An MPI-based MPSoC Platform in FPGA

Roseli Uhlendorf, Eduardo Silva, Felipe Viel, *Member, IEEE* and Cesar Zeferino, *Member, IEEE*

***Abstract*—The increase in the level of integration of components on silicon became possible to build systems with multiple processors on a single chip. These systems are named MPSoCs (Multi-Processor Systems-on-Chip), and some of them use Networks-on-Chip (NoC) as the communication infrastructure to interconnect their components. This work presents an MPSoC platform designed for performance evaluation of an NoC in FPGA. The platform relies on the use of 32-bit programmable processors and an MPI (Message Passing Interface) communication library, providing a flexible infrastructure for traffic generation and analysis, and enabling the development of parallel applications. For evaluation, we implemented a proof-of-concept instance of the proposed platform in FPGA, and the experimental results show that the software layer of the communication infrastructure dominates communication latency.**

***Index Terms*—Systems-on-Chip, Multiprocessor Architecture, Networks-on-Chip, FPGA.**

I. Introdução

Nas últimas três décadas, o avanço das tecnologias de fabricação dos circuitos tem propiciado o aumento do nível de integração de componentes em silício, o que viabilizou a construção de sistemas computacionais completos em um único chip. Esses sistemas integrados são denominados SoCs (do inglês, Systems-on-Chip) [1], [2]. SoCs com múltiplos elementos de processamento, sejam eles homogêneos ou heterogêneos, são denominados Multiprocessor SoCs ou MPSoCs (Multi-Processor System-on-Chip). Um MPSoC combina processadores embarcados, hardware digital especializado e, frequentemente, circuitos de sinal misto [3]. Para atender às pressões do mercado e amortizar os custos, as metodologias de projeto de MPSoC se baseiam no reuso de componentes [4], tal como o projeto baseado em núcleo [5]. Um núcleo (do inglês, *core*) é um bloco de silício pré-projetado e pré-verificado com pelo menos 5.000 portas lógicas que pode ser usando na construção de uma aplicação maior ou mais complexa [6]. Dentre os diferentes tipos de núcleo, destacam-se os processadores embarcados, aceleradores para processamento de sinal (áudio, imagem e video), memórias, controladores de interfaceamento e infraestrutura de comunicação, entre outros. Além do reuso de núcleos pré-existentes, um MPSoC pode empregar circuitos especialmente projetados para atender a aplicação alvo.

MPSoCs podem ser construídos utilizando diferentes tecnologias de circuito integrado, como ASIC (Application-specific Integrated Circuit) e FPGA (Field-programmable Gate Array). A tecnologia FPGA possui menor custo de projeto em comparação com a tecnologia ASIC e apresenta um ciclo de desenvolvimento mais curto, o que favorece o atendimento de requisitos de tempo para colocação no mercado (*time-to-market*) [7]. Além disso, o aumento da densidade lógica dos FPGAs, a maior diversidade de núcleos oferecidos nas plataformas de desenvolvimento e a disponibilização de ferramentas de síntese de alto nível para a geração automática de aceleradores de hardware têm viabilizado o rápido desenvolvimento de MPSoCs com um número de componentes cada vez maior [8]–[11]. No entanto, o aumento da quantidade de núcleos em um MPSoC esbarra nas limitações de desempenho e de escalabilidade das arquiteturas de comunicação tradicionalmente usadas em sistemas computacionais, como o barramento e as conexões ponto-a-ponto *ad hoc* [12]. Uma solução para resolver esse problema reside no uso de redes de interconexão chaveadas, as quais são conhecidas como NoCs (do inglês, Networks-on-Chip). Esse tipo de arquitetura de comunicação oferece: (i) confiabilidade e eficiência no gerenciamento de energia; (ii) escalabilidade da largura de banda; (iii) reusabilidade; e (iv) decisões de roteamento distribuídos [13].

O desenvolvimento de uma NoC é constituído por etapas de especificação, projeto arquitetural, implementação, validação e avaliação de desempenho, custos de silício e consumo de energia. Em especial, as etapas de validação e avaliação podem ser realizadas por meio de simulação computacional [14] ou por emulação em hardware [15]–[30]. Nesta última abordagem, o sistema é implementado em circuito físico e o paralelismo inerente da implementação em hardware permite obter tempos de experimentação muito inferiores àqueles alcançados com o uso de simulação computacional [27].

Dado o contexto acima, pesquisadores da Universidade do Vale do Itajaí exploraram soluções para avaliação de desempenho da rede SoCIN (SoC Inteconnection Network) [31] empregando emulação em FPGA. Em [17], os autores desenvolveram dois processadores de propósito único (SPP – Single Purpose Processor) [32] para geração e medição de tráfego em uma plataforma de validação e avaliação de NoC em FPGA. O gerador de tráfego é um sistema digital dedicado composto de uma máquina de estados finitos (controle) e uma unidade de processamento (caminho de dados). Esses blocos são especialmente projetados para injetar pacotes na rede de acordo com um conjunto de parâmetros de configuração do tráfego ($e.g.$, destinatário, tamanho do pacote e taxa de injeção). O medidor de tráfego, também um sistema digital dedicado, coleta dados de cada pacote entregue em um terminal da rede para posterior avaliação do desempenho da NoC. Essa solução se caracteriza por apresentar uma baixa latência para geração de tráfego, mas pouca flexibilidade para modificações no modelo de tráfego. O gerador de tráfego é capaz de injetar apenas quatro fluxos de comunicação diferentes, o que foi

R. Uhlendorf é egressa do Mestrado em Computação Aplicada da Universidade do Vale do Itajaí – Univali, Itajaí, 88302-901 Brasil, e-mail: esor@univali.br

E. Silva, F. Viel e C. Zeferino são professores da Universidade do Vale do Itajaí – Univali, Itajaí, 88302-901 Brasil, e-mail:eas,viel,zeferino@univali.br.

Manuscrito recebido em *data*; revisado em *data*

uma decisão de projeto para evitar um sobrecusto excessivo de recursos lógicos. Já em [33], os autores disponibilizaram uma plataforma física para avaliação de NoCs com geradores de tráfego baseados em um processador embarcado de 8 bits. Essa solução é mais flexível do que aquela apresentada em [17] por utilizar um processador programável para geração do tráfego. Contudo, apresenta uma maior latência para a construção dos pacotes. Como o processador possui uma palavra de dados quatro vezes menor que a largura do canal da rede (32 bits), o *device driver* da interface de rede, escrito em linguagem de montagem, precisa construir palavras de 32 bits e gasta quatro ciclos de relógio na construção de cada palavra da mensagem.

