 3,1 vezes mais rápida que o simulador. Porém, se for considerado apenas o tempo associado à execução do experimento, a aceleração é de cerca 5.319 vezes.

# 6 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de uma plataforma MPSoC para avaliação de desempenho da NoC SoCIN em FPGA. A plataforma desenvolvida é baseada em processador programável de propósito geral (Nios II) e em uma biblioteca de comunicação para programação paralela (ocMPI). O trabalho envolveu uma pesquisa bibliográfica para idenficação dos conceitos base e dos trabalhos relacionados, o projeto e a implementação da plataforma, a verificação física e a extração de métricas para avaliação.

A plataforma desenvolvida oferece flexibilidade para geração de tráfego e uso da biblioteca de comunicação traz benefícios para o desenvolvimento de aplicações com programação paralela, o que favorece o aumento do desempenho. Porém, a plataforma apresenta um sobrecusto expressivo tanto em relação ao consumo de recursos físicos do FPGA, quanto à latência da comunicação. Esses sobrecustos limitam o tamanho do sistema e o uso da largura de banda da rede.

Os experimentos realizados permitiram responder as perguntas de pesquisa e verificar as hipóteses estabelecidas. Com relação aos objetivos específicos da dissertação, considerando-se os trabalhos anteriores do grupo de pesquisa, conseguiu-se aumentar a flexibilidade para geração de tráfego em FPGA e reduzir o tempo para avaliação de desempenho em relação à simulação. Porém, como já citado, não foi possível reduzir a latência da comunicação, devido ao uso da biblioteca ocMPI. Nesse aspecto, é importante destacar que o trabalho permitiu evidenciar que a latência da NoC para a transmissão de uma mensagem tem impacto pequeno na latência total da comunicação. As camada de software da biblioteca MPI possui um alto custo de processamento impactam de sobremaneira no tempo de comunicação, o que aponta para a necessidade de otimizar essas camadas. Uma solução possível de ser considerada é implementar a camada mais baixa, de empacotamento e desempacotamento, em hardware.

A principal contribuição do trabalho reside na disponibilização de uma plataforma física operacional, com flexibilidade para programação de aplicações sintéticas ou reais baseadas na biblioteca de comunicação ocMPI. A plataforma ainda possui algumas limitações e, por isso, como trabalhos futuros, propõe-se:

1. Implementar soluções para contornar as limitações do nodo de monitoramento para coleta de dados (memória) e também para aumentar a escalabilidade da solução;
2. Integrar um sistema com mais nodos de processamento no kit de desenvolvimento DE1-SoC e avaliar diferentes cenários de tráfego sintético com a geração de múltiplos fluxos de comunicação (ex. tráfego uniforme);
3. Implementar e avaliar a execução de *benchmarks* de programação paralela (ex. ordenação de vetores, transformada rápida de Fourier, entre outros); e
4. Portar as funcionalidades de empacotamento e desempacotamento para o nível de hardware de forma a reduzir a latência da comunicação.

# REFERÊNCIAS

ALHONEN, A.; SALMINEN, E.; NIEMINEN, J.; HAMALAINEN, T. D. A Scalable, Non-Interfering, Synthesizable Network-on-chip Monitor. In: NORCHIP, 28., 2010, Tampere. **Proceedings…** [S.l.]: IEEE Computer Society, 2010. p. 1-6.

ALTERA, Corporation. **Nios II Processor Reference**: Handbook. San Jose: Altera, v. 11, 2011.

ALTERA, Corporation. **Nios II Hardware Development**: Tutorial. San Jose: Altera, 2014.

ALTERA, Corporation. **Cyclone IV Device Handbook**. San Jose: Altera, v. 2.0, 2016.

BALWAIK, R. R.; NAYAK, S. R.; JEYAKUMAR, A.; Open-source 32 bits risc soft-core processors.In: INTERNATIONAL ORGANIZATION OF SCIENTIFIC RESEARCH JOURNAL OF VLSI AND SIGNAL PROCESSING (IOSR-JVSP) 2013. **Proceedings…** [S.l.: s.n.], 2013. p. 43-46.

