UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOSÉ DIJON DE OLIVEIRA NETO

IMPLEMENTAÇÃO EM VHDL DA RPU DE IPNOSYS

MOSSORÓ
2017

JOSÉ DIJON DE OLIVEIRA NETO

IMPLEMENTAÇÃO EM VHDL DA RPU DE IPNOSYS

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Sílvio Roberto Fernandes de Araújo, Prof. Dr.

MOSSORÓ

2017

Ficha catalográfica elaborada pelo Sistema de Bibliotecas
da Universidade Federal Rural do Semi-Árido, com os dados fornecidos pelo(a) autor(a)

N469i Neto, José Dijon de Oliveira.
Implementação em VHDL da RPU de IPNoSys / José Dijon de Oliveira Neto. - 2017.
209 f. : il.

Orientador: Sílvio Roberto Fernandes de Araújo.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência da Computação, 2017.

1. Arquitetura de Computadores. 2. MPSoC. 3. NoC. 4. IPNoSys. 5. VHDL. I. Araújo, Sílvio Roberto Fernandes de, orient. II. Título.

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

JOSÉ DIJON DE OLIVEIRA NETO

IMPLEMENTAÇÃO EM VHDL DA RPU DE IPNOSYS

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência da Computação.

Defendida em: 22 / 05 / 2017.

BANCA EXAMINADORA

Sílvio Roberto Fernandes de Araújo, Prof. Dr. (UFERSA)
Presidente

Leonardo Augusto Casillo, Prof. Dr. (UFERSA)
Membro Examinador

Denis Freire Lopes Nunes, Prof. Me.
Membro Examinador

Ao meu pai, Francisco Belo de Oliveira Neto,
por me mostrar o caminho do bem e da caridade,
pela excelência na condução dos trabalhos,
pelas virtudes e pelas boas lembranças.
Você é o nosso exemplo!
Tenho orgulho de lhe chamar de Pai!

À minha mãe, Micheline Vieira Xavier de Oliveira,
por toda vida dedicada a minha existência,
por acreditar fielmente nos meus esforços,
pelo amor e carinho destinados a minha formação,
pelo apoio e compreensão em todos os momentos de minha vida.

AGRADECIMENTOS

Agradeço primeiramente ao Pai Celestial, pelo dom da vida e por tudo que tens me proporcionado.

Aos meus pais, Neto *(in memoriam)* e Micheline, serei eternamente grato por todo amor, ensinamentos, confiança e, acima de tudo, por terem acreditado em mim e na minha capacidade de conquistar os objetivos almejados. Meu pai, o senhor é o meu exemplo de sabedoria, de homem de bem, de pai e de amor pela família. Guardo em minhas lembranças o teu orgulho ao me ver apresentar este trabalho.

À minha irmã Michelle, por me dar forças nos momentos em que precisei e pela ajuda na revisão deste presente trabalho. Aos meus irmãos, Marília e Matheus, obrigada por me apoiarem em todos os momentos.

À minha namorada, Ana Flávia, pelo amor que transmites e por estar sempre ao meu lado. Você é, sem dúvida, a responsável pelo meu crescimento e amadurecimento.

Tenho gratidão e respeito, ao Prof. Silvio Fernandes, por dar a oportunidade de participar, mesmo que de forma voluntária, da iniciação científica cujo trabalho gerou diversos frutos e aprendizados. Agradeço, também, por aceitar ser orientador desse presente trabalho e conduzi-lo com muita sabedoria e paciência. Obrigado por mostrar tamanha atenção e compreensão diante a análise do referido trabalho.

Aos membros da banca, Leonardo Casillo e Dênis Freire, pelas dúvidas sanadas e auxílio dado diversas vezes, ao longo desta caminhada, bem como, por participarem da defesa e contribuírem com excelência para a melhoria do trabalho monográfico.

Aos amigos conquistados durante a graduação, dentre outros: David Anderson, Efraim Rodrigues, Iverton Perboyre, Alessandro Pino, Artur Afonso e, mais recentemente, Allef Schmidt e Arthur Aleksandro. Tenham certeza que vocês agregaram, e muito, neste ciclo que está se encerrando.

Essa conquista dedico a cada um de vocês!

RESUMO

Os avanços da microeletrônica possibilitaram incorporação de sistemas multiprocessados em um único chip, de modo a obter ganhos de desempenho através da exploração do paralelismo em diversos níveis. IPNoSys é um processador *multicore* com arquitetura baseada em redes-em-chip. O ambiente de programação e simulação deste processador permite executar aplicações reais, contudo, alguns resultados necessitam da descrição e síntese do *harware*. Diante do exposto, no presente trabalho buscou-se apresentar o projeto, implementação e testes, em VHDL, para o componente RPU de IPNoSys. A metodologia utilizada envolve a adoção de um estilo de desenvolvimento orientado a testes, que prioriza a criação de casos de teste antes da implementação. Testes automatizados, contendo aplicações sintéticas e reais (sequenciais e/ou paralelas), foram elaborados para verificar a correta implementação dos componentes internos da RPU e da matriz 2D de RPUs. Os resultados apresentados pelas simulações revelam ganhos em produtividade, qualidade de código, modularidade e menor propensão a erros.

Palavras-chave: Arquitetura de computadores. MPSoC. NoC. IPNoSys. VHDL.

ABSTRACT

The advances in microelectronics enabled the incorporation of multiprocessor systems into a single chip, in order to obtain performance gains through the exploration of parallelism in several levels. IPNoSys is a multicore processor based network-on-chip architecture. The programming and simulation environment of this processor allows you to execute real applications, however, some results require the description and synthesis of the hardware. In view of the above, the present work aimed to present the design, implementation and testing, in VHDL, for the RPU component of IPNoSys. The methodology used involves the adoption of a style of development oriented to tests, which prioritizes the creation of test cases before implementation. Automated tests, containing synthetic and real (sequential and/or parallel) applications, were designed to verify the correct implementation of the internal components of the RPU and the 2D matrix of RPUs. The results presented by the simulations show gains in productivity, code quality, modularity and lower error propensity.

Keywords: Computer architecture. MPSoC. NoC. IPNoSys. VHDL.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2D	Duas dimensões
ALU	Arithmetic Logic Unit
CLK	Clock
CPU	Central Processing Unit
DEMUX	Demultiplexador
DMA	Direct Memory Access
FIFO	First In First Out
FPGA	Field-programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
IDE	Integrated Development Environment
IP	Intellectual Property
IPR	Instruções por RPU
IPNoSys	Integrated Processing NoC System
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MAU	Memory Access Unit
MUX	Multiplexador
NoC	Network-on-Chip
P2P	Peer-to-peer
PCT	Packet
RST	Reset
RPU	Routing and Processing Unit
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SU	Syncronization Unit
TDD	Test Driven Development
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits

SUMÁRIO

1 INTRODUÇÃO

A arquitetura Von Neumann foi um dos principais modelos arquiteturais desenvolvidos, que serviram de base para os computadores atuais. Seus componentes – uma memória, uma unidade aritmética e lógica, uma unidade central de processamento e uma unidade de controle e um sistema de entrada e saída – são interligados por meio de barramento para executar instruções de forma sequencial.

Visando aumentar o desempenho dos processadores, técnicas de paralelismo foram propostas em diversos níveis. No nível de instrução, destaca-se a técnica de *pipeline* (PATTERSON; HENNESSY, 2005), que permite dividir os componentes de um processador em grupos que correspondem a um dos estágios de execução de uma instrução, de modo a maximizar a utilização dos componentes internos (evitando estados de ociosidade). Outra técnica, chamada de modelo superescalar (HWANG, 1992), consiste na multiplicação dos componentes internos de um processador para realizar execuções simultâneas de instruções no mesmo ciclo de *clock*.

Como alternativa aos processadores superescalares, modelos de arquiteturas paralelas foram desenvolvidos. Aliado a isso, o aumento da capacidade de integração nos chips possibilitou a construção de sistemas multiprocessados em um único chip (MPSoC ou Multiprocessor System-on-Chip). Estes processadores com múltiplos núcleos viabilizaram o surgimento de técnicas de paralelismo no nível de tarefa, como multiprocessamento ou *multitasking* (PATTERSON; HENNESSY, 2005) e *multithreading* (HWANG, 1992). *Multitasking* distribui várias tarefas, entre os núcleos disponíveis, para execução em paralelo. Enquanto que *multithreading* divide uma tarefa em vários fluxos de execução, podendo ser executado de forma paralela (quando há mais de um núcleo) ou concorrente (quando há somente um núcleo).

Os mecanismos de interconexão entre os núcleos de processamento diferem quanto à escalabilidade, latência, paralelismo na comunicação, reusabilidade, área em chip e dissipação de potência. A solução para as limitações dos mecanismos tradicionais, como o barramento e a hierarquia de barramento é conhecido como redes em chip (NoC ou Network-on-Chip) (BENINI; MICHELI, 2002), que tem como base conceitos de redes de computadores.

A característica comum dos sistemas MPSoCs baseados em NoC, como por exemplo, a plataforma STORM (REGO, 2006), diz respeito a utilizar um elemento para realizar exclusivamente a comunicação (roteador) e, outro, IP (*Intellectual Property*) como o

processamento (núcleo) e memórias. Desta forma, nenhum processamento útil é feito no mecanismo de comunicação entre núcleos.

Sendo assim, o processador IPNoSys (*Integrated Processing NoC System*) surge como proposta às limitações encontradas nos tradicionais MPSoCs baseados em NoC, ao permitir que o processamento seja realizado durante a comunicação. O componente responsável por essa operação é chamado de RPU (*Routing and Processing Unit*), que será abordado no capítulo seguinte. A organização de IPNoSys é baseada em redes em chip tirando proveito da escalabilidade e comunicação paralela para exploração do paralelismo das aplicações, principalmente em nível de tarefas.

O ambiente de programação e simulação dessa arquitetura, implementado em SystemC (ACCELLERA, 2005), permite a execução de aplicações reais com precisão de ciclos. Contudo, alguns resultados, tais como frequência de operação máxima, área em chip e potência dissipada, só podem ser obtidos com a implementação em linguagens de descrição de *hardware* (HDL).

Portanto, este trabalho tem como objetivo geral a implementação do componente RPU da arquitetura IPNoSys de forma sintetizável em FPGA. Os objetivos específicos são: projetar o componente que será implementado; descrever o componente em uma HDL; e, realizar testes individuais e de integração em uma malha 2D.

O texto deste trabalho está organizado da seguinte forma: o capítulo 2 apresenta o referencial teórico sobre arquiteturas paralelas, arquiteturas de interconexão entre núcleos de processamento, arquitetura IPNoSys e trabalhos relacionados; o capítulo 3 mostra a metodologia utilizada no desenvolvimento, desde o projeto até os testes, abordando, também, o estilo de desenvolvimento guiado a teste e a estratégia de teste; o capítulo 4 discorre sobre o projeto e a implementação de cada componente da RPU, ilustrando sempre que conveniente; o capítulo 5 apresenta os resultado obtidos nos estudos de caso e na simulação de uma aplicação real, além de discussões sobre o resultado de síntese; por fim, o capítulo 6 apresenta as considerações finais e os trabalhos futuros, seguido pelas referências utilizadas no texto e pelo apêndice, onde será disponibilizado o código-fonte em VHDL.

2 REFERENCIAL TEÓRICO

2.1 Arquiteturas de processamento paralelo

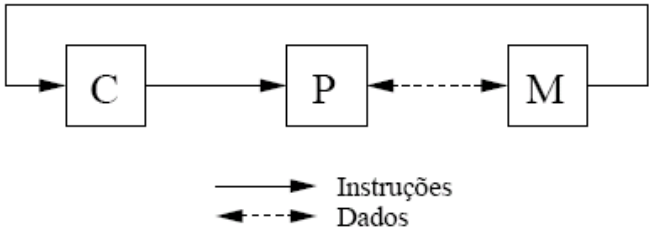
Os processadores baseados no modelo de Von Neumann possuem apenas uma única unidade central de processamento (CPU) que executa instruções em um fluxo e gerencia a comunicação entre a memória e os dispositivos de entrada e saída. Alguns fatores contribuem para aumentar o desempenho dos processadores, são eles: aumento da frequência do *clock* e o número de transistores em um chip. Entretanto percebeu-se que havia um limite para o aumento da frequência do *clock* de processador, uma vez que influencia diretamente na temperatura do semicondutor (LEÃO; CARDOSO, 2004).

Por isso, para manter o aumento de desempenho diante dessas restrições, foi necessário explorar o paralelismo. O paralelismo implica a utilização simultânea de recursos do processador. As primeiras técnicas envolviam o paralelismo no nível de instrução para uma única CPU, tais como *pipeline* e processadores superescalares. As técnicas no nível de tarefa desenvolvidas posteriormente, como *multithreading* e *multitasking*, maximizam a utilização dos recursos disponibilizados a partir do surgimento de modelos arquiteturais paralelos.

Há diversas classificações para arquiteturas paralelas que consideram, por exemplo, o nível de paralelismo ou o compartilhamento de memória. Porém, a principal classificação de arquiteturas paralelas é a classificação de (FLYNN, 1972), que possui quatro categorias baseadas na quantidade de fluxo de instruções e dados: SISD (*Single Instruction Single Data*), SIMD (*Single Instruction Multiple Data*), MISD (*Multiple Instruction Single Data*) e MIMD (*Multiple Instruction Multiple Data*). De acordo com (FLYNN, 1972), um fluxo de execução implica em uma sequência de instruções ou dados executados por um processador.

Assim, máquinas SISD (Figura 1) executam apenas um fluxo de instrução e um fluxo de dados, sendo representado por processadores baseados na arquitetura de Von Neumann, uma vez que possui uma única unidade de controle gerenciando o processador para atuar em apenas um dado contido na memória.

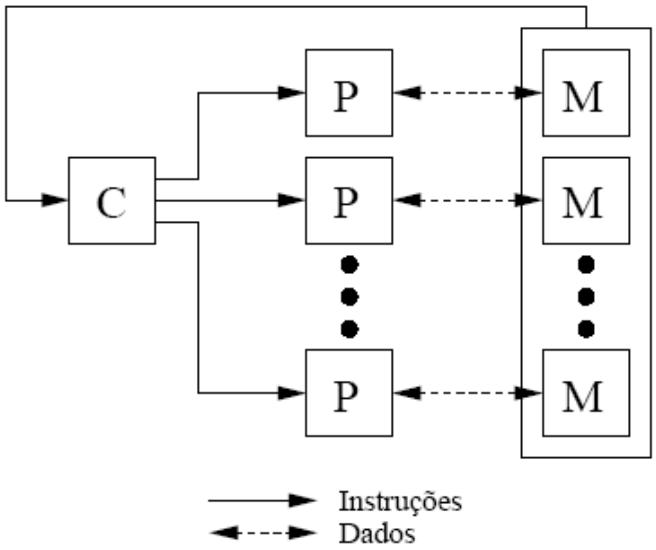
Figura 1– Máquinas SISD da classificação de Flynn.



Fonte: Oliveira (2017).

Enquanto que os SIMD (Figura 2) executam um fluxo único de instruções em múltiplos dados simultaneamente. Neste caso, a unidade de controle da arquitetura recebe o fluxo de instruções que atuam sobre os múltiplos processadores para manipular diferentes dados, ao mesmo tempo, onde estes são lidos e, posteriormente, escritos na memória. Esta categoria pode ser representada por processadores com arquitetura vetorial.

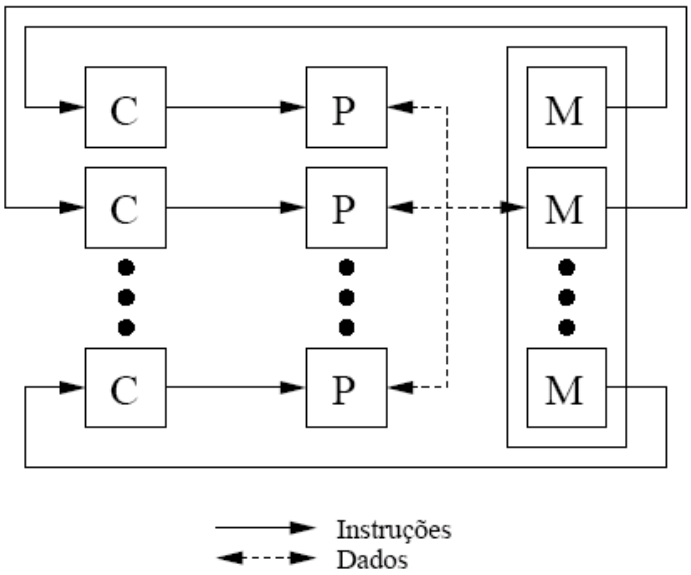
Figura 2 – Máquinas SIMD da classificação de Flynn.



Fonte: Oliveira (2017).

As arquiteturas MISD (Figura 3) executam múltiplos fluxos de instrução que atuam em um único fluxo de dados, porém, esta arquitetura é impraticável, não havendo nenhuma implementação para esta proposta.

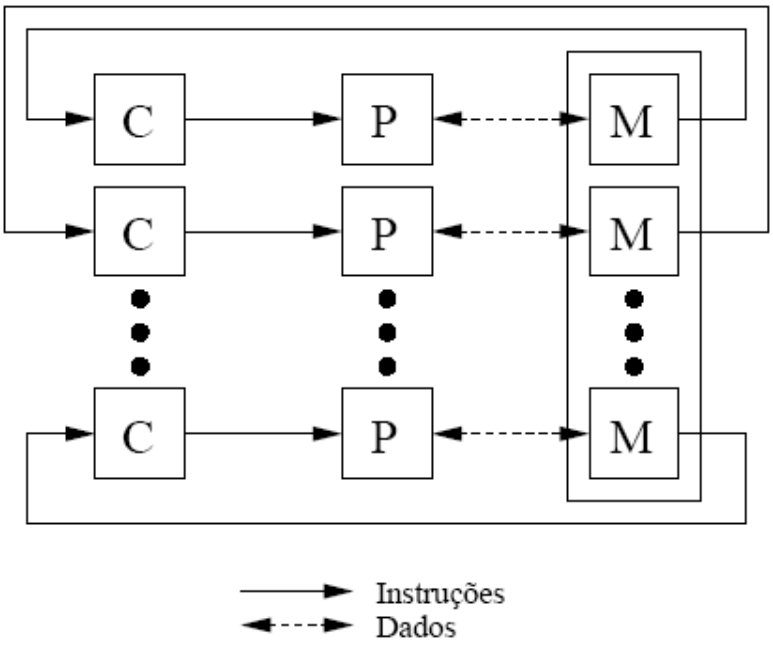
Figura 3 – Máquinas MISD da classificação de Flynn.



Fonte: Oliveira (2017).

Por último, máquinas com arquitetura MIMD executam múltiplas fluxos de instruções simultaneamente em múltiplos processadores para manipular múltiplos dados na memória. Como pode ser visto na Figura 4, existe uma unidade de controle (C) para cada unidade de processamento (P). Esta arquitetura corresponde aos multiprocessadores atuais.

Figura 4 – Máquinas MIMD da classificação de Flynn.

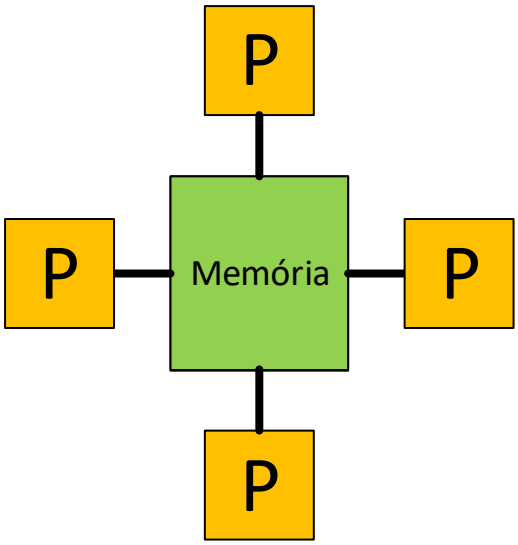


Fonte: Oliveira (2017).

A classe MIMD da classificação de Flynn pode, ainda, ser subdividida de acordo com o compartilhamento de memória, obtendo dois tipos de arquiteturas: multiprocessadores ou

multicomputadores. Multiprocessadores (Figura 5) utilizam memória compartilhada, isto é, há um único espaço de endereçamento que é compartilhado entre múltiplos processadores. Este tipo de arquitetura não permite escalabilidade, visto que está limitado os recursos computacionais da máquina, e é comum o paralelismo no nível de tarefa. Os multiprocessadores diferem de arquiteturas multinúcleo pois estas contêm vários núcleos de processamento dentro de um mesmo chip.

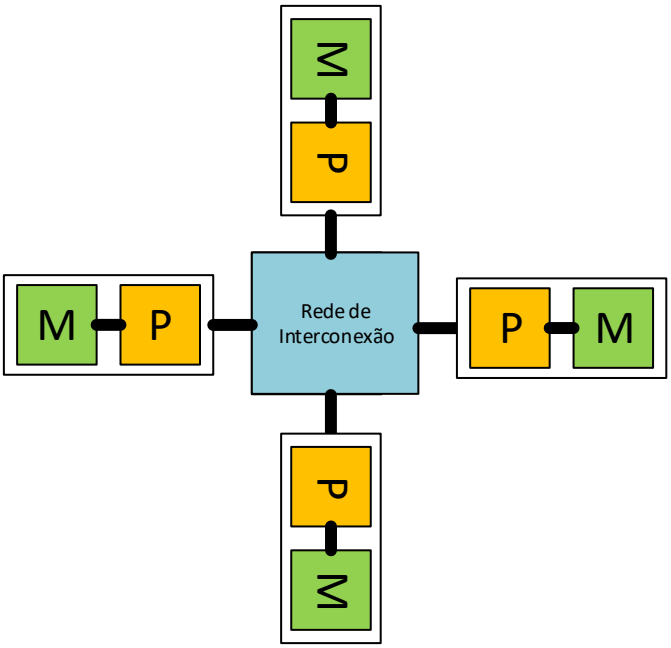
Figura 5 – Arquitetura MIMD para Multiprocessadores.



Fonte: Próprio autor.

Os multicomputadores (Figura 6) utilizam memória distribuída, onde cada computador possui um espaço de endereçamento privado para seus processadores e comunica-se com outros computadores, por meio de troca de mensagens através de uma rede de interconexão, para realizar o processamento paralelo. Esta arquitetura permite que nós de processamento sejam adicionados, permitindo, assim, que haja um aumento de processadores na medida em que se faz necessário. Contudo, o atraso na comunicação e sincronização podem limitar o desempenho.

Figura 6 – Arquitetura MIMD para Multicomputadores.



Fonte: Próprio autor.

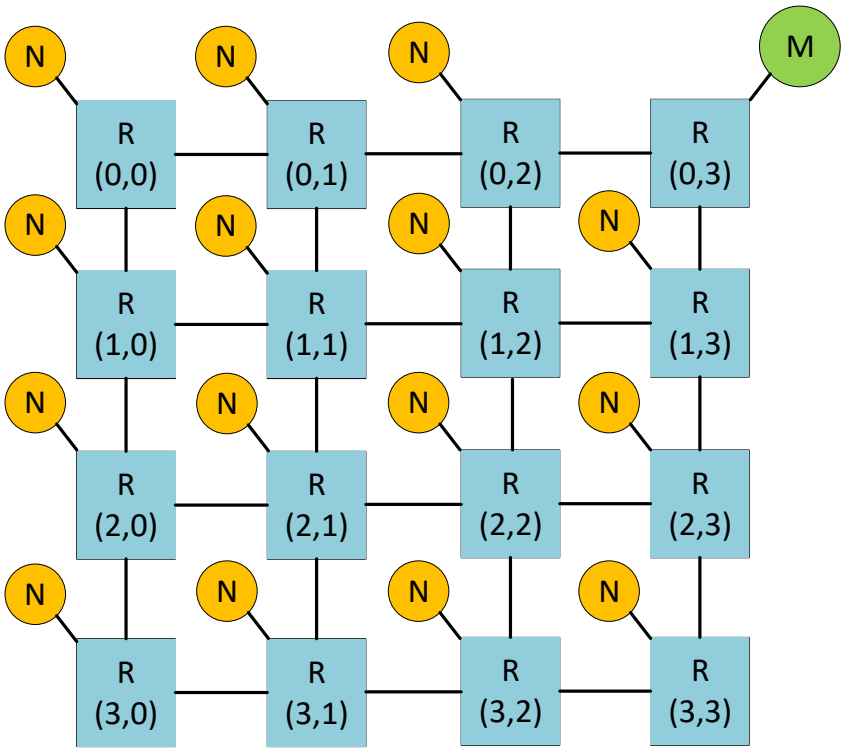
2.2 Arquiteturas de interconexão

Existem várias arquiteturas de interconexão que podem ser utilizadas em MPSoC (*Multiprocessor System-on-Chip*), como, por exemplo, canais ponto-a-ponto, canais multiponto e redes em chip. Nos canais ponto-a-ponto (ou Peer-to-Peer) cada núcleo possui um canal de comunicação exclusivo com demais os núcleos, enquanto que os canais multiponto compartilham a estrutura física de comunicação por todos os núcleos. Esta arquitetura de conexão é mais conhecida como barramentos e tem como característica a fácil adição de núcleos implicando de forma negativa no desempenho, ao aumentar o tempo de espera dos núcleos para transmissão, e no consumo de energia. Uma alternativa ao barramento é a utilização de hierarquia de barramentos, porém, em ambas, a baixa escalabilidade compromete sua adoção.

Portanto, de acordo com (ALI, WELZL et al., 2008) a melhor arquitetura de interconexão entre núcleos de MPSoC são as redes em chip (ou NoC, *Network on Chip*). As NoC são formadas por interfaces de rede, roteadores e enlaces. As interfaces de rede definem o padrão de comunicação entre os núcleos. Já os roteadores são responsáveis por rotear os pacotes de um núcleo origem para um núcleo destino. Por fim, os enlaces correspondem aos canais físicos que conectam os núcleos aos roteadores. O modelo de computação dirigido a pacotes será abordado na próxima seção.

Assim como em redes de computadores, é possível implementar várias topologias em uma NoC. Dentre elas, a topologia direta indica que cada núcleo está conectado diretamente a um roteador, enquanto que na topologia indireta nem todos os roteadores conectam-se aos núcleos. A topologia apresentada neste trabalho corresponde à topologia direta em malha 2D, conforme ilustrado, por um MPSoC tradicional baseado em NoC, na Figura 7. Na topologia deste modelo, uma matriz 2D 4x4 é definida, em que tem-se cada roteador posicionado em um índice da matriz e conectado a um núcleo (exceto o roteador (0,3) que está conectado a uma memória).

Figura 7 – Modelo tradicional de MPSoC baseado em NoC.



Fonte: Próprio autor.

Para transmitir pacotes entre dois núcleos é necessário definir um algoritmo de roteamento que indica por onde um pacote deverá ser encaminhado. O algoritmo tradicional de roteamento é o algoritmo determinístico XY, que deve encaminhar um pacote para a mesma linha do núcleo destino e, em seguida, a mesma coluna, alcançando, assim, o núcleo alvo.

Os enlaces ou canais físicos podem ser divididos em canais virtuais de modo a transmitir mais de um pacote pelo mesmo meio de transmissão, através do chaveamento das palavras desses pacotes (escolher a palavra de um dos pacotes que será transmitida). Para decidir qual dos pacotes será transmitido, as portas de saídas possuem árbitros que utilizam algoritmos de agendamento, como o Round Robin (ARPACI-DUSSEAU; ARPACI-

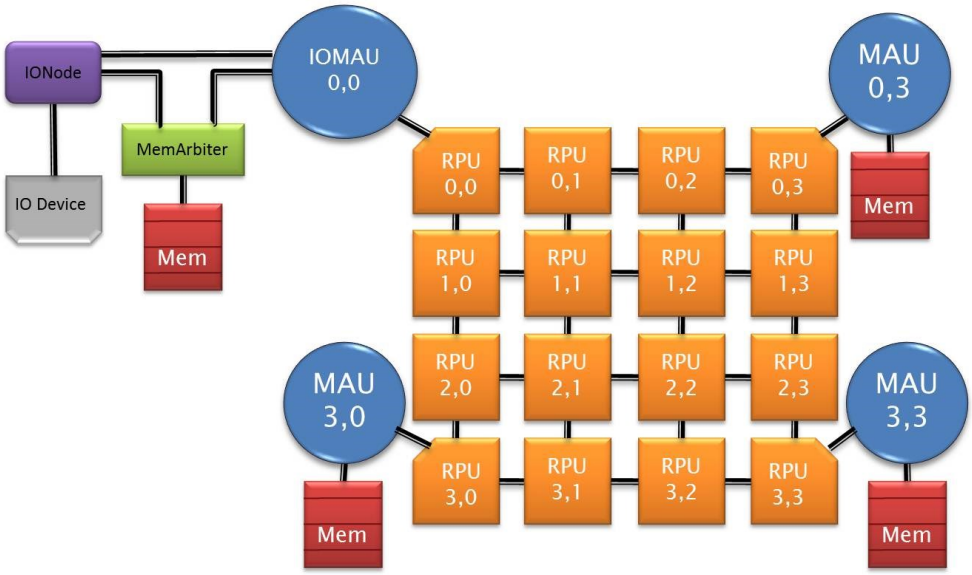
DUSSEAU, 2012). Como é possível haver disputas pelo canal de transmissão, estratégias de memorização devem ser adotadas para armazenar temporariamente parte ou um pacote completo. Em geral, por simplicidade, *buffers* do tipo FIFO (*First In First Out*) são adotados nesta tarefa.

2.3 Plataforma IPNoSys

A plataforma IPNoSys compreende um processador (organização), seu conjunto de instruções (arquitetura) e as ferramentas para geração de aplicações e simulação da execução desenvolvidas em SystemC e C++. (ARAÚJO, 2012).

A organização do processador IPNoSys (*Integrated Processing NoC System*) (Figura 8) é um processador de propósito geral com arquitetura não-convencional, que aproveita a infraestrutura de uma rede-em-chip e características que favorecem o paralelismo. Sua organização é formada por uma malha 2D de RPUs (*Routing and Processing Units*) conectada, pelos cantos, as MAUs (*Memory Access Unit*) e um sistema de entrada e saída baseado em DMA (*Direct Memory Access*). O modelo de computação dirigido a pacotes (FERNANDES et al., 2010), de sua arquitetura, une o processamento e a comunicação, uma vez que os programas são descritos por instruções e dados em um formato de pacote próprio. Tais pacotes são armazenados, gerenciados e injetados pelas MAUs (módulos de memória), nos cantos da rede. As RPUs (processadores) estão encarregadas de rotear os pacotes até o destino e processá-lo durante seu percurso pela rede.

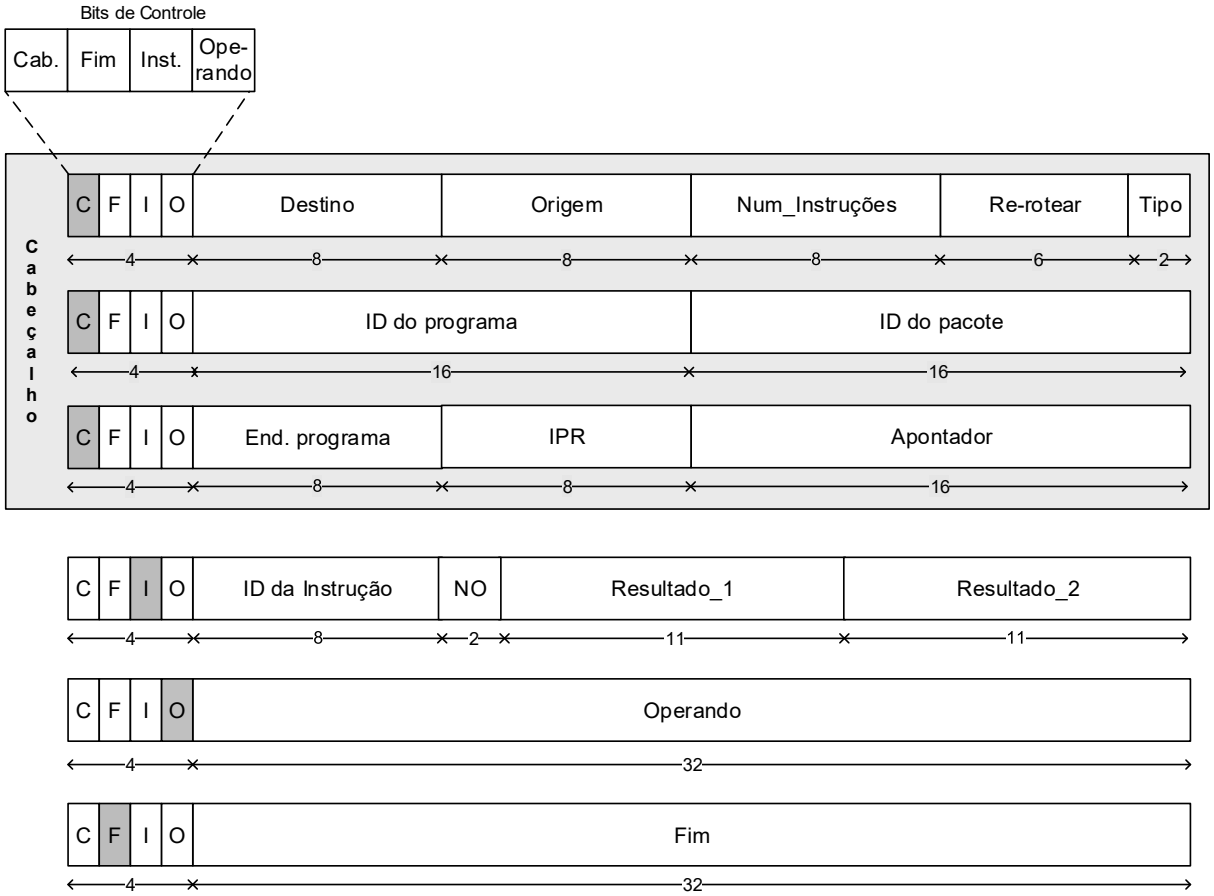
Figura 8 – Organização de IPNoSys.



Fonte: Araújo (2012).

Em IPNoSys, as aplicações são descritas por meio de um ou mais pacotes. O formato geral desses pacotes, ilustrado na Figura 9, é composto por palavras com 36 bits de largura, em que os 4 bits mais significativos correspondem aos bits de controle e os 32 bits restantes, ao conteúdo da palavra. Os bits de controle indicam o tipo da palavra, que pode ser um cabeçalho, uma instrução, um operando ou o fim do pacote. Somente um dos quatros bits de controle pode estar ativo (nível lógico 1) em cada palavra do pacote; todos os demais devem estar desativados (nível lógico 0).

Figura 9 – Formato geral dos pacotes.



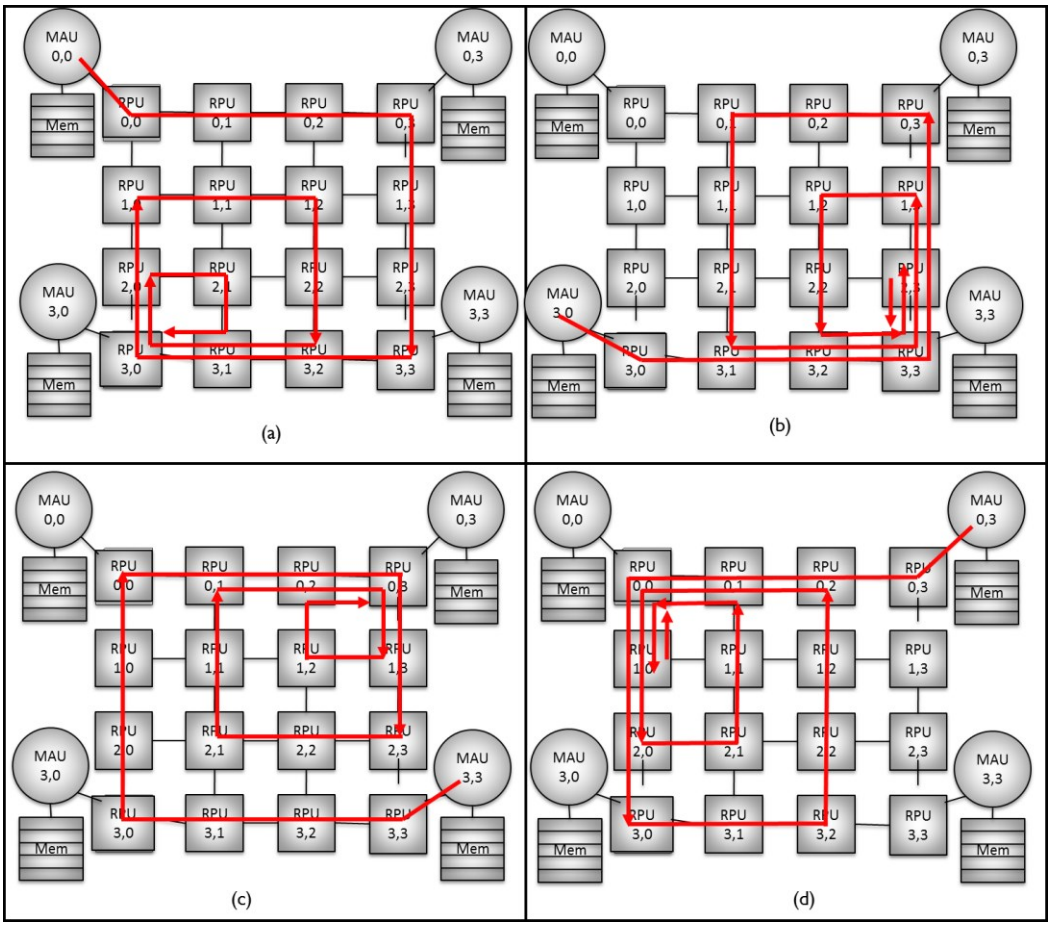
Fonte: Araújo (2012).

Há quatro tipos de pacotes (identificados no campo Tipo do pacote): regular (00_2), de controle (01_2), interrompido (10_2) e *caller* (11_2). Os pacotes regulares possuem instruções que serão executadas na RPU durante o percurso na rede. Por outro lado, os pacotes de controle contém uma instrução que a ser executada na MAU. Pacotes interrompidos e *caller* são pacotes regulares, em que o primeiro aguarda um evento de entrada e saída e o segundo, aguarda o retorno de uma função executada por outro pacote. Neste trabalho será abordado apenas os pacotes regulares, uma vez que o canal virtual para os pacotes de controle ainda não foi implementado.

Os pacotes regulares contêm 3 palavras de cabeçalho, uma ou mais instruções seguida de seus respectivos operandos e uma palavra de fim de pacote. A primeira palavra do cabeçalho possui o endereço da RPU que iniciou (origem) e onde deve finalizar (destino) o roteamento, além do número de instruções contidas e o tipo do pacote e o campo de re-rotear para calcular o novo endereço de destino. Na segunda palavra do cabeçalho há um identificador único para uma aplicação e os identificadores dos pacotes que compõe esta aplicação. Já a terceira palavra de cabeçalho possui o endereço da MAU em que o programa foi iniciado, o número de instruções executadas em cada RPU (IPR) (geralmente apenas uma instrução por RPU) e o apontador, que indica a próxima instrução a ser executada. A palavra de instrução inclui o identificador único da instrução a ser executada, o número de operandos dessa instrução e os campos de resultados que indicam a posição relativa do pacote onde o resultado da execução deve ser inserido. A palavra de operando possui apenas o dado usado pela instrução. Enquanto que a palavra de fim de pacote tem todos os bits em zero para indicar que o pacote chegou no fim.

Após ser inserido na rede, o pacote realiza seu processamento em cada RPU que percorrer. Para que seja possível executar o maior número de instruções, o algoritmo de roteamento *Spiral Complement* (ARAÚJO, 2012) foi desenvolvido para calcular o destino que fosse o mais distante possível da origem e, caso o pacote chegue ao destino sem executar todas instruções, ter seu destino calculado novamente, continuando o roteamento e execução do pacote. Para isso, o primeiro destino de um pacote é calculado como o complemento da origem. Quando o destino é atingido e ainda há instruções a serem executadas, um novo destino é calculado. Isto é feito com o incremento ou decremento, de forma alternada, da coordenada X ou Y da RPU de origem do pacote, através do campo re-rotear do cabeçalho. A Figura 10 ilustra quatro possibilidades de espiral formada pelo algoritmo, uma vez que um pacote pode ser inserido por uma das quatro MAUs, que estão nos cantos da rede.

Figura 10 – Espirais formada pelo algoritmo de roteamento *Spiral Complement*.



Fonte: Araújo (2012).

Situações de *deadlock* podem ocorrer quando dois pacotes (ou até mesmo um pacote com muitas instruções, como exibido pelas espirais) tentam transmitir pelo mesmo canal. Eventuais *deadlocks* são tratados por meio da execução localizada, ou seja, o pacote para de transmitir e retoma a execução de instruções até que a transmissão esteja novamente disponível. As informações da quantidade de instruções e de RPUs disponíveis na rede são decisivas para evitar casos de *deadlock*.

No que diz respeito à execução, a arquitetura fornece ao programador um total de 32 instruções, distribuídas entre instruções: aritméticas, lógicas, de deslocamento, de acesso à memória, de sincronização entre pacotes, de desvio condicional e incondicional, auxiliares, de entrada e saída, de sistema e de chamada de procedimentos, mostrados na Tabela 1.

Tabela 1 – Conjunto de instruções da arquitetura IPNoSys.

Código	Instrução	Tipo	Nº de Operandos	Descrição
0	ADD	Aritmética	2	Soma 2 inteiros
1	SUB	Aritmética	2	Subtrai 2 inteiros
2	MUL	Aritmética	2	Multiplica 2 inteiros
3	DIV	Aritmética	2	Divide 2 inteiros
4	NOT	Lógica	1	Negação de 1 valor
5	AND	Lógica	2	Conjunção de 2 valores
6	OR	Lógica	2	Disjunção de 2 valores
7	XOR	Lógica	2	Ou-exclusivo de 2 valores
8	RSHIFT	Deslocamento	2	Desloca n bits de um valor à direita
9	LSHIFT	Deslocamento	2	Desloca n bits de um valor à esquerda
10	LOAD	Acesso à memória	1	Solicita um valor da memória
11	STORE	Acesso à memória	Vários	Armazena um valor na memória
12	EXEC	Sincronização	1	Ordena a injeção imediata de um pacote
13	SYNEXEC	Sincronização	Vários	Ordena a injeção de um pacote após sincronização
14	SYNC	Sincronização	1	Sinal de sincronização para um pacote
15	RELOAD	Acesso à Memória	1	Retorna um valor carregado da memória
16	BE	Condicional	2	Desvia se igual
17	BNE	Condicional	2	Desvia se diferente
18	BL	Condicional	2	Desvia se menor
19	BG	Condicional	2	Desvia se maior
20	BLE	Condicional	2	Desvia se menor ou igual
21	BGE	Condicional	2	Desvia se maior ou igual

22	JUMP	Incondicional	0	Desvia incondicionalmente
23	COPY	Auxiliar	1	Copia 1 valor para outra instrução no mesmo pacote
24	NOP	Auxiliar	0	Sem operação
25	SEND	Sincronização	2	Envia um valor para um ser inserido em um pac.
26	IN	Entrada/Saida	3	Recebe bytes do controlador de E/S
27	OUT	Entrada/Saida	3	Envia bytes ao controlador de E/S
28	WAKEUP	Sistema	1	Ordena a reinjetar um pacote antes interrompido
29	NOTIFY	Sistema	1	Notifica estado de um pacote
30	CALL	Procedimento	Vários	Faz a chamada de uma função/pacote
31	RETURN	Procedimento	2	Retorna o resultado de uma função para o chamador

Fonte: Araújo (2012).

Conforme veremos nos capítulos que abordam o projeto, a implementação e os resultados, as instruções executadas pela RPU correspondem à: ADD, SUB, MUL, DIV, NOT, AND, OR, XOR, RSHIFT, LSHIFT, BE, BNE, BL, BLE, BGE, JUMP, COPY, NOP, CALL, RELOAD. Enquanto que as instruções executadas na MAU são: LOAD, STORE, EXEC, SYNEXEC, SYNC, SEND. As instruções encaminhadas à MAU pela RPU são chamadas de instrução de controle. A RPU monta um pacote de controle específico para esta instrução e o roteia até chegar na MAU. Apenas algumas instruções serão abordadas neste trabalho, uma vez que o escopo se mantém na RPU e na matrix 2D de RPUs.

2.4 Trabalhos relacionados

Já foram propostas variações na organização original de IPNoSys em SystemC como em Nunes e Fernandes (2016). Adicionalmente, em Pereira e Fernandes (2013) é apresentado uma rede de IPNoSys conectadas por roteadores e em Damasceno et al. (2016) é analisado o impacto da adição de hierarquia de memória das MAUs. Soluções de programabilidade também foram propostos em Gadelha et al. (2011) e Oliveira e Fernandes (2015). Este último descreve a implementação de um ambiente de desenvolvimento (IDE) e simulação integrado. Sobre compiladores, em Couto (2016) foi discorrido sobre a otimização do gerador de código

para a plataforma IPNoSys objetivando a exploração de paralelismo. Além disso, em relação à implementações em VHDL, Soares et al. (2015) propuseram uma nova organização baseado em IPNoSys, em Oliveira Neto e Fernandes (2015) foi proposto o projeto da implementação em VHDL de uma RPU simplificada da IPNoSys original que realiza processamento sequencial (fluxo único de execução) de pacotes regulares e, por fim, em Neto e Fernandes (2016) foi desenvolvido uma versão paralela da RPU que possui múltiplos fluxos de execução, resolvendo disputas entre recursos e prevenindo *deadlock*.

3 METODOLOGIA DE PROJETO USANDO VHDL

Quando se elabora um sistema, é importante seguir uma série de passos previsíveis, ou seja, um roteiro que ajude a criar um resultado de alta qualidade e dentro do prazo estabelecido. Este roteiro é de fundamental importância porque propicia estabilidade, controle e organização para uma atividade que pode se tornar bastante caótica sem controle. Entretanto, uma abordagem moderna (ágil) deve demandar apenas as atividades, controles e artefatos que sejam apropriados para a equipe do projeto e para o produto a ser produzido. (PRESSMAN; MAXIM, 2016).

Muitos problemas, durante a implementação de um projeto, ocorrem pela falta de uma sistemática ou pelo seguimento incorreto de uma metodologia. Portanto, para realizar a implementação da RPU com qualidade e de forma eficiente, foi necessário definir uma metodologia que auxiliasse o desenvolvimento da arquitetura escolhida, desde a etapa de especificação dos requisitos até a execução dos testes. A metodologia adotada neste trabalho é uma variação da exposta em Costa (2012), onde foi acrescido a utilização de uma técnica de desenvolvimento orientado a testes, chamada de TDD. Os detalhes desta técnica, da estratégia de teste e da metodologia serão abordadas nas seções a seguir.

3.1 Test Driven Development

O desenvolvimento guiado por testes (BECK, 2003) é uma técnica de desenvolvimento de software, criada por Kent Beck, que tem como principal objetivo a produção de “código-fonte limpo” que funciona.

Há três etapas a serem realizadas durante a programação. Primeiramente, na etapa *Red*, o desenvolvedor deve escrever os casos de teste automatizados para uma determinada funcionalidade antes de qualquer implementação. Como ainda não há implementação, este teste irá falhar. Então, na etapa *Green*, o código-fonte é escrito rapidamente, para a funcionalidade, de forma a passar nos testes. Uma vez que a implementação foi validada pelos testes, então é possível realizar a refatoração do código (etapa *Refactor*), ou seja, o código deve ser otimizado, eliminando duplicidades que possam ter sido inseridas na etapa anterior e considerando sempre a aplicação de boas práticas de programação.

Como benefícios da aplicação desta técnica, Borges (2006) afirma que ela permite reduzir a complexidade do software e facilitar a manutenção, bem como, permite que falhas sejam identificadas mais rapidamente, enquanto novo código é adicionado ao sistema.

3.2 Estratégia de Teste

De acordo com Pressman e Maxim (2016), a estratégia frequentemente utilizada por equipes de desenvolvimento é adotar uma visão incremental do teste, ou seja, deve-se começar por testes unitários individuais, projetar testes de integração das unidades e finalizar com testes que verificam o sistema construído.

Os testes de unidade verificam o componente que representa a menor unidade definida para o projeto. A complexidade e os erros descobertos neste tipo de teste estão limitados ao escopo estabelecido. Uma vez que os módulos tenham sido testados, não significa que poderão ser facilmente integrados, visto que a integração das interfaces entre estes componentes pode ter um efeito imprevisível. Para isso, os testes de integração são elaborados com o objetivo de descobrir erros relacionados às interfaces entre os módulos e permitir a construção da arquitetura desejada.

3.3 Metodologia adotada

A metodologia de projeto desta arquitetura foi dividida nas seguintes atividades:

a) relacionar características do processador alvo: definição das características da arquitetura e dos componentes internos do processador;
b) descrever os microprogramas usando fluxogramas: descrição de como a unidade de controle interpreta e executa as tarefas dentro do processador, identificando, assim, as estruturas do processador são usadas para o processamento do conjunto de instruções e em que sequência as tarefas são efetuadas;
c) desenho do caminho de dados: definição do circuito lógico combinacional necessário para a execução de todas as operações;
d) descrição da máquina de estados: descrição do comportamento da unidade de controle, a fim de determinar quais ações serão executadas e em que instante;
e) criação de casos de teste automatizados: escrita de *scripts* de teste de unidade ou de integração (dependendo do componente a ser implementado) contendo todos os casos de testes;
f) descrição do hardware em HDL: desenvolvimento de pequenos módulos e, posteriormente, a integração dos mesmos para formar componentes maiores;

g) teste do componente através de simulação: execução dos *scripts* para verificar se o componente foi implementado corretamente;
h) refatoração: otimização do código testado com sucesso.

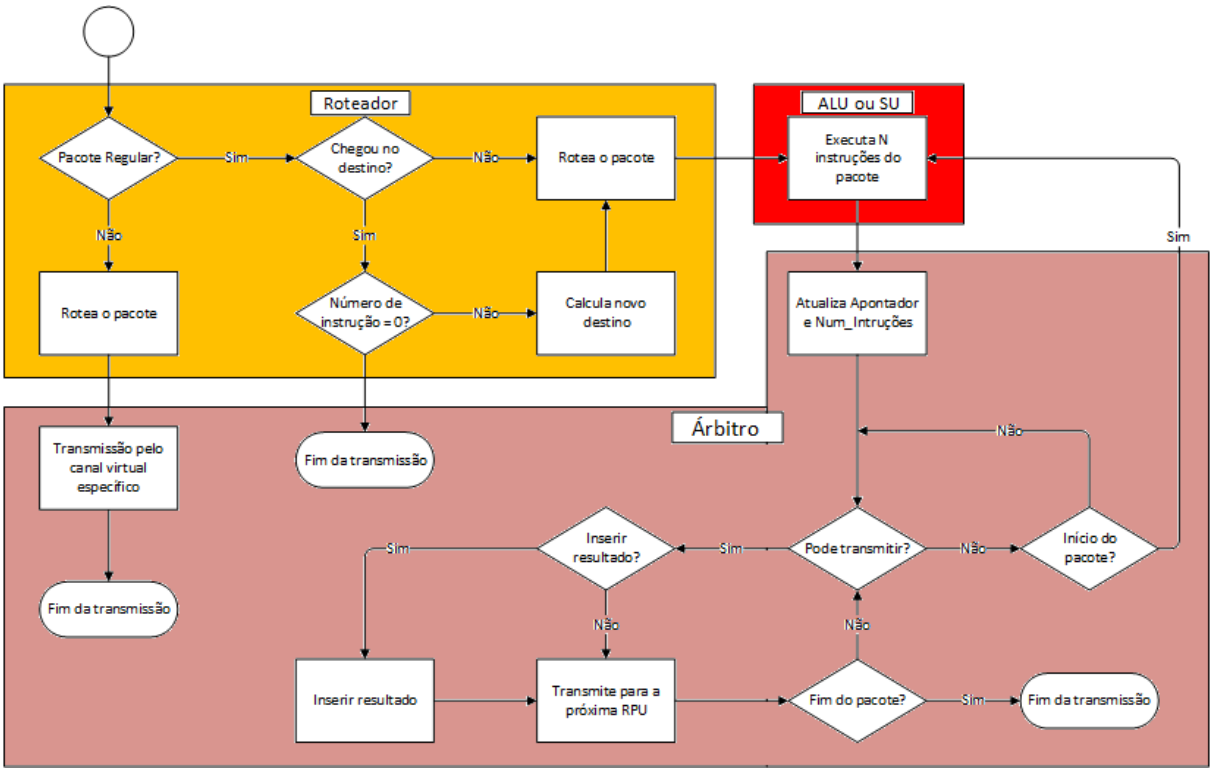
As características da arquitetura da RPU de IPNoSys foram descritas em Araújo(2012). O projeto (fluxograma, caminho de dados e máquina de estados) e a implementação (código em VHDL) serão abordados no capítulo seguinte, para cada componente interno da RPU. Enquanto que os testes (*scripts* e simulações) serão apresentados no capítulo dos resultados obtidos.