Para contornar as limitações dos trabalhos supracitados, este artigo apresenta uma plataforma MPSoC baseada em NoC integrada em FPGA que utiliza processadores embarcados de 32 bits para emulação de tráfego visando a avaliação do desempenho da SoCIN em FPGA. A solução implementada, apresentada inicialmente em [34], também utiliza a biblioteca de troca de mensagens MPI (Message Passing Interface) para prover uma API (Application Program Interface) de comunicação para os nodos comunicantes que oferece flexibilidade para geração de tráfego estocástico. Além disso, o uso da biblioteca MPI oferece facilidade para desenvolvimento de aplicações paralelas, o que é uma vantagem desta solução em relação às abordagens apresentadas em [17] e [33]. A plataforma proposta foi sintetizada em FPGA, avaliada e caracterizada em relação aos seus custos e desempenho. Os resultados obtidos evidenciam que a plataforma é operacional e que a camada de software da infraestrutura de comunicação é responsável por maior parte da latência de comunicação.

O restante deste artigo é organizado em seis seções. A Seção II discute trabalhos relacionados ao contexto das plataformas de emulação e avaliação do desempenho de NoCs em FPGA. Seguindo, a Seção III apresenta detalhes da arquitetura da plataforma MPSoC proposta. Na sequência, a Seção IV descreve os procedimentos de implementação e verificação, enquanto a Seção V apresenta e analisa os resultados experimentais obtidos. Concluindo, na Seção VI, são apresentadas as considerações finais.

II. Trabalhos Relacionados

O desenvolvimento de plataformas em FPGA para avaliação de desempenho de arquiteturas de NoC tem sido apresentado na literatura ao longo dos últimos 15 anos, com ênfase à avaliação de redes baseadas na topologia em malha 2D [15]–[30]. Enquanto algumas propostas de plataforma utilizam geradores de tráfego estocástico [15], [17], [20], [21], [24], [29] ou baseados em *traces* [15], [20], outras executam aplicações reais [18], [19], [22], [23], [27], [28], [30]. Para a geração de tráfego estocástico, algumas soluções empregam processadores de propósito único construídos especialmente para injetar tráfego sintético na rede [17], [21]. No entanto, a grande maioria das soluções encontradas na literatura utilizam processadores embarcados programáveis para obter maior flexibilidade [15], [16], [18]–[20], [22]–[30]. Uma desvantagem do uso de processadores programáveis é que a latência de processamento para a geração de tráfego estocástico é maior do que quando são utilizados processadores de propósito único. Essa latência pode limitar a taxa de injeção de pacotes e a habilidade dos geradores de tráfego saturarem a rede; a identificação do ponto de saturação é um dos principais objetivos de experimentos de avaliação de desempenho de NoCs. Por outro lado, permitem avaliar a rede em um cenário de sistema muito mais realista.

Em relação às métricas de avaliação mais comumente empregadas nos trabalhos analisados, destacam-se: (i) a aceleração em relação à execução do mesmo experimento em simulador [15], [18]–[20], [23], [27]–[30]; (ii) a latência média dos pacotes transferidos [16], [17], [21], [22], [24], [27], [29]; (iii) o tráfego aceito pela rede (vazão) [16], [17], [26], [27], [29]; e (iv) os recursos ocupados do FPGA [15]–[21], [23]–[25], [28].

Os trabalhos descritos ilustram o cenário de desenvolvimento de soluções MPSoC baseadas em NoC implementadas em FPGA. Em especial, o foco dos trabalhos reside em acelerar a avaliação do desempenho de NoCs, pois o tempo para executar um experimento em hardware, via emulação, é menor que em software, via simulação. As evidências apresentadas nos trabalhos analisados serviram de referência para a definição da arquitetura MPSoC apresentada neste artigo, a qual é descrita a seguir.

III. Plataforma MPSoC

A. Arquitetura da Plataforma

A Fig. 1 apresenta a arquitetura da plataforma proposta. Ela é formada por um computador Supervisor (S) e por um MPSoC implementado em FPGA. O MPSoC integra três tipos diferentes de nodo: (i) Nodo Supervisor – NS; (ii) Nodo de Monitoramento – NM; e (iii) Nodos de Processamento – NPs. Uma rede SoCIN com topologia em malha 3×1 interconecta o NS e os NPs, enquanto um barramento interconecta o NS e o NM a uma memória externa compartilhada. Ressalta-se que o MPSoC ilustrado na figura possui o tamanho mínimo necessário para demonstrar o funcionamento da plataforma proposta. Essa configuração é utilizada ao longo deste trabalho para a avaliação e caracterização da plataforma, servindo como prova de conceito da solução proposta.

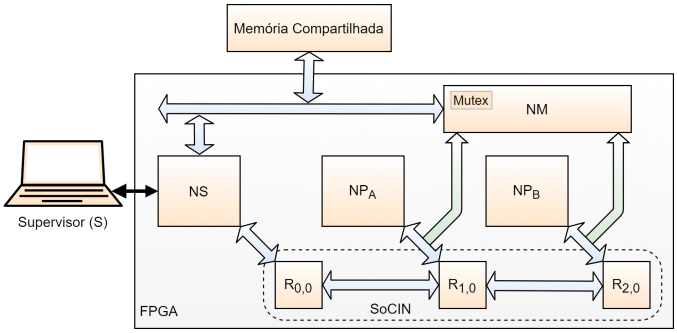

Fig. 1. Arquitetura MPSoC.

Os componentes da plataforma operam da seguinte forma. O computador Supervisor (um computador IBM-PC compatível) provê a interface com o usuário para configurar a plataforma e visualizar os resultados dos experimentos. No início de um

experimento, ele envia um arquivo de configuração ao NS. Esse nodo é responsável por configurar os NPs, controlar a execução do experimento, recuperar e consolidar os dados coletados pelo NM e, por fim, enviar esses dados ao computador Supervisor para cálculo dos indicadores de desempenho. Os NPs são responsáveis pela geração e injeção dos pacotes na SoCIN, bem como pelo recebimento de pacotes transferidos pela rede. Além disso, cada NP deve informar o NS quando concluir a sua injeção de pacotes. O NM é conectado aos terminais da rede e monitora os pacotes transferidos por esses terminais. Ele extrai as informações necessárias ao cálculo dos indicadores de desempenho e as armazena na memória compartilhada com o NS.

Um módulo *mutex* em hardware, integrado ao NM, provê a exclusão mútua no acesso a essa memória. O NM e o NS realizam uma leitura a um registrador do Mutex para obter o direito de acesso à memória compartilhada. Se esse acesso já tiver sido concedido à outra parte, o nodo permanecerá em espera ocupada, realizando um *polling* nesse registrador, até que o recurso compartilhado seja liberado. Pela forma como são programados, o NM é o primeiro a acessar o Mutex e obter o direito de acesso à memória. O NS acessa o Mutex mais tarde pois antes precisa configurar todos os NPs. Portanto, durante a execução do experimento, o acesso à memória é concedido ao NM. Quando esse nodo conclui o armazenamento dos dados coletados da rede, a concessão de acesso é transferida ao NS. O NS então lê a memória compartilhada e envia os dados nela armazenados ao computador Supervisor para que efetue a análise dos indicadores de desempenho. As subseções a seguir apresentam detalhes adicionais sobre a arquitetura da plataforma.

