PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

TADEU DE SOUSA MARCHESE

INTERFACES DE REDE SEGURAS PARA CONEXÃO DE PERIFÉRICOS EM MPSOCS

Porto Alegre
2022

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

INTERFACES DE REDE SEGURAS PARA CONEXÃO DE PERIFÉRICOS EM MPSOCS

TADEU DE SOUSA MARCHESE

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre
2022

Ficha Catalográfica

M316i Marchese, Tadeu de Sousa

Interfaces de Rede Seguras para Conexão de Periféricos em MPSoCs / Tadeu de Sousa Marchese. – 2022.
63.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Interface de rede. 2. Rede intra-chip. 3. Segurança. 4. Mecanismos de comunicação de entrada e saída. I. Moraes, Fernando Gehm. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

TADEU DE SOUSA MARCHESE

INTERFACES DE REDE SEGURAS PARA CONEXÃO DE PERIFÉRICOS EM MPSOCS

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado(a) em 13 de Janeiro de 2022.

BANCA EXAMINADORA:

Prof. Dr. Rafael Fraga Garibotti (PPGEE/PUCRS)

Prof. Dr. César Augusto Missio Marcon (PPGCC/PUCRS)

Prof. Dr. Fernando Gehm Moraes (PPGCC/PUCRS - Orientador)

INTERFACES DE REDE SEGURAS PARA CONEXÃO DE PERIFÉRICOS EM MPSOCS

RESUMO

O aumento da complexidade em projetos de circuitos integrados (CIs) devido à quantidade de módulos de hardware pré-validados (em inglês, *Intellectual Property* - IPs) integrados em um mesmo *System-on-Chip* (SoC), motiva o desenvolvimento de arquiteturas de comunicação escaláveis, como *Network-on-Chips* (NoCs). Este modelo traz conceitos de redes de computadores, aplicações paralelas e distribuídas para o projeto de CIs. Módulos IPs são utilizados como forma de reduzir o custo de engenharia e tempo de projeto, sendo sua reutilização facilitada pela adoção de interfaces e protocolos padronizados. A possibilidade de executar aplicações de origens e perfis diferentes paralelamente em um *Multiprocessor-System-on-Chip* (MPSoC) torna-o suscetível a ataques de aplicações maliciosas, que podem se aproveitar do compartilhamento de recursos provido pela plataforma para extrair informações sensíveis ou impedir o seu funcionamento. Este trabalho revisa o estado-da-arte em interfaces de rede seguras para NoCs, considerando um modelo de ameaças com os seguintes princípios de segurança: autenticação, integridade, não repúdio, confidencialidade, disponibilidade, autorização. O trabalho propõe uma interface de rede segura para MPSoC com o objetivo de abstrair o seu protocolo interno aos periféricos externos, permitindo a integração de IPs de terceiros como aceleradores de hardware, além de prover um controle de acesso aos periféricos comandado por parte do MPSoC, e também garantir a integridade e confidencialidade das mensagens provenientes de tarefas executando no MPSoC (em trânsito) durante a comunicação com periféricos externos e durante o armazenamento em memórias externas conectadas ao MPSoC. Como prova de conceito foi desenvolvida uma memória externa com interface e protocolo padronizado, integrada no MPSoC base Memphis juntamente com uma *Application-Programming-Interface* (API) desenvolvida no nível de kernel. A API implementa um esquema de alocação de recursos que permite controlar o acesso aos periféricos, assim como impedir a injeção na rede de pacotes forjados. Mostramos através dos experimentos que é possível mitigar ataques oriundos da execução de tarefas maliciosas e IPs de terceiros por meio da adoção de uma interface de rede segura que implementa mecanismos verificação de assinaturas nas mensagens, criptografia de dados, alocação de recursos e controle de acesso.

Palavras-Chave: Interface de rede, Rede intra-chip, Segurança, Mecanismos de comunicação de entrada e saída.

SECURE NETWORK INTERFACES FOR PERIPHERAL CONNECTION IN MPSOCS

ABSTRACT

The increase in complexity in Integrated Circuits (ICs) designs due to the amount of pre-validated hardware modules (Intellectual Property - IPs) integrated into the same System-on-Chip (SoC) motivates the development of scalable communication models such as Networks-on-Chip (NoCs). This model brings concepts of computer networks, parallel and distributed applications to IC projects. IP modules are used to reduce engineering costs and design time, and their reuse is facilitated by the adoption of standardized interfaces and protocols. The possibility of running applications from different sources and profiles in parallel in a Multiprocessor-System-on-Chip (MPSoC) makes it susceptible to attacks from malicious applications, which can use the resources sharing provided by the platform to extract information or prevent its functioning. This work reviews the state-of-the-art in security network interfaces for NoCs, considering a threat model with the following security principles: security, integrity, non-repudiation, confidentiality, availability, authorization. We propose a secure network interface (NI) for MPSoCs to abstract their internal protocol to external peripherals. This NI allows the integration of third-party IPs as hardware accelerators, in addition to providing access control for the peripherals connected to the MPSoC and also ensuring integrity and confidentiality of the messages coming from tasks running on it (in-transit) during the communication with external peripherals and during storage in memories connected to the MPSoC. As proof of concept, an external memory with a standardized interface and protocol was developed, integrated into the Memphis MPSoC, with the API developed at the kernel level. The API implements a resource allocation scheme that allows controlling access to peripherals and prevents the injection of forged packets into the network. We show through the experiments that it is possible to mitigate attacks arising from the execution of malicious tasks and third-party IPs by adopting the secure NI that implements message signature verification, data encryption, resource allocation, and control access mechanisms.

Keywords: Network interface, Intra-chip network, Safety, Input and output communication mechanisms.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

3PIP	Third-Party IPs
AES	Advanced Encryption System
AMBA	Advanced Microcontroller Bus Architecture
AXI	Advanced Extensible Interface
CI	Circuito Integrado
DMA	Direct Memory Access
DMNI	Direct Memory Network Interface
E/S	Entrada e Saída
ECDH	Elliptic Curves Diffie-Hellman
FIFO	First-Input First-Output
GALS	Globally Asynchronous, Locally Synchronous
GAPH	Hardware design support group
GMP	Global Manager Processor
HeMPS	HERMES Multiprocessor System
I/O	Input and Output
IoT	Internet-of-Things
IP	Intellectual Property
LMP	Local Manager Processor
MAC	Message Authentication Code
Memphis	Many-core Modeling Platform for Heterogenous SoCs
MNI	Master Network Interface
MP	Manager Processor
MPSoC	Multiprocessor System-on-Chip
NA	Network Adapter
NI	Network Interface
NoC	Network-on-Chip
OCP	Open Core Protocol
OPB	On-Chip Peripheral Bus
OSI	Open Systems Interconnection
PE	Processing Element
QoS	Quality of Service
RSA	Rivest–Shamir–Adleman
SNI	Slave Network Interface
SoC	System-on-Chip
SP	Slave Processor
GPPC	General Purpose Processing Cores
PRNG	Pseudo Random Number Generator

SUMÁRIO

1 INTRODUÇÃO

A constante evolução tecnológica, que segue a lei de Moore [MOO06], permite que mais transistores sejam integrados em um mesmo circuito, ou então que o mesmo número de transistores seja integrado em um circuito menor. Como forma de reduzir o tempo de projeto, e por consequência reduzir o custo de engenharia, cada vez mais se reutiliza módulos de hardware pré-validados [BJE06], denominados núcleos de propriedade intelectual (IP - *intelectual-property*). A reutilização de IPs em projetos de CI é fortemente facilitada pelo desenvolvimento e adoção de protocolos de comunicação padronizados, sendo hoje fundamental para a indústria.

A forma de comunicação baseada em barramentos não é escalável, pois esta apresenta um aumento na latência de comunicação que cresce proporcionalmente ao número de IPs integrados em um sistema-em-chip (SoC - *system-on-chip*), sendo assim ineficiente em projetos que integram um grande número de IPs [KUM02, GUE00]. Uma outra forma de comunicação é interconectar os IPs diretamente, utilizando um *crossbar*. Porém, esta solução também não é escalável pois conforme o número de IPs cresce, a complexidade das interconexões se torna maior que a complexidade dos componentes lógicos em si. As redes *intra-chip* (NoCs – *networks-on-chip*) surgiram para mitigar o problema de escalabilidade, trazendo conceitos de redes de computadores e aplicações paralelas e distribuídas para o projeto de CIs.

A necessidade de utilização de NoCs aumenta à medida que mais IPs são integrados em um mesmo CI e existe a necessidade de interligá-los para fins de comunicação. Uma classe de CI que utiliza NoCs são os MPSoCs (*Multiprocessor-system-on-chip*). Estes sistemas possuem grande poder de processamento e são capazes de executar em paralelo diversas aplicações de diferentes perfis. Exemplos de aplicações que demonstram a aplicabilidade de arquiteturas MPSoC incluem:

- Processamento digital de sinais: MPEG4 (H264), vídeo 4K, vídeo 8K, renderização de imagem em tempo real, realidade aumentada;
- Reconhecimento de padrões: biometria, reconhecimento facial e de voz, sequenciamento de DNA;
- Sistemas embarcados, Internet das coisas (*IoT – Internet of Things*) e plataformas móveis;
- Aprendizado de máquina;
- Veículos autônomos.

A possibilidade de executar aplicações de origens e perfis diferentes paralelamente, em uma mesma plataforma, torna-a suscetível a ataques de aplicações maliciosas que podem se aproveitar do compartilhamento de recursos provido pela plataforma para extrair informações sensíveis ou impedir o funcionamento de outra aplicação [SAN21]. Caimi et al. utilizam a técnica de zonas seguras para mitigar ataques realizados por tarefas maliciosas [CAI19a]. Zonas seguras isolam a execução de aplicações sensíveis do restante do sistema como forma de evitar ataques de aplicações maliciosas, protegendo-as desde a transmissão da aplicação para os elementos de processamento (PEs, do inglês *processing elements*), até a liberação dos recursos utilizados pela aplicação.

Aplicações, sejam elas sensíveis ou não, necessitam frequentemente realizar operações de entrada e saída. Assim, faz-se necessário um mecanismo seguro de comunicação com periféricos, restringindo o tráfego de informações apenas para entidades confiáveis.

A criptografia é uma forma de proteção de dados muito utilizada na transmissão e armazenamento de informações sensíveis. Embora muitos sistemas embarcados possuam requisitos de segurança, a sua grande maioria também possui uma limitação de recursos de hardware e por isto necessitam de mecanismos de segurança de baixo custo de área. Um dos desafios de engenharia no projeto destes dispositivos é o balanceamento entre requisitos de segurança, custo de área e desempenho. Os algoritmos de criptografia leve (LWC - *Lightweight Cryptography*), surgem para suprir esta necessidade trazendo um nível de segurança aceitável com um baixo impacto no sistema em geral [OLI18].

A interface de rede proposta neste trabalho atua no controle da comunicação entre os PEs e periféricos conectados ao MPSoC, sendo responsável por: (*i*) avaliar as requisições de leitura e escrita realizadas, podendo recusá-las ou aceitá-las; (*ii*) empacotar as informações, isto é, traduzir os sinais específicos do periférico para forma de pacotes a serem transmitidos na NoC; e por fim (*iii*) proteger os dados das mensagens através de criptografia evitando a exposição de informações sensíveis.

O MPSoC Memphis (*Many-core Modeling Platform for Heterogenous SoCs*) é uma plataforma com alto poder de processamento paralelo que utiliza a NoC HERMES para interconectar os elementos de processamento [CAR09, RUA19]. A mesma foi desenvolvida pelo grupo de pesquisa GAPH – Grupo de Apoio a Projetos de Hardware [GAP18], e atualmente utiliza processadores Plasma (arquitetura similar ao processador MIPS), existindo também estudos deste mesmo grupo para suportar outros processadores como MB-Lite [WAC11].

As contribuições deste trabalho consistem em evoluir a plataforma Memphis em três aspectos: (*i*) permitir a integração de periféricos de terceiros implementando o suporte a protocolos padronizados; (*ii*) restringir a utilização dos periféricos somente a aplicações autorizadas através da definição de políticas de acesso; (*iii*) garantir a confidencialidade e autenticidade das mensagens trocadas internamente no MPSoC entre as tarefas e periféricos através do uso de criptografia leve.

1.1 Motivação

A motivação do presente trabalho se baseia na necessidade de um modelo de *comunicação seguro* entre tarefas que executam no MPSoC e periféricos externos. A proposta de interface de rede segura deve cobrir os seguintes pontos:

(i) impedir que IPs de terceiros possam explorar vulnerabilidades por conhecer o protocolo interno injetando pacotes forjados;

(ii) impedir que aplicações não autorizadas possam interferir na utilização de recursos externos pelas demais;

(iii) impedir a exposição de informações sensíveis provenientes da comunicação entre tarefas e periféricos externos.

1.2 Objetivos

O objetivo estratégico deste trabalho é propor uma *interface de rede* (NI, do inglês *Network Interface*) *segura* para a comunicação entre PEs de um MPSoC e periféricos, incluindo um método de comunicação seguro para a troca de informação. Os objetivos específicos deste trabalho incluem:

- Definição de interface e protocolo padronizado para comunicação entre o MPSoC e seus periféricos a fim de permitir a integração de periféricos de terceiros (*Third party intellectual property* - 3PIP).
- Definição de política de acesso aos periféricos a fim de restringir a utilização dos mesmos somente a aplicações previamente autorizadas, não permitindo interferências como contenção de dados ocasionadas por outras aplicações ou até mesmo periféricos (o que poderia levar a um ataque de negação de serviço – *Denial-of-Service*, DOS).
- Proteção aos dados que trafegam internamente no MPSoC por criptografia leve a fim de garantir a confidencialidade e autenticidade das mensagens trocadas entre as tarefas e os periféricos.

1.3 Metodologia

A presente Dissertação tem por base a NoC Hermes [MOR04] e plataforma MPSoC Memphis [RUA19]. Inicialmente desenvolve-se uma NI conectada à NoC Hermes, visando a validação da mesma no nível RTL. Nesta etapa também é avaliado o suporte ao protocolo padrão de indústria Wishbone, através do desenvolvimento de um periférico simples (Anexo I). Na sequência, integra-se a NI ao MPSoC Memphis com o objetivo de desenvolver a API de comunicação entre tarefas e periféricos e é desenvolvido um periférico de memória externa em Wishbone. A avaliação é realizada por simulação VHDL que permite a visualização dos pacotes que trafegam na NoC, validando o protocolo. Durante a simulação VHDL as tarefas em execução nos PEs geram arquivos contendo relatórios com informações relativas aos protocolos desenvolvidos.

1.4 Estrutura do documento

O conteúdo desta Dissertação está distribuído ao longo dos próximos capítulos como segue:

- Capítulo 2: apresenta o modelo de ameaças considerado neste trabalho de pesquisa.
- Capítulo 3: apresenta o referencial teórico em diferentes áreas, que incluem: (*i*) a arquitetura de referência, ou seja, o MPSoC Memphis; (*ii*) propostas de técnicas de seguranças aplicadas a MPSoCs; (*iii*) protocolo Wishbone; (*iv*) algoritmos de criptografia leve implementados em hardware.
- Capítulo 4: apresenta trabalhos relacionados ao desenvolvimento de NIs.
- Capítulo 5: detalha a contribuição principal da presente Dissertação, correspondendo à proposta da NI segura, e sua avaliação.
- Capítulo 6: conclui este trabalho, e aponta direções para trabalhos futuros.

2 MODELO DE AMEAÇAS

A comunicação entre os PEs de um MPSoC com periféricos de terceiros pode criar oportunidades de ataques. É possível que esta comunicação seja interrompida, monitorada e modificada de forma maliciosa através de software malicioso ou Hardware Trojans [WEB20]. No nosso modelo de ameaças consideramos que um atacante é capaz de infectar periféricos ou IPs conectados à NoC tornando possível a ativação de Hardware Trojans como apresentado em [ANC14].

Com o objetivo de definir as ameaças que afetam a segurança da comunicação entre PEs e periféricos relacionamos as vulnerabilidades apresentadas no modelo STRIDE da Microsoft [KOH99] com os princípios de segurança apresentados no trabalho de Caimi et al. [CAI19b] e por fim apresentamos uma ilustração do modelo de ameaças utilizado como base neste trabalho.

- ***Spoofing of user identity*** **(Falsificação de identidade) - Autenticação**

Neste caso um atacante poderia se apresentar falsamente como uma entidade autêntica para receber informações sigilosas, como por exemplo uma chave criptográfica utilizada para codificação de dados.

Mitigação: A distribuição das chaves se dá na etapa denominada autenticação mútua através de uma rede segura, como proposto em [CAI19b]. Negação de pacotes que chegam na NI oriundos de PEs não autorizados através do endereço contido no cabeçalho do pacote e em uma assinatura contida no corpo do pacote.

- ***Tampering with data*** **(Violação de dados) – Integridade**

Neste caso um atacante pode modificar um dado armazenado ou em transmissão, de forma detectável ou indetectável a fim de comprometer a segurança do sistema.

Mitigação: A assinatura no cabeçalho do pacote garante a sua integridade. Utilizando o algoritmo SipHash [AUM12] garantimos que apenas quem possui a chave é capaz de gerar uma assinatura autêntica. A fim de garantir também a integridade dos dados da operação, seja de leitura ou escrita é necessário um mecanismo de assinatura para os dados.

- ***Repudiability*** **(Repudiabilidade) – Não repúdio**

Um PE realizando uma requisição maliciosa não rastreável a algum periférico ou de forma que seja possível negar a sua autoria.

Mitigação: A memória interna da NI que armazena as requisições realizadas poderia ser adaptada para servir como relatório de auditoria já que ela é utilizada pela NI para gerenciar as requisições verificando assinaturas e permissões. O problema relacionado à mitigação deste princípio de segurança é o custo de memória associado ao armazenamento das transações realizadas, sendo assim não abordamos este princípio de segurança neste trabalho.

- ***Information disclosure*** **(Divulgação de informações) – Confidencialidade**

O vazamento de informações sensíveis pode ocorrer tanto na transmissão ao periférico (em trânsito) ou até mesmo no armazenamento dos buffers internos do MPSoC.

Mitigação: Codificação das mensagens utilizando chave criptográfica.

- ***Denial of Service*** **(Negação de serviço) - Disponibilidade**

Este tipo de vulnerabilidade impacta o funcionamento do sistema tornando-o inacessível de forma momentânea ou permanente. Caso um periférico nunca responda para o PE este poderia ficar esperando uma resposta indefinidamente, ou então caso o PE nunca libere o periférico, este poderia ficar inacessível a qualquer outro PE. Muitas requisições de um mesmo PE poderiam impactar o funcionamento do periférico também.

Mitigação: Mecanismo de priorização e justiça no atendimento das requisições, além de tempo de expiração caso o periférico não responda ou o PE não libere o recurso. A NI ignora as mensagens oriundas de PEs não autorizados, não influenciando no funcionamento do periférico.

- ***Elevation of Privilege*** **(Elevação de privilégios) - Autorização**

Um usuário com mais direitos de acesso do que deveria podendo comprometer a segurança ou interromper completamente o funcionamento do MPSoC de forma não detectável. O usuário poderia ser considerado um periférico que é acoplado no MPSoC e seria capaz de enviar informações para a rede.

Mitigação: Os periféricos somente respondem a solicitações realizadas pelos PEs, atuando assim como escravos em um modelo de comunicação mestre-escravo. Tanto as mensagens de requisição à NI como as respostas dos periféricos têm seus cabeçalhos assinados com o algoritmo SIPHash que garante a integridade e autenticidade das informações.

A Figura 1 ilustra os pontos de vulnerabilidade em um MPSoC abordados por este trabalho tendo como referência [SEP21] e [SAN21].

1. Através da injeção de pacotes na rede, um periférico externo malicioso pode causar congestionamento, violando o princípio de disponibilidade.
2. Uma aplicação mal-intencionada executando em um PE pode tentar dominar o acesso a algum periférico externo sem autorização, impedindo as demais tarefas de fazer uso do mesmo.
3. Um roteador infectado por um Hardware Trojan que faça a leitura ou altere os pacotes que trafegam por ele está violando a integridade das dos pacotes.
4. Através de técnicas de ataque de canal lateral (SCA, *side-channel attack*) é possível monitorar o tráfego de dados sensíveis que estão em trânsito entre roteadores, violando assim o princípio da confidencialidade.

5. A leitura de dados sensíveis que estão armazenados em buffers ou periféricos compartilhados, como por exemplo uma memória externa, viola o princípio da confidencialidade.

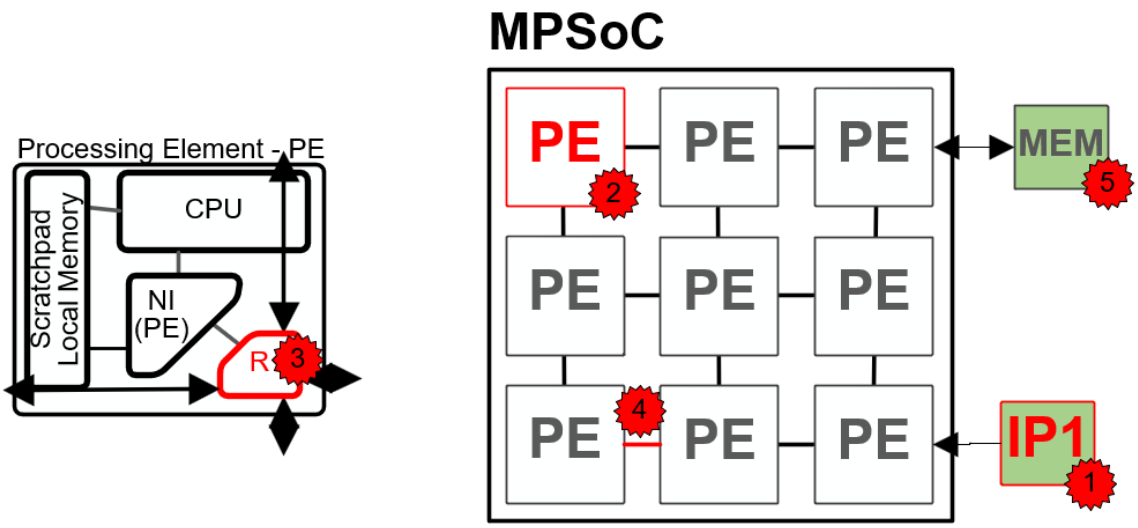


Figura 1 – Superfície de ataque no MPSoC (Fonte: Autor).

3 REFERENCIAL TEÓRICO

Este Capítulo está organizado em 4 seções, relacionadas ao desenvolvimento da NI segura:

- apresentação da arquitetura de referência, MPSoC Memphis (Many-corE Modeling Platform for Heterogenous SoCs) [RUA19];
- propostas de técnicas de seguranças aplicadas no MPSoC HeMPS [CAR09], arquitetura predecessora do MPSoC Memphis [RUA19];
- protocolo Wishbone;
- propostas de criptografia leve em hardware.

3.1 Arquitetura de Referência

Esta Seção apresenta a NoC HERMES, e o MPSoC de referência denominado Memphis.

3.1.1 HERMES NoC

A rede HERMES [MOR04] possui topologia malha e pode ser parametrizada através da ferramenta Atlas. O principal componente da NoC HERMES é o roteador. Seu principal objetivo é prover a correta transferência de mensagens entre PEs. O roteador implementa o algoritmo de roteamento, buffer para armazenamento temporário de informações, e sinais de controle. A rede HERMES possui 5 portas bidirecionais: local, norte, sul, leste, oeste.

Dentre os modos de chaveamento, o escolhido foi *wormhole* por requerer menor profundidade de *buffers*, e prover baixa latência. A lógica de controle interna do roteador é simples, e consiste em basicamente conectar as portas de entrada com as de saída, de acordo com o destino desejado, criando um canal para a passagem dos dados, na forma de um *pipeline.*

Na rede HERMES, ao longo do tempo, foram implementados diferentes tipos de algoritmos de roteamento. O mais utilizado é o roteamento XY. Este consiste em percorrer primeiramente os roteadores que possuem a mesma coordenada do eixo X e então depois percorrer no sentido do eixo Y até chegar ao roteador destino.