4 PROJETO E IMPLEMENTAÇÃO

4.1 RPU

A RPU (*Routing and Processing Unit*), principal entidade deste projeto, é responsável por armazenar os pacotes que chegam em suas entradas, rotear para uma saída através da direção escolhida pelo algoritmo de roteamento, realizar o processamento de, no mínimo, uma instrução (ou solicitar a criação de um pacote de controle) e transmitir o restante do pacote para a RPU seguinte, como exposto pelo fluxograma da Figura 11.

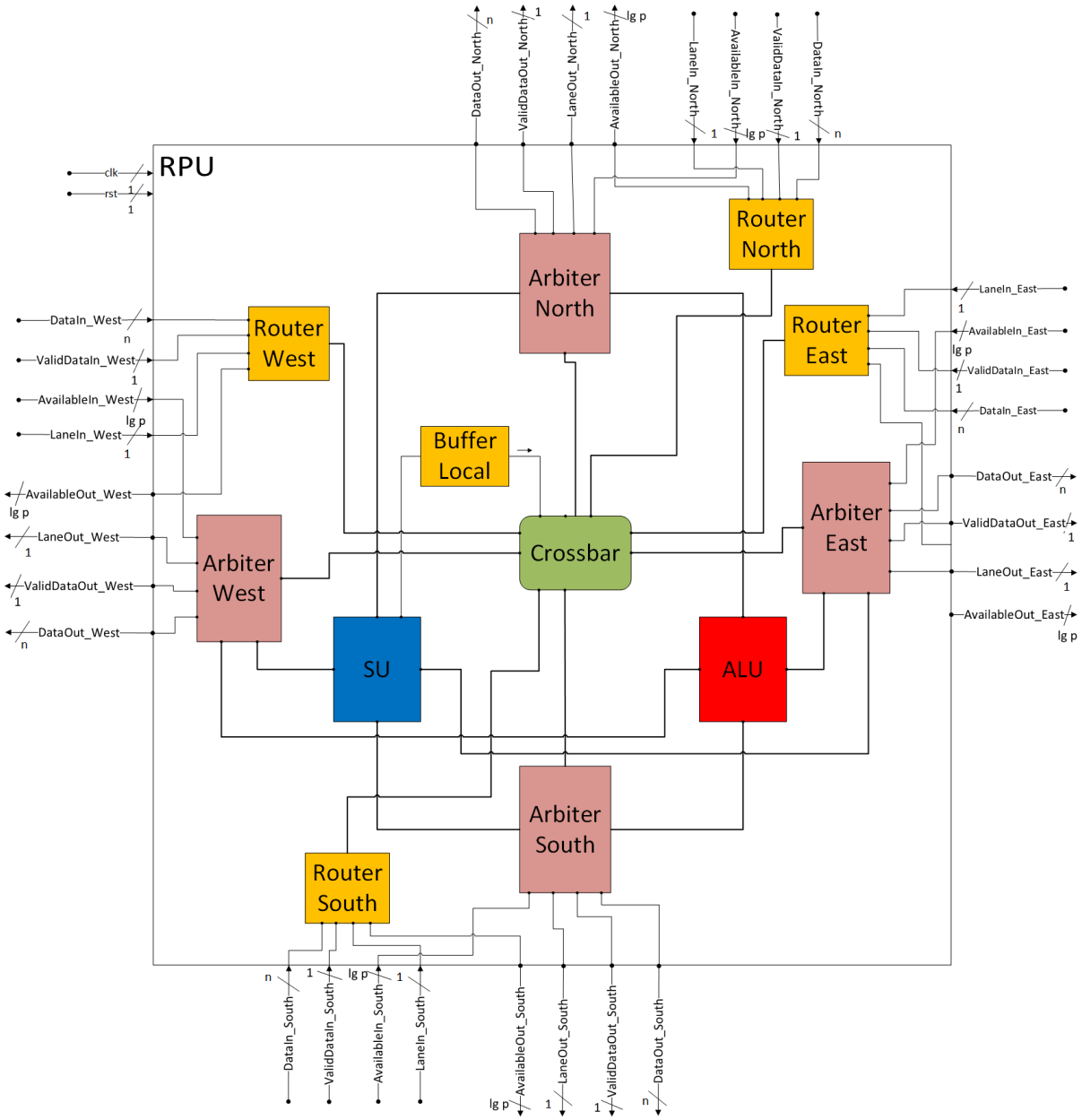
Figura 11 – Fluxograma do funcionamento da RPU.



Fonte: Araújo (2012).

Cada uma dessas funções está dividida em módulos que compõem sua arquitetura, são eles: roteadores nas entradas, um *crossbar*, uma ALU (*Arithmetic Logic Unit*), uma SU (*Synchronization Unit*) e árbitros nas saídas, conforme ilustrado na Figura 12 e descrito nas seções seguintes.

Figura 12 – Arquitetura simplificada da RPU.

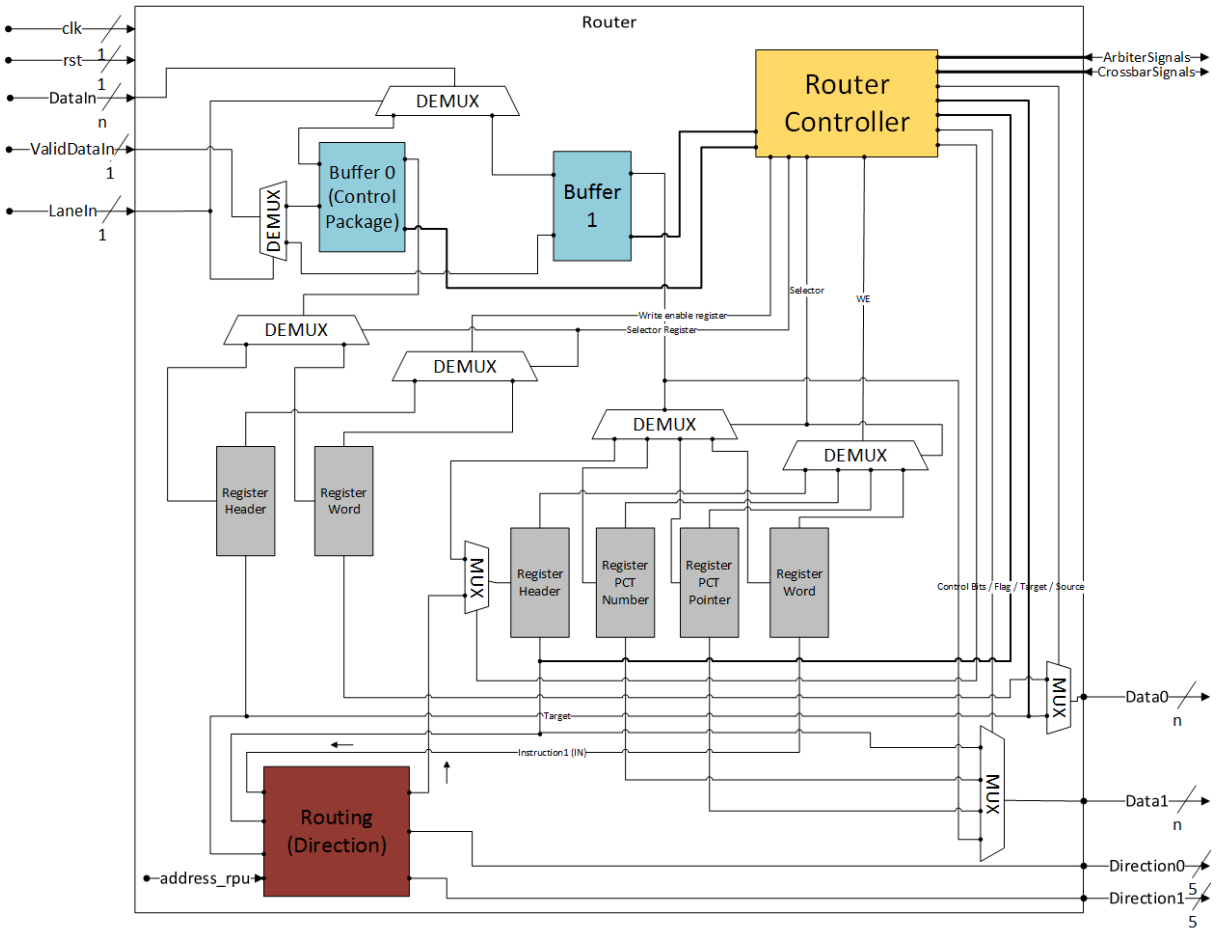


Fonte: Próprio autor.

4.2 Router

A Figura 13 ilustra o esquema geral dos componentes internos de um *Router* ou Roteador. Para facilitar o entendimento, a apresentação dos módulos foi dividida em duas subseções 4.2.1 e 4.2.2, mostradas a seguir.

Figura 13 – Visão geral do Roteador.

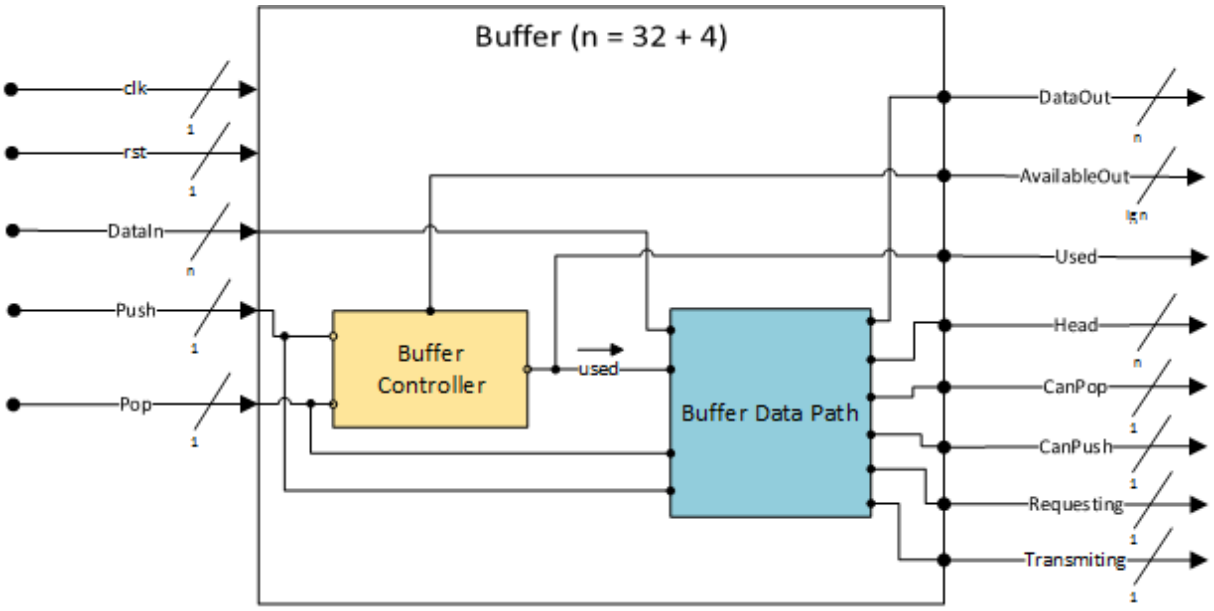


Fonte: Próprio autor.

4.2.1 Buffer

Inicialmente, os *buffers* (Figura 14), contidos nos roteadores, recebem e armazenam o pacote. Há um buffer para cada canal virtual. Um canal armazena os pacotes de controle e o outro armazena os demais tipos. Os sinais de entrada *DataIn*, *ValidDataIn*, *LaneIn*, significam, respectivamente, o dado recebido, a informação de que aquele dado é válido e em qual dos dois buffers o dado deve ser inserido, atuando como um seletor dos *DEMUXs*. O sinal *ValidDataIn* funciona como um sinal de *push* para o *buffer*, pois uma vez que o dado é válido, este poderá ser inserido. Enquanto que o sinal de pop é ativado pelo *Router Controller*, quando necessário.

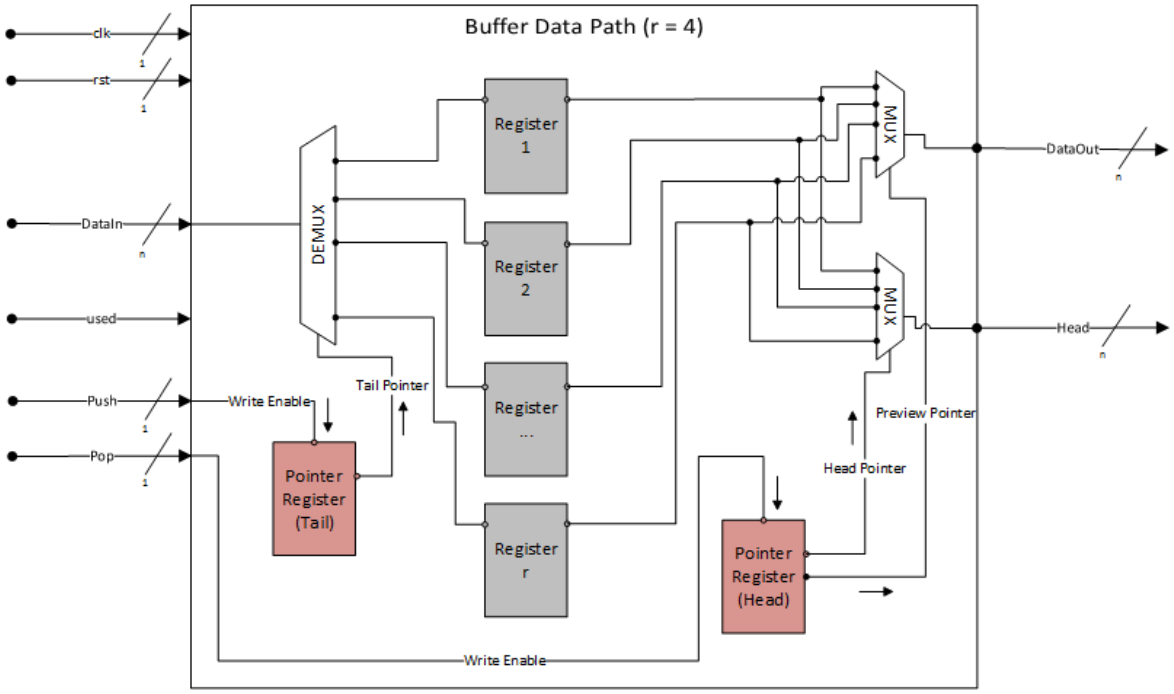
Figura 14 – Visão geral do *Buffer*.



Fonte: Neto e Fernandes (2013).

O *Buffer* é composto pelo *Buffer Controller* e pelo *Buffer Data Path*. O *Buffer Controller* gerencia a quantidade de palavras armazenadas, enquanto que o *Buffer Data Path* (Figura 15) armazena o dado recebido no *array* de registradores, conforme disponibilidade apontada pelos ponteiros *Head* e *Tail*.

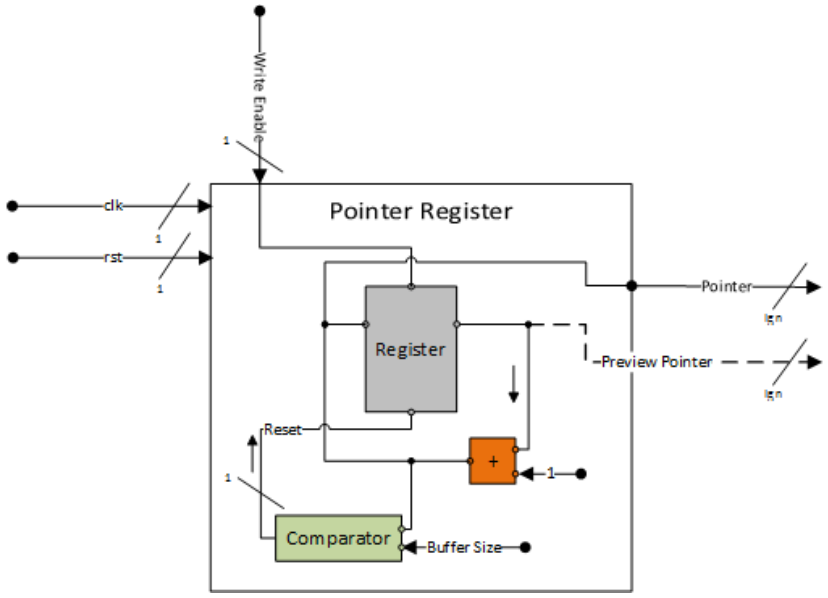
Figura 15 – Visão geral do *Buffer Data Path*.



Fonte: Neto e Fernandes (2013).

Por sua vez, os dois ponteiros, denominados de *Pointer Register* (Figura 16), são responsáveis por atualizar o endereço de apontamento da cabeça e calda, funcionando como seletor nas operações de *push* e *pop*, em relação aos registradores do Buffer. Por se tratar de um *buffer* em fila circular, a informação sobre a quantidade de palavras armazenadas é utilizada para saber se o limite inferior ou superior foi atingido e, com isso, mover o ponteiro para o início ou fim da fila.

Figura 16 – Detalhamento do *Pointer Register*.

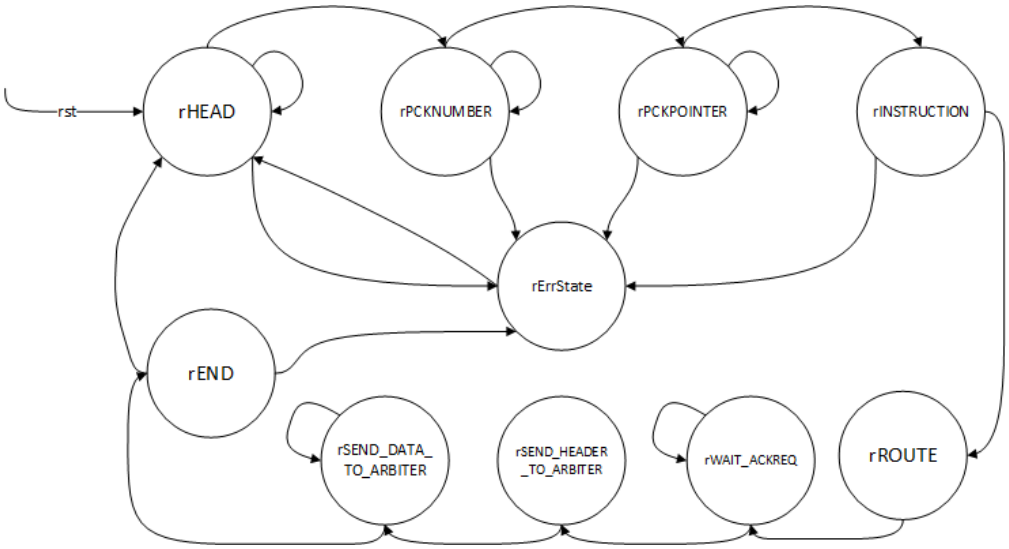


Fonte: Neto e Fernandes (2013).

4.2.2 Router Controller e Routing

O *Router Controller* implementa a FSM (*Finite State Machine*) da Figura 17. Cada estado da FSM realiza uma das operações descritas abaixo.

Figura 17 – Máquina de Estados para o *Router Controller*.



Fonte: Próprio autor.

Enquanto não houver uma palavra válida no buffer, o estado mantém-se o mesmo. Quando a primeira palavra do pacote (referente ao cabeçalho) for recebida, o *Router Controller* identifica o tipo do pacote e, posteriormente, extrai as palavras de cabeçalho. Se for um pacote regular, as palavras de cabeçalho serão armazenadas nos registradores *Header*, *PCT Number* e *PCT Pointer*, respectivamente. Caso contrário, se for um pacote de controle, a palavra de cabeçalho, removida do buffer, será armazenada no registrador *Header*.

Após isso, o componente *Routing* realiza o cálculo da direção (Norte, Sul, Oeste, Leste) na qual o pacote será enviado (através do algoritmo *Spiral Complement* para pacotes regulares e do algoritmo XY para pacotes de controle), com base nos campos de origem e destino, contidos no cabeçalho, no endereço da RPU atual e no tipo da primeira instrução que será calculada. O resultado obtido do cálculo fica à disposição da porta de saída *Direction*.

O estado seguinte da FSM diz respeito à requisição ao *Crossbar* para verificar se o Árbitro, da porta calculada anteriormente, está disponível. A cada ciclo, o *Router Controller* fica requisitando o *Crossbar*, enquanto não recebe a confirmação de que aquele Árbitro está livre. Ao receber uma resposta positiva, por meio do *AckRequest*, o roteador envia pelo *Crossbar* o cabeçalho (conforme apresentado no trecho de código da Figura 18) e, em seguida, as palavras que ainda estão no buffer.

Figura 18 – Trecho de código para requisição do *Crossbar*.

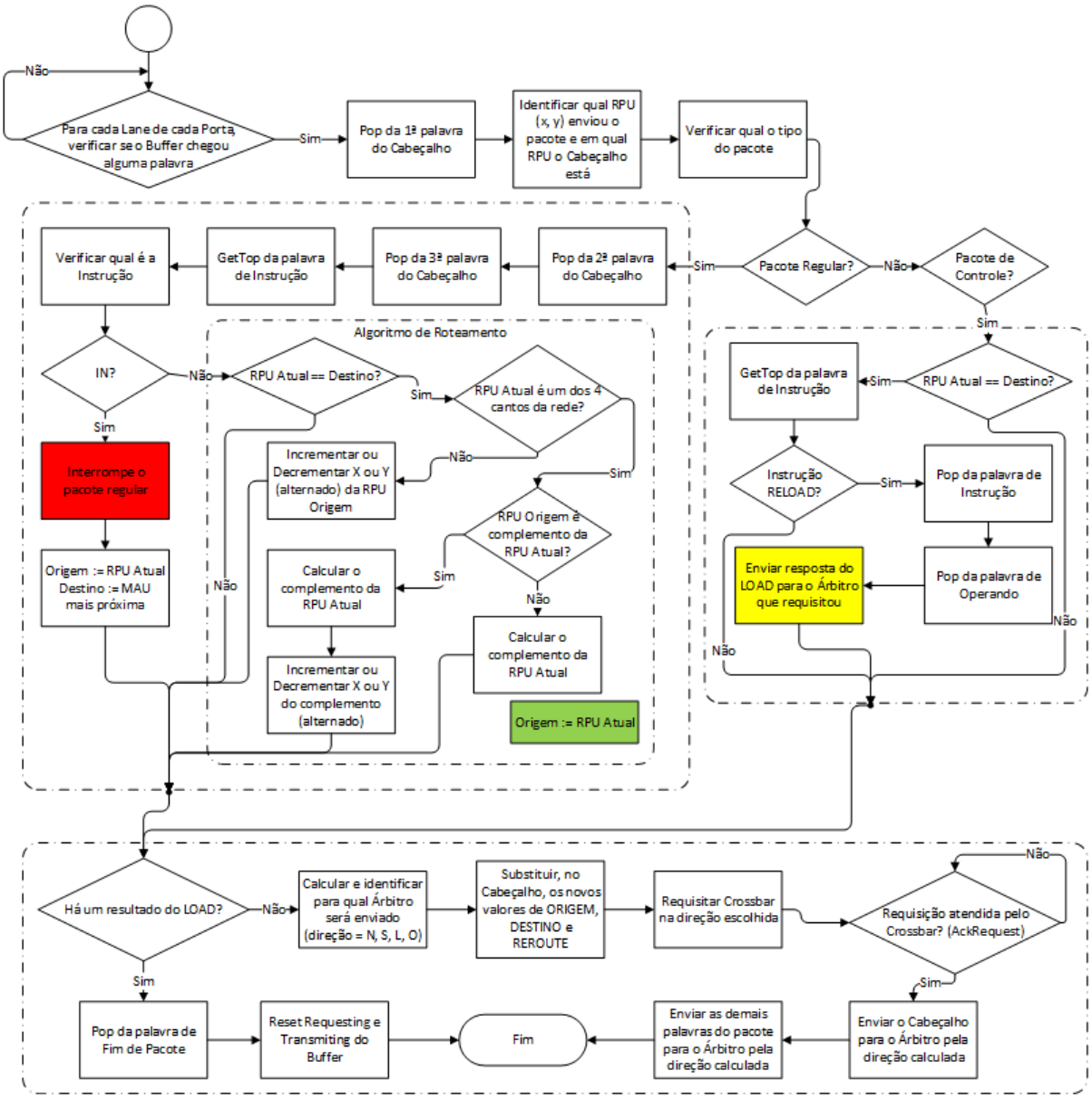
```
WHEN rWAIT_ACKREQ =>
   IF (cb_ackreq = '1') THEN
      next_state_route <= rSEND_HEADER_TO_ARBITER;
   ELSE
      var_req := '1';

      next_state_route <= rWAIT_ACKREQ;
   END IF;
```

Fonte: Próprio autor.

Quando o controlador identifica uma palavra de fim de pacote, os registradores são resetados e a FSM volta ao estado inicial, aguardando por uma inserção no *buffer*.

O fluxograma da Figura 19 condensa, com maiores detalhes, o funcionamento do roteador exposto anteriormente.

Figura 19 – Fluxograma do funcionamento do Roteador.

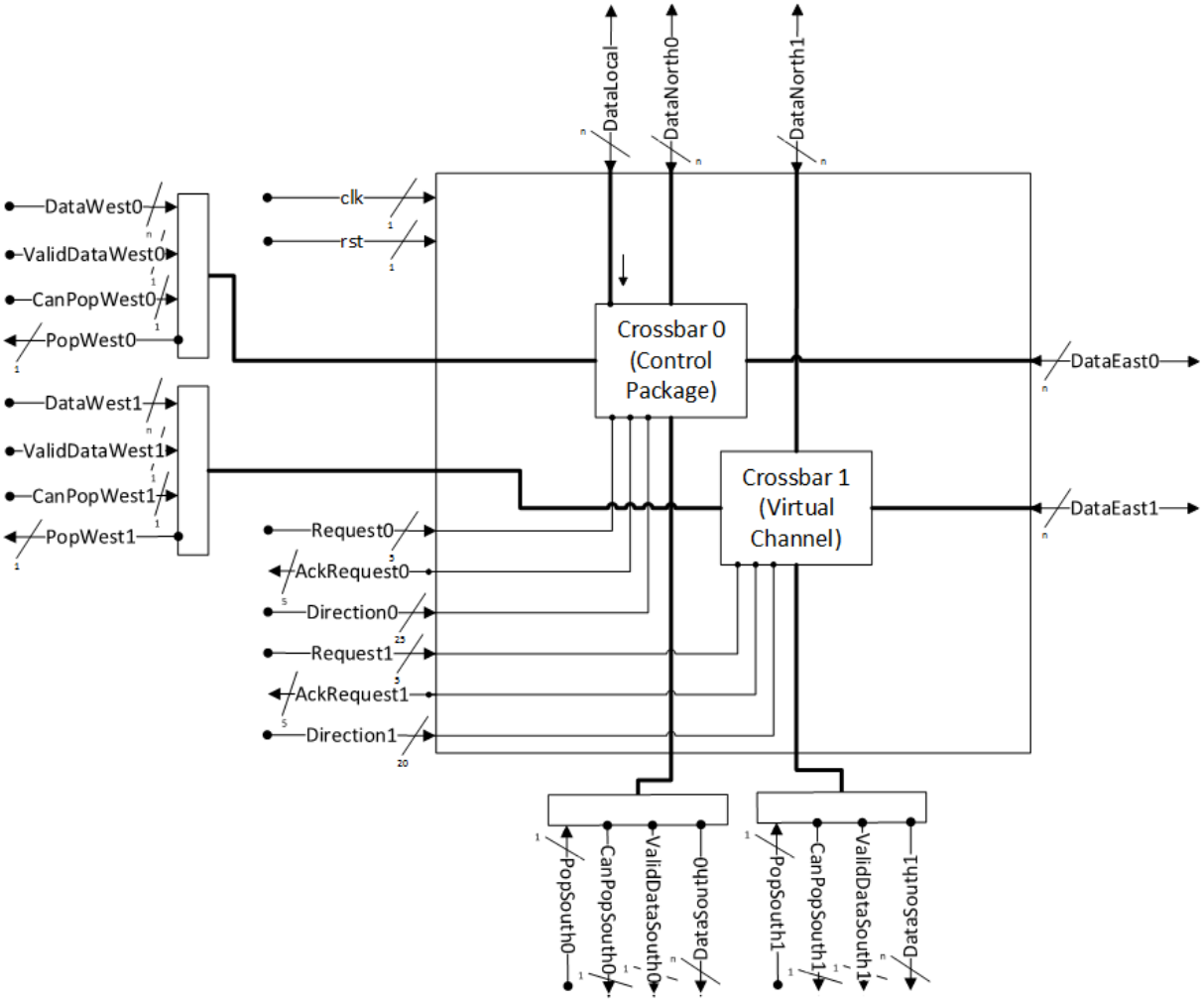


Fonte: Próprio autor.

4.3 Crossbar

A Figura 20 ilustra uma visão geral do *Crossbar*, que agrega dois componentes de mesmo nome. O número que aparece no nome destes componentes indica o tipo de pacote que será transmitido, isto é, *Crossbar0* transmite pacotes de controle e *Crossbar1* transmite demais pacotes. De agora em diante, o termo *Crossbar* será usado para referir-se ao componente interno *Crossbar0*, uma vez que este possui o diferencial, em relação ao *Crossbar1*, de receber dados do *Buffer Local*. O *Buffer Local* é o componente da RPU que recebe o pacote de controle construído pela SU e roteia, até chegar na MAU desejada, para executar uma instrução na memória.

Figura 20 – Visão geral do Crossbar.



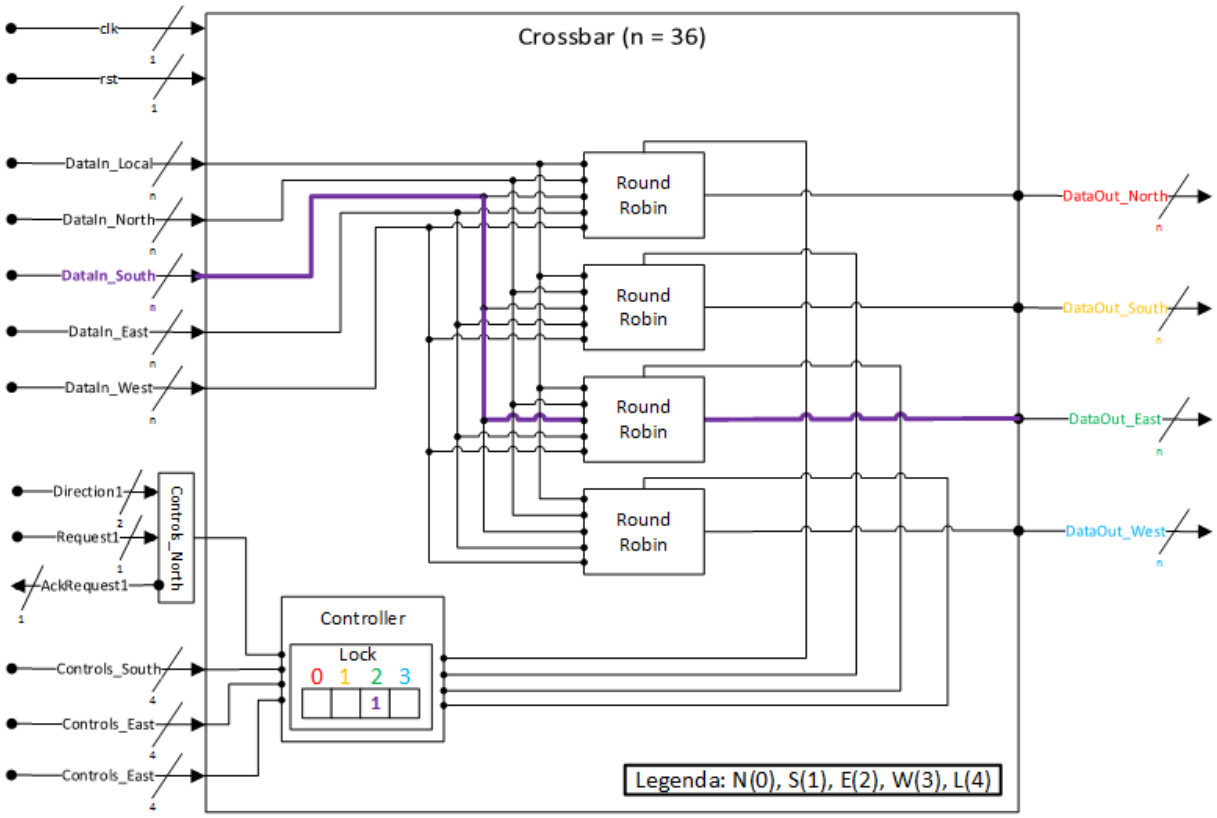
Fonte: Próprio autor.

O *Crossbar* (Figura 21) possui quatro componentes (um para cada Árbitro) que implementa uma versão modificada do algoritmo *Round Robin* (KRILL, 2016) para receber as requisições dos roteadores e resolver disputas, indicando o roteador escolhido. Esta versão

modificada foi utilizada em outros componentes exibidos neste trabalho, tendo como diferencial a adição do sinal de *done* para manter uma concessão feita (*grant*) enquanto alguma operação não é concluída.

Além do *Round Robin*, é usado um protocolo de lock, de modo que quando uma saída é alocada, para transmissão de um buffer, esta não pode ser usada por outro roteador até o fim da transmissão do pacote do buffer reservado.

Figura 21 – Visão interna do componente Crossbar0 (Control Package).



Fonte: Próprio autor.

O Crossbar também possui um *array* de duas dimensões, utilizado para chavear os dados do buffer para o Árbitro, com base na direção reservada, confirme mostra o trecho de código da Figura 22. Desta forma, o Árbitro opera o *buffer* (que está no roteador) de forma transparente, ou seja, sinais como, por exemplo, *Pop* e *CanPop* são enviados ao *Crossbar*, que fica encarregado de rotear para a saída correta, conforme trava já estabelecida.

Figura 22 – Trecho de código do Crossbar: chaveamento de dados.

```
FOR i IN 0 TO ROUTER_PORTS-1 LOOP
   IF (reserved(i) = P_NORTH) THEN
      sig_dataOutPorts(i) <= dataIn_north;
      sig_headerOutPorts(i) <= headerIn_north;
      sig_pcknumberOutPorts(i) <= pcknumberIn_north;
      sig_pckpointerOutPorts(i) <= pckpointerIn_north;
      canPop_Out(i) <= canPop_In(P_NORTH);
      buf_pop_Out(P_NORTH) <= buf_pop_In(i);
      validDataOut(i) <= validDataIn(P_NORTH);
   ELSIF(reserved(i) = P_SOUTH) THEN
      sig_dataOutPorts(i) <= dataIn_south;
      sig_headerOutPorts(i) <= headerIn_south;
      sig_pcknumberOutPorts(i) <= pcknumberIn_south;
      sig_pckpointerOutPorts(i) <= pckpointerIn_south;
      canPop_Out(i) <= canPop_In(P_SOUTH);
      buf_pop_Out(P_SOUTH) <= buf_pop_In(i);
      validDataOut(i) <= validDataIn(P_SOUTH);
   ELSIF(reserved(i) = P_EAST) THEN
      sig_dataOutPorts(i) <= dataIn_east;
      sig_headerOutPorts(i) <= headerIn_east;
      sig_pcknumberOutPorts(i) <= pcknumberIn_east;
      sig_pckpointerOutPorts(i) <= pckpointerIn_east;
      canPop_Out(i) <= canPop_In(P_EAST);
      buf_pop_Out(P_EAST) <= buf_pop_In(i);
      validDataOut(i) <= validDataIn(P_EAST);
   ELSIF(reserved(i) = P_WEST) THEN
      sig_dataOutPorts(i) <= dataIn_west;
      sig_headerOutPorts(i) <= headerIn_west;
      sig_pcknumberOutPorts(i) <= pcknumberIn_west;
      sig_pckpointerOutPorts(i) <= pckpointerIn_west;
      canPop_Out(i) <= canPop_In(P_WEST);
      buf_pop_Out(P_WEST) <= buf_pop_In(i);
      validDataOut(i) <= validDataIn(P_WEST);
   ELSE
      sig_dataOutPorts(i) <= (OTHERS => '0');
      sig_headerOutPorts(i) <= (OTHERS => '0');
      sig_pcknumberOutPorts(i) <= (OTHERS => '0');
      sig_pckpointerOutPorts(i) <= (OTHERS => '0');
      canPop_Out(i) <= '0';
      buf_pop_Out(i) <= '0';
      validDataOut(i) <= '0';
   END IF;
END LOOP;
```

Fonte: Próprio autor.

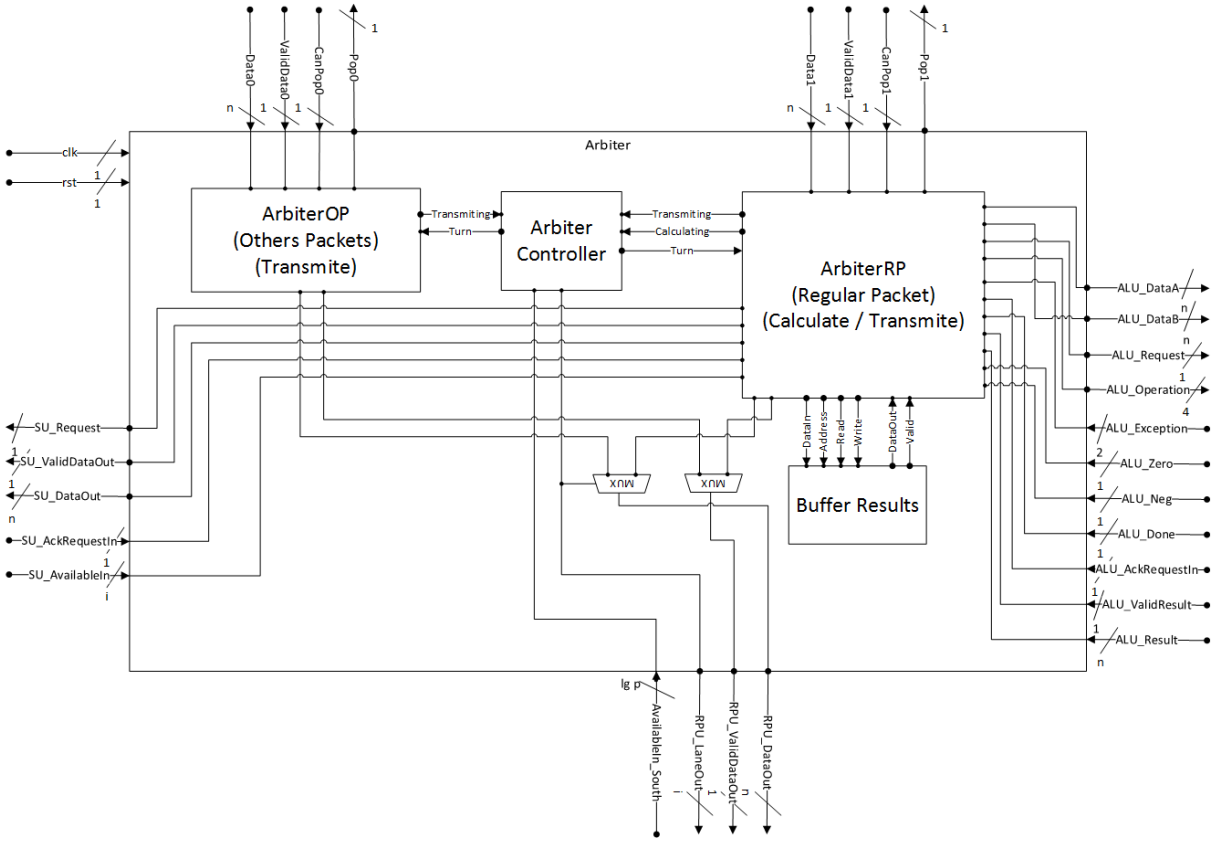
Quando o controlador verifica que a palavra de fim de pacote foi entregue do *buffer* ao Árbitro, então a respectiva trava é desfeita e marcada como “não-reservado”.

4.4 Arbiter

A Figura 23 ilustra os quatros componentes do *Arbiter* (ou Árbitro): *ArbiterOP*, *ArbiterRP*, *Arbiter Controller* e *Buffer Results* (ou *Buffer* de Resultados). Há um componente responsável por cada canal virtual. O *ArbiterOP* é o componente responsável somente por transmitir, por um canal, os pacotes de controle, interrompidos e *caller*. Enquanto que o *ArbiterRP* (Figura 24) manipula e transmite, por outro canal, apenas os pacotes regulares. Antes de transmitir, o *ArbiterRP* realiza a execução da primeira instrução do pacote. O tipo da instrução irá definir se a ALU será utilizada (e depois ter o resultado salvo no *Buffer Results*) ou se o pacote de controle será construído na SU. Os dados recebidos ou enviados pelo *ArbiterRP* são armazenados temporariamente em registradores. O *Arbiter Controller* gerencia

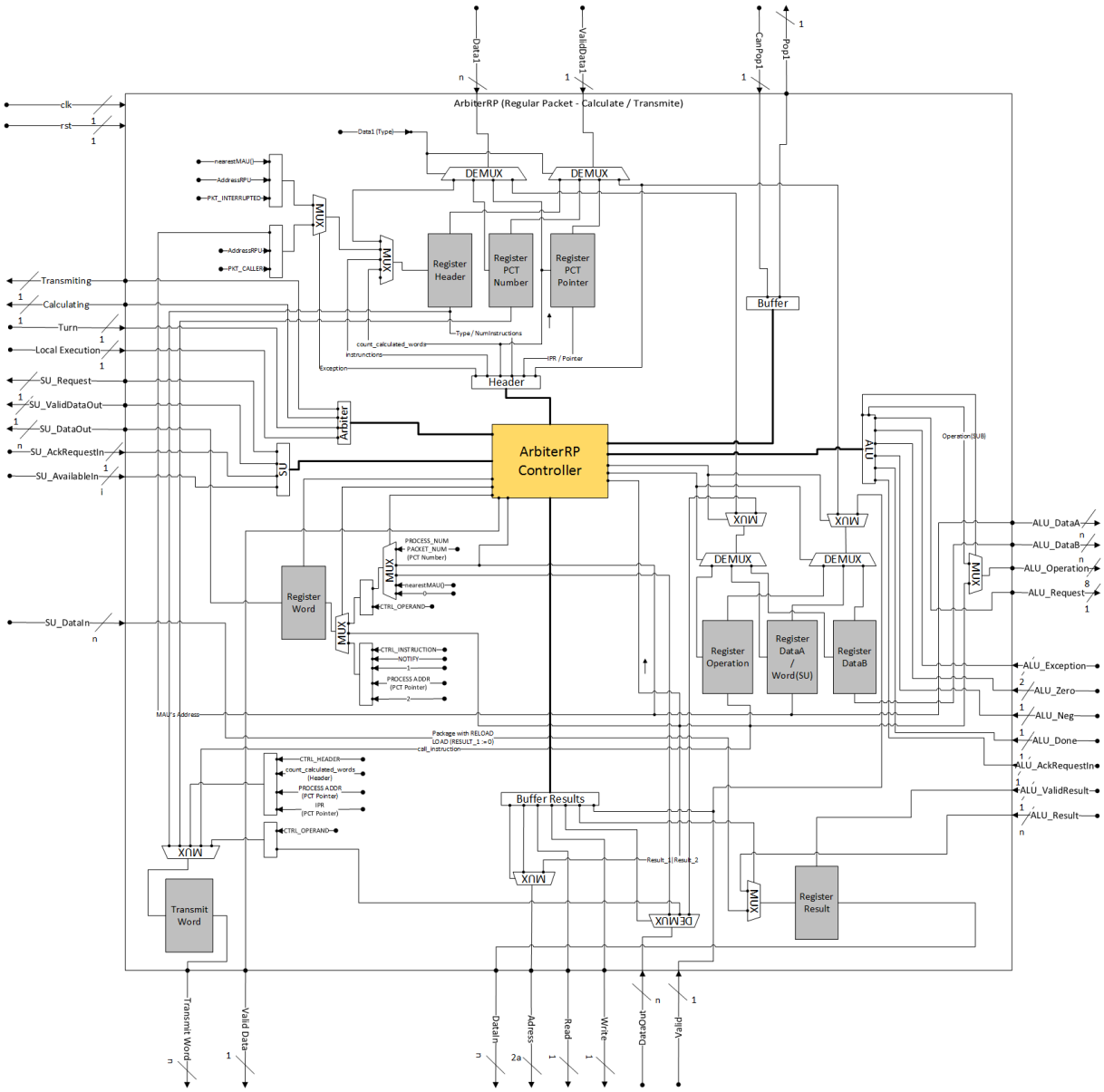
disputas pelo canal físico. Caso haja dois pacotes a serem transmitidos, o *Arbiter Controller* fará o chaveamento *wormhole*, ou seja, as palavras dos pacotes serão enviadas de forma alternada. Através do sinal *Turn*, é dado a permissão para o componente que transmitir e, por meio da porta *LaneOut*, é indicado à RPU seguinte em qual canal virtual a palavra do pacote deve ser armazenada. Neste trabalho será abordado apenas o componente *ArbiterRP* e o Buffer de Resultados, que tratam pacotes regulares, uma vez que o canal virtual para pacotes de controle ainda não foi implementado.

Figura 23 – Visão geral do Árbitro.



Fonte: Próprio autor.

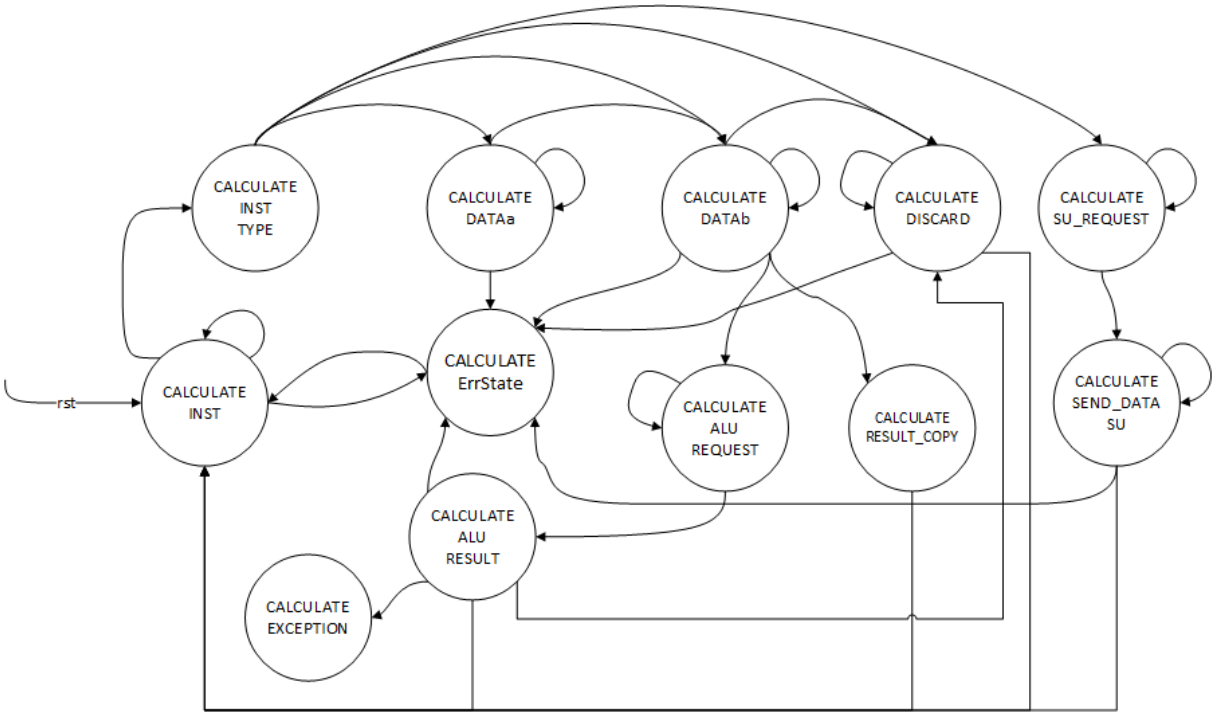
Figura 24 – Visão geral do *ArbiterRP*.



Fonte: Próprio autor.

O ArbiterRP Controller implementa uma FSM para executar uma instrução e outra para transmitir o pacote. Elas são utilizadas de forma alternada e independente, onde cada estado trata uma palavra do pacote. Caso o pacote seja regular, a FSM que calcula (Figura 25) é ativada e, com isso, identifica o tipo da instrução. Caso seja uma instrução de controle, o Árbitro solicita à SU que crie um pacote de controle com tal instrução. Entretanto, esta funcionalidade não será abordada em virtude do canal virtual (por onde o pacote de controle deve ser transmitido) ainda não ter sido implementado.

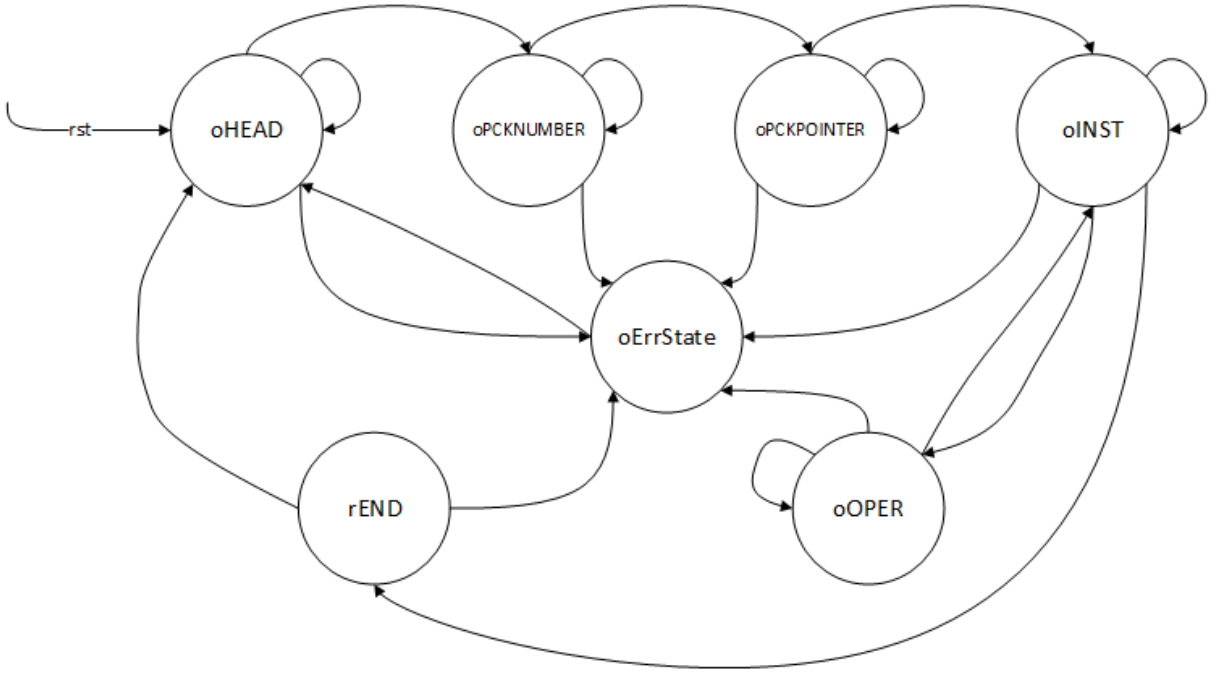
Figura 25 – Máquina de Estado para executar uma instrução no Árbitro.



Fonte: Próprio autor.

Por outro lado, caso seja uma instrução regular, a requisição é feita à ALU para que execute a instrução com seus respectivos operandos. Se a ALU estiver disponível, uma resposta à requisição é enviada e, posteriormente, o resultado da computação, retornado pela ALU, é armazenado no Buffer de Resultados, contido no árbitro requisitante. Finalizado a execução, a FSM que transmite (Figura 26) é ativada e a transmissão do pacote para a próxima RPU é iniciada. Um contador é incrementado a cada palavra transmitida. Se houver uma palavra válida, no Buffer de Resultados, na posição do contador, então a palavra é inserida no pacote e a transmissão das demais palavras é retomada. Senão, o buffer do roteador é consultado.

Figura 26 – Máquina de Estado para transmitir um pacote pelo Árbitro.



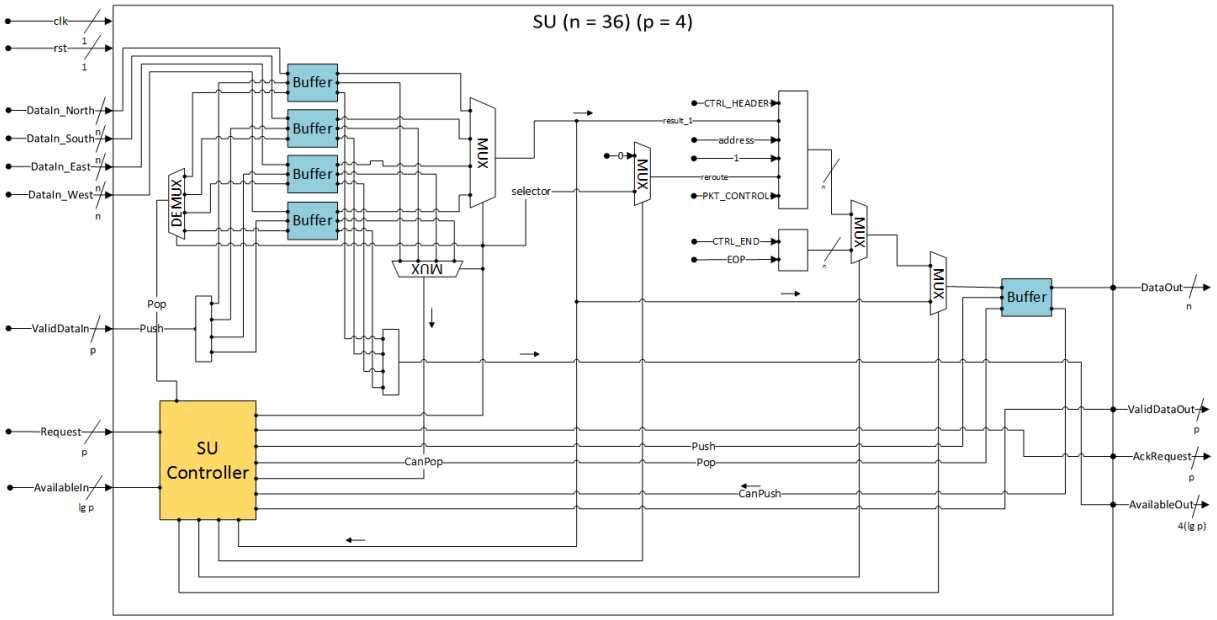
Fonte: Próprio autor.