B. Nodos de Processamento e de Monitoramento

Todos os NPs da plataforma são idênticos e são constituídos de um processador embarcado, memória local, interface de rede e periféricos. A estrutura interna dos NPs é ilustrada na Fig. 2a.

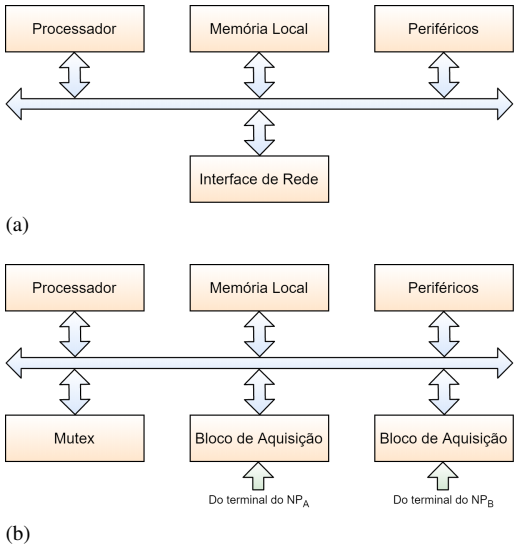

Fig. 2. Organização interna dos nodos: (a) NP; e (b) NM.

O NM possui estrutura similar, mas, ao invés de uma interface de rede, ele possui blocos de aquisição de medidas para coletar dados sobre os pacotes transferidos entre os NPs. Para cada NP do sistema, há um bloco de aquisição de medidas que é acoplado aos canais de injeção e de ejeção do terminal da rede ao qual o NP está conectado. Como já citado, o NM também integra o *mutex* para prover exclusão mútua no acesso à memória compartilhada com o NS. A Fig. 2b ilustra a estrutura interna do NM.

C. Camadas de Comunicação

A integração dos nodos à NoC é feita por meio de uma estrutura composta de três camadas de software (API, Transporte e Driver) e uma camada de hardware (a interface de rede ou NI – Network Interface). A API implementa as primitivas de comunicação MPI, a camada Transporte realiza o empacotamento e o desempacotamento de dados e a camada Driver faz a transferência desses pacotes com a interface de rede. A NI, por sua vez, lida com os serviços do nível de enlace, enviando e recebendo dados da rede.

Para o envio de uma mensagem recebida da API, a camada Transporte constrói um pacote contendo um cabeçalho e os dados da mensagem e encaminha esse pacote para a camada Driver. Essa camada recebe o pacote e acrescenta dois bits de enquadramento a cada palavra do pacote, conforme o protocolo da SoCIN definido em [31]. Esses bits são denominados BOP (*begin-of-packet*) e EOP (*end-of-packet*). O BOP sinaliza a palavra inicial do pacote (cabeçalho), enquanto o EOP sinaliza a palavra final do pacote (terminador). Cada palavra do pacote forma um flit (*flow control unit*), que é a menor unidade de dados sobre a qual é realizada a regulação de tráfego nos elaces da rede. A Fig. 3 ilustra a estrutura do pacote da SoCIN. O pacote é composto de um flit de cabeçalho (BOP = 1) seguido dos flits de carga útil (conteúdo da mensagem) e um flit de terminação (EOP = 1) contendo a última palavra da mensagem. Importante destacar que os dois bits de enquadramento são iguais a 0 nos flits da carga útil.

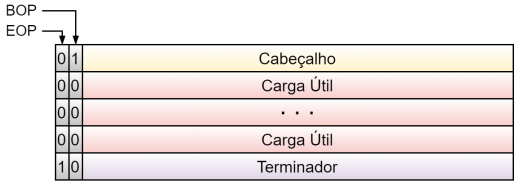

Fig. 3. Formato do pacote.

A arquitetura da NI é composta de duas camadas: (i) *front-end*, que realiza a comunicação direta com o núcleo de processamento por meio do barramento; e (ii) *back-end*, que faz a interface direta com a rede, realizando armazenamento temporário de pacotes em memórias do tipo FIFO (do inglês, First-in,First-out) e a inserção e extração dos bits de enquadramento. Para o envio de um pacote, a camada Driver verifica se a FIFO de saída da NI está disponível. Se estiver, então ela escreve os flits do pacote na FIFO e envia um comando para iniciar a transmissão. Os bits de enquadramento são inseridos na medida em que os flits são injetados na rede. Para o recebimento de um pacote, a camada Driver monitora

um bit de estado da interface de rede. Quando um pacote é completamente recebido na FIFO de entrada, esse pacote é lido pelo *device driver* e entregue à camada Transporte. Esta última, por sua vez, realiza o desempacotamento e a entrega dos dados recebidos à API.

D. Etapas de um Experimento

As configurações dos geradores de tráfego do sistema são repassadas ao Supervisor por meio do $RedScarf$ [14], um ambiente de simulação para avaliação de desempenho de NoCs baseado em modelos SystemC. O $RedScarf$ oferece uma interface gráfica para configuração de experimentos e visualização dos seus resultados e um conjunto de ferramentas dentre as quais se destaca a *gtr*, ferramenta responsável pela geração do arquivo de configuração de tráfego (*traffic.cfg*). Na plataforma desenvolvida, o computador Supervisor utiliza o arquivo *traffic.cfg* para configurar o experimento.

A execução de um experimento na plataforma compreende a execução desta sequência de etapas:

- *Configuração*: Nesta etapa, o Supervisor encaminha uma solicitação ao NS para que esse nodo configure a geração de tráfego na rede. Após receber a solicitação de configuração de tráfego, o NS encaminha para todos os NPs os descritores dos fluxos a serem gerados. Observa-se que é encaminhada uma solicitação para cada NP.
- *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 porque a rede não suporta *multicasting*. Uma vez que os NPs estejam habilitados, é realizada a geração de tráfego na rede pelos NPs e a coleta de informações dos pacotes transferidos, a qual é efetuada pelo NM.
- *Finalização*: Após concluir a geração de tráfego na rede, cada NP encaminha ao NS uma mensagem para informar o término da injeção de todos os pacotes previstos.
- *Leitura dos dados*: Ao término da fase de finalização dos NPs, o NS realiza a leitura da memória compartilhada para recuperar os dados coletados da rede pelo NM. Após o envio dos dados coletados pelo NM, o NS informa o término do experimento ao Supervisor para realização do cálculo das métricas de desempenho.

E. Biblioteca de Comunicação

A comunicação entre os núcleos é realizada via MPI com base na biblioteca ocMPI apresentada em [35]. A biblioteca ocMPI oferece onze funções MPI, além das declarações de constantes e variáveis. A Tabela I apresenta as primitivas da biblioteca ocMPI implementadas neste trabalho.