Atlas é um conjunto de ferramentas que automatiza processos relacionados ao projeto de NoCs [OST05], possibilitando ao projetista rapidamente avaliar diferentes configurações em ambiente de simulação. A Figura 2 apresenta a interface principal da ferramenta ATLAS, que compreende: (*i*) geração de NoCs, permite parametrizar parâmetros como topologia, algoritmo de roteamento, largura de *flit*, profundidade dos buffers etc; (*ii*) geração de tráfego, permite definir uma distribuição estatística, tanto temporal quanto espacial, para o tráfego; (*iii*) simulação, realiza a chamada ao simulador Moldelim; (*iv*) avaliação de desempenho, gera gráficos relativos ao desempenho da rede; (*v*) avaliação de consumo de energia, realiza a estimativa de potência dissipada na rede.

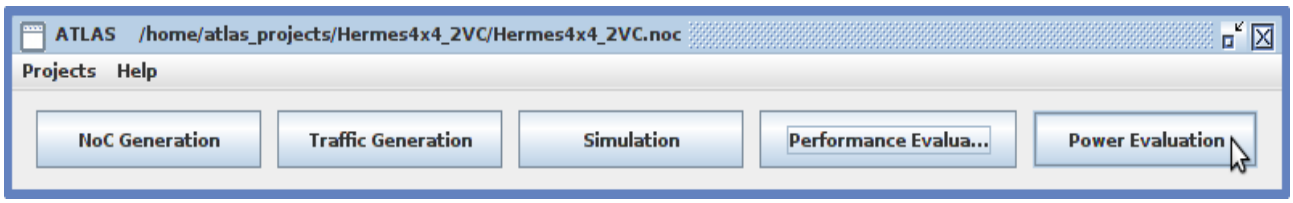


Figura 2 – Interface principal da ferramenta ATLAS [OST05].

3.1.2 MPSoC Memphis

A Figura 3(a) apresenta uma visão geral dos componentes do MPSoC Memphis. O sistema contém duas regiões: PEs propósito geral (GPPC) e periféricos. O GPPC contém um conjunto de PEs idênticos que executam aplicações de propósito geral. Periféricos são IPs especializados. Os periféricos são conectados nas bordas do GPPC. A conexão dos periféricos nas bordas da GPPC permite a geração de planta baixa (*floorplanning*) regular para os PEs, com periféricos distribuídos ao longo do limite do GPPC.

Cada PE (Figura 3(b)) contém uma CPU, memória local, um roteador (PS) e uma NI com suporte à acesso direto à memória (DMNI). A memória local possui duas portas de acesso, armazenando código e instruções. O objetivo de usar este modelo de memória é reduzir o consumo de energia relacionado aos controladores de cache e ao tráfego NoC (transferência de linhas de cache). Se alguma aplicação requerer um espaço de memória maior do que o disponível na memória local, é possível ter memórias compartilhadas conectadas ao sistema.

O roteador PS é utilizado na rede Hermes. O roteador é configurado com largura de *flit* igual a 32 bits, roteamento XY, arbitragem *round-robin*, buffer de entrada com capacidade de 8 *flits*, controle de fluxo baseado em crédito e comutação de pacotes wormhole.

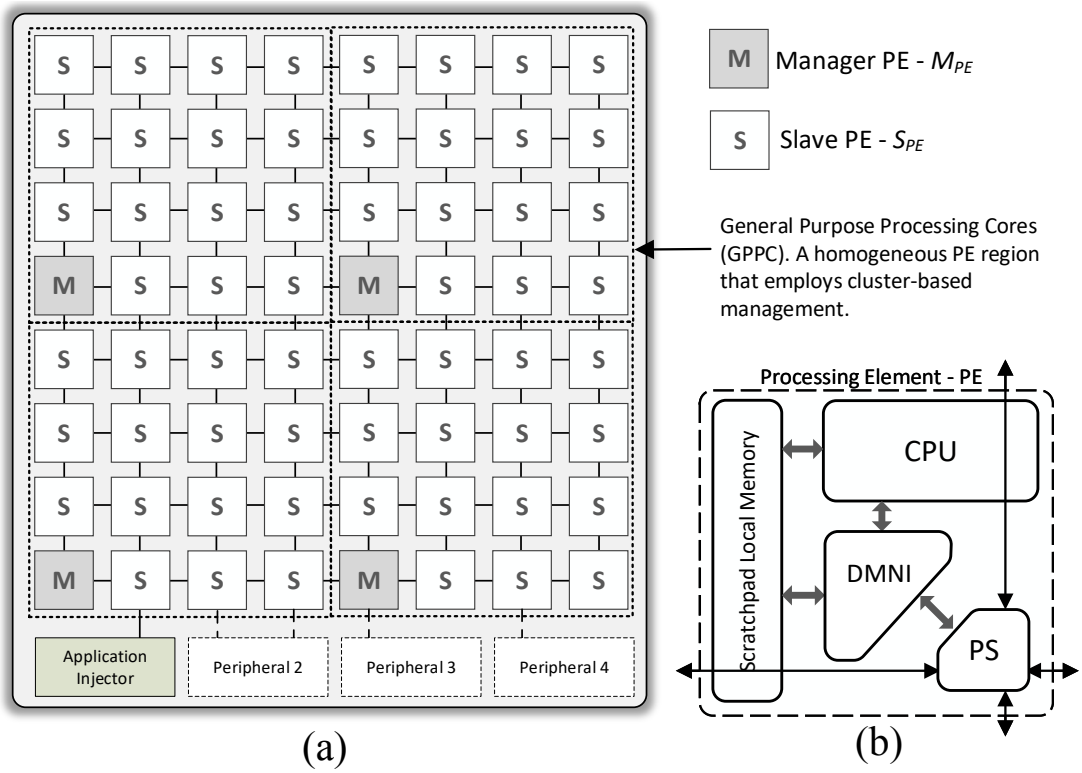


Figura 3 – Visão geral da Memphis. (a) MPSoC e periféricos; (b) Arquitetura dos PEs [RUA19].

O MPSoC possui um periférico especial (App_{Inj} - Injetor de Aplicações - *Application Injector*), responsável por injetar o código objeto das aplicações no MPSoC e dar início a execução, atuando assim como mestre se considerarmos um protocolo mestre-escravo. Os periféricos devem possuir uma interface baseada em créditos para estabelecer comunicação com os roteadores. Eles podem servir como aceleradores de hardware, processadores dedicados para processamento de imagens, áudio, vídeo, criptografia, entre outros. Estes também podem se conectar com dispositivos externos ao MPSoC como rede *Ethernet*, interface USB, memória DDR, entre outros.

As aplicações são desenvolvidas em linguagem de programação C. Um sistema operacional simples, *microkernel*, provê suporte multitarefa e comunicação entre tarefas mapeadas em processadores distintos. Cada tarefa pode ser mapeada estaticamente ou dinamicamente em um elemento de processamento da plataforma. A configuração do mapeamento, estática ou dinâmica, é definida em um arquivo de configuração em formato texto.

3.2 Técnica de segurança aplicadas no MPSoC HeMPS

A apresentação destas técnicas, zonas seguras e HNoC, são importantes para que o leitor compreenda os métodos utilizados para agregar segurança às aplicações em execução no MPSoC.

Em comum, estas técnicas não possuem um método que integre uma NI segura, sendo esta uma motivação para a presente Dissertação.

3.2.1 Segurança em MPSoCs através de zonas seguras

Caimi et al. [CAI17][CAI19b] propõem um protocolo para solucionar dois problemas de segurança presentes em MPSoCs: a admissão segura de App_{sec}s (aplicações com requisitos de segurança), e a prevenção de compartilhamento de recursos durante a execução das mesmas. Um dos requisitos desta solução é que ela deve ser de baixo custo computacional, descartando o uso de soluções existentes para redes de computadores.

O protocolo inicia por um processo de autenticação, sendo criado neste processo uma chave criptográfica – K_e. Esta chave é utilizada para a geração de um MAC (*Message Authentication Code*), acrescido ao final do código objeto das tarefas. O MAC garante a autenticidade e a integridade das aplicações.

- Autenticidade: garante que a mensagem é autêntica, isto é, quem criou a mensagem é uma entidade confiável. A chave K_e é utilizada na geração do MAC que é enviado para o destinatário. Logo, somete através da posse desta chave é possível verificar a corretude do MAC recebido.
- Integridade: garante a integridade das informações, isto é, a mensagem transmitida não foi alterada por alguma entidade maliciosa durante a sua transmissão. O MAC transmitido através da mensagem deve ser igual ao MAC gerado pelo destinatário a partir da mensagem transmitida da chave K_e.

A plataforma de base é a HeMPS [CAR09]. Este trabalho incluiu no PE, um gerador

de números pseudoaleatórios (*Pseudo Random Number Generator* - PRNG). O sistema contém duas NoCs, uma de controle e outra de dados. Estas NoCs contam com um sistema de *wrappers* em suas portas de controle de fluxo, que quando ativados criam a zona segura. A Figura 4 apresenta a arquitetura para suportar a técnica de zonas seguras.

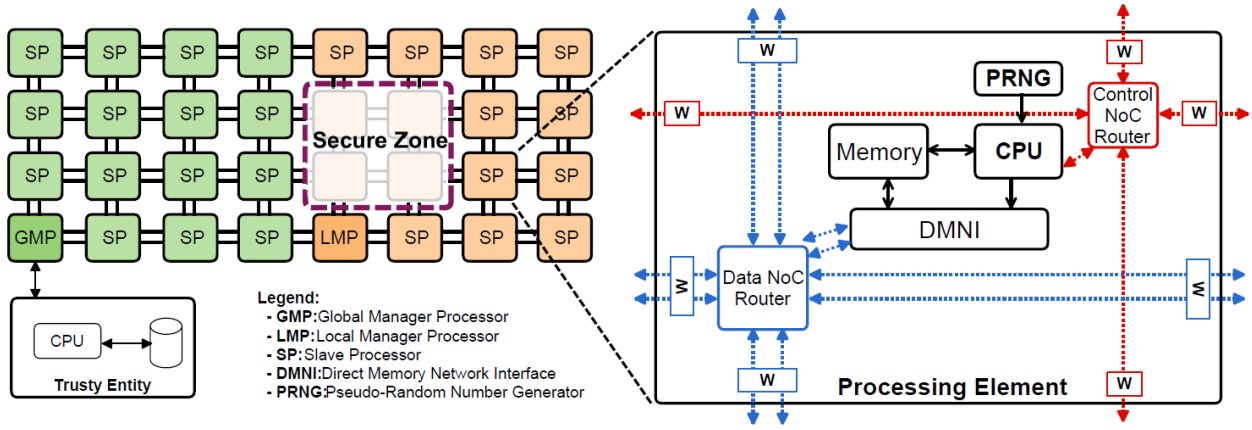


Figura 4 - Arquitetura HeMPS com suporte a zonas seguras [CAI19b].

O protocolo proposto pelo artigo é composto por 7 fases: (*i*) inicialização do sistema; (*ii*) registro de entidades confiáveis; (*iii*) admissão e mapeamento de tarefas; (*iv*) protocolo interno de troca de chaves; (*v*) alocação de tarefas e verificação de MAC; (*vi*) isolamento e execução da aplicação; (*vii*) liberação dos recursos.

1. A inicialização do sistema envolve dois atores: o GMP (PE gerente do sistema) e o injetor de aplicações (*trust entity* na Figura 4). Na inicialização do sistema um número aleatório é gerado por cada ator. Este número é utilizado para a seleção de um determinado polinômio. A partir do polinômio é calculado um conjunto de números primos e pontos sobre uma curva elíptica. O resultado desta fase é a geração de uma dupla {ID, Pu_k} (identificador e chave pública), além da chave privada de cada ator. Cada ator publica sua dupla {ID, Pu_k}.
2. Geração da chave K_e, utilizando-se as duplas {ID, Pu_k}. Utiliza-se o mecanismo ECDH (*Elliptic Curves Diffie-Hellman)* devido a seu bom desempenho e nível de segurança em comparação com outros meios de geração de chaves.
3. Realizar o mapeamento da aplicação em uma região retangular na NoC, para que seja possível executar a App_{sec} requerida.
4. Transmissão da K_e para os SPs que executarão as tarefas da App_{sec}.
5. O código objeto das tarefas é enviado aos SPs definidos na etapa 3, com uma MAC anexado ao final do código objeto. O MAC é verificado para garantir a autenticidade e integridade do código objeto transmitido. A geração do MAC utiliza o algoritmo de SIPHASH e a chave K_e.
6. Após a alocação das tarefas, a zona segura é fechada ativando-se os *wrappers* dos SPs (PEs que executam tarefas) que estão na fronteira da zona segura. Desta forma é garantido que a execução da App_{sec} não sofrerá interferências externas de possíveis entidades maliciosas.

7. Após a execução da App_{sec}, a última fase consiste em preparar os SPs da zona segura para receber tarefas de outras aplicações. Estes SPs têm a sua memória zerada para evitar vazamento de dados e então os *wrappers* de proteção são desativados para a reabertura da zona segura.

O trabalho habilita a execução de App_{sec}s em zonas seguras, isolando os elementos de processamento do restante do sistema, inibindo possíveis ataques de entidades maliciosas. Além disto, é proposto um protocolo que garante a autenticidade e integridade na admissão das App_{sec}s. A solução proposta trás segurança sem impactar no desempenho de execução das aplicações, uma vez que estas estão mapeadas em PEs dentro de uma zona segura. O impacto do método está no tempo para inicializar a execução da aplicação, uma vez que é necessário calcular o MAC de cada código objeto, e controlar os *wrappers* de fronteira.

Em [CAI19a], o Autor estende sua proposta para realizar a comunicação com periféricos, tendo por foco a API de comunicação, e não a proposta de uma NI. Funções específicas para o envio e recebimento de pacotes à periféricos externos foram criados. Além do *Send()* e *Receive()* para troca de dados entre tarefas, neste artigo foram introduzidas as primitivas *IO_Send()* e *IO_Receive(),* que são responsáveis por formatar o pacote das mensagens de entrada e saída a fim de transmiti-lo ao periférico de destino. O algoritmo de roteamento utilizado para estes pacotes é o XY. Como ilustrado na Figura 5, o pacote de saída, que sai do roteador em direção ao periférico, percorre as coordenadas do eixo X e em seguida Y. O pacote de entrada do roteador, que parte do periférico também percorre primeiro o eixo X e em seguida o Y. O mecanismo apresentado neste artigo permite o controle dos *wrappers*, através de *pontos de acesso*. Para a saída de dados abre-se apenas o *wrapper* de saída de dados (círculo vermelho na Figura 5), e para a entrada de dados abre-se apenas o *wrapper* de entrada de dados (círculo azul na Figura 5). O mecanismo proposto é mestre-escravo, onde toda comunicação parte dos PEs dentro da zona segura. Os *wrappers* são abertos para apenas um pacote. A cada nova transação, os *pontos de acesso* são novamente abertos. Este processo garante segurança, pois impede que dados sejam extraídos da zona segura.

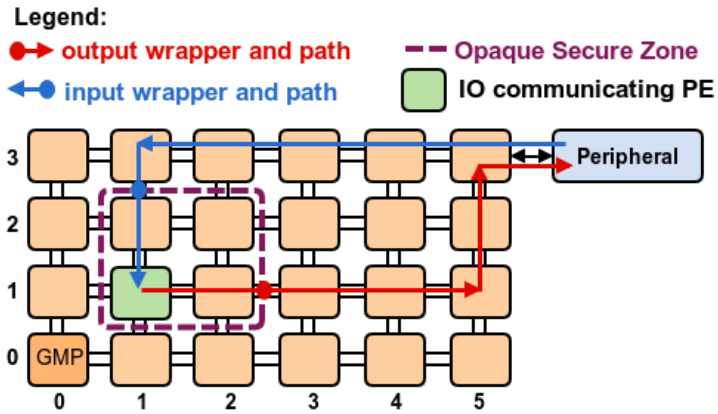


Figura 5 – Comunicação entre PE e periférico [CAI19a].

O artigo ainda aborda possíveis casos de ataques em que elementos mal-intencionados poderiam se aproveitar da janela de oportunidade em que um PE espera pelo retorno do periférico. Entretanto, como os dados são criptografados de ponta a ponta, só o

PE interno à zona segura é capaz de descriptografar o conteúdo do pacote, rejeitando qualquer pacote malicioso. Da mesma forma, somente o periférico com sua chave privada é capaz de decifrar a mensagem transmitida pelo PE, codificada com a chave pública do PE.

O mecanismo de segurança proposto para a comunicação entre periféricos e PEs adiciona cerca de 12% a 15% de *overhead* de desempenho nas transações de E/S. Neste trabalho os testes para avaliação de desempenho não consideram o módulo de criptografia, porém segundo o autor este não influenciaria no resultado geral visto que atua em forma de pipeline, adicionando apenas uma latência inicial para o processamento dos dados.

3.2.2 Security Vulnerabilities and Countermeasures in MPSoCs [SAN21]

O trabalho [SAN21] aborda vulnerabilidades de segurança em MPSoCs, e propõe um conjunto de mecanismos de baixo impacto de área e desempenho como contramedida para proteger o mesmo. O estudo adota como base uma arquitetura homogênea e comunicação baseada em NoC, e assume quatro premissas: (*i*) nenhum ataque ocorre durante o carregamento do sistema operacional, garantindo a integridade do mesmo; (*ii*) nenhum ataque ocorre durante o carregamento das aplicações, garantindo a integridade do código das mesmas; (*iii*) as aplicações se comunicam apenas com o sistema operacional que executa nos SPs (*Slave* PEs), sendo assim elas não tem acesso aos MPs (*Manager* PEs), elementos de processamento mestre que gerenciam a admissão das aplicações, distribuição de tarefas e mapeamento; (*iv*) o MPSoC não possui Hardware Trojans (HT).

Vulnerabilidades podem ser exploradas através de ataques de negação de serviço (DoS), *spoofing* e *hijacking* colocando em risco os recursos compartilhados do MPSoC, e comprometendo a confidencialidade e integridade dos dados. Entre as vulnerabilidades na infraestrutura de comunicação o artigo cita o congestionamento de pacotes que pode impactar o tráfego do sistema baseado em NoC e ainda pacotes malformados, com o tamanho do payload informado no cabeçalho diferente do enviado, sem o *flit* de término, ou então endereço de origem e destino errados. Entre as vulnerabilidades do nível de software o artigo menciona o recebimento de mensagens com serviços não definidos, cujo drivers do kernel não são capazes de interpretar, *man-in-the-middle*, em que uma tarefa executando em outro PE intercepta as mensagens trocadas entre outros PEs vizinhos e ainda ataques de memória se o sistema operacional não possui proteção e controle de acesso por diferentes tarefas.

O artigo propõe a adoção de quatro mecanismos de segurança:

- isolamento espacial;
- utilização de uma rede dedicada para dados sensíveis – HNoC;
- filtro de bloqueio de tráfego;
- criptografia leve.

A abordagem de isolar as tarefas pertencentes a uma mesma aplicação das demais tarefas na hora do mapeamento previne o vazamento de informações sensíveis, porém também faz com que o MPSoC seja subutilizado em poder de processamento.

A proposta da adoção de uma rede dedicada para a transmissão de dados sensíveis consiste em conectar os PEs de forma serial através de um caminho hamiltoniano, onde apenas os MPs podem injetar dados (Figura 6(a)). O HRouter ilustrado na Figura 6(b) tem a função de receber e armazenar os dados sensíveis.

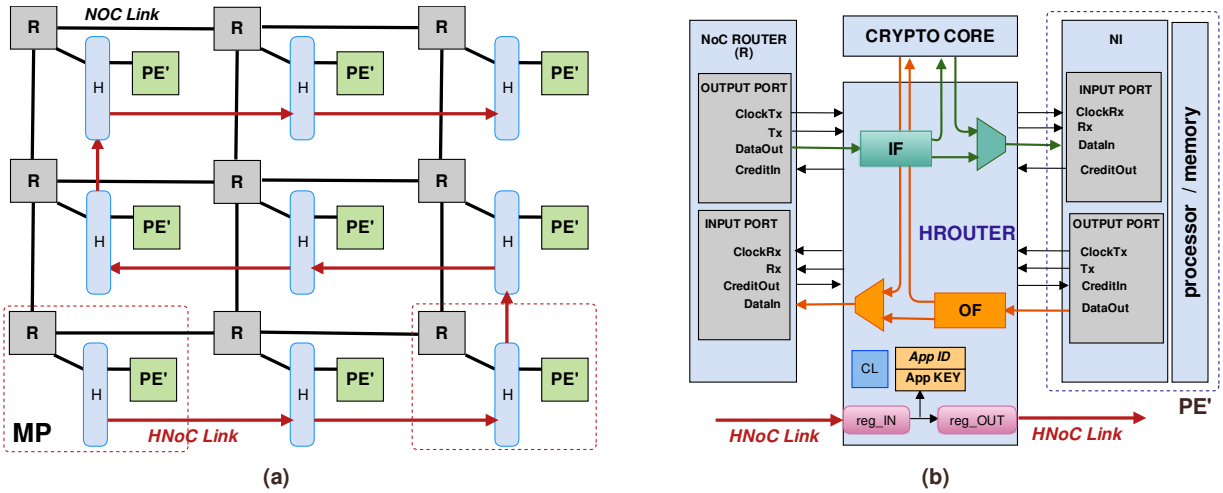


Figura 6 - (a) HNoC, representado por HRouter (H) e links vermelhos, em um MPSoC 3x3. R: roteador de dados, PE': NI / processador / memória local; (b) HRouter com filtros de saída e entrada (OF e IF) - CL: lógica de controle [SAN21].

O mecanismo de bloqueio de tráfego é implementado no HRouter através de filtros de entrada (IF) e saída (OF), além de uma lógica de controle e comunicação com um IP de criptografia. O bloco de controle (CL na Figura 6) avalia o conteúdo dos pacotes. Para pacotes de configuração, o endereço do cabeçalho é verificado com o endereço do HRouter, e havendo igualdade os próximos pacotes são os dados a serem armazenados nos registradores que indicam o identificador da aplicação e a chave criptográfica. O filtro OF previne ataques do tipo *man-in-the-middle* verificando o conteúdo do quarto *flit*, que é o que identifica a aplicação. O filtro IF garante a admissão de dois tipos de pacotes, primeiro com o identificador da aplicação correto, e segundo pacotes de gerenciamento transmitidos pelo MPs. Os demais pacotes são descartados prevenindo ataques de negação de serviço (*DoS*).

Este artigo avalia a utilização de dois diferentes módulos de criptografia, AES e SIMON, sendo o último de criptografia leve. O algoritmo SIMON apresentou um menor custo de área (5x menor) e de energia (25x menor) em relação ao AES, que por sua vez é mais rápido, apresentando uma latência 3 vezes menor que o SIMON.

Em geral, a rede dedicada HNoC possuí baixo custo de área e se mostrou promissora para ser utilizada na transmissão de dados sensíveis e distribuição de chaves criptográficas, com as quais é possível garantir confidencialidade e integridade das informações por aplicação.

3.3 Protocolo Wishbone

O protocolo Wishbone [WIS10] é muito utilizado para desenvolvimento de IPs públicos (3PIP). A implementação deste protocolo é simples, possuindo poucos sinais de controle em comparação com outros protocolos, como AXI, e é baseado em um modelo mestre-

escravo em que as transações de leitura e escrita ocorrem em um endereço específico definido pelo mestre. Os termos transações e operações são usados como sinônimo neste contexto.

As interconexões Wishbone podem ser realizadas de quatro maneiras:

- Ponto-a-ponto. É a forma mais simples de conectar dois IPs com interface Wishbone. Este modo permite uma única interface mestre, conectada a uma única interface escravo. Este é o método utilizado neste trabalho, em que a NI é o mestre e o periférico é o escravo (Figura 7).