Antes de transmitir o árbitro verifica o espaço disponível no buffer de entrada da RPU que receberá o pacote. Caso não haja espaço suficiente, o Árbitro entra no estado de execução localizada, ou seja, ativa a FSM que calcula e executa a próxima instrução do pacote. Em seguida, volta a verificar o espaço disponível da RPU destino até que seja possível a transmissão ou até não haver mais instruções no pacote.

4.5 SU

A SU (*Syncronization Unit*) é composto por quatro *buffers* nas entradas, um *buffer* na saída e o *SU Controller*, ilustrado na (Figura 27). Esta entidade gera um pacote de controle a partir de informações da instrução e operando(s) enviados por um Árbitro. Este pacote é composto pela primeira palavra de um novo cabeçalho, juntamente com tal instrução e operandos recebidos e a palavra de fim de pacote.

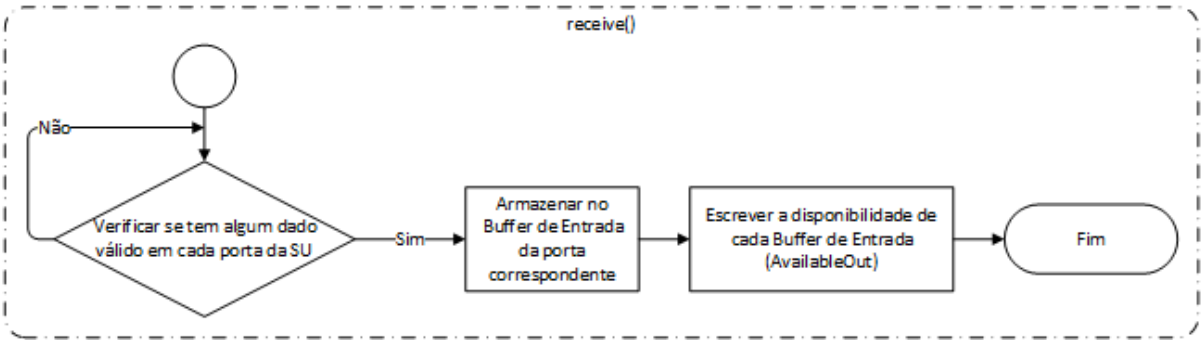
Figura 27 – Visão geral da SU.



Fonte: Próprio autor.

Os buffers de entrada armazenam paralelamente os dados válidos passados pelos Árbitros vinculados à cada porta, conforme apresentado no fluxograma da Figura 28.

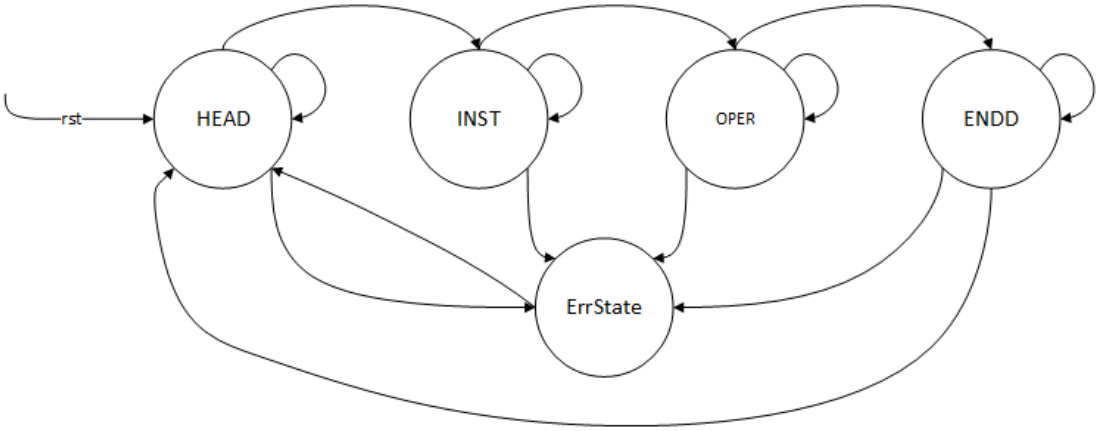
Figura 28 – Fluxograma do funcionamento dos *buffers* de entrada da SU.



Fonte: Próprio autor.

O *SU Controller* implementa uma versão modificada do algoritmo *Round Robin* (KRILL, 2016) para receber e agendar as requisições dos Árbitros e resolver disputas, selecionando o *buffer* de entrada que enviará os dados para criação do pacote de controle. O *SU Controller* também possui uma FSM que contém um estado para cada tipo de palavra do pacote, como mostra a Figura 29.

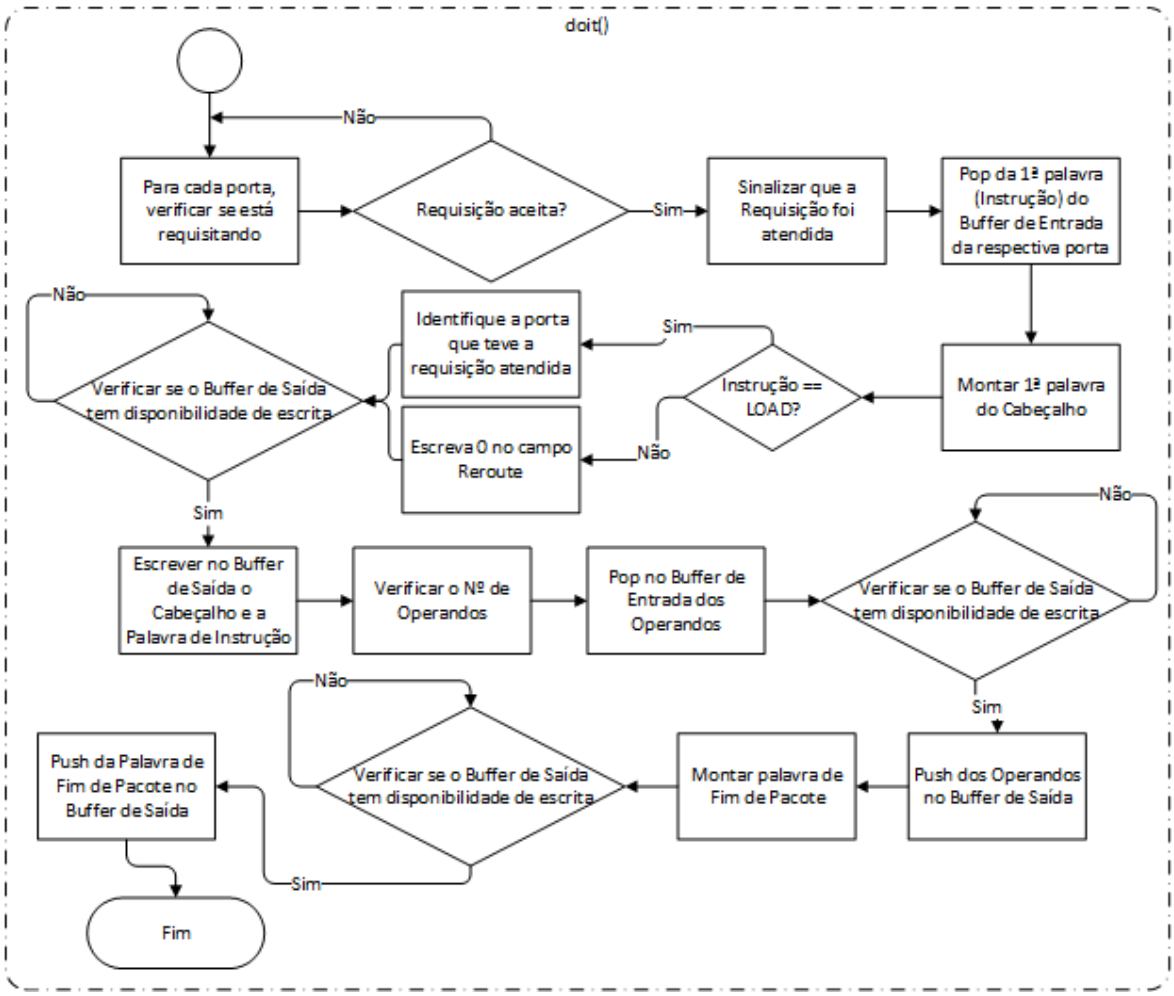
Figura 29 – Máquina de Estados da SU para montar pacote de controle.



Fonte: Próprio autor.

Esta FSM é responsável por montar o pacote de controle (escolhendo quais dados serão multiplexados para formar uma palavra do pacote) e, em seguida, inserir no *buffer* de saída da SU. Essa lógica de requisição e montagem é mostrada no fluxograma da Figura 30.

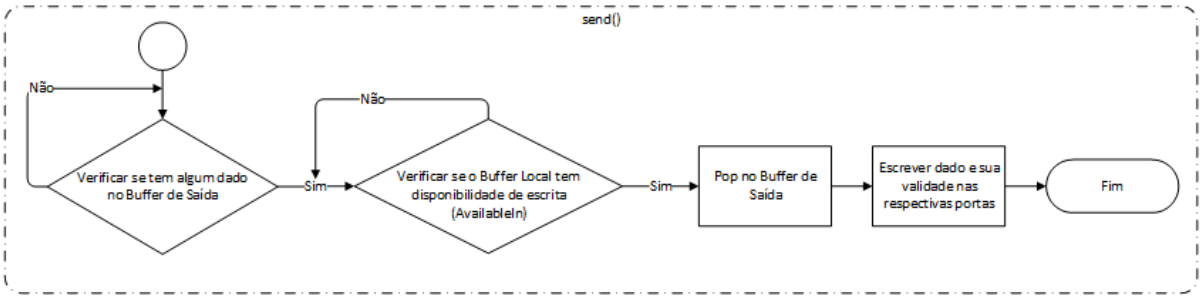
Figura 30 – Fluxograma da montagem do pacote de controle na SU.



Fonte: Próprio autor.

Na medida que dados são inserido no *buffer* de saída, estes vão sendo encaminhados para o *Buffer Local* (contido na RPU), de acordo com o espaço disponível, ilustrado no fluxograma da Figura 31. Quando a palavra de fim de pacote é, então, transmitida para o *Buffer Local*, o *SU Controller* recebe um sinal de *done*, informando que o ciclo de criação de pacote de controle foi concluído e que está disponível para o próximo requisitante. Com isso, a FSM retorna ao seu estado inicial.

Figura 31 – Fluxograma do funcionamento do *buffer* de saída da SU.



Fonte: Próprio autor.

4.6 ALU

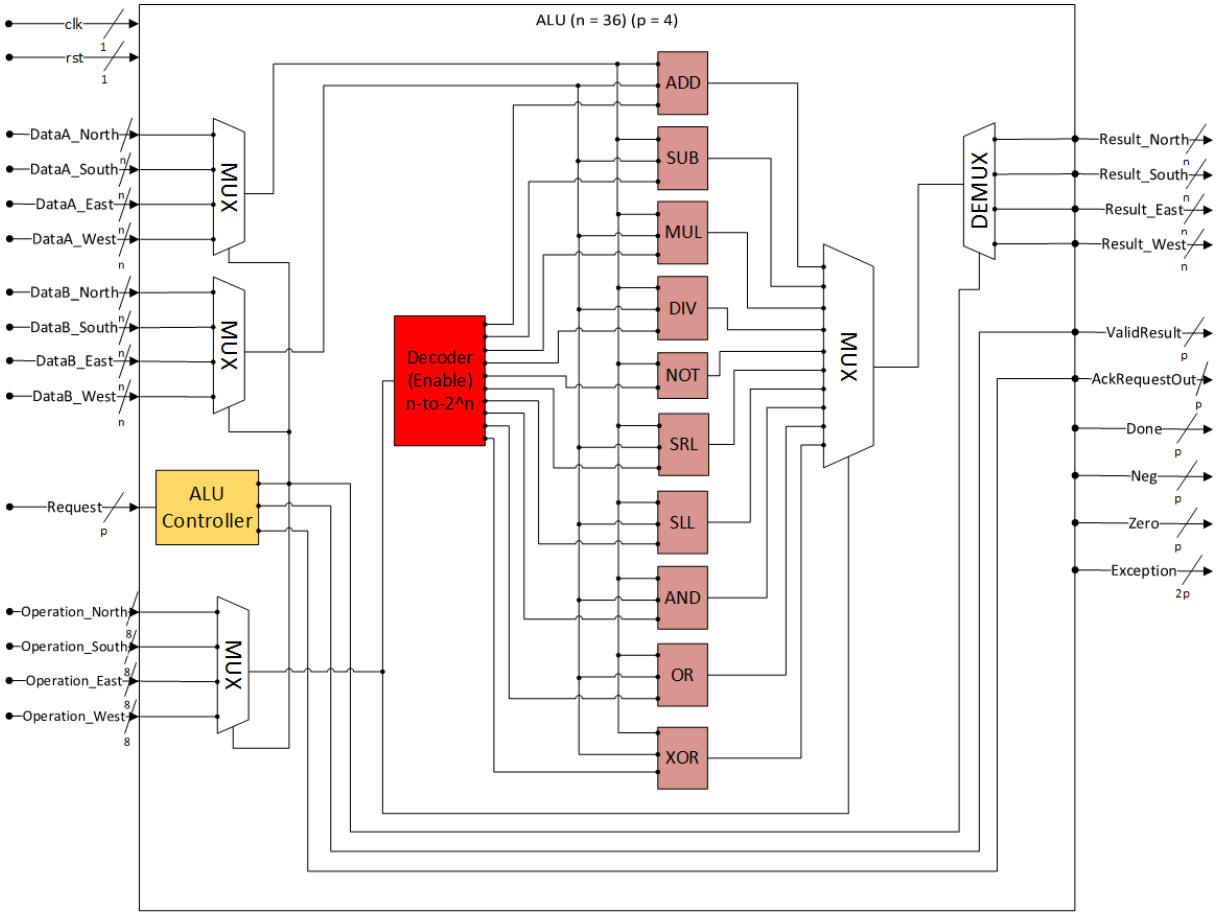
A ALU (*Arithmetic Logic Unit*) (Figura 32) é capaz de realizar a execução das seguintes instruções: adição, subtração, mutiplicação e divisão de inteiros; deslocamento à direita e à esquerda; e as operações lógicas NOT, AND, OR e XOR.

O *ALU Controller* também implementa uma versão modificada do algoritmo *Round Robin* (KRILL, 2016) para receber e agendar as requisições dos Árbitros e resolver disputas, selecionando o Árbitro que será atendido, que, por sua vez, envia os dados da instrução e do(s) operando(s).

Ao atender uma requisição, os bits referentes à operação são encaminhados para um decodificador que ativa somente o componente da operação desejada. Após isso, os o resultado da computação é retornado, juntamente com as *flags*: *ValidResult*, *Neg*, *Zero* e *Exception*, caso o resultado seja válido, negativo, igual a zero e se houve uma exceção, respectivamente.

Por fim, o sinal de *done* é enviado à ALU Controller para informar que a execução foi concluída com sucesso e que está disponível para o próximo requisitante.

Figura 32 – Visão geral da ALU.



Fonte: Próprio autor.

4.7 Buffer Results

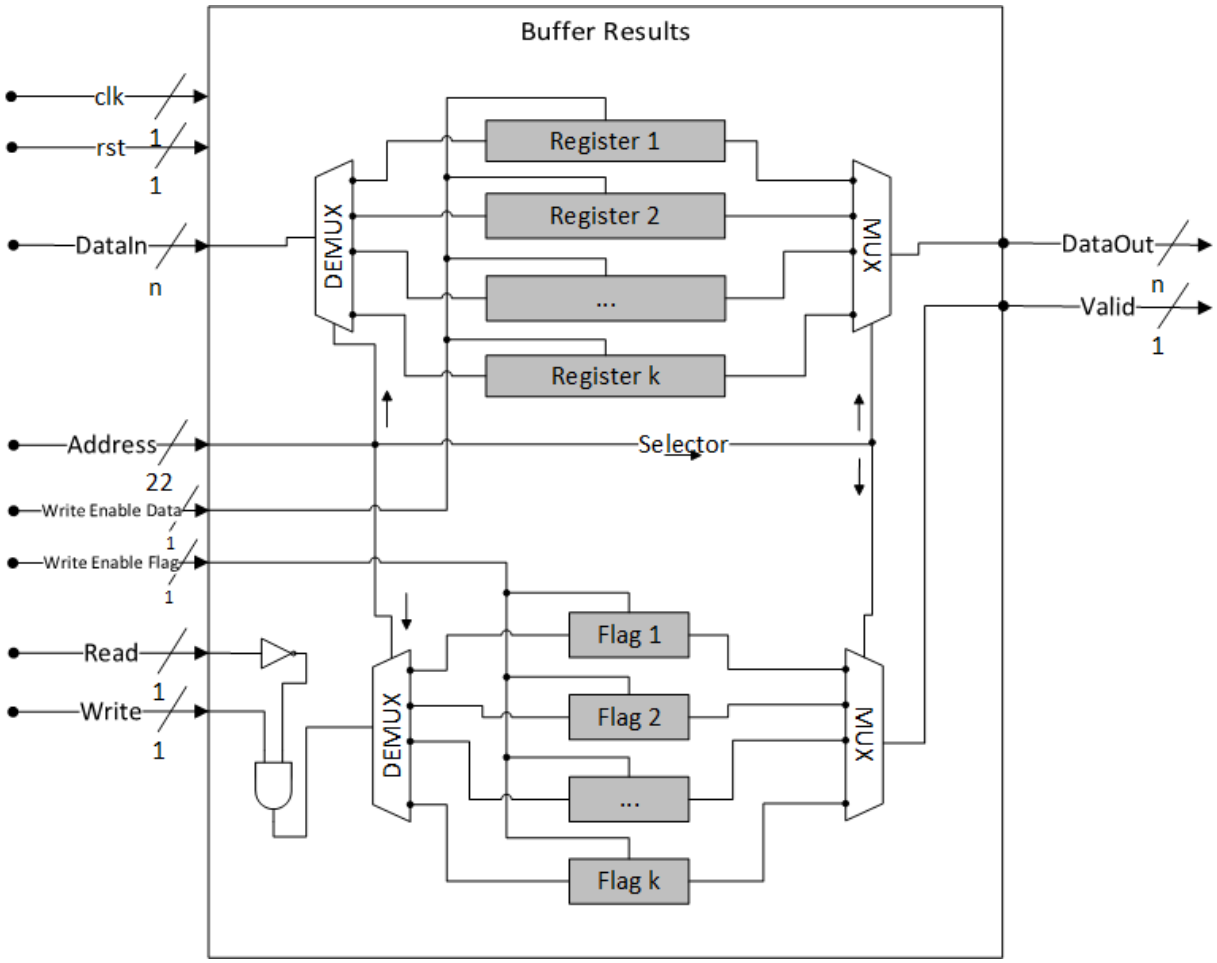
O Buffer de Resultados (Figura 33), contido no Árbitro, é o componente responsável por armazenar os resultados da computação feito na ALU e, além disso, durante a transmissão do pacote, é feito a retirada do dado guardado em uma determinada posição, caso ele seja válido, e inserida no pacote, pelo Árbitro, na mesma posição.

O armazenamento dos dados é realizado por meio de um banco de registradores para os dados e outro para as *flags*. O circuito lógico combinacional ligado às portas *Read* e *Write* atuam como indicador de dado válido, ou seja, quando uma operação de *Read* acontece, então é escrito no registrador da *flag* o valor 0. Caso seja uma operação de *Write*, o valor escrito será 1.

A porta de entrada de endereço representa a posição onde será guardado ou resgatado um dado. É possível escrever um dado em duas posições ao mesmo tempo, exceto se ambas forem iguais. Entretanto, a leitura só pode ser feita de uma posição por vez. Assim, os bits

passados por parâmetro, variam de 22 bits à 11 bits, respectivamente, conforme operação realizada.

Figura 33 – Visão geral do *Buffer* de Resultados.



Fonte: Oliveira Neto e Fernandes (2015).

A Figura 34 apresenta um trecho de código referente à implementação do *Buffer Results*. Para simplificar a forma de armazenamento dos dados, o banco de registradores foi construído por meio de um *array* bidimensional, considerando a quantidade máxima de posições do buffer de resultados (definido por *RESULT_SIZE*) e o tamanho do registrador (definido por *LEN_WIDTH*). Note que a quantidade total de bits do registrador equivale ao tamanho da palavra mais um, ou seja, o bit mais significativo da palavra corresponde ao local onde a *flag* é salva, aproveitando a estrutura do banco de registradores já definida. Além disso, durante a escrita, é verificado se os endereços recebidos não ultrapassam o limite superior das posições, definido por *RESULT_SIZE - 1*.

Figura 34 – Trecho de código do *Buffer* de Resultados.

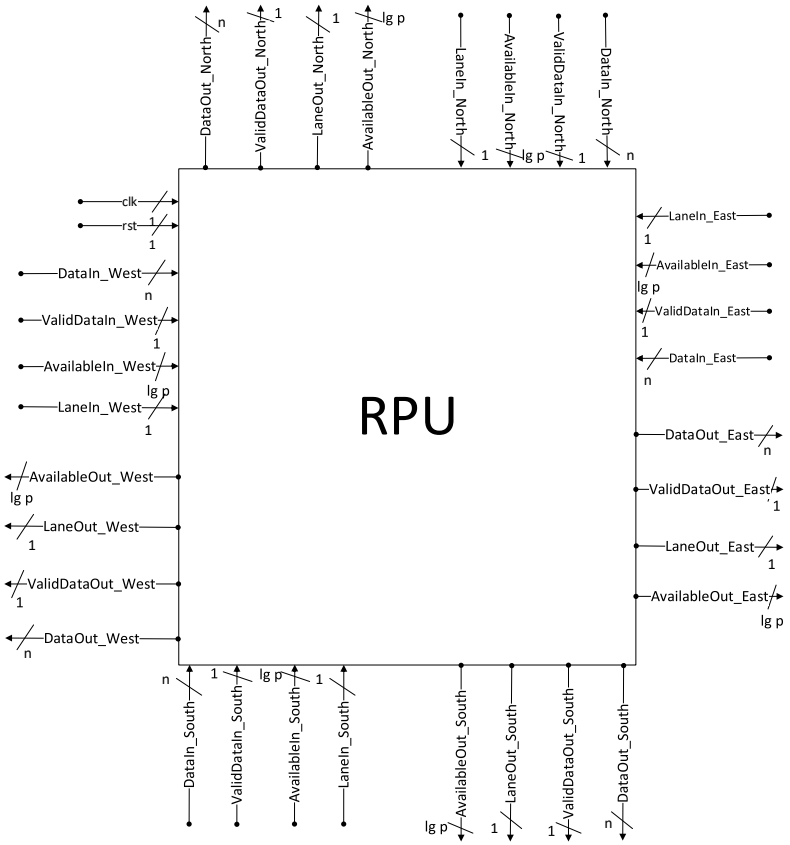
```
-------------------------------------------
ARCHITECTURE behaviour OF buffer_results IS
-------------------------------------------
TYPE STOREAGE_TYPE IS ARRAY(RESULT_SIZE-1 DOWNTO 0) OF STD_LOGIC_VECTOR(LEN_WIDTH DOWNTO 0);
SIGNAL registers : STOREAGE_TYPE;

SIGNAL addrR1 : INTEGER RANGE 0 TO RESULT_SIZE-1;

BEGIN
------------
PROCESS(rst, clk)
------------
VARIABLE addrW1 : INTEGER;
VARIABLE addrW2 : INTEGER;

  BEGIN
    IF (rst = '1') THEN
      FOR i IN 0 TO (RESULT_SIZE-1) LOOP
         registers(i) <= (OTHERS => '0');
      END LOOP;
    ELSIF rising_edge(clk) THEN
      IF (wr = '1') THEN
         addrW1 := to_integer(unsigned(addressWrite(B_RESULT_1 DOWNTO E_RESULT_1)));
         addrW2 := to_integer(unsigned(addressWrite(B_RESULT_2 DOWNTO E_RESULT_2)));
         IF (addrW1 = addrW2) THEN
           IF (addrW1 <= RESULT_SIZE-1) THEN
             registers(addrW1)(LEN_WIDTH-1 DOWNTO 0) <= data_in;
             registers(addrW1)(LEN_WIDTH) <= '1';
           END IF;
         ELSE
           IF (addrW1 <= RESULT_SIZE-1) THEN
             registers(addrW1)(LEN_WIDTH-1 DOWNTO 0) <= data_in;
             registers(addrW1)(LEN_WIDTH) <= '1';
           END IF;

           IF (addrW2 <= RESULT_SIZE-1) THEN
             registers(addrW2)(LEN_WIDTH-1 DOWNTO 0) <= data_in;
             registers(addrW2)(LEN_WIDTH) <= '1';
           END IF;
         END IF;
      ELSIF (rd = '1') THEN
         registers(addrR1)(LEN_WIDTH) <= '0';
      END IF;
    END IF;
END PROCESS;
  addrR1 <= to_integer(unsigned(addressRead));
   -- perform reading ports
  data_out <= registers(addrR1)(LEN_WIDTH-1 DOWNTO 0);
  valid_data <= registers(addrR1)(LEN_WIDTH);

END behaviour;
```

Fonte: Próprio autor.

4.8 Malha 2D 4x4

Uma malha 2D corresponde ao último passo do projeto desta arquitetura (em relação aos núcleos), onde é feito interligação de várias RPUs em uma rede, tornando-se, portanto, a entidade *top level* deste projeto. A Figura 35 explicita os sinais de entrada e saída de uma RPU.

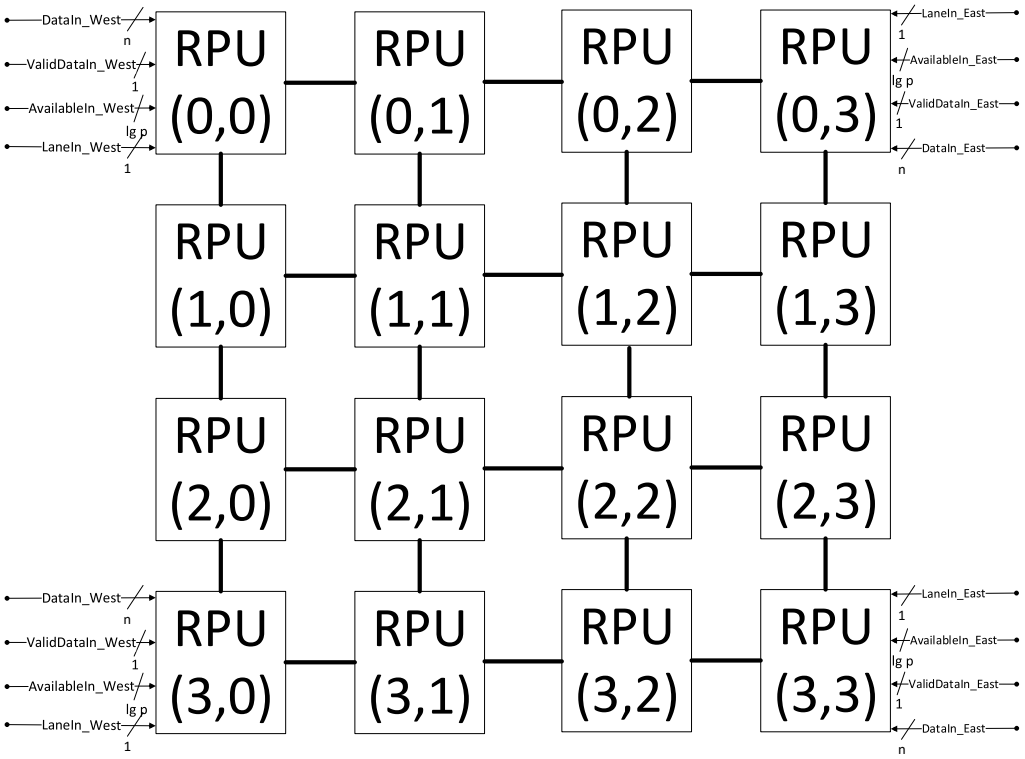
Figura 35 – Portas de entrada e saída de uma RPU para cada direção (Norte, Sul, Leste, Oeste).



Fonte: Próprio autor.

Estes sinais serão conectados aos sinais de seus vizinhos, conforme ilustrado na Figura 36, permitindo, assim, que um pacote possa ser injetado em um dos quatro cantos na rede (simulando o trabalho de uma MAU), roteado através de um algoritmo, processado durante seu percurso e, com isso, ter todo um programa executado (e não mais uma única instrução). Além disso, é possível explorar o paralelismo desta arquitetura com a inserção simultânea de pacotes.

Figura 36 – Visão geral da malha 2D 4x4 de RPUs.



Fonte: Próprio autor.

5 RESULTADOS

Com o objetivo de validar a implementação dos componentes abordados anteriormente, casos de teste foram elaborados de modo a verificar várias possíveis situações em cada entidade. Esta verificação foi realizada por meio de testes automatizados (*scripts*) e simulações. Além disso, resultados da síntese da malha 2D 4x4 de RPUs serão analisados e comparados com duas versões de implementações já feitas da RPU. A compilação foi realizada através do software Quartus II 16.0, enquanto que a simulação envolveu o software ModelSim, ambos da fabricante Altera (ALTERA, 2017).

Essas versões da RPU foram discutidas em (OLIVEIRA NETO e FERNANDES, 2015) e em (NETO e FERNANDES, 2016). A primeira versão expõe resultados de simulação preliminares em uma RPU simplificada, uma vez que esta realiza somente processamento sequencial (apenas um fluxo de execução) de pacotes regulares. A segunda versão discorre sobre a implementação de uma RPU paralela, mostrando resultados para algumas situações de processamento paralelo (múltiplos fluxos de execução), como disputa por recursos e execução localizada. A versão apresentada neste trabalho consiste no aprimoramento das versões anteriores para formar uma malha 2D 4x4 de RPUs, possibilitando a execução completa de vários pacotes através da rede de interconexão das RPUs.

5.1 Estudo de casos

A Tabela 2 apresenta a lista que casos de teste abordados nesta seção. Esta lista foi numerada para que os casos pudessem ser referenciados durante a discussão. Adicionalmente, os testes foram classificados a partir do componente a ser testado e do tipo de teste (unidade, integração e sistema). Quando conveniente, trechos de código dos *scripts* serão mostrados. Para melhorar o entendimento e a leitura das simulações, na medida em que a complexidade dos casos de teste aumenta, alguns dados serão apresentados em hexadecimal (não mais em binário) e modelos de pacotes injetados serão exibidos. Estes modelos contêm de forma objetiva as informações necessárias para compreensão dos resultados das simulações.

As simulações utilizam pacotes sintéticos para testes específicos de cada funcionalidade. Somente o tópico que aborda a malha 2D 4x4 de RPUs é exemplificado uma aplicação real simples para ilustrar a computação e transmissão do pacote durante seu percurso na rede.

Tabela 2 – Lista de casos de teste contemplados.

Nº	Componente	Tipo de Teste	Descrição	Observação
1	Buffer	Unidade	Pop em Buffer vazio	
2		Unidade	Pop e push ao mesmo tempo	
3		Unidade	Push em Buffer cheio	
4	Buffer Results	Unidade	Endereços fora dos limites	
5		Unidade	Escrita em paralelo.	
6		Unidade	Sobrescrita de valores	
7	Round Robin	Unidade	Apenas um elemento solicita e a vez é desse elemento	
8		Unidade	Dois elementos solicitam e a vez é do 1º. Elemento	
9		Unidade	Dois elementos solicitam e a vez é do 2º. Elemento	
10		Unidade	Todos os elementos possíveis solicitam simultaneamente	
11		Unidade	Elementos ficam requisitando mas só é dado a vez quando o *done* é ativado.	
12	Crossbar	Integração	Rotear um único pacote para todas as direções	(ex: oeste para leste, oeste para norte, oeste para sul e oeste para oeste)
13		Integração	Rotear dois pacotes simultaneamente para direções diferentes	(ex: oeste para leste e norte para sul)
14		Integração	Rotear quatro pacotes simultaneamente para direções diferentes	(ex: norte para oeste, oeste para sul, sul para leste, leste para norte)
15		Integração	Rotear quatro pacotes para mesma direção simultaneamente	(ex: oeste para leste, norte para leste, sul para leste e leste para leste)
16		Integração	Rotear quatro pacotes para mesma direção simultaneamente	Testar AckRequest

17		Integração	Rotear três pacotes para mesma direção simultaneamente	(ex: oeste para leste, norte para leste e sul para leste)
18		Integração	Rotear dois pacotes para a mesma direção simultaneamente	(ex: oeste para leste e norte para leste)
19	ALU	Integração	Todas as operações funcionam corretamente? Indicam zero e overflow corretamente?	
20		Integração	Quando há mais de uma solicitação o Round Robin funciona corretamente?	
21	SU	Integração	Consegue montar pacotes de controle para as 6 instruções de controle?	
22		Integração	Quando há mais de uma solicitação o Round Robin funciona corretamente?	
23	RPU (Árbitro)	Sistema	Recebe um pacote com instruções lógico/aritmética executa na ALU transmite e inserindo o resultado	
24		Sistema	Recebe um pacote com instruções lógico/aritmética executa na ALU e quando vai transmitir o buffer de destino não tem espaço – ele deve executar a próxima instrução na ALU e inserir o resultado no lugar correto	
25		Sistema	Recebe vários pacotes que são roteados para saídas diferentes – o round Robin funciona? O pacote prioritário tem sua instrução executada? O pacote é transmitido corretamente inserindo resultado enquanto os	

			outros esperam?	
26		Sistema	Recebe vários pacotes que são roteados para a mesma saída – o round Robin funciona? O pacote prioritário tem sua instrução executada? O pacote é transmitido corretamente inserindo resultado enquanto os outros esperam?	
27	Malha 2D 4x4 de RPUs	Sistema	Aplicação real: o pacote é computado e transmitido corretamente pela rede?	

Fonte: Próprio autor.

5.1.1 Casos de Teste

5.1.1.1 *Buffer*

Os casos de teste para o *Buffer* envolvem realizar: *pop* em *buffer* vazio, *pop* e *push* ao mesmo tempo, e *push* em *buffer* cheio. A Figura 37 mostra um trecho de código para a execução de *pop* e *push* ao mesmo tempo (Caso de teste Nº 2). Os sinais que compõem o trecho de código e a simulação são: *clock* (clk), *reset* (rst), dado de entrada (s_DI), dado de saída (s_DO), *flags* para indicar que dados podem ser lidos (s_CPop) ou escritos (s_CPush) no *buffer*, bem como, o espaço disponível para escrita (s_AO) e os sinais para inserção (s_Push) e retirada (s_Pop) de um dado do *buffer*.

Neste trecho de código, o dado a ser inserido corresponde a um operando de valor 3 e as operações serão de *push* e *pop* simultaneamente. Após isso, o script espera por um período, que equivale a um ciclo de *clock*, e, então confere os resultados mostrados nos sinais conectados às portas de saída. As palavras-chave *Assert* e *Report* são usadas, respectivamente, para: comparar a saída com o resultado esperado e notificar quando o teste falhar.

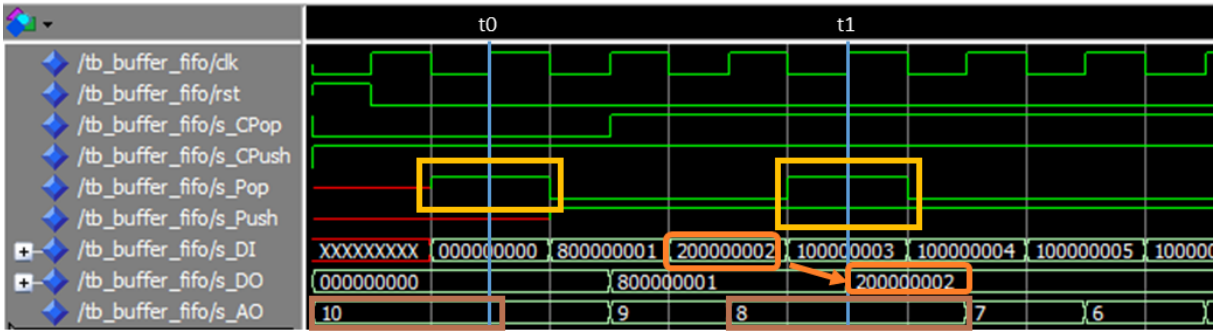
Figura 37 – Trecho de código do *script* de teste do *Buffer* - Caso de teste Nº 2.

```
------------------------------
-- Caso de Teste:
-----------------
-- Push e Pop simultaneamente.
------------------------------
s_DI <= CTRL_OPERAND & std_logic_vector(to_signed(3, LEN_WIDTH-4));
s_Push <= '1';
s_Pop <= '1';
WAIT FOR period;

-- DataOut = Instruction
ASSERT s_DO = CTRL_INSTRUCTION & std_logic_vector(to_signed(2, LEN_WIDTH-4)) AND
       s_Transmit = '1' AND
       s_AO = 2
   REPORT "DataOut 1 OR Transmiting OR Size Buffer failed"
      SEVERITY Error;

s_Pop <= '0';
```

Fonte: Próprio autor.

Na simulação da Figura 38, no instante t0, pode-se verificar que a operação de pop é feita porém nada acontece ao Buffer, pois nenhum dado foi inserido anteriormente. Note que o sinal de *CanPop* está desativado e o espaço disponível ficou inalterado. Já no instante t1, é feito uma inserção e remoção de um dado simultaneamente, logo, o espaço disponível não é alterado, mantendo-se o valor 8, e o dado é removido com sucesso.

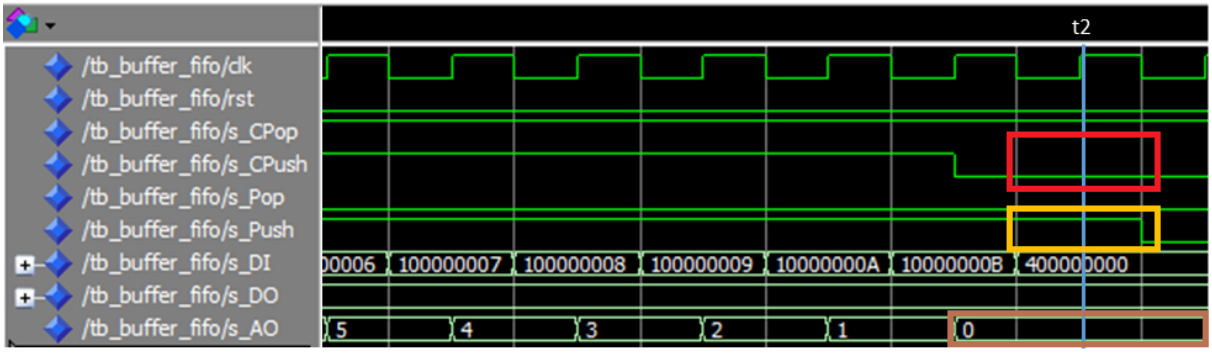
Figura 38 – Resultado da simulação do *Buffer* - Caso de teste Nº 1 e 2.



Fonte: Próprio autor.

A simulação da Figura 39 mostra a adição continuada de dados no buffer, até chegar no seu limite. No instante t2, verifica-se a tentativa sem sucesso de inserção (sinal s_Push), uma vez que não possui espaço para tal operação (conforme indicado na marcação vermelha). Assim, o espaço disponível ficou inalterado, destacado pelo retângulo marrom.

Figura 39 – Continuação do resultado da simulação do *Buffer* - Caso de teste Nº 3.



Fonte: Próprio autor.

5.1.1.2 *Buffer Results*

Os casos de teste para o *Buffer Results* consistem em inserir dados em endereços fora dos limites, realizar escrita em paralelo e sobrescrita de valores. O trecho de código mostrado na Figura 40 corresponde à escrita de um dado em dois endereços e, posteriormente, é realizado leitura em cada endereço individualmente (Caso de teste Nº 5). Os sinais que compõem o trecho de código e a simulação são: *clock* (clk), *reset* (rst), dado de entrada (s_DI), dado de saída (s_DO), uma *flag* para indicar que o dado de saída é válido (s_VD), os endereços de leitura (s_AddrR) e escrita (s_AddrW) no buffer e, principalmente, os sinais para escrita (s_WR) e leitura (s_RD).

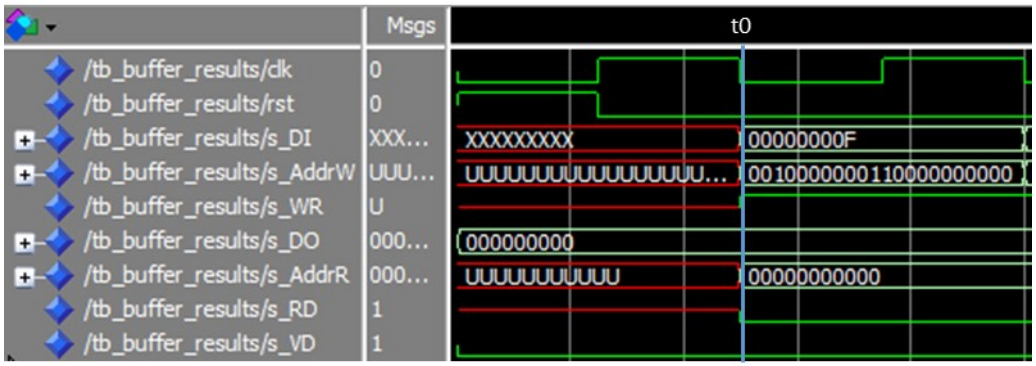
Figura 40 – Trecho de código do script de teste do *Buffer Results* - Caso de teste Nº 5.

```
--------------------------
-- Caso de Teste:
-----------------
-- Escrita em Paralelo.
--------------------------

s_DI <= std_logic_vector(to_signed(32, LEN_WIDTH));
s_AddrW <= "00000000010" & "00000000001";
s_WR <= '1';
s_AddrR <= (OTHERS => '0');
s_RD <= '0';
WAIT FOR period;

-- Leitura

s_DI <= (OTHERS => 'U');
s_AddrW <= (OTHERS => 'U');
s_WR <= 'U';
s_AddrR <= "00000000010";
s_RD <= '1';
WAIT FOR period;

ASSERT s_DO = std_logic_vector(to_signed(32, LEN_WIDTH)) AND
       s_VD = '1'
   REPORT "Read 3 failed"
      SEVERITY Error;

s_DI <= (OTHERS => 'U');
s_AddrW <= (OTHERS => 'U');
s_WR <= 'U';
s_AddrR <= "00000000001";
s_RD <= '1';
WAIT FOR period;

ASSERT s_DO = std_logic_vector(to_signed(32, LEN_WIDTH)) AND
       s_VD = '1'
   REPORT "Read 4 failed"
      SEVERITY Error;
```

Fonte: Próprio autor.

No instante t0, da Figura 41, é exibido uma tentativa de inserir um dado nas posições 257_{10} e 1024_{10}, embora o limite definido seja de 256 posições, isto é, de 0 à 255_{10}. Conforme apresentado no trecho de código da implementação do *Buffer Results*, é verificado se o endereço passado está dentro da faixa de valores permitido. Sendo assim, esta tentativa de escrita é simplesmente ignorada.

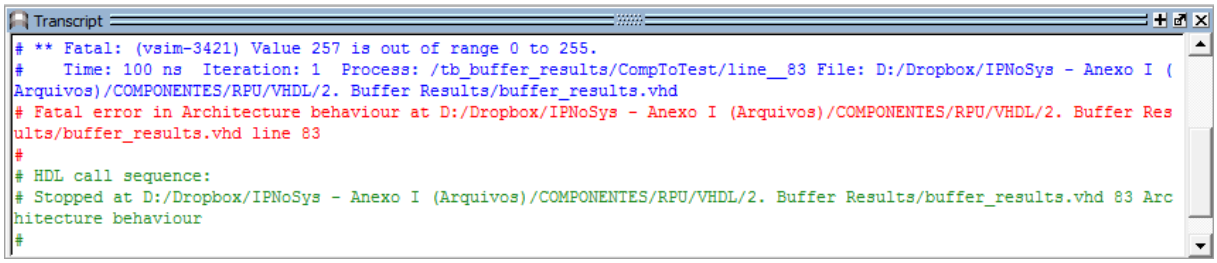
Figura 41 – Resultado da simulação do *Buffer Results* - Caso de teste Nº 4.



Fonte: Próprio autor.

Neste caso, não é possível fazer a leitura desses dados para comprovação do resultado, pois o simulador acusa o erro mostrado na Figura 42, informando que o endereço de leitura precisa estar no intervalo. Por isso, a leitura para este caso não será exibida.

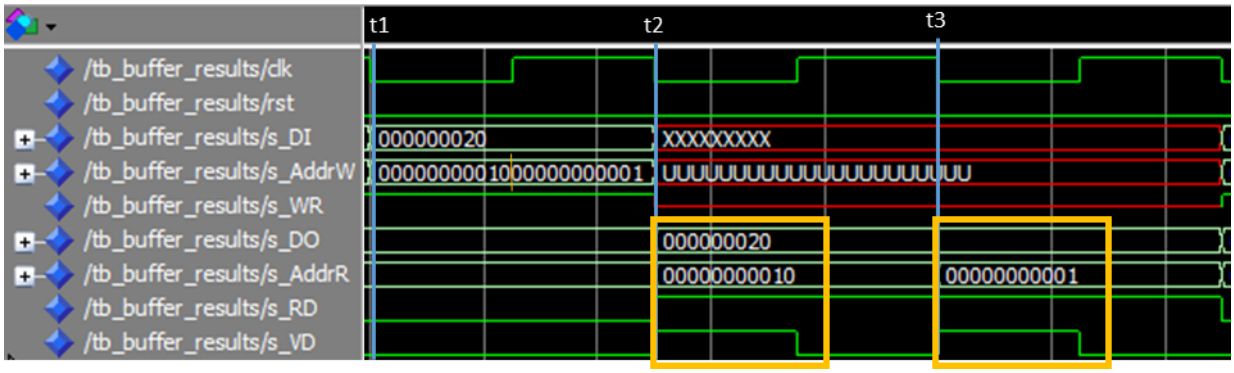
Figura 42 – Erro ao tentar ler uma palavra fora dos limites do *Buffer Results* - Caso de teste Nº 4.



Fonte: Próprio autor.

O instante t1, da Figura 43, expõe a escrita de um dado qualquer, nas posições 2_{10} (00000000010_2) e 1_{10} (00000000001_2), e, em seguida, a leitura do dado (mostrado no sinal s_DO) nos referidos endereços. Note que o sinal de dado válido está ativado nas ambas operações (t2 e t3).

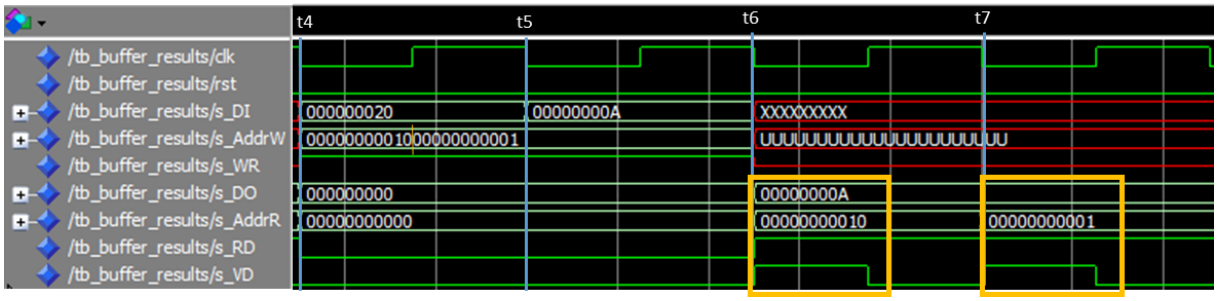
Figura 43 – Resultado da simulação do *Buffer Results* - Caso de teste Nº 5.



Fonte: Próprio autor.

A simulação da Figura 44, mostra a sobrescrita de valores quando selecionado o mesmo endereço para realizar operações de escrita em momentos diferentes. No instante t4, é feito uma inserção com os mesmos parâmetros do caso de teste anterior e, no instante t5, outro dado é escrito nos mesmos endereços. O resultado é apresentado nos instantes t6 e t7, onde é feito a leitura em cada endereço e o dado de saída corresponde à 00000000A, ou seja, houve sobrescrita.

Figura 44 – Resultado da simulação do *Buffer Results* - Caso de teste Nº 6.



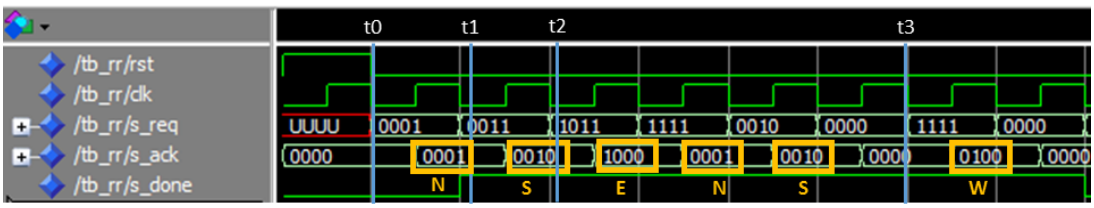
Fonte: Próprio autor.

5.1.1.3 *Round Robin*

O *Round Robin* possui casos de teste que verificam as seguintes situações: apenas um elemento solicita e a vez é desse elemento, dois elementos solicitam e a vez é do primeiro elemento, dois elementos solicitam e a vez é do segundo elemento, todos os elementos possíveis solicitam simultaneamente e elementos ficam requisitando mas só é dado a vez quando o *done* é ativado. A simulação é composta pelos seguintes sinais: reset (rst), clock (clk), vetor de requisições (s_req), vetor de resposta ou confirmação de requisições (s_ack) e o sinal de entrada *done* para receber o aviso de quando uma tarefa foi finalizada. Para esta e demais simulações, o índice do *array* (neste caso, requisições e confirmação) representa uma porta na qual possui uma interface, ou seja: 3 (leste ou *east*), 2 (oeste ou *west*), 1 (sul ou *south*) e 0 (norte ou *north*). A leitura desses vetores deve ser feita da direita para esquerda.

A Figura 45, no instante t0, apresenta o primeiro caso de teste deste componente, em que apenas a porta norte faz uma requisição (s_req[0] = 1) e recebe a confirmação. No instante t2, é possível verificar que, apesar de haver duas requisições (porta norte e sul), apenas a porta sul recebe o *grant*, pois, como a porta norte já foi atendida, agora é a vez da porta sul. O intervalo entre os instantes t2 e t3 correspondem à variação do atendimento às requisições seguindo a ordem definida pelo algoritmo. No instante t3, todos os elementos requisitam, mas apenas um é atendido, no caso, a porta oeste.

Figura 45 – Resultado da simulação do *Round Robin* - Casos de teste Nº 7, 9 e 10.



Fonte: Próprio autor.

No instante t4 da Figura 46, as portas norte e sul estão requisitando, mas apenas o primeiro elemento (porta norte) recebe a confirmação. O instante t5 ilustra o caso em que todas as portas estão requisitando, mas o sinal *done* está desativado, logo, a última confirmação é mantida até que o *done* seja ativado. Note que a ordem do *Round Robin* não foi alterada, isto é, quando o *done* ficou ativo, a ordem de atendimento das requisições retomou a partir da última confirmação realizada (da porta sul para a porta oeste).

Figura 46 – Resultado da simulação do *Round Robin* - Casos de teste Nº 8 e 11.

Fonte: Próprio autor.

5.1.1.4 *Crossbar*

O *Crossbar* promove a interligação entre os roteadores e os Árbitros a fim de mapear uma entrada para uma saída. Para verificar se a funcionalidade está implementada corretamente, os seguintes casos de teste foram construídos: rotear um único pacote para todas as direções, rotear dois pacotes simultaneamente para direções diferentes, rotear quatro pacotes para direções diferentes, rotear quatro pacotes para mesma direção simultaneamente, rotear três pacotes para mesma direção simultaneamente e rotear dois pacotes para a mesma direção simultaneamente. Assim, todas as simulações deste componente são formadas pelos seguintes sinais: clock (clk), reset (rst), uma entrada de dados (s_DIN, s_DIS, s_DIW e s_DIE) e uma saída de dados (s_DON, s_DOS, s_DOW, s_DOE,) para cada porta, dois *arrays* para os sinais de dado válido (para mapear a entrada sig_valid_In e na saída sig_valid_Out), para saber se pode ser feito pop no buffer (novamente, mapeando a entrada sig_canPop_In na saída sig_canPop_Out) e para realizar efetivamente o pop no buffer (mapeando a entrada sig_buf_pop_In na sig_buf_pop_Out), e, finalmente, um *array* para requisição (s_req) e resposta (s_ackReq) e a direção escolhida (isto é, o Árbitro que receberá os dados) para cada porta (s_desired_l, s_desired_n, s_desired_s, s_desired_w, s_desired_e).