TABELA I
PRIMITIVAS OCMPI UTILIZADAS NA PLATAFORMA

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*)

IV. IMPLEMENTAÇÃO E VERIFICAÇÃO

A. Materiais

Os componentes de hardware desenvolvidos para a construção da plataforma MPSoC foram implementados utilizando VHDL. Como processador embarcado, foi empregado o *soft-core* Nios®II de 32 bits. Tanto o desenvolvimento como a verificação desses componentes e da própria plataforma foram realizados com o uso de ferramentas da Intel®FPGA, incluindo: Quartus®Prime Edition de 64 bits, Nios®II Software Build Tools (SBT) for Eclipse e ModelSim-Intel®FPGA Starter Edition. Para verificação física, foi utilizado o kit de desenvolvimento DE1-SoC da Terasic Inc. que contém um FPGA da família Cyclone®V da Intel®FPGA. As próximas subseções descrevem os métodos empregados na implementação e na verificação da plataforma.

B. Integração do Núcleo com a NoC

A primeira etapa de desenvolvimento da plataforma consistiu na implementação e verificação da comunicação entre o NP e a interface de rede. Esse sistema foi formado por um nodo de processamento e uma interface de rede em *loop-back*, como ilustra a Fig. 4a. A interface de rede é estruturada em dois módulos: transmissor (TX) e receptor (RX), sendo que cada módulo possui uma FIFO baseada em um *buffer* circular [7]. Essas FIFOs possuem profundidade parametrizável, mas, neste trabalho, foram adotadas FIFOs com capacidade para armazenar um pacote inteiro, cujo tamanho da carga útil é limitado a 8 flits. No experimento de verificação realizado, o tempo decorrido desde o início do processo de transmissão até o envio ao enlace em *loop-back* foi de 1.847 ciclos (ou seja, 18,47 µs para um relógio operando a 100 MHz). Já o processo de recepção, desde a chegada dos dados pelo enlace até a leitura da FIFO de entrada pela aplicação foi de 3.045 ciclos (ou seja 30,45 µs). Essa latência elevada é especialmente devida às camadas executadas em software.

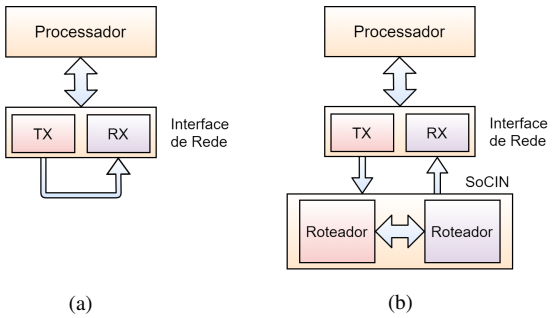

Fig. 4. Testbench para avaliação da transmissão e recepção de pacotes pela interface de rede.

A etapa seguinte buscou verificar a integração das interfaces de rede com a SoCIN. Foi construído o *testbench* representado na Fig. 4b, o qual é composto de apenas duas NIs e uma

SoCIN 2×1, dado o foco de somente verificar a integração da comunicação no nível da rede.

A Fig. 5 apresenta o diagrama de formas de ondas de simulação para o processo de transmissão e recepção na SoCIN. Foram inseridos dois marcadores no diagrama para identificar o envio e a entrega de um pacote pela rede. O marcador A indica quando o sinal o_OUT_VAL (validação de saída) é ativado pela interface de rede, sinalizando o envio do pacote para a SoCIN. O marcador B indica quando o sinal i_IN_VAL (validação de entrada) é ativado pela SoCIN, sinalizando o envio do pacote da rede para a NI.

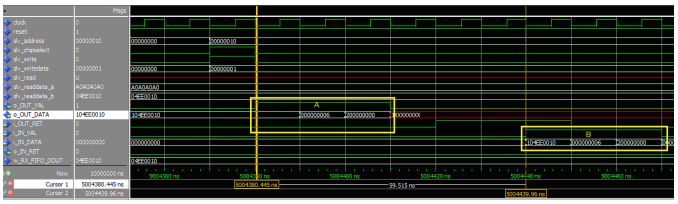

Fig. 5. Diagrama de formas de ondas da verificação da integração entre a interface de rede e a SoCIN.

A latência da rede é dada por:

$$t_{NoC} = 3N + L - 1 \quad (1)$$

onde N é o número de roteadores entre os nodos emissor e receptor e L é o comprimento total do pacote (número de flits), considerando o cabeçalho e o terminador. Na ausência de contenção, o cabeçalho gasta três ciclos para atravessar cada roteador. Essa latência inclui o tempo para ser armazenado na FIFO de entrada do roteador, requisitar uma porta de saída e ter essa requisição avaliada e atendida pelo árbitro da porta de saída solicitada.

No experimento de verificação realizado, o tempo total gasto com a transferência do cabeçalho do pacote foi de 6 ciclos (ou seja, 60 ns para um frequência de relógio de 100 MHz). Já a transferência do restante do pacote consumiu 1 ciclo de relógio ($i.e.$ 10 ns) para cada palavra do corpo do pacote. Comparando esse resultado com as latências apresentadas anteriormente, nota-se que a latência de comunicação da rede é praticamente desprezível.

C. Adaptação da Camada de Software

Na terceira etapa, foi realizado o porte da biblioteca ocMPI para o protocolo de comunicação da SoCIN. Para isso, foram realizadas alterações nos códigos da biblioteca ocMPI e das camadas de Transporte e Driver do código original [35]. Como a SoCIN utiliza um sistema de coordenadas diferente da rede alvo da implementação original da ocMPI, foi necessário ajustar as funções de mapeamento de endereços de sistemas para endereços de rede compatíveis com a SoCIN. Além disso, o tamanho da mensagem enviada para a interface de rede, antes limitado a 8 flits, foi alterado para que mensagens maiores sejam quebradas em pacotes de até 8 flits cada (uma restrição da biblioteca ocMPI).

D. Construção e Integração da Plataforma

A quarta etapa consistiu na integração dos núcleos à rede. Primeiramente, foi realizada a integração do Supervisor com o NS e a SoCIN. Conforme já citado, o arquivo de configuração dos geradores de tráfego (*traffic.tcf*) é obtido a partir do $RedScarf$ e enviado pelo Supervisor ao NS por meio de uma interface serial RS-232. Para isso, foi implementada uma ferramenta de comunicação serial (Serial_App) que permite: (i) estabelecer a conexão serial com o nodo NS; (ii) enviar o arquivo de configuração ao NS no momento desejado; e (iii) registrar o intervalo de tempo entre o envio do pacote de configuração ao NS e o retorno dos dados pelo NS.

Após receber o arquivo de configuração, o NS extrai as informações necessárias para a montagem da mensagem de configuração de cada NP. A Fig. 6 apresenta a mensagem de configuração para um NP. Os primeiros cinco flits incluem informações do protocolo de comunicação ocMPI. No sexto flit, o campo *Number of Flows* informa a quantidade de fluxos que o NP deve gerar.