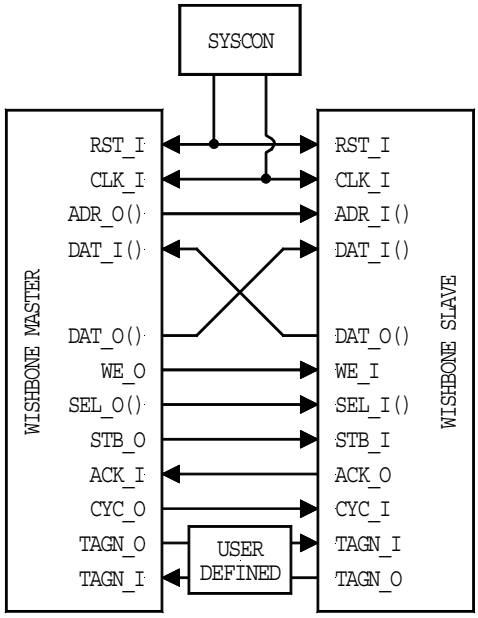


Figura 7 – Interconexão de ponto a ponto [WIS10].

- Fluxo de dados (*data flow*). Neste modo de conexão, múltiplos IPs são ligados em sequência, com os dados sendo transmitidos dos mestres para os escravos (Figura 8).

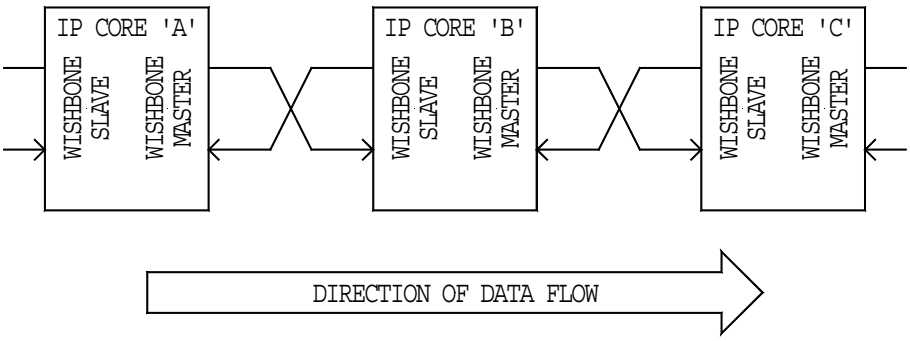


Figura 8 – Interconexão *data flow* [WIS10].

- Barramento compartilhado. Este modo permite conectar dois ou mais IPs mestre com um ou mais IPs escravos, em que na etapa inicial de configuração do barramento é que se define o alvo da operação de leitura ou escrita. Nesta abordagem é necessário um arbitro, que determina quando um mestre ganha acesso ao barramento compartilhado.

- *Crossbar*. Este modo é utilizado para conectar dois ou mais mestres de forma que cada um consiga acessar dois ou mais escravos. Um arbitro é necessário para definir quando cada mestre ganha acesso ao escravo indicado na etapa de configuração. Diferentemente do barramento compartilhado, nesta abordagem é possível estabelecer canais de comunicação operando em paralelo.

Adotando a interconexão do tipo ponto-a-ponto e modo de operação *pipeline* podemos obter uma vazão mais alta quando comparado a operações individuais. A Figura 9 mostra uma operação de leitura deste tipo. A comunicação se inicia pelo mestre e durante toda operação o sinal *CYC_O* permanece ativo. A configuração das operações é realizada no ciclo inicial, chamado de ciclo de barramento, ou *bus cycle*, em que o sinal STB_O está ativo. Neste ciclo inicial, o modo leitura é selecionado através do sinal WE_O em zero e o endereço da operação é informado através do sinal ADDR_O. Após o processamento das informações de configuração o escravo notifica o recebimento através do sinal ACK_I e, então, envia o dado solicitado através do barramento DAT_I; o ACK_I notifica também que os dados transmitidos em DAT_I são válidos.

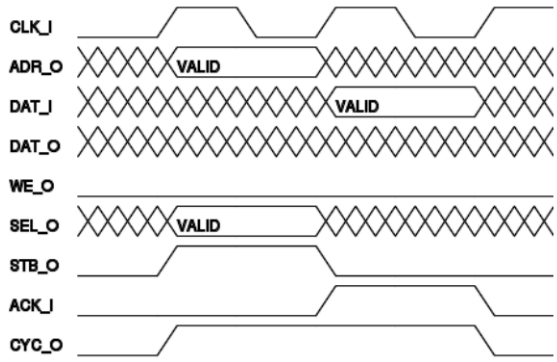


Figura 9 - Operação de leitura utilizando protocolo Wishbone – *Pipelined single read* [WIS10].

A operação de escrita ilustrada na Figura 10 também se inicia pelo mestre e durante toda operação o sinal *CYC_O* permanece ativo. No ciclo inicial de configuração, o modo de escrita é selecionado através do sinal *WE_O* em nível lógico alto, o endereço da operação é informado através do sinal *ADDR_O* e o dado também é informado em *DAT_O*. Após o processamento das informações, o escravo notifica o recebimento através do sinal *ACK_I* e então a operação se encerra.

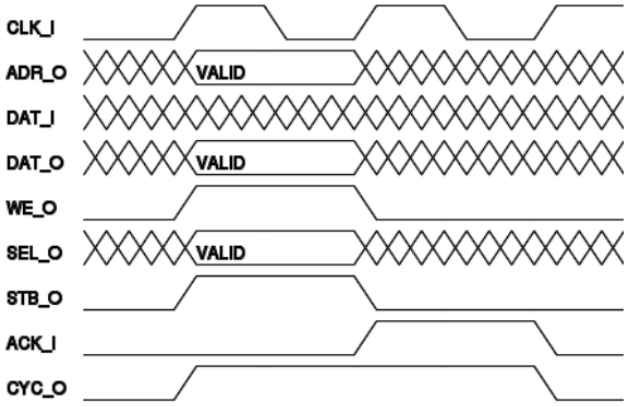


Figura 10 - Operação de escrita utilizando protocolo Wishbone – *Pipelined single write* [WIS10].

A presente dissertação adota o protocolo Wishbone devido à sua maior simplicidade comparada a outros padrões, como OCP [ACC13] ou AMBA AXI [ARM13], além de possuir licença aberta de uso.

3.4 Propostas de criptografia leve em hardware

A criptografia é utilizada na transmissão e armazenamento de informações sensíveis. Muitos sistemas embarcados possuem requisitos de segurança e uma limitação de recursos de hardware e por isto necessitam de mecanismos de segurança de baixo custo. Os algoritmos de criptografia leve, ou *lightweight cryptography algorithms,* surgem para suprir esta necessidade. Esta subseção apresenta estudos relacionados ao uso de criptografia leve, onde há a comparação entre diferentes abordagens. Ao final desta subseção apresenta-se o método utilizado neste trabalho.

3.4.1 A Survey of Lightweight-Cryptography Implementations [EIS07]

A principal contribuição deste artigo é a apresentação de um estudo comparativo entre os algoritmos de criptografia desenvolvido para serem implementados em hardware. Os algoritmos podem ser divididos em dois grupos, simétricos e assimétricos:

- Simétricos: AES, DES, DESL, DESXL, Present, Clefia.
- Assimétricos: ECC, RSA.

Dentre os algoritmos avaliados, os que se destacam pela sua eficiência são Present e Clefia. Como podemos observar na Tabela 1, a implementação do AES necessita de 1.032 ciclos para realizar as operações criptográficas enquanto o Clefia necessita apenas de 36 ciclos e o Present 32. Em relação a área ocupada o AES utiliza 3.400 portas lógicas equivalentes (GE – *gate equivalente*), enquanto o Clefia utiliza aproximadamente 5.000 e o Present 1.500 GEs.

Tabela 1 - Tabela comparativa de *lightweight ciphers* [EIS07].

Cipher	Key bits	Block bits	Cycles per block	Throughput at 100 kHz (Kbps)	Logic process	Area (GEs)
Block ciphers						
Present	80	64	32	200.00	0.18 µm	1,570
AES	128	128	1,032	12.40	0.35 µm	3,400
Hight	128	64	34	188.20	0.25 µm	3,048
Clefia	128	128	36	355.56	0.09 µm	4,993
mCrypton	96	64	13	492.30	0.13 µm	2,681
DES	56	64	144	44.40	0.18 µm	2,309
DESXL	184	64	144	44.40	0.18 µm	2,168
Stream ciphers						
Trivium[5]	80	1	1	100.00	0.13 µm	2,599
Grain[5]	80	1	1	100.00	0.13 µm	1,294

*AES: Advanced Encryption Standard; DES: Data Encryption Standard; DESXL: lightweight DES with key whitening.

A Figura 11 ilustra o bloco de dados do *cipher Present*. Podemos observar o barramento de entrada de dados (*plaintext*), a chave criptográfica (*key*), contador de rounds (*counter*), sinal de controle (*n_reset*) e finalmente a saída do dado criptografado (*ciphertext*). O *Present* utiliza tamanho do bloco de 64 bits e o tamanho da chave pode ser de 80 ou 128 bits. A transformação não linear utiliza uma única S-box de 4 bits.

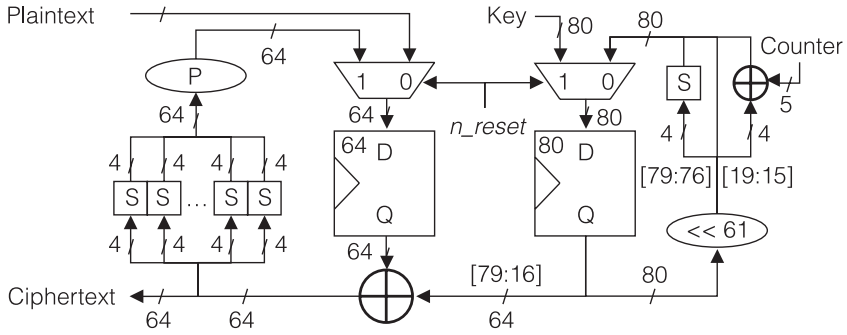


Figura 11 – Bloco de dados do *cipher* Present [EIS07].

3.4.2 Hardware Comparison of the ISO/IEC 29192-2 Block Ciphers [HAN12]

O artigo [HAN12] apresenta um estudo comparativo entre dois algoritmos de criptografia leve: *Clefia* e *Present*. O algoritmo AES é utilizado como base para fins de comparação.

- AES. Este algoritmo utiliza redes de substituição e permutação (*substitution-permutation network*) com blocos de 128 bits a chaves de 128, 192 ou 256 bits. Dependendo do tamanho da chave são necessários 10, 12 ou 14 rounds.
- Clefia. Este algoritmo é capaz de operar com os mesmos tamanhos de blocos e chaves que o AES. A função de Feistel, na qual este algoritmo se baseia opera com palavras de 32 bits e consiste em três estágios: mistura de chaves, camada de substituição e matriz de difusão. Este algoritmo pode requerer 18, 22 ou 26 rounds em função do tamanho da chave.
- Present. Ele é capaz de operar com blocos de tamanho 64 bits e chaves de 80 ou 128 bits. Baseando-se em SPN, este algoritmo foi desenvolvido com o objetivo de eficiência e realiza a cifragem em 31 *rounds*.

A Tabela 2 ilustra o resultado comparativo entre as versões iterativas e sequenciais dos algoritmos. Como podemos observar, o consumo de área do cipher *Present* é inferior aos demais e sua frequência de operação é superior para ambas as abordagens iterativa e serial, porém, nota-se que este apresenta uma latência superior no início do processo de criptografia.

3.4.3 Alternative Security Option in the 5G and IoT Era

O artigo [SCH17] discute módulos de criptografia leve (*LightWeight Cryptography* – LWC) e apresenta um comparativo entre componentes disponíveis na indústria. Os algoritmos simétricos são utilizados para geração de MACs, verificação de integridade e codificação de mensagens. Os assimétricos são utilizados quando é necessário um controle sobre quem codifica e quem descodifica, requerendo um maior gerenciamento das chaves e mecanismos de *nonrepudiation*, em que não é possível negar a autoria da codificação.

Tabela 2 - Tabela comparativa bloco *cipher* [HAN12], para dispositivos FPGA Virtex 5.

			S-Box	# Flip-Flops	# LUTs 6-Input	# Slices	Max Freq MHz	Latency Cycles	State Size bits	Tp MBps	Tp/Slice kbps/slice	Tps[1] MBps	Tp/Slice[1] kbps/slice
1	AES	Iterative	[26]	271	1391	456	96.04	26	128	472.81	1036.87	66.76	146.40
2			LUT	271	1266	359	136.84	26	128	673.67	1876.53	66.76	185.95
3		Serial	[26]	286	274	137	77.29	160	128	61.83	451.33	10.85	79.18
4			LUT	286	258	113	113.25	160	128	90.60	801.77	10.85	96.00
5	Clefia	Iterative	[11]	409	816	243	108.66	46	128	302.36	1244.27	37.73	155.28
6			LUT	409	809	267	267.00	46	128	742.96	2782.61	37.73	141.32
7		Serial	[11]	329	469	156	110.69	272	128	52.09	333.91	6.38	40.90
8			LUT	329	467	155	93.96	272	128	44.22	285.27	6.38	41.17
9	Clefia[2]	Iterative	[11]	409	816	243	108.66	34	128	409.07	1683.43	51.05	210.08
10			LUT	409	809	267	267.00	34	128	1005.18	3764.71	51.05	191.20
11		Serial	[11]	329	469	156	110.69	160	128	88.55	567.64	10.85	69.54
12			LUT	329	467	155	93.96	160	128	75.17	484.95	10.85	69.99
13	Present	Iterative	LUT	200	285	87	250.89	47	64	341.64	3926.87	18.46	212.24
14		Serial	LUT	203	237	70	245.76	295	64	53.32	761.68	2.94	42.03

[1] Using clock of 13.56MHz, ISO standard for contactless smart cards [25]
[2] Pre-mixed Key

Como podemos observar na Tabela 3, o *cipher* Present se destaca por seu desempenho, mas também pelo seu baixo consumo de potência e área em portas lógicas, se comparado com o AES.

Tabela 3 - Tabela comparativa de *lightweight ciphers* [SCH17].

Table 1.
High-level comparison of lightweight ciphers.

Algorithm	Key (bits)	Block size (bits)	# Rounds	Area (GE)	Delay[1] (ns)	Average Power (*mW*)	Platform
AES[2]	128	128	10	21,274	4.08	699.1	SW/HW
PRESENT[2]	80/128	64	31	2,186	2.32	75.8	HW
PRINCE[2]	128	64	12	2,650	4.09	112.5	HW
Midori64[2]	128	64	16	2,450	2.12	71.2	HW
Grain 128a	128	1	1	N/A[3]	0.5[3]	N/A	HW
ChaCha20	256	512	20	14,280[4]	2.57	N/A	varies[5]

[1]Refers to critical path delay.
[2]Area/Delay are for STM 90 nm Logic Process [12].
[3]Metrics for TSMC 90 nm. Area available only in μm^2 (5,831) [27].
[4]Smallest design for maximal speed at 0.18 μm CMOS.
[5]Various configurations exist, see [28].

3.4.4 SIMON

Além destes métodos de criptografia leve, há no grupo de pesquisa a implementação um core de criptografia leve que utiliza o algoritmo SIMON [SIL18], integrado ao roteador da rede Hermes. O algoritmo SIMON foi desenvolvido para ser aplicado em hardware, sendo um algoritmo de criptografia leve, de chave privada, por bloco. A execução do algoritmo compreende a geração de chaves por iteração. Cada iteração criptografa os dados utilizando rede de Feistel [BEA15]. O principal motivo para a escolha do algoritmo SIMON, em relação ao *Present* é a sua reduzida área de silício [SIL18] e facilidade de implementação.

A separação de comunicação e computação realizada pela NI torna estes componentes independentes, permitindo a realização de experimentos com diferentes IPs de criptografia leve.

4 ESTADO-DA-ARTE

Este Capítulo apresenta trabalhos relacionadas ao desenvolvimento de NIs, iniciando através de uma revisão literária (*survey*) [AGH20].

4.1 Network Adapter Architectures in Network on Chip: Comprehensive Literature Review [AGH20]

Aghaei et al. [AGH20] descreve que os desafios de comunicação vêm aumentando à medida que mais PEs são integrados em um mesmo SoC. O artigo utiliza adaptador de rede (NA) como um termo mais abrangente para um módulo que se comunica e conecta duas interfaces diferentes, *NoC* e *core* (Figura 12). O termo *core* é equivalente ao PE para o Autor.

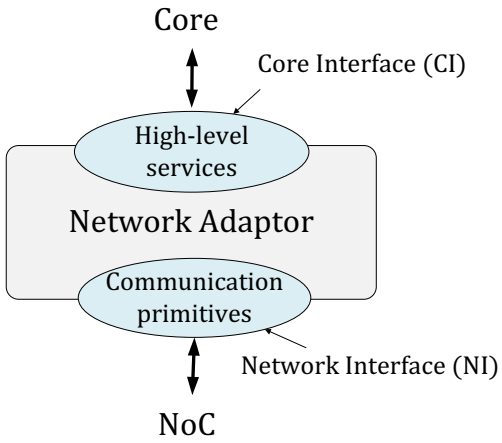


Figura 12 – Arquitetura do adaptador de rede [AGH20].

O adaptador de rede é considerado uma entidade de hardware responsável por desacoplar computação e comunicação, tornando possível a reutilização de *cores*. O *Core Interface* atua na camada de sessão do modelo OSI, enquanto a *Network Interface* implementa a camada de transporte, sendo este o primeiro nível que tem conhecimento da rede, responsável pelo empacotamento e desempacotamento de pacotes. Questões de garantia de entrega de mensagem, controle de fluxo, qualidade de serviço (QoS) e correções de erros também são tratadas nesta camada de transporte.

Em um modelo baseado em transações existem dois tipos de *cores*: mestres e escravos. O mestre é sempre o que inicia a transação e o escravo apenas responde o que lhe é solicitado. A Figura 13 mostra três tipos diferentes de transações: (*i*) o mestre realiza uma operação de leitura e então o escravo retorna o dado solicitado; (*ii*) o mestre atua realizando uma operação de escrita e enviando o dado a ser escrito; (*iii*) o mestre também realiza uma escrita, porém o escravo envia o sinal de reconhecimento (*ack*).

Os pacotes gerados pelo adaptador de rede para a NoC devem ser otimizados para não impactar de forma negativa o desempenho da rede, e não aumentar a sua complexidade em relação ao tamanho de pacote, estrutura, algoritmo de roteamento e número de sinais.

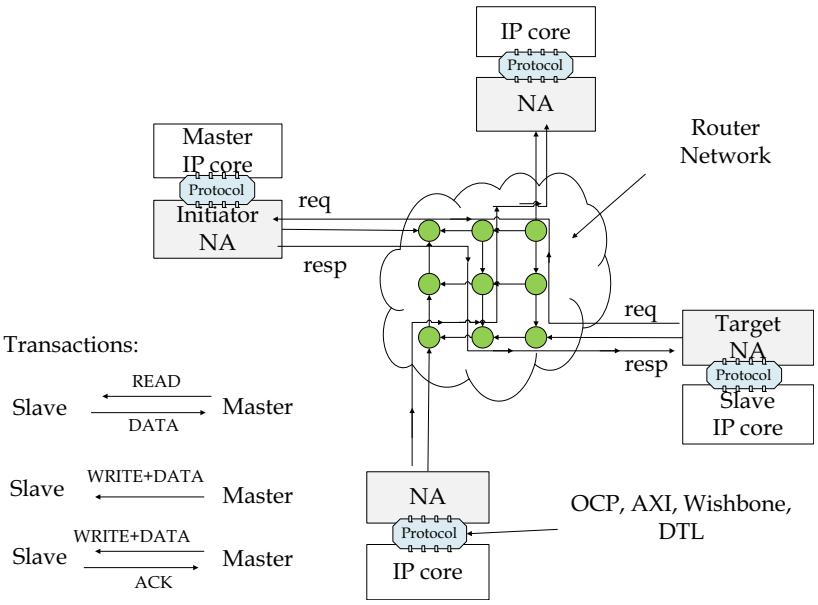


Figura 13 – Modelos de transação em NoC [AGH20].

Um dos principais desafios de projeto de SoCs é a distribuição de relógio, o que tem tornado soluções GALS (globalmente assíncrono, localmente síncrono) comuns. Mesmo que o relógio de todos os blocos de hardware de um chip seja distribuído sob a mesma frequência, ainda assim se faz necessário adaptações de fase para sincronizar a comunicação entre os blocos. A comunicação entre os elementos de processamento das NoCs é muito mais simples e modular, o que permite o aumento da frequência de forma seletiva a fim de reduzir a latência.

Este artigo classifica as diferentes arquiteturas propostas na literatura para adaptadores de rede (Figura 14). Dentre elas destacamos as arquiteturas DMNI e Wishbone por se relacionarem diretamente com a nossa pesquisa, sendo que a primeira arquitetura foi desenvolvida por membros do grupo de pesquisa GAPH sob a mesma plataforma de referência (Memphis) e baseando-se em acesso direto à memória (DMA – *Direct Memory Access).* A segunda se baseia em transações, implementando um protocolo de comunicação de código aberto com inúmeros IPs disponíveis de forma gratuita no Open-Cores [WIS10].

- **DMNI:** Este adaptador [RUA16] conecta a rede diretamente à memória do processador unindo as funcionalidades de DMA e NI em um único componente. Este adaptador permite a recepção e envio de pacotes simultaneamente. Se comparado com a utilização de módulos separados de NI e DMA, a DMNI possui vantagens de redução de latência, área e tempo de execução.
- **Wishbone:** Este adaptador [SWA14] promete ser mais rápido que adaptadores existentes que implementam protocolos como AMBA, OCP baseados crédito, *handshake* e DMA. Sua implementação se restringe a operações únicas de leitura e escrita. Na escrita os dados oriundos da NoC são desempacotados e inseridos em uma FIFO assíncrona, sendo os dados consumidos pela interface Wishbone. Na leitura os dados oriundos da interface Wishbone são empacotados e inseridos em uma FIFO assíncrona, sendo os dados consumidos pela interface da NoC.

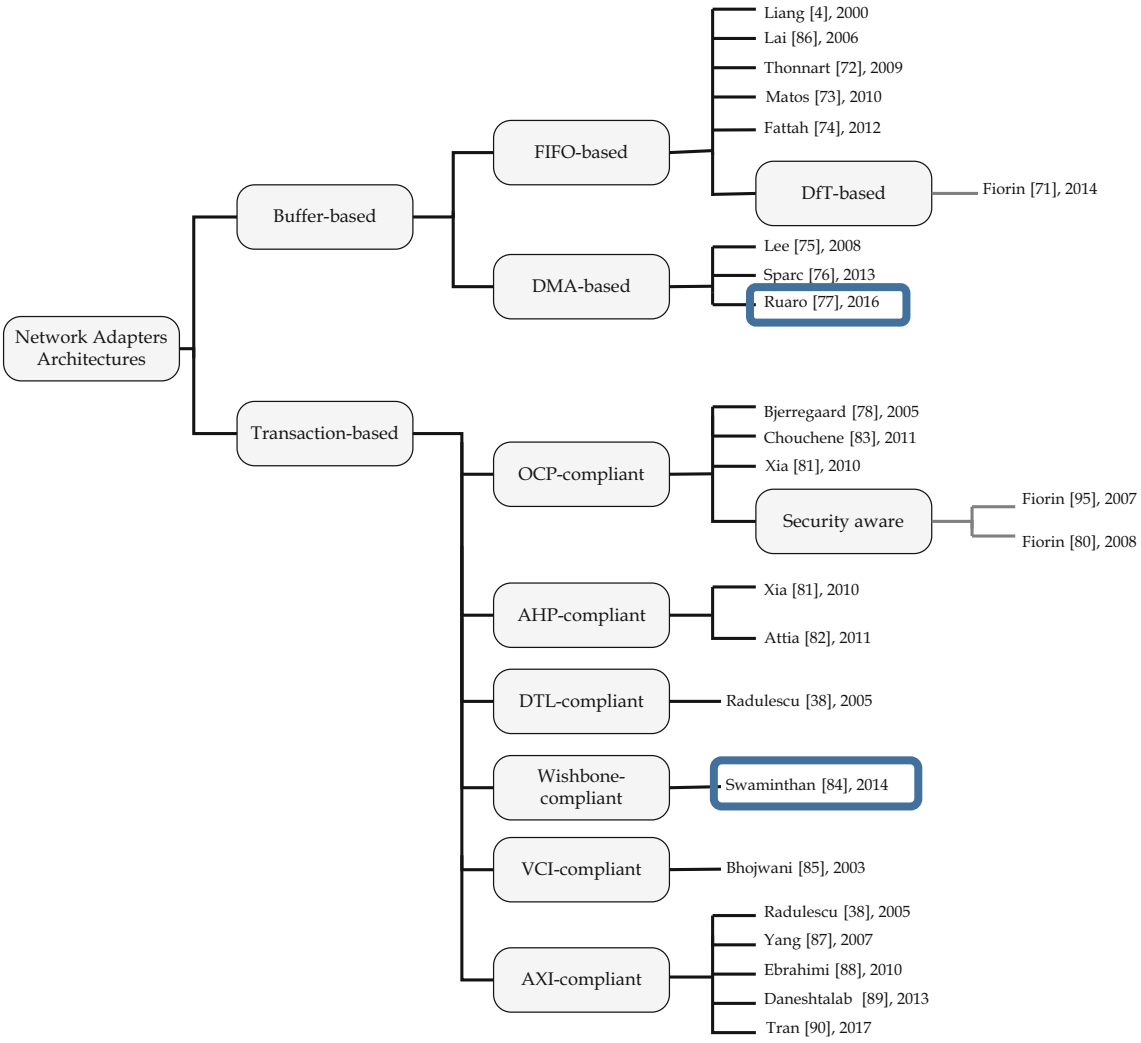


Figura 14 – Classificação de arquitetura de adaptadores de rede [AGH20].