BRUCH, Jaison Valmor; PIZZONI, Magnos Roberto; ZEFERINO, Cesar Albenes. BrownPepper: a SystemC-based simulator for performance evaluation of Networks-on-Chip. In: IFIP/IEEE INT. CONFERENCE ON VERY LARGE SCALE INTEGRATION (VLSI-SOC 2009), 17., 2009, Florianópolis. **Proceedings…** [S.l.: s.n.], 2009. p. 1-4.

CIORDAS, C.; BASTEN T.; RADULESCU, A.; GOOSSENS, K.; MEERBERGEN, J. V. An Event-Based Network-On-Chip monitoring Service. In: HIGH-LEVEL DESIGN VALIDATION AND TEST WORKSHOP, 9., 2004, Washington. **Proceedings…** [S.l.]: IEEE Computer Society, 2004. p. 149-154.

COPOLLA, M.; GRAMMATIKAKIS, M. D.; LOCATELLI, R.; MARUCCIA, G.; PIERALISI, L. **Design of cost-efficient interconnect processing units**: Spidergon STNoC. Broken Sound Parkway: CRC Press, 2009.

DALLY, Willian James; TOWLES, Brian. **Principles and practices of interconnection networks**. San Francisco: Elsevier, 2004.

DORTA, T.; JIMÉNEZ, J.; MARTIN, J. L.; BIDARTE, U.; ASTARLOA, A. Reconfigurable Multiprocessor Systems: A Review. **International Journal of Reconfigurable Computing**, New York: Hindawi Publishing Corporation, 2010.

DUATO; José; YALAMANCHILI, Sudhakar; NI, Lionel**. Interconnection networks**: an engineering approach. San Francisco: Elsevier, 2003.

FERNANDEZ-ALONSO, Eduard; CASTELLS-RUFAS, David. A NoC-based Multi-{soft} core with 16 cores. In: INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS, AND SYSTEMS (ICECS), 17., 2010, Athens. **Proceedings…** [S.l.]: IEEE Computer Society, 2010. p. 259-262.

FERNANDEZ-ALONSO, Eduard; CASTELLS-RUFAS, David; JOVEN, Jaume; CARRABINA, Jordi. Embedding MPI in Many-Soft-Core Processors. In: INTERNATIONAL CONFERENCE ON

HIGH-PERFORMANCE AND EMBEDDED ARCHITECTURES AND COMPILERS. (HIP3ES), 8., 2013, Berlin. **Proceedings…** [S.l.]: Computers and Electrical Engineering. Elsevier,2012. v.38 p.756-771.

FRANTZ, Lucio. **Gerador de tráfego para Redes-em-Chip baseado no Picoblaze**. 2008. Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade do Vale do Itajaí, Conselho Nacional de Desenvolvimento Científico e Tecnológico. Itajaí, 2008.

GEBALI, F.; ELMILIGI, H.; KHARASHI, M. W. **Networks-on-Chip**: theory and practice. Broken Sound Parkway: CRC Press, 2009.

GENKO, N.; ATIENZA, D.; MICHELI, G. D.; MENDIAS, J. M.; HERMIDAS, R.; CATTHORR, F. A Complete Network-On-Chip Emulation Framework. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE - DATE, 2005, Munich. **Proceedings…** Washington: IEEE Computer Society, 2005. v.1, p. 246-251.

HARTONO, D.; LEE, S. W.; YAP, V. V.; NG, M. S.; TANG, C. M. A Multicore System using NoC Communication for Parallel Coarse-Grain Data Processing. In: CONFERENCE ON NEW MIDIA STUDIES(CoNMedia), 2013, Tangerang. **Proceedings…** Washington: IEEE Computer Society, 2013. p. 1-5.

HECK, G.; GUAZZELLI, R.; SOARES, R. I.; MORAES, F. G.; CALAZANS, N. HardNoC: A Platform to Validate Networks on Chip through FPGA Prototyping. In: SOUTHERN PROGRAMMABLE LOGIC CONFERENCE - SPL, 8., 2012, Bento Gonçalves. **Proceedings…** Pelotas: UFPEL, 2012. v.1, p. 15-20.