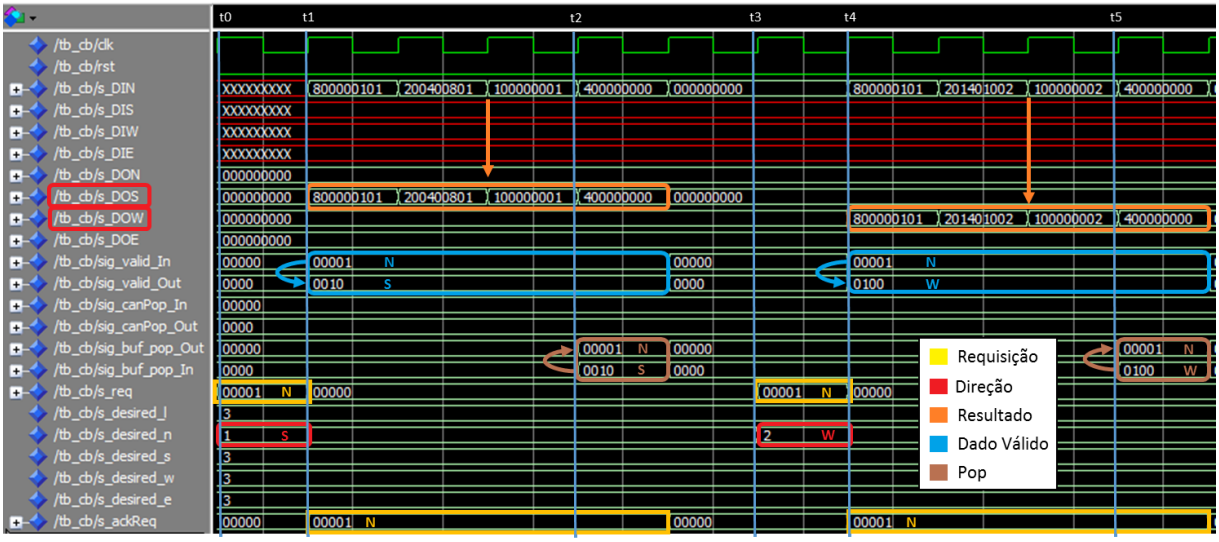
O caso de teste Nº 12 aborda o envio de um pacote qualquer para todas as direções, neste caso, norte, sul, oeste e leste. A simulação da Figura 47 apresenta os resultados para este caso, onde, no instante t0, a porta norte realiza uma requisição ao *Crossbar* para transmitir

pela direção sul (destacado de vermelho). No tempo t1, a requisição é aceita e os dados já começam a ser transmitidos da porta norte para a porta sul, inclusive com seu respectivo sinal de dado válido (do dado atribuído no índice 0 do *array* para o índice 1).

Os dois sinais que representam a operação *pop* são utilizados pelo Árbitro para remover alguma palavra do buffer de entrada (contido no roteador). A afirmação de que o Árbitro opera o *buffer* de forma transparente é verdadeira, pois é o *Crossbar* que gerencia esse mapeamento, como se pode observar na simulação, no instante t2. Logo, o *sig_buf_pop_In*(1) representa o sinal ligado ao Árbitro da direção sul e o *sig_buff_pop_Out*(0) representa o sinal ligado ao roteador (ou buffer de entrada) da direção norte. As setas indicam o sentido do envio dos dados. Apesar de ser necessário efetuar a remoção (*pop*) de cada palavra do buffer, como forma de simplificar, o sinal de *pop* só foi ativado no instante t2, uma vez que, para destravar um mapeamento, é necessário ter recebido a palavra de fim de pacote e o *pop* do *buffer*. Desta forma, indicamos ao *Crossbar* que a última palavra do pacote foi retirada e enviada corretamente para o Árbitro e que não há mais dados. Logo, a trava é desfeita e aquela direção torna-se disponível novamente.

Os instantes t3, t4 e t5 são equivalentes aos instantes t0, t1 e t2, visto que eles realizam as mesmas operações descritas anteriormente e na mesma ordem, apenas diferindo na porta destino onde o pacote está sendo chaveado, neste caso, para a porta oeste e não mais para a porta sul.

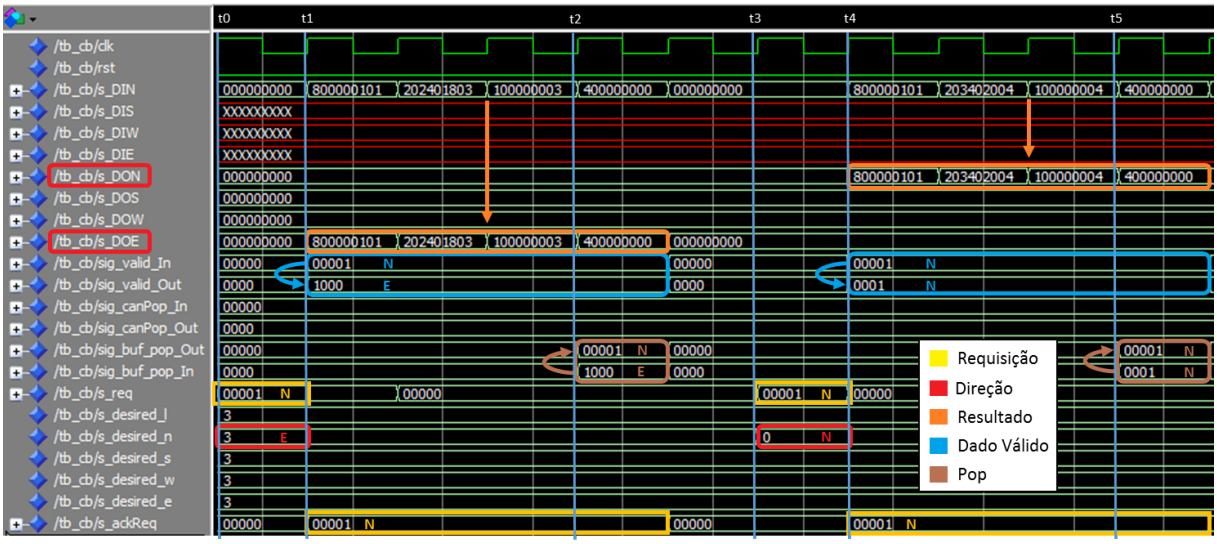
Figura 47 – Resultado da simulação do *Crossbar* para envio de um pacote para várias direções – Parte 1/2 - Casos de teste Nº 12.



Fonte: Próprio autor.

A Figura 48 continua a exibição dos resultados da simulação para o envio de um pacote para demais portas: leste (instantes t0, t1 e t2) e norte (instantes t3, t4 e t5).

Figura 48 – Resultado da simulação do *Crossbar* para envio de um pacote para várias direções – Parte 2/2 - Casos de teste Nº 12.

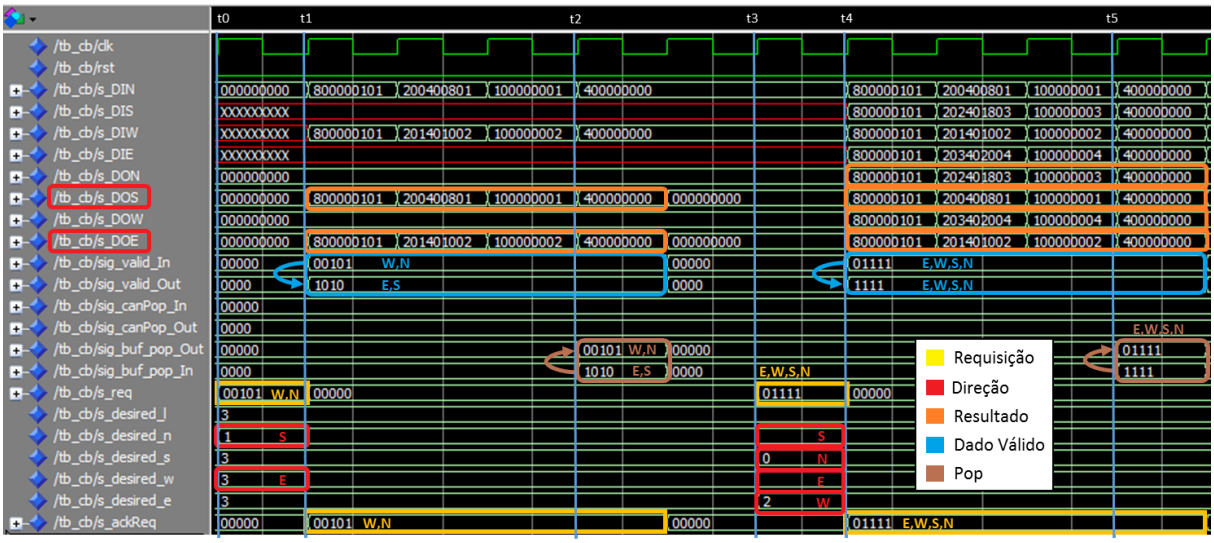


Fonte: Próprio autor.

O segundo e terceiro caso de teste trata do envio de dois e quatro pacotes para diferentes saídas, respectivamente. Os resultados para estes casos são apresentados na Figura 49. Note que no instante t0, as portas norte e oeste requisitam as portas sul e leste, respectivamente. A confirmação é recebida para as duas portas no instante t1. Para diferenciar os pacotes, a terceira palavra dos pacotes (referente ao operando) estão numeradas de 1 à 2 ou 1 à 4, de acordo com o caso de teste. Além disso, note que apenas os índices corretos (correspondente as portas solicitadas) dos *arrays* *sig_canPop_Out* e *sig_buf_pop_Out* estão sendo ativados no instante t1 e t2.

No caso de teste Nº 14, os pacotes estão sendo mapeados: da porta norte para sul, da porta sul para norte, da porta oeste para leste e da porta leste para oeste, conforme direção escolhida durante a requisição, no instante t3. Os instantes t4 e t5 mostram a saída dos dados, se eles são válidos e o momento da remoção da última palavra do pacote.

Figura 49 – Resultado da simulação do *Crossbar* para envio de dois e quatro pacotes para várias direções - Casos de teste Nº 13 e 14.

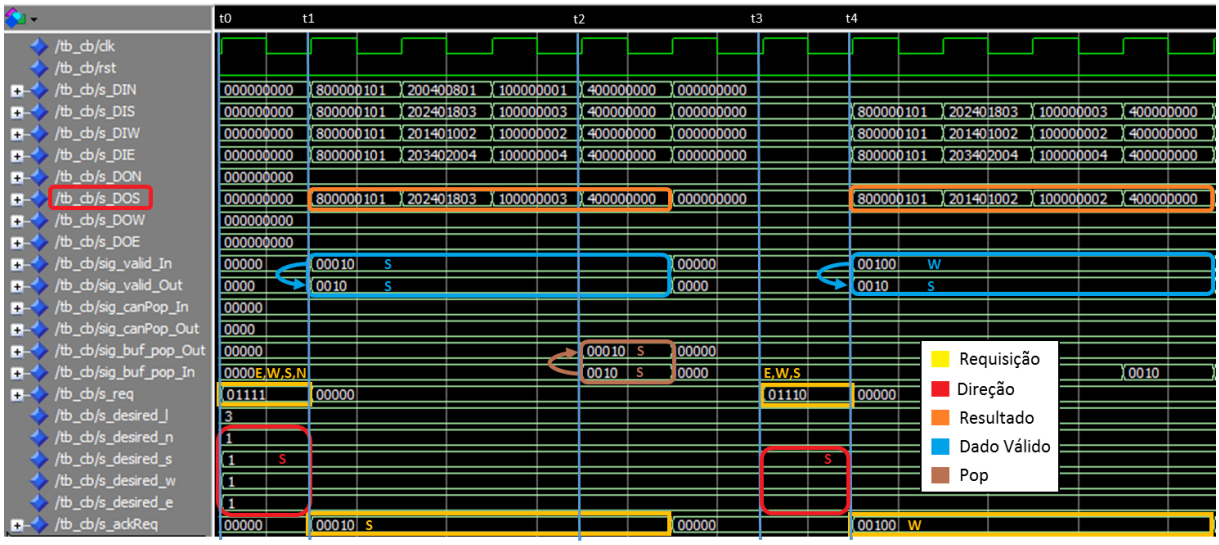


Fonte: Próprio autor.

As simulações mostradas a seguir apresentam o caso de haver concorrência de roteadores para o envio de um pacote para o mesmo Árbitro. Como não é possível atender todos os roteadores ao mesmo tempo, apenas um é escolhido por vez, enquanto os demais aguardam. Na Figura 50, note que durante a requisição (no instante t0), todas as portas requisitam a mesma saída sul (1), entretanto, apenas a porta sul é atendida, conforme exibido no instante t1. Sendo assim, por mais que todas as portas enviem seus pacotes somente a porta agendada terá o pacote roteado, como exemplificado no instante t1 e t2 para o dado transmitido, o sinal de válido e para o *pop*.

No instante t3 trata-se do caso semelhante, em que três pacotes são roteados para mesma saída. Conforme exposto no instante t4, o *Crossbar* resolveu esse conflito através da escolha da porta oeste para transmissão. Note que apesar da porta sul também requisitar, a porta oeste foi a escolhida, uma vez que a porta sul já tinha sido atendida anteriormente.

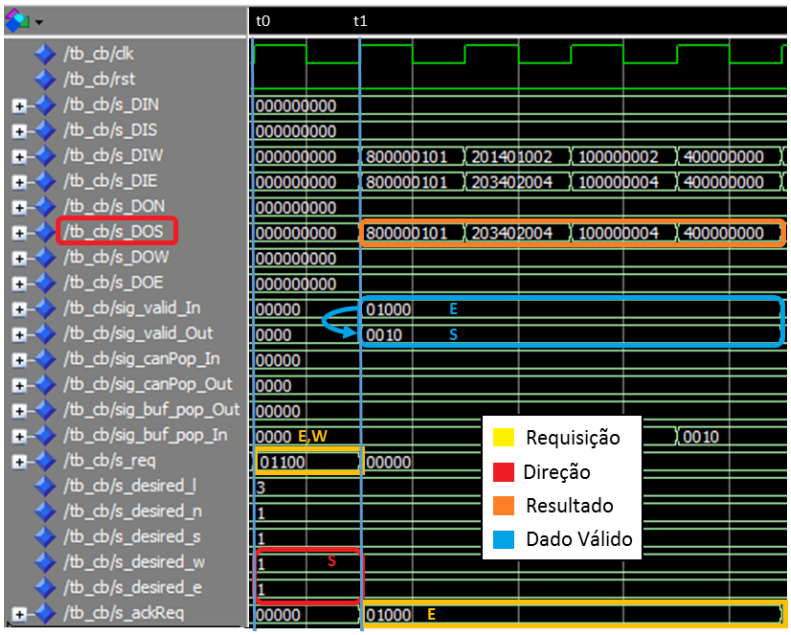
Figura 50 – Resultado da simulação do *Crossbar* para envio de quatro e três pacotes para a mesma direção/saída - Casos de teste Nº 15, 16 e 17.



Fonte: Próprio autor.

Por fim, a simulação da Figura 51 exibe o último caso de teste para o *Crossbar*, em que tem-se a requisição de duas portas para a saída sul, conforme exibido no instante t0. Novamente, o *Crossbar* escolheu uma das duas portas para transmitir a partir do instante t1, neste caso, a porta leste.

Figura 51 – Resultado da simulação do *Crossbar* para envio de dois pacotes para a mesma direção/saída - Casos de teste Nº 18.



Fonte: Próprio autor.

5.1.1.5 ALU

Esta subseção apresenta os resultados para os casos de teste para o componente ALU, são eles: verificar se as operações da ALU geram o resultado correto, juntamente com as flags de exceção, zero e overflow e verificar se o Round Robin funciona corretamente quando há mais de uma solicitação. Para estas simulações foram usados os seguintes sinais: clock (clk), reset (rst), entradas de dados, uma para cada porta, do primeiro operando ou *DataA* (s_AN, s_AS, s_AW e s_AE), do segundo operando ou *DataB* (s_BN, s_BS, s_BW e s_BE), operação (s_ON, s_OS, s_OW e s_OE), bem como, saídas de dados, uma para cada porta, do resultado (s_RN, s_RS, s_OW e s_RE) e exceção (s_EN, s_ES, s_EW e s_EE) e, por fim, *arrays* para os sinais de dado válido na entrada (s_validIn) e saída (s_vR), de resultado negativo (s_neg), de resultado igual a zero (s_zero) e para requisição (s_req) e resposta (s_ack).

A Tabela 3 exibe as operações que podem ser executadas na ALU e seus respectivos códigos que serão referenciados nas simulações.

Tabela 3 – Lista das instruções executadas pela ALU.

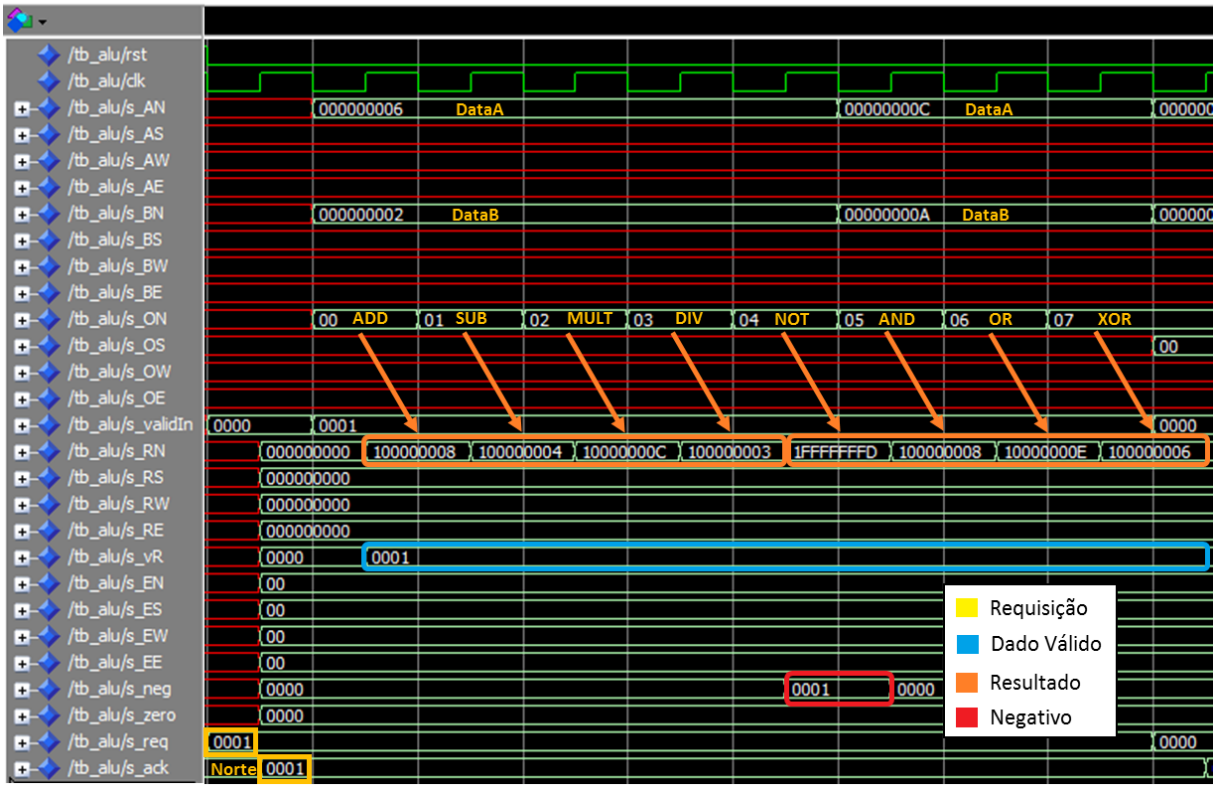
Instrução	Descrição	Tipo da Instrução	Código
ADD	Adição	Aritmética	00
SUB	Subtração	Aritmética	01
MULT	Multiplicação	Aritmética	02
DIV	Divisão	Aritmética	03
NOT	Negação	Lógica	04
AND	E	Lógica	05
OR	Ou	Lógica	06
XOR	Ou exclusivo	Lógica	07
RSHIFT	Deslocamento à direita	Deslocamento	08
LSHIFT	Deslocamento à esquerda	Deslocamento	09

Fonte: Próprio autor.

A Figura 52 exibe os resultados para a simulação da ALU ao executar operações aritméticas e lógicas. Para facilitar o entendimento, neste caso de teste, as operações foram executadas somente pela porta norte. Primeiramente, é feito a requisição à ALU e obtido a confirmação, pois não há concorrência. Note que a requisição está sendo feita em cada ciclo de *clock* para manter uma boa visualização da simulação, permitindo que os resultados fiquem

lado a lado. O próximo passo consiste em submeter os dados à ALU e receber os resultados. A execução de uma instrução leva apenas um ciclo de *clock*. Observe que os resultados correspondem às operações aritmética e, posteriormente, lógicas entre o dado A e dado B. Note que a operação NOT é feita sob o valor $2_{10} = (10)_2$, tendo como retorno o valor $(1FFFFFFFD)_{16} = (00011111111111111111111111111101)_2$ que representa o bit de controle mais o resultado. Além disso, é realizado a operação AND, OR e XOR entre ($C_{16} = 12_{10} = 1100_2$) e ($A_{16} = 10_{10} = 1010_2$), gerando, respectivamente, ($8_{16} = 1000_2$), ($E_{16} = 14_{10} = 1110_2$) e ($6_{16} = 0110_2$).

Figura 52 – Resultado da simulação da ALU para execução de instruções - Parte 1/2 - Casos de teste Nº 19.



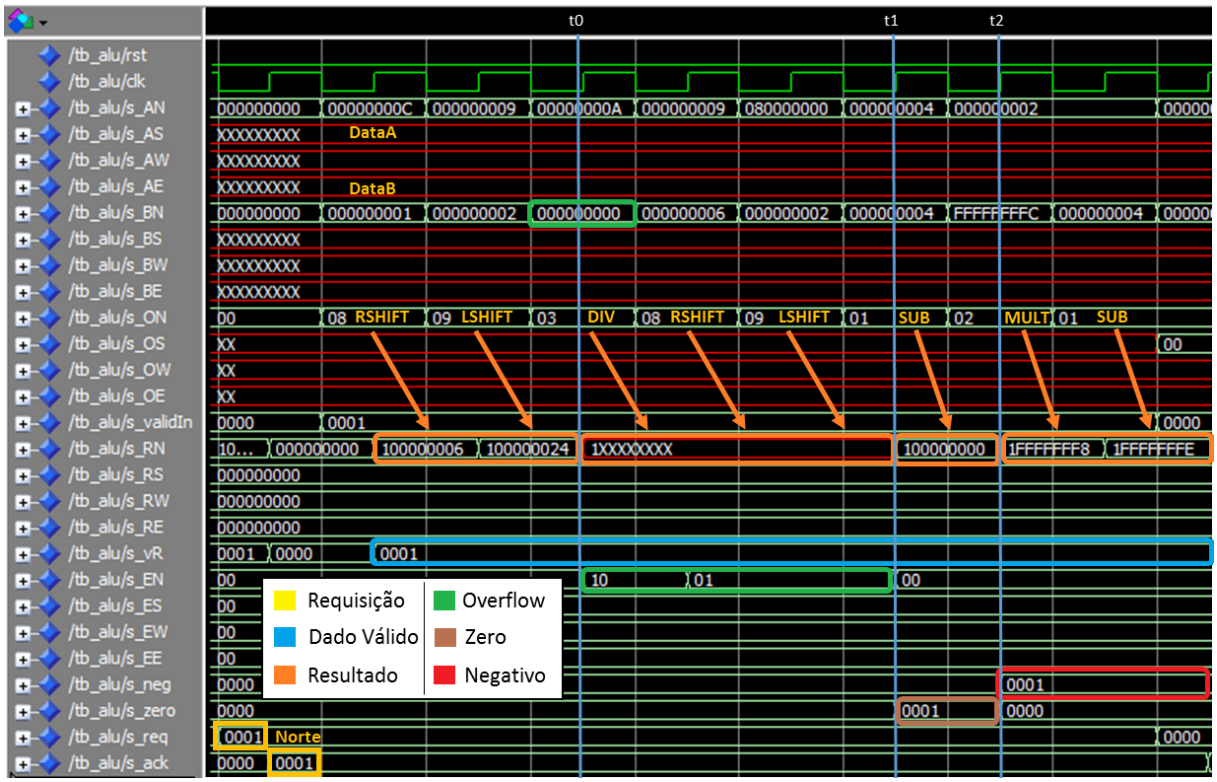
Fonte: Próprio autor.

A simulação da Figura 53 compreende os resultados para as operações de deslocamento, e os resultados de exceção, zero e negativo. Os operandos das instruções de deslocamento implicam no valor e na quantidade de bits que deseja deslocar. Ao deslocar um bit à direita do valor ($C_{16} = 12_{10} = 1100_2$), temos ($6_{16} = 0110_2$). Enquanto que deslocar dois bits à esquerda de de ($9_{16} = 1001_2$), resulta em ($24_{16} = 36_{10} = 0010\ 0100_2$).

O instante t0 apresenta o resultado para o cálculo que envolve divisão por zero e overflow, onde o código do tipo corresponde à 10_2 e 01_2, respectivamente. Neste caso, deseja-se obter apenas a notificação de que houve exceção e o tipo dela. O sinal de válido é ativado

para indicar que, apesar de não haver uma resposta válida, há uma exceção válida. Os ciclos seguintes tratam da situação de overflow ao deslocar um valor acima do limite inferior e superior de um vetor de bits. O instante t1 ilustra o caso de haver uma subtração de dois números iguais, tendo zero como resultado. A *flag* zero é utilizada em instruções de *branch* para verificar se os operandos são iguais. No instante t2 é verificado a multiplicação entre 2 e -4 e subtração entre 2 e 4, tendo como resultados os valores ($0001FFFFFFF8_{16}$ = -8_{10}) e ($0001FFFFFFFE_{16}$ = -2_{10}) e a indicação para tal situação, através da *flag* neg.

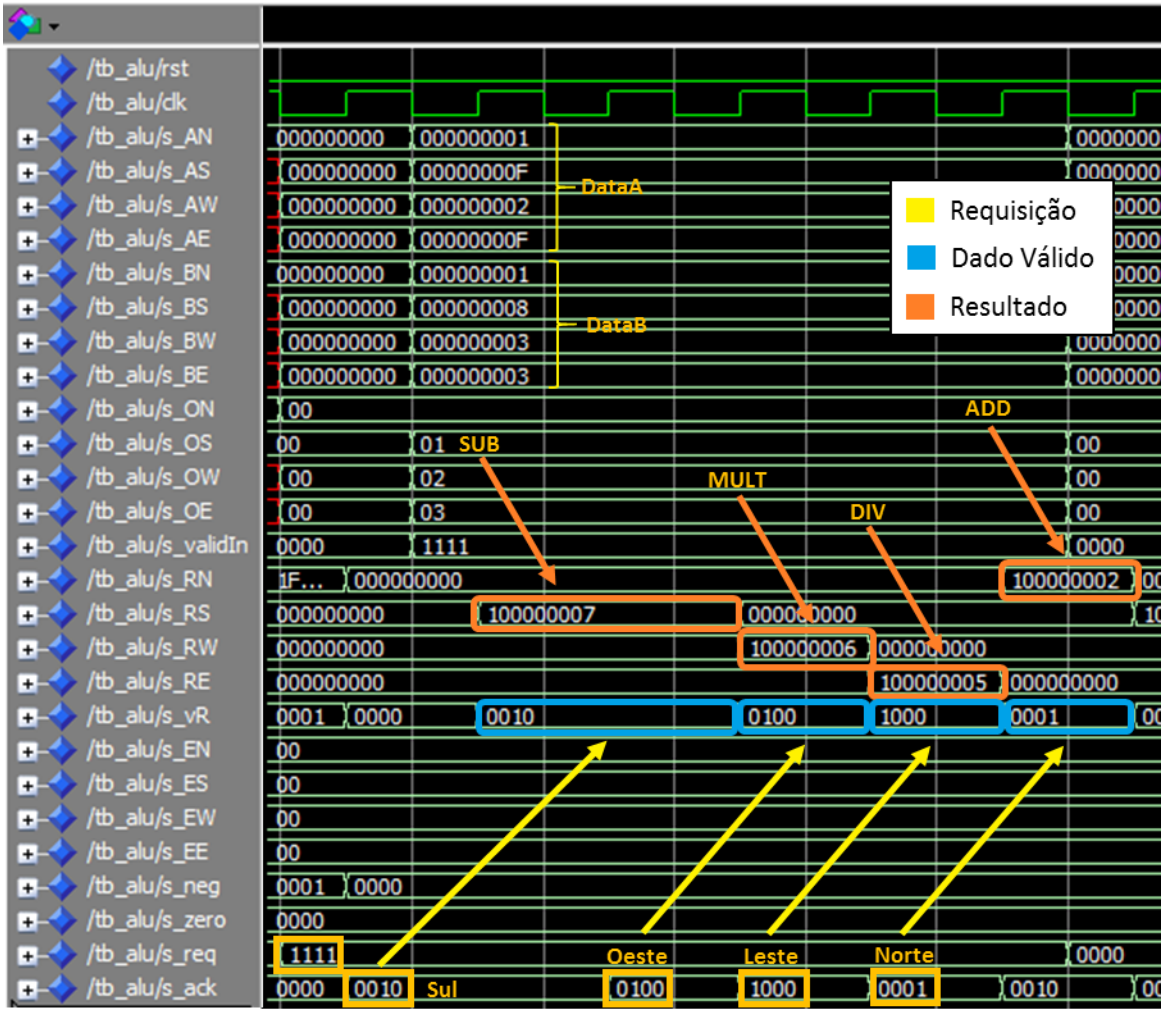
Figura 53 – Resultado da simulação da ALU para execução de instruções – Parte 2/2 - Casos de teste Nº 19.



Fonte: Próprio autor.

A última simulação (Figura 54) deste componente verifica o correto funcionamento do *Round Robin* para resolver disputas entre Árbitros que desejam ter a execução de uma instrução na ALU. Primeiramente, pode-se observar que todas as portas requisitam a ALU mas apenas a porta sul é atendida. No ciclo seguinte, o resultado já é enviado e marcado como válido, isto é, a subtração entre (F_{16} = 15_{10}) e (8_{16}) é igual a (7_{16}). Enquanto isso, todas as portas mantem a requisição e a próxima porta escolhida é a oeste. Os operandos são capturados, a operação é executada e o resultado exibido. Perceba que uma porta já atendida espera até que as demais também sejam atendidas e que apenas a porta que executou alguma instrução tem seu resultado marcado como válido, no índice correto do *array* s_VR.

Figura 54 – Resultado da simulação da ALU para concorrência - Caso de teste Nº 20.



Fonte: Próprio autor.

5.1.1.6 SU

Os casos de teste da unidade de sincronização consistem em verificar se o pacote de controle é construído corretamente para as seis instruções de controle (LOAD, SEND, STORE, EXEC, SYNEXEC e SYNC) e se o *Round Robin* gerencia as requisições corretamente quando há concorrência.

As simulações são composta pelos seguintes sinais: clock (clk), reset (rst), endereço da RPU atual (addr), uma entrada de dados para cada porta (s_DIN, s_DIS, s_DIE e s_DIW) e uma porta para saída de dados (s_DO), bem como, a disponibilidade do *buffer* de entrada de cada porta (s_AON, s_AOS, s_AOE e s_AOW) e do *buffer* de saída (s_AI). Além disso, *arrays* foram utilizados nas requisições (s_req), na resposta ou confirmação de requisições

(s_ackReq) e para indicar quando o dado de entrada é válido (s_vDI). Por fim, o dado de saída é marcado como válido através do sinal s_VDO.

A Figura 55 apresenta um trecho de código do *script* para ilustrar o processo de criação de pacote de controle. No primeiro momento, é feito a requisição, seguida do envio dos dados (instrução mais operando(s)) e, por último, a verificação dos resultados obtidos. Por simplicidade, nesta figura é mostrado apenas o teste para o cabeçalho gerado.

Figura 55 – Trecho de código do script de teste da SU para instrução SEND - Caso de teste Nº 21.

```
--------------------------
-- Caso de Teste:
-----------------
-- Porta North
-- Instrução SEND
-- 2 Operandos
--------------------------

s_req(P_NORTH) <= '1';
WAIT FOR period;
IF (s_ackReq(P_NORTH) = '1') THEN
   s_req(P_NORTH) <= '0';
END IF;

-- NORTH: Instruction Word (SEND)
s_DIN(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
s_DIN(B_INSTRUCTION DOWNTO E_INSTRUCTION) <= std_logic_vector(to_unsigned(SEND, LEN_INSTRUCTION));
s_DIN(B_N_OPERANDS DOWNTO E_N_OPERANDS) <= std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
s_DIN(B_RESULT_1 DOWNTO E_RESULT_1) <=  std_logic_vector(to_unsigned(15, LEN_RESULT)); -- Endereço da MAU
s_DIN(B_RESULT_2 DOWNTO E_RESULT_2) <= std_logic_vector(to_unsigned(5, LEN_RESULT));   -- Posição para inserção
s_vDI(P_NORTH) <= '1';
WAIT FOR period;

s_DIN(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
s_DIN(B_PROCESS_NUM DOWNTO E_PROCESS_NUM) <= std_logic_vector(to_signed(1, LEN_PROCESS_NUM)); -- ID Programa
s_DIN(B_PACKET_NUM DOWNTO E_PACKET_NUM) <= std_logic_vector(to_signed(1, LEN_PACKET_NUM)); -- ID Pacote
WAIT FOR period;

s_DIN(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
s_DIN(B_OPERAND DOWNTO E_OPERAND) <= std_logic_vector(to_signed(1, LEN_OPERAND)); -- Valor a ser inserido

-- "1000" & "00001111" & "00010001" & "00000001" & "000000" & "00"
ASSERT s_DO =   CTRL_HEADER
              & std_logic_vector(to_unsigned(15, LEN_RESULT-3)) -- Endereço da MAU
              & addr                                            -- Endereço da RPU
              & std_logic_vector(to_unsigned(1, LEN_N_INSTRUCTIONS)) -- Número de instruções é constante = 1
              & std_logic_vector(to_unsigned(0, LEN_REROUTE)) -- Reroute é a porta que está sendo atendida
              & PKT_CONTROL
   REPORT "Header failed"
      SEVERITY Error;

WAIT FOR period;
```

Fonte: Próprio autor.

O trecho do *script* da Figura 56 exibe o caso de teste Nº 22, em que há duas requisições e apenas uma é atendida. Observe que se a porta sul receber a confirmação, então a requisição é desativada.

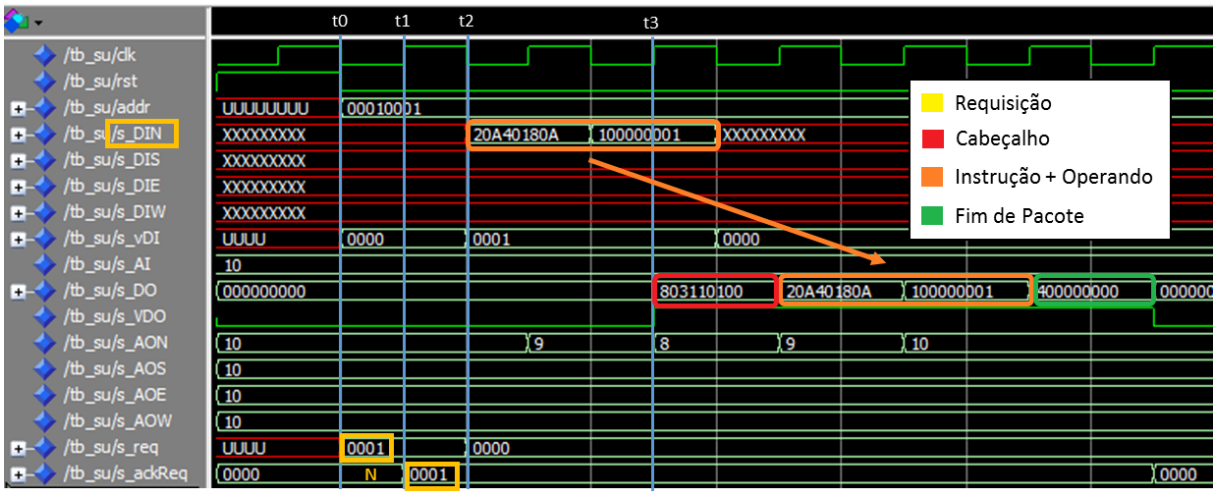
Figura 56 – Trecho de código do script de teste da SU para instrução SYNC com concorrência - Caso de teste Nº 22.

```
--------------------------
-- Caso de Teste:
-----------------
-- Porta North
-- Instrução SYNC
-- 2 Operandos
--------------------------

s_req(P_NORTH) <= '1';
s_req(P_SOUTH) <= '1';
WAIT FOR period;
s_req(P_NORTH) <= '0';
IF (s_ackReq(P_SOUTH) = '1') THEN
   s_req(P_SOUTH) <= '0';
END IF;
```

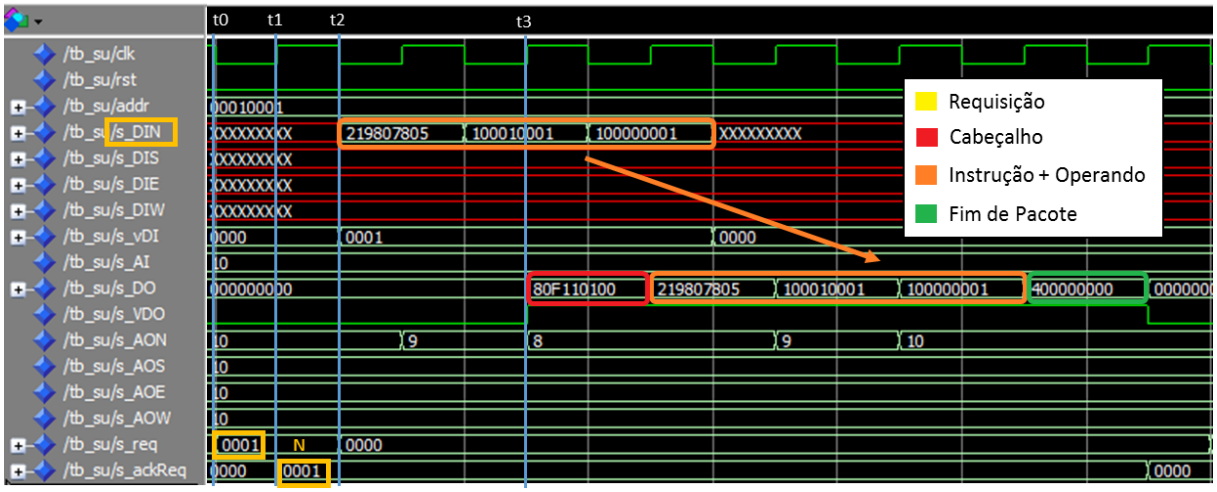
Fonte: Próprio autor.

As simulações das 6 instruções (Figura 57) realizam, basicamente, os seguintes passos: (1) requisição, (2) confirmação, (3) inserção de instrução e operando(s) no *buffer* da porta requisitante e (4) construção e envio do pacote de controle pelo *buffer* de saída. A diferença entre as simulações está apenas na palavra de cabeçalho, no tipo da instrução e na quantidade de operandos. Esses passos foram mapeados para os instantes t0, t1, t2 e t3, respectivamente, de cada figura.

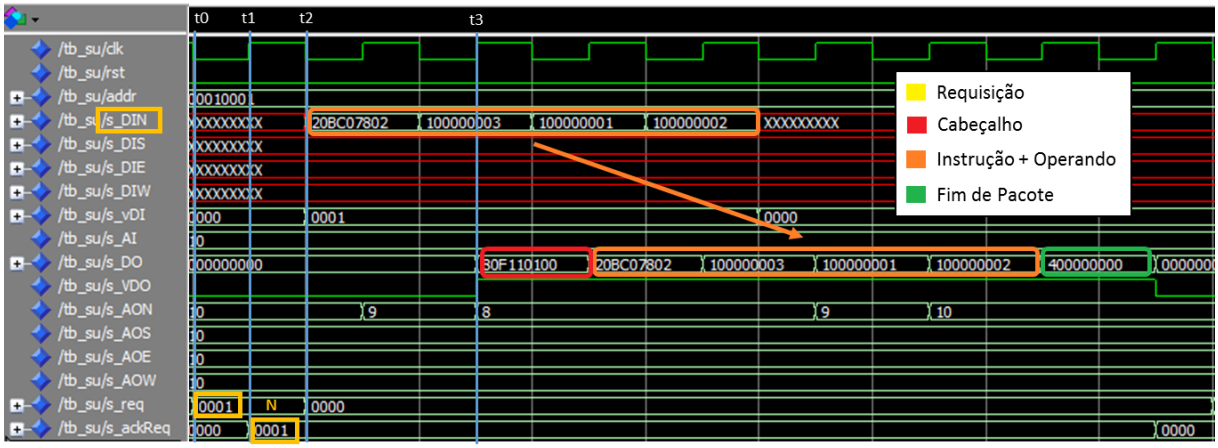
Figura 57 – Resultados das simulações da SU (Caso de teste Nº 21) para as instruções de controle: (a) LOAD; (b) SEND; (c) STORE; (d) EXEC; (e) SYNEXEC.



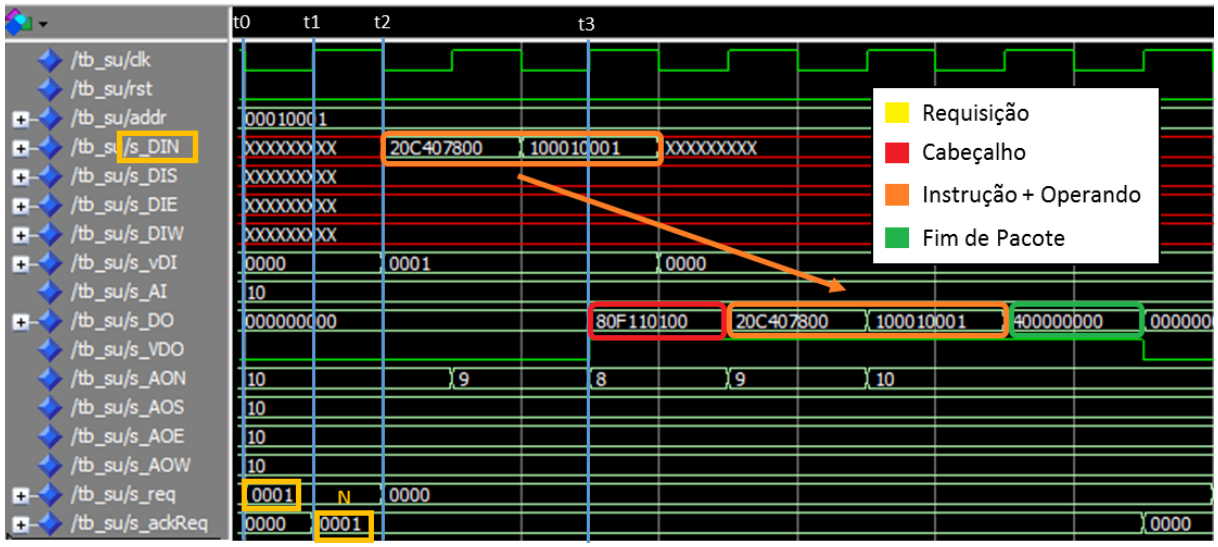
(a)


t0
t1
t2
t3
/tb_su/clk
/tb_su/rst
/tb_su/addr
/tb_su/s_DIN
/tb_su/s_DIS
/tb_su/s_DIE
/tb_su/s_DIW
/tb_su/s_vDI
/tb_su/s_AI
/tb_su/s_DO
/tb_su/s_VDO
/tb_su/s_AON
/tb_su/s_AOS
/tb_su/s_AOE
/tb_su/s_AOW
/tb_su/s_req
/tb_su/s_ackReq
Requisição
Cabeçalho
Instrução + Operando
Fim de Pacote

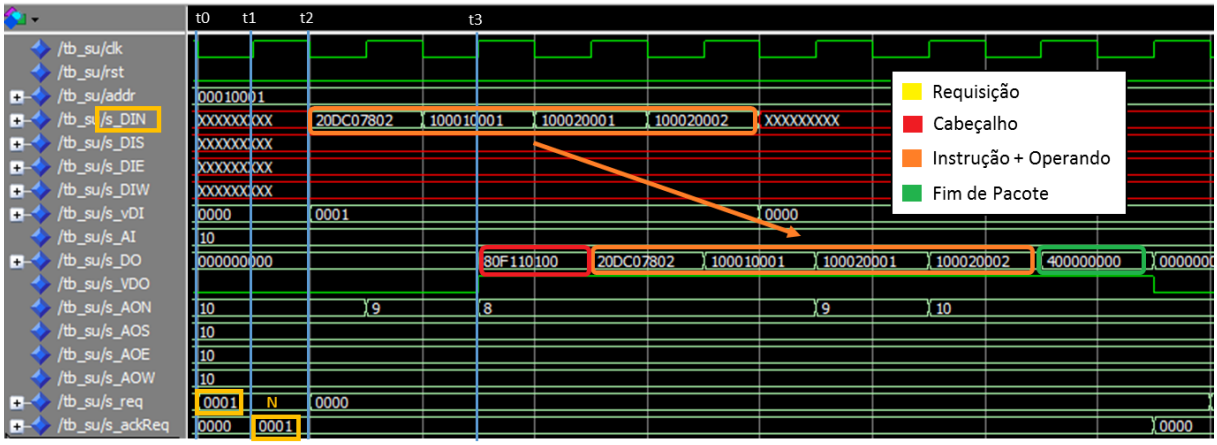
(b)


t0
t1
t2
t3
/tb_su/clk
/tb_su/rst
/tb_su/addr
/tb_su/s_DIN
/tb_su/s_DIS
/tb_su/s_DIE
/tb_su/s_DIW
/tb_su/s_vDI
/tb_su/s_AI
/tb_su/s_DO
/tb_su/s_VDO
/tb_su/s_AON
/tb_su/s_AOS
/tb_su/s_AOE
/tb_su/s_AOW
/tb_su/s_req
/tb_su/s_ackReq
Requisição
Cabeçalho
Instrução + Operando
Fim de Pacote

(c)


t0
t1
t2
t3
/tb_su/clk
/tb_su/rst
/tb_su/addr
/tb_su/s_DIN
/tb_su/s_DIS
/tb_su/s_DIE
/tb_su/s_DIW
/tb_su/s_vDI
/tb_su/s_AI
/tb_su/s_DO
/tb_su/s_VDO
/tb_su/s_AON
/tb_su/s_AOS
/tb_su/s_AOE
/tb_su/s_AOW
/tb_su/s_req
/tb_su/s_ackReq
Requisição
Cabeçalho
Instrução + Operando
Fim de Pacote

(d)

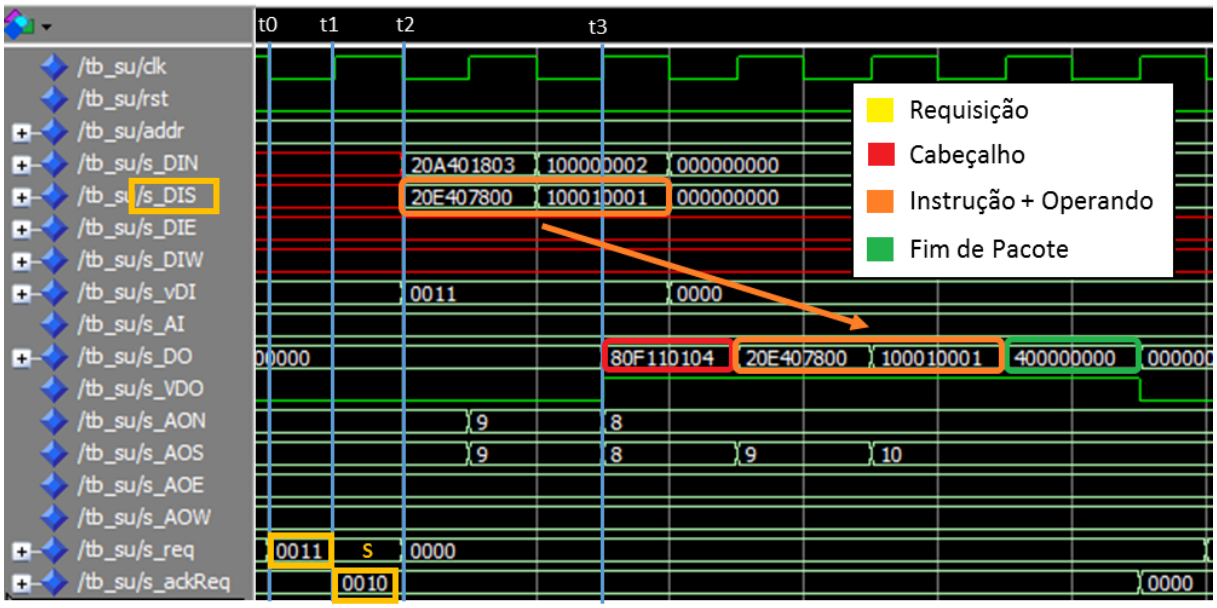


(e)

Fonte: Próprio autor.

A Figura 58 apresenta o caso de disputa pela SU em que é gerado um pacote de controle com a instrução SYNC. Observe, no instante t0, que as portas norte e sul estão requisitando, mas apenas a porta sul recebe a confirmação, no instante t1. No instante t2 e t3 é possível perceber que somente os dados inseridos na porta sul são enviados pela saída.

Figura 58 – Resultado da simulação da SU para instrução SYNC com concorrência entre as portas Norte e Sul - Caso de teste Nº 21 e 22.



Fonte: Próprio autor.

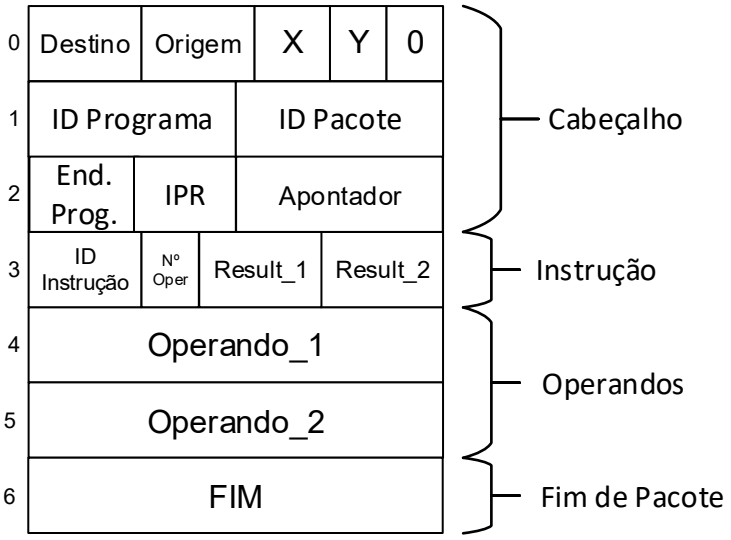
5.1.1.7 RPU (Árbitro)

Para testar o Árbitro, seria necessário conectar este com os demais componentes abordados e manipular os sinais manualmente. Além disso, essa interligação representa

exatamente as funcionalidades e o funcionamento de uma RPU. Por esse motivo, os testes que seriam atribuídos ao Árbitro, serão executados por uma RPU, permitindo, assim, testar automaticamente e conjuntamente o Árbitro e RPU.

O pacote simbólico da Figura 59 apresenta um modelo para os pacotes usados nas simulações mostradas a seguir. Este pacote é composto por três palavras de cabeçalho, uma ou mais instruções, para cada instrução um ou dois operandos e uma palavra de fim de pacote. No cabeçalho, dá-se ênfase para os campos de destino, origem, IPR e apontador.

Figura 59 – Pacote regular simbólico usado como modelo para a carga injetada na RPU.

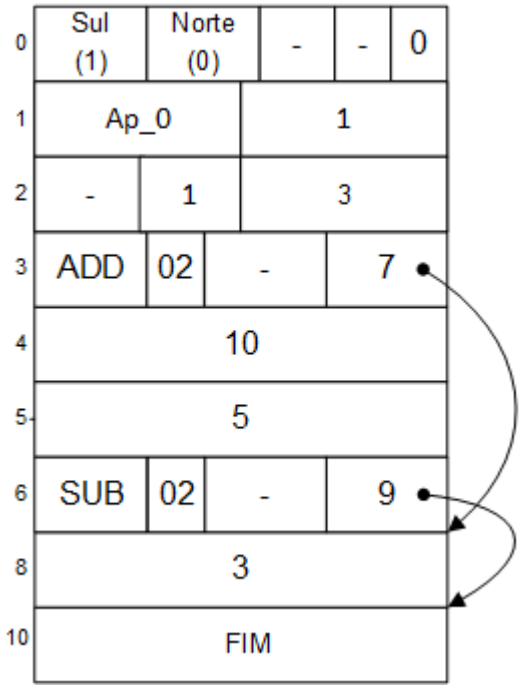


Fonte: Próprio autor.