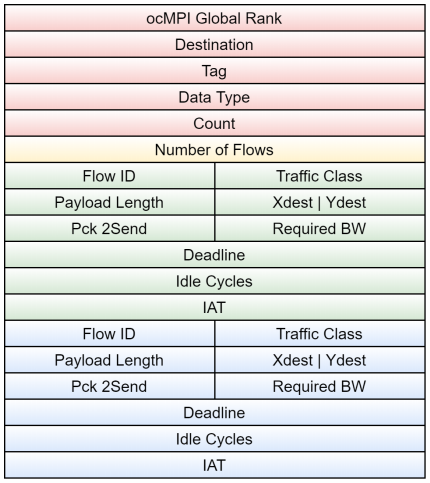

Fig. 6. Mensagem de configuração para um NP.

Os próximos campos da mensagem de configuração do NP contém a especificação de cada um dos fluxos a serem gerados pelo nodo (a mensagem do exemplo descreve dois fluxos). Esses campos são extraídos do arquivo de configuração gerado pelo $RedScarf$ e contém as seguintes informações (mais detalhes em [14]):

- *Flow ID*: Identificador do fluxo.
- *Traffic Class*: Tipo de tráfego.
- *Payload Length*: Quantidade de flits por pacote.
- *Xdest, Ydest*: Endereço de rede do destinatário.
- *Pck2Send*: Quantidade de pacotes para injetar na rede.
- *Required BW*: Largura de banda requerida.
- *Deadline*: Latência limite para entrega do pacote.
- *Idle Cycles*: Intervalo de tempo entre pacotes.
- *IAT*: Período de criação de pacotes (Inter-arrival Time).

E. Comunicação entre os NPs

O *testbench* para verificação da comunicação entre os NPs foi composto de dois NPs e duas NIs integrados a uma rede 2×1. Esse experimento também serviu para verificar as

alterações realizadas nas camadas de software. O NP_A enviou um pacote pela SoCIN ao NP_B, o qual extraiu as informações e respondeu corretamente à solicitação gerada pelo NP_A.

Foi desenvolvido um aplicativo que realizou um *ping* entre os NPs. O pacote utilizado nessa comunicação contém um flit de cabeçalho, cinco flits de configuração do protocolo ocMPI e um flit de dado. Um componente denominado Performance Counter foi acrescentado à arquitetura do nodo de processamento NP_A para mensurar o sobrecusto da aplicação. O Performance Counter é um contador em hardware que mede o tempo de execução do código escolhido.

A Fig. 7 apresenta os tempos de execução medidos para as funções $ocMPI\_Send$ e $ocMPI\_Receiver$. Observa-se que a função $ocMPI\_Send$ gastou 47 µs para montar todo o pacote e enviá-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 Fig. 8. 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 gasta apenas 0,06 µs para transferir o cabeçalho do pacote e mais 0,01 µs para cada flit da carga útil.

```
--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 |
+---------------+-----+------------+---------------+------------+
```

Fig. 7. Relatório de desempenho do console Nios®II Build Tools for Eclipse.

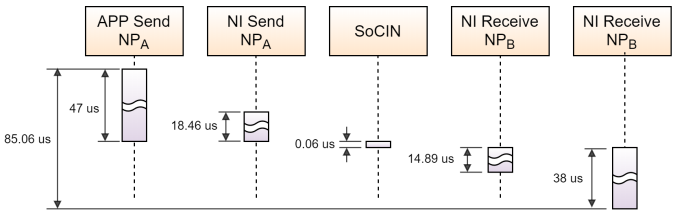

Fig. 8. Diagrama de sequência do sistema de verificação da integração entre NPs e a SoCIN.

Nota-se, portanto, que a biblioteca ocMPI acrescenta um alto sobrecusto à latência de comunicação, o qual é devido às rotinas da API, da camada Transporte e do Driver. Esse sobrecusto, porém, se justifica por oferecer flexibilidade para a geração de tráfego e facilitar o desenvolvimento de aplicações.

F. Comunicação entre Supervisor, NS e NPs

A Fig. 9 ilustra o sistema utilizado para verificar a comunicação entre Supervisor, NS e NPs, assim como o envio dos pacotes de configuração do gerador de tráfego do NS para o NPs via SoCIN. O NS é posicionado no roteador de endereço (0,0) e os nodos de processamento nos roteadores de endereço (1,0) e (2,0).

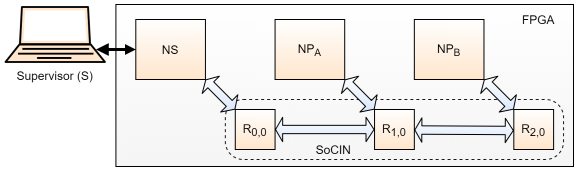

Fig. 9. Arquitetura de validação da comunicação.

No experimento realizado, cada um dos NPs recebeu do NS os parâmetros para injetar um fluxo de 40 pacotes com 8 flits (cabeçalho seguido de 224 bits de dados) 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. Os resultados obtidos demonstraram que o sistema executou corretamente as etapas de configuração e habilitação do experimento.

G. Coleta de Dados Pelo NM

Para verificar se o NM coleta corretamente os dados transferidos nos terminais da rede e, também, o acesso à memória compartilhada entre o NS e o NM, foi utilizado o sistema ilustrado inicialmente na Fig. 1. O NM monitora os pacotes injetados e ejetados na rede. Para os pacotes injetados, ele registra o cabeçalho do pacote e o ciclo em que este cabeçalho é entregue à rede. Para os pacotes ejetados, ele registra o cabeçalho e o ciclo em que o terminador do pacote é entregue ao NP destinatário.

A Fig. 10 apresenta um exemplo com informações extraídas durante o experimento de verificação. A moldura A apresenta os ciclos de injeção (1.964.255.193) e ejeção (1.964.255.207) do pacote com cabeçalho igual a `0x1CC12010`. Observa-se que a latência desse pacote foi de 14 ciclos de relógio (*i.e.* $207 - 193$). Desse total, 6 ciclos são gastos para transferência do cabeçalho pelos roteadores da rede, 1 ciclo para recebimento do cabeçalho na interface de rede e 7 ciclos para recebimento do restante do pacote (1 ciclo por flit). Como não há concorrência com outros fluxos, todos os demais pacotes experimentaram a mesma latência, como, por exemplo, o pacote identificado pela moldura B.

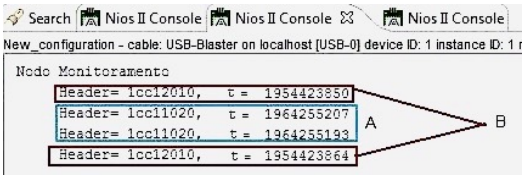

Fig. 10. Tempos extraídos pelo NM observados por meio do console do Nios®II Build Tools for Eclipse.

Dois aspectos devem ser destacados em relação à coleta de dados pelo NM. Primeiro, todos os nodos compartilham o mesmo sinal de relógio e os carimbos de tempo (ou *timestamps*) são criados apenas pelo NM. Segundo, o NM não atua na regulação do fluxo de dados nos enlaces. Isso é realizado pelos NPs, o que justifica o uso desses nodos como destinatários para retirar os pacotes da rede. Cabe também ressaltar que, em um experimento de avaliação de desempenho, o NP destinatário não processa as informações contidas

nos pacotes recebidos. Contudo, a plataforma proposta foi projetada para também permitir o desenvolvimento de uma aplicação MPSoC real. Nesse caso, cada pacote recebido por um NP conterá uma informação útil à aplicação e que será tratada pela tarefa em execução no processador.