HOU, N.; ZHANG, D.; DU, G.; SONG, Y.; WEN, H. Design and Performance Evaluation of virtual-channel Based NoC. In: ANTI-COUNTERFEITING, SECURITY, AND IDENTIFICATION IN COMMUNICATION, 3., 2009, Hong Kong. **Proceeding…** Piscataway: IEEE Press, 2009. p. 294-298.

HUBNER, Michael; BECKER, Jurgen. **Multiprocessor System-on-Chip**: Hardware design and tool integration. New York: Springer, 2011.

HWANG, Kai; XU, Zhiwei. **Scalable parallel computing**: technology, architecture, programming. New York: McGraw-Hill, 1998.

JANTSCH, Axel; TENHUNEN, Hannu. **Networks on Chip** (eds). Dordrecht: Kluwer, 2003.

JERGER, Natalie E.; SHIUAN, Li. **On-Chip Networks**. Madison: Morgan & Claypool, 2009.

LOTLIKAR, S.; PAI, V.; GRATZ, P. V. AcENoCs: Configurable HW/SW Plataform for FPGA Accelerated NoC Emulation. In: CONFERENCE ON VLSI DESIGN, 24., Chennai, 2011. **Proceedings…** Washington: IEEE Computer Society, 2011. p. 147-152.

LUO-FENG, G.; GAO-MING, D.; DUO-LI, Z.; MING-LUN, G.; NING, H.; YU-KUN, S. Design and Performance Evaluation of 2D-Mesh Network on Chip Prototype Using FPGA. In: CONFERENCE ON CIRCUIT SYSTEM - IEEE Asia Pacific, Macao, 2008. **Proceedings…** [S.l.]: EEE Computer Society, 2008. p. 1264-1267.

MAHADEVAN, S.; ANGIOLINI, F.; STORGAARD, M.; OLSEN, R. G.; SPARSO, J.; MADSEN, J. A Network Traffic Generator Model for Fast Network-on-Chip Simulation. In: DESIGN, AUTOMATION AND TEST IN EUROPE, Munich, 2005. **Proceedings…**Washington: IEEE Computer Society, 2005. v.2, p. 780-785.

MESSAGE PASSING INTERFACE FORUM. MPI: **A Message-Passing Interface Standard**. University of Tennessee. V 3.1, 2015.

PASRICHA, Sudeep; DUTT, Nikil. **On-Chip Communication Architectures**: System on Chip Interconnect. New York: Elsevier, 2008.

PEREIRA, Thiago F. **Gerador de Tráfego Sintetizável para Redes-em-Chip**. Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade do Vale do Itajaí, Conselho Nacional de Desenvolvimento Científico e Tecnológico. Itajaí, 2008.

PIZZONI, Magnos R. **Plataforma para avaliação de desempenho de Redes-em-Chip em FPGA**. Dissertação (Mestrado em Computação Aplicada) – Programa de Pós Graduação em Computação Aplicada, Universidade do Vale do Itajaí, Conselho Nacional de Desenvolvimento Científico e Tecnológico. Itajaí, 2010.

RAJSUMAN, Rochit. **System on-a-Chip**: Design and Test. Santa Clara: Artech House, 2000.

RUFAS, D. C.; ALONSO, F. E.; CARRABINA, J. Performance analysis techniques for multi-soft-core and many-soft-core systems. In: **International Journal of Reconfigurable Computing**, 2012. New York, Hindawi publishing corporation, 2012.

SANTO, F. G. M. E.; ZEFERINO, C. A.; SUSIN, A. A. Uma Arquitetura de Roteador Parametrizável para a Síntese de Redes-em-Chip. In: CONGRESSO BRASILEIRO DE COMPUTAÇÃO, 4, 2004, Porto Alegre. **Proceedings…** [S.l.: s.n.], 2004.

SILVA, Eduardo A. **RedScarf:** ambiente para avaliação de desempenho de Rede-em-Chip. Itajaí, 2014. 87p. Trabalho Técnico-científico de Conclusão de Curso (Graduação em Ciência da Computação) – Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2014.

TEDESCO, L. P. **Uma proposta para geração de tráfego e avaliação de desempenho para NoCs**. 2005, 108 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2005.

TEDESCO, T.; MELLO, A.; GARIBOTTI, D.;CALAZANS, N.; MORAES, F.Traffic generation and performance evaluation for mesh-based NoCs. In: ANNUAL SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, 18., 2005, Florianopolis. **Proceedings…** New York: ACM, 2005. p. 184-189.