As simulações da RPU são composta pelos seguintes sinais: clock (clk), reset (rst), endereço da RPU atual (addr), 4 entradas (s_DIN) e 4 saídas (s_DOUT) de dados, bem como, *flags* para indicar se o dado é válido na entrada (s_vDI) e na saída (s_vDO) e para a disponibilidade do buffer de entrada (s_aIn) e do buffer da RPU destino (s_aOut). Os sinais s_vDI, s_vDO, s_aOut e s_aIn estão dispostos na simulação como um *array* unidimensional e, os sinais s_DIN e s_DOUT, como *array* bidimensional, seguindo a mesma leitura e interpretação dos índices já comentada.

O primeiro caso de teste da RPU, Nº 23, diz respeito à execução de um pacote regular. Conforme ilustrado na Figura 60, o pacote é injetado na porta norte, deve executar uma soma entre os valores 10 e 5 e transmitir o restante do pacote pela porta sul, em que deve injetar o resultado da computação na posição 7.

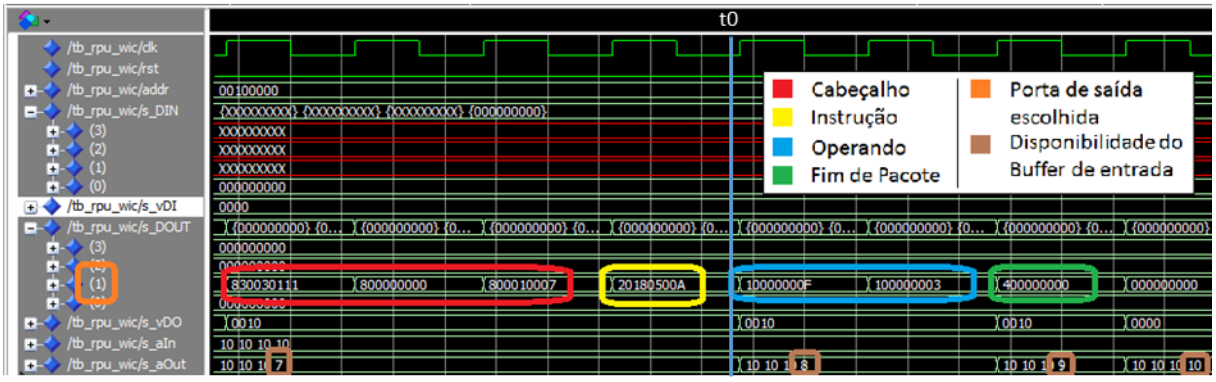
Figura 60 – Modelo de carga injetada na RPU para execução em fluxo único de um pacote regular - Caso de teste Nº 23.



Fonte: Próprio autor.

O resultado desta simulação é exibido na Figura 61, em que destacamos a transmissão do pacote pelo sinal s_DOUT(1). Além disso, vale salientar que o sinal s_aOut apresenta a disponibilidade do *buffer* da porta de origem (norte) e, no instante t0, o resultado da soma é inserido no pacote. Esta disponibilidade aumenta na medida que o pacote é transmitido.

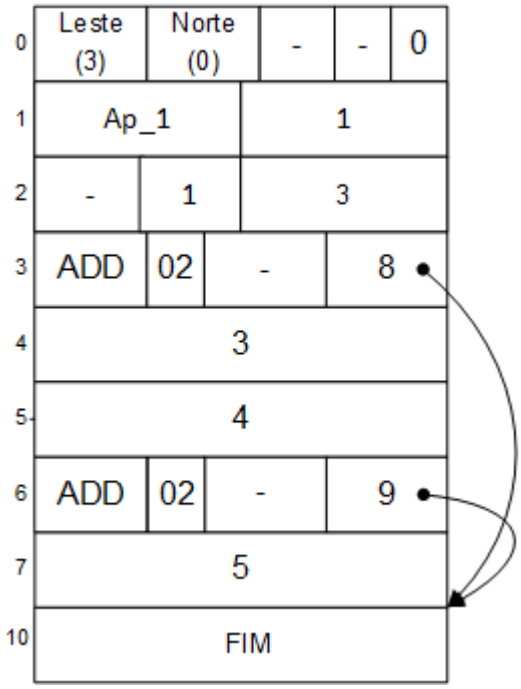
Figura 61 – Resultado da simulação da RPU para pacote regular - Caso de teste Nº 23.



Fonte: Próprio autor.

A Figura 62 exibe o pacote injetado na RPU para o segundo caso de teste. Neste caso, o pacote é injetado pela porta norte, que deve executar a soma de 3 e 4 e transmitir o restante do pacote pela porta leste, inserindo o resultado da computação na posição 8.

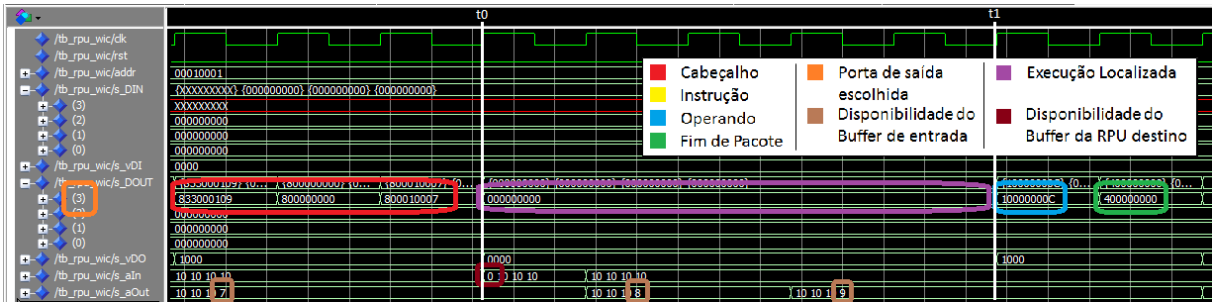
Figura 62 – Modelo de carga injetada na RPU para execução em fluxo único, de um pacote regular, com Execução Localizada - Caso de teste Nº 24.



Fonte: Neto e Fernandes (2016).

Entretanto, conforme mostrado na simulação da Figura 63, no instante t0, é verificado que o *buffer* destino não possui espaço suficiente (s_aIn[3] = 0), logo, a execução localizada é ativada e a segunda instrução executada. No *clock* seguinte, o *buffer* destino já possui espaço disponível. Porém, somente no instante t1, é possível verificar a retomada da transmissão com o resultado da segunda soma, ou seja, (3 + 4) + 5 = 12_{dec} = C_{hex}. Observe que durante o intervalo t0 e t1 o dado de saída não é válido (s_vDO[3] = 0).

Figura 63 – Resultado da simulação da RPU para fluxo único com Execução Localizada - Caso de teste Nº 24.

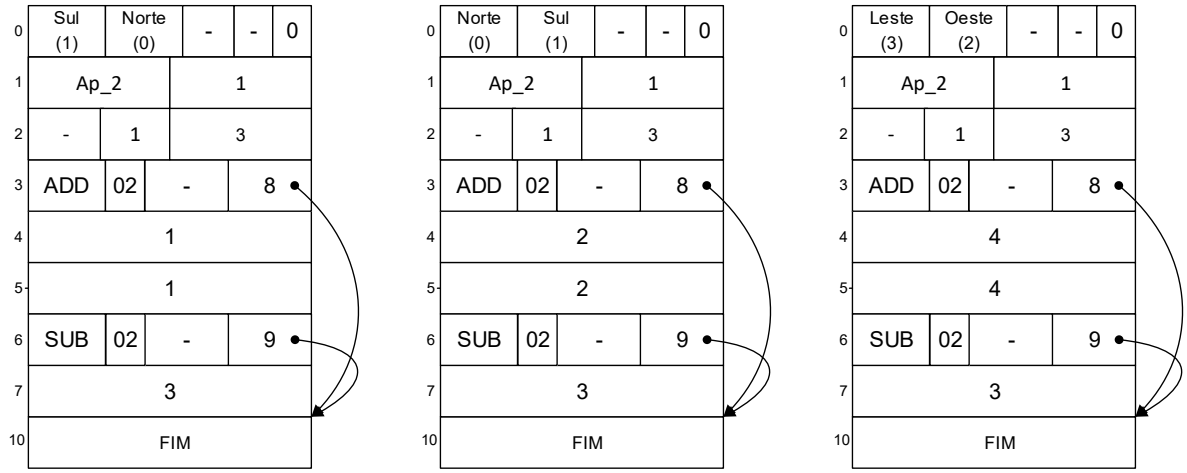


Fonte: Neto e Fernandes (2016).

As próximas simulações tratam do processamento paralelo de pacotes para dois casos: **sem** e **com** disputa do canal de transmissão. A Figura 64 contém os pacotes inseridos simultaneamente nas portas norte (0), sul (1) e oeste (2) da RPU e são roteados,

respectivamente, para diferentes portas (sem disputa): sul (1), norte (0) e leste (3). Eles diferem, apenas, nos campos de origem, destino e, para identificá-los, nos operandos.

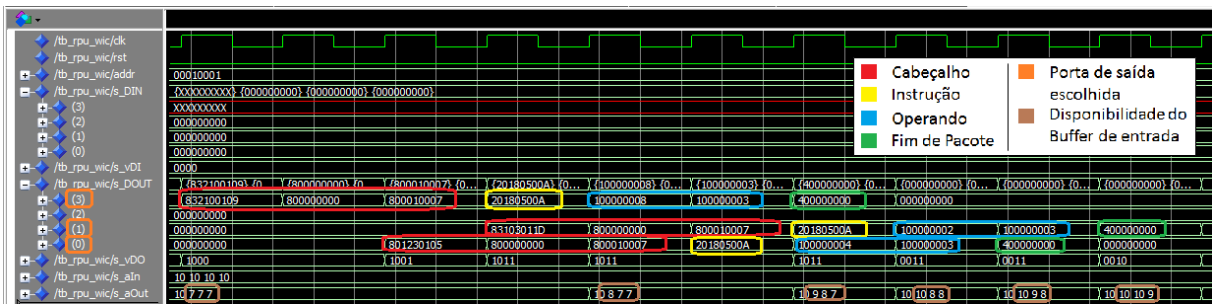
Figura 64 – Modelo de cargas injetada na RPU para execução paralela, de três pacotes regulares, **sem** disputa do canal de transmissão - Caso de teste Nº 25.



Fonte: Próprio autor.

Como resultado, a simulação da Figura 65 mostra o efeito da concorrência na utilização da ALU ao executar uma instrução de cada pacote. Como os pacotes chegam à ALU ao mesmo tempo e ela só atende um pacote por vez, independentemente se houver requisições simultâneas, há um atraso na transmissão dos pacotes que são atendidos posteriormente, uma vez que precisam aguardar sua vez para obter o resultado da computação e continuar a transmissão. Note que na medida que os pacotes vão sendo transmitidos, a disponibilidade do *buffer* de entrada da porta de origem, do pacote, vai aumentando.

Figura 65 – Resultado da simulação da RPU para execução paralela sem disputa - Caso de teste Nº 25.

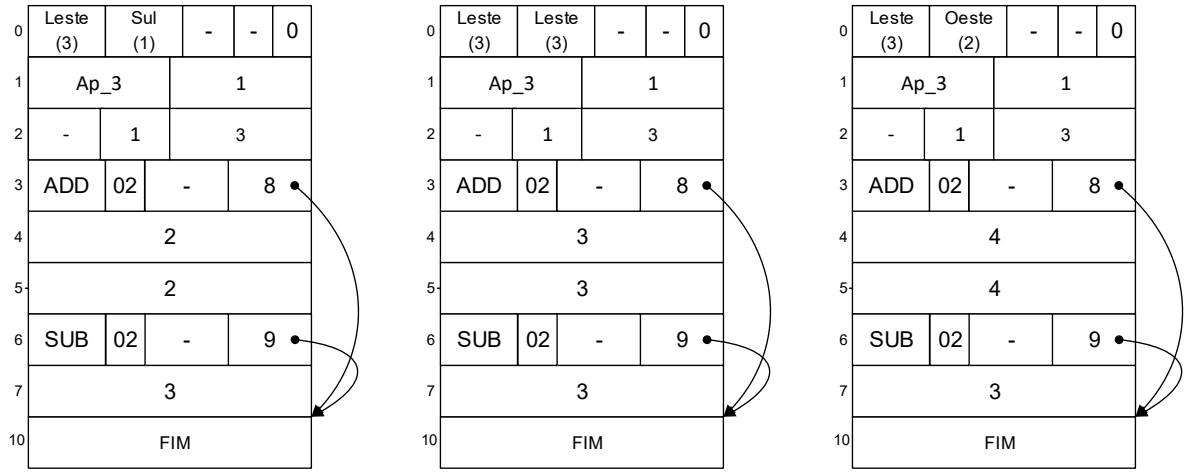


Fonte: Neto e Fernandes (2016).

O último caso de teste da refere-se à inserção simultânea de três pacotes em portas diferentes e o roteamento para o mesmo destino, neste caso, para a porta leste, conforme

exibido na Figura 66. Como na simulação anterior, as únicas diferenças entre os pacotes são os campos de origem e dos operandos, para identificá-los durante a simulação.

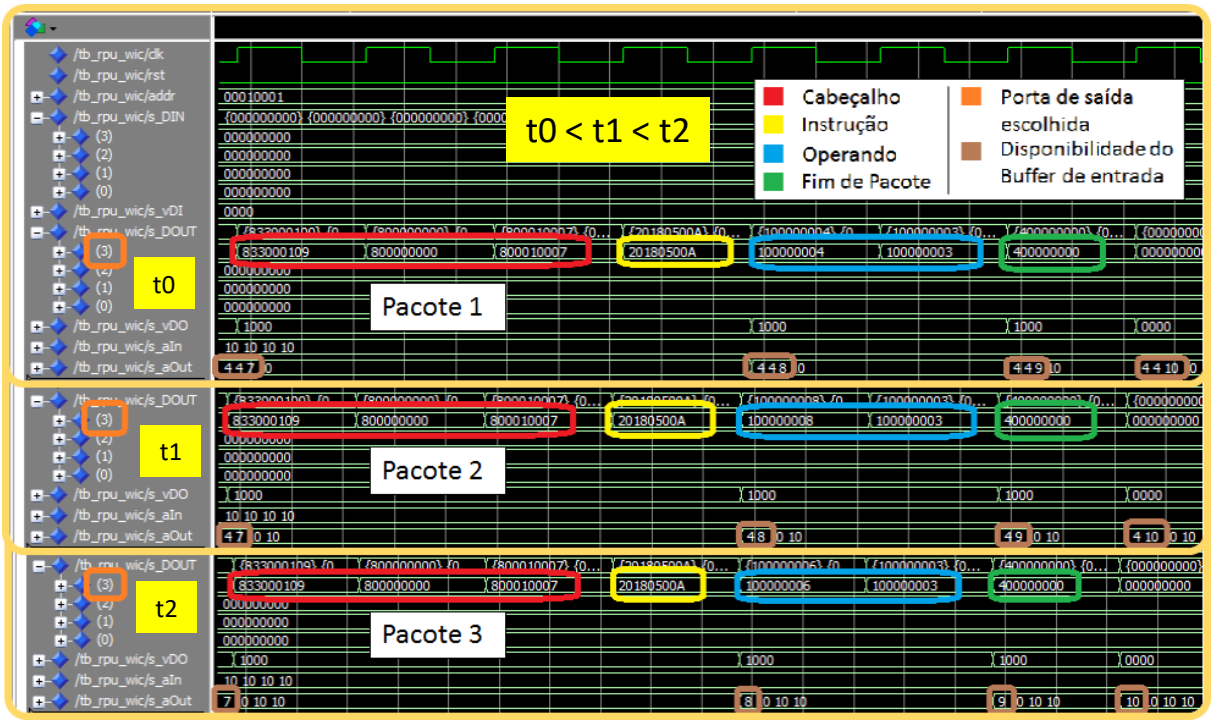
Figura 66 – Modelo de carga injetada na RPU para execução paralela, de três pacotes regulares, **com** disputa do canal de transmissão - Caso de teste Nº 26.



Fonte: Próprio autor.

O resultado da computação paralela com disputa é exibido na Figura 67 que mostra que os pacotes são roteados, executados e transmitidos **completamente**, apenas um por vez, até que não exista mais requisições para a mesma porta. Os três pacotes transmitidos **sequencialmente**, em tempos diferentes, foram agrupados para uma melhor visualização. Como forma de comprovar a transmissão serial, o espaço disponível de apenas um buffer de entrada é aumentado por vez.

Figura 67 – Resultado da simulação da RPU para execução paralela com disputa - Caso de teste Nº 26.



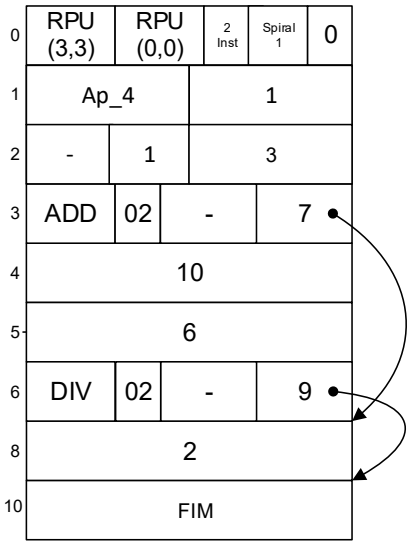
Fonte: Neto e Fernandes (2016).

5.1.1.8 Aplicação real

Os casos de teste apresentados nesta subseção abordam a execução completa de um pacote. Duas aplicações foram propostas para demonstrar o correto processamento das instruções de um pacote através de seu percurso pela rede. Para melhorar a exibição dos resultados, um procedimento foi criado para imprimir na tela do terminal a quantidade de ciclos de clock transcorridos e os dados transmitidos entre as RPUs.

A primeira aplicação trata-se de uma média aritmética, cujo pacote injetado é mostrado na Figura 68. Este pacote tem como origem a RPU(0,0) e destino a RPU(3,3), sendo realizado, neste percurso, uma soma entre 10 e 6 e tem o resultado inserido como operando na próxima instrução DIV, que efetua uma divisão por 2.

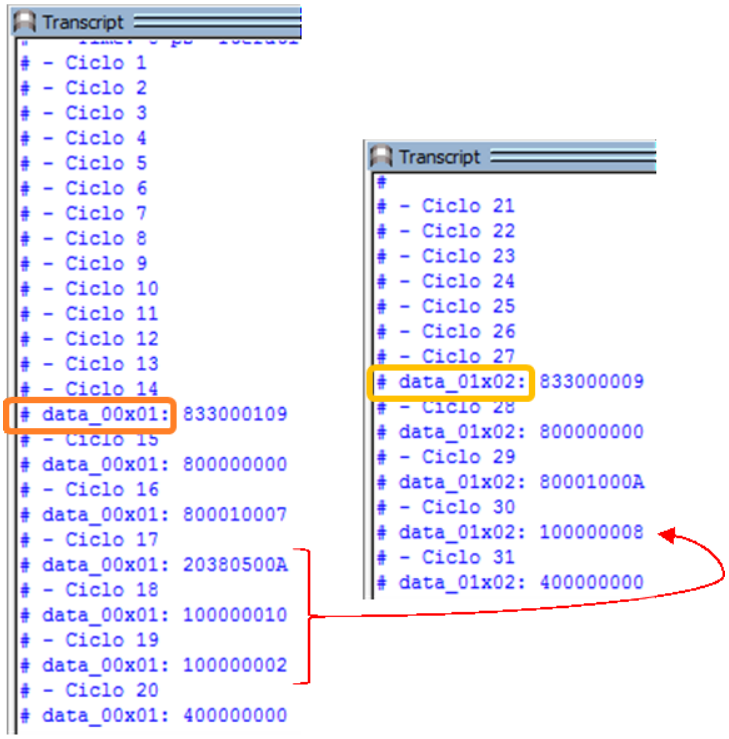
Figura 68 – Modelo de carga injetada na malha 2D 4x4 de RPUs para a aplicação Média Aritmética - Caso de teste Nº 27.



Fonte: Próprio autor.

A Figura 69 apresenta o resultado da execução da média aritmética na rede. O pacote é processado na RPU(0,0) e, no ciclo 14, começa a transmissão para a RPU(0,1). Perceba que no ciclo 18 é transmitido o resultado da soma. Na RPU(0,1), a divisão é feita e o resultado final ((10+6)/2 = 8) é transmitido no ciclo de número 30.

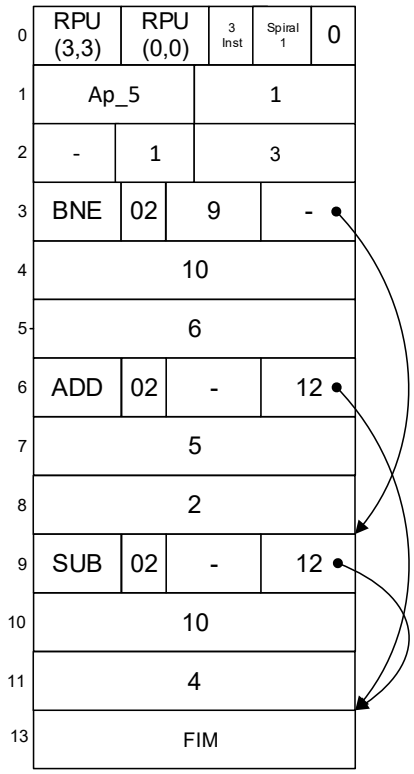
Figura 69 – Resultado da simulação para a aplicação Média Aritmética - Caso de teste Nº 27.



Fonte: Próprio autor.

A segunda aplicação diz respeito à operação de desvio condicional. O pacote modelo (Figura 70) possui a mesma origem e destino da aplicação anterior, mas executa a instrução *Branch Not Equal* para os valores 10 e 6, ou seja, ambos valores serão comparados e constatados que não são iguais. Assim, a posição 9 é definida para salto (RESULT_1). Logo, as palavras 6, 7 e 8 serão descartadas e a instrução seguinte (subtração) será executada.

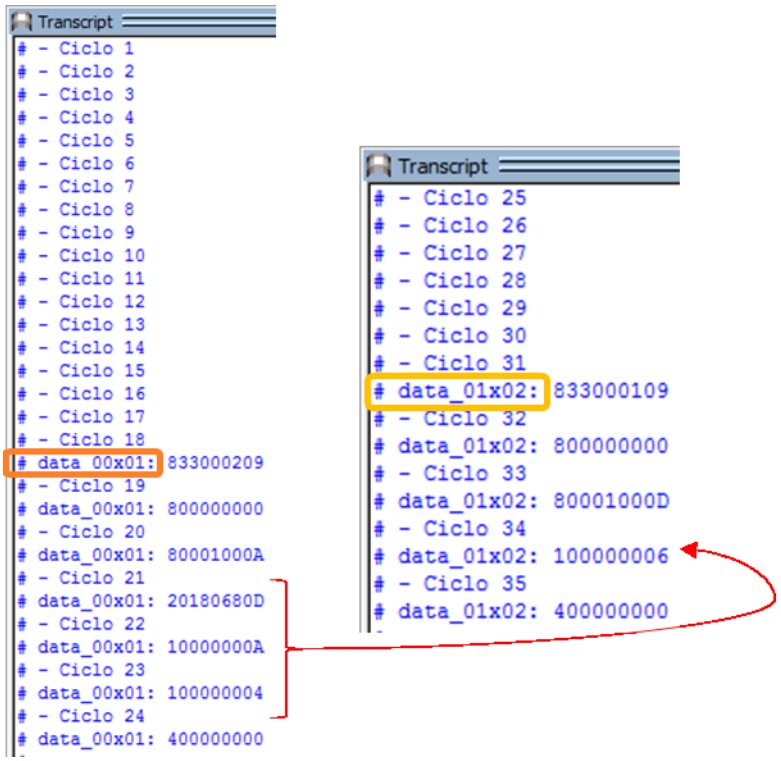
Figura 70 – Modelo de carga injetada na malha 2D 4x4 de RPUs para a aplicação Desvio condicional - Caso de teste Nº 27.



Fonte: Próprio autor.

A Figura 71 apresenta o resultado da execução de uma instrução de desvio ao percorrer a malha 2D. Na RPU(0,0), é verificado que os operandos do BNE são diferentes e, por isso, um desvio precisa ser feito. Note que somente no ciclo Nº 18 é que a primeira palavra de cabeçalho é transmitida, ou seja, diferentemente da aplicação anterior que começou a transmissão no ciclo 14, a instrução *branch* necessitou de mais ciclos para poder descartar as palavras que estavam antes da posição para desvio (soma entre 5 e 2). Após o desvio realizado, a próxima instrução de subtração entre 10 e 4 é computada na RPU(0,1) e transmitida para a RPU(0,2). O resultado final é transmitido somente no ciclo Nº 34.

Figura 71 – Resultado da simulação para a aplicação de Desvio Condicional - Caso de teste Nº 27.



Fonte: Próprio autor.

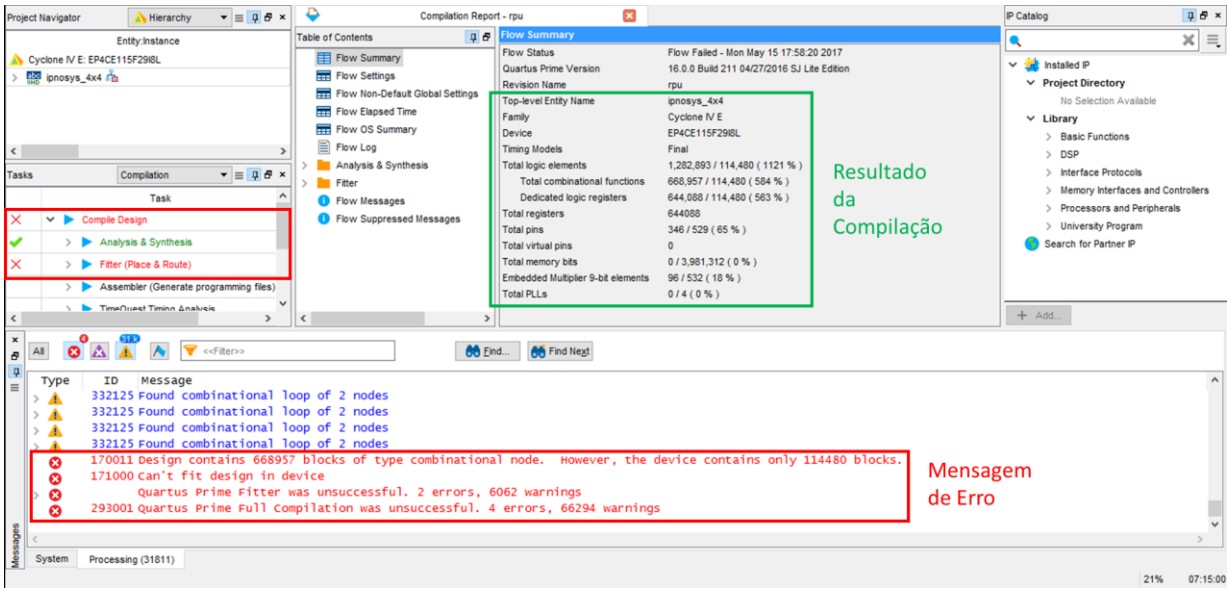
5.2 Resultado de síntese

Segundo Altera (2014), para realizar a compilação de uma descrição em VHDL e computar os resultados desejados para a síntese é necessário percorrer um fluxo de compilação definido pelas seguintes etapas: (1) análise e síntese, (2) montador (*Fitter*), (3) análise de tempo (*Timing Analysis*) e (4) análise de potência (*Power Analysis*). Na primeira etapa é verificado a sintaxe do código VHDL e criado um banco de dados otimizado do projeto, que será utilizado na etapa seguinte. A etapa de *fitter* realiza o mapeamento da lógica e tempo requeridos no projeto para os recursos disponíveis do dispositivo-alvo. A análise de tempo e potência dependem do mapeamento feito na etapa anterior.

A primeira compilação da malha 2D 4x4 de RPUs utilizou o mesmo dispositivo que as versões simplificada e paralela da RPU, isto é, o dispositivo EP4CE115F29I8L da família Cyclone IV E, da fabricante Altera. Entretanto, a compilação não foi realizada totalmente com sucesso em virtude do dispositivo alvo não ter recursos (elementos lógicos) suficientes para a arquitetura desejada, conforme pode ser visto no resultado da síntese na Figura 72.

Contudo, é possível observar que a etapa de análise e síntese foi realizada com sucesso, inclusive, informando que a quantidade de elementos lógicos é superior em 1121% em relação ao disponibilizado pelo dispositivo.

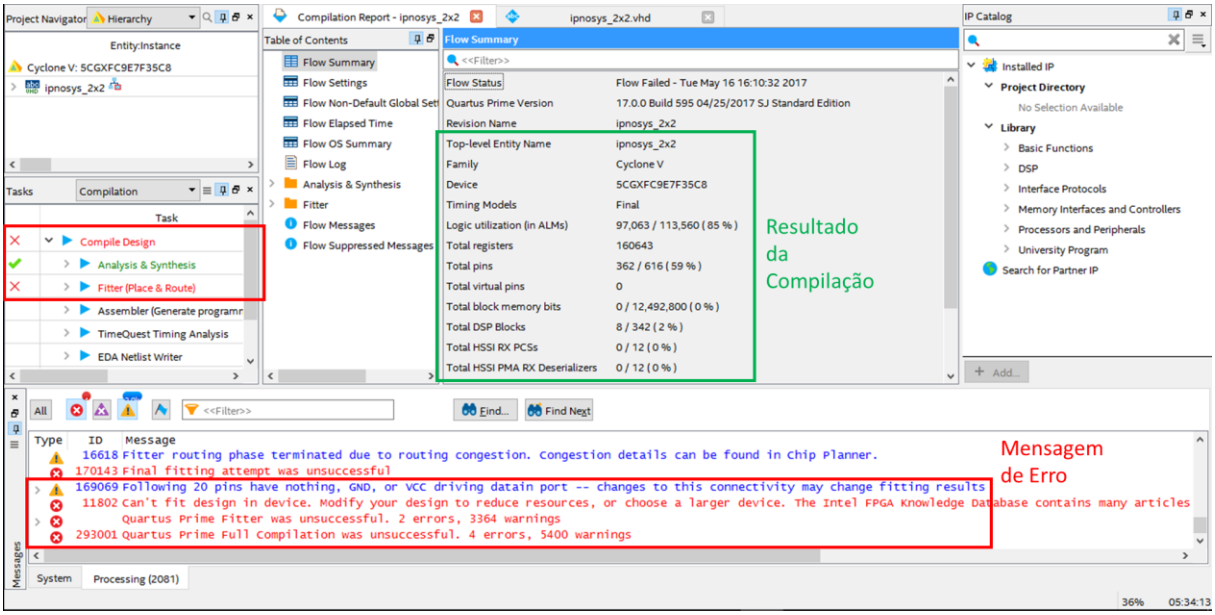
Figura 72 – Resultado da compilação para uma malha 2D 4x4 de RPUs.



Fonte: Próprio autor.

Para contornar essa restrição, é sugerido a escolha de um dispositivo de maior capacidade de recursos ou modificação no projeto (visando a diminuição da quantidade de componentes ou a otimização do código). Por isso, uma malha 2D 2x2 de RPUs foi elaborada e sintetizada no dispositivo 5CGXFC9E7F35C8 da família Cyclone V, também da Altera. Porém, novamente, conforme exibido na Figura 73, apesar do projeto conter menos componentes que a versão 4x4, a implementação da versão 2x2 não cabe no dispositivo.

Figura 73 – Resultado da compilação para uma malha 2D 2x2 de RPUs.



Fonte: Próprio autor.

Por estes motivos, os dados para frequência de operação e potência das duas versões de malha 2D de RPUs não foram coletados.

Sendo assim, a Tabela 4 apresenta os resultados obtidos para área em chip, frequência máxima de operação e potência, gerados a partir da compilação da descrição em VHDL do hardware da RPU simplificada (OLIVEIRA NETO; FERNANDES, 2015), paralela (NETO; FERNANDES, 2016) e da malha 2D 2x2 e 4x4 de RPUs, discutida neste trabalho.

Tabela 4 – Resultado obtido com a descrição do hardware das malhas 2D de RPUs.

Versão	**Área em Chip (LE)**	**Frequência Máxima de Operação (MHz)**	**Potência (mW)**
RPU simplificada	4,182	91.02	154.41
RPU paralela	24,062	6.22	236.84
RPUs em Malha 2D 2x2	97,063	-	-
RPUs em Malha 2D 4x4	1,282,893	-	-

Fonte: Próprio autor.

Como pode-se observar, as duas versões da malha 2D de RPUs foram compiladas apenas na etapa de análise e síntese para obter os resultados possíveis. Logo, em relação à área em chip, houve um aumento considerável de aproximadamente 303% para a matriz 2x2 e de 7501% para a matrix 4x4, em relação à RPU paralela.

6 CONSIDERAÇÕES FINAIS

O desenvolvimento de um processador exige a definição de uma metodologia a ser seguida durante toda etapa do projeto para que erros não sejam incluídos ou transmitidos para as etapas seguintes. Arquiteturas não-convencionais (MPSoCs baseado em NoC) demandam maior cautela no projeto, implementação e teste para que todas as funcionalidades sejam atendidas corretamente.

Sendo assim, este trabalho apresentou uma metodologia que integra elementos ágeis da Engenharia de Software para trazer produtividade e qualidade no sistema construído. Este elemento refere-se ao estilo de desenvolvimento orientado a testes, que prioriza a criação de casos de teste automatizados (*scripts*) antes da implementação de um componente. Adicionalmente, foi apresentado o projeto (fluxograma, caminho de dados e máquina de estados), a implementação (trechos de código em VHDL) e os testes para várias situações possíveis (trechos de *scripts* e simulações) de cada componente interno da RPU e de uma malha 2D 4x4 de RPUs, permitindo, portanto, a execução de aplicações reais e o processamento paralelo completo de pacotes injetados na rede.

Tendo em vista os aspectos observados, percebe-se que a implementação realizada com base nos desenhos de projeto, na estratégia de teste incremental e nos casos de teste permitiram uma codificação mais rápida, modular e com menos chances de erros. Além disso, os *scripts* permitiram adicionar funcionalidades, bem como, aperfeiçoar e otimizar as funcionalidades existentes, sempre garantindo, ao executar os testes, que nenhum componente foi prejudicado com alguma mudança no código.

Por fim, na compilação da arquitetura desenvolvida, foram abordados os resultados parciais de síntese para uma malha 2D 4x4 e 2x2 de RPUs, e analisados perante duas versões (que realiza processamento sequencial e paralelo) da RPU. Estes resultados ficaram limitados à área em chip, uma vez que necessitavam de um dispositivo com capacidade maior de recursos ou da otimização do código em VHDL. De todo modo, esta análise comparativa pôde dimensionar a evolução obtida com o desenvolvimento incremental da RPU e de uma malha 2D.

6.1 Trabalhos futuros

Como trabalhos futuros será feito uma adaptação na RPU para tratar os pacotes do tipo interrompido e *caller*, bem como, a adição de um canal virtual para o caso particular dos

pacotes de controle (que são criados na SU a partir de uma instrução de controle e seus respectivos operando(s) para serem executadas na MAU). Com isso, deve-se verificar a correta comunicação entre Árbitro, SU, *Buffer Local* e *Crossbar*, garantindo que o pacote seja roteado pela porta exata e entregue ao destino esperado. Em caso de ser uma instrução do tipo LOAD, deve-se permitir a recepção de pacotes contendo a instrução de RELOAD.

Além disso, uma malha 2D possibilita a realização de simulações com pacotes reais como, por exemplo, laços de repetição e múltiplos pacotes simultaneamente.

Adicionalmente, o código-fonte em VHDL será otimizado para melhorar a frequência máxima de operação e para permitir a completa compilação de uma malha 2D 4x4 de RPUs, a fim de obter resultados para síntese.

Finalmente, será realizado o projeto, implementação e teste das outras unidades de IPNoSys, isto é: memórias, MAUs, IOMAU, IONode, IODevice e MemArbiter. Como consequência desses trabalhos, teremos a arquitetura completa de IPNoSys sintetizável em FPGA.

REFERÊNCIAS

ACCELLERA. Systems Initiative. SystemC. 2005. Disponível em: <http://www.accellera.org/downloads/standards/systemc >. Acesso em: 08 mai. 2017.

ALI, M.; WELZL, M.; ZWICKNAGL, M. Networks on Chips: Scalable interconnects for future systems on chips. In: **Circuits and Systems for Communications**, 2008. ECCSC 2008. 4th European Conference on, 2008. p.240-245.

ALTERA. Quartus II Help v14.1 - About Place & Route. 2014. Disponível em: <http://quartushelp.altera.com/14.1/mergedProjects/comp/comp/comp_about_place.htm>. Acesso em: 11 mai. 2017.

ALTERA. Official Site. Disponível em: < https://www.altera.com/>. Acesso em: 09 mai. 2017.

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

ARPACI-DUSSEAU, R.; ARPACI-DUSSEAU, A. **Operating Systems**: Three Easy Pieces. [s.n.], 2012. ISBN 9781105979125. Disponível em: <https://books.google.com.br/books?id=orxwMwEACAAJ>.

BECK, K. **Test-driven Development:** By Example. Addison-Wesley, 2003. (Kent Beck signature book). ISBN 9780321146533. Disponível em: <https://books.google.com.br/books?id=CUlsAQAAQBAJ>.

BENINI, L.; MICHELI, G. D. Networks on Chips: A New SoC Paradigm. **IEEE Computer Society Press.** v. 35, p. 70-78, 2002.

BORGES, E. N. **Conceitos e benefícios do test driven development.** In: Universidade Federal do Rio Grande do Sul / Instituto de Informática. [s.n.], 2006. Disponível em: <http://www.inf.ufrgs.br/~cesantin/TDD-Eduardo.pdf>.

COSTA, R. V. et al. SICXE: Improving experience with didactic processors. In: **Studant Forum** (SForum'2012). João Pessoa, 2012.

COUTO, Juliene V. **Geração de código otimizado visando a exploração de paralelismo na arquitetura IPNoSys**. Dissertação de Mestrado. Departamento de Ciências Exatas e Naturais – Universidade Federal Rural do Semi-árido, Mossoró, 2016.

DAMASCENO, A.; FERNANDES, S.; GIRÃO, G. B. (2016). Proposta de uma Hierarquia de Memória para a Arquitetura IPNoSys.

FERNANDES, S. R. et al. Processing while routing: a network-on-chip-based parallel system. **IET Computers & Digital Techniques**, v. 3, n. 5, p. 525–538, 2009.

FERNANDES, S. R.; SILVA, I. S; KREUTZ, M. Packet-driven General Purpose Instruction Execution on Communication-based Architectures. **Journal of Integrated Circuits and Systems (JICS)**, v. 5, p. 53–66. 2010.

FLYNN, M. J. Some Computer Organizations and Their Effectiveness. **IEEE Transactions on Computers**, p.948-960, 1972.

GADELHA, M.; CORREA, E.; KREUTZ, M. (2011). A tool for developing IPNoSys programs using a high-level programming language. In: Workshop on Circuits and System Design. João Pessoa: WCAS, 2011.

HWANG, K. **Advanced Computer Architecture:** Parallelism, Scalability, Programmability. McGraw-Hill Higher Education, 1992. 771 ISBN 0070316228.

KRILL, B. VHDL Round-Robin Arbiter. [S.l.], 2016. Disponível em: <https://www.krll.de/portfolio/vhdl-round-robin-arbiter/ >. Acesso em: 01 mai. 2017.

LEÃO, G. B.; CARDOSO, F. C. Solução ao Processamento Maciço de Dados e Limite da Freqüência de Clock com o Uso de Aglomerado de Computadores por Meio da Hibridização de um Cluster de Alto Desempenho com um de Balanceamento de Carga. In: VI Encontro de Estudantes de Informática do Estado do Tocantins, 2004. Palmas/TO: ENCOINFO, 2004. Disponível em: <http://arquivo.ulbra-to.br/ensino/43020/artigos/anais2004/anais/gabrielaLeaoClusterEncoinfo2004.pdf>. Acesso em: 24 mai. 2017.

NETO, J.; FERNANDES, S. Metodologia de projeto para descrição da arquitetura ipnosys em vhdl. In: **EPOCA** 2013. Mossoró/RN: [s.n.], 2013.

NETO, J.; FERNANDES, S. Implementação em vhdl da rpu paralela de ipnosys. In: WSCAD-WIC 2016. [s.n.], 2016. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wscad-wic/2016/012.pdf >.

NUNES, D.; FERNANDES, S. (2016). Melhorando o Desempenho da IPNoSys com Software Pipelining.

OLIVEIRA, L.; FERNANDES, S. IPNoSys IDE Ambiente de Desenvolvimento e Simulação Integrado para uma Arquitetura não Convencional. International. **Journal of Computer Architecture Education (IJCAE)**, v. 4, n. 1, p. 21–24, 2015.

OLIVEIRA NETO, D.; FERNANDES, S. (2015). Design and Implementation in VHDL of RPU components of IPNoSys architecture In: 15th Microelectronics Students Forum. Salvador: Proceedings of SForum, 2015.

OLIVEIRA, S. Arquiteturas Paralelas. Disponível em: <http://www-usr.inf.ufsm.br/~sandro/elc139/tarefa1.php>. Acesso em: 12 mai. 2017.

PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization and Design:** the Hardware/Software Interface. 3rd. San Francisco: Elsevier, 2005. 684

PEREIRA, A. L. V.; FERNANDES, S. R. (2013). MPSoC with IPNoSys as Processing Element. In: **3th Workshop on Circuits and Systems Design**. Thessaloniki: WCAS, 2013.

PRESSMAN, R.; MAXIM, B. **Engenharia de Software** – 8ª Edição. [s.n.], 2016. ISBN 9788580555349. Disponível em:<https://books.google.com.br/books?id=wexzCwAAQBAJ>.

REGO, R. S. D. L. S. **Projeto e Implementação de uma Plataforma MP-SoC usando SystemC.** 2006. 144p. Dissertação de Mestrado (Mestrado). Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal, 2006.

SOARES, T. R. B. S.; SILVA, I. S.; FERNANDES, S. R. IPNoSys II: A New Architecture for IPNoSys Programming Model. In: Proceedings of the 28th Symposium on Integrated Circuits and Systems Design. Salvador: SBCCI, 2015. Disponível em: <http://doi.acm.org/10.1145/2800986.2801012> Acesso em: 28 abr. 2017.