H. Comunicação entre NM e NS

A Fig. 11 ilustra o processo de depuração da comunicação entre os nodos NM e NS e mostra um sequência de operações de escrita do NM na memória compartilhada. No exemplo, as primeiras informações destacadas se referem ao cabeçalho de um pacote (`0x1cc12010`) e o carimbo de tempo que indica o ciclo da passagem desse cabeçalho pelo canal de injeção de um terminal da rede (*i.e.* 1.954.423.850). As duas últimas informações destacadas se referem à passagem do terminador desse mesmo pacote pelo canal de ejeção de outro terminal no ciclo 1.954.423.864. Após o NS ler esses dados da memória e o encaminhar ao Supervisor, caberá ao software do Supervisor identificar corretamente, para cada pacote, quais informações se referem ao tempos de injeção e ejeção do pacote, dado que este último será sempre maior que o primeiro. No exemplo ilustrado, observa-se que a latência para transferência do referido pacote é igual a 14 ciclos.

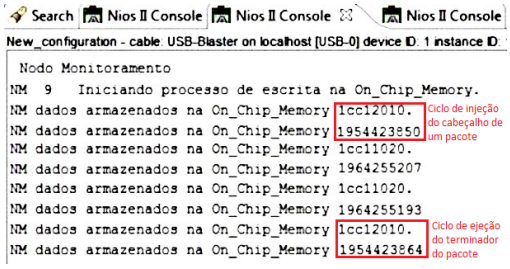

Fig. 11. Dados coletados pelo NM observados por meio do console do Nios®II Build Tools for Eclipse.

V. Resultados

A. Custo de Silício

O MPSoC foi prototipado no kit de desenvolvimento DE1-SoC da Terasic Inc., o qual utiliza o dispositivo FPGA 5CSEMA5F31C6N SoC da família Cyclone®V da Intel®FPGA. Esse componente possui 32.070 ALMs (Adaptive Logic Module), 85.000 LEs (Logic Elements) e 3.972 Kbits de RAM [36]. Cada ALM é composto de uma LUT (Look-Up Table) fracionável com oito entradas e quatro FFs (Flip-Flops). Na plataforma implementada, o NM utiliza memória externa para armazenar os dados coletados dos pacotes injetados na rede. A plataforma foi sintetizada com o uso da ferramenta Quartus®Prime Edition.

A Tabela II reporta os recursos ocupados do FPGA para os principais componentes do sistema, considerando uma configuração de rede 3×1. A linha Sistema agrupa os custos de diversos componentes necessários à integração e depuração do MPSoC. Os resultados mostram que o NM é o nodo com maior custo devido aos blocos de aquisição. Os demais nodos tem custos lógicos similares. O NS requer uma quantidade maior de memória para armazenar o código da biblioteca de comunicação com o computador Supervisor. A tabela também apresenta as taxas de ocupação que evidenciam que o sistema implementado ocupa 36% dos ALMs e 47% da capacidade de memória do dispositivo. Portanto, esse dispositivo possui recursos disponíveis para integrar um sistema maior do que o empregado nos experimentos realizados neste trabalho.

TABELA II
Recursos Lógicos Ocupados do FPGA

Componente	ALMs	ALUTs	FFs	RAM (Kbits)
Sistema	1.357,5	2.158	1.758	513
NM	4.357,6	4.583	8.361	75
NP_A	1.258,5	1.596	2.236	139
NP_B	1.249,4	1.584	2.228	139
NS	1.295,0	1.644	2.397	1.011
NoC	1.917,7	1.795	4.095	0
Total	**11.435,7**	**13.360**	**21.075**	**1.877**
Taxa de ocupação	**36%**	**16%**	**25%**	**47%**

Ainda sobre os custos de silício, as Tabelas III e IV apresentam os recursos ocupados pelos processadores Nios®II e pelas interfaces de rede dos nodos do sistema.

TABELA III
Recursos Ocupados pelos Processadores Nios®II

Processador	ALMs	ALUTs	FFs
NM-NIOS-II	482,0	711	679
NP_A-Nios-II	456,0	660	664
NP_B-Nios-II	460,5	655	661
NS-Nios-II	475,8	712	702
Média	**468.6**	**684.5**	**676.5**

TABELA IV
Recursos Ocupados pelas Interfaces de Rede

Componente	ALMs	ALUTs	FFs
NP_A-NI	508,6	454	1.107
NP_B-NI	493,6	451	1.105
NS-NI	468,7	449	1.105
Média	**490,3**	**451,3**	**1.105,7**

B. Avaliação de Desempenho

Uma métrica para avaliação do desempenho da plataforma proposta é o tempo para execução de um experimento no FPGA, o qual 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 NP é configurado para enviar 40 pacotes de 8 flits para o outro NP. O intervalo de tempo entre dois pacotes sucessivos é limitado pelo tempo gasto pela camada de software para construir o pacote e injetá-lo na rede, ou seja, 47 µs. Com isso, a taxa de injeção de dados máxima é de 4,76 Mbps (224 bits de dados / 47 µs).

Para realizar um experimento, é necessário que o Supervisor envie o arquivo de configuração ao NS pela conexão serial RS-232 com uma taxa de dados de 115,2 Kbps para que o NS configure cada NP. Após, os NPs enviam pacotes um para o outro, enquanto o NM coleta e armazena informações sobre os pacotes na memória compartilhada. Essa memória é lida pelo NS ao final do experimento para envio dos dados ao Supervisor. Considerando todas essas etapas, a execução deste experimento consumiu 1,1 s.

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 × 47 µs por pacote (não está contabilizada a latência da rede por ser muito menor que a latência das camadas de software). Como os fluxos enviados pelos dois NPs não concorrem um com o outro pelos canais da rede, pois estão em oposição, os dois fluxos de pacotes são iniciados e finalizados quase ao mesmo tempo; deve-se lembrar que a habilitação da injeção de pacotes pelos NPs é realizada sequencialmente pelo NS, conforme descrito na Subseção III-D. Dessa forma, cerca de 99,82% do tempo de execução deve-se às etapas inicial (configuração do experimento) e final (envio dos dados), e apenas 0,17% ao experimento em si. Para amortizar o custo da configuração e coleta é necessário simular um volume muito maior de pacotes. Além disso, é possível reduzir esse custo por meio do uso de uma interface de comunicação serial com maior taxa de dados. Esses aspectos serão tratados em um trabalho futuro.