Uma série de sugestões na área de pesquisa de adaptadores de rede é listada na conclusão deste artigo:

- Análise de comportamento de tráfego a possíveis ataques, e integração de um sistema de monitoramento com estratégias para detectar ameaças;
- Adaptação de *cores* para interface de protocolo padronizado;
- Tornar componentes da NoC testáveis e tolerantes a falhas;
- Sugestão de parâmetros para serem considerados no projeto e avaliação de NA;
- Explorar QoS em NI, sendo um tópico pouco abordado na literatura (segundo os Autores do artigo).

4.2 Network Interface Sharing for SoCs based NoC [ATT1]

Attia et al. [ATT11] propõe uma NI para NoCs, a fim de suprir a demanda por reutilização de *cores*. O trabalho aborda um conjunto de técnicas no projeto de NI para reduzir a utilização de área e o consumo de energia. As técnicas adotadas no projeto de NI

apresentadas pelo Autor foram implementadas utilizando o protocolo OCP (*Open Core Protocol*), porém estas podem ser estendidas para suportar outros protocolos como AXI (*Advanced eXtensible Interface*) e VCI (*Virtual Component Interface*).

O protocolo OCP possui três modos de operação de configuração de operações em rajada (*burst*): *precise burst*, *imprecise burst* e *single request multiple data burst*. A vantagem de se utilizar operações em *burst* é que a largura de banda é utilizada de forma efetiva, sendo necessário enviar apenas o endereço inicial e a quantidade de dados para iniciar uma transferência.

A fim de extrair dados comparativos de velocidade e energia foram implementados dois tipos de NI: *Master NI* (MNI) e *Slave NI* (SNI) (o texto descreve em detalhes apenas o funcionamento da MNI). A comunicação com a NoC também possui duas implementações, a primeira baseada em *handshake*, e a segunda baseada em créditos.

A principal função da MNI é receber as requisições do core IP e empacotar em formato de pacotes para ser transmitido na NoC. O inverso também é válido, isto é, receber as informações da NoC, desempacotar e enviar para o core no formato OCP. A principal diferença da SNI para a MNI é que a primeira não é capaz de iniciar as transações, atuando como escravo, ou seja, fornecendo respostas às transações de forma reativa.

A arquitetura da MNI é composta por dois caminhos de fluxo de dados, *request-dataflow* e *response-dataflow*, sendo que o primeiro possui três blocos de hardware para tratar os diferentes tipos de operações em burst (Figura 15).

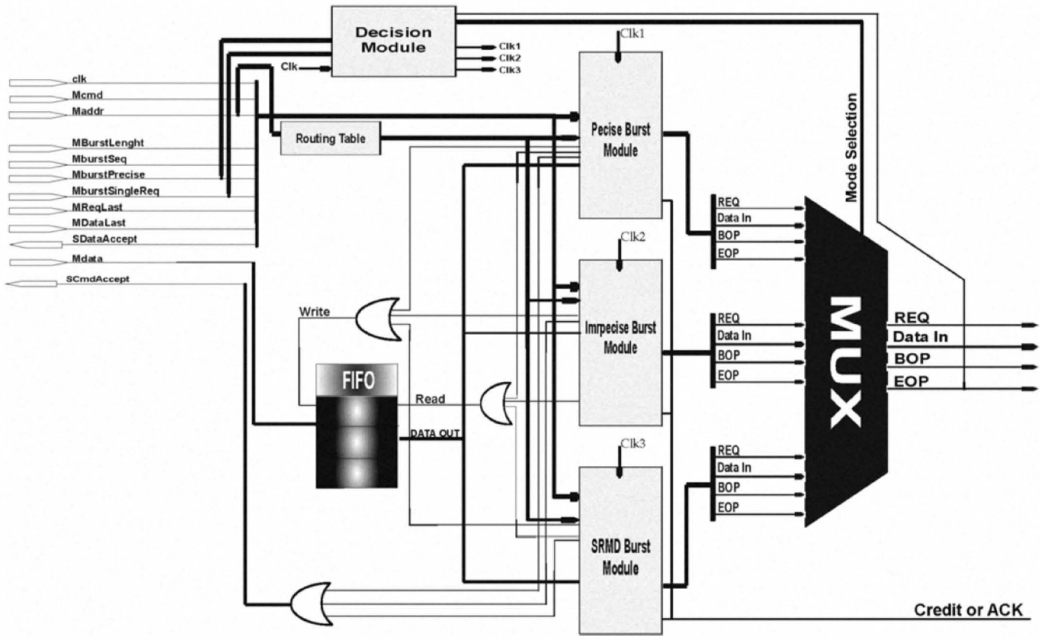


Figura 15 – Blocos de hardware para tratar diferentes tipos de operações em burst [ATT11].

Os módulos de hardware dedicados para o tratamento dos três tipos de operações suportadas pela interface de rede ocasionam o aumento do consumo de energia. A fim de controlar este aumento de energia, os autores propõem a utilização de técnicas de *clock-gating*, por meio de um módulo de decisão, que seleciona qual é o bloco de hardware que deve atuar de acordo com o tipo de requisição. Esta técnica permite controlar as transações

nas áreas ociosas do circuito, ativando-as de forma dinâmica através do controle do sinal de *clock*.

O bloco de hardware para tratar o tipo de operação *single request multiple data burst* (SRMD) é composto pelos módulos apresentados na Figura 16. Estes módulos gerenciam a aquisição de dados para escrita, construção do cabeçalho da mensagem e ainda controlam a transmissão do pacote para a rede NoC.

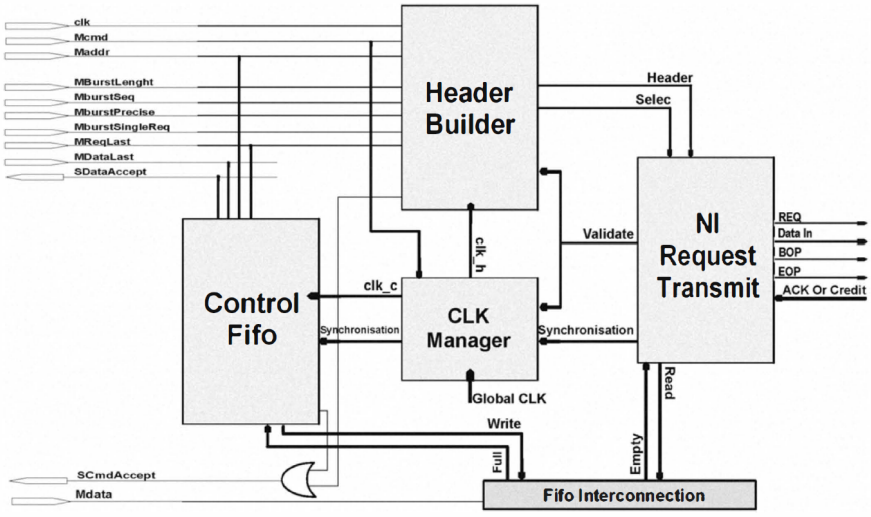


Figura 16 – Arquitetura do módulo de operações SRMD [ATT11].

Os resultados dos experimentos realizados pelos autores indicam que a área ocupada pela NI baseada em *handshake* é maior se comparada com a área ocupada pela NI baseada em crédito. A máquina de estados da NI baseada em *handshake* precisa de pelo menos o dobro de estados da outra, além de 4 ciclos de relógio adicionais para controle das quatro fases.

O consumo de energia da versão baseada em crédito é menor do que o da versão baseada em *handshake* para a interface de rede do tipo mestre (MNI). Para a interface do tipo escravo (SNI) o resultado de consumo de energia é muito semelhante.

Quanto aos resultados, o artigo menciona que a versão baseada em *handshake* suporta uma frequência de operação superior para interfaces de rede do tipo mestre, porém utiliza o dobro de ciclos de relógio em comparação a versão baseada em créditos.

4.3 Efficient security zones implementation through hierarchical group key management at NoC-based MPSoCs

Sepúlveda [SEP17] propõe uma interface de rede segura (SNI – *Segurity Network Interface*) para MPSoCs baseados em NoC, na qual seu principal objetivo é estabelecer uma comunicação segura entre as zonas seguras e IPs. A microarquitetura da SNI é apresentada na Figura 17.

A interface de rede é responsável pela autenticação, controle de acesso e confidencialidade, além de empacotar e desempacotar os pacotes transmitidos. A autenticação e o controle de acesso são implementados através de uma tabela de segurança (*security table* - Figura 17) enquanto a confidencialidade é realizada através da

criptografia de dados.

A tabela de segurança contém a informação de quais PEs são autorizados a se comunicar com a zona segura. Esta memória tem o seu conteúdo mantido pelo GMP, processador dedicado que executa um software embarcado.

Cada vez que um pacote chega na SNI as informações de roteamento do pacote, como origem e destino, são extraídas para verificação junto ao conteúdo armazenado na tabela de segurança. Caso o pacote seja transmitido de um PE não autorizado a transação é negada.

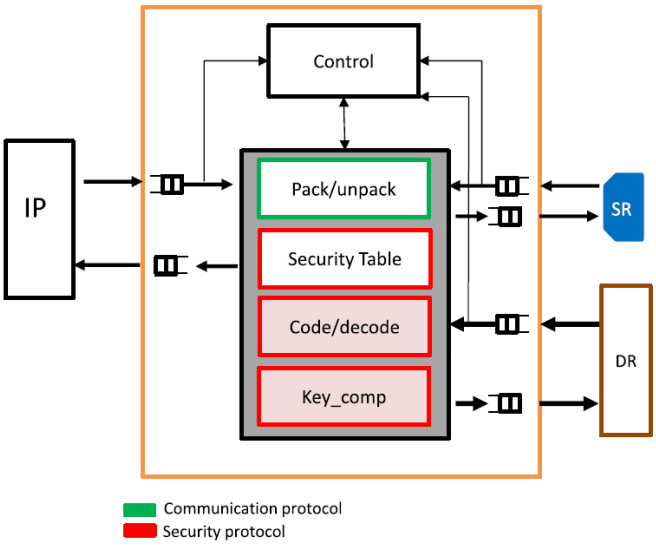


Figura 17 – Microarquitetura da SNI [SEP17].

O overhead de área adicionado pela SNI, em relação à implementação não segura, é de aproximadamente 9,2%, 9,8% de energia e 14,8% de latência para uma zona segura de tamanho 8, podendo chegar a 27,8% de energia e 40% de latência para zonas-seguras de tamanho 64.

4.4 Interface de comunicação extensível para a rede-em-chip SOCIN [MEL12]

A dissertação de mestrado [MEL12] consiste no estudo e implementação de uma NI para a NoC SOCIN. Na Figura 18 podemos observar que a NI (posicionada ao centro) está dividida em três partes: específica, genérica e rede.

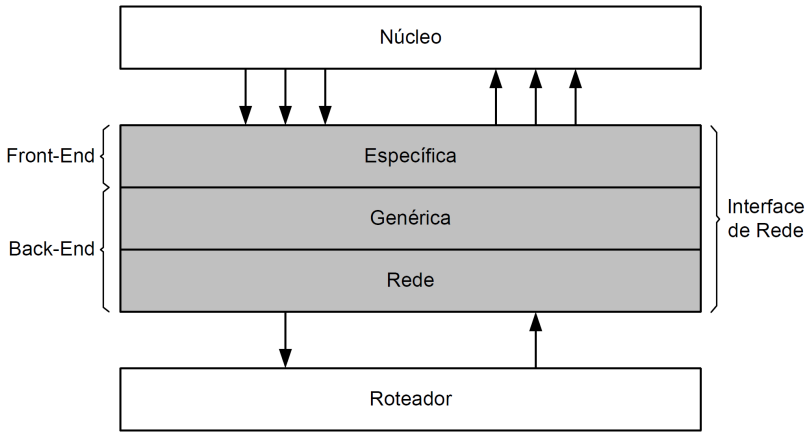


Figura 18 – Versão simplificada da NI proposta [MEL12].

O principal objetivo desta divisão é modularizar a implementação de protocolo, de forma a tornar a interface extensível para que seja possível suportar outros protocolos além do Avalon (protocolo utilizado pela Altera). A camada genérica é responsável pelo empacotamento e desempacotamento das mensagens, podendo atuar como mestre ou escravo. Na Figura 19 podemos observar uma CPU conectada a rede através de uma NI mestre, que é capaz de iniciar uma comunicação com os demais periféricos, que se comportam como escravos.

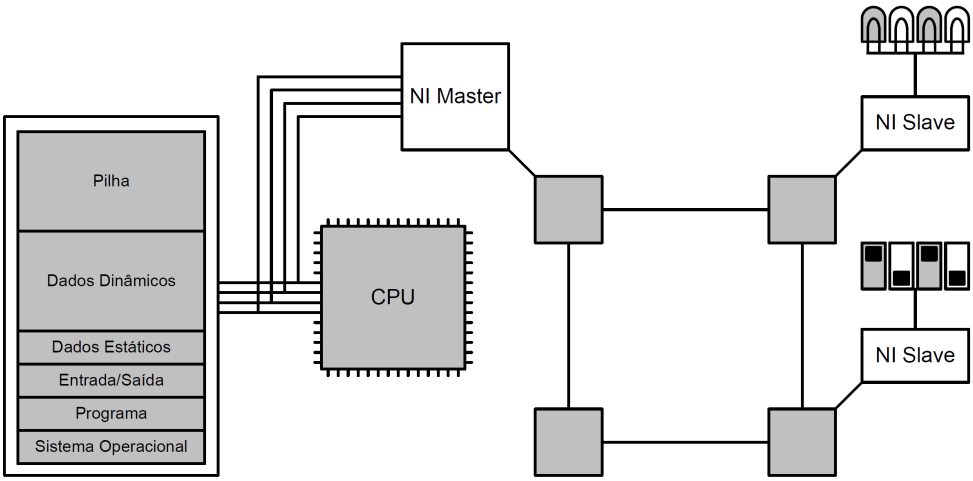


Figura 19 – Mapeamento de memória para comunicação com dispositivos externos em uma NoC [MEL12].

A Tabela 4 mostra o que está contemplado neste trabalho, e o que está previsto para trabalhos futuros. O serviço que contempla segurança é apresentado como trabalho futuro.

Tabela 4 - Serviços contemplados na versão atual da NI e previstos

Serviço	Camada	Versão Atual	Trabalhos Futuros
Adaptação ao barramento Avalon	Específica	×	
Adaptação a outros barramentos e padrões	Específica		×
Empacotamento de dados	Genérica	×	
Desempacotamento de dados	Genérica	×	
Diferenciação de classes de tráfego	Genérica	×	
Controle de integridade em nível de enlace	Rede	×	
Controle de integridade em nível de pacote	Rede		×
Codificação de barramento bus invert	Rede	×	
Controle de fluxo	Rede	×	
Transferência em rajadas	Rede		×
Reordenamento de pacotes	Rede		×
Segurança	Rede		×

4.5 On connecting cores to packet switched on-chip networks: A case study with microblaze processor cores [HOL04]

Holsmark et al. [HOL04] apresentam o desenvolvimento e prototipação de uma NI, sem requisitos de segurança, para NoCs. A interface de rede (BNI - *Bus-to-Network*) é responsável pela comunicação entre a rede (protocolo *Nostrum backbone* [MIL04]) e IP (protocolo OPB [IBM01]).

Na arquitetura da BNI, ilustrada na Figura 20, podemos observar máquinas de estados que controlam o fluxo de dados nos dois sentidos, rede e PE, fazendo o desempacotamento e empacotamento dos dados.

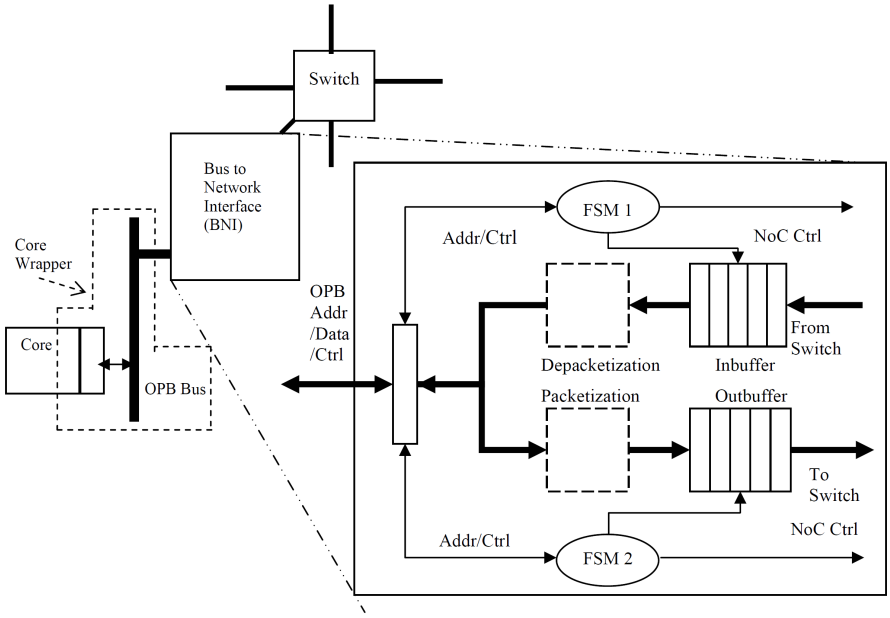


Figura 20 – Arquitetura da BNI [HOL04].

A prototipação foi realizada em uma plataforma Virtex II com FPGA e foram utilizados apenas 46% dos recursos de área disponíveis, sendo que a BNI ocupou apenas 1% do FPGA empregado pelo Autor. O desempenho foi medido em latência de um IP para outro passando por duas BNIs, resultando em 380 ns a uma frequência de 100MHz.

4.6 Network interface for NoC based architectures [SIN07]

Singh et al. [SIN07] propõem o desenvolvimento e prototipação de uma NI para conectar aceleradores de hardware a NoCs. Um *wrapper* genérico também foi desenvolvido e utilizado com um IP de compressão de imagens (JPEG – *Joint Photographic Experts Group*) a fim permitir a comunicação do IP e da NI através de um protocolo padrão (UART - *Universal Asynchronous Receiver Transmitter*). A implementação foi realizada em FPGA e os resultados mostram que a interface de rede adiciona um *overhead* de área mínimo no MPSoC, cerca de 48 *slices* e 57 *flip-flops* são adicionados para o *wrapper* do IP JPEG, 227 *slices* e 228 *flip-flops* para a interface de rede.

A arquitetura da NI (Figura 21) proposta por este trabalho possuí três tarefas:

- Abstrair o protocolo de comunicação com a rede, permitindo desenvolver esta parte de forma independente e tornando a interface genérica para qualquer IP.
- Tornar possível a comunicação com qualquer IP que implemente um protocolo específico através do uso do wrapper que permite a configuração de tamanho do bloco de dados.
- Empacotamento e desempacotamento das mensagens para envio e recebimento através da NoC.

Uma arquitetura de memória também foi desenvolvida com o objetivo de suportar IPs que trabalham com blocos de dados de tamanhos variados.

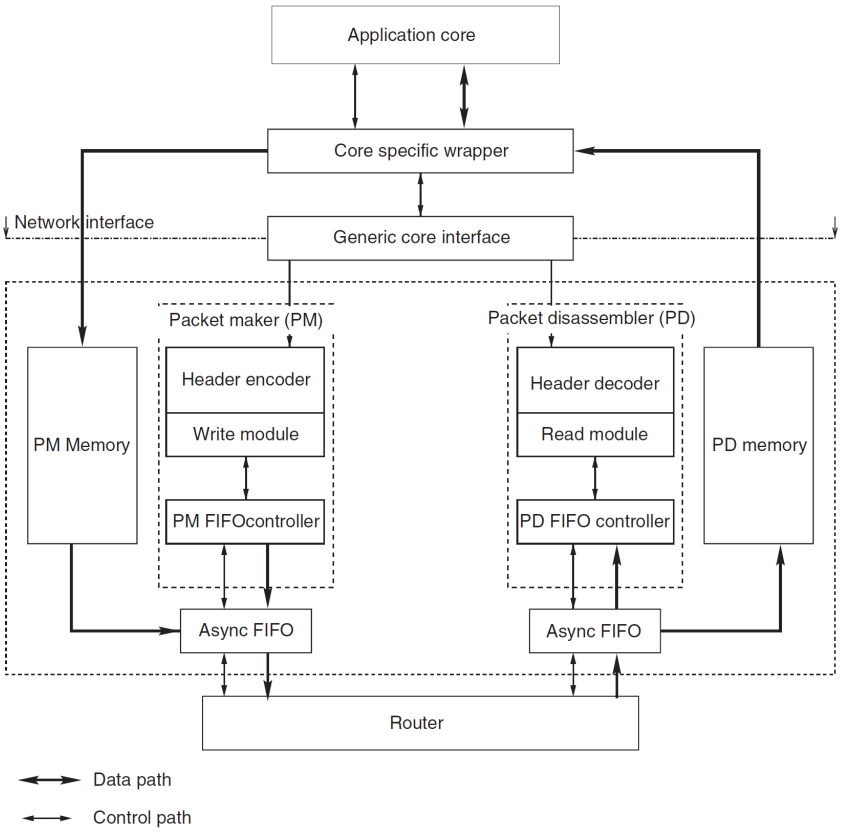


Figura 21 – Arquitetura da interface de rede [SIN07].

4.7 Considerações finais

A maioria das publicações revisadas visam o desenvolvimento de NIs para integração de IPs de terceiros, que podem ser utilizados como aceleradores de hardware, memória externa, entre outros. Devido a adoção por estes IPs de protocolos padronizados torna-se necessário a utilização de uma interface de rede compatível com o mesmo protocolo. Apenas o trabalho [SEP17] apresenta uma abordagem de NI segura.

Utilizamos os princípios de segurança mencionados no modelo de ameaças (Capítulo 2) para mapear o estado-da-arte em interfaces de redes para MPSoCs e propor uma interface de rede segura.

- **Autenticação** – Consideramos como autenticação uma verificação de autenticidade na comunicação estabelecida entre as entidades (PE e periférico) a fim de reservar o recurso, sendo assim outras mensagens não oriundas da entidade autenticada são ignoradas.
- **Integridade** – Assinatura que garanta que o dado no pacote não seja modificado.
- **Confidencialidade** – Criptografia dos dados sensíveis.
- **Elevação de privilégios** – A possibilidade de um IP de terceiro injetar pacotes malformados ou com conteúdo que possa comandar o MPSoC, atuando como mestre em um modelo de comunicação mestre-escravo.