APÊNDICE A – CÓDIGO-FONTE EM VHDL

- Malha 2D 4x4 de RPUs

```
LIBRARY ieee, work, STD;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE work.ipnosys_package.all;
USE STD.textio.all;
USE IEEE.std_logic_textio.all;
----------------------------
ENTITY ipnosys_4x4 IS
----------------------------
  PORT(
    -- System signals
    clk : IN STD_LOGIC; -- clock
    rst : IN STD_LOGIC; -- reset

    --RPU 0x0
    dataIn_west_00         : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataIn_00         : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    availableIn_west_00    : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneIn_west_00         : IN INTEGER RANGE N_LANES DOWNTO 0;

    dataOut_west_00        : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    availableOut_west_00   : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneOut_west_00        : OUT INTEGER RANGE N_LANES DOWNTO 0; -- Lanes for
virtual channel (lane 0: control packets; others: regular packets)

--=================================================================

    --RPU 0x3
    dataIn_east_03         : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataIn_03         : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    availableIn_east_03    : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneIn_east_03         : IN INTEGER RANGE N_LANES DOWNTO 0;

    dataOut_east_03        : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    availableOut_east_03   : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneOut_east_03        : OUT INTEGER RANGE N_LANES DOWNTO 0;

--=================================================================

    --RPU 3x0
    dataIn_west_30      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataIn_30         : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    availableIn_west_30  : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
```

```
    laneIn_west_30         : IN INTEGER RANGE N_LANES DOWNTO 0;

    dataOut_west_30     : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    availableOut_west_30 : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneOut_west_30     : OUT INTEGER RANGE N_LANES DOWNTO 0; -- Lanes for
virtual channel (lane 0: control packets; others: regular packets)

--================================================================

    --RPU 3x3
    dataIn_east_33         : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataIn_33         : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    availableIn_east_33    : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneIn_east_33         : IN INTEGER RANGE N_LANES DOWNTO 0;

    dataOut_east_33        : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    availableOut_east_33   : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneOut_east_33        : OUT INTEGER RANGE N_LANES DOWNTO 0);

END ipnosys_4x4;

---------------------------------------
ARCHITECTURE behaviour OF ipnosys_4x4 IS
---------------------------------------

-------------
-- SIGNALS
-------------
SIGNAL data_00x01     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_00x10     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_01x00     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_01x11     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_10x00     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_11x01     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_11x10     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_10x11     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_01x02     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_02x01     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_11x12     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_12x11     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_02x12     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_12x02     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_02x03     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_03x02     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_12x13     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_13x12     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_03x13     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_13x03     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_10x20     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
```

```
SIGNAL data_11x21     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_12x22     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_13x23     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_20x10     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_21x11     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_22x12     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_23x13     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_20x21     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_21x20     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_21x22     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_22x21     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_22x23     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_23x22     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_20x30     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_30x20     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_21x31     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_31x21     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_22x32     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_32x22     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_23x33     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_33x23     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_30x31     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_31x30     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_31x32     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_32x31     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_32x33     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL data_33x32     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

SIGNAL available_00x01     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_00x10     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_01x00     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_01x11     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_10x00     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_11x01     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_11x10     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_10x11     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_01x02     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_02x01     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_11x12     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_12x11     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_02x12     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_12x02     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_02x03     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_03x02     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_12x13     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_13x12     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_03x13     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_13x03     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_10x20     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
```

```
SIGNAL available_11x21     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_12x22     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_13x23     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_20x10     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_21x11     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_22x12     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_23x13     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_20x21     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_21x20     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_21x22     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_22x21     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_22x23     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_23x22     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_20x30     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_30x20     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_21x31     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_31x21     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_22x32     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_32x22     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_23x33     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_33x23     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_30x31     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_31x30     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_31x32     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_32x31     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_32x33     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
SIGNAL available_33x32     : INTEGER RANGE BUFFER_SIZE DOWNTO 0;

SIGNAL lane_00x01     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_00x10     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_01x00     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_01x11     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_10x00     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_11x01     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_11x10     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_10x11     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_01x02     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_02x01     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_11x12     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_12x11     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_02x12     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_12x02     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_02x03     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_03x02     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_12x13     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_13x12     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_03x13     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_13x03     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_10x20     : INTEGER RANGE N_LANES DOWNTO 0;
```

```
SIGNAL lane_11x21     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_12x22     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_13x23     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_20x10     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_21x11     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_22x12     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_23x13     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_20x21     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_21x20     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_21x22     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_22x21     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_22x23     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_23x22     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_20x30     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_30x20     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_21x31     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_31x21     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_22x32     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_32x22     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_23x33     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_33x23     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_30x31     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_31x30     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_31x32     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_32x31     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_32x33     : INTEGER RANGE N_LANES DOWNTO 0;
SIGNAL lane_33x32     : INTEGER RANGE N_LANES DOWNTO 0;


SIGNAL validData_00in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_01in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_02in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_03in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_10in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_11in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_12in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_13in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_20in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_21in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_22in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_23in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_30in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_31in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_32in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_33in: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";

SIGNAL validData_00out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_01out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_02out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
```

```
SIGNAL validData_03out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_10out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_11out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_12out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_13out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_20out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_21out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_22out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_23out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_30out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_31out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_32out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";
SIGNAL validData_33out: STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := "0000";

--\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
SIGNAL addressRPU0x0  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0000" &
"0000"; -- RPU's address (8 bits)
SIGNAL addressRPU0x1  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0000" &
"0001"; -- RPU's address (8 bits)
SIGNAL addressRPU0x2  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0000" &
"0010"; -- RPU's address (8 bits)
SIGNAL addressRPU0x3  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0000" &
"0011"; -- RPU's address (8 bits)
SIGNAL addressRPU1x0  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0001" &
"0000"; -- RPU's address (8 bits)
SIGNAL addressRPU1x1  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0001" &
"0001"; -- RPU's address (8 bits)
SIGNAL addressRPU1x2  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0001" &
"0010"; -- RPU's address (8 bits)
SIGNAL addressRPU1x3  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0001" &
"0011"; -- RPU's address (8 bits)
SIGNAL addressRPU2x0  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0010" &
"0000"; -- RPU's address (8 bits)
SIGNAL addressRPU2x1  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0010" &
"0001"; -- RPU's address (8 bits)
SIGNAL addressRPU2x2  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0010" &
"0010"; -- RPU's address (8 bits)
SIGNAL addressRPU2x3  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0010" &
"0011"; -- RPU's address (8 bits)
SIGNAL addressRPU3x0  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0011" &
"0000"; -- RPU's address (8 bits)
SIGNAL addressRPU3x1  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0011" &
"0001"; -- RPU's address (8 bits)
SIGNAL addressRPU3x2  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0011" &
"0010"; -- RPU's address (8 bits)
SIGNAL addressRPU3x3  : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0) := "0011" &
"0011"; -- RPU's address (8 bits)
```

```
SIGNAL data_null      : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0) :=
"00000000000000000000000000000000";
SIGNAL available_null : INTEGER RANGE BUFFER_SIZE DOWNTO 0 := 0;
SIGNAL lane_null      : INTEGER RANGE N_LANES DOWNTO 0 := 0;

PROCEDURE debug_print_data(signal sig_data: IN STD_LOGIC_VECTOR(LEN_WIDTH-1
DOWNTO 0); constant name: IN string) IS
   variable debug_line : line;  -- type 'line' comes from textio
   BEGIN
      IF sig_data /= "00000000000000000000000000000000" THEN
         write(debug_line, string'(name));
         write(debug_line, string'(": "));
         hwrite(debug_line,sig_data);
         writeline(output, debug_line);
         IF sig_data(B_CONTROL DOWNTO E_CONTROL) = CTRL_END THEN
            write(debug_line, string'(""));
            writeline(output, debug_line);
         END IF;
      END IF;
   END PROCEDURE debug_print_data;

BEGIN

-- print signal values
PROCESS(clk,rst)
----------------
BEGIN
   IF (rst='1') THEN

   ELSIF (clk'EVENT AND clk='1') THEN
      debug_print_data(data_00x01, data_00x01'simple_name);
      debug_print_data(data_00x10, data_00x10'simple_name);
      debug_print_data(data_01x00, data_01x00'simple_name);
      debug_print_data(data_01x11, data_01x11'simple_name);
      debug_print_data(data_10x00, data_10x00'simple_name);
      debug_print_data(data_11x01, data_11x01'simple_name);
      debug_print_data(data_11x10, data_11x10'simple_name);
      debug_print_data(data_10x11, data_10x11'simple_name);
      debug_print_data(data_01x02, data_01x02'simple_name);
      debug_print_data(data_02x01, data_02x01'simple_name);
      debug_print_data(data_11x12, data_11x12'simple_name);
      debug_print_data(data_12x11, data_12x11'simple_name);
      debug_print_data(data_02x12, data_02x12'simple_name);
      debug_print_data(data_12x02, data_12x02'simple_name);
      debug_print_data(data_02x03, data_02x03'simple_name);
      debug_print_data(data_03x02, data_03x02'simple_name);
      debug_print_data(data_12x13, data_12x13'simple_name);
      debug_print_data(data_13x12, data_13x12'simple_name);
      debug_print_data(data_03x13, data_03x13'simple_name);
```

```
      debug_print_data(data_13x03, data_13x03'simple_name);
      debug_print_data(data_10x20, data_10x20'simple_name);
      debug_print_data(data_11x21, data_11x21'simple_name);
      debug_print_data(data_12x22, data_12x22'simple_name);
      debug_print_data(data_13x23, data_13x23'simple_name);
      debug_print_data(data_20x10, data_20x10'simple_name);
      debug_print_data(data_21x11, data_21x11'simple_name);
      debug_print_data(data_22x12, data_22x12'simple_name);
      debug_print_data(data_23x13, data_23x13'simple_name);
      debug_print_data(data_20x21, data_20x21'simple_name);
      debug_print_data(data_21x20, data_21x20'simple_name);
      debug_print_data(data_21x22, data_21x22'simple_name);
      debug_print_data(data_22x21, data_22x21'simple_name);
      debug_print_data(data_22x23, data_22x23'simple_name);
      debug_print_data(data_23x22, data_23x22'simple_name);
      debug_print_data(data_20x30, data_20x30'simple_name);
      debug_print_data(data_30x20, data_30x20'simple_name);
      debug_print_data(data_21x31, data_21x31'simple_name);
      debug_print_data(data_31x21, data_31x21'simple_name);
      debug_print_data(data_22x32, data_22x32'simple_name);
      debug_print_data(data_32x22, data_32x22'simple_name);
      debug_print_data(data_23x33, data_23x33'simple_name);
      debug_print_data(data_33x23, data_33x23'simple_name);
      debug_print_data(data_30x31, data_30x31'simple_name);
      debug_print_data(data_31x30, data_31x30'simple_name);
      debug_print_data(data_31x32, data_31x32'simple_name);
      debug_print_data(data_32x31, data_32x31'simple_name);
      debug_print_data(data_32x33, data_32x33'simple_name);
      debug_print_data(data_33x32, data_33x32'simple_name);
   END IF;
END PROCESS;

rpu0x0 : entity work.rpu
   PORT MAP(clk, rst, addressRPU0x0,
   --       NORTH             SOUTH             EAST              WEST
            data_null,        data_10x00,       data_01x00,
      dataIn_west_00,   validData_00in,      -- data IN
            available_null,   available_10x00,  available_01x00,
 availableIn_west_00,                   -- available IN
            lane_null,        lane_10x00,       lane_01x00,
      laneIn_west_00,                       -- lane IN

            open,             data_00x10,       data_00x01,
      dataOut_west_00,  validData_00out,     -- data OUT
            open,             available_00x10,  available_00x01,
 availableOut_west_00,                  -- available OUT
            open,             lane_00x10,       lane_00x01,
      laneOut_west_00);                     -- lane OUT
```

```
rpu0x1 : entity work.rpu
   PORT MAP(clk, rst, addressRPU0x1,
   --       NORTH             SOUTH             EAST              WEST
            data_null,        data_11x01,       data_02x01,       data_00x01,
      validData_01in,      -- data IN
            available_null,   available_11x01,  available_02x01,
 available_00x01,                       -- available IN
            lane_null,        lane_11x01,       lane_02x01,       lane_00x01,
                           -- lane IN

            open,             data_01x11,       data_01x02,       data_01x00,
      validData_01out,     -- data OUT
            open,             available_01x11,  available_01x02,
 available_01x00,                       -- available OUT
            open,             lane_01x11,       lane_01x02,       lane_01x00);
                           -- lane OUT

rpu0x2 : entity work.rpu
   PORT MAP(clk, rst, addressRPU0x1,
   --       NORTH             SOUTH             EAST              WEST
            data_null,        data_12x02,       data_03x02,       data_01x02,
      validData_02in,      -- data IN
            available_null,   available_12x02,  available_03x02,
 available_01x02,                       -- available IN
            lane_null,        lane_12x02,       lane_03x02,       lane_01x02,
                           -- lane IN

            open,             data_02x12,       data_02x03,       data_02x01,
      validData_02out,     -- data OUT
            open,             available_02x12,  available_02x03,
 available_02x01,                       -- available OUT
            open,             lane_02x12,       lane_02x03,       lane_02x01);
                           -- lane OUT

rpu0x3 : entity work.rpu
   PORT MAP(clk, rst, addressRPU0x3,
   --       NORTH             SOUTH             EAST              WEST
            data_null,        data_13x03,       dataIn_east_03,   data_02x03,
      validData_03in,      -- data IN
            available_null,   available_13x03,  availableIn_east_03,
available_02x03,                    -- available IN
            lane_null,        lane_13x03,       laneIn_east_03,   lane_02x03,
                           -- lane IN

            open,             data_03x13,       dataOut_east_03,  data_03x02,
      validData_03out,     -- data OUT
            open,             available_03x13,  availableOut_east_03,
available_03x02,                   -- available OUT
```

```
            open,             lane_03x13,       laneOut_east_03,  lane_03x02);
                           -- lane OUT

rpu1x0 : entity work.rpu
   PORT MAP(clk, rst, addressRPU1x0,
   --       NORTH             SOUTH             EAST              WEST
            data_00x10,       data_20x10,       data_11x10,       data_null,
        validData_10in,      -- data IN
            available_00x10,  available_20x10,  available_11x10,
 available_null,                         -- available IN
            lane_00x10,       lane_20x10,       lane_11x10,       lane_null,
                            -- lane IN

            data_10x00,       data_10x20,       data_10x11,       open,
            validData_10out,     -- data OUT
            available_10x00,  available_10x20,  available_10x11,
 open,                                   -- available OUT
            lane_10x00,       lane_10x20,       lane_10x11,       open);
                              -- lane OUT

rpu1x1 : entity work.rpu
   PORT MAP(clk, rst, addressRPU1x1,
   --       NORTH             SOUTH             EAST              WEST
            data_01x11,       data_21x11,       data_12x11,       data_10x11,
      validData_11in,      -- data IN
            available_01x11,  available_21x11,  available_12x11,
 available_10x11,                        -- available IN
            lane_01x11,       lane_21x11,       lane_12x11,       lane_10x11,
                           -- lane IN

            data_11x01,       data_11x21,       data_11x12,       data_11x10,
      validData_11out,     -- data OUT
            available_11x01,  available_11x21,  available_11x12,
 available_11x10,                        -- available OUT
            lane_11x01,       lane_11x21,       lane_11x12,       lane_11x10);
                           -- lane OUT

rpu1x2 : entity work.rpu
   PORT MAP(clk, rst, addressRPU1x2,
   --       NORTH             SOUTH             EAST              WEST
            data_02x12,       data_22x12,       data_13x12,       data_11x12,
      validData_12in,      -- data IN
            available_02x12,  available_22x12,  available_13x12,
 available_11x12,                        -- available IN
            lane_02x12,       lane_22x12,       lane_13x12,       lane_11x12,
                           -- lane IN

            data_12x02,       data_12x22,       data_12x13,       data_12x11,
      validData_12out,     -- data OUT
```

```
            available_12x02,  available_12x22,  available_12x13,
 available_12x11,                        -- available OUT
            lane_12x02,       lane_12x22,       lane_12x13,       lane_12x11);
                           -- lane OUT

rpu1x3 : entity work.rpu
   PORT MAP(clk, rst, addressRPU1x3,
   --       NORTH             SOUTH             EAST              WEST
            data_03x13,       data_23x13,       data_null,        data_12x13,
      validData_13in,      -- data IN
            available_03x13,  available_23x13,  available_null,
  available_12x13,                         -- available IN
            lane_03x13,       lane_23x13,       lane_null,        lane_12x13,
                           -- lane IN

            data_13x03,       data_13x23,       open,             data_13x12,
      validData_13out,     -- data OUT
            available_13x03,  available_13x23,  open,
            available_13x12,                       -- available OUT
            lane_13x03,       lane_13x23,       open,             lane_13x12);
                           -- lane OUT

rpu2x0 : entity work.rpu
   PORT MAP(clk, rst, addressRPU2x0,
   --       NORTH             SOUTH             EAST              WEST
            data_10x20,       data_30x20,       data_21x20,       data_null,
       validData_20in,      -- data IN
            available_10x20,  available_30x20,  available_21x20,
 available_null,                         -- available IN
            lane_10x20,       lane_30x20,       lane_21x20,       lane_null,
                           -- lane IN

            data_20x10,       data_20x30,       data_20x21,       open,
            validData_20out,     -- data OUT
            available_20x10,  available_20x30,  available_20x21,
 open,                                   -- available OUT
            lane_20x10,       lane_20x30,       lane_20x21,       open);
                             -- lane OUT

rpu2x1 : entity work.rpu
   PORT MAP(clk, rst, addressRPU2x1,
   --       NORTH             SOUTH             EAST              WEST
            data_11x21,       data_31x21,       data_22x21,       data_20x21,
      validData_21in,      -- data IN
            available_11x21,  available_31x21,  available_22x21,
 available_20x21,                        -- available IN
            lane_11x21,       lane_31x21,       lane_22x21,       lane_20x21,
                           -- lane IN
```

```
            data_21x11,       data_21x31,       data_21x22,       data_21x20,
      validData_21out,     -- data OUT
            available_21x11,  available_21x31,  available_21x22,
 available_21x20,                        -- available OUT
            lane_21x11,       lane_21x31,       lane_21x22,       lane_21x20);
                           -- lane OUT

rpu2x2 : entity work.rpu
   PORT MAP(clk, rst, addressRPU2x2,
   --       NORTH             SOUTH             EAST              WEST
            data_12x22,       data_32x22,       data_23x22,       data_21x22,
      validData_22in,      -- data IN
            available_12x22,  available_32x22,  available_23x22,
 available_21x22,                        -- available IN
            lane_12x22,       lane_32x22,       lane_23x22,       lane_21x22,
                           -- lane IN

            data_22x12,       data_22x32,       data_22x23,       data_22x21,
      validData_22out,     -- data OUT
            available_22x12,  available_22x32,  available_22x23,
 available_22x21,                        -- available OUT
            lane_22x12,       lane_22x32,       lane_22x23,       lane_22x21);
                           -- lane OUT

rpu2x3 : entity work.rpu
   PORT MAP(clk, rst, addressRPU2x3,
   --       NORTH             SOUTH             EAST              WEST
            data_13x23,       data_33x23,       data_null,        data_22x23,
      validData_23in,      -- data IN
            available_13x23,  available_33x23,  available_null,
  available_22x23,                         -- available IN
            lane_13x23,       lane_33x23,       lane_null,        lane_22x23,
                           -- lane IN

            data_23x13,       data_23x33,       open,             data_23x22,
      validData_23out,     -- data OUT
            available_23x13,  available_23x33,  open,
            available_23x22,                         -- available OUT
            lane_23x13,       lane_23x33,       open,             lane_23x22);
                           -- lane OUT

rpu3x0 : entity work.rpu
   PORT MAP(clk, rst, addressRPU3x0,
   --       NORTH             SOUTH             EAST              WEST
            data_20x30,       data_null,        data_31x30,
      dataIn_west_30,   validData_30in,      -- data IN
            available_20x30,  available_null,   available_31x30,
 availableIn_west_30,                    -- available IN
```

```
            lane_20x30,       lane_null,        lane_31x30,
      laneIn_west_30,                        -- lane IN

            data_30x20,       open,             data_30x31,
      dataOut_west_30,  validData_30out,     -- data OUT
            available_30x20,  open,             available_30x31,
 availableOut_west_30,                  -- available OUT
            lane_30x20,       open,             lane_30x31,
      laneOut_west_30);                     -- lane OUT

rpu3x1 : entity work.rpu
   PORT MAP(clk, rst, addressRPU3x1,
   --       NORTH             SOUTH             EAST              WEST
            data_21x31,       data_null,        data_32x31,       data_30x31,
      validData_31in,      -- data IN
            available_21x31,  available_null,   available_32x31,
 available_30x31,                       -- available IN
            lane_21x31,       lane_null,        lane_32x31,       lane_30x31,
                           -- lane IN

            data_31x21,       open,             data_31x32,       data_31x30,
      validData_31out,     -- data OUT
            available_31x21,  open,             available_31x32,
 available_31x30,                       -- available OUT
            lane_31x21,       open,             lane_31x32,       lane_31x30);
                           -- lane OUT

rpu3x2 : entity work.rpu
   PORT MAP(clk, rst, addressRPU3x2,
   --       NORTH             SOUTH             EAST              WEST
            data_22x32,       data_null,        data_33x32,       data_31x32,
      validData_32in,      -- data IN
            available_22x32,  available_null,   available_33x32,
 available_31x32,                       -- available IN
            lane_22x32,       lane_null,        lane_33x32,       lane_31x32,
                           -- lane IN

            data_32x22,       open,             data_32x33,       data_32x31,
      validData_32out,     -- data OUT
            available_32x22,  open,             available_32x33,
 available_32x31,                       -- available OUT
            lane_32x22,       open,             lane_32x33,       lane_32x31);
                           -- lane OUT

rpu3x3 : entity work.rpu
   PORT MAP(clk, rst, addressRPU3x3,
   --       NORTH             SOUTH             EAST              WEST
            data_23x33,       data_null,        dataIn_east_33,   data_32x33,
      validData_33in,      -- data IN
```

```
            available_23x33,  available_null,   availableIn_east_33,
available_32x33,                    -- available IN
            lane_23x33,       lane_null,        laneIn_east_33,   lane_32x33,
                          -- lane IN

            data_33x23,       open,             data_33x32,
      dataOut_east_33,  validData_33out,     -- data OUT
            available_33x23,  open,             available_33x32,
 availableOut_east_33,                   -- available OUT
            lane_33x23,       open,             lane_33x32,
      laneOut_east_33);                     -- lane OUT


validData_00in(P_NORTH) <= '0';
validData_00in(P_SOUTH) <= validData_10out(P_NORTH);
validData_00in(P_EAST) <= validData_01out(P_WEST);
validData_00in(P_WEST) <= validDataIn_00(P_WEST);

validData_01in(P_NORTH) <= '0';
validData_01in(P_SOUTH) <= validData_11out(P_NORTH);
validData_01in(P_EAST) <= validData_02out(P_WEST);
validData_01in(P_WEST) <= validData_00out(P_EAST);

validData_02in(P_NORTH) <= '0';
validData_02in(P_SOUTH) <= validData_12out(P_NORTH);
validData_02in(P_EAST) <= validData_03out(P_WEST);
validData_02in(P_WEST) <= validData_01out(P_EAST);

validData_03in(P_NORTH) <= '0';
validData_03in(P_SOUTH) <= validData_13out(P_NORTH);
validData_03in(P_EAST) <= validDataIn_03(P_EAST);
validData_03in(P_WEST) <= validData_02out(P_EAST);

validData_10in(P_NORTH) <= validData_00out(P_SOUTH);
validData_10in(P_SOUTH) <= validData_20out(P_NORTH);
validData_10in(P_EAST) <= validData_11out(P_WEST);
validData_10in(P_WEST) <= '0';

validData_11in(P_NORTH) <= validData_01out(P_SOUTH);
validData_11in(P_SOUTH) <= validData_21out(P_NORTH);
validData_11in(P_EAST) <= validData_12out(P_WEST);
validData_11in(P_WEST) <= validData_10out(P_EAST);

validData_12in(P_NORTH) <= validData_02out(P_SOUTH);
validData_12in(P_SOUTH) <= validData_22out(P_NORTH);
validData_12in(P_EAST) <= validData_13out(P_WEST);
validData_12in(P_WEST) <= validData_11out(P_EAST);

validData_13in(P_NORTH) <= validData_03out(P_SOUTH);
```

```
validData_13in(P_SOUTH) <= validData_23out(P_NORTH);
validData_13in(P_EAST) <= '0';
validData_13in(P_WEST) <= validData_12out(P_EAST);

validData_20in(P_NORTH) <= validData_10out(P_SOUTH);
validData_20in(P_SOUTH) <= validData_30out(P_NORTH);
validData_20in(P_EAST) <= validData_21out(P_WEST);
validData_20in(P_WEST) <= '0';

validData_21in(P_NORTH) <= validData_11out(P_SOUTH);
validData_21in(P_SOUTH) <= validData_31out(P_NORTH);
validData_21in(P_EAST) <= validData_22out(P_WEST);
validData_21in(P_WEST) <= validData_20out(P_EAST);

validData_22in(P_NORTH) <= validData_12out(P_SOUTH);
validData_22in(P_SOUTH) <= validData_31out(P_NORTH);
validData_22in(P_EAST) <= validData_23out(P_WEST);
validData_22in(P_WEST) <= validData_21out(P_EAST);

validData_23in(P_NORTH) <= validData_13out(P_SOUTH);
validData_23in(P_SOUTH) <= validData_33out(P_NORTH);
validData_23in(P_EAST) <= '0';
validData_23in(P_WEST) <= validData_22out(P_EAST);

validData_30in(P_NORTH) <= validData_20out(P_SOUTH);
validData_30in(P_SOUTH) <= '0';
validData_30in(P_EAST) <= validData_31out(P_WEST);
validData_30in(P_WEST) <= validDataIn_30(P_WEST);

validData_31in(P_NORTH) <= validData_21out(P_SOUTH);
validData_31in(P_SOUTH) <= '0';
validData_31in(P_EAST) <= validData_32out(P_WEST);
validData_31in(P_WEST) <= validData_30out(P_EAST);

validData_32in(P_NORTH) <= validData_22out(P_SOUTH);
validData_32in(P_SOUTH) <= '0';
validData_32in(P_EAST) <= validData_33out(P_WEST);
validData_32in(P_WEST) <= validData_31out(P_EAST);

validData_33in(P_NORTH) <= validData_23out(P_SOUTH);
validData_33in(P_SOUTH) <= '0';
validData_33in(P_EAST) <= validDataIn_33(P_EAST);
validData_33in(P_WEST) <= validData_32out(P_EAST);

END behaviour;
```

- Testbench: Malha 2D 4x4 de RPUs

```
LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.all;
USE work.ipnosys_package.all;
USE STD.textio.all;
USE IEEE.std_logic_textio.all;

-----------------
ENTITY tb_ipnosys_4x4 IS
END tb_ipnosys_4x4;
-----------------

ARCHITECTURE TBArch OF tb_ipnosys_4x4 IS

SIGNAL clk, rst: STD_LOGIC := '0';

    --RPU 0x0
    SIGNAL dataIn_west_00        : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    SIGNAL validDataIn_00        : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    SIGNAL availableIn_west_00   : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    SIGNAL laneIn_west_00        : INTEGER RANGE N_LANES DOWNTO 0;

    SIGNAL dataOut_west_00       : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    SIGNAL availableOut_west_00  : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    SIGNAL laneOut_west_00       : INTEGER RANGE N_LANES DOWNTO 0; -- Lanes
for virtual channel (lane 0: control packets; others: regular packets)

--================================================================

    --RPU 0x3
    SIGNAL dataIn_east_03        : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    SIGNAL validDataIn_03        : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    SIGNAL availableIn_east_03   : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    SIGNAL laneIn_east_03        : INTEGER RANGE N_LANES DOWNTO 0;

    SIGNAL dataOut_east_03       : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    SIGNAL availableOut_east_03  : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    SIGNAL laneOut_east_03       : INTEGER RANGE N_LANES DOWNTO 0;

--================================================================

    --RPU 3x0
    SIGNAL dataIn_west_30        : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    SIGNAL validDataIn_30        : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    SIGNAL availableIn_west_30   : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    SIGNAL laneIn_west_30        : INTEGER RANGE N_LANES DOWNTO 0;
```

```
    SIGNAL dataOut_west_30       : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    SIGNAL availableOut_west_30  : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    SIGNAL laneOut_west_30       : INTEGER RANGE N_LANES DOWNTO 0; -- Lanes
for virtual channel (lane 0: control packets; others: regular packets)

--================================================================

    --RPU 3x3
    SIGNAL dataIn_east_33        : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    SIGNAL validDataIn_33        : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    SIGNAL availableIn_east_33   : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    SIGNAL laneIn_east_33        : INTEGER RANGE N_LANES DOWNTO 0;

    SIGNAL dataOut_east_33       : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    SIGNAL availableOut_east_33  : INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    SIGNAL laneOut_east_33       : INTEGER RANGE N_LANES DOWNTO 0;

--DEBUG

SIGNAL clken_p : boolean := true;
CONSTANT period : TIME := 50 ns;
SIGNAL count_cycle : INTEGER := 1;

PROCEDURE debug_print_data IS
   variable debug_line : line;  -- type 'line' comes from textio
   BEGIN
      IF clk = '1' THEN
         write(debug_line, string'("- Ciclo "));
         write(debug_line,count_cycle);
         writeline(output, debug_line);
      END IF;
   END PROCEDURE debug_print_data;

BEGIN
   CompToTest: entity work.ipnosys_4x4
      PORT MAP (clk, rst,
                dataIn_west_00, validDataIn_00, availableIn_west_00,
laneIn_west_00, dataOut_west_00, availableOut_west_00, laneOut_west_00,
                dataIn_east_03, validDataIn_03, availableIn_east_03,
laneIn_east_03, dataOut_east_03, availableOut_east_03, laneOut_east_03,
                dataIn_west_30, validDataIn_30, availableIn_west_30,
laneIn_west_30, dataOut_west_30, availableOut_west_30, laneOut_west_30,
                dataIn_east_33, validDataIn_33, availableIn_east_33,
laneIn_east_33, dataOut_east_33, availableOut_east_33, laneOut_east_33
      );

   -- Reset generation
   reset: rst <= '1', '0' AFTER period/2;
```

```
-- Clock generation
clock_proc: PROCESS
   BEGIN
      WHILE clken_p LOOP
        --clk <= NOT clk AFTER period/2;
        clk <= '0'; WAIT FOR period/2;
        clk <= '1'; WAIT FOR period/2;
        count_cycle <= count_cycle + 1;
        debug_print_data;
      END LOOP;
     WAIT;
END PROCESS;

-- Waveform generation
waveGen_Proc: PROCESS

   BEGIN
      --reset
      WAIT FOR period/2;

      ---------------------------------
      -- Caso de Teste:
      -----------------
      -- Aplicação Real: Média
      ---------------------------------

      --------------
      -- 1 Header Word
      --------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
      dataIn_west_00(B_TARGET DOWNTO E_TARGET) <= "0011" & "0011";
      dataIn_west_00(B_SOURCE DOWNTO E_SOURCE) <= "0000" & "0000";
      dataIn_west_00(B_N_INSTRUCTIONS DOWNTO E_N_INSTRUCTIONS) <=
std_logic_vector(to_unsigned(2, LEN_N_INSTRUCTIONS));
      dataIn_west_00(B_REROUTE DOWNTO E_REROUTE) <= SPIRAL_1;
      dataIn_west_00(B_TYPE DOWNTO E_TYPE) <= PKT_REGULAR;

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      ---------------------
      -- 2 Packet Number Word
      ---------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
      dataIn_west_00(B_PROCESS_NUM DOWNTO E_PROCESS_NUM) <= (OTHERS => '0');
      dataIn_west_00(B_PACKET_NUM DOWNTO E_PACKET_NUM) <= (OTHERS => '0');

      validDataIn_00(P_WEST) <= '1';
```

```
      WAIT FOR period;

      ----------------------
      -- 3 Packet Pointer Word
      ----------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
      dataIn_west_00(B_PROCESS_ADDR DOWNTO E_PROCESS_ADDR) <= (OTHERS => '0');
      dataIn_west_00(B_INST_RPU DOWNTO E_INST_RPU) <=
std_logic_vector(to_unsigned(1, LEN_INST_RPU));
      dataIn_west_00(B_POINTER DOWNTO E_POINTER) <=
std_logic_vector(to_unsigned(4, LEN_POINTER)); -- Apontador sempre deve comeca
em 3

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --////////////////////////////////////////////////////////////////

      --------------------------------
      -- 4 Instruction Word (ADD)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(ADD, LEN_INSTRUCTION));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(8, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(8, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 5 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(10, LEN_OPERAND)); -- 1

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 6 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
```

```
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(6, LEN_OPERAND)); -- 2

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --------------------------------
      -- 7 Instruction Word (DIV)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(DIV, LEN_INSTRUCTION));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(10, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(10, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 9 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(2, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -------
      -- 11 END
      ---------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_END;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <= EOP;

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      dataIn_west_00 <= (OTHERS => '0');
      validDataIn_00(P_WEST) <= '0';

      FOR i IN 0 TO 32 LOOP
         --IF count_cycle = 52 THEN
         -- clken_p <= true;
         --END IF;
         WAIT FOR period;
```

```
        END LOOP;

        ---------------------------------
        -- Caso de Teste:
        -----------------
        -- Aplicação Real: Branch
        ---------------------------------

        --------------
        -- 1 Header Word
        --------------
        dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
        dataIn_west_00(B_TARGET DOWNTO E_TARGET) <= "0011" & "0011";
        dataIn_west_00(B_SOURCE DOWNTO E_SOURCE) <= "0000" & "0000";
        dataIn_west_00(B_N_INSTRUCTIONS DOWNTO E_N_INSTRUCTIONS) <=
std_logic_vector(to_unsigned(3, LEN_N_INSTRUCTIONS));
        dataIn_west_00(B_REROUTE DOWNTO E_REROUTE) <= SPIRAL_1;
        dataIn_west_00(B_TYPE DOWNTO E_TYPE) <= PKT_REGULAR;

        validDataIn_00(P_WEST) <= '1';
        WAIT FOR period;

        ---------------------
        -- 2 Packet Number Word
        ---------------------
        dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
        dataIn_west_00(B_PROCESS_NUM DOWNTO E_PROCESS_NUM) <= (OTHERS => '0');
        dataIn_west_00(B_PACKET_NUM DOWNTO E_PACKET_NUM) <= (OTHERS => '0');

        validDataIn_00(P_WEST) <= '1';
        WAIT FOR period;

        ----------------------
        -- 3 Packet Pointer Word
        ----------------------
        dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
        dataIn_west_00(B_PROCESS_ADDR DOWNTO E_PROCESS_ADDR) <= (OTHERS => '0');
        dataIn_west_00(B_INST_RPU DOWNTO E_INST_RPU) <=
std_logic_vector(to_unsigned(1, LEN_INST_RPU));
        dataIn_west_00(B_POINTER DOWNTO E_POINTER) <=
std_logic_vector(to_unsigned(4, LEN_POINTER)); -- Apontador sempre deve comeca
em 3

        validDataIn_00(P_WEST) <= '1';
        WAIT FOR period;

        --////////////////////////////////////////////////////////

        --------------------------------
```

```
      -- 4 Instruction Word (Branch)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(BNE, LEN_INSTRUCTION));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(10, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(0, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 5 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(10, LEN_OPERAND)); -- 1

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 6 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(6, LEN_OPERAND)); -- 2

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --------------------------------
      -- 7 Instruction Word (ADD)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(ADD, LEN_INSTRUCTION));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(13, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(13, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
```

```
      WAIT FOR period;

      -----------------
      -- 8 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(5, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 9 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(2, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --------------------------------
      -- 10 Instruction Word (SUB)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(SUBB, LEN_INSTRUCTION));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(13, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(13, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 11 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(10, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
```

```
      -- 12 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(4, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -------
      -- 14 END
      ---------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_END;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <= EOP;

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      dataIn_west_00 <= (OTHERS => '0');
      validDataIn_00(P_WEST) <= '0';

      FOR i IN 0 TO 50 LOOP
         --IF count_cycle = 52 THEN
         -- clken_p <= true;
         --END IF;
         WAIT FOR period;
      END LOOP;

      ---------------------------------
      -- Caso de Teste:
      -----------------
      -- Aplicação Real: Loop
      ---------------------------------

      --------------
      -- 1 Header Word
      --------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
      dataIn_west_00(B_TARGET DOWNTO E_TARGET) <= "0011" & "0011";
      dataIn_west_00(B_SOURCE DOWNTO E_SOURCE) <= "0000" & "0000";
      dataIn_west_00(B_N_INSTRUCTIONS DOWNTO E_N_INSTRUCTIONS) <=
std_logic_vector(to_unsigned(7, LEN_N_INSTRUCTIONS));
      dataIn_west_00(B_REROUTE DOWNTO E_REROUTE) <= SPIRAL_1;
      dataIn_west_00(B_TYPE DOWNTO E_TYPE) <= PKT_REGULAR;

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      ---------------------
```

```
      -- 2 Packet Number Word
      ---------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
      dataIn_west_00(B_PROCESS_NUM DOWNTO E_PROCESS_NUM) <= (OTHERS => '0');
      dataIn_west_00(B_PACKET_NUM DOWNTO E_PACKET_NUM) <= (OTHERS => '0');

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      ----------------------
      -- 3 Packet Pointer Word
      ----------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_HEADER;
      dataIn_west_00(B_PROCESS_ADDR DOWNTO E_PROCESS_ADDR) <= (OTHERS => '0');
      dataIn_west_00(B_INST_RPU DOWNTO E_INST_RPU) <=
std_logic_vector(to_unsigned(1, LEN_INST_RPU));
      dataIn_west_00(B_POINTER DOWNTO E_POINTER) <=
std_logic_vector(to_unsigned(4, LEN_POINTER)); -- Apontador sempre deve comeca
em 3

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --////////////////////////////////////////////////////////

      --------------------------------
      -- 4 Instruction Word (COPY)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(COPY, LEN_INSTRUCTION));
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(1, LEN_N_OPERANDS));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(7, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(10, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 5 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(0, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
```

```
      WAIT FOR period;

      --------------------------------
      -- 6 Instruction Word (BE)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(BE, LEN_INSTRUCTION));
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(28, LEN_RESULT)); -- TODO útima posição
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(0, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 8 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(1, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --------------------------------
      -- 9 Instruction Word (ADD)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(ADD, LEN_INSTRUCTION));
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(13, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(13, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 11 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
```

```
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(1, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --------------------------------
      -- 12 Instruction Word (COPY)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(COPY, LEN_INSTRUCTION));
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(1, LEN_N_OPERANDS));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(15, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(18, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --------------------------------
      -- 14 Instruction Word (BE)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(BE, LEN_INSTRUCTION));
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(20, LEN_RESULT)); -- TODO útima posição
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(0, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 16 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(1, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --------------------------------
```

```
      -- 17 Instruction Word (ADD)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(ADD, LEN_INSTRUCTION));
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(23, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(0, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 19 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(1, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      --------------------------------
      -- 20 Instruction Word (MULT)
      --------------------------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_INSTRUCTION;
      dataIn_west_00(B_INSTRUCTION DOWNTO E_INSTRUCTION) <=
std_logic_vector(to_unsigned(MUL, LEN_INSTRUCTION));
      dataIn_west_00(B_N_OPERANDS DOWNTO E_N_OPERANDS) <=
std_logic_vector(to_unsigned(2, LEN_N_OPERANDS));
      dataIn_west_00(B_RESULT_1 DOWNTO E_RESULT_1) <=
std_logic_vector(to_unsigned(31, LEN_RESULT));
      dataIn_west_00(B_RESULT_2 DOWNTO E_RESULT_2) <=
std_logic_vector(to_unsigned(0, LEN_RESULT));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -----------------
      -- 21 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(3, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
```

```
      WAIT FOR period;

      -----------------
      -- 22 Operand Word
      -----------------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_OPERAND;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <=
std_logic_vector(to_signed(3, LEN_OPERAND));

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      -------
      -- 23 END
      ---------
      dataIn_west_00(B_CONTROL DOWNTO E_CONTROL) <= CTRL_END;
      dataIn_west_00(B_OPERAND DOWNTO E_OPERAND) <= EOP;

      validDataIn_00(P_WEST) <= '1';
      WAIT FOR period;

      dataIn_west_00 <= (OTHERS => '0');
      validDataIn_00(P_WEST) <= '0';

      FOR i IN 0 TO 100 LOOP
         IF count_cycle = 52 THEN
            clken_p <= true;
         END IF;
         WAIT FOR period;
      END LOOP;

      clken_p <= false;
      WAIT;
   END PROCESS;
END TBArch;
```

- RPU

```
LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE work.ipnosys_package.all;
----------------------------
ENTITY rpu IS
----------------------------
  PORT(
    -- System signals
    clk : IN STD_LOGIC; -- clock
```

```
    rst : IN STD_LOGIC; -- reset

    addressRPU : IN STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0); -- RPU's address
(8 bits)

    dataIn_north : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_south : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_east  : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_west  : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataIn  : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    availableIn_north : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0; -- Espaço do
Buffer(Só um? Controle ou Regular???) de outra RPU para o árbitro.
    availableIn_south : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableIn_east  : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableIn_west  : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneIn_north      : IN INTEGER RANGE N_LANES DOWNTO 0; -- Lanes for
virtual channel (lane 0: control packets; others: regular packets)
    laneIn_south      : IN INTEGER RANGE N_LANES DOWNTO 0;
    laneIn_east       : IN INTEGER RANGE N_LANES DOWNTO 0;
    laneIn_west       : IN INTEGER RANGE N_LANES DOWNTO 0;

    dataOut_north : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataOut_south : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataOut_east  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataOut_west  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataOut  : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    availableOut_north : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0; -- Espaço do
Buffers (N,S,W,E) Internos da RPU
    availableOut_south : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableOut_east  : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableOut_west  : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    laneOut_north      : OUT INTEGER RANGE N_LANES DOWNTO 0;
    laneOut_south      : OUT INTEGER RANGE N_LANES DOWNTO 0;
    laneOut_east       : OUT INTEGER RANGE N_LANES DOWNTO 0;
    laneOut_west       : OUT INTEGER RANGE N_LANES DOWNTO 0); -- Lanes for
virtual channel (lane 0: control packets; others: regular packets)
END rpu;

---------------------------------------
ARCHITECTURE behaviour OF rpu IS
---------------------------------------

TYPE TYPE_CB_BUFFER_IN IS ARRAY(ROUTER_PORTS DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
TYPE TYPE_DATA IS ARRAY(ROUTER_PORTS-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
TYPE TYPE_DIRECTION IS ARRAY(ROUTER_PORTS DOWNTO 0) OF INTEGER RANGE
ROUTER_PORTS DOWNTO 0;
```

```
TYPE TYPE_INSTRUCTION IS ARRAY(ROUTER_PORTS-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
TYPE TYPE_EXCEPTION IS ARRAY(ROUTER_PORTS-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
TYPE TYPE_BUFFER_AVAILABLE IS ARRAY(RESULT_SIZE-1 DOWNTO 0) OF INTEGER RANGE
BUFFER_SIZE DOWNTO 0;
-------------
-- SIGNALS
-------------
SIGNAL sig_buf_cb_canpop     : STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);
SIGNAL sig_cb_arb_canpop     : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_cb_buf_pop        : STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);
SIGNAL sig_arb_cb_pop        : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_buf_dataOut       : TYPE_CB_BUFFER_IN;
SIGNAL sig_buf_cb_validData  : STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);
SIGNAL sig_cb_arb_validData  : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

SIGNAL sig_cb_req, sig_cb_ackreq: STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);
SIGNAL sig_cb_direction : TYPE_DIRECTION;
SIGNAL sig_cb_dataOut   : TYPE_DATA;
SIGNAL sig_buf_cb_header, sig_buf_cb_pcknumber, sig_buf_cb_pckpointer :
TYPE_DATA;
SIGNAL sig_cb_arb_header, sig_cb_arb_pcknumber, sig_cb_arb_pckpointer :
TYPE_DATA;

SIGNAL sig_alu_dataA       : TYPE_DATA;
SIGNAL sig_alu_dataB       : TYPE_DATA;
SIGNAL sig_alu_operation   : TYPE_INSTRUCTION;
SIGNAL sig_alu_validDataIn : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_alu_result      : TYPE_DATA;
SIGNAL sig_alu_exception   : TYPE_EXCEPTION;
SIGNAL sig_alu_validResult : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_alu_neg         : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_alu_zero        : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_alu_request     : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_alu_ackRequest  : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

SIGNAL sig_su_dataIn       : TYPE_DATA;
SIGNAL sig_su_validDataIn  : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_su_dataOut      : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_su_validDataOut : STD_LOGIC;
SIGNAL sig_su_availableIn  : TYPE_BUFFER_AVAILABLE;
SIGNAL sig_su_doneOut      : STD_LOGIC;
SIGNAL sig_su_availableOut : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_su_request      : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_su_ackRequest   : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

BEGIN
```

```
sig_cb_req(P_MOD) <= '0';

rpuPort_N : entity work.rpu_port
   PORT MAP(clk, rst,
            addressRPU, dataIn_north, validDataIn(P_NORTH), laneIn_north,
availableOut_north, -- RPU
            sig_cb_ackreq(P_NORTH), sig_cb_buf_pop(P_NORTH),
            sig_cb_req(P_NORTH), sig_cb_direction(P_NORTH), -- Crossbar
            sig_buf_dataOut(P_NORTH), sig_buf_cb_validData(P_NORTH),
sig_buf_cb_canpop(P_NORTH),
            sig_buf_cb_header(P_NORTH), sig_buf_cb_pcknumber(P_NORTH),
sig_buf_cb_pckpointer(P_NORTH)); -- Árbitro

rpuPort_S : entity work.rpu_port
   PORT MAP(clk, rst,
            addressRPU, dataIn_south, validDataIn(P_SOUTH), laneIn_south,
availableOut_south, -- RPU
            sig_cb_ackreq(P_SOUTH), sig_cb_buf_pop(P_SOUTH),
            sig_cb_req(P_SOUTH), sig_cb_direction(P_SOUTH), -- Crossbar
            sig_buf_dataOut(P_SOUTH), sig_buf_cb_validData(P_SOUTH),
sig_buf_cb_canpop(P_SOUTH),
            sig_buf_cb_header(P_SOUTH), sig_buf_cb_pcknumber(P_SOUTH),
sig_buf_cb_pckpointer(P_SOUTH)); -- Árbitro

rpuPort_E : entity work.rpu_port
   PORT MAP(clk, rst,
            addressRPU, dataIn_east, validDataIn(P_EAST), laneIn_east,
availableOut_east, -- RPU
            sig_cb_ackreq(P_EAST), sig_cb_buf_pop(P_EAST),
            sig_cb_req(P_EAST), sig_cb_direction(P_EAST), -- Crossbar
            sig_buf_dataOut(P_EAST), sig_buf_cb_validData(P_EAST),
sig_buf_cb_canpop(P_EAST),
            sig_buf_cb_header(P_EAST), sig_buf_cb_pcknumber(P_EAST),
sig_buf_cb_pckpointer(P_EAST)); -- Árbitro

rpuPort_W : entity work.rpu_port
   PORT MAP(clk, rst,
            addressRPU, dataIn_west, validDataIn(P_WEST), laneIn_west,
availableOut_west, -- RPU
            sig_cb_ackreq(P_WEST), sig_cb_buf_pop(P_WEST),
            sig_cb_req(P_WEST), sig_cb_direction(P_WEST), -- Crossbar
            sig_buf_dataOut(P_WEST), sig_buf_cb_validData(P_WEST),
sig_buf_cb_canpop(P_WEST),
            sig_buf_cb_header(P_WEST), sig_buf_cb_pcknumber(P_WEST),
sig_buf_cb_pckpointer(P_WEST)); -- Árbitro

cb : entity work.cb
   PORT MAP(clk, rst,
            sig_buf_dataOut(P_MOD),
```

```
            sig_buf_dataOut(P_NORTH),
            sig_buf_cb_header(P_NORTH), sig_buf_cb_pcknumber(P_NORTH),
sig_buf_cb_pckpointer(P_NORTH),
            sig_buf_dataOut(P_SOUTH),
            sig_buf_cb_header(P_SOUTH), sig_buf_cb_pcknumber(P_SOUTH),
sig_buf_cb_pckpointer(P_SOUTH),
            sig_buf_dataOut(P_EAST),
            sig_buf_cb_header(P_EAST), sig_buf_cb_pcknumber(P_EAST),
sig_buf_cb_pckpointer(P_EAST),
            sig_buf_dataOut(P_WEST),
            sig_buf_cb_header(P_WEST), sig_buf_cb_pcknumber(P_WEST),
sig_buf_cb_pckpointer(P_WEST),
            --
            sig_buf_cb_validData,
            --
            sig_buf_cb_canpop,
            sig_cb_buf_pop,

            sig_cb_dataOut(P_NORTH),
            sig_cb_arb_header(P_NORTH), sig_cb_arb_pcknumber(P_NORTH),
sig_cb_arb_pckpointer(P_NORTH),
            sig_cb_dataOut(P_SOUTH),
            sig_cb_arb_header(P_SOUTH), sig_cb_arb_pcknumber(P_SOUTH),
sig_cb_arb_pckpointer(P_SOUTH),
            sig_cb_dataOut(P_EAST),
            sig_cb_arb_header(P_EAST), sig_cb_arb_pcknumber(P_EAST),
sig_cb_arb_pckpointer(P_EAST),
            sig_cb_dataOut(P_WEST),
            sig_cb_arb_header(P_WEST), sig_cb_arb_pcknumber(P_WEST),
sig_cb_arb_pckpointer(P_WEST),
            --
            sig_cb_arb_validData,
            --
            sig_cb_arb_canpop,
            sig_arb_cb_pop,

            sig_cb_req,
            sig_cb_direction(P_MOD), sig_cb_direction(P_NORTH),
sig_cb_direction(P_SOUTH), sig_cb_direction(P_EAST), sig_cb_direction(P_WEST),
            sig_cb_ackreq);

aluComp: entity work.alu
   PORT MAP(clk, rst,
            sig_alu_dataA(P_NORTH), sig_alu_dataA(P_SOUTH),
sig_alu_dataA(P_EAST), sig_alu_dataA(P_WEST),
            sig_alu_dataB(P_NORTH), sig_alu_dataB(P_SOUTH),
sig_alu_dataB(P_EAST), sig_alu_dataB(P_WEST),
            sig_alu_operation(P_NORTH), sig_alu_operation(P_SOUTH),
sig_alu_operation(P_EAST), sig_alu_operation(P_WEST),
```

```
            sig_alu_validDataIn,
            sig_alu_request,
            sig_alu_result(P_NORTH), sig_alu_result(P_SOUTH),
sig_alu_result(P_EAST), sig_alu_result(P_WEST),
            sig_alu_exception(P_NORTH), sig_alu_exception(P_SOUTH),
sig_alu_exception(P_EAST), sig_alu_exception(P_WEST),
            sig_alu_validResult, sig_alu_ackRequest, sig_alu_neg,
sig_alu_zero);

suComp: entity work.su
   PORT MAP(clk, rst,
            addressRPU,
            sig_su_dataIn(P_NORTH), sig_su_dataIn(P_SOUTH),
sig_su_dataIn(P_EAST), sig_su_dataIn(P_WEST),
            sig_su_validDataIn,
            10, -- Conectar ao Buffer Local
            sig_su_request,
            sig_su_dataOut, sig_su_validDataOut,
            sig_su_availableIn(P_NORTH), sig_su_availableIn(P_SOUTH),
sig_su_availableIn(P_EAST), sig_su_availableIn(P_WEST),
            sig_su_ackRequest);

arbiter_N: entity work.arbiter
   PORT MAP(clk, rst,
            dataOut_north, validDataOut(P_NORTH), availableIn_north,
addressRPU, -- RPU
            sig_arb_cb_pop(P_NORTH), sig_cb_arb_canpop(P_NORTH),
sig_cb_dataOut(P_NORTH), sig_cb_arb_validData(P_NORTH), -- Buffer (Pacote
Regular)
            sig_cb_arb_header(P_NORTH), sig_cb_arb_pcknumber(P_NORTH),
sig_cb_arb_pckpointer(P_NORTH),
            sig_su_dataIn(P_NORTH), sig_su_validDataIn(P_NORTH),
sig_su_request(P_NORTH), sig_su_availableIn(P_NORTH),
sig_su_ackRequest(P_NORTH), -- SU
            sig_alu_dataA(P_NORTH), sig_alu_dataB(P_NORTH),
sig_alu_operation(P_NORTH),
            sig_alu_validDataIn(P_NORTH), sig_alu_request(P_NORTH),
            sig_alu_result(P_NORTH), sig_alu_exception(P_NORTH),
sig_alu_validResult(P_NORTH),
            sig_alu_neg(P_NORTH),
sig_alu_zero(P_NORTH), sig_alu_ackRequest(P_NORTH)); -- ALU

arbiter_S: entity work.arbiter
   PORT MAP(clk, rst,
            dataOut_south, validDataOut(P_SOUTH), availableIn_south,
addressRPU, -- RPU
            sig_arb_cb_pop(P_SOUTH), sig_cb_arb_canpop(P_SOUTH),
sig_cb_dataOut(P_SOUTH), sig_cb_arb_validData(P_SOUTH),-- Buffer (Pacote
Regular)
```

```
            sig_cb_arb_header(P_SOUTH), sig_cb_arb_pcknumber(P_SOUTH),
sig_cb_arb_pckpointer(P_SOUTH),
            sig_su_dataIn(P_SOUTH), sig_su_validDataIn(P_SOUTH),
sig_su_request(P_SOUTH), sig_su_availableIn(P_SOUTH),
sig_su_ackRequest(P_SOUTH), -- SU
            sig_alu_dataA(P_SOUTH), sig_alu_dataB(P_SOUTH),
sig_alu_operation(P_SOUTH),
            sig_alu_validDataIn(P_SOUTH), sig_alu_request(P_SOUTH),
            sig_alu_result(P_SOUTH), sig_alu_exception(P_SOUTH),
sig_alu_validResult(P_SOUTH),
            sig_alu_neg(P_SOUTH),
sig_alu_zero(P_SOUTH), sig_alu_ackRequest(P_SOUTH)); -- ALU

arbiter_E: entity work.arbiter
   PORT MAP(clk, rst,
            dataOut_east, validDataOut(P_EAST), availableIn_east, addressRPU,
-- RPU
            sig_arb_cb_pop(P_EAST), sig_cb_arb_canpop(P_EAST),
sig_cb_dataOut(P_EAST), sig_cb_arb_validData(P_EAST),-- Buffer (Pacote
Regular)
            sig_cb_arb_header(P_EAST), sig_cb_arb_pcknumber(P_EAST),
sig_cb_arb_pckpointer(P_EAST),
            sig_su_dataIn(P_EAST), sig_su_validDataIn(P_EAST),
sig_su_request(P_EAST), sig_su_availableIn(P_EAST), sig_su_ackRequest(P_EAST),
-- SU
            sig_alu_dataA(P_EAST), sig_alu_dataB(P_EAST),
sig_alu_operation(P_EAST),
            sig_alu_validDataIn(P_EAST), sig_alu_request(P_EAST),
            sig_alu_result(P_EAST), sig_alu_exception(P_EAST),
sig_alu_validResult(P_EAST),
            sig_alu_neg(P_EAST),
sig_alu_zero(P_EAST),   sig_alu_ackRequest(P_EAST)); -- ALU

arbiter_W: entity work.arbiter
   PORT MAP(clk, rst,
            dataOut_west, validDataOut(P_WEST), availableIn_west, addressRPU,
-- RPU
            sig_arb_cb_pop(P_WEST), sig_cb_arb_canpop(P_WEST),
sig_cb_dataOut(P_WEST), sig_cb_arb_validData(P_WEST),-- Buffer (Pacote
Regular)
            sig_cb_arb_header(P_WEST), sig_cb_arb_pcknumber(P_WEST),
sig_cb_arb_pckpointer(P_WEST),
            sig_su_dataIn(P_WEST), sig_su_validDataIn(P_WEST),
sig_su_request(P_WEST), sig_su_availableIn(P_WEST), sig_su_ackRequest(P_WEST),
-- SU
            sig_alu_dataA(P_WEST), sig_alu_dataB(P_WEST),
sig_alu_operation(P_WEST),
            sig_alu_validDataIn(P_WEST), sig_alu_request(P_WEST),
```

```
            sig_alu_result(P_WEST), sig_alu_exception(P_WEST),
sig_alu_validResult(P_WEST),
            sig_alu_neg(P_WEST),
sig_alu_zero(P_WEST),   sig_alu_ackRequest(P_WEST));  -- ALU

END behaviour;
```

- *Router*

```
LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE work.ipnosys_package.all;
----------------------------
ENTITY router IS
----------------------------
  PORT(
    -- System signals
    clk : IN STD_LOGIC; -- clock
    rst : IN STD_LOGIC; -- reset

    addressRPU : IN STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0); -- RPU's address
(8 bits)

    -- FROM RPU
    dataIn       : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataIn  : IN STD_LOGIC;
    laneIn       : IN INTEGER RANGE N_LANES DOWNTO 0; -- Lanes for virtual
channel (lane 0: control packets; others: regular packets)
    -- TO RPU
    availableOut : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0; -- FIX-ME: Vetor ou
integer?

    -- FROM CROSSBAR
    cb_ackreq  : IN STD_LOGIC;
    cb_buf_pop : IN STD_LOGIC; -- FROM ARBITER THROUGH CROSSBAR
    -- TO CROSSBAR
    cb_req       : OUT STD_LOGIC;
    cb_direction : OUT INTEGER RANGE ROUTER_PORTS DOWNTO 0;
    cb_dataOut   : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0); -- TO ARBITER
THROUGH CROSSBAR
    cb_validData : OUT STD_LOGIC;
    cb_canPop    : OUT STD_LOGIC; -- TO ARBITER THROUGH CROSSBAR
    cb_header, cb_pcknumber, cb_pckpointer : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1
DOWNTO 0) -- TO ARBITER THROUGH CROSSBAR
    );
END router;
```

```
---------------------------------------
ARCHITECTURE behaviour OF router IS
---------------------------------------

-------------
-- STATES
-------------

TYPE state_type_route IS (rHEAD, rPCKNUMBER, rPCKPOINTER, -- r = route
                          rINSTRUCTION,
                          rINSTRUCTION_CONTROL, rOPERAND_CONTROL,
rEND_CONTROL,
                          rROUTE, rWAIT_ACKREQ, rSEND_HEADER_TO_ARBITER,
rSEND_TO_ARBITER,
                          rEND,
                          rErrState);
SIGNAL current_state_route, next_state_route : state_type_route;

-------------
-- SIGNALS
-------------
SIGNAL sig_buf_dataIn     : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_buf_canpop     : STD_LOGIC;
SIGNAL sig_buf_push       : STD_LOGIC;
SIGNAL sig_buf_pop        : STD_LOGIC;
SIGNAL sig_buf_dataOut    : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_buf_requesting : STD_LOGIC;

SIGNAL header, pcknumber, pckpointer : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

SIGNAL sig_route_pop    : STD_LOGIC;
SIGNAL sig_arbiter_pop  : STD_LOGIC;
SIGNAL sig_rpu_receiver : STD_LOGIC_VECTOR(LEN_TARGET-1 DOWNTO 0);
SIGNAL sig_rpu_sender   : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0);
SIGNAL sig_reroute      : STD_LOGIC_VECTOR(LEN_REROUTE-1 DOWNTO 0);

SIGNAL sig_xLocal : INTEGER; -- Coordenadas Locais
SIGNAL sig_yLocal : INTEGER;
SIGNAL sig_xSend  : INTEGER; -- Quem enviou
SIGNAL sig_ySend  : INTEGER;

SIGNAL sig_direction : INTEGER RANGE ROUTER_PORTS DOWNTO 0 := (UNRESERVED);

BEGIN

buffer_regular: entity work.buffer_fifo PORT MAP(
            clk, rst,
            sig_buf_dataIn,
            OPEN, sig_buf_canpop,
```

```
            sig_buf_push, sig_buf_pop,
            sig_buf_dataOut,
            sig_buf_requesting, OPEN, availableOut);

cb_canPop   <= sig_buf_canpop;
sig_buf_pop <= sig_route_pop;
cb_dataOut  <= sig_buf_dataOut;
cb_direction <= sig_direction;

-- current state
PROCESS(clk,rst)
----------------
BEGIN
   IF (rst='1') THEN
      current_state_route   <= rHEAD;
   ELSIF (clk'EVENT AND clk='1') THEN
      current_state_route   <= next_state_route;
   END IF;
END PROCESS;

------------------------------
receive_buffer: PROCESS(validDataIn, dataIn)
------------------------------
BEGIN
   -- Adiciono no Buffer de Entrada que recebe demais tipos de pacotes
   sig_buf_dataIn <= dataIn;
   sig_buf_push <= validDataIn;
END PROCESS;

-----------------------------------
route: PROCESS(current_state_route,
               sig_buf_canPop, sig_buf_dataOut, sig_buf_requesting,
               addressRPU, header,
               sig_rpu_receiver, sig_rpu_sender, sig_reroute,
               sig_xLocal, sig_yLocal,
               sig_xSend, sig_ySend,
               cb_ackreq,
               pcknumber, pckpointer,
               cb_buf_pop, sig_direction)
-----------------------------------

VARIABLE var_pop : STD_LOGIC;
VARIABLE var_rpu_sender_temp : STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0);

VARIABLE var_diff_x : INTEGER;
VARIABLE var_diff_y : INTEGER;
VARIABLE var_direction : INTEGER;

VARIABLE var_req : STD_LOGIC;
```

```
VARIABLE var_validData : STD_LOGIC;

BEGIN
   var_pop := '0';
   var_req := '0';
   var_validData := '0';
   var_direction := sig_direction;

   CASE current_state_route IS
      WHEN rHEAD =>
         IF(sig_buf_canPop = '1') THEN
            IF (sig_buf_dataOut(B_CONTROL DOWNTO E_CONTROL) = CTRL_HEADER)
THEN
               header <= sig_buf_dataOut;
               var_pop := '1';

               var_direction := UNRESERVED; -- Limpar lixo do crossbar

               sig_rpu_receiver <= sig_buf_dataOut(B_TARGET DOWNTO E_TARGET);
-- Quem vai receber
               var_rpu_sender_temp := sig_buf_dataOut(B_SOURCE DOWNTO
E_SOURCE); -- Quem enviou
               sig_reroute <= sig_buf_dataOut(B_REROUTE DOWNTO E_REROUTE);

               sig_xLocal <= to_integer(unsigned(addressRPU(ADDR-1 DOWNTO
ADDR/2))); -- Coordenadas Locais/Atual
               sig_yLocal <= to_integer(unsigned(addressRPU((ADDR/2)-1 DOWNTO
0)));
               sig_xSend <= to_integer(unsigned(var_rpu_sender_temp(ADDR-1
DOWNTO ADDR/2))); -- Quem enviou
               sig_ySend <= to_integer(unsigned(var_rpu_sender_temp((ADDR/2)-1
DOWNTO 0)));

               sig_rpu_sender <= var_rpu_sender_temp;

               CASE sig_buf_dataOut(B_TYPE DOWNTO E_TYPE) IS
                  WHEN PKT_REGULAR =>
                     next_state_route <= rPCKNUMBER;
                  WHEN PKT_CONTROL =>
                     -- TODO: next_state_route <= rINSTRUCTION_CONTROL; --
Recebo e começo a transmitir!
                  WHEN PKT_INTERRUPTED | PKT_CALLER =>
                     next_state_route <= rPCKNUMBER;
                  WHEN OTHERS =>
                     next_state_route <= rHEAD;
               END CASE;
            ELSE
               next_state_route <= rHEAD;
```

```
            END IF;
         ELSE
            next_state_route <= rHEAD;
         END IF;
      WHEN rPCKNUMBER =>
         IF(sig_buf_canPop = '1') THEN
            pcknumber <= sig_buf_dataOut;
            var_pop := '1';

            next_state_route <= rPCKPOINTER;
         ELSE
            next_state_route <= rPCKNUMBER;
         END IF;
      WHEN rPCKPOINTER =>
         IF(sig_buf_canPop = '1') THEN
            pckpointer <= sig_buf_dataOut;
            var_pop := '1';

            IF((header(B_TYPE DOWNTO E_TYPE) = PKT_INTERRUPTED) OR
               (header(B_TYPE DOWNTO E_TYPE) = PKT_CALLER)) THEN
               next_state_route <= rROUTE;
            ELSE
               next_state_route <= rINSTRUCTION;
            END IF;
         ELSE
            next_state_route <= rPCKPOINTER;
         END IF;
      WHEN rINSTRUCTION =>
            IF(sig_buf_dataOut(B_CONTROL DOWNTO E_CONTROL) = CTRL_INSTRUCTION)
THEN
               IF(to_integer(unsigned(sig_buf_dataOut(B_INSTRUCTION DOWNTO
E_INSTRUCTION))) = INN) THEN -- instrucao de entrada interrompe o pacote
regular
                  -- Calcular o endereco da MAU mais proxima
                  -- TODO: sig_rpu_receiver = nearestMau();
                  sig_rpu_sender <= addressRPU;
               ELSE
                  IF(sig_rpu_receiver = addressRPU) THEN
                     IF( ((sig_xLocal = 0) OR (sig_xLocal = SIDEE-1)) AND
                         ((sig_yLocal = 0) OR (sig_yLocal = SIDEE-1)) ) THEN -
- located in one corner -- SIDEE = tamanho da matriz

                        IF( (sig_xSend = SIDEE-1-sig_xLocal) AND (sig_ySend =
SIDEE-1-sig_yLocal) ) THEN -- source is the complement
                           -- TODO: sig_rpu_receiver = calcDestiny();
                        ELSE
                           IF((sig_xLocal = 0) AND (sig_yLocal = 0)) THEN --
tipo 1
                              sig_reroute <= SPIRAL_1; -- bits 000010;
```

```
                           ELSIF((sig_xLocal = SIDEE-1) AND (sig_yLocal = 0))
THEN -- tipo 4
                              sig_reroute <= SPIRAL_2; -- bits 000111;
                           ELSIF((sig_xLocal = SIDEE-1) AND (sig_yLocal =
SIDEE-1)) THEN -- tipo 3
                              sig_reroute <= SPIRAL_3; -- bits 000001;
                           ELSIF((sig_xLocal = 0) AND (sig_yLocal = SIDEE-1))
THEN -- tipo 2
                              sig_reroute <= SPIRAL_4; -- bits 000100;
                           END IF;

                           sig_rpu_receiver(ADDR-1 DOWNTO ADDR/2) <=
std_logic_vector(to_unsigned(SIDEE-1-sig_xLocal, LEN_TARGET/2)); -- xd
                           sig_rpu_receiver((ADDR/2)-1 DOWNTO 0) <=
std_logic_vector(to_unsigned(SIDEE-1-sig_yLocal, LEN_TARGET/2)); -- yd
                        END IF;
                     ELSE
                        -- TODO: sig_rpu_receiver = calcDestiny();
                     END IF;

                     sig_rpu_sender <= addressRPU;
                  END IF;
               END IF;

               next_state_route <= rROUTE;
            ELSE
               next_state_route <= rErrState;
            END IF;
      WHEN rROUTE =>
         -- Enquanto que o pacote regular calcula o reroute e a rpu destino.
         -- Estado para calcular a direção (N,S,E,W, MOD) por onde o pacote
será roteado.

         var_diff_x := (to_integer(unsigned(sig_rpu_receiver(ADDR-1 DOWNTO
ADDR/2))) - sig_xLocal); -- xo = Quem vai receber
         var_diff_y := (to_integer(unsigned(sig_rpu_receiver(ADDR/2-1 DOWNTO
0))) - sig_yLocal);    -- yo = Quem vai receber

         -- traditional routing XY
         IF (var_diff_y /= 0) THEN -- diff_y = 0: Significa que está na mesma
coluna
            IF(var_diff_y > 0) THEN
               var_direction := P_EAST;
            ELSE
               var_direction := P_WEST;
            END IF;
         ELSE
            IF(var_diff_x /= 0) THEN -- diff_x = 0: Significa que está na
mesma linha
```

```
                IF(var_diff_x > 0) THEN
                   var_direction := P_SOUTH;
                ELSE
                   var_direction := P_NORTH;
                END IF;
             ELSE
                -- direction = P_MOD; // Buffer Local
                -- Must be sent to MAU...
                IF(sig_yLocal = 0) THEN
                   var_direction := P_WEST;
                ELSIF(sig_yLocal = SIDEE-1) THEN
                   var_direction := P_EAST;
                END IF;
             END IF;
          END IF;

          -- Faz as modificações finais de roteamento antes de enviar ao
árbitro
          header(B_TARGET DOWNTO E_TARGET) <= sig_rpu_receiver;
          header(B_SOURCE DOWNTO E_SOURCE) <= sig_rpu_sender;
          header(B_REROUTE DOWNTO E_REROUTE) <= sig_reroute;

          -- Flag dizendo que terminou
          var_req := '1';

          cb_header <= header;
          cb_header(B_TARGET DOWNTO E_TARGET) <= sig_rpu_receiver;
          cb_header(B_SOURCE DOWNTO E_SOURCE) <= sig_rpu_sender;
          cb_header(B_REROUTE DOWNTO E_REROUTE) <= sig_reroute;
          cb_pcknumber <= pcknumber;
          cb_pckpointer <= pckpointer;

          next_state_route <= rWAIT_ACKREQ;
       WHEN rWAIT_ACKREQ =>
          IF (cb_ackreq = '1') THEN
             next_state_route <= rSEND_HEADER_TO_ARBITER;
          ELSE
             var_req := '1';

             next_state_route <= rWAIT_ACKREQ;
          END IF;
       WHEN rSEND_HEADER_TO_ARBITER =>
             var_validData := '1';

             next_state_route <= rSEND_TO_ARBITER;

       WHEN rSEND_TO_ARBITER =>
          IF(cb_buf_pop = '1') THEN
             var_pop := '1';
```

```
            IF(sig_buf_dataOut(B_CONTROL DOWNTO E_CONTROL) = CTRL_END) THEN
               cb_header        <= (OTHERS => '0');
               cb_pcknumber     <= (OTHERS => '0');
               cb_pckpointer    <= (OTHERS => '0');
               header           <= (OTHERS => '0');
               pcknumber        <= (OTHERS => '0');
               pckpointer       <= (OTHERS => '0');
               sig_rpu_receiver <= (OTHERS => '0');
               sig_rpu_sender   <= (OTHERS => '0');
               sig_reroute      <= (OTHERS => '0');
               next_state_route <= rEND;
            ELSE
               next_state_route <= rSEND_TO_ARBITER;
            END IF;
         ELSE
            -- Dado não é válido!
            next_state_route <= rSEND_TO_ARBITER;
         END IF;
      WHEN rEND =>
         next_state_route <= rHEAD;

      WHEN rErrState =>
         next_state_route <= rHEAD;
      WHEN OTHERS =>
         next_state_route <= rHEAD;
   END CASE;

   sig_route_pop <= var_pop;
   cb_req <= var_req;
   sig_direction <= var_direction;
   cb_validData <= var_validData;
END PROCESS;

END behaviour;
```