Outra métrica utilizada para avaliar plataformas de emulação em FPGA é a redução de tempo para realizar o experimento em comparação com ambientes de simulação. Essa redução resulta em uma aceleração do processo da avaliação. Para identificar essa aceleração, 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 reproduzir 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 / 47 µs para injeção de um pacote, para um relógio de 100 MHz). O experimento foi executado em dois $laptops$ com processadores baseados em duas litografias diferentes daquela utilizada no FPGA. Os resultados obtidos são apresentados na Tabela V. Em relação ao primeiro computador, a aceleração foi de 9,1 vezes, o que está na ordem de grandeza dos resultados obtidos em [19]. Já o tempo de simulação no segundo computador foi inferior ao tempo de emulação no FPGA. Destaca-se que este computador utiliza um processador com uma litografia menor que a do FPGA e opera a uma frequência ainda maior que aquela do primeiro computador.

TABELA V
TEMPO DE EXECUÇÃO

Processador	Frequência	Litografia	Ano	$T_{execução}$
Intel®Core™2 Duo	2.1 GHz	45 nm	2008	10.0 s
Intel®Core™i7 7th Gen.	2.7 GHz	14 nm	2017	0.6 s
Cyclone®V	100 MHz	28 nm	2012	1.1 s

Alguns aspectos e limitações devem ser observados em relação aos resultados acima. Em primeiro lugar, a plataforma MPSoC utilizada possui a configuração mínima necessária para validação da arquitetura proposta e contém apenas dois nodos de processamento. Quanto maior o sistema simulado, maior será o tempo de simulação no $laptop$ pois toda a infraestrutura do sistema (geradores de tráfego, rede e medidores de tráfego) é executada sequencialmente. Já a emulação no FPGA irá se beneficiar do paralelismo inerente ao hardware. Segundo, o volume de tráfego utilizado no experimento não é suficiente amortizar o alto custo das etapas de configuração e coleta de dados. Se as etapas de configuração e coleta de dados fossem desconsideradas, seria obtida uma aceleração da ordem de 5.319 vezes (10 s / 1,88 ms) no primeiro computador e de 319 vezes (600 ms / 1,88 ms) no segundo $laptop$. Logo, é necessário reduzir o custo da comunicação pela troca do padrão RS-232 por outro com maior taxa de dados ($e.g.$ USB).

VI. CONCLUSÕES

A plataforma desenvolvida oferece flexibilidade para geração de tráfego e traz benefícios para o desenvolvimento de aplicações com programação paralela por meio do uso da biblioteca ocMPI. Porém, como visto nos resultados, a flexibilidade obtida traz como consequência um sobrecusto, pois as camadas de comunicação nos nodos de processamento produzem uma latência muito maior que aquela devida à rede. Em vista disso, como trabalho futuro, pretende-se explorar o espaço de projeto para reduzir o custo da comunicação, transferindo parte das funcionalidades implementadas em software para o nível de hardware.

Os resultados obtidos mostram que a rede é capaz de atender demanda dos nodos de processamento no cenário avaliado. No entanto, considera-se necessário integrar um maior número de núcleos e executar cenários intensivos com fluxos de comunicação concorrentes ($e.g.$ tráfego uniforme) para melhor avaliar a capacidade da rede em atender a demanda de comunicação de um MPSoC. Essa avaliação também será realizada em um próximo trabalho.

Outro aspecto observado a partir dos experimentos realizados reside na necessidade de se reduzir a latência da transferência de dados entre o computador Supervisor e o Nodo Supervisor por meio de um canal de comunicação com maior taxa de dados. Isso se faz necessário para que a plataforma traga reais vantagens em relação à simulação em software e será alvo de uma próxima atividade de desenvolvimento.

Por fim, pretende-se atualizar a plataforma, substituindo o Nios®II pelo processador RISC-V, o qual tem se tornado um padrão *de facto* por se tratar de uma arquitetura de conjunto de instruções aberta e desenvolvida de forma colaborativa.

AGRADECIMENTOS

Os autores agradecem o apoio recebido do Programa UNIEDU do Governo do Estado de Santa Catarina e do Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (Processos No. 315287/2018-7 e 436982/2018-8). Os autores também agradecem ao Prof. Eduard Fernandez Alonso da Universitat Ramon Llull – Espanha pela disponibilização do código fonte original da biblioteca ocMPI.

REFERÊNCIAS

[1] L. Benini and G. De Micheli, "Networks on chip: A new paradigm for systems on chip design," in *Proc. of Design, Automation and Test in Europe Conf. and Exhibition*, 2002, pp. 418–419.

[2] S. Swapna, A. K. Swain, and K. K. Mahapatra, "Design and analysis of five port router for network on chip," in *Asia Pacific Conf. on Postgraduate Research in Microelectronics and Electronics*, 2012, pp. 51–55.

[3] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proc. of Design Automation Conf.*, 2004, pp. 681–685.

[4] C. A. Zeferino, "Redes-em-chip: arquiteturas e modelos para avaliação de área e desempenho," Ph.D. dissertation, Universidade Federal do Rio Grande do Sul, 2003.

[5] P. Marwedel, "Processor-core based design and test," in *Proc. of Asia and South Pacific Design Automation Conf.*, 1997, pp. 499–502.

[6] R. K. Gupta and Y. Zorian, "Introducing core-based system design," *IEEE Des. Test. Comput.*, vol. 14, no. 4, pp. 15–25, 1997.

[7] F. Vahid, *Digital design*. Wiley Hoboken, 2007.

[8] T. Dorta, J. Jiménez, J. L. Martín, U. Bidarte, and A. Astarloa, "Reconfigurable multiprocessor systems: a review," *Int. J. Reconfigurable Comput.*, vol. 2010, pp. 1–10, 2010.

[9] M. Ramirez, M. Daneshtalab, J. Plosila, and P. Liljeberg, "NoC-AXI interface for FPGA-based MPSoC platforms," in *Proc. of on Field Programmable Logic and Applications*, 2012, pp. 479–480.

[10] D. Bouthaina, M. Baklouti, S. Niar, and M. Abid, "Shared hardware accelerator architectures for heterogeneous mpsocs," in *Proc. of Int. Wksp. on Reconfigurable and Communication-Centric Systems-on-Chip*, 2013, pp. 1–6.

[11] G. Führ, S. H. Hamurcu, D. Pala, T. Grass, R. Leupers, G. Ascheid, and J. F. Eusse, "Automatic energy-minimized hw/sw partitioning for fpga-accelerated mpsocs," *IEEE Embedded Syst. Lett.*, vol. 11, no. 3, pp. 93–96, 2019.

[12] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and L. Pieralisi, *Design of cost-efficient interconnect processing units: Spidergon STNoC*. CRC press, 2018.

[13] F. Gebali, H. Elmiligi, and M. W. El-Kharashi, *Networks-on-chips: theory and practice*. CRC press, 2011.

[14] E. Silva, M. Kreutz, and C. Zeferino, "RedScarf: an open-source multi-platform simulation environment for performance evaluation of Networks-on-Chip," *J. Syst. Archit.*, vol. 99, 2019.

[15] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, and F. Catthoor, "A complete Network-on-Chip emulation framework," in *Proc. of Design, Automation and Test in Europe*, 2005, pp. 246–251.