WANG, Laung-Terng; STROUD, Charles A.; TOUBA, Nur. A. **System on Chip Test Architectures**: nanometer design for testability. Burlington: Elsevier, 2008.

WEN, H.; DU, G.; ZHANG, D.; GENG, L.; GAO, M.; CHEN, Y. Design of An On-line Configurable Traffic Generator for NoC. In: CONFERENCE ON ANTI-COUNTERFEITING,

SECURITY, AND IDENTIFICATION IN COMMUNICATION – ASID, 3., 2009, Hong Kong. **Proceedings…** Piscataway: IEEE Press, 2009. p. 556-559.

WOLF, W.; JERRAYA, A. A.; MARTIN, G. Multiprocessor System-on-Chip (MPSoC) Technology. In: COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, 2008. **Proceedings…** Piscataway: IEEE Press, 2008. v.27. p. 1701-1713.

WOLKOTTE, Pascal.; HOLZENSPIES, P. K. F.; SMIT, G. J. M. Fast, Accurate and Detailed NoC Simulations. In: INTERNACIONAL SYMPOSIUM ON NETWORK-ON-CHIP - NOC, 2007, Princeton. **Proceedings…** Washington: IEEE Computer Society, 2007. p. 323-332.

ZEFERINO, C. A. **Redes-em-Chip:** arquiteturas e modelos para avaliação de área e desempenho. Tese (Doutorado em Ciência da Computação) - Programa de Pós-Graduação em Computação, UFRGS, Porto Alegre, 2003.

ZEFERINO, C. A.; SUSIN, A. A. SoCIN: a parametric and scalable Network-on-Chip. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS, 16., 2003, São Paulo. **Proceedings…** Washington: IEEE Computer Society, 2003. p. 169.

ZEFERINO, C. A.; BRUCH, J. V.; PEREIRA, T. F.; KREUTZ, M. E.; SUSIN, A. A. Avaliação de desempenho de Rede-em-Chip modelada em SystemC. In: WORKSHOP DE DESEMPENHO DE SISTEMAS COMPUTACIONAIS E DE COMUNICAÇÃO – WPERFORMANCE, 2007, Rio de Janeiro. **Anais do XXVII Congresso da Sociedade Brasileira de Computação**. Porto Alegre: Sociedade Brasileira de Computação, 2007. p. 559-578.

ZEFERINO, C. A.; PEREIRA, T. F. Soft Cores for Performance Evalution of NoCs in FPGA. In SOUTH SYMPOSIUM ON MICROELECTRONICS, 23., 2008, Pelotas. **Proceedings…** [S.l:.s.n], 2008.

ZHOU, Q.; SONG, Yu-kun; ZHANG, Duo-li; DU, Gao-ming. A design of multi-core system based Avalon bus. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND NETWORK TECHNOLOGY, 2011, HARBIN. **Proceedings...** Washington: IEEE Computer Society, 2011. p. 1456-1459.

# APÊNDICE A – PROTOCOLO DE BUSCA

A revisão sistema sistemática é uma forma de pesquisa que utiliza como fonte de dados a literatura, trabalhos recentes sobre o assunto de pesquisa, permitindo a integração de um conjunto de estudo realizados separadamente que podem apresentar resultados conflitantes ou coincidentes, auxiliando nas orientações para investigações futuras. Na revisão sistemática o plano de condução da pesquisa é descrito através de um protocolo de busca, onde são levantados questionamentos sobre o tema, informações como será realizado o processo de busca, critérios de inclusão e exclusão.

**Perguntas de Pesquisa:**

- Que tipo de técnica de avaliação de desempenho está sendo utilizado em NoCs?
- Quais os tipos de geradores e medidores são utilizados para realizar a avaliação de desempenho?
- Quais ferramentas/ metodologias/ procedimentos/ técnicas/ tecnologias são utilizadas para avaliação de desempenho em NoCs?
- Como é implementada a rede?
- Como é implementado o gerador de tráfego?
- Como é implementado o medidor de tráfego?
- Como o sistema é configurado?
- Como os resultados são coletados?
- Como é feita a análise dos resultados?
- Qual é a HDL utilizada?
- Qual é o FPGA?
- Qual é o processador?
- Quais as principais métricas adotadas para realizar a avaliação de desempenho?
- Quais as principais dificuldades encontradas para medir desempenho de uma rede?