- *Buffer*

```
--------------------------------------------------------------------------------
-- PROJECT: IPNoSys
-- ENTITY : buffer_fifo
--------------------------------------------------------------------------------
-- DESCRIPTION: A FIFO entity offering for alternatives of implementation:
-- (a) RING:   a ring  FIFO architecture based on logic and flip-flops
--------------------------------------------------------------------------------
-- AUTHORS: Frederico G. M. do Espirito Santo
--          Cesar Albenes Zeferino
-- CONTACT: zeferino@univali.br OR cesar.zeferino@gmail.com
```

```
--------------------------------------------------------------------------------
LIBRARY ieee, lpm, work;
USE lpm.lpm_components.all;
USE ieee.std_logic_1164.all;
USE work.ipnosys_package.all;

--------------
--------------
ENTITY buffer_fifo IS
--------------
--------------
  PORT(
    -- System signals
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    -- FIFO interface
   data_in     : IN  STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);  -- input  data
channel
   can_push    : OUT STD_LOGIC;  -- FIFO has room to be written (not full)
   can_pop     : OUT STD_LOGIC;  -- FIFO has a data to be read  (not empty)
   push        : IN  STD_LOGIC;  -- command to write a data into de FIFO
   pop         : IN  STD_LOGIC;  -- command to read a data from the FIFO

    data_out    : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);   -- output data
channel
    --head        : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
  requesting  : OUT STD_LOGIC;  -- Header that just entered the buffer and
must be treated
  transmiting : OUT STD_LOGIC;   -- Packet is being transmitted and not yet
come to an end

  available_out : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0
  );
END buffer_fifo;

------------------------------
------------------------------
ARCHITECTURE behaviour OF buffer_fifo IS
------------------------------
------------------------------
SIGNAL used : INTEGER RANGE BUFFER_SIZE DOWNTO 0;         -- current FIFO
state
SIGNAL sig_data_out : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

BEGIN
-----------
-----------
fifo_ring :
```

```
-----------
-----------
  IF (FIFO_TYPE="RING") GENERATE
    ------------------
    C1: entity work.fifo_controller
    ------------------
      PORT MAP(
        rst   => rst,
        clk   => clk,
        push  => push,
        pop   => pop,
        used  => used
      );

    ----------------------
    C2: entity work.fifo_data_path
    ----------------------
      PORT MAP(
        clk         => clk,
        rst         => rst,
        push        => push,
        pop         => pop,
        data_in     => data_in,
        data_out    => data_out,
        --head        => head,
        can_push    => can_push,
        can_pop     => can_pop,
        requesting  => requesting,
        transmiting => transmiting,
        used        => used,
        available_out   => available_out
      );
  END GENERATE;
END behaviour;


--------------------------------------------------------------------------------
-- PROJECT: IPNoSys
-- ENTITY : fifo_controller
--------------------------------------------------------------------------------
-- DESCRIPTION: Control unit responsible to update the state of the FIFO at
-- each read or pushite operation.
--------------------------------------------------------------------------------
-- AUTHORS: Frederico G. M. do Espirito Santo
--          Cesar Albenes Zeferino
-- CONTACT: zeferino@univali.br OR cesar.zeferino@gmail.com
--------------------------------------------------------------------------------
LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
```

```
USE ieee.std_logic_arith.all;
USE work.ipnosys_package.all;
-------------------------
-------------------------
ENTITY fifo_controller IS
-------------------------
-------------------------
  PORT(
    -- System signals
    clk   : IN  STD_LOGIC;  -- clock
    rst   : IN  STD_LOGIC;  -- reset

    -- FIFO interface
    pop    : IN  STD_LOGIC;  -- command to read a data from the FIFO
    push   : IN  STD_LOGIC;  -- command to push a data into the FIFO

    -- Control interface
    used : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0      -- current FIFO state
  );
END fifo_controller;

----------------------------------------
----------------------------------------
ARCHITECTURE behaviour OF fifo_controller IS
----------------------------------------
----------------------------------------
SIGNAL current_state   : INTEGER RANGE BUFFER_SIZE DOWNTO 0; -- current state
SIGNAL next_state      : INTEGER RANGE BUFFER_SIZE DOWNTO 0; -- next state
BEGIN
  ------------- next state
  p_next_state:PROCESS(current_state,push,pop)
  ------------------------
  -- This process determines the next state of the FIFO taking into account
  -- the current state and the pushite and read commands, as follows:
  BEGIN
    -- FIFO EMPTY
    IF (current_state = 0) THEN
      IF (push='1') THEN
        next_state <= current_state+1;
      ELSE
        next_state <= current_state;
      END IF;

    -- FIFO FULL
    ELSIF (current_state = BUFFER_SIZE) THEN
      IF (pop='1') THEN
        next_state <= current_state-1;
      ELSE
        next_state <= current_state;
```

```
      END IF;

    -- FIFO NEITHER EMPTY, NEITHER FULL
    ELSE
      IF (push='1') THEN
        IF (pop='1') THEN
          next_state <= current_state;    --  pop &  push
        ELSE
          next_state <= current_state+1;  -- /pop &  push
        END IF;
      ELSIF (pop='1') THEN
        next_state <= current_state-1;    --  pop & /push
      ELSE
        next_state <= current_state;      -- /pop & /push
      END IF;
    END IF;
  END PROCESS p_next_state;

  -- current state
  p_current_state:PROCESS(clk,rst)
  ----------------
  BEGIN
    IF (rst='1') THEN
      current_state  <= 0;
    ELSIF (clk'EVENT AND clk='1') THEN
      current_state  <= next_state;
    END IF;
  END PROCESS p_current_state;

  ---------
  -- OUTPUT
  ---------
  used <= current_state;
END behaviour;


--------------------------------------------------------------------------------
-- PROJECT: IPNoSys
-- ENTITY : fifo_data_path (ring)
--------------------------------------------------------------------------------
-- DESCRIPTION: A datapath for the FIFO based on a ring of register with DEPTH
-- cells, each one with WIDTH bits. A new data is written into a position
-- defined by a write pointer. In the same way, the cell to be accessed during
-- a reading is defined by a read pointer. Such pointers are updated at a
-- writing or a reading.
--------------------------------------------------------------------------------
-- AUTHORS: Frederico G. M. do Espirito Santo
--          Cesar Albenes Zeferino
-- CONTACT: zeferino@univali.br OR cesar.zeferino@gmail.com
```

```
--------------------------------------------------------------------------------
LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE work.ipnosys_package.all;
----------------------------
----------------------------
ENTITY fifo_data_path IS
----------------------------
----------------------------
  PORT(
    -- System signals
    clk   : IN  STD_LOGIC;  -- clock
    rst   : IN  STD_LOGIC;  -- reset

    -- FIFO interface
    can_pop   : OUT STD_LOGIC;  -- FIFO has a data to be read  (not empty)
    can_push  : OUT STD_LOGIC;  -- FIFO has room to be written (not full)
    pop       : IN  STD_LOGIC;  -- command to read a data from the FIFO
    push      : IN  STD_LOGIC;  -- command to write a data into de FIFO
    data_in   : IN  STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);  -- input  data
channel
    data_out  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);  -- output data
channel
    --head      : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    requesting  : OUT STD_LOGIC;  -- Header that just entered the buffer and
must be treated
   transmiting : OUT STD_LOGIC;   -- Packet is being transmitted and not yet
come to an end

    -- Control to datapath interface
    used : IN  INTEGER RANGE BUFFER_SIZE DOWNTO 0;  -- current FIFO state
    available_out : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0);
END fifo_data_path;

-------------------------------------------
-------------------------------------------
ARCHITECTURE behaviour OF fifo_data_path IS
-------------------------------------------
-------------------------------------------
-- Type and signal to implement the FIFO
TYPE FIFO_TYPE IS ARRAY (BUFFER_SIZE-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL fifo          : FIFO_TYPE := (OTHERS => (OTHERS => '0'));

-- Pointers
SIGNAL pop_ptr_reg   : INTEGER RANGE BUFFER_SIZE-1 DOWNTO 0; -- read  pointer
SIGNAL push_ptr_reg  : INTEGER RANGE BUFFER_SIZE-1 DOWNTO 0; -- write pointer
```

```
SIGNAL next_pop_ptr  : INTEGER RANGE BUFFER_SIZE-1 DOWNTO 0; -- next read
pointer
SIGNAL next_push_ptr : INTEGER RANGE BUFFER_SIZE-1 DOWNTO 0; -- next write
pointer
SIGNAL sig_data_out  : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_req     : STD_LOGIC := '0';
SIGNAL sig_trans   : STD_LOGIC := '0';

BEGIN
----------- next write pointer
p_next_push_ptr:PROCESS(used, push, push_ptr_reg)
------------------------------
  -- Determines the next write pointer, by incrementing the current one
  -- when the FIFO is not full and there is a writing into the FIFO.
  BEGIN
    IF ((used/=(BUFFER_SIZE)) AND (push='1')) THEN
      IF (push_ptr_reg=(BUFFER_SIZE-1)) THEN
        next_push_ptr <= 0;
      ELSE
        next_push_ptr <= push_ptr_reg+1;
      END IF;
    ELSE
      next_push_ptr <= push_ptr_reg;
    END IF;
  END PROCESS p_next_push_ptr;

----------- next read pointer
p_next_pop_ptr:PROCESS(used, pop, pop_ptr_reg)
------------------------------
  -- Determines the next read pointer, by decrementing the current one
  -- when the FIFO is not empty and there is a reading from the FIFO.
  BEGIN
    IF ((used/=0) AND (pop='1')) THEN
      IF (pop_ptr_reg=(BUFFER_SIZE-1)) THEN
        next_pop_ptr <= 0;
      ELSE
        next_pop_ptr <= pop_ptr_reg+1;
      END IF;
    ELSE
      next_pop_ptr <= pop_ptr_reg;
    END IF;
  END PROCESS p_next_pop_ptr;

------ pointers
p_reg:PROCESS(clk,rst)
----------------
  -- Implements the pointer registers
  BEGIN
    IF (rst='1') THEN
```

```
      push_ptr_reg <= 0;
      pop_ptr_reg <= 0;
    ELSIF (clk'EVENT AND clk='1') THEN
      push_ptr_reg <= next_push_ptr;
      pop_ptr_reg <= next_pop_ptr;
    END IF;
  END PROCESS p_reg;

---- push2fifo
PROCESS(clk)
------------
  -- Implements the FIFO memory
  VARIABLE index  : INTEGER RANGE BUFFER_SIZE-1 DOWNTO 0;
  BEGIN
    IF (clk'EVENT AND clk='1') THEN
      IF ((push='1') AND (used/=BUFFER_SIZE)) THEN
        FOR index IN 0 TO (BUFFER_SIZE-1) LOOP
          IF (index=push_ptr_reg) THEN
            fifo(index) <= data_in;
          ELSE
            fifo(index) <= fifo(index);
          END IF;
        END LOOP;
      END IF;
    END IF;
  END PROCESS;

----- data
-- OUTPUTS
----------
  --data_out <= fifo(pop_ptr_reg) WHEN ((pop = '1') AND (clk'EVENT AND
clk='1'));
  sig_data_out <= fifo(pop_ptr_reg); -- WHEN ((pop = '1') AND (clk'EVENT AND
clk='1')) ELSE
             --sig_data_out;

  --data_out <= sig_data_out;
  data_out <= fifo(pop_ptr_reg);
  --head <= fifo(pop_ptr_reg);

  requesting <= sig_req;
  transmiting <= sig_trans;

-------- status
PROCESS (used)
---------------
  BEGIN
    IF (used=0) THEN
      can_pop <= '0';
```

```
      can_push <= '1';
      available_out <= BUFFER_SIZE;
    ELSIF (used=(BUFFER_SIZE)) THEN
      can_pop <= '1';
      can_push <= '0';
      available_out <= 0;
    ELSE
      can_pop <= '1';
      can_push <= '1';
      available_out <= BUFFER_SIZE - used;
    END IF;

  END PROCESS;

------------
PROCESS(push, pop, data_in, sig_data_out, used)
------------
VARIABLE var_req   : STD_LOGIC;
VARIABLE var_trans : STD_LOGIC;
  -- Treat header and end
  BEGIN

  var_req := '0';
  var_trans := '0';

  IF (push = '1') THEN
    IF (data_in(B_CONTROL DOWNTO E_CONTROL) = CTRL_HEADER) THEN
      var_req := '1';
    END IF;
  END IF;

  IF (pop = '1') THEN
    IF (sig_data_out(B_CONTROL DOWNTO E_CONTROL) = CTRL_HEADER) THEN
      var_trans := '1';
    ELSIF (sig_data_out(B_CONTROL DOWNTO E_CONTROL) = CTRL_END) THEN
      var_trans := '0';
      IF (used > 1) THEN -- Na verdade, deveria ser 0. Por causa do atraso, o
sinal permanece em '1'.
        var_req := '1';
      ELSE
        var_req := '0';
      END IF;
    END IF;
  END IF;

  sig_req <= var_req;
  sig_trans <= var_trans;

END PROCESS;
```

```
END behaviour;
```

- *Crossbar*

```
LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE work.ipnosys_package.all;
--------------
ENTITY cb IS
--------------
  PORT(
    -- System signals
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    dataIn_local : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_north : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    headerIn_north, pcknumberIn_north, pckpointerIn_north : IN
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_south : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    headerIn_south, pcknumberIn_south, pckpointerIn_south : IN
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_east  : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    headerIn_east, pcknumberIn_east, pckpointerIn_east : IN
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_west  : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    headerIn_west, pcknumberIn_west, pckpointerIn_west : IN
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

    validDataIn  : IN STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);

    canPop_In   : IN STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);
    buf_pop_Out : OUT STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);

    dataOut_north : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    headerOut_north, pcknumberOut_north, pckpointerOut_north : OUT
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataOut_south : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    headerOut_south, pcknumberOut_south, pckpointerOut_south : OUT
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataOut_east  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    headerOut_east, pcknumberOut_east, pckpointerOut_east : OUT
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataOut_west  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    headerOut_west, pcknumberOut_west, pckpointerOut_west : OUT
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
```

```
    validDataOut : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

    canPop_Out  : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    buf_pop_In  : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

    request          : IN STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);

    desired_port_local : IN INTEGER RANGE ROUTER_PORTS DOWNTO 0;
    desired_port_north : IN INTEGER RANGE ROUTER_PORTS DOWNTO 0;
    desired_port_south : IN INTEGER RANGE ROUTER_PORTS DOWNTO 0;
    desired_port_east  : IN INTEGER RANGE ROUTER_PORTS DOWNTO 0;
    desired_port_west  : IN INTEGER RANGE ROUTER_PORTS DOWNTO 0;

    ackRequest       : OUT STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0)
    );
END cb;

-------------------------------
ARCHITECTURE behaviour OF cb IS
-------------------------------

------------
--SIGNALS
------------

TYPE state_lock IS (s_idle,lock);       -- State declaration
SIGNAL current_state_lock, next_state_lock : state_lock;  -- State container

TYPE state_unlock IS (u_idle,unlock);       -- State declaration
SIGNAL current_state_unlock, next_state_unlock : state_unlock;  -- State
container

CONSTANT UNRESERVED_CB : INTEGER := 5;

TYPE CB_Ports_IN IS ARRAY (ROUTER_PORTS DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
TYPE CB_Ports_OUT IS ARRAY (ROUTER_PORTS-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
TYPE CB_ReservedPorts IS ARRAY (ROUTER_PORTS-1 DOWNTO 0) OF INTEGER RANGE
ROUTER_PORTS+1 DOWNTO 0;

SIGNAL sig_dataOutPorts : CB_Ports_OUT;
SIGNAL sig_headerOutPorts, sig_pcknumberOutPorts, sig_pckpointerOutPorts  :
CB_Ports_OUT;
SIGNAL reserved : CB_ReservedPorts := (UNRESERVED_CB, UNRESERVED_CB,
UNRESERVED_CB, UNRESERVED_CB); -- Inicialização com 5
SIGNAL sig_unlock : STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0) := (OTHERS =>
'0');
```

```
SIGNAL sig_lock : STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0) := (OTHERS => '0');

SIGNAL sig_ports_unlock : STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0) := (OTHERS
=> '0');

-------------------------------------------------------------
-- São 4 componentes que recebe 5 sinais.
TYPE CB_ReqPorts IS ARRAY (ROUTER_PORTS-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);
SIGNAL sig_req : CB_ReqPorts := ((OTHERS => '0'), (OTHERS => '0'), (OTHERS =>
'0'), (OTHERS => '0'));
SIGNAL sig_ack : CB_ReqPorts := ((OTHERS => '0'), (OTHERS => '0'), (OTHERS =>
'0'), (OTHERS => '0'));
SIGNAL sig_done : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0) := (OTHERS =>
'0');

-------------------------------------------------------------

BEGIN

   -- São 4 componentes que recebe 5 sinais.
   forCB: FOR i IN 0 TO ROUTER_PORTS-1 GENERATE
      compCB: entity work.round_robin
            GENERIC MAP(ROUTER_PORTS+1)
            PORT MAP (rst, clk, sig_req(i), sig_done(i), sig_ack(i));
   END GENERATE;

--------------------------
PROCESS(desired_port_local, desired_port_north, desired_port_south,
      desired_port_east, desired_port_west,
      request, sig_ack, reserved)
--------------------------
BEGIN

   -- No índice do reservado (porta de saída), escreva a porta de entrada

   FOR i IN 0 TO ROUTER_PORTS-1 LOOP
      IF (desired_port_north = i) THEN
         sig_req(i)(P_NORTH) <= request(P_NORTH);
         ackRequest(P_NORTH) <= sig_ack(i)(P_NORTH);

         IF(reserved(i) = UNRESERVED_CB AND sig_ack(i)(P_NORTH) = '1') THEN
            sig_lock(P_NORTH) <= '1';
         ELSE
            sig_lock(P_NORTH) <= '0';
         END IF;
      ELSE
         sig_req(i)(P_NORTH) <= '0';
      END IF;
```

```
------------------------------
IF(desired_port_south = i) THEN
   sig_req(i)(P_SOUTH) <= request(P_SOUTH);
   ackRequest(P_SOUTH) <= sig_ack(i)(P_SOUTH);

   IF(reserved(i) = UNRESERVED_CB AND sig_ack(i)(P_SOUTH) = '1') THEN
      sig_lock(P_SOUTH) <= '1';
   ELSE
      sig_lock(P_SOUTH) <= '0';
   END IF;
ELSE
   sig_req(i)(P_SOUTH) <= '0';
END IF;
------------------------------
IF(desired_port_east = i) THEN
   sig_req(i)(P_EAST) <= request(P_EAST);
   ackRequest(P_EAST) <= sig_ack(i)(P_EAST);

   IF(reserved(i) = UNRESERVED_CB AND sig_ack(i)(P_EAST) = '1') THEN
      sig_lock(P_EAST) <= '1';
   ELSE
      sig_lock(P_EAST) <= '0';
   END IF;
ELSE
   sig_req(i)(P_EAST) <= '0';
END IF;
------------------------------
IF(desired_port_west = i) THEN
   sig_req(i)(P_WEST) <= request(P_WEST);
   ackRequest(P_WEST) <= sig_ack(i)(P_WEST);

   IF(reserved(i) = UNRESERVED_CB AND sig_ack(i)(P_WEST) = '1') THEN
      sig_lock(P_WEST) <= '1';
   ELSE
      sig_lock(P_WEST) <= '0';
   END IF;
ELSE
   sig_req(i)(P_WEST) <= '0';
END IF;
------------------------------
IF(desired_port_local = i) THEN
   sig_req(i)(P_MOD) <= request(P_MOD);
   ackRequest(P_MOD) <= sig_ack(i)(P_MOD);

   IF(reserved(i) = UNRESERVED_CB AND sig_ack(i)(P_MOD) = '1') THEN
      sig_lock(P_MOD) <= '1';
   ELSE
      sig_lock(P_MOD) <= '0';
   END IF;
```

```
      ELSE
         sig_req(i)(P_MOD) <= '0';
      END IF;
   END LOOP;

END PROCESS;

-----------------------------------
state_assignment: PROCESS (rst, clk)
-----------------------------------
BEGIN
   IF rst = '1' THEN -- asynchronous reset (active high)
      current_state_unlock <= u_idle;
   ELSIF clk'event and clk = '1' THEN  -- rising clock edge
      current_state_unlock <= next_state_unlock;
   END IF;
END PROCESS;

-------------------------------------------------
proc_unlock: PROCESS (current_state_unlock,
                  dataIn_north, dataIn_south, dataIn_east, dataIn_west,
dataIn_local,
                  buf_pop_In, sig_ports_unlock,
                  desired_port_north, desired_port_south,
                  desired_port_east, desired_port_west, desired_port_local,
                  reserved)
-------------------------------------------------

VARIABLE var_unlock : STD_LOGIC_VECTOR(ROUTER_PORTS DOWNTO 0);

BEGIN
   var_unlock := (OTHERS => '0');

   CASE current_state_unlock IS
      WHEN u_idle =>
            IF (desired_port_north /= UNRESERVED AND
               reserved(desired_port_north) = P_NORTH AND
               dataIn_north(B_CONTROL DOWNTO E_CONTROL) = CTRL_END AND
               buf_pop_In(desired_port_north) = '1') THEN -- Verifico se
pop='1' na porta que desejo enviar
               sig_ports_unlock(P_NORTH) <= '1';
               sig_done(desired_port_north) <= '1';
            END IF;

            IF (desired_port_south /= UNRESERVED AND
               reserved(desired_port_south) = P_SOUTH AND
               dataIn_south(B_CONTROL DOWNTO E_CONTROL) = CTRL_END AND
               buf_pop_In(desired_port_south) = '1') THEN
               sig_ports_unlock(P_SOUTH) <= '1';
```

```
                sig_done(desired_port_south) <= '1';
            END IF;

            IF (desired_port_east /= UNRESERVED AND
                reserved(desired_port_east) = P_EAST AND
                dataIn_east(B_CONTROL DOWNTO E_CONTROL) = CTRL_END AND
                buf_pop_In(desired_port_east) = '1') THEN
                sig_ports_unlock(P_EAST) <= '1';
                sig_done(desired_port_east) <= '1';
            END IF;

            IF (desired_port_west /= UNRESERVED AND
                reserved(desired_port_west) = P_WEST AND
                dataIn_west(B_CONTROL DOWNTO E_CONTROL) = CTRL_END AND
                buf_pop_In(desired_port_west) = '1') THEN
                sig_ports_unlock(P_WEST) <= '1';
                sig_done(desired_port_west) <= '1';
            END IF;

            IF (desired_port_local /= UNRESERVED AND
                reserved(desired_port_local) = P_MOD AND
                sig_dataOutPorts(desired_port_west)(B_CONTROL DOWNTO E_CONTROL)
= CTRL_END AND -- dataIn_local
                buf_pop_In(desired_port_local) = '1') THEN
                sig_ports_unlock(P_MOD) <= '1';
                sig_done(desired_port_local) <= '1';
            END IF;

            IF(sig_ports_unlock /= "00000") THEN
                next_state_unlock <= unlock;
            ELSE
                next_state_unlock <= u_idle;
            END IF;

        WHEN unlock =>
            IF (desired_port_north /= UNRESERVED AND
                sig_ports_unlock(P_NORTH) = '1') THEN
                var_unlock(P_NORTH) := '1';
                sig_ports_unlock(P_NORTH) <= '0';
                sig_done(desired_port_north) <= '0';
            END IF;

            IF (desired_port_south /= UNRESERVED AND
                sig_ports_unlock(P_SOUTH) = '1') THEN
                var_unlock(P_SOUTH) := '1';
                sig_ports_unlock(P_SOUTH) <= '0';
                sig_done(desired_port_south) <= '0';
            END IF;
```

```
         IF (desired_port_east /= UNRESERVED AND
            sig_ports_unlock(P_EAST) = '1') THEN
            var_unlock(P_EAST) := '1';
            sig_ports_unlock(P_EAST) <= '0';
            sig_done(desired_port_east) <= '0';
         END IF;

         IF (desired_port_west /= UNRESERVED AND
            sig_ports_unlock(P_WEST) = '1') THEN
            var_unlock(P_WEST) := '1';
            sig_ports_unlock(P_WEST) <= '0';
            sig_done(desired_port_west) <= '0';
         END IF;

         IF (desired_port_local /= UNRESERVED AND
            sig_ports_unlock(P_MOD) = '1') THEN
            var_unlock(P_MOD) := '1';
            sig_ports_unlock(P_MOD) <= '0';
            sig_done(desired_port_local) <= '0';
         END IF;

         next_state_unlock <= u_idle;
   END CASE;

   sig_unlock <= var_unlock;
END PROCESS;

-------------------------------------------
reservePorts : PROCESS (sig_lock, sig_unlock,
                        desired_port_north, desired_port_south,
                        desired_port_east, desired_port_west,
desired_port_local)
-------------------------------------------
BEGIN

   -- North
   IF(sig_lock(P_NORTH) = '1') THEN
      reserved(desired_port_north) <= P_NORTH;
   END IF;

   IF (sig_unlock(P_NORTH) = '1') THEN
      reserved(desired_port_north) <= UNRESERVED_CB;
   END IF;

   -- South
   IF(sig_lock(P_SOUTH) = '1') THEN
      reserved(desired_port_south) <= P_SOUTH;
   END IF;
```

```
   IF (sig_unlock(P_SOUTH) = '1') THEN
      reserved(desired_port_south) <= UNRESERVED_CB;
   END IF;

   -- East
   IF(sig_lock(P_EAST) = '1') THEN
      reserved(desired_port_east) <= P_EAST;
   END IF;

   IF (sig_unlock(P_EAST) = '1') THEN
      reserved(desired_port_east) <= UNRESERVED_CB;
   END IF;

   -- West
   IF(sig_lock(P_WEST) = '1') THEN
      reserved(desired_port_west) <= P_WEST;
   END IF;

   IF (sig_unlock(P_WEST) = '1') THEN
      reserved(desired_port_west) <= UNRESERVED_CB;
   END IF;

   -- Local
   IF(sig_lock(P_MOD) = '1') THEN
      reserved(desired_port_local) <= P_MOD;
   END IF;

   IF (sig_unlock(P_MOD) = '1') THEN
      reserved(desired_port_local) <= UNRESERVED_CB;
   END IF;

END PROCESS;

--------------------------
PROCESS(reserved,
       dataIn_north, dataIn_south, dataIn_east, dataIn_west, dataIn_local,
       headerIn_north, pcknumberIn_north, pckpointerIn_north,
       headerIn_south, pcknumberIn_south, pckpointerIn_south,
       headerIn_east, pcknumberIn_east, pckpointerIn_east,
       headerIn_west, pcknumberIn_west, pckpointerIn_west,
       canPop_In, buf_pop_In, sig_dataOutPorts, validDataIn)
--------------------------
BEGIN

   -- No índice do reservado (porta de saída), escreva a porta de entrada

   FOR i IN 0 TO ROUTER_PORTS-1 LOOP
      IF (reserved(i) = P_NORTH) THEN
         sig_dataOutPorts(i) <= dataIn_north;
```

```
            sig_headerOutPorts(i) <= headerIn_north;
            sig_pcknumberOutPorts(i) <= pcknumberIn_north;
            sig_pckpointerOutPorts(i) <= pckpointerIn_north;
            canPop_Out(i) <= canPop_In(P_NORTH);
            buf_pop_Out(P_NORTH) <= buf_pop_In(i);
            validDataOut(i) <= validDataIn(P_NORTH);
         ELSIF(reserved(i) = P_SOUTH) THEN
            sig_dataOutPorts(i) <= dataIn_south;
            sig_headerOutPorts(i) <= headerIn_south;
            sig_pcknumberOutPorts(i) <= pcknumberIn_south;
            sig_pckpointerOutPorts(i) <= pckpointerIn_south;
            canPop_Out(i) <= canPop_In(P_SOUTH);
            buf_pop_Out(P_SOUTH) <= buf_pop_In(i);
            validDataOut(i) <= validDataIn(P_SOUTH);
         ELSIF(reserved(i) = P_EAST) THEN
            sig_dataOutPorts(i) <= dataIn_east;
            sig_headerOutPorts(i) <= headerIn_east;
            sig_pcknumberOutPorts(i) <= pcknumberIn_east;
            sig_pckpointerOutPorts(i) <= pckpointerIn_east;
            canPop_Out(i) <= canPop_In(P_EAST);
            buf_pop_Out(P_EAST) <= buf_pop_In(i);
            validDataOut(i) <= validDataIn(P_EAST);
         ELSIF(reserved(i) = P_WEST) THEN
            sig_dataOutPorts(i) <= dataIn_west;
            sig_headerOutPorts(i) <= headerIn_west;
            sig_pcknumberOutPorts(i) <= pcknumberIn_west;
            sig_pckpointerOutPorts(i) <= pckpointerIn_west;
            canPop_Out(i) <= canPop_In(P_WEST);
            buf_pop_Out(P_WEST) <= buf_pop_In(i);
            validDataOut(i) <= validDataIn(P_WEST);
         ELSIF(reserved(i) = P_MOD) THEN
            sig_dataOutPorts(i) <= dataIn_local;
            --FIX-ME: Corrigir para LOCAL.
--          sig_headerOutPorts(i) <= headerIn_north;
--          sig_pcknumberOutPorts(i) <= pcknumberIn_north;
--          sig_pckpointerOutPorts(i) <= pckpointerIn_north;
--          validDataOut(i) <= validDataIn(P_NORTH);
         ELSE
            sig_dataOutPorts(i) <= (OTHERS => '0');
            sig_headerOutPorts(i) <= (OTHERS => '0');
            sig_pcknumberOutPorts(i) <= (OTHERS => '0');
            sig_pckpointerOutPorts(i) <= (OTHERS => '0');
            canPop_Out(i) <= '0';
            buf_pop_Out(i) <= '0';
            validDataOut(i) <= '0';
         END IF;
      END LOOP;

      buf_pop_Out(P_MOD) <= '0';
```

```
END PROCESS;

dataOut_north <= sig_dataOutPorts(P_NORTH);
dataOut_south <= sig_dataOutPorts(P_SOUTH);
dataOut_east  <= sig_dataOutPorts(P_EAST);
dataOut_west  <= sig_dataOutPorts(P_WEST);

headerOut_north <= sig_headerOutPorts(P_NORTH);
headerOut_south <= sig_headerOutPorts(P_SOUTH);
headerOut_east  <= sig_headerOutPorts(P_EAST);
headerOut_west  <= sig_headerOutPorts(P_WEST);

pcknumberOut_north <= sig_pcknumberOutPorts(P_NORTH);
pcknumberOut_south <= sig_pcknumberOutPorts(P_SOUTH);
pcknumberOut_east  <= sig_pcknumberOutPorts(P_EAST);
pcknumberOut_west  <= sig_pcknumberOutPorts(P_WEST);

pckpointerOut_north <= sig_pckpointerOutPorts(P_NORTH);
pckpointerOut_south <= sig_pckpointerOutPorts(P_SOUTH);
pckpointerOut_east  <= sig_pckpointerOutPorts(P_EAST);
pckpointerOut_west  <= sig_pckpointerOutPorts(P_WEST);

END behaviour;
```