- **Disponibilidade** – Critério justo para alocação de periféricos a fim de não permitir que um PE impeça os demais de utilizá-los. Também é considerado aqui um tempo de expiração (*timeout*) na resposta dos IPs a fim de evitar o bloqueio do PE que aguarda uma resposta.
- **Protocolo padronizado** – O suporte na interface a protocolos padrões de indústria para permitir a integração de uma variedade de IPs de terceiros que os implementam.

Em trabalhos mais recentes observamos a adoção de criptografia leve para comunicação *intra-chip* entre elementos de uma NoC, além de mecanismos para reduzir impacto adicionado pelo processo de criptografia, como por exemplo criptografia incremental [SUB20], que subdivide os dados em blocos menores e considera que será criptografado apenas a diferença entre blocos criptografados anteriormente, para que este modelo funcione é necessário uma memória para armazenar os blocos anteriormente criptografados. No trabalho [MES20] uma interface de rede é utilizada para desacoplar computação de comunicação, esta interface também é responsável por proteger os dados na comunicação, para isto uma série de algoritmos de criptografia são analisados, assim como AES - *Advanced Encryption Standard* e LED - *Light Encryption Device*.

5 INTERFACE DE REDE SEGURA PARA MPSOCS

A Figura 22 apresenta uma visão geral do MPSoC com os elementos que compõe a proposta da NI segura. A NI segura corresponde à principal contribuição desta Dissertação. O PE contém suporte de hardware e software para a comunicação com periféricos:

- Suporte de software: API de comunicação com funções para o suporte à comunicação com os periféricos e suporte no nível de *kernel* para realizar a comunicação com a NoC;
- Suporte de hardware: bloco de criptografia leve conectado à porta local do roteador que permite encriptar, ou não, os dados conforme os requisitos da aplicação.

A NI segura é uma implementação de hardware, que recebe pacotes de PEs, os interpreta, e os encaminha para os periféricos usando o protocolo *Wishbone*. Outros protocolos poderiam ser utilizados, porém o presente trabalho limitou-se a este protocolo, para fins de demonstração do método. O objetivo em se utilizar um protocolo padronizado é duplo. Primeiro, o uso de um protocolo padronizado permite a utilização de 3PIPs, sem necessidade de adaptação. Segundo, e mais importante, o protocolo interno do MPSoC não é exposto a terceiros, reduzindo a probabilidade de geração de ataques a partir de periféricos.

Adicionalmente, a NoC teve sua lógica de controle modificada para permitir a transmissão e a recepção de dados pelos roteadores de borda, e a integração com um *crypto core*.

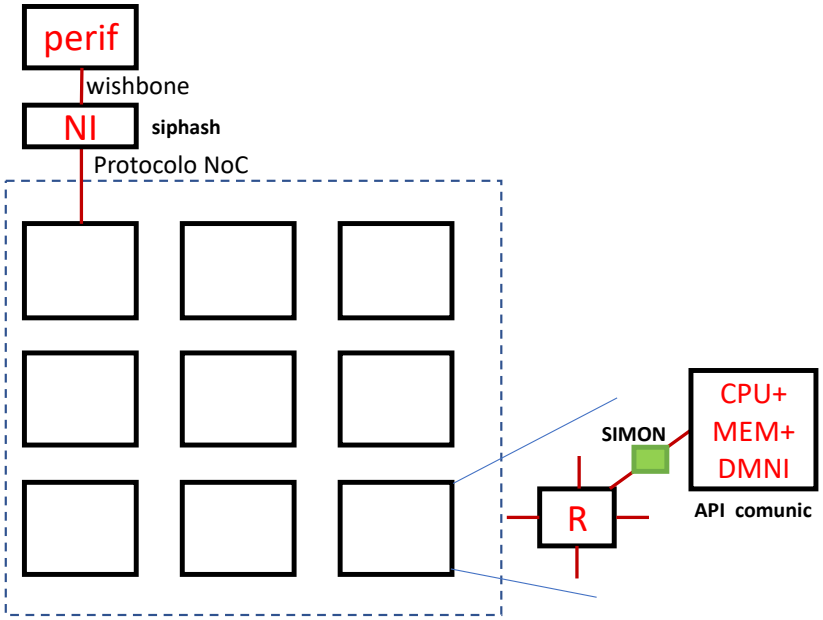


Figura 22 – Visão geral do MPSoC com os elementos que compõe a proposta da NI segura (Fonte: Autor).

Este capítulo está organizado em 4 seções: descrição dos componentes de software (5.1) e hardware (5.2) necessários à comunicação com os periféricos; exemplificação da interface de comunicação entre a NI e o periférico, utilizando o protocolo Wishbone (5.3); apresentação dos resultados (5.4).

5.1 Projeto da NI segura no MPSoC Memphis – Componentes de Software

Esta seção detalha o protocolo de comunicação com periféricos (5.1.1), o suporte no nível de aplicação (5.1.2) e no nível de *kernel* (5.1.3).

5.1.1 Protocolo de Comunicação com Periféricos

As aplicações que desejam comunicar-se com periféricos externos, como aceleradores de hardware conectados ao MPSoC através da NI, devem ser adaptadas para invocar as rotinas de sistemas desenvolvidas em nível de *kernel*. Justifica-se o desenvolvimento de uma API de comunicação com periféricos para prover a diferenciação do tráfego entre PEs e tráfego entre PEs e periféricos. Esta diferenciação permite ao *kernel* utilizar um protocolo distinto em relação ao protocolo de comunicação entre PEs, com transações adicionais que proveem segurança.

A API desenvolvida segue o modelo de comunicação mestre-escravo, sendo as transações sempre iniciadas pelos PEs. A Figura 23 ilustra, através de um diagrama de sequência, o protocolo de comunicação com periféricos para operação de escrita no periférico.

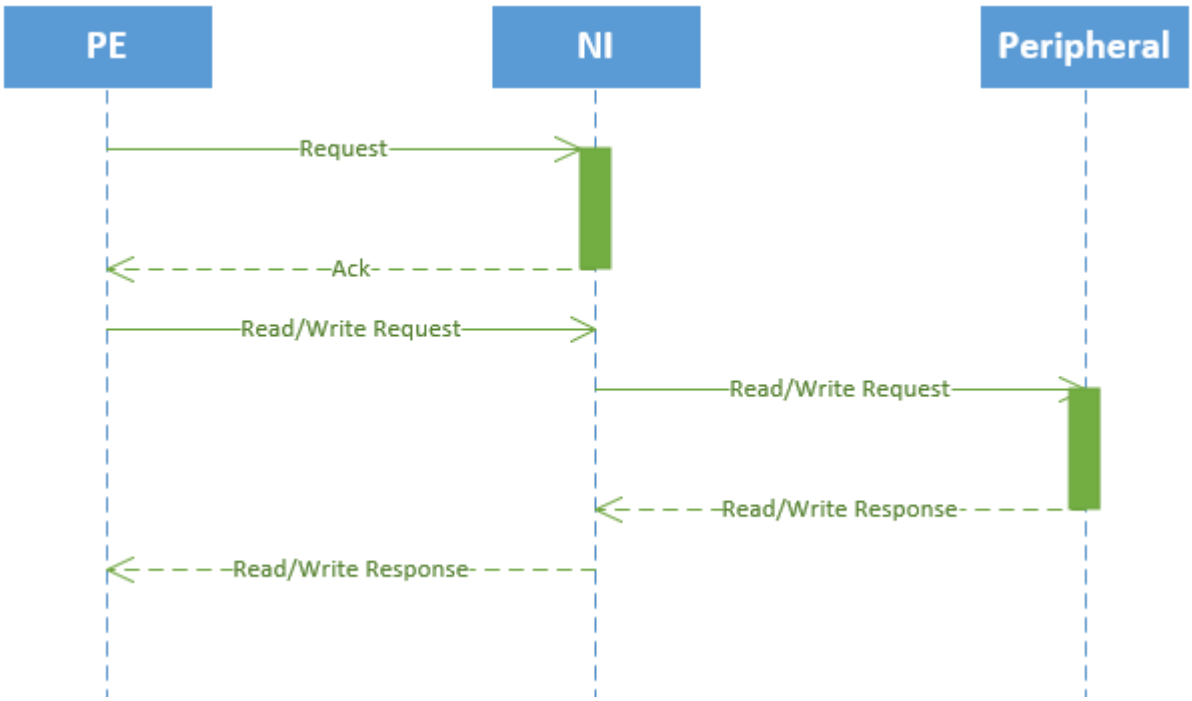


Figura 23 - Protocolo de comunicação com periféricos, para operação de escrita no periférico (Fonte: Autor).

Uma requisição de acesso é executa toda vez que uma aplicação deseja escrever ou ler de um periférico. Este mecanismo atua como controle de acesso, gerenciando as operações com o periférico quando múltiplas aplicações queiram utilizá-lo simultaneamente, evitando conflitos entre operações distintas e postergação infinita. Assim, a NI pode decidir, baseando-se em critérios de justiça a ordem em que as aplicações acessarão o recurso enviando sinais de ACK quando uma requisição é aceita e NACK caso contrário.

Após solicitar acesso ao periférico, o *kernel* aguarda a resposta relativa à requisição. Após a resposta positiva (ACK), é enviado no pacote de escrita (*write request*), o endereço de escrita e os dados. Caso seja uma operação de leitura (*read request*) é enviado o endereço para a leitura.

Após o envio da operação de escrita ou leitura o *kernel* aguarda a resposta. Para as escritas o retorno deve ser apenas um código de *status*, informando se a operação sucedeu ou falhou. Para as leituras são recebidos os dados juntamente com o status da operação.

A Figura 24 ilustra o cenário onde a NI responde que não pode se comunicar com o periférico. Dentre as razões para a negação à solicitação cita-se: (i) periférico já comunicando-se com outro PE; (*ii*) PE que realizou a solicitação não tem permissão de acesso ao periférico.

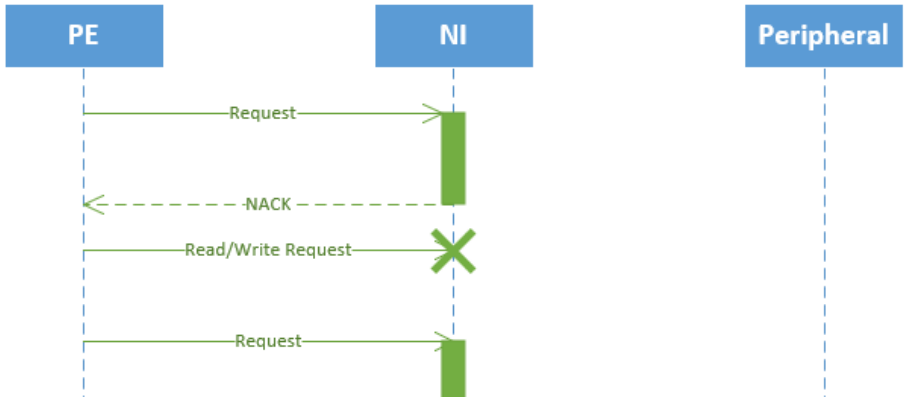


Figura 24 - Protocolo de comunicação com periféricos, com recebimento der NACK (Fonte: Autor).

5.1.2 Suporte de Comunicação no Nível de Aplicação

No nível de aplicação os detalhes de funcionamento internos da NI e dos periféricos são abstraídos, ou seja, a aplicação não conhece o protocolo implementado pelo periférico. A aplicação utiliza duas macros para a comunicação com periféricos, conforme apresentado no Código 1.

```
SendIO(result, &msg, WB_PERIPHERAL, write_address);
ReceiveIO(result, &msg, WB_PERIPHERAL, read_address);
```

Código 1 - Macros de comunicação com periféricos no nível de aplicação.

Onde:

- *result* – resultado da operação, indicando sucesso ou falha.
- *&msg* – ponteiro relacionado à estrutura de dados contendo o dado a ser enviado ou recebido do periférico;
- *WB_PERIPHERAL* – nome do periférico. O endereço do periférico é conhecido pelo *kernel*, evitando que aplicações tentem executar acessos diretos a periféricos inexistentes, o que poderia bloquear pacotes na rede.
- *write_address/read_address* – endereços base no periférico para ações de escrita ou leitura.

Estas duas macros invocam serviços de *kernel*, o qual implementam o protocolo descrito na Seção 5.1.3. Observar que estas macros são bloqueantes. A aplicação fica suspensa até que o *kernel* conclua a execução das mesmas.

5.1.3 Serviços de *kernel* para o Suporte ao Protocolo de Comunicação com Periféricos

As rotinas de sistema são implementadas no nível de *kernel* como forma de separar as responsabilidades entre as diferentes camadas do sistema. Neste caso, as aplicações ficam responsáveis apenas pela implementação lógica do programa, enquanto os detalhes internos de comunicação com os periféricos externos são de responsabilidade do *kernel*.

A macro *SendIO* (Código 1) gera uma interrupção para o *kernel*, através da chamada de sistema *SystemCall.* O Código 2 apresenta a expansão da macro *SendIO*. A chamada de sistema informa o código de interrupção, IOSEND, juntamente com as demais informações via registradores, como o endereço de memória onde está armazenada a mensagem, periférico de destino e endereço no periférico. Observe que o laço *while* só é concluído se a chamada de sistema retornar um valor diferente de zero, sendo este valor atribuído na variável *result*, em que a aplicação pode verificar o status da operação.

```
#define SendIO(result, msg, io_target, addr) while(!(result = SystemCall(IOSEND,
(unsigned int*)msg, io_target, addr)))
```

Código 2 – Expansão da macro *SendIO*.

A interrupção é acionada para atender a requisição de escrita ou leitura solicitada pela aplicação, e é neste momento que o *kernel* constrói os pacotes necessários para a execução do protocolo descrito na Seção 5.1.1. Sete serviços foram adicionados ao *kernel*, os quais geram pacotes conforme apresentado na Figura 25.

Flits / Services	1	2	3	4	5	6	7	8	9
Request	Target	Size	Service	Source ID	Task ID				
Request ACK	Target	Size	Service	Peripheral ID	Task ID				
Request NACK	Target	Size	Service	Peripheral ID	Task ID				
Read Request	Target	Size	Service	Source ID	Task ID	Read Address	Size	hash	
Read Response	Target	Size	Service	Peripheral ID	Task ID	Status	Size	hash	Data
Write Request	Target	Size	Service	Source ID	Task ID	Write Address	Size	hash	Data
Write Response	Target	Size	Service	Peripheral ID	Task ID	Status		hash	

Figura 25 – Estrutura dos pacotes relacionado aos serviços que suportam a API de comunicação com periféricos (Fonte: Autor).

Estes serviços compreendem:

1. *Request* – requisição de reserva de periférico;
2. *Request ACK* – reserva de periférico aceita;
3. *Request NACK* – reserva de periférico recusada;
4. *Write request* – requisição de escrita com o endereço inicial e dados em sequência;
5. *Write response* – resposta de escrita com o status que informa se ela ocorreu ou não;

6. *Read request* – requisição de leitura com o endereço alvo e tamanho;
7. *Read response* – resposta de leitura com o status e os dados em sequência.

Ao executar a chamada de sistema, o *kernel* solicita o acesso ao periférico por meio da função *send_io_request*, que é responsável por montar o conjunto de *flits* da mensagem conforme a definição de API utilizando as informações provenientes da chamada de sistema, *task_id, target_peripheral, requesting_pe*. Após a construção do pacote e envio dos *flits*, a tarefa é suspensa, i.e., não é escalonada novamente, já que é necessária uma mensagem de retorno para prosseguir com a operação. O atributo *waiting_msg* (verdadeiro ou falso) controla se o processo deve ser incluído no escalonador ou não (ver Código 3).

```
send_io_request(task_id, target_peripheral, requesting_pe);
scheduling_ptr->waiting_msg = 1;
```

Código 3 – Código de requisição de acesso ao periférico.

Quando a mensagem de aprovação de requisição (ACK) é recebida no PE, esta causa uma interrupção. No tratamento da interrupção, o *kernel* prepara a mensagem de leitura ou escrita. O Código 3 (parte da função *OS_InterruptServiceRoutine*) ilustra o tratamento do pacote de ACK.

```
case IO_REQ_ACK:
msg_ptr = (Message *)(tcb_ptr->offset | tcb_ptr->reg[3]);
addr = tcb_ptr->reg[5];
send_io_write(p->consumer_task, p->peripheral_id, p->requesting_pe, addr, msg_ptr);
```

Código 4 – Código de processamento do ACK na rotina de tratamento de interrupção.

O resultado da operação de escrita é recebido pelo PE de forma similar ao ACK, também com uma interrupção de sistema, porém agora o status da operação é retornado ao nível de aplicação por meio do banco de registradores e o processo é escalonado para que continue a sua execução. Observar no Código 5 que o atributo *waiting_msg* recebe agora o valor 0.

```
case IO_WRITE_RESPONSE:
tcb_ptr->reg[0] = 1;
tcb_ptr->scheduling_ptr->waiting_msg = 0;
```

Código 5 – Código de processamento da confirmação de escrita, na rotina de tratamento de interrupção.

5.2 Projeto da NI segura no MPSoC Memphis – Componentes de Hardware

A Figura 26 apresenta a NI, posicionada entre o roteador e o periférico. A NI comunica-se com a rede seguindo o protocolo baseado em créditos e com o periférico seguindo o protocolo Wishbone. Como trabalho futuro, a NI pode ser estendida para suportar outros protocolos, como OCP e AXI.

Esta Seção apresenta suporte a periféricos no nível da NoC (5.2.1), a integração do criptocore SIMON à NOC (5.2.2), e o projeto da NI (5.2.3).

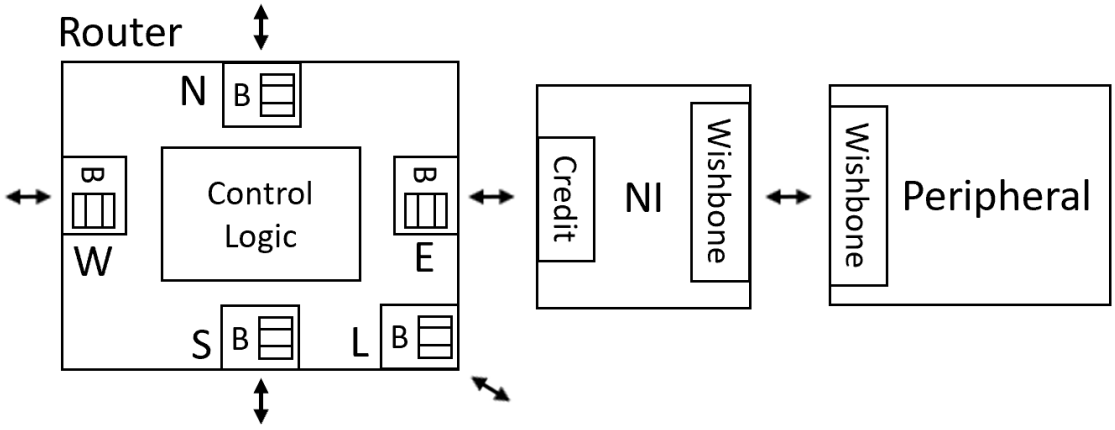


Figura 26 – Conexões da NI (Fonte: Autor).

5.2.1 Suporte a periféricos no nível da NoC

Na rede Hermes o protocolo de comunicação dos roteadores baseia-se em créditos, com o seguinte conjunto de sinais: RX (notifica dados a receber), TX (notifica dados a transmitir), CREDIT_I (notifica que o destino está pronto para receber, tem crédito) e CREDIT_O (notifica que a origem está pronta para receber, tem crédito). O sinal RX do roteador origem é ligado no TX do roteador destino, e o sinal CREDIT_I do roteador origem é ligado no CREDIT_O do roteador destino. Embora esteja implementado o conceito de empacotamento de mensagens, separando a comunicação da computação, o protocolo baseado em créditos utilizado nos roteadores limita o acoplamento de 3PIPs que adotam protocolos padronizados.

Os periféricos são conectados nas portas externas dos roteadores mais externos da rede. Observar que a conexão dos periféricos é realizada em portas que normalmente estariam desconectadas, como por exemplo a porta Norte dos roteadores localizados na parte superior da NoC. Tendo em vista a adição do suporte a periféricos foi necessário alterar a lógica de controle e a estrutura dos pacotes a fim de redirecionar o pacote à porta em que o periférico se encontra conectado, através da NI.

O Anexo 1 apresenta resultados relacionados à simulação da integração da NoC à NI, e da NI ao periférico Wishbone. Neste experimento foi desenvolvido um periférico somador de inteiros simples que se comunica através do protocolo Wishbone.

Os roteadores foram modificados de forma a permitir o encaminhamento dos pacotes aos periféricos. Três bits adicionais foram adicionados no cabeçalho da mensagem para definir se (*i*) o pacote deve ser encaminhado para um periférico conectado na borda da rede; e (*ii*) a direção na qual o pacote será encaminhado, seguindo em ordem norte (00), sul (01), leste (10), oeste (11). Para realizar esta função foi necessário apenas modificar o módulo *switch control*, conforme apresentado no Código 6.

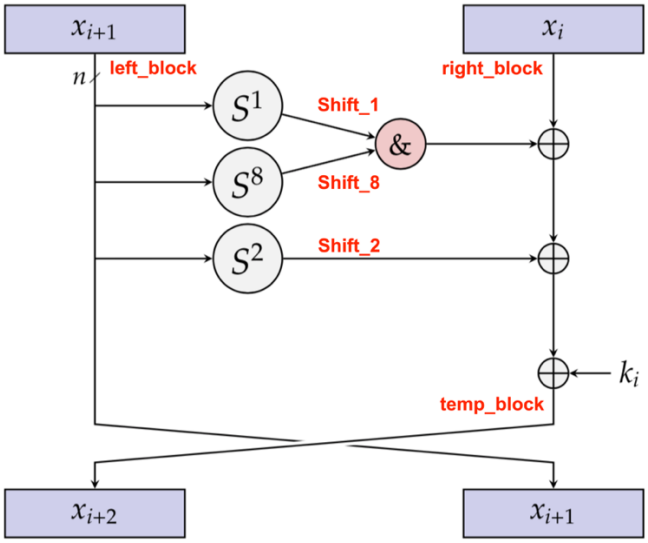
```
io_dir <= header(TAM_FLIT-1 downto TAM_FLIT-4);
dirx <= WEST when (lx > tx) or (io_dir(3) = '1' and io_dir(2) = '0' and io_dir(1) =
'1') else EAST;
diry <= NORTH when (ly < ty ) or (io_dir(3) = '1' and io_dir(2) = '1' and io_dir(1) =
'0') else SOUTH;
```

Código 6 – Código relativo ao encaminhamento de pacotes para periféricos.

5.2.2 Integração do criptocore SIMON

O algoritmo SIMON [SIL18] é simétrico, utilizando uma rede de Feistel [BEA15] para realizar a criptografia dos dados. Apesar deste algoritmo ser parametrizável, utilizamos no presente projeto chaves de 128 bits e blocos de 128 bits, que é a configuração recomendada para se obter um maior nível de segurança. Nesta configuração, os dados são gerados após 68 iterações (*rounds*) sobre a chave de entrada.

A Figura 27 ilustra o processo de geração dos dados, conforme a chave atual (*round_key,* k_i na Figura 27). O processo é iterativo, realizando-se as 68 rodadas de geração da chave de sessão, a partir da chave recebida. A geração das chaves de sessão também utiliza apenas deslocamentos e operações *xor.* Assim, o tempo necessário para criptografar ou descriptografar um bloco de 128 bits é de 70 ciclos de *clock*.




```
shift_1      <= left_block(63 downto 0) & left_block(63);
shift_2      <= left_block(61 downto 0) & left_block(63 downto 62);
shift_8      <= left_block(55 downto 0) & left_block(63 downto 56);
temp_block   <= round_key xor (shift_2 xor (right_block xor (shift_1 and shift_8)));
```

Figura 27 - Rede de Fiestel e código VHDL correspondente [SIL18].