- Quais grupos de pesquisa têm se destacado neste assunto?

**Processo de Busca**

Fontes:

- Artigos, livros, teses, dissertações, internacionais e nacionais desde o ano 2000.
- IEEExplore, ACM DL, pesquisadores/grupos, banco de teses CAPES e outras universidades a partir de 2000.

Sites:

- http: //scholar.google.com.br/
- http://bdtd.ibict.br/
- http://www.citeulike.org
- http://ieeexplore.ieee.org
- http://portal.acm.org'
- http://www.sbc.org.br/bibliotecadigital
- http://citeseerx.ist.psu.edu

**Critérios de Inclusão**

Para a questão primária: Serão incluídos no estudo trabalhos cujos títulos e resumos contenham informações referente a desempenho em NoCs, desempenho em NoCs utilizando MPSoCS. A conclusão será analisada para verificar a contribuição do trabalho. A data de publicação do trabalho deve ser superior ou igual ao ano 2000.

Para a questão secundária: Os mesmos critérios da questão primária, porém o título e resumo devem conter também a informação sobre avaliação de desempenho NoCs.

**Critérios de Exclusão**

- Para a questão primária: Serão excluídos do estudo trabalhos cujos títulos e resumos sejam conflitantes, ou seja, o título remete a um assunto enquanto o resumo remete a outro assunto. Os trabalhos publicados antes do ano 2000 não serão analisados.
- Para a questão secundária: Os mesmos critérios da questão primária além de que o título e resumo que não estiverem informando sobre desempenho NoC.

**Processo de seleção preliminar**

As estratégias de pesquisa serão aplicadas para identificar os estudos primários potenciais. Caso um trabalho selecionado não atenda aos critérios de inclusão e também não atenda aos critérios de exclusão, o mesmo será incluído.

**Processo de seleção final**

Cópias dos trabalhos incluídos como resultados da pesquisa inicial serão revisados. Esta revisão conclui a seleção de trabalhos a serem incluídos no processo de extração de dados.

**Reunião de Dados**

- Informação para referência bibliográfica(Fonte /Ano de publicação/ vínculo/ publicação);
- Contexto do trabalho (área e sub-area);
- Tipo de artigo: teórico, experimental ou ambos;
- Problema alvo;
- Solução proposta;
- Metodologia ou materiais utilizados;
- Resultados obtidos;
- Métricas de avaliação;
- Problemas em aberto;

- Vantagens, desvantagens e limitações da solução proposta;
- Real contribuição do trabalho;
- Técnicas de avaliação de desempenho em NoCs;
- Tipos de geradores e medidores utilizados na avaliação de desempenho;
- Ferramentas/metodologias/procedimentos/técnicas utilizadas para avaliação de desempenho em NoCs;
- Métricas de avaliação de desempenho;
- Dificuldades encontradas para medir desempenho de uma NoCs;
- Grupos de pesquisa tem se destacado neste assunto.

**Síntese dos dados extraídos**

Os resultados foram organizados em tabelas. A partir da tabulação dos dados, foram extraídos os dados que possui a maior quantidade de incidentes, identificando qual é o item mais tratado pelos trabalhos relacionados.

# ANEXO A – ARQUIVO.TCF GERADO PELO REDSCARF

```
tg_0_0
0


tg_0_1
0


tg_1_0
1
0  2  0  0  0  0  40  0  320.0000  8  72  0  0  0  0.00  0.00


tg_1_1
0


tg_2_0
1
0  1  0  0  0  1  40  0  320.0000  8  72  0  0  0  0.00  0.00


tg_2_1
0




// Parameters
 0  type
 1  x_dest
 2  y_dest
 3  flow_id
 4  traffic_class
 5  switching_type
 6  pck_2send
 7  deadline
 8  required_bw
 9  payload_length
10  idle_cycles
11  iat
12  burst_size
13  last_payload_length
14  alfa_on
15  alfa_off
```