[16] P. T. Wolkotte, P. K. Holzenspies, and G. J. Smit, "Fast, accurate and detailed NoC simulations," in *Proc. of Int. Symp. on Networks-on-Chip*, 2007, pp. 323–332.

[17] T. F. Pereira and C. A. Zeferino, "A set of VHDL IPs to evaluate performance of Networks-on-Chip," in *Proc. of IP Based Electronic Conf. and Exhibition*, 2008, pp. 239–243.

[18] G. Luo-Feng, D. Gao-ming, Z. Duo-Li, G. Ming-Lun, H. Ning, and S. Yu-Kun, "Design and performance evaluation of a 2D-mesh Network-on-Chip prototype using FPGA," in *Proc. of IEEE Asia Pacific Conf. on Circuits and Systems*, 2008, pp. 1264–1267.

[19] E. Fernandez-Alonso, D. Castells-Rufas, S. Risueño, J. Carrabina, and J. Joven, "A NoC-based multi-{soft} core with 16 cores," in *Proc. of IEEE Int. Conf. on Electronics, Circuits and Systems*, 2010, pp. 259–262.

[20] S. Lotlikar, V. Pai, and P. V. Gratz, "AcENoCs: A configurable HW/SW platform for FPGA accelerated NoC emulation," in *Proc. of Int. Conf. on VLSI Design*, 2011, pp. 147–152.

[21] G. Heck, R. Guazzelli, F. Moraes, N. Calazans, and R. Soares, "HardNoC: A platform to validate Networks-on-Chip through FPGA prototyping," in *Proc. of Southern Conf. on Programmable Logic*, 2012, pp. 1–6.

[22] D. Hartono, S. W. Lee, V. V. Yap, M. S. Ng, and C. M. Tang, "A multicore system using NoC communication for parallel coarse-grain data processing," in *Proc. of Conf. on New Media Studies*, 2013, pp. 1–5.

[23] J. A. Ambrose, T. Li, D. Murphy, S. Gargg, N. Higgins, and S. Parameswaran, "Argus: A framework for rapid design and prototype of heterogeneous multicore systems in FPGA," in *Proc. of Int. Conf. on VLSI Design*, 2015, pp. 29–34.

[24] J. Rettkowski, M. Eldafrawy, D. Göhringer *et al.*, "An event-based network-on-chip debugging system for FPGA-based MPSoCs," in *Proc. of Int. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2017, pp. 30–37.

[25] R. Salem, Y. Salah, I. Bennour, and M. Atri, "FPGA prototyping and design evaluation of a NoC-based MPSoC," *Int. J. of Advanced Computer Science and Applications*, vol. 8, no. 11, pp. 311–319, 2017.

[26] J. W. Tang, Y. W. Hau, N. Shaikh-Husin, and M. N. Marsono, "Application profiling and mapping on NoC-based MPSoC emulation platform on reconfigurable logic," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 15, no. 3, pp. 1040–1047, 2017.

[27] M. Khamis, S. El-Ashry, A. Shalaby, M. AbdElsalam, and M. W. El-Kharashi, "A configurable RISC-V for NoC-based MPSoCs: A framework for hardware emulation," in *Proc. of Int. Wksp. on Network on Chip Architectures*, 2018, pp. 1–6.

[28] J. Rettkowski and D. Goehringer, "High-level synthesis of software-defined MPSoCs," in *Proc. of Int. Symp. on Applied Reconfigurable Comput.*, 2018, pp. 407–419.

[29] A. Monemi, J. W. Tang, M. Palesi, and M. N. Marsono, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors and Microsyst.*, vol. 54, pp. 60–74, 2017.

[30] M. S. Mohammed, J. W. Tang, A. A.-H. Ab Rahman, N. Paraman, and M. Marsono, "Rapid prototyping of NoC-based MPSoC based on dataflow modeling of real-world applications," in *Proc. of IEEE Control and System Graduate Research Colloq.*, 2018, pp. 217–222.

[31] C. A. Zeferino and A. A. Susin, "SoCIN: a parametric and scalable Network-on-Chip," in *Proc. of Symp. on Integrated Circuits and Systems Design*, 2003, pp. 169–174.

[32] F. Vahid and T. Givargis, *Embedded system design: A unified hardware/software approach*. John Wiley & Sons, 2001.

[33] L. Frantz and C. A. Zeferino, "Gerador de tráfego para Redes-em-Chip baseado no PicoBlaze," in *Proc. of Conf. Ibérica de Sistemas e Tecnologias de Informação*, 2009, pp. 1–8.

[34] R. S. Uhlendorf, "MPSoC para avaliação de desempenho de uma Rede-em-Chip em FPGA," Master's thesis, Universidade do Vale do Itajaí, 2017.

[35] E. Fernandez-Alonso, J. Carrabina, D. Castells-Rufas, and J. Joven, "Embedding MPI in many-soft-core processors," in *Proc. of HIP3ES Wksp.*, 2013, pp. 1–5.

[36] Altera, *Cyclone V Device Handbook – Volume 1: Device Overview and Datasheet*, 11th ed., Altera, San Jose, CA, 2012.

Roseli Uhlendorf recebeu seu título de Mestre em Computação Aplicada pela Universidade do Vale do Itajaí, Brasil, em 2020. Atua como consultora na área de projeto de sistemas digitais em FPGA. Suas áreas de interesse incluem: Sistemas Eletrônicos, Projeto de Sistemas e Networks-on-Chip.

Eduardo Silva recebeu seu título de Mestre em Computação Aplicada pela Universidade do Vale do Itajaí, Brasil, em 2017. É Professor Assistente da Escola do Mar, Ciência e Tecnologia da Univali, Brasil, desde 2017, coordenador dos cursos de Bacharelado em Ciência da Computação e de Tecnologia em Sistemas para Internet e pesquisador do Laboratory of Embedded and Distributed Systems da Univali. Suas áreas de interesse incluem: Simulação, Networks-on-Chip e Sistemas Embarcados.

Felipe Viel recebeu seu título de Mestre em Computação Aplicada pela Universidade do Vale do Itajaí, Brasil, em 2019. É Professor Assistente da Escola do Mar, Ciência e Tecnologia da Univali, Brasil, desde 2019, e pesquisador do Laboratory of Embedded and Distributed Systems da Univali. Suas áreas de interesse incluem: Sistemas Embarcados, Sistemas Reconfiguráveis, Aceleradores em Hardware, Processamento Digital de Imagens, Aviônica e Sistemas de Alta Confiabilidade.

Cesar Zeferino recebeu seu título de Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul, Brasil, em 2003, com estágio na Université Paris-Sorbonne, França. É professor titular da Escola do Mar, Ciência e Tecnologia da Univali, Brasil, desde 2002, Gerente de Pesquisa e Pós-Graduação da Univali e líder do Laboratory of Embedded and Distributed Systems da Univali. Seus tópicos de interesse são Aceleradores de Hardware, Internet das Coisas e Networks-on-Chip.