No presente projeto foram adicionados dois blocos SIMON conectados à porta local do roteador. Desta forma, a interface externa do roteador não é alterada. O *payload* do pacote é criptografado ou descriptografado caso o bit mais significativo do pacote seja igual a ‘1’.

A Figura 28 apresenta a área de silício para o roteador Hermes (*flit*: 32 bits, profundidade do buffer: 8 *flits*), com os dois módulos SIMON, utilizando-se uma biblioteca de células com tecnologia 28 nm. Observa-se que o roteador requer 4.468 portas lógicas, enquanto cada módulo SIMON 2.570 portas lógicas (valor médio). É importante observar que o módulo de criptografia em si é pequeno, 1.370 portas lógicas (valor médio). Há um consumo de área no *wrapper* devido à necessidade de buffers de 128 bits para entrada e saída do SIMON, além de lógica de controle. Assim, a área do roteador, com dois módulos de criptografia leve teve um aumento de 115%. Este é um custo aceitável, por duas razões principais: (*i*) segurança agregada ao projeto; (*ii*) a área do roteador representa no máximo 10% da área do PE, sendo a maior parte da área do PE consumida pela memória. Assim, mesmo que a área do roteador dobre, isto representa para o PE um acréscimo de área de no máximo 10%.

```
============================================================
Generated by:           Genus(TM) Synthesis Solution 18.14-s037_1
Module:                 RouterCC_S
Technology libraries:   C28SOI
============================================================

Instance                    Module                  Cell Count  Cell Area
------------------------------------------------------------------------
RouterCC_S                                             9609      16570
   router             RouterCC_address                 4468       6248
   wrapper_out        simon_wrapper_                   2581       5199
     encription       simon                            1387       3147
   wrapper_in         simon_wrapper                    2560       5122
     encription       simon                            1365       3070
```

Figura 28 - Relatório de síntese do roteador Hermes com dois módulos SIMON (Fonte: Autor).

A Figura 29 ilustra o processo de envio de dados entre o roteador 0 (canto inferior esquerdo) e o roteador 8 (canto superior direito) em uma NoC 3x3. As marcações na Figura 29 indicam:

1. Injeção por parte do PE de um bloco de 128 bits (4 *flits*). Na primeira injeção de dados é também transmitido o cabeçalho do pacote, o qual não é criptografado. Observar que a cada 4 *flits* de *payload* o PE deve aguardar 70 ciclos de *clock* antes de transmitir os próximos 4 *flits*. Dado que utilizamos um processo de DMA para injeção dos dados na NoC, este tempo entre blocos de 4 *flits* não interrompe o processador.
2. Buffer que armazena 4 *flits*, de forma a gerar um bloco de 128 bits para o SIMON. A latência para preencher o buffer e realizar a comunicação com o SIMON é de 6 ciclos de *clock*.
3. Dado criptografado, injetado na rede.
4. Recepção dos dados criptografados, no buffer de entrada do SIMON.
5. Dados descriptografados, entregues para o PE destino.

A latência de rede é igual ao número de hops vezes 5, não havendo congestionamento de pacotes. Neste exemplo, temos 5 hops entre a origem e o destino, resultando em 25 ciclos de *clock*. Assim, acrescenta-se para cada bloco de 4 *flits* (128 bits) uma latência adicional, em relação à NoC de referência, de 152 ciclos (2*70 ciclos para o SIMON, e 2*6 para gerência dos buffers de entrada e saída).

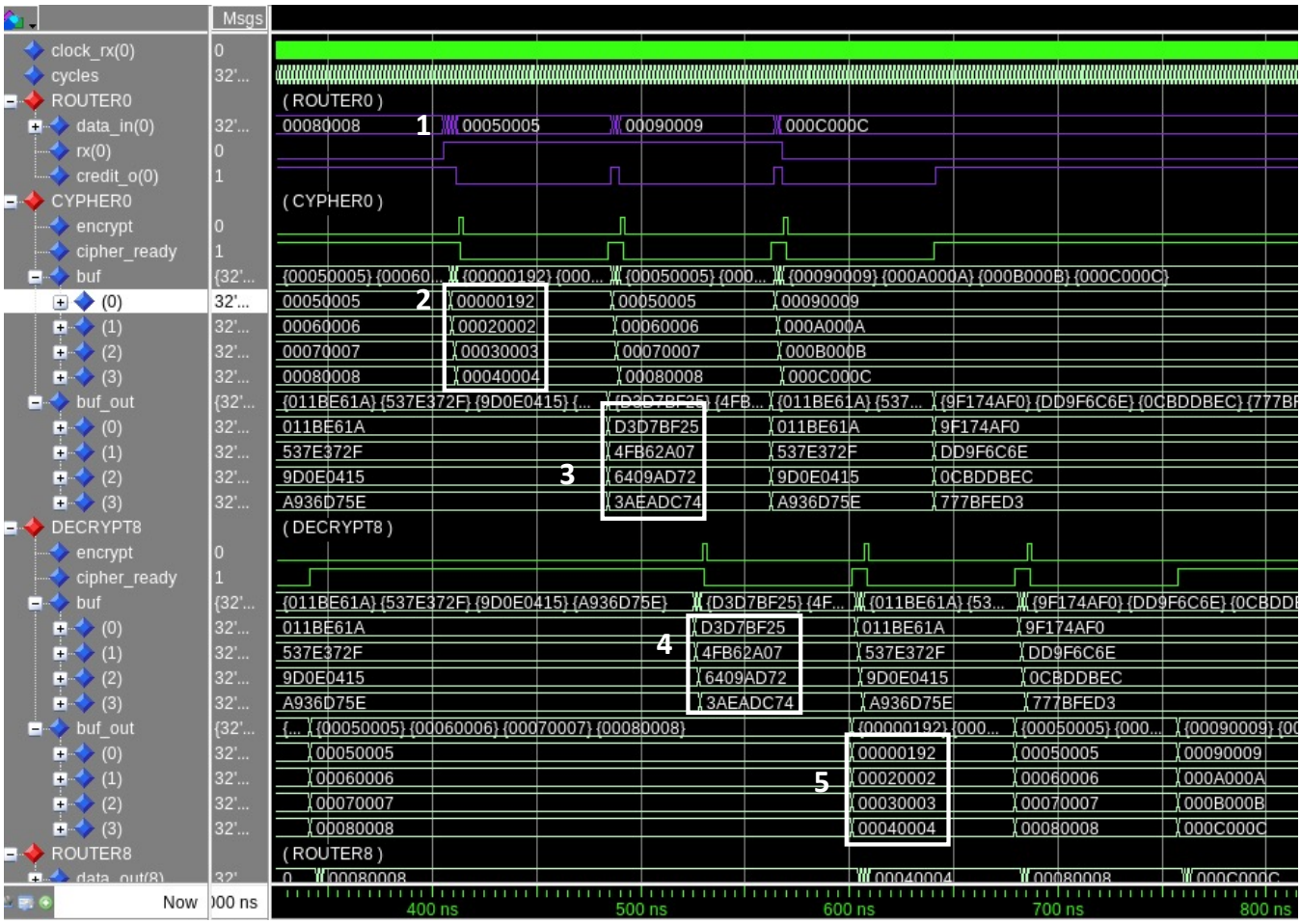


Figura 29 - Processo de criptografar e descriptografar os dados utilizando o algoritmo SIMON (Fonte: Autor).

5.2.3 Interface de rede

Esta sessão apresenta o projeto da NI. O desenvolvimento foi realizado em linguagem de descrição de hardware VHDL e a simulação realizada no simulador ModelSim. Como forma de organizar as funções da NI, a mesma foi dividida em quatro blocos principais, conforme ilustrado na Figura 30:

- *NoC interface:* envio e recepção de pacotes para/da NoC;
- *Packet processing*: processamento de pacotes oriundos da NoC;
- *IP interface*: comunicação com o periférico;
- *Packet analysis*: avalia as requisições enviadas à NI.

Uma memória interna, organizada como CAM (*content-addressed memory*), é responsável por armazenar as requisições realizadas para comunicação com os periféricos. Na Figura 30 ainda constam os módulos SipHash, responsável por autenticar os *flits* de controle dos pacotes, e o módulo SIMON. Este módulo SIMON é o mesmo que abordamos

anteriormente (Seção 5.2.2), mas no contexto da NI, ele atua criptografando os pacotes com destino ao periférico e descriptografando os pacotes de origem do periférico. Esta operação de criptografia é condicional, podendo ser configurada no header da mensagem em função dos requisitos da comunicação.

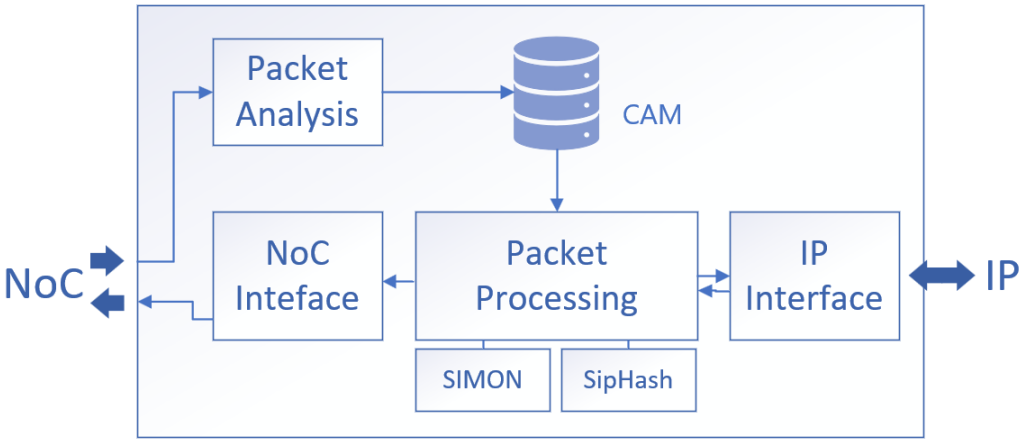


Figura 30 – Arquitetura geral da interface de rede proposta (Fonte: Autor).

A CAM armazena as requisições de reserva de periféricos. Esta memória possui dupla porta para que possam ocorrer operações de escrita e leitura simultaneamente pelas máquinas de estados apresentadas na Figura 31 e na Figura 32. Desta forma as requisições podem ser ordenadas de acordo com o algoritmo de seleção. Atualmente utiliza-se a ordem de chegada das requisições, de forma similar a uma fila do tipo *first-in-first-out*.

O processamento de pacotes oriundos da NoC é realizado por duas máquinas de estados, sendo a primeira responsável por armazenar as requisições, ilustrada na Figura 31. Esta máquina de estados pode recusar requisições caso a CAM esteja cheia, enviando uma mensagem de NACK como retorno. A segunda máquina de estados é responsável por desempacotar as mensagens da NoC e analisar as requisições para enviá-las ou não aos periféricos. Uma requisição pode ser negada caso o periférico esteja reservado a outro PE.

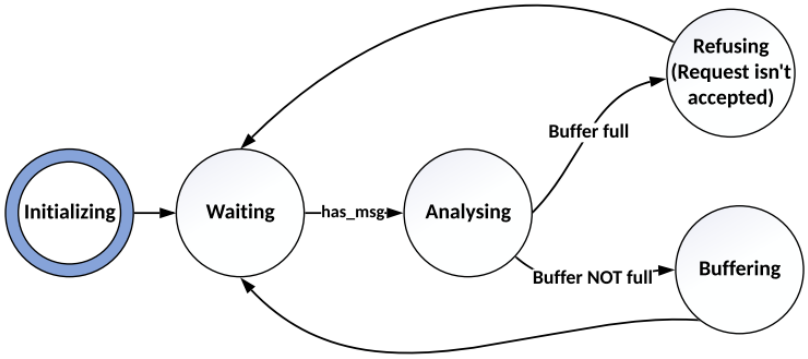


Figura 31 - Análise inicial e armazenamento das requisições, realizada no bloco *Packet Analysis* (Fonte: Autor).

O desempacotamento das mensagens oriundas da NoC é realizado pela máquina de estados ilustrada na Figura 32. A operação se inicia ao ler uma requisição da CAM, em

seguida, no estado *Accepting* a NI envia uma mensagem de ACK para a origem. Desta forma o PE solicitante é notificado de que o periférico foi reservado e está pronto para receber sua requisição de leitura ou escrita. Requisições de leitura ou escrita de outros PE não autorizados serão automaticamente ignoradas no estado *analysing*, em que é feito a verificação do *TaskID* e assinatura de mensagem com uma chave criptográfica, garantindo a autenticidade e integridade das mensagens. No estado *receiving* o conjunto de pacotes é lido da NoC e então a máquina de estados *IP interface* é ativada para propagá-los para o periférico seguindo seu protocolo padrão e fazendo o controle de forma apropriada de acordo com o tipo de operação, os dados são propagados ao periférico em operações de escrita. No estado *Responding*, em operações de escrita, a máquina de estados *NoC Interface* é ativada para enviar o status da requisição para o PE solicitante. Em operações de leitura, a máquina de estados *IP interface* atua em conjunto para receber os dados do periférico, cuja transmissão para NoC é coordenada pela máquina *NoC Interface*.

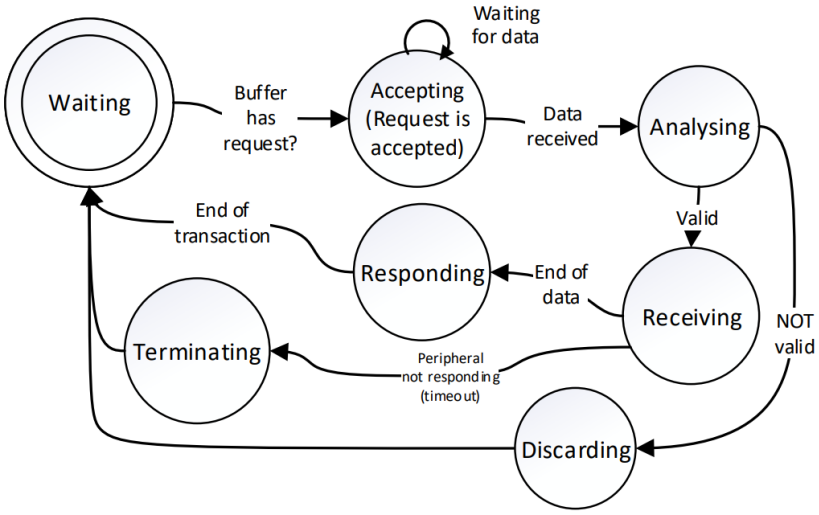


Figura 32 - Processamento das requisições de leitura e escrita, realizada no bloco *Packet Processing* (Fonte: Autor).

O envio de pacotes para a NoC é realizado pela máquina de estados ilustrada na Figura 33. Esta é responsável pelo empacotamento da mensagem e envio para a NoC utilizando o protocolo baseado em créditos. A máquina de estados passa de um estado ocioso para um estado ativo quando recebe algum comando proveniente da máquina de estados *Packet Processing*: *accept*, *refuse* ou *respond_write, respond_read*.

A comunicação com o periférico é realizada pela máquina de estados ilustrada na Figura 34. Ela é responsável por ativar os sinais do protocolo específico do periférico de acordo com a requisição, seja ela de leitura ou escrita (a Seção 3.3 descreve a operação do protocolo Wishbone). A máquina de estados passa de um estado ocioso para um estado ativo quando algum sinal de *send_write* ou *send_read* é ativado. Estes sinais representam que o barramento de dados possui um dado válido para ser enviado para o periférico, ou recebido do periférico. Na escrita no periférico, caso a transmissão de dados da NoC seja mais rápida do que o periférico pode processar, será realizada uma contenção dos dados através do sinal de crédito e enquanto o dado permanece válido à disposição do periférico a máquina de estados continua no estado de *sending_write*, caso contrário a máquina de estados entrará em estado de *waiting* notificando que o periférico está esperando os dados

da NoC. Na leitura, caso a transmissão de dados do periférico seja mais rápida do que a NoC é capaz de processar será realizada a contenção de dados mantendo o mesmo endereço de leitura para o periférico, porém interrompendo a transmissão à NoC, caso contrário, o estado de *waiting* será atribuído a máquina de estados notificando o periférico de que a rede está aguardando. Ao final dos ciclos de leitura ou escrita o estado de *cooldown* é atribuído a máquina de estados, este ciclo basicamente encerra a operação desativando os sinais *cyc_o* e *stb_o* do protocolo Wishbone.

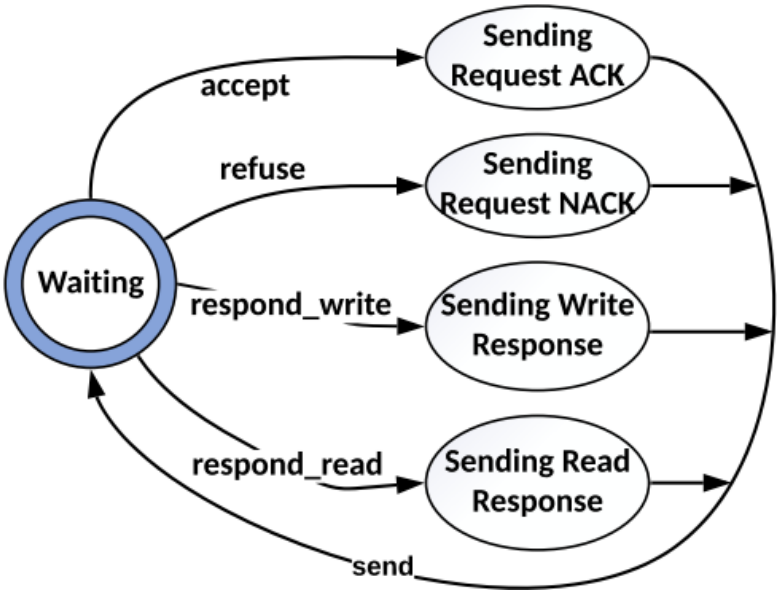


Figura 33 - Envio de respostas para a NoC, realizada no bloco *NoC Interface* (Fonte: Autor).

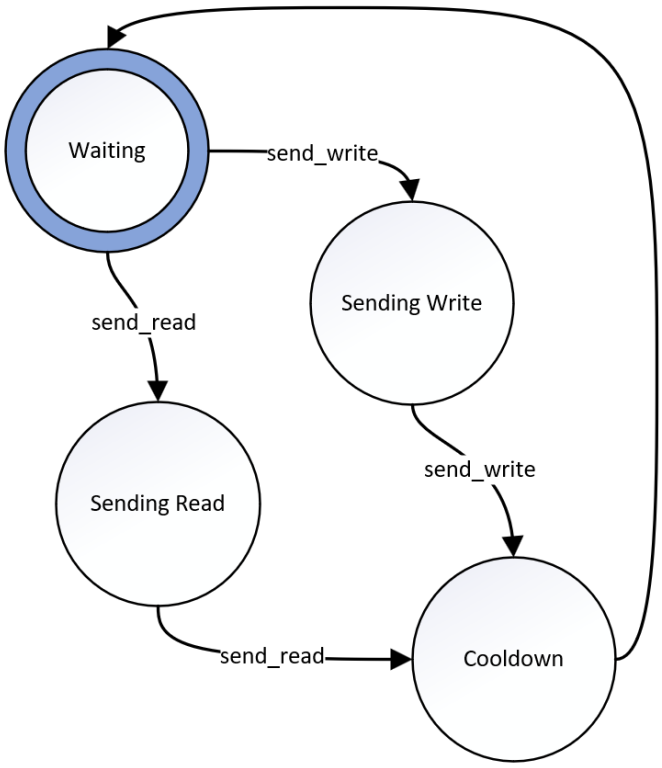


Figura 34 - Envio de dados para periférico, realizado no bloco *IP interface* (Fonte: Autor).

5.3 Periférico Externo

O Código 7 ilustra as portas de conexão da NI com a NoC e o periférico. Os sinais da NI estão agrupados em dois grandes grupos: interface com a NoC e interface com o periférico. Observa-se que o periférico possui apenas os sinais do protocolo Wishbone. Toda a tradução entre protocolos, controle de acesso, e tratamento de pacotes é função da NI. O periférico segue o protocolo Wishbone e o tipo de transação *classic cycle pipelined mode* (Seção 3.3).

```
network_interface : entity work.network_interface
port map (
       clock => clock,
       reset => reset,
       -- interface com a NoC
       rx => memphis_wb_peripheral_tx,
       tx => memphis_wb_peripheral_rx,
       credit_in => memphis_wb_peripheral_credit_o,
       credit_out => memphis_wb_peripheral_credit_i,
       data_in => memphis_wb_peripheral_data_out,
       data_out => memphis_wb_peripheral_data_in,
        -- interface wishbone
       per_clock => wb_clock,
       per_reset => wb_reset,
       address => wb_address,
       data_i => wb_data_i,
       data_o => wb_data_o,
       write_en  => wb_write_en,
       stb => wb_stb,
       ack => wb_ack,
       cyc => wb_cyc,
       stall => wb_stall
);

wb_memory : entity work.wb_256x2_bytes_memory
port map(
       clock => wb_clock,
       reset => wb_reset,
       adr_i => wb_address,
       dat_i => wb_data_o,
       dat_o => wb_data_i,
       we_i  => wb_write_en,
       stb_i => wb_stb,
       ack_o => wb_ack,
       cyc_i => wb_cyc,
       stall_o => wb_stall
);
```

Código 7 – Conexão do periférico (*wb_memory*) à NI (Fonte: Autor).

A descrição de hardware da memória externa consiste em um conjunto de sinais de controle do protocolo Wishbone que coordenam a escrita e leitura em um banco de registradores instanciados em VHDL na linha 2 do Código 8. O processo da linha 14 coordena o sinal de *write enable*, para que quando o sistema seja resetado a escrita fique desativada. O processo da linha 23 tem por função coordenar a operação de escrita no banco de registradores enquanto o processo da linha 41 tem como função coordenar a leitura. Podemos observar na linha 33 a atribuição do barramento de dados para o banco de registradores caso os sinais *stb*, *we* e *cyc* estejam ativos. A leitura ocorre na linha 47,

em que a posição do buffer especificada em *adr_i* é atribuída ao barramento de dados de saída.

```
01: type t_buffer is array (0 to MEMORY_SIZE-1) of std_logic_vector(DATA_LENGTH-1 downto
0);
02: signal buff : t_buffer := (others => (others => '0'));
03: signal tmp_data : std_logic_vector(DATA_LENGTH-1 downto 0);
04: signal tmp_adr : std_logic_vector(ADDRESS_LENGTH-1 downto 0);
05: signal s_ack_write : std_logic;
06: signal s_ack_read : std_logic;
07: signal s_we_i : std_logic;
08: signal s_dat_o : std_logic_vector(DATA_LENGTH-1 downto 0);
09:
10: dat_o <= s_dat_o when s_ack_read = '1' else, (others => '0');
11: ack_o <= '0' when stb_i = '0' or cyc_i = '0' else s_ack_write or s_ack_read;
12: stall_o <= '0';
13:
14: process(reset, clock)
15: begin
16:  if reset = '1' then
17:    s_we_i <= '0';
18:  elsif (rising_edge(clock)) then
19:    s_we_i <= we_i;
20:  end if;
21: end process;
22:
23: process(reset, clock)
24: begin
25:  if reset = '1' then
26:    s_ack_write <= '0';
27:    tmp_data <= (others => '0');
28:    tmp_adr <= (others => '0');
29:  elsif (rising_edge(clock)) then
30:    if stb_i = '1' and s_we_i = '1' and cyc_i = '1' then
31:      tmp_data <= dat_i;
32:      tmp_adr <= adr_i;
33:      buff(to_integer(ieee.numeric_std.unsigned(tmp_adr))) <= tmp_data;
34:      s_ack_write <= '1';
35:    else
36:      s_ack_write <= '0';
37:    end if;
38:  end if;
39: end process;
40:
41: process(reset, clock)
42: begin
43:  if reset = '1' then
44:    s_ack_read <= '0';
45:    s_dat_o <= (others => '0');
45:  elsif (rising_edge(clock)) then
46:    if stb_i = '1' and cyc_i = '1' and s_we_i = '0' then
47:      s_dat_o <= buff(to_integer(ieee.numeric_std.unsigned(adr_i)));
48:      s_ack_read <= '1';
49:    else
50:      s_ack_read <= '0';
51:    end if;
52:  end if;
53: end process;
```

Código 8 – Descrição do periférico memória (Fonte: Autor).

5.4 Resultados

Esta seção apresenta a operação da NI através de formas de onda geradas com o programa WaveDrown, análogas as formas de onda geradas com a ferramenta ModelSim, estas são formas de onda simplificadas, buscando mostrar ao leitor os principais sinais). Nos cenários apresentados a seguir, consideramos um MPSoC Memphis de dimensão 3x3. As requisições são enviadas do PE (0,0) para o periférico conectado na porta leste do PE (2,2). O periférico é uma memória que se comunica através do protocolo Wishbone. Observe que a NoC desconhece o protocolo do periférico, assim como o mesmo desconhece o protocolo interno da NoC. Cabe à NI a tarefa de comunicar as duas partes.

5.4.1 Solicitação simultânea e armazenamento na CAM

Este cenário consiste em enviar uma solicitação de acesso ao periférico (memória externa) para posteriormente realizar duas operações em sequência: escrita e leitura. A forma de onda da Figura 35 mostra os dados provenientes da NoC, as transações de estados do módulo *packet analysis* (Figura 30) e também a memória interna da NI, a qual nos referimos como CAM. O primeiro *flit* do barramento *Data from NoC* é o cabeçalho da mensagem (header), contendo o endereço de destino e tamanho da mensagem, que no caso é direcionada ao periférico. A seguir, no segundo *flit* a NI recebe o tipo de operação, neste caso trata-se de uma requisição de acesso (*request*, Seção 5.1.3). No terceiro *flit* a NI recebe o endereço de origem da mensagem e por fim, no quarto *flit*, a identificação da operação (TaskID, B) é recebida. Depois disto, este conjunto de informações da requisição é armazenado na memória CAM. Na Figura 35 podemos observar duas solicitações armazenadas na CAM, identificadas como A e B, as duas têm como origem o PE de coordenada X 0 e Y 0 e destino X 2 Y 2.

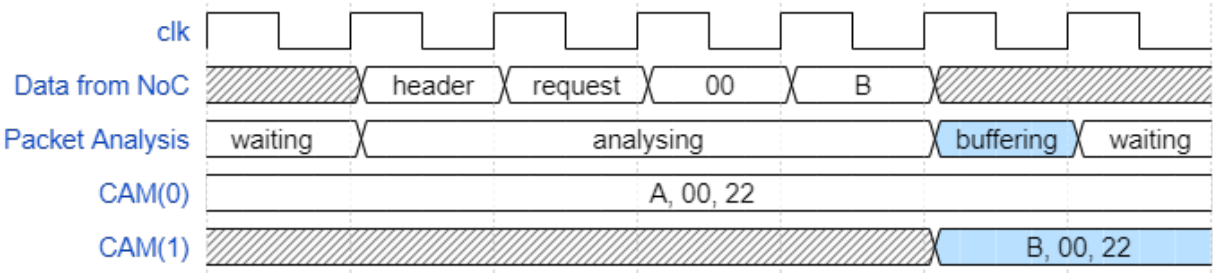


Figura 35 - Solicitação simultânea e armazenamento na CAM (Fonte: Autor).

As requisições salvas na CAM são solicitadas através de mensagens curtas que não passam por nenhuma verificação de segurança. Esta verificação é realizada posteriormente através de uma assinatura no cabeçalho. Uma vez que as requisições estão salvas na CAM é possível ordená-las utilizando algoritmos como Round-Robin ou FIFO e então atendê-las de forma justa, prevenindo o domínio do periférico por um único PE, e garantindo assim a disponibilidade para os demais.

5.4.2 Solicitação aceita

Na forma de onda da Figura 36 observamos a requisição de escrita da tarefa A sendo

aceita pela NI através das transações de estados do barramento *Packet Processing* (Figura 31). Neste processo, uma mensagem de ACK é enviada da NI para o PE (barramento *To NoC*), que a partir de então está autorizado a enviar os pacotes para realizar a operação de escrita no periférico.

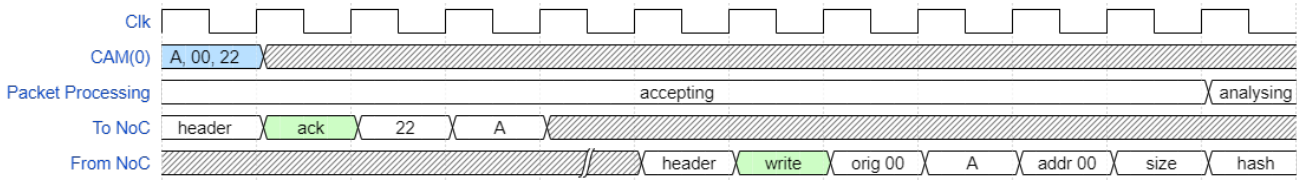


Figura 36 – Notificação de aceite para transação com periférico (Fonte: Autor).

Após o envio do ACK para o PE a NI se prepara para receber a operação de escrita. A identificação da operação é feita pelo TaskID e a requisição é removida da CAM uma vez que é atendida. A NI controla qual tarefa está permitida a utilizar o recurso externo e todas as requisições não autorizadas são recusadas.

No barramento *From NoC* observamos a sequência de *flits* que compõe a operação de escrita, cujos dados são enviados do PE para a NI: cabeçalho, tipo de operação, PE de origem, identificação da tarefa, endereço de escrita, tamanho da escrita e assinatura para verificação de integridade e autenticidade.

5.4.3 Solicitação negada

Na forma da Figura 37 uma nova requisição proveniente da *TaskID* C é negada pela NI pois o buffer se encontra cheio. Neste caso, é enviada uma mensagem de NACK (seção 5.1.3) para informar ao PE, que então pode estabelecer uma regra para solicitar novamente depois de um tempo. No estado *analysing* (Figura 31) a NI verifica se há espaço para guardar mais uma solicitação na memória CAM, transacionando para o estado subsequente apropriado, *refusing* ou *buffering.*

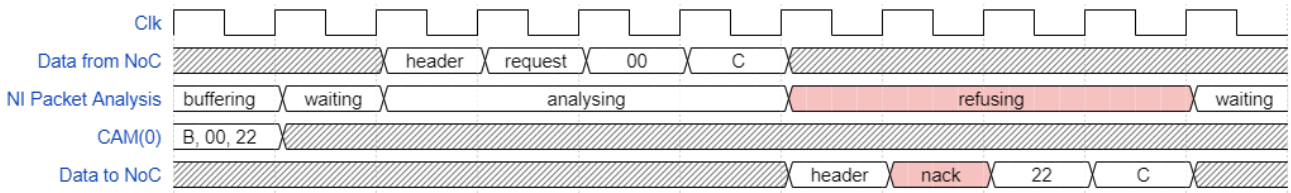


Figura 37 – Notificação de negação de acesso ao periférico (Fonte: Autor).

5.4.4 Operação de escrita com pacote autenticado

A Figura 38 ilustra a operação do módulo *packet processing* (Figura 32) ao aceitar uma requisição de escrita. Esta, transmite inicialmente o seu cabeçalho (header), seguido do tipo de operação (*write request*), endereço de origem (00), TaskID (A), endereço do início da escrita no periférico e assinatura para verificação de integridade e autenticidade. Como uma assinatura válida foi fornecida nesta simulação os dados serão propagados ao periférico. Podemos observar ainda que o módulo *IP interface* atua no controle dos sinais

do protocolo Wishbone para comunicação com o periférico através de uma operação do protocolo Wishbone do tipo *classic cycle pipelined mode* (Seção 3.3).

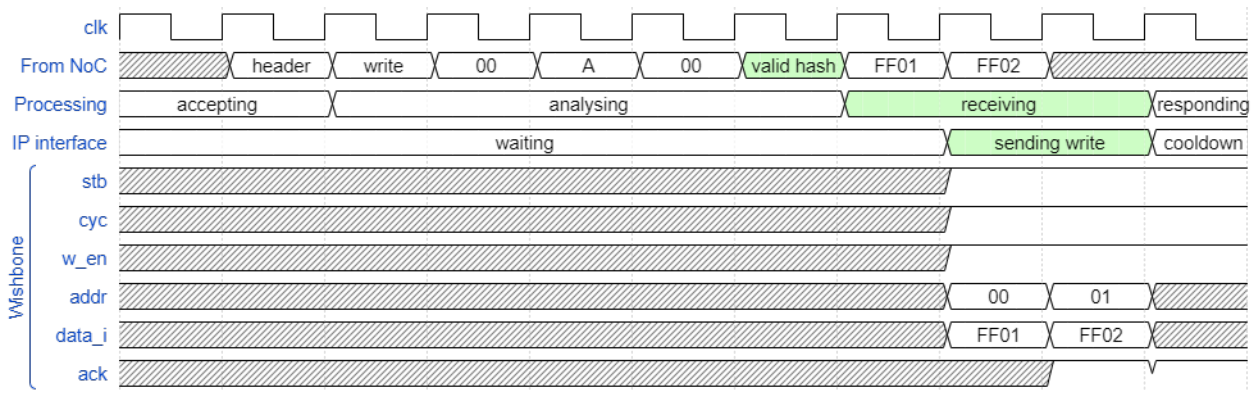


Figura 38 – Operação de escrita com pacote autenticado (Fonte: Autor).

A assinatura do cabeçalho é realizada com uma chave criptográfica simétrica através do algoritmo SipHash [AUM12]. Assim, apenas entidades em posse da mesma chave simétrica podem assinar e verificar a assinatura. Neste contexto, pode-se optar por distribuir diferentes chaves criptográficas para cada PE, ou ainda as distribuir na etapa de admissão de tarefa para que o conjunto de PEs de uma determinada tarefa possuam a mesma chave da NI.

As informações contidas no cabeçalho da mensagem têm sua integridade protegidas pois a verificação de assinatura não terá sucesso caso ocorra qualquer violação na mensagem no decorrer do caminho. Para estes casos a mensagem é descartada por inteiro. Ainda neste contexto, a mensagem também é descartada se assinada por uma chave diferente da esperada para a tarefa em execução prevenindo falsificação de identidade, já que cada mensagem é acompanhada de seu TaskID e é assinada com sua chave criptográfica, cuja assinatura é verificada pela NI.

5.4.5 Resultado da operação de escrita

Após a escrita no periférico, a NI informa o status da operação enviando uma mensagem do tipo *write response* (Seção 5.1.3) de volta para o PE que solicitou a escrita. A Figura 39 ilustra a transição de estados da máquina *Packet Processing* para coordenar o recebimento dos pacotes e da máquina *Send responses to NoC* para enviar o resultado. O resultado é enviado na forma de código de erro, em que 0 significa sucesso e qualquer valor diferente disto é entendido pelo PE como um código de erro.

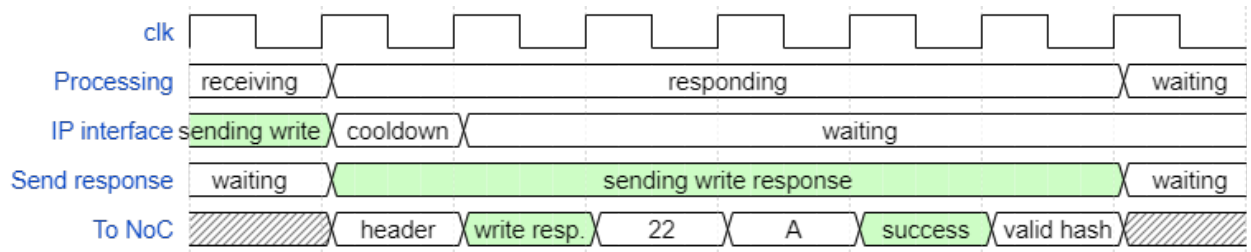


Figura 39 – Retorno da operação de escrita para o PE (Fonte: Autor).

A mensagem de retorno também é assinada, o que garante que ela foi gerada pela NI que possui a chave criptográfica, e que a mensagem não foi alterada por nenhuma entidade maliciosa durante o percurso.

5.4.6 Operação de escrita com pacote alterado

A Figura 40 ilustra a operação do módulo *packet processing* (Figura 31) ao descartar uma requisição de escrita que não passou pelo teste de integridade e autenticidade realizado na etapa de análise. A assinatura inválida pode ser resultado de uma violação dos dados da mensagem (integridade) ou também a assinatura realizada com uma chave que não corresponde a do TaskID informado (autenticidade).

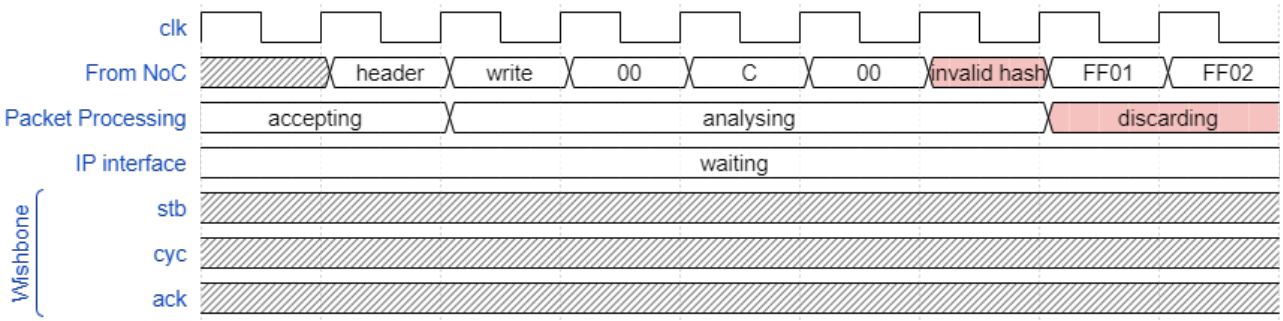


Figura 40 – Descarte de pacote (Fonte: Autor).

Na implementação atual o PE de origem não é informado quando a mensagem é descartada por problemas de autenticidade ou integridade, então para o mesmo seria como se a operação tivesse sido realizada com sucesso, já que a mensagem não está integra, não podemos confiar que a identificação de origem está correta para enviar um retorno.

5.4.7 Operação de leitura com pacote autenticado

As operações de leitura são análogas as de escrita, sendo que neste caso os dados retornam do periférico para o PE através da NI. Na Figura 41 observamos uma requisição de leitura, tarefa A, lendo a partir do endereço 00 e com a assinatura válida. A máquina de estados da NI processa a mensagem e estabelece a comunicação com o periférico utilizando o protocolo Wishbone. O periférico externo passa a enviar os dados enquanto a NI os empacota e injeta na rede com destino para o PE solicitante.

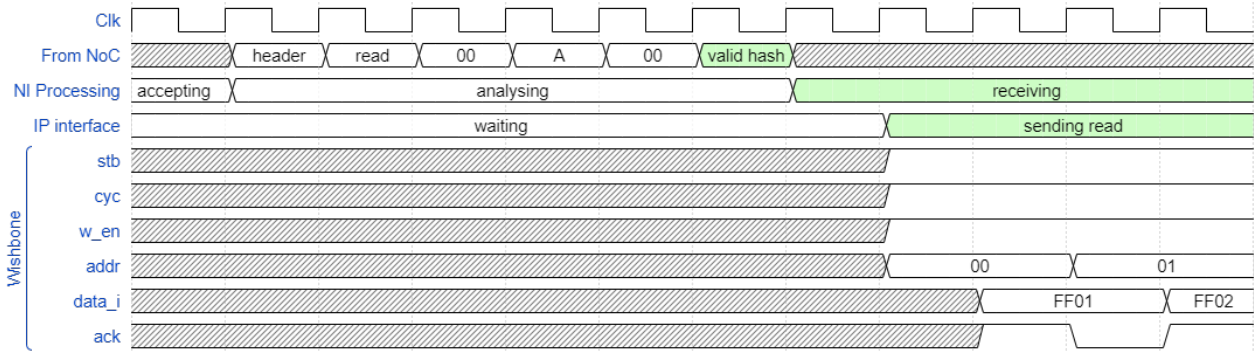


Figura 41 – Operação de Leitura (Fonte: Autor).

6 CONCLUSÃO E TRABALHOS FUTUROS

A presente Dissertação apresentou o projeto de uma NI capaz de abstrair o protocolo interno do MPSoC para os periféricos externos (formato dos pacotes, serviços definidos nos pacotes). Ao suportar um protocolo padrão permitimos a integração de IPs de terceiros (3PIPs), como aceleradores de hardware, simplificando assim a conexão de periféricos já desenvolvidos e disponíveis publicamente sem expor o protocolo interno do MPSoC, dificultando assim ataques que exploram vulnerabilidades no sistema forjando pacotes justamente por conhecer o protocolo interno.

A NI provê controle de acesso aos periféricos utilizando um modelo de comunicação mestre-escravo e alocação de recursos, i.e., todas as transações devem ser iniciadas por um PE, que solicita a utilização do recurso externo e somente mediante aprovação estabelece-se a comunicação entre as partes. Esta característica impede que periféricos iniciem ataques através da injeção de pacotes maliciosos na rede e impede que PEs não autorizados prejudiquem os demais na utilização de recursos externos. O uso de criptografia leve garante a confidencialidade dos dados em trânsito através da rede assim como os dados em memórias externas por meio de criptografia.

Ao decorrer deste trabalho foram levados em consideração cinco princípios de segurança para o modelo de ameaças, são eles: autenticação, autorização, integridade, confidencialidade e disponibilidade, para os quais foram propostos e desenvolvidos mecanismos de mitigação conforme os itens abaixo:

1. A injeção de pacotes por um periférico externo é inibida pelo modelo de comunicação adotado mestre-escravo em que a comunicação sempre se inicia pela NI, e é respondida pelo periférico, tornando-o incapaz de iniciar a comunicação por conta própria e injetar pacotes indesejados na NoC (a única exceção é o injetor de aplicações, considerado uma entidade segura).
2. Os acessos aos periféricos são controlados e ordenados pela NI, que verifica a origem e autorização de acesso através de criptografia simétrica.
3. As mensagens são assinadas com chaves criptográficas específicas de cada tarefa para proteger a integridade das mensagens e garantir a sua autenticidade, já que a assinatura só é válida se feita por entidades que possuem a chave criptográfica.
4. A confidencialidade dos dados em trânsito dos PEs para os periféricos é garantida pela integração de um módulo de criptografia leve na comunicação de ponta a ponta do PE ao periférico e vice-versa.
5. A confidencialidade dos dados em memórias ou buffers externos pode ser configurada por um bit no cabeçalho da mensagem. Se este bit estiver ativo, a NI armazena os dados criptografados, caso contrário os descriptografa antes de enviar ao periférico.

Em relação ao impacto de área adicionado pela NI, iniciamos a avaliação pelo módulo de criptografia leve, cujo acréscimo avaliado em área do roteador é de 115%, sendo aceitável por representar apenas 10% da área total do PE e agregar segurança ao projeto. Em trabalhos futuros pretende-se continuar esta avaliação para os demais módulos que

compõem a NI segura.

Além da avaliação dos impactos relacionados à avaliação PPA (*power, performance, area*) diversas outras frentes de pesquise podem ser elencadas:

- Efetiva integração da NI segura (componentes de hardware e software) à proposta de zonas seguras opacas;
- Inclusão do mecanismo de distribuição de chaves à NI. Implica em definir uma chave por NI, determinar quais aplicações usam o periférico, e distribuir esta chave ao *kernel* das tarefas que se comunicam com os periféricos;
- Incluir políticas de controle de acesso que evitem que PEs executando aplicações maliciosas executem ataques do tipo DOS;
- Avaliação da proposta da NI segura com periféricos de maior complexidade, avaliando-se a necessidade de se enriquecer a API de comunicação desenvolvida.

REFERÊNCIAS

[ACC13] Accellera Systems Initiative. *Open Core Protocol Specification - Release 3.0*. Capturado em: https://www.accellera.org/, 2021.

[AGH20] Babak Aghaei, Midia Reshadi, Mohammad Masdari, Seyed Hadi Sajadi, Mehdi Hosseinzadeh, Aso Mohammad Darwesh. *Network adapter architectures in network on chip: comprehensive literature review*. Cluster Computing, v.23(1), March 2020, pp. 321-346.

[ANC14] Dean Michael Ancajas, Koushik Chakraborty, Sanghamitra Roy. *Fort-NoCs: Mitigating the Threat of a Compromised NoC*. In: DAC, 2014, pp. 158:1-158:6.

[ARM13] ARM. *AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite*. Capturado em: https://developer.arm.com/documentation/ihi0022/e/, 2021.

[ATT11] Brahim Attia, Wissem Chouchene, Abdelkrim Zitouni, Rached Tourki. *Network interface sharing for SoCs based NoC*. In: CCCA, 2011, pp. 1-6.

[AUM12] Jean-Philippe Aumasson, Daniel J. Bernstein. *SipHash: a fast short-input PRF*. In: INDOCRYPT, 2012, pp. 489–508.

[BEA15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, Louis Wingers. *The SIMON and SPECK Lightweight Block Ciphers*. In: DAC, 2015, pp. 175:1-175:6.

[BJE06] Tobias Bjerregaard, Shankar Mahadevan. *A survey of research and practices of Network-on-chi*p. ACM Computing Surveys, v.38(1), June 2006, pp 1:1-1:51.

[CAI17] Luciano Caimi, Vinicius Fochi, Eduardo Wächter, Daniel Munhoz, Fernando Gehm Moraes. *Secure admission and execution of applications in many-core systems.* In: SBCCI, 2017, pp. 65-71.

[CAI19a] Luciano Caimi, Fernando Gehm Moraes. *Security in Many-Core SoCs Leveraged by Opaque Secure Zones*. In: ISVLSI, 2019, pp. 471-476.

[CAI19b] Luciano Caimi. *Secure Admission and Execution of Applications in Many-Core Systems*. Tese de Doutorado, PPGCC-PUCRS, 2019, 110p.

[CAR09] Everton Carara, Roberto Oliveira, Ney Calazans, Fernando Gehm Moraes. *HeMPS - A Framework for NoC-Based MPSoC Generation.* In: ISCAS, 2009, pp. 1345-1348.

[EIS07] Thomas Eisenbarth, Sandeep S. Kumar, Christof Paar, Axel Poschmann, Leif Uhsadel. *A Survey of Lightweight-Cryptography Implementations*. IEEE Design and Test of Computers, v.24(6), November-December 2007, pp. 522-533.

[GAP18] GAPH. *Hardware design support group*. Capturado em: www.inf.pucrs.br/gaph, 2021.

[GUE00] Pierre Guerrier, Alain Greiner. *A Generic Architecture for On-Chip Packet-Switched Interconnections*. In: DATE, 2000, pp. 250-256.

[HAN12] Neil Hanley, Máire O'Neill. *Hardware Comparison of the ISO/IEC 29192-2 Block Ciphers.* In: ISVLSI, 2012, pp. 57-62.

[HOL04] Rickard Holsmark, Alf Johansson, Shashi Kumar. *On connecting cores to packet switched on-chip networks – a case study with Microblaze processor cores*. In: CiteSeer^{X}, 2004, pp. 1-8.

[IBM01] IBM. *On-Chip Peripheral Bus - Architecture Specifications Version 2.1*. Capturado em: http://www.cs.columbia.edu/~sedwards/classes/2005/emsys-summer/opb_ibm_spec.pdf, 2021.

[KOH99] Loren Kohnfelder, Praerit Garg. *The threats to our products*. Capturado em: https://www.microsoft.com/security/blog/2009/08/27/the-threats-to-our-products/, 2021.

[KUM02] Shashi Kumar, Axel Jantsch, Mikael Millberg, Johnny Öberg, Juha-Pekka Soininen, Martti Forsell, Kari Tiensyrjä, Ahmed Hemani. *A Network on Chip Architecture and Design Methodology*. In: ISVLSI, 2002, pp. 117-124.

[MEL12] Douglas Rossi de Melo. *Interface de comunicação extensível para a rede-em-chip SOCIN*. Dissertação de Mestrado Acadêmico em Computação Aplicada, UNIVALI, 2012, 90p.

[MES20] Hassen Mestiri, Yahia Salah, Achref Addali Baroudi. *A Secure Network Interface for on-Chip Systems*. In: STA, 2020, 5p.

[MIL04] Mikael Millberg, Erland Nilsson, Rikard Thid, Shashi Kumar, Axel Jantsch. *The Nostrum Backbone: a communication protocol stack for Networks on Chip*. In: VLSID, 2004, pp. 693-696.

[MOO06] Gordon E. Moore. *Cramming more components onto integrated circuits, reprinted from Electronics, v.38(8), April 1965*. IEEE Solid-State Circuits Society Newsletter, v.11(3), September 2006, pp. 33-35.

[MOR04] Fernando Gehm Moraes, Ney Calazans, Aline Mello, Leandro Möller, Luciano Ost. *HERMES: An Infrastructure for Low Area Overhead Packet switching Networks on Chip*. Integration, the VLSI Journal, v.38(1), October 2004, pp. 69–93.

[OLI18] Bruno S. Oliveira, Rafael Reusch, Henrique Martins Medina, Fernando Moraes. *Evaluating the cost to cipher the NoC communication*. In: LASCAS, 2018, 4p.

[OST05] Luciano Ost, Aline Mello, José Palma, Fernando Gehm Moraes, Ney Calazans. *MAIA: a framework for networks on chip generation and verification*. In: ASP-DAC, 2005, pp. 49-52.

[RUA16] Marcelo Ruaro, Felipe B. Lazzarotto, César Marcon, Fernando Gehm Moraes. *DMNI: A specialized network interface for NoC-based MPSoCs*. In: ISCAS, 2016, pp.1202-1205.

[RUA19] Marcelo Ruaro, Luciano Caimi, Vinicius Fochi, Fernando Gehm Moraes. *Memphis: a framework for heterogeneous many-core SoCs generation and validation*. Design Automation for Embedded Systems, v.23(3-4), August 2019, pp. 113-122.

[SAN21] Anderson Camargo Sant'Ana, Henrique Martins Medina, Fernando Gehm Moraes. *Security Vulnerabilities and Countermeasures in MPSoCs*. IEEE Design and Test of Computers, v.38(4), August 2021, pp. 70-77.

[SCH17] Dimitris Schinianakis. *Alternative Security Options in the 5G and IoT Era*. IEEE Circuits and Systems Magazine, v.17(4), Fourthquarter 2017, pp. 6-28.

[SEP17] Martha Johanna Sepúlveda, Daniel Flórez, Vincent Immler, Guy Gogniat, Georg Sigl. *Efficient security zones implementation through hierarchical group key management at NoC-based MPSoCs*. Microprocessors and Microsystems, v.50, May 2017, pp. 164-174.

[SEP21] Martha Johanna Sepúlveda. *Secure Cryptography Integration: NoC-Based Microarchitectural Attacks and Countermeasures*. Network On-Chip Security and Privacy, Springer, 2021, pp. 153-179.

[SIL18] Cristovam Lage da Silva. *Design of a Lightweight Cryptography Module for FPGA based on Simon and Speck Algorithm*s. Trabalho de Conclusão de Curso, Engenharia de Computação, PUCRS, 2018, 36p.

[SIN07] S. P. Singh, S. Bhoj, Dheera Balasubramanian, T. Nagda, Dinesh Bhatia, Poras Balsara. *Network interface for NoC based architectures.* International Journal of Electronics, v.94(5), August 2007, pp. 531-547.

[SUB20] Subodha Charles, Prabhat Mishra. *Securing Network-on-Chip using Incremental Cryptography*. In: ISVLSI, 2020, pp. 168-175.

[SWA14] K. Swaminathan, G. Lakshminarayanan, Seok-Bum Ko. *Design and verification of an efficient WISHBONE-based network interface for network on chip*. Computers and Electrical Engineering, v.40(6), August 2014, pp. 1838-1857.

[WAC11] Eduardo Wachter. *Integração de novos processadores em arquiteturas MPSoC: Um estudo de caso*. Dissertação Mestrado, PPGCC-PUCRS, 2011, 92p.

[WEB20] Iaçanã Weber, Geaninne Marchezan, Luciano Caimi, César Marcon, Fernando Gehm Moraes. *Open-source NoC-based Many-Core for Evaluating Hardware Trojan Detection Methods*. In: ISCAS, 2020, 5p.

[WIS10] OpenCores. WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores. Capturado em: https://cdn.opencores.org/downloads/wbspec_b4.pdf, 2021.

ANEXO I

Neste anexo ilustramos os resultados do trabalho preliminar realizado como forma de familiarização do Autor com o sistema de rede NoC HERMES. O trabalho consiste no desenvolvimento de uma interface de rede que permita o acoplamento de periféricos Wishbone. Nesta simulação utilizamos a ferramenta Atlas para a geração de uma NoC de dimensões 3x3, com a rede foi configurada com o algoritmo de roteamento XY.

Na Figura 42 observamos uma mensagem que parte do roteador 00 para o 22 da NoC. O primeiro *flit* do último sinal, 0xC122 (1100 0001 0010 0010) indica o destino (0x22), que é um dado de periférico (terceiro byte igual a 1), e porta norte (0xC). Esta mensagem é composta por 17 *flits*, 2 para o cabeçalho e 15 para *payload*, que devem ser enviados ao periférico externo acoplado a rede (PER) através da interface de rede (NI).

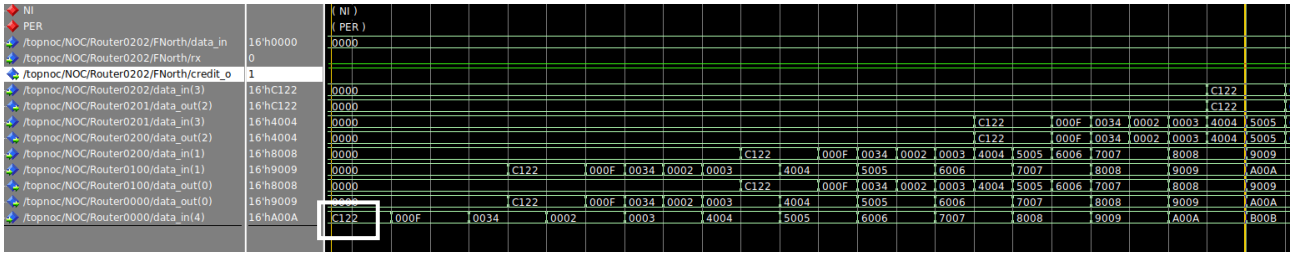


Figura 42 - Mensagem do roteador 00 ao 22 para escrita (Fonte: Autor).

A Figura 43 mostra o tratamento realizado pela interface de rede (NI) para traduzir a mensagem transmitida através da NoC para uma transação que segue o protocolo Wishbone. Como podemos observar, durante o recebimento dos dados o sinal de controle da comunicação da NI com a rede NoC (*interface_noc_state*) permanece com o valor *receiving*. Então após a transmissão do último *flit* de dados este estado se altera para *sending*, onde será realizada a comunicação com o periférico. Neste experimento a fim de simplificar a implementação reduzindo a complexidade de controle das interfaces todo o pacote foi armazenado em um buffer de entrada. Uma possível melhoria seria interpretar o cabeçalho do pacote e enviar os dados diretamente ao periférico sem mantê-los em memória. Na parte inferior da Figura 43 podemos observar ainda a comunicação com o periférico Wishbone. Esta comunicação é controlada por uma máquina de estados e podemos observar as transições de estado através do sinal *interface_wishbone_state*.

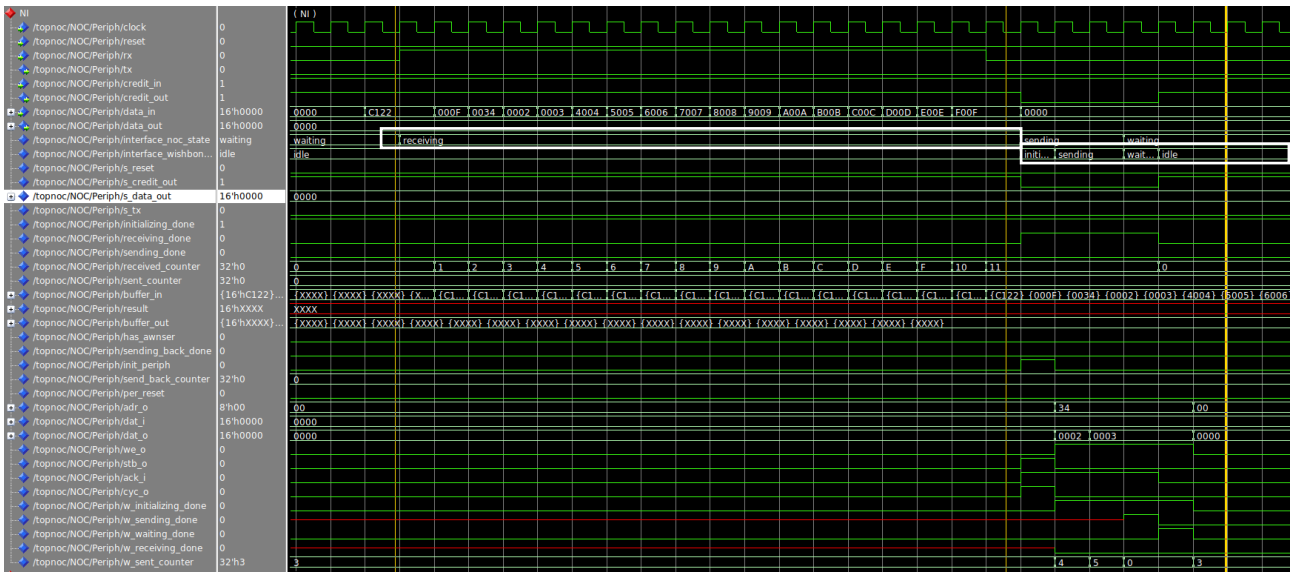


Figura 43 - Tratamento da mensagem pela interface de rede (Fonte: Autor).

No início da forma de onda apresentada na Figura 44 podemos observar o ciclo de configuração marcado pelo sinal *stb_i* (A) e seguido do sinal *we_i (B),* que configura o periférico para uma transação de escrita. Para fins de testes neste experimento foi desenvolvido um periférico simples, somador de 2 inteiros. No exemplo da Figura 44 os inteiros 2 e 3 (C) são enviados ao periférico.

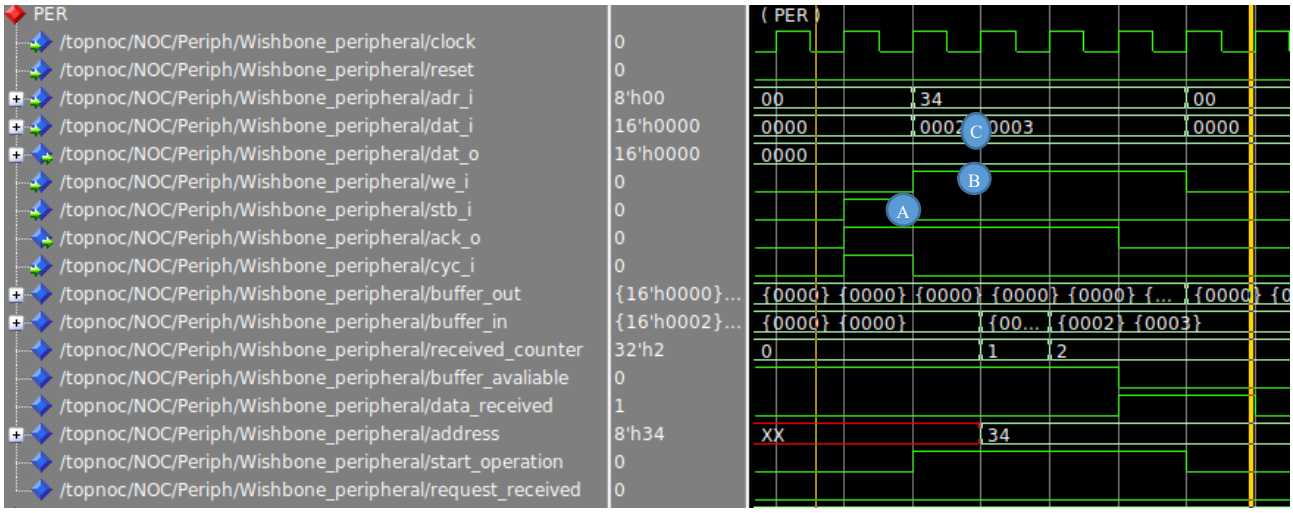


Figura 44 - Transação de escrita no periférico (Fonte: Autor).

A Figura 45 mostra uma mensagem de leitura que parte do roteador 00 para o 22. Esta leitura é realizada para capturar a resposta da operação de soma realizada anteriormente.

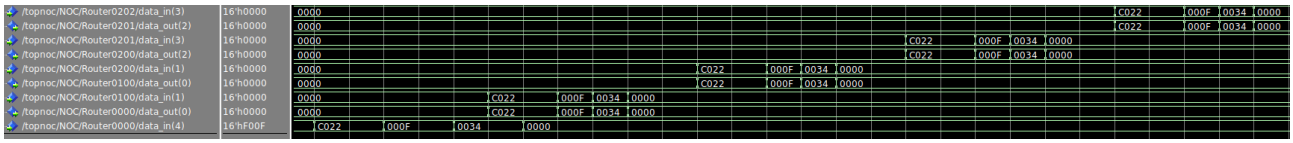


Figura 45 - Mensagem do roteador 00 ao 22 para leitura (Fonte: Autor).

A Figura 46 ilustra o tratamento da mensagem de leitura pela interface de rede, e as transições de estados necessárias para efetuar a comunicação com o periférico Wishbone.

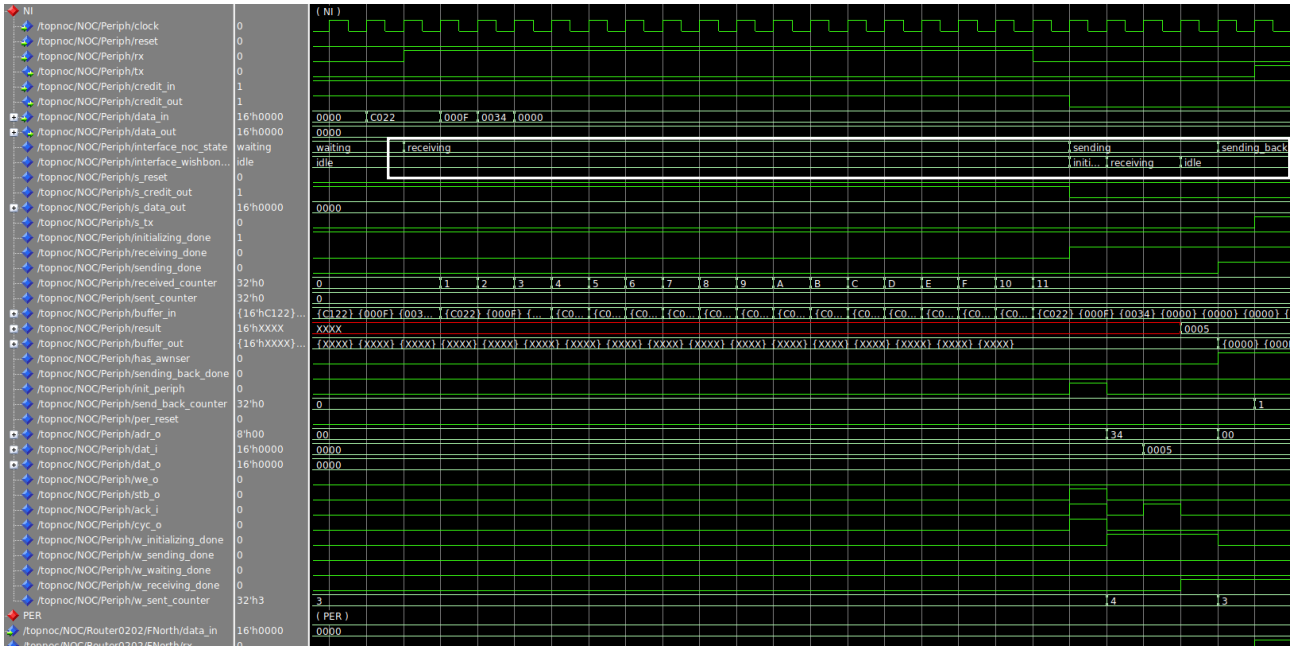


Figura 46 - Recebimento de mensagem de leitura e tratamento na interface de rede (Fonte: Autor).

O periférico Wishbone por sua vez recebe a transação de leitura e retorna o resultado da soma efetuada anteriormente através do barramento *dat_o* (Figura 47- A). Este periférico atua como uma memória externa, recebendo transações de escrita via protocolo Wishbone, armazenando o resultado em um banco de registradores e então respondendo a transações de leitura. Abaixo podemos observar o ciclo de configuração do protocolo Wishbone e o sinal *we_i* em nível lógico baixo (Figura 47- B), o que configura o periférico para uma operação de leitura.

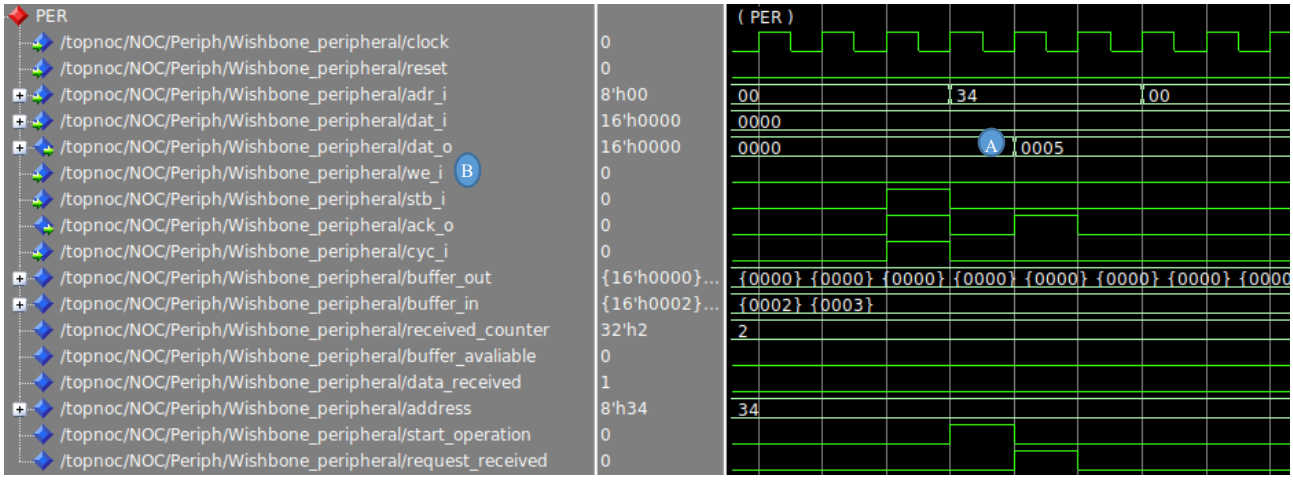


Figura 47 - Resultado calculado pelo periférico (Fonte: Autor).

Após receber o resultado da operação de leitura, a interface de rede (Figura 48) é responsável por empacotar o resultado (A-B) e enviar ao roteador origem, neste caso o 00.

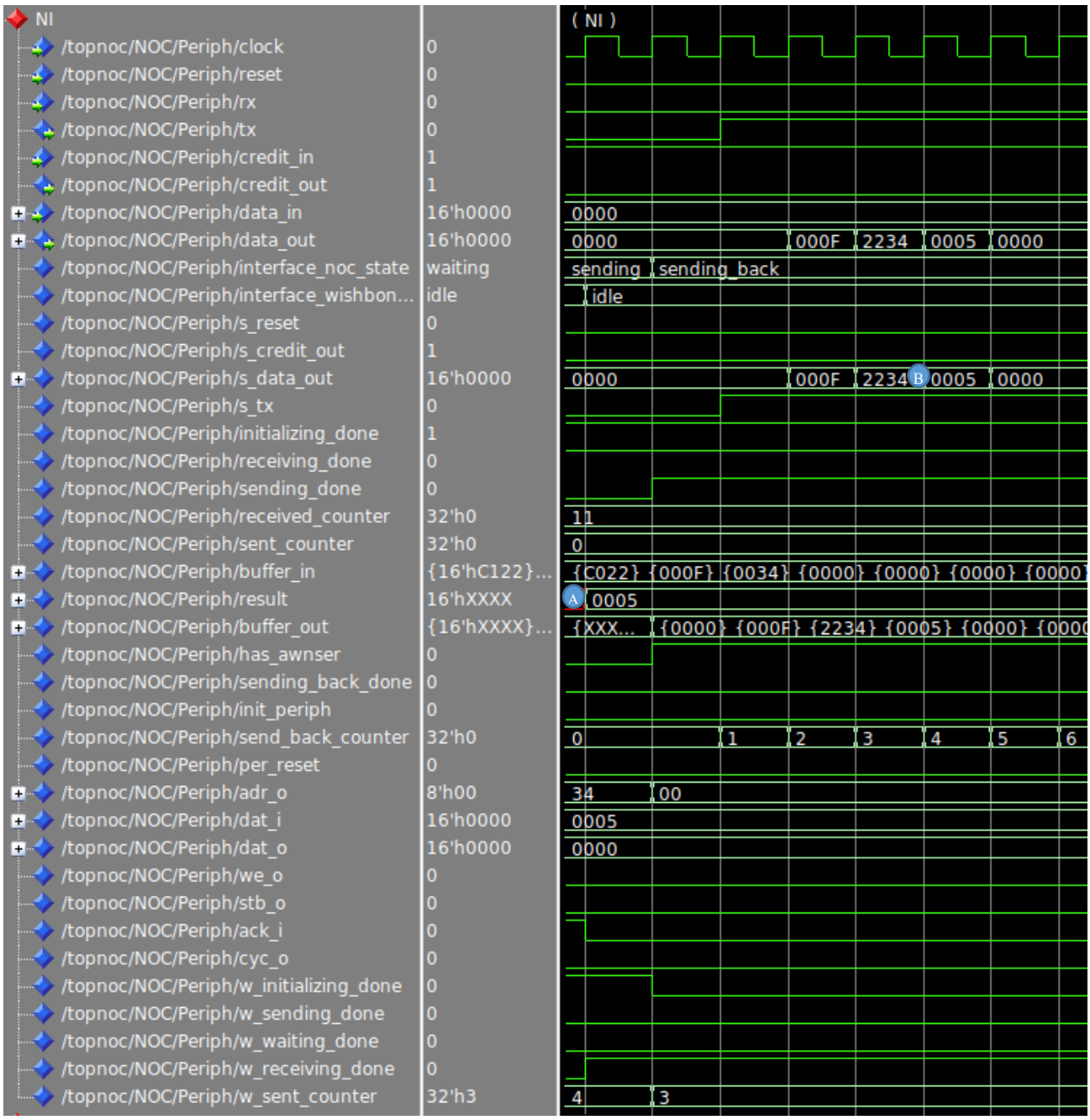


Figura 48 - Empacotamento do resultado (Fonte: Autor).

Como podemos observar na Figura 49, o pacote contendo a resposta da operação de soma efetuada anteriormente é enviado do roteador 22 para o 00, este que é a origem da operação.

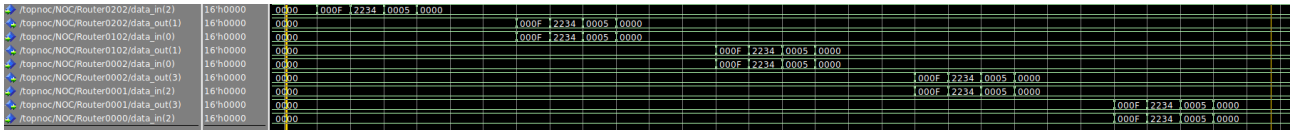


Figura 49 - Retorno do resultado ao roteador 00 (Fonte: Autor).

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br