- *Arbiter*

```
LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE work.ipnosys_package.all;
----------------------------
ENTITY arbiter IS
----------------------------
  PORT(
    -- System signals
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    -----------------
    -- RPU Ports
    -----------------
    -- TO RPU
    dataOut : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataOut : OUT STD_LOGIC;
    -- FROM RPU
    availableIn : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    addressRPU : IN STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0);
```

```
    -----------------
    -- Crossbar Ports
    -----------------
    -- TO Buffer
    cb_pop         : OUT  STD_LOGIC;
    -- FROM Buffer
    cb_canPop      : IN STD_LOGIC;
    cb_dataIn      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    cb_validData   : IN STD_LOGIC;
    cb_header, cb_pcknumber, cb_pckpointer : IN STD_LOGIC_VECTOR(LEN_WIDTH-1
DOWNTO 0);

    -------------
    -- SU Ports
    -------------
    -- TO SU
    su_dataOut      : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    su_validDataOut : OUT STD_LOGIC;
    su_request      : OUT STD_LOGIC;
    -- FROM SU
    su_availableIn : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    su_ackRequest  : IN STD_LOGIC;

    --------------
    -- ALU Ports
    --------------
    -- TO ALU
    alu_dataA     : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    alu_dataB     : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    alu_operation : OUT STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
    alu_validDataIn : OUT STD_LOGIC;
    alu_request   : OUT STD_LOGIC;
    -- FROM ALU
    alu_result      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    alu_exception   : IN STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
    alu_validResult : IN STD_LOGIC;
    alu_neg         : IN STD_LOGIC;
    alu_zero        : IN STD_LOGIC;
    alu_ackRequest  : IN STD_LOGIC);
END arbiter;

----------------------------------------
ARCHITECTURE behaviour OF arbiter IS
----------------------------------------

------------------------------
-- Buffer Results - Signals
------------------------------
```

```
SIGNAL sig_br_dataIn : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_br_addrWr : STD_LOGIC_VECTOR((LEN_RESULT*2)-1 DOWNTO 0);
SIGNAL sig_br_wr     : STD_LOGIC;
SIGNAL sig_br_addrRd : STD_LOGIC_VECTOR(LEN_RESULT-1 DOWNTO 0);
SIGNAL sig_br_rd     : STD_LOGIC;
SIGNAL sig_br_result : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_br_valid  : STD_LOGIC;


TYPE state_type_calculate IS (CALCULATE_INST, CALCULATE_INST_TYPE,
CALCULATE_DATAa,
                              CALCULATE_DATAb, CALCULATE_REQUEST,
CALCULATE_RESULT,
                              CALCULATE_RESULT_COPY, CALCULATE_DISCARD,
                              CALCULATE_CALL, CALCULATE_SU_REQUEST,
CALCULATE_SEND_DATA_SU,
                              CALCULATE_NOP,
                              CALCULATE_EXCEPTION, CALCULATE_ErrState);
SIGNAL current_state_calculate, next_state_calculate : state_type_calculate;

TYPE state_type_transmite IS (oHEAD, oPCKNUMBER, oPCKPOINTER,
                              oINST, oOPER, oEND, oErrState); -- o = out
SIGNAL current_state_transmite, next_state_transmite : state_type_transmite;


SIGNAL calculate, transmite : STD_LOGIC;
SIGNAL header, pcknumber, pckpointer, operand : STD_LOGIC_VECTOR(LEN_WIDTH-1
DOWNTO 0) := (OTHERS => '0');
SIGNAL address : STD_LOGIC_VECTOR((LEN_RESULT*2)-1 DOWNTO 0);
SIGNAL operation, operationOrig : STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO
0);
SIGNAL dataA, dataB : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

SIGNAL sig_receive_pop : STD_LOGIC := '0';
SIGNAL sig_calculate_pop : STD_LOGIC := '0';
SIGNAL sig_transmite_pop : STD_LOGIC := '0';
SIGNAL sig_receive_br_rd : STD_LOGIC;
SIGNAL sig_calculate_br_rd : STD_LOGIC;
SIGNAL sig_transmite_br_rd : STD_LOGIC;

SIGNAL sig_count_pointer, sig_count_pointer_reset : STD_LOGIC;

SIGNAL sig_count_oper : STD_LOGIC;
SIGNAL sig_count_oper_reset : STD_LOGIC;

SIGNAL sig_count_CalcInst_reset : STD_LOGIC;
SIGNAL sig_count_IPR_reset : STD_LOGIC;
```

```
SIGNAL sig_count_data_SU_reset : STD_LOGIC;

-----------
-- Fixed
-----------

SIGNAL num_header_words : INTEGER RANGE 3 DOWNTO 0 := 0;
SIGNAL num_inst_per_RPU : INTEGER RANGE LEN_INST_RPU DOWNTO 0 := 0;
SIGNAL num_pointer_words : INTEGER RANGE RESULT_SIZE DOWNTO 0 := 0;
SIGNAL num_inst : INTEGER RANGE LEN_N_INSTRUCTIONS DOWNTO 0 := 0;
SIGNAL num_oper_calculate_SU : INTEGER RANGE LEN_RESULT+1 DOWNTO 0 := 0; -- O
tamanho máximo deste campo é definido em Result_2.
SIGNAL num_oper_transmite : INTEGER RANGE LEN_N_OPERANDS DOWNTO 0 := 0;
SIGNAL num_discarded_words : INTEGER RANGE LEN_RESULT DOWNTO 0 := 0;

------------
-- Counts
------------

SIGNAL count_inst_per_RPU : INTEGER RANGE LEN_INST_RPU DOWNTO 0 := 0;
-- Útil para verificar se há dados válidos no Buffer Results
-- e para atualizar no PCKPOINTER o apontador (posição atual
-- da palavra) para que a próxima RPU continue o processamento.
-- O apontador indica em qual momento deve inserir algo.
SIGNAL count_pointer_words : INTEGER RANGE RESULT_SIZE DOWNTO 0 := 0;
-- Útil para atualizar no HEADER o número de instruções (totalF = totalI-
calculadas) contidas no pacote.
-- Diz se ha ou nao instrucoes no pacote (a serem calculadas).
SIGNAL count_calculated_inst : INTEGER RANGE LEN_N_INSTRUCTIONS DOWNTO 0 := 0;
-- Util para saber quantos operandos tem na instruçao para que possa dar o pop
corretamente do Buffer ou BResult.
SIGNAL count_oper : INTEGER RANGE LEN_N_OPERANDS DOWNTO 0 := 0;
SIGNAL count_data_SU : INTEGER RANGE LEN_RESULT+1 DOWNTO 0 := 0; -- O tamanho
máximo deste campo é definido em Result_2.

-------------------------------------------------
-- Sinais de Verificação de Etapas Concluídas.
-------------------------------------------------

SIGNAL sig_calculate_result_ok : STD_LOGIC;
SIGNAL sig_calculate_inst_oper_word_ok : STD_LOGIC;
SIGNAL sig_transmite_inst_oper_word_ok : STD_LOGIC;
SIGNAL sig_calculate_data_SU_ok : STD_LOGIC;

-----------------------------------------------------------------
-- Sinais para On/Off do Calculate e Transmite (usado nos processos).
-----------------------------------------------------------------

SIGNAL sig_recv_calculate_on,  sig_recv_transmite_on : STD_LOGIC;
```

```
SIGNAL sig_recv_calculate_off, sig_recv_transmite_off : STD_LOGIC;

SIGNAL sig_calc_calculate_on,  sig_calc_transmite_on : STD_LOGIC;
SIGNAL sig_calc_calculate_off, sig_calc_transmite_off : STD_LOGIC;

SIGNAL sig_trans_calculate_on,  sig_trans_transmite_on : STD_LOGIC;
SIGNAL sig_trans_calculate_off, sig_trans_transmite_off : STD_LOGIC;

BEGIN

cb_pop <= sig_receive_pop OR sig_calculate_pop OR sig_transmite_pop;
sig_br_rd <= sig_receive_br_rd OR sig_calculate_br_rd OR sig_transmite_br_rd;
sig_br_addrRd <= std_logic_vector(to_unsigned(count_pointer_words +
num_pointer_words, LEN_RESULT));
sig_count_pointer <= sig_calculate_inst_oper_word_ok OR
sig_transmite_inst_oper_word_ok; -- Incremento pois o dado pode ter vindo do
Buffer ou do BResults.

buf_results: entity work.buffer_results
   PORT MAP(clk, rst, sig_br_dataIn, sig_br_addrWr, sig_br_wr,
            sig_br_addrRd, sig_br_rd, sig_br_result, sig_br_valid);

countOper : entity work.counter
   GENERIC MAP (0)
   PORT MAP(clk, sig_count_oper_reset, sig_count_oper, '0', count_oper);
countCalculatedInst : entity work.counter
   GENERIC MAP (0)
   PORT MAP(clk, sig_count_CalcInst_reset, sig_calculate_result_ok, '0',
count_calculated_inst);
countIPR : entity work.counter
   GENERIC MAP (0)
   PORT MAP(clk, sig_count_IPR_reset, sig_calculate_result_ok, '0',
count_inst_per_RPU);
countDataSU : entity work.counter
   GENERIC MAP (0)
   PORT MAP(clk, sig_count_data_SU_reset, sig_calculate_data_SU_ok, '0',
count_data_SU);

-- Inst_Oper
count_pointer : entity work.counter
   GENERIC MAP (0)
   PORT MAP(clk, sig_count_pointer_reset, sig_count_pointer, '0',
count_pointer_words);

----------------
-- current state
----------------
PROCESS(clk,rst)
----------------
```

```
BEGIN

   IF (rst='1') THEN
      current_state_calculate <= CALCULATE_INST;
      current_state_transmite <= oHEAD;
   ELSIF (clk'EVENT AND clk='1') THEN
      current_state_calculate <= next_state_calculate;
      current_state_transmite <= next_state_transmite;
   END IF;

END PROCESS;

-------------------------------------------
OnOffCalcTrans : PROCESS (sig_recv_calculate_on,  sig_calc_calculate_on,
sig_trans_calculate_on,
                          sig_recv_calculate_off, sig_calc_calculate_off,
sig_trans_calculate_off,
                          sig_recv_transmite_on,  sig_calc_transmite_on,
sig_trans_transmite_on,
                          sig_recv_transmite_off, sig_calc_transmite_off,
sig_trans_transmite_off)
-------------------------------------------
BEGIN

   IF(sig_recv_calculate_on = '1' OR
      sig_calc_calculate_on = '1' OR
      sig_trans_calculate_on = '1') THEN

      calculate <= '1';
   END IF;

   IF(sig_recv_calculate_off = '1' OR
      sig_calc_calculate_off = '1' OR
      sig_trans_calculate_off = '1') THEN

      calculate <= '0';
   END IF;

   IF(sig_recv_transmite_on = '1' OR
      sig_calc_transmite_on = '1' OR
      sig_trans_transmite_on = '1') THEN

      transmite <= '1';
   END IF;

   IF(sig_recv_transmite_off = '1' OR
      sig_calc_transmite_off = '1' OR
      sig_trans_transmite_off = '1') THEN
```

```
      transmite <= '0';
   END IF;

END PROCESS;

------------------------------
receive_buffer: PROCESS(header,num_inst, cb_validData,
                        cb_header, cb_pcknumber, cb_pckpointer)
------------------------------
VARIABLE var_header, var_pcknumber, var_pckpointer :
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
VARIABLE var_num_inst : INTEGER RANGE LEN_N_INSTRUCTIONS DOWNTO 0;

VARIABLE var_calculate_on, var_transmite_on : STD_LOGIC;
VARIABLE var_calculate_off, var_transmite_off : STD_LOGIC;

BEGIN
   var_header := header;
   var_num_inst := num_inst;

   var_calculate_on := '0';
   var_calculate_off := '0';
   var_transmite_on := '0';
   var_transmite_off := '0';

         IF(cb_validData = '1') THEN
            IF (cb_header(B_CONTROL DOWNTO E_CONTROL) = CTRL_HEADER)
THEN

               var_header := cb_header;

               CASE cb_header(B_TYPE DOWNTO E_TYPE) IS
                  WHEN PKT_REGULAR =>
                     -- nº de instruçoes | Somente no Regular que calcula e
remove instruções
                     var_num_inst :=
(to_integer(unsigned(cb_header(B_N_INSTRUCTIONS DOWNTO E_N_INSTRUCTIONS))));
                     num_header_words <= 3;

                     IF(cb_pcknumber(B_CONTROL DOWNTO E_CONTROL) = CTRL_HEADER
AND
                        cb_pckpointer(B_CONTROL DOWNTO E_CONTROL) =
CTRL_HEADER) THEN

                        var_calculate_on := '1';
                        var_transmite_off := '1';

                        pcknumber <= cb_pcknumber;
                        pckpointer <= cb_pckpointer;
```

```
                        num_pointer_words <=
(to_integer(unsigned(cb_pckpointer(B_POINTER DOWNTO E_POINTER))));
                        num_inst_per_RPU <=
(to_integer(unsigned(cb_pckpointer(B_INST_RPU DOWNTO E_INST_RPU))));
                     END IF;
                  WHEN PKT_CONTROL =>
                     -- Os pacotes de controle apresentam no máximo uma
instrução a ser executada na MAU.
                     -- nº de instruçoes | Somente no Regular que calcula e
remove instruções
                     num_header_words <= 1;
                     num_inst_per_RPU <= 0;
                     var_calculate_off := '1';
                     var_transmite_on := '1';

                     pcknumber  <= (OTHERS => '0');
                     pckpointer <= (OTHERS => '0');
                  WHEN PKT_INTERRUPTED | PKT_CALLER =>
                     -- Os pacotes interrompidos e caller são pacote regulares
                     -- nº de instruçoes | Somente no Regular que calcula e
remove instruções
                     var_num_inst :=
(to_integer(unsigned(cb_header(B_N_INSTRUCTIONS DOWNTO E_N_INSTRUCTIONS))));

                     num_header_words <= 3;

                     IF(cb_pcknumber(B_CONTROL DOWNTO E_CONTROL) = CTRL_HEADER
AND
                        cb_pckpointer(B_CONTROL DOWNTO E_CONTROL) =
CTRL_HEADER) THEN
                        var_calculate_off := '1';
                        var_transmite_on := '1';

                        pcknumber <= cb_pcknumber;
                        pckpointer <= cb_pckpointer;
                        num_pointer_words <=
(to_integer(unsigned(cb_pckpointer(B_POINTER DOWNTO E_POINTER))));
                        num_inst_per_RPU <=
(to_integer(unsigned(cb_pckpointer(B_INST_RPU DOWNTO E_INST_RPU))));
                     END IF;
                  WHEN OTHERS =>
--                   next_state_receive <= iErrState;
               END CASE;
            END IF;
         END IF;

   header <= var_header;
   num_inst <= var_num_inst;
```

```
   sig_recv_calculate_on  <= var_calculate_on;
   sig_recv_calculate_off <= var_calculate_off;
   sig_recv_transmite_on  <= var_transmite_on;
   sig_recv_transmite_off <= var_transmite_off;
END PROCESS;

---------------------------------
alu_calculate: PROCESS(calculate, current_state_calculate, operation,
operationOrig,
                       operand, dataA, dataB, alu_result, address,
                       cb_canPop, cb_dataIn, sig_br_valid, sig_br_result,
                       alu_ackRequest, alu_zero, alu_neg, alu_exception,
                       pckpointer, count_pointer_words, num_discarded_words,
                       count_data_SU, num_oper_calculate_SU, pcknumber,
                       count_inst_per_RPU, num_inst_per_RPU,
                       su_ackRequest, su_availableIn)
---------------------------------
VARIABLE var_br_wr : STD_LOGIC;
VARIABLE var_calculate_word_ok   : STD_LOGIC;
VARIABLE var_calculate_result_ok : STD_LOGIC;
VARIABLE var_calculate_data_SU_ok, var_count_data_SU_reset : STD_LOGIC;
VARIABLE var_pop   : STD_LOGIC;
VARIABLE var_br_rd : STD_LOGIC;
VARIABLE var_num_oper_calculate_SU : INTEGER;

VARIABLE var_operation : STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
VARIABLE var_alu_request : STD_LOGIC;
VARIABLE var_dataA, var_dataB, var_result : STD_LOGIC_VECTOR(LEN_WIDTH-1
DOWNTO 0);

VARIABLE var_su_dataOut      : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
VARIABLE var_su_validDataOut : STD_LOGIC;
VARIABLE var_su_request      : STD_LOGIC;

VARIABLE var_calculate_on, var_transmite_on   : STD_LOGIC;
VARIABLE var_calculate_off, var_transmite_off : STD_LOGIC;

BEGIN
   var_br_wr := '0';
   var_calculate_word_ok := '0';
   var_calculate_result_ok := '0';
   var_calculate_data_SU_ok := '0';
   var_count_data_SU_reset := '0';
   var_pop := '0';
   var_br_rd := '0';
   var_num_oper_calculate_SU := 0;

   var_alu_request := '0';
   var_dataA := dataA;
```

```
   var_dataB := dataB;

   var_su_dataOut := (OTHERS => '0');
   var_su_validDataOut := '0';
   var_su_request := '0';

   var_calculate_on  := '0';
   var_calculate_off := '0';
   var_transmite_on  := '0';
   var_transmite_off := '0';

   IF(calculate = '1') THEN
      CASE current_state_calculate IS
         WHEN CALCULATE_INST =>
            IF(cb_canPop = '1') THEN
               IF (cb_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_INSTRUCTION)
THEN
                  IF(count_inst_per_RPU = num_inst_per_RPU) THEN
                     var_calculate_off := '1';
                     var_transmite_on  := '1';

                     var_dataA := (OTHERS => '0');
                     var_dataB := (OTHERS => '0');
                     var_operation := (OTHERS => '0');

                     next_state_calculate <= CALCULATE_INST;
                  ELSE
                     address <= cb_dataIn(B_RESULT_1 DOWNTO E_RESULT_2);
                     operation <= cb_dataIn(B_INSTRUCTION DOWNTO
E_INSTRUCTION);
                     operationOrig <= cb_dataIn(B_INSTRUCTION DOWNTO
E_INSTRUCTION);

                     next_state_calculate <= CALCULATE_INST_TYPE;
                  END IF;
               ELSIF(cb_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_END) THEN
                  var_calculate_off := '1';
                  var_transmite_on  := '1';

                  next_state_calculate <= CALCULATE_INST;
               ELSE
                  next_state_calculate <= CALCULATE_ErrState;
               END IF;
            ELSE
               next_state_calculate <= CALCULATE_INST;
            END IF;
         WHEN CALCULATE_INST_TYPE =>
            var_operation := operation;
```

```
            var_pop := '1';
            var_calculate_word_ok := '1';

            CASE (to_integer(unsigned(var_operation))) IS
               WHEN ADD | SUBB | MUL | DIV | RSHIFT | LSHIFT | ANDD | ORR |
XORR =>
                  operation <= var_operation;
                  next_state_calculate <= CALCULATE_DATAa;
               WHEN NOTT =>
                  operation <= var_operation;
                  next_state_calculate <= CALCULATE_DATAb;
               WHEN COPY =>
                  -- Não preciso solicitar nada a ALU.
                  -- Apenas armazeno no BR.
                  operation <= var_operation;
                  next_state_calculate <= CALCULATE_DATAb;
               WHEN BE | BNE | BL | BG | BLE | BGE =>
                  -- Operação é SUBB, pois estou comparando. Logo preciso
subtrair: A - B == 0
                  operation <= std_logic_vector(to_unsigned(SUBB,
LEN_INSTRUCTION));
                  next_state_calculate <= CALCULATE_DATAa;
               WHEN JUMP =>
                  -- Não preciso calcular nada. Só descartar as palavras.
                  num_discarded_words <=
to_integer(unsigned(cb_dataIn(B_RESULT_1 DOWNTO E_RESULT_1)));
                  next_state_calculate <= CALCULATE_DISCARD;
               WHEN NOP =>
                  var_calculate_result_ok := '1'; -- Incrementar numero de
instruções executadas.
                  next_state_calculate <= CALCULATE_NOP;
               WHEN CALL =>
                  next_state_calculate <= CALCULATE_CALL;
               WHEN OTHERS =>
                  -- Send instruction to SU that will create a control packet
                  -- Lógica do Request da SU.
                  var_su_request := '1';

                  next_state_calculate <= CALCULATE_SU_REQUEST;
            END CASE;
         WHEN CALCULATE_NOP =>
            next_state_calculate <= CALCULATE_INST;
         WHEN CALCULATE_DATAa =>
            -- Operando pode vir do Buffer Results ou do Buffer de Entrada.
            IF(sig_br_valid = '1') THEN
               IF (sig_br_result(B_CONTROL DOWNTO E_CONTROL) = CTRL_OPERAND)
THEN
                  var_dataA := sig_br_result;
                  var_br_rd := '1';
```

```
                  var_calculate_word_ok := '1';
                  next_state_calculate <= CALCULATE_DATAb;
               ELSE
                  next_state_calculate <= CALCULATE_ErrState;
               END IF;
            ELSIF (cb_canPop = '1') THEN
               IF (cb_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_OPERAND) THEN
                  var_dataA := cb_dataIn;
                  var_pop := '1';

                  var_calculate_word_ok := '1';
                  next_state_calculate <= CALCULATE_DATAb;
               ELSE
                  next_state_calculate <= CALCULATE_ErrState;
               END IF;
            ELSE
               next_state_calculate <= CALCULATE_DATAa;
            END IF;
         WHEN CALCULATE_DATAb =>
            IF(sig_br_valid = '1') THEN
               IF (sig_br_result(B_CONTROL DOWNTO E_CONTROL) = CTRL_OPERAND)
THEN
                  var_dataB := sig_br_result;
                  var_br_rd := '1';

                  var_calculate_word_ok := '1';
                  IF (to_integer(unsigned(operation)) = COPY) THEN
                     next_state_calculate <= CALCULATE_RESULT_COPY;
                  ELSE
                     var_alu_request := '1';
                     next_state_calculate <= CALCULATE_REQUEST;
                  END IF;
               ELSE
                  next_state_calculate <= CALCULATE_ErrState;
               END IF;
            ELSIF (cb_canPop = '1') THEN
               IF (cb_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_OPERAND) THEN
                  var_dataB := cb_dataIn;
                  var_pop := '1';

                  var_calculate_word_ok := '1';
                  IF (to_integer(unsigned(operation)) = COPY) THEN
                     next_state_calculate <= CALCULATE_RESULT_COPY;
                  ELSE
                     var_alu_request := '1';
                     next_state_calculate <= CALCULATE_REQUEST;
                  END IF;
               ELSE
```

```
                next_state_calculate <= CALCULATE_ErrState;
            END IF;
         ELSE
            next_state_calculate <= CALCULATE_DATAb;
         END IF;
      WHEN CALCULATE_REQUEST =>
         IF(alu_ackRequest = '1') THEN
            alu_dataA <= var_dataA;
            alu_dataB <= var_dataB;
            alu_operation <= operation;
            alu_validDataIn <= '1';

            next_state_calculate <= CALCULATE_RESULT;
         ELSE
            var_alu_request := '1';
            next_state_calculate <= CALCULATE_REQUEST;
         END IF;
      WHEN CALCULATE_RESULT =>
         IF (alu_validResult = '1') THEN
            alu_dataA <= (OTHERS => '0');
            alu_dataB <= (OTHERS => '0');
            alu_operation <= (OTHERS => '0');
            alu_validDataIn <= '0';

            -- Da ALU só será necessário o Zero e Neg
            CASE (to_integer(unsigned(operationOrig))) IS
               WHEN BE =>
                  IF (alu_zero = '1') THEN -- numbers are equals
                     num_discarded_words <=
to_integer(unsigned(address(B_RESULT_1 DOWNTO E_RESULT_1)));
                     next_state_calculate <= CALCULATE_DISCARD;
                  ELSE
                     var_calculate_result_ok := '1'; -- Informo ao processo
principal (clk e contador) que já tenho o resultado da operação.
                     next_state_calculate <= CALCULATE_INST;
                  END IF;
               WHEN BNE =>
                  IF (alu_zero = '0') THEN -- numbers are differents
                     num_discarded_words <=
to_integer(unsigned(address(B_RESULT_1 DOWNTO E_RESULT_1)));
                     next_state_calculate <= CALCULATE_DISCARD;
                  ELSE
                     var_calculate_result_ok := '1'; -- Informo ao processo
principal (clk e contador) que já tenho o resultado da operação.
                     next_state_calculate <= CALCULATE_INST;
                  END IF;
               WHEN BL =>
                  IF (alu_neg = '1') THEN -- less
```

```
                        num_discarded_words <=
to_integer(unsigned(address(B_RESULT_1 DOWNTO E_RESULT_1)));
                        next_state_calculate <= CALCULATE_DISCARD;
                     ELSE
                        var_calculate_result_ok := '1'; -- Informo ao processo
principal (clk e contador) que já tenho o resultado da operação.
                        next_state_calculate <= CALCULATE_INST;
                     END IF;
                  WHEN BG =>
                     IF (alu_neg = '0') THEN -- greater
                        num_discarded_words <=
to_integer(unsigned(address(B_RESULT_1 DOWNTO E_RESULT_1)));
                        next_state_calculate <= CALCULATE_DISCARD;
                     ELSE
                        var_calculate_result_ok := '1'; -- Informo ao processo
principal (clk e contador) que já tenho o resultado da operação.
                        next_state_calculate <= CALCULATE_INST;
                     END IF;
                  WHEN BLE =>
                     IF (alu_neg = '1' OR alu_zero = '1') THEN -- less or
equal
                        num_discarded_words <=
to_integer(unsigned(address(B_RESULT_1 DOWNTO E_RESULT_1)));
                        next_state_calculate <= CALCULATE_DISCARD;
                     ELSE
                        var_calculate_result_ok := '1'; -- Informo ao processo
principal (clk e contador) que já tenho o resultado da operação.
                        next_state_calculate <= CALCULATE_INST;
                     END IF;
                  WHEN BGE =>
                     IF (alu_neg = '0' OR alu_zero = '1') THEN -- greater or
equal
                        num_discarded_words <=
to_integer(unsigned(address(B_RESULT_1 DOWNTO E_RESULT_1)));
                        next_state_calculate <= CALCULATE_DISCARD;
                     ELSE
                        var_calculate_result_ok := '1'; -- Informo ao processo
principal (clk e contador) que já tenho o resultado da operação.
                        next_state_calculate <= CALCULATE_INST;
                     END IF;
                  WHEN OTHERS =>
                     IF(alu_exception = "00") THEN -- Se não houver exceção.
                        sig_br_dataIn <= alu_result;
                        sig_br_addrWr <= address; -- Apontador deve sempre
começar em 3
                        var_br_wr := '1';

                        var_calculate_result_ok := '1'; -- Informo ao processo
principal (clk e contador) que já tenho o resultado da operação.
```

```
                        next_state_calculate <= CALCULATE_INST;
                     ELSE
                        var_su_dataOut(B_CONTROL DOWNTO E_CONTROL) :=
CTRL_INSTRUCTION; -- instruction
                        var_su_dataOut(B_INSTRUCTION DOWNTO E_INSTRUCTION) :=
std_logic_vector(to_unsigned(NOTIFY, LEN_INSTRUCTION)); -- the process was
interrupted and store in this MAU
                        var_su_dataOut(B_N_OPERANDS DOWNTO E_N_OPERANDS) :=
std_logic_vector(to_unsigned(1, LEN_N_OPERANDS)); -- need 1 operand
                        var_su_dataOut(B_RESULT_1 DOWNTO E_RESULT_1) := "000"
& pckpointer(B_PROCESS_ADDR DOWNTO E_PROCESS_ADDR); -- MAU's address that keep
the process -- FIX-ME: A concatenação está correta?
                        var_su_dataOut(B_RESULT_2 DOWNTO E_RESULT_2) :=
std_logic_vector(to_unsigned(2, LEN_RESULT)); -- type 2 means the end of
process

                        var_su_validDataOut := '1';

                        next_state_calculate <=
CALCULATE_EXCEPTION;
                     END IF;
               END CASE;
            ELSE
               next_state_calculate <= CALCULATE_RESULT;
            END IF;
         WHEN CALCULATE_RESULT_COPY => -- Estado APENAS para Instrução COPY.
            sig_br_dataIn <= var_dataB;
            sig_br_addrWr <= address;
            var_br_wr := '1';

            var_calculate_result_ok := '1'; -- Informo ao processo principal
(clk e contador) que conte "Instrução++".
            next_state_calculate <= CALCULATE_INST;
         WHEN CALCULATE_DISCARD =>
            IF((count_pointer_words + num_pointer_words) =
num_discarded_words) THEN
               var_calculate_result_ok := '1'; -- Informo ao contador que
conte "Instrução++".
               next_state_calculate <= CALCULATE_INST;
            ELSE
               -- Descarto as palavras até o apontador alcançar
"num_discarded_words".
               -- Pois sempre será alguma posição à frente da atual.
               -- Quando acontecer, passo para outro estado.
               IF(sig_br_valid = '1') THEN
                  IF (sig_br_result(B_CONTROL DOWNTO E_CONTROL) =
CTRL_OPERAND) THEN
                     var_br_rd := '1';
```

```
                  var_calculate_word_ok := '1';
                  next_state_calculate <= CALCULATE_DISCARD;
               ELSE
                  next_state_calculate <= CALCULATE_ErrState;
               END IF;
            ELSIF (cb_canPop = '1') THEN
               var_pop := '1';

               var_calculate_word_ok := '1';
               next_state_calculate <= CALCULATE_DISCARD;
            ELSE
               -- Vou permanecer neste estado enquanto não tiver descartado
todas as palavras.
               next_state_calculate <= CALCULATE_DISCARD;
            END IF;
         END IF;
      WHEN CALCULATE_SU_REQUEST =>
         IF(su_ackRequest = '1') THEN
            var_num_oper_calculate_SU :=
to_integer(unsigned(cb_dataIn(B_N_OPERANDS DOWNTO E_N_OPERANDS)));

            --arguments (operands)
            IF(var_num_oper_calculate_SU = 3) THEN
               -- Quantidade de Operandos definida em Result_2
               var_num_oper_calculate_SU :=
(to_integer(unsigned(cb_dataIn(B_RESULT_2 DOWNTO E_RESULT_2)))) + 1;
            END IF;

            IF((to_integer(unsigned(var_operation))) = SYNC) THEN
               var_num_oper_calculate_SU := var_num_oper_calculate_SU + 2;
            ELSIF((to_integer(unsigned(var_operation))) = INN OR
(to_integer(unsigned(var_operation))) = OUTT) THEN
               var_num_oper_calculate_SU := var_num_oper_calculate_SU + 3;
            ELSE
               var_num_oper_calculate_SU := var_num_oper_calculate_SU + 1;
            END IF;

            next_state_calculate <= CALCULATE_SEND_DATA_SU;
         ELSE
            var_su_request := '1';
            next_state_calculate <= CALCULATE_SU_REQUEST;
         END IF;
      WHEN CALCULATE_SEND_DATA_SU =>
         IF (su_availableIn - 1 > 1) THEN
            IF(sig_br_valid = '1') THEN
               IF (sig_br_result(B_CONTROL DOWNTO E_CONTROL) =
CTRL_OPERAND) THEN
                  var_su_dataOut := sig_br_result;
                  var_su_validDataOut := '1';
```

```
                    var_br_rd := '1';

                    var_calculate_word_ok := '1';
                    var_calculate_data_SU_ok := '1';

                    next_state_calculate <= CALCULATE_SEND_DATA_SU;
                 ELSE
                    next_state_calculate <= CALCULATE_ErrState;
                 END IF;
              ELSE -- Se não estiver no Buffer Results:
                 IF(count_data_SU = num_oper_calculate_SU-1) THEN -- O último
parâmetro!
                    IF((to_integer(unsigned(operation))) = SYNC) THEN -- must
be include the number of current packet
                       var_su_dataOut(B_CONTROL DOWNTO E_CONTROL) :=
CTRL_OPERAND;
                       var_su_dataOut(B_PROCESS_NUM DOWNTO E_PROCESS_NUM) :=
pcknumber(B_PROCESS_NUM DOWNTO E_PROCESS_NUM);
                       var_su_dataOut(B_PACKET_NUM DOWNTO E_PACKET_NUM) :=
pcknumber(B_PACKET_NUM DOWNTO E_PACKET_NUM);
                       var_su_validDataOut := '1';

                       var_calculate_word_ok := '1';
                       var_calculate_data_SU_ok := '1';
                       var_count_data_SU_reset := '1';

                       next_state_calculate <= CALCULATE_INST;
                    ELSIF(((to_integer(unsigned(operation))) = INN) OR
((to_integer(unsigned(operation))) = OUTT)) THEN --must be include the number
of interrupted pac.
                       var_su_dataOut(B_CONTROL DOWNTO E_CONTROL) :=
CTRL_OPERAND;
                       var_su_dataOut(B_PROCESS_NUM DOWNTO E_PROCESS_NUM) :=
pcknumber(B_PROCESS_NUM DOWNTO E_PROCESS_NUM ); -- process
                       var_su_dataOut(B_PACKET_NUM DOWNTO E_PACKET_NUM) :=
pcknumber(B_PACKET_NUM DOWNTO E_PACKET_NUM ); -- interrupted pac
                       var_su_validDataOut := '1';

                       var_calculate_word_ok := '1';
                       var_calculate_data_SU_ok := '1';
                       var_count_data_SU_reset := '1';

                       next_state_calculate <= CALCULATE_INST;
                    ELSE
                       IF(cb_canPop = '1') THEN
                          var_su_dataOut := cb_dataIn;
                          var_su_validDataOut := '1';
                          var_pop := '1';
```

```
                        var_calculate_word_ok := '1';
                        var_calculate_data_SU_ok := '1';
                        var_count_data_SU_reset := '1';

                        next_state_calculate <= CALCULATE_INST;
                     ELSE
                        next_state_calculate <= CALCULATE_SEND_DATA_SU; --
Se não é possível fazer pop, aguarde...
                     END IF;
                  END IF;
               ELSIF(count_data_SU = num_oper_calculate_SU-2) THEN --
Penúltimo parâmetro!
                  IF((to_integer(unsigned(operation))) = INN) THEN --must
be include the MAU's address that store interrupted pac
                     var_su_dataOut(B_CONTROL DOWNTO E_CONTROL) :=
CTRL_OPERAND;
                     --var_su_dataOut(B_OPERAND DOWNTO E_OPERAND) :=
nearestMau(); -- TODO: nearst MAU's address
                     var_su_validDataOut := '1';

                     var_calculate_word_ok := '1';
                     var_calculate_data_SU_ok := '1';
                  ELSIF((to_integer(unsigned(operation))) = OUTT) THEN
                     var_su_dataOut(B_CONTROL DOWNTO E_CONTROL) :=
CTRL_OPERAND;
                     var_su_dataOut(B_OPERAND DOWNTO E_OPERAND) :=
std_logic_vector(to_unsigned((0), LEN_OPERAND));
                     var_su_validDataOut := '1';

                     var_calculate_word_ok := '1';
                     var_calculate_data_SU_ok := '1';
                  ELSE
                     IF(cb_canPop = '1') THEN
                        var_su_dataOut := cb_dataIn;
                        var_su_validDataOut := '1';
                        var_pop := '1';

                        var_calculate_word_ok := '1';
                        var_calculate_data_SU_ok := '1';
                     END IF;
                  END IF;

                  next_state_calculate <= CALCULATE_SEND_DATA_SU;

               ELSE
                  IF(cb_canPop = '1') THEN
                     var_su_dataOut := cb_dataIn;
                     var_su_validDataOut := '1';
                     var_pop := '1';
```

```
                        var_calculate_word_ok := '1';
                        var_calculate_data_SU_ok := '1';
                     END IF; -- End Checar Buffer de Entrada

                     next_state_calculate <= CALCULATE_SEND_DATA_SU;
                  END IF; -- End Iteração
               END IF; -- End Buffers

               IF(count_data_SU = 0) THEN -- Instruction word
                  IF((to_integer(unsigned(operation))) = SYNC) THEN --
std_logic_vector(to_unsigned((num_inst - count_calculated_inst),
LEN_N_INSTRUCTIONS));
                     var_su_dataOut(B_N_OPERANDS DOWNTO E_N_OPERANDS) :=
std_logic_vector(to_unsigned((to_integer(unsigned(var_su_dataOut(B_N_OPERANDS
DOWNTO E_N_OPERANDS)))) + 1, LEN_N_OPERANDS)); -- increment the number of
operands
                  ELSIF(((to_integer(unsigned(operation))) = INN) OR
((to_integer(unsigned(operation))) = OUTT)) THEN
                     var_su_dataOut(B_RESULT_2 DOWNTO E_RESULT_2) :=
std_logic_vector(to_unsigned((to_integer(unsigned(var_su_dataOut(B_RESULT_2
DOWNTO E_RESULT_2)))) + 2, LEN_RESULT)); -- MAU address and number of pac
                  END IF;
               END IF;

               -- TODO: Nas instrucoes SYNC, IN e OUT foram incluidas palavras
extras,
               -- as quais devem ser subtraidas do contador do pacote regular
do
               -- qual estam inseridas antes de serem enviadas para o SU
            ELSE
               next_state_calculate <= CALCULATE_SEND_DATA_SU;
            END IF;

         WHEN CALCULATE_ErrState =>
            next_state_calculate <= CALCULATE_INST;
         WHEN CALCULATE_EXCEPTION =>
            next_state_calculate <= CALCULATE_INST;
         WHEN OTHERS =>
            next_state_calculate <= CALCULATE_INST;
      END CASE;
   END IF; -- End of Calculate

   sig_calculate_pop <= var_pop;
   sig_calculate_br_rd <= var_br_rd;
   sig_br_wr <= var_br_wr;
   sig_calculate_inst_oper_word_ok <= var_calculate_word_ok;
   sig_calculate_result_ok <= var_calculate_result_ok;
   num_oper_calculate_SU <= var_num_oper_calculate_SU;
```

```
   sig_count_data_SU_reset <= var_count_data_SU_reset;

   sig_calculate_data_SU_ok <= var_calculate_data_SU_ok;
   alu_request <= var_alu_request;
   dataA <= var_dataA;
   dataB <= var_dataB;

   su_dataOut <= var_su_dataOut;
   su_validDataOut <= var_su_validDataOut;
   su_request <= var_su_request;

   sig_calc_calculate_on  <= var_calculate_on;
   sig_calc_calculate_off <= var_calculate_off;
   sig_calc_transmite_on  <= var_transmite_on;
   sig_calc_transmite_off <= var_transmite_off;
END PROCESS;

-----------------------------------------------------
transmite_rpu: PROCESS(transmite, availableIn, num_header_words,
                       current_state_transmite,
                       header, num_inst, count_calculated_inst,
                       pcknumber, pckpointer, num_pointer_words,
count_pointer_words,
                       cb_canPop, cb_dataIn, sig_br_valid, sig_br_result,
                       count_oper, num_oper_transmite)
-----------------------------------------------------
VARIABLE var_dataOut : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
VARIABLE var_valid   : STD_LOGIC;
VARIABLE var_pop     : STD_LOGIC;
VARIABLE var_br_rd : STD_LOGIC;
VARIABLE var_transmite_inst_oper_ok : STD_LOGIC;
VARIABLE var_count_oper, var_count_oper_reset : STD_LOGIC;
VARIABLE var_count_pointer_reset, var_count_CalcInst_reset : STD_LOGIC;
VARIABLE var_count_IPR_reset : STD_LOGIC;
VARIABLE var_num_oper_transmite : INTEGER;

VARIABLE var_calculate_on, var_transmite_on : STD_LOGIC;
VARIABLE var_calculate_off, var_transmite_off : STD_LOGIC;

BEGIN
   var_dataOut := (OTHERS => '0');
   var_valid  := '0';
   var_pop := '0';
   var_br_rd := '0';
   var_transmite_inst_oper_ok := '0';
   var_count_oper := '0';
   var_count_oper_reset := '0';
   var_count_pointer_reset := '0';
   var_count_CalcInst_reset := '0';
```

```
   var_count_IPR_reset := '0';
   var_num_oper_transmite := num_oper_transmite;

   var_calculate_on  := '0';
   var_calculate_off := '0';
   var_transmite_on  := '0';
   var_transmite_off := '0';

   IF(transmite = '1') THEN
      CASE current_state_transmite IS
         WHEN oHEAD =>
            IF(availableIn >= num_header_words) THEN
               var_dataOut := header;
               var_dataOut(B_N_INSTRUCTIONS DOWNTO E_N_INSTRUCTIONS) :=
std_logic_vector(to_unsigned((num_inst - count_calculated_inst),
LEN_N_INSTRUCTIONS));
               var_valid := '1';

               CASE var_dataOut(B_TYPE DOWNTO E_TYPE) IS
                  WHEN PKT_REGULAR =>
                     next_state_transmite <= oPCKNUMBER;
                  --WHEN PKT_CONTROL =>
                  --WHEN PKT_INTERRUPTED | PKT_CALLER =>
                  WHEN OTHERS =>
                     next_state_transmite <= oHEAD;
               END CASE;
            ELSE -- Execução Localizada
               IF(header(B_TYPE DOWNTO E_TYPE) = PKT_REGULAR) THEN
                  var_calculate_on  := '1';
                  var_transmite_off := '1';

                  var_count_IPR_reset := '1';
               ELSE -- PKT_CONTROL OR PKT_CALLER OR PKT_INTERRUPTED
                  var_calculate_off := '1';
                  var_transmite_on  := '1';
               END IF;

               next_state_transmite <= oHEAD;
            END IF; -- END of availableIn
         WHEN oPCKNUMBER =>
            var_dataOut := pcknumber;
            var_valid := '1';

            next_state_transmite <= oPCKPOINTER;
         WHEN oPCKPOINTER =>
            var_dataOut(B_CONTROL DOWNTO E_CONTROL) := CTRL_HEADER;
            var_dataOut(B_POINTER DOWNTO E_POINTER) :=
std_logic_vector(to_unsigned((num_pointer_words + count_pointer_words),
LEN_POINTER));
```

```
            var_dataOut(B_PROCESS_ADDR DOWNTO E_PROCESS_ADDR) :=
pckpointer(B_PROCESS_ADDR DOWNTO E_PROCESS_ADDR);
            var_dataOut(B_INST_RPU DOWNTO E_INST_RPU) := pckpointer(B_INST_RPU
DOWNTO E_INST_RPU);
            var_valid := '1';

            next_state_transmite <= oINST;
         WHEN oINST =>
            IF(availableIn >= 1) THEN
               -- Operando pode vir do Buffer Results ou do Buffer de Entrada.
               IF(sig_br_valid = '1') THEN
                  IF (sig_br_result(B_CONTROL DOWNTO E_CONTROL) =
CTRL_OPERAND) THEN
                     var_dataOut := sig_br_result;
                     var_br_rd := '1';
                     var_valid := '1';

                     var_transmite_inst_oper_ok := '1';

                     next_state_transmite <= oINST;
                  ELSE
                     next_state_transmite <= oErrState;
                  END IF;
               ELSIF (cb_canPop = '1') THEN
                  IF (cb_dataIn(B_CONTROL DOWNTO E_CONTROL) =
CTRL_INSTRUCTION) THEN
                     var_dataOut := cb_dataIn;
                     var_pop := '1';
                     var_valid := '1';

                     var_count_oper_reset := '1'; -- A cada instrução,
reinicie o contador de operandos.

                     --arguments (operands)
                     IF((to_integer(unsigned(var_dataOut(B_N_OPERANDS DOWNTO
E_N_OPERANDS)))) < 3) THEN
                        var_num_oper_transmite :=
(to_integer(unsigned(var_dataOut(B_N_OPERANDS DOWNTO E_N_OPERANDS)))); --No
máximo dois operandos
                     ELSE
                        var_num_oper_transmite :=
(to_integer(unsigned(var_dataOut(B_RESULT_2 DOWNTO E_RESULT_2)))) + 1; --Posso
ter vários operandos
                     END IF;

                     var_transmite_inst_oper_ok := '1';

                     -- Testar se existe argumentos.
```

```
                     IF(var_num_oper_transmite > 0) THEN -- se tiver algum
argumento
                        next_state_transmite <= oOPER;
                     ELSE
                        next_state_transmite <= oINST;
                     END IF;
                  ELSIF(cb_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_OPERAND)
THEN
                     next_state_transmite <= oOPER;
                  ELSIF(cb_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_END) THEN
                     var_dataOut := cb_dataIn;
                     var_pop := '1';
                     var_valid := '1';

                     var_transmite_inst_oper_ok := '1';

                     next_state_transmite <= oEND;
                  ELSE
                     next_state_transmite <= oErrState;
                  END IF;
               ELSE
                  next_state_transmite <= oINST;
               END IF;
            ELSE -- Execução Localizada
               IF(header(B_TYPE DOWNTO E_TYPE) = PKT_REGULAR) THEN
                  var_calculate_on  := '1';
                  var_transmite_off := '1';

                  var_count_IPR_reset := '1';
               ELSE -- PKT_CONTROL OR PKT_CALLER OR PKT_INTERRUPTED
                  var_calculate_off := '1';
                  var_transmite_on  := '1';
               END IF;

               next_state_transmite <= oINST;
            END IF; -- END of availableIn
         WHEN oOPER =>
            IF(availableIn >= 1) THEN
               -- Operando pode vir do Buffer Results ou do Buffer de Entrada.
               IF(sig_br_valid = '1') THEN
                  IF (sig_br_result(B_CONTROL DOWNTO E_CONTROL) =
CTRL_OPERAND) THEN
                     var_dataOut := sig_br_result;
                     var_br_rd := '1';
                     var_valid := '1';

                     var_transmite_inst_oper_ok := '1';

                     IF (count_oper = num_oper_transmite-1) THEN
```

```
                        next_state_transmite <= oINST;
                     ELSE
                        var_count_oper := '1';
                        next_state_transmite <= oOPER;
                     END IF;
                  ELSE
                     next_state_transmite <= oErrState;
                  END IF;
               ELSIF (cb_canPop = '1') THEN
                  IF (cb_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_OPERAND)
THEN
                     var_dataOut := cb_dataIn;
                     var_pop := '1';
                     var_valid := '1';

                     var_transmite_inst_oper_ok := '1';

                     IF (count_oper = num_oper_transmite-1) THEN
                        next_state_transmite <= oINST;
                     ELSE
                        var_count_oper := '1';
                        next_state_transmite <= oOPER;
                     END IF;
                  ELSE
                     IF (count_oper = num_oper_transmite-1) THEN
                        next_state_transmite <= oINST;
                     ELSE
                        var_count_oper := '1';
                        next_state_transmite <= oINST;
                     END IF;
                  END IF;
               ELSE
                  next_state_transmite <= oOPER; -- Se não tiver operandos no
Buffer nem no BR, então aguarde.
               END IF;
            ELSE -- Execução Localizada
               IF(header(B_TYPE DOWNTO E_TYPE) = PKT_REGULAR) THEN
                  var_calculate_on  := '1';
                  var_transmite_off := '1';

                  var_count_IPR_reset := '1';
               ELSE -- PKT_CONTROL OR PKT_CALLER OR PKT_INTERRUPTED
                  var_calculate_off := '1';
                  var_transmite_on  := '1';
               END IF;

               next_state_transmite <= oOPER;
            END IF; -- END of availableIn
         WHEN oEND =>
```

```
            --------------
            --EndOfPackage
            --------------
            var_calculate_off := '1';
            var_transmite_off := '1';
            -------------------------

            var_count_CalcInst_reset := '1';
            var_count_IPR_reset := '1';
            -- Depois de enviar o PCKPOINTER posso zerar as palavras
calculadas.
            var_count_pointer_reset := '1';

            next_state_transmite <= oHEAD;
         WHEN oErrState =>
            next_state_transmite <= oHEAD;
         WHEN OTHERS =>
            next_state_transmite <= oHEAD;
      END CASE;
   END IF; -- END of transmite

   dataOut <= var_dataOut;
   validDataOut <= var_valid;
   sig_transmite_pop <= var_pop;
   sig_transmite_br_rd <= var_br_rd;
   sig_transmite_inst_oper_word_ok <= var_transmite_inst_oper_ok;
   sig_count_oper <= var_count_oper;
   sig_count_oper_reset <= var_count_oper_reset;
   sig_count_pointer_reset <= var_count_pointer_reset;
   sig_count_CalcInst_reset <= var_count_CalcInst_reset;
   sig_count_IPR_reset <= var_count_IPR_reset;
   num_oper_transmite <= var_num_oper_transmite;

   sig_trans_calculate_on  <= var_calculate_on;
   sig_trans_calculate_off <= var_calculate_off;
   sig_trans_transmite_on  <= var_transmite_on;
   sig_trans_transmite_off <= var_transmite_off;
END PROCESS;

END behaviour;
```

- SU

```
LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE work.ipnosys_package.all;
--------------
ENTITY su IS
```

```
--------------
  PORT(
    -- System signals
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    -- IN interface
    addressRPU : IN STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0); -- RPU's address
(8 bits)

    dataIn_north : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_south : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_east  : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_west  : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataIn : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

    availableIn : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0; -- Buffer na RPU onde
a SU envia os pacotes de controle construídos.

    request     : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

    -- OUT interface
    dataOut      : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataOut : OUT STD_LOGIC;

    availableOut_north : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableOut_south : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableOut_east  : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableOut_west  : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;

    ackRequest  : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0));
END su;

-------------------------------
ARCHITECTURE behaviour OF su IS
-------------------------------

SIGNAL selector : INTEGER RANGE ROUTER_PORTS DOWNTO 0;
SIGNAL sig_done : STD_LOGIC;

BEGIN

   su_ctrl: entity work.su_controller
      PORT MAP(clk,rst,request,sig_done,ackRequest,selector);

   su_dp: entity work.su_data_path
      PORT MAP(clk,rst,
               addressRPU,
               dataIn_north,dataIn_south,dataIn_east,dataIn_west,
```

```
               validDataIn,
               selector,
               availableIn,
               dataOut,
               validDataOut,
               availableOut_north,availableOut_south,availableOut_east,availab
leOut_west,
               sig_done);

END behaviour;


LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE work.ipnosys_package.all;
-------------------------
ENTITY su_controller IS
-------------------------
  PORT(
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    request : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    done    : IN STD_LOGIC;

    ackRequest : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    selector   : OUT INTEGER RANGE ROUTER_PORTS DOWNTO 0);
END su_controller;

----------------------------------------
ARCHITECTURE behaviour OF su_controller IS
----------------------------------------

SIGNAL sig_grant : STD_LOGIC_VECTOR(ROUTER_PORTS-1 downto 0);

BEGIN

RR : entity work.round_robin
   PORT MAP(rst, clk, request, done, sig_grant);

ackRequest <= sig_grant;

WITH sig_grant SELECT
   selector <= 0 WHEN "0001",
               1 WHEN "0010",
               2 WHEN "0100",
               3 WHEN "1000",
               UNRESERVED WHEN OTHERS;
```

```
END behaviour;


LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE work.ipnosys_package.all;
----------------------------
ENTITY su_data_path IS
----------------------------
  PORT(
    -- System signals
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    -- IN interface
    addressRPU : IN STD_LOGIC_VECTOR(LEN_SOURCE-1 DOWNTO 0); -- RPU's address
(8 bits)

    dataIn_north : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_south : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_east  : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataIn_west  : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

    validDataIn : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    selector    : IN INTEGER RANGE ROUTER_PORTS DOWNTO 0;
    availableIn : IN INTEGER RANGE BUFFER_SIZE DOWNTO 0; -- Buffer na RPU onde
a SU envia os pacotes de controle construídos.

    -- OUT interface
    dataOut      : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    validDataOut : OUT STD_LOGIC;

    availableOut_north : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableOut_south : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableOut_east  : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;
    availableOut_west  : OUT INTEGER RANGE BUFFER_SIZE DOWNTO 0;

    done : OUT STD_LOGIC);
END su_data_path;

----------------------------------------
ARCHITECTURE behaviour OF su_data_path IS
----------------------------------------

----------------------------------------
-- Type and signal to implement the FIFO
----------------------------------------
```

```
TYPE SU_buffers IS ARRAY (ROUTER_PORTS-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_dataOutBuffs : SU_buffers;
SIGNAL sig_can_pop  : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_pop      : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
SIGNAL sig_dataIn : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

SIGNAL sig_BO_dataIn : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_BO_dataOut : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL sig_BO_push    : STD_LOGIC;
SIGNAL sig_BO_pop     : STD_LOGIC;
SIGNAL sig_BO_canpush : STD_LOGIC;
SIGNAL sig_BO_canpop  : STD_LOGIC;
SIGNAL sig_BO_validOut : STD_LOGIC;

SIGNAL sig_BO_pu : STD_LOGIC;

------------------------------
-- Instruction Register Signals
------------------------------

SIGNAL sig_instru : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

----------------
-- Count Signals
----------------

SIGNAL sig_flag_count, sig_flag_reset_count : STD_LOGIC;
SIGNAL count_oper : INTEGER; --:= 1;
SIGNAL n_operands : INTEGER RANGE 0 TO RESULT_SIZE := 0;

--------------
-- FSM Signals
--------------

TYPE state_type IS (HEAD, INST, OPER, ENDD, ErrState);
SIGNAL current_state, next_state : state_type;

BEGIN

------------
-- Registers
------------

countOper : entity work.counter
   PORT MAP(clk, sig_flag_reset_count, sig_flag_count, '0', count_oper);

----------
-- Buffers
```

```
----------

-- IN
buf_I_N: entity work.buffer_fifo
   PORT MAP(clk, rst, dataIn_north, OPEN, sig_can_pop(P_NORTH),
validDataIn(P_NORTH),
                       sig_pop(P_NORTH), sig_dataOutBuffs(P_NORTH),
                       OPEN, OPEN, availableOut_north);

buf_I_S: entity work.buffer_fifo
   PORT MAP(clk, rst, dataIn_south, OPEN, sig_can_pop(P_SOUTH),
validDataIn(P_SOUTH),
                       sig_pop(P_SOUTH), sig_dataOutBuffs(P_SOUTH),
                       OPEN, OPEN, availableOut_south);

buf_I_E: entity work.buffer_fifo
   PORT MAP(clk, rst, dataIn_east, OPEN, sig_can_pop(P_EAST),
validDataIn(P_EAST),
                       sig_pop(P_EAST), sig_dataOutBuffs(P_EAST),
                       OPEN, OPEN, availableOut_east);

buf_I_W: entity work.buffer_fifo
   PORT MAP(clk, rst, dataIn_west, OPEN, sig_can_pop(P_WEST),
validDataIn(P_WEST),
                       sig_pop(P_WEST), sig_dataOutBuffs(P_WEST),
                       OPEN, OPEN, availableOut_west);

-- OUT
buf_O: entity work.buffer_fifo
   PORT MAP(clk, rst, sig_BO_dataIn, sig_BO_canpush, sig_BO_canpop,
sig_BO_push,
                       sig_BO_pop, sig_BO_dataOut,
                       OPEN, OPEN, OPEN);

-------------------------------------------------------------------------
-------
PROCESS(selector, sig_can_pop, sig_BO_canpush, sig_dataOutBuffs)
-------------------------------------------------------------------------
-------
VARIABLE var_dataIn : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

BEGIN
   --initialize values
   var_dataIn := (OTHERS => '0');

   IF (selector /= UNRESERVED) THEN
      IF(sig_can_pop(selector) = '1') THEN
         IF(sig_BO_canpush = '1') THEN -- So posso dar pop se houver onde
armazenar
```

```
            var_dataIn := sig_dataOutBuffs(selector);
          END IF;
       END IF;
    END IF;

    sig_dataIn <= var_dataIn;
END PROCESS;

-- current state
PROCESS(clk,rst)
----------------
  BEGIN
    IF (rst='1') THEN
      current_state <= HEAD;
    ELSIF (clk'EVENT AND clk='1') THEN
      current_state <= next_state;
    END IF;
  END PROCESS;

------------------------------
PROCESS(current_state, sig_dataIn, sig_BO_pu, selector, addressRPU,
count_oper, n_operands, sig_instru, sig_BO_canpush)
------------------------------
VARIABLE var_pop        : STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
VARIABLE var_muxReroute : STD_LOGIC_VECTOR(LEN_REROUTE-1 DOWNTO 0); -- reroute
VARIABLE var_muxOutput  : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
VARIABLE var_BO_push, var_flag_count, var_flag_reset_count : STD_LOGIC;

BEGIN
    var_pop := (OTHERS => '0');
    var_muxReroute := (OTHERS => '0');
    var_muxOutput  := (OTHERS => '0');
    var_BO_push := '1';
    var_flag_count := '0';
    var_flag_reset_count := '0';

    CASE current_state IS
       WHEN HEAD =>
          IF (sig_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_INSTRUCTION) THEN
             var_flag_reset_count := '1';

             sig_instru <= sig_dataIn;

             IF((to_integer(unsigned(sig_dataIn(B_N_OPERANDS DOWNTO
E_N_OPERANDS)))) < 3) THEN
                n_operands <= (to_integer(unsigned(sig_dataIn(B_N_OPERANDS
DOWNTO E_N_OPERANDS))));
             ELSE
```

```
               n_operands <= (to_integer(unsigned(sig_dataIn(B_RESULT_2 DOWNTO
E_RESULT_2)))) + 1;
            END IF;

            IF (sig_dataIn(B_INSTRUCTION DOWNTO E_INSTRUCTION) =
std_logic_vector(to_unsigned(LOAD, LEN_INSTRUCTION))) THEN
               var_muxReroute := std_logic_vector(to_unsigned(0,
LEN_REROUTE)); --"000000"; -- Instruction is LOAD "00001010"
            ELSE
               var_muxReroute := std_logic_vector(to_unsigned(selector,
LEN_REROUTE));
            END IF;

            var_muxOutput(B_CONTROL DOWNTO E_CONTROL) := CTRL_HEADER;
            var_muxOutput(B_TARGET DOWNTO E_TARGET) := sig_dataIn(B_RESULT_1-3
DOWNTO E_RESULT_1);
            var_muxOutput(B_SOURCE DOWNTO E_SOURCE) := addressRPU;
            var_muxOutput(B_N_INSTRUCTIONS DOWNTO E_N_INSTRUCTIONS) :=
std_logic_vector(to_unsigned(1, LEN_N_INSTRUCTIONS));
            var_muxOutput(B_REROUTE DOWNTO E_REROUTE) := var_muxReroute;
            var_muxOutput(B_TYPE DOWNTO E_TYPE) := PKT_CONTROL;

            next_state <= INST;
         ELSE
            next_state <= HEAD;
         END IF;
      WHEN INST =>
         IF(sig_BO_canpush = '1') THEN
            var_muxOutput := sig_instru;
            var_pop(selector) := '1';

            next_state <= OPER;
         ELSE
            next_state <= INST;
         END IF;
      WHEN OPER =>
         IF (sig_dataIn(B_CONTROL DOWNTO E_CONTROL) = CTRL_OPERAND) THEN
            var_muxOutput := sig_dataIn;
            var_pop(selector) := '1';

            var_flag_count := '1';

            IF (count_oper = n_operands) THEN
               next_state <= ENDD;
            ELSE
               next_state <= OPER;
            END IF;
         ELSE
            next_state <= OPER;
```

```
         END IF;
      WHEN ENDD =>
         IF(sig_BO_canpush = '1') THEN
            var_muxOutput(B_CONTROL DOWNTO E_CONTROL) := CTRL_END;
            var_muxOutput(B_END DOWNTO E_END) := EOP;

            next_state <= HEAD;
         ELSE
            next_state <= ENDD;
         END IF;
      WHEN ErrState =>
         next_state <= HEAD;
      WHEN OTHERS =>
         var_muxOutput := (OTHERS => 'X');
         var_BO_push := '0';

         next_state <= HEAD;
   END CASE;

   sig_BO_dataIn <= var_muxOutput;
   sig_BO_push <= var_BO_push; -- push so estara ativo quando a palavra
estiver montada.
   sig_pop <= var_pop;
   sig_flag_count <= var_flag_count;
   sig_flag_reset_count <= var_flag_reset_count;
END PROCESS;

-----------------------------------------------------
PROCESS(sig_BO_dataOut, availableIn, sig_BO_canpop)
-----------------------------------------------------
VARIABLE var_dataOut : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
VARIABLE var_BO_pop  : STD_LOGIC;
VARIABLE var_valid   : STD_LOGIC;
VARIABLE var_done    : STD_LOGIC;

BEGIN
   var_dataOut := (OTHERS => '0');
   var_BO_pop  := '0';
   var_valid   := '0';
   var_done    := '0';

   IF(sig_BO_canpop = '1') THEN
      IF(availableIn > 1) THEN
         var_dataOut := sig_BO_dataOut;
         var_BO_pop := '1';

         IF(var_dataOut(B_CONTROL DOWNTO E_CONTROL) /= "0000") THEN
            var_valid := '1';
         END IF;
```

```
         IF(var_dataOut(B_CONTROL DOWNTO E_CONTROL) = CTRL_END) THEN
            var_done := '1';
         END IF;
      END IF;
   END IF;

   done <= var_done;
   dataOut <= var_dataOut;
   validDataOut <= var_valid;
   sig_BO_pop <= var_BO_pop;
END PROCESS;

END behaviour;
```

- ALU

```
LIBRARY ieee, lpm, work;
USE lpm.lpm_components.all;
USE ieee.std_logic_1164.all;
USE work.ipnosys_package.all;
--------------
ENTITY alu IS
--------------
  PORT(
    -- System signals
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    -- ALU interface IN
    dataA_north     : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataA_south     : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataA_east      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataA_west      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

    dataB_north     : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataB_south     : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataB_east      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataB_west      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

    operation_north : IN STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
    operation_south : IN STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
    operation_east  : IN STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
    operation_west  : IN STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);

    validDataIn     : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

    request   : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
```

```
    -- ALU interface OUT
    result_north  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    result_south  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    result_east   : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    result_west   : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

    exception_north : OUT STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
    exception_south : OUT STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
    exception_east  : OUT STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
    exception_west  : OUT STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);

    validResult   : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    ackRequest    : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    neg           : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    zero          : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0));
END alu;

---------------------------------
ARCHITECTURE behaviour OF alu IS
---------------------------------

SIGNAL selector : INTEGER RANGE ROUTER_PORTS DOWNTO 0;
SIGNAL sig_done : STD_LOGIC;

BEGIN

   alu_ctr: entity work.alu_controller
      PORT MAP(
      clk, rst,
      request,
      sig_done,
      ackRequest,
      selector);

   alu_dp: entity work.alu_data_path
      PORT MAP(
      clk, rst,
      dataA_north, dataA_south, dataA_east, dataA_west,
      dataB_north, dataB_south, dataB_east, dataB_west,
      operation_north, operation_south, operation_east, operation_west,
      validDataIn,
      --
      selector,
      --
      result_north, result_south, result_east, result_west,
      exception_north, exception_south, exception_east, exception_west,
      validResult,
      neg, zero,
```

```
      sig_done);

END behaviour;


LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE IEEE.STD_LOGIC_arith.all;
USE work.ipnosys_package.all;
-------------------------
ENTITY alu_controller IS
-------------------------
  PORT(
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    request : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    done    : IN STD_LOGIC;

    ackRequest : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    selector   : OUT INTEGER RANGE ROUTER_PORTS DOWNTO 0);
END alu_controller;

-----------------------------------------
ARCHITECTURE behaviour OF alu_controller IS
-----------------------------------------

SIGNAL sig_grant : STD_LOGIC_VECTOR(ROUTER_PORTS-1 downto 0);

BEGIN

RR : entity work.round_robin
   PORT MAP(rst, clk, request, done, sig_grant);

ackRequest <= sig_grant;

WITH sig_grant SELECT
   selector <= 0 WHEN "0001",
               1 WHEN "0010",
               2 WHEN "0100",
               3 WHEN "1000",
               UNRESERVED WHEN OTHERS;

END behaviour;


LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE work.ipnosys_package.all;
```

```
----------------------------
ENTITY alu_data_path IS
----------------------------
  PORT(
    -- System signals
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    -- ALU interface IN
    dataA_north     : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataA_south     : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataA_east      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataA_west      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

    dataB_north     : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataB_south     : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataB_east      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    dataB_west      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

    operation_north : IN STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
    operation_south : IN STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
    operation_east  : IN STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
    operation_west  : IN STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);

    validDataIn     : IN STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);

    -- Selector of MUX IN
    selector : IN INTEGER RANGE ROUTER_PORTS DOWNTO 0;

    -- ALU interface OUT
    result_north  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    result_south  : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    result_east   : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
    result_west   : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);

    exception_north : OUT STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
    exception_south : OUT STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
    exception_east  : OUT STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
    exception_west  : OUT STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);

    validResult   : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    neg           : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    zero          : OUT STD_LOGIC_VECTOR(ROUTER_PORTS-1 DOWNTO 0);
    done          : OUT STD_LOGIC);
END alu_data_path;

-------------------------------------------
ARCHITECTURE behaviour OF alu_data_path IS
-------------------------------------------
```

```
SIGNAL dataA       : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL dataB       : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
SIGNAL operation   : STD_LOGIC_VECTOR(LEN_INSTRUCTION-1 DOWNTO 0);
SIGNAL sig_validDataIn : STD_LOGIC;

SIGNAL sel : INTEGER RANGE ROUTER_PORTS DOWNTO 0;

BEGIN

--------------
PROCESS(selector, dataA_north, dataB_north, operation_north, dataA_south,
dataB_south, operation_south,
dataA_east, dataB_east, operation_east, dataA_west, dataB_west,
operation_west, validDataIn)
--------------
BEGIN
   CASE selector IS
      WHEN P_NORTH =>
         dataA <= dataA_north;
         dataB <= dataB_north;
         operation <= operation_north;
         sig_validDataIn <= validDataIn(P_NORTH);
      WHEN P_SOUTH =>
         dataA <= dataA_south;
         dataB <= dataB_south;
         operation <= operation_south;
         sig_validDataIn <= validDataIn(P_SOUTH);
      WHEN P_WEST =>
         dataA <= dataA_west;
         dataB <= dataB_west;
         operation <= operation_west;
         sig_validDataIn <= validDataIn(P_WEST);
      WHEN P_EAST =>
         dataA <= dataA_east;
         dataB <= dataB_east;
         operation <= operation_east;
         sig_validDataIn <= validDataIn(P_EAST);
      WHEN OTHERS =>
         dataA <= (OTHERS => '0');
         dataB <= (OTHERS => '0');
         operation <= (OTHERS => '0');
         sig_validDataIn <= '0';
   END CASE;

   sel <= selector;
END PROCESS;

-- current state
```

```
PROCESS(clk,rst)
----------------

--declaration of variables
   VARIABLE var_a  : SIGNED(LEN_WIDTH-5 DOWNTO 0); -- DataA
   VARIABLE var_b  : SIGNED(LEN_WIDTH-5 DOWNTO 0); -- DataB
   VARIABLE var_op : INTEGER;                      -- Operation
   VARIABLE var_r  : SIGNED(LEN_WIDTH-5 DOWNTO 0); -- Result
   VARIABLE var_z  : SIGNED(0 DOWNTO 0);           -- Zero
   VARIABLE var_n  : SIGNED(0 DOWNTO 0);           -- Neg
   VARIABLE var_e  : SIGNED(1 DOWNTO 0);           -- Exception
   VARIABLE var_v  : SIGNED(0 DOWNTO 0);           -- ValidResult
   VARIABLE var_o  : SIGNED(LEN_WIDTH-5 DOWNTO 0); -- Overflow

   VARIABLE var_mult : SIGNED((LEN_WIDTH*2)-9 DOWNTO 0);

   VARIABLE var_done : STD_LOGIC;

   VARIABLE sig_result  : STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);
   VARIABLE sig_except  : STD_LOGIC_VECTOR(LEN_EXCEPTION-1 DOWNTO 0);
   VARIABLE sig_neg      : STD_LOGIC;
   VARIABLE sig_zero     : STD_LOGIC;
   VARIABLE sig_valid    : STD_LOGIC;

  BEGIN
    IF (rst='1') THEN
      var_r    := (OTHERS => '0');
      var_z(0) := '0';
      var_n(0) := '0';
      var_e    := "00";
      var_v(0) := '0';
      var_done := '0';
    ELSIF (clk'EVENT AND clk='1') THEN
     --initialize values
      var_a    := SIGNED(dataA(LEN_WIDTH-5 DOWNTO 0));
      var_b    := SIGNED(dataB(LEN_WIDTH-5 DOWNTO 0));

      IF(sig_validDataIn = '1') THEN
         var_op   := to_integer(unsigned(operation));
      ELSE
         var_op   := 11; -- Não faço operação alguma.
      END IF;

      var_r    := (OTHERS => '0');
      var_z(0) := '0';
      var_n(0) := '0';
      var_e    := "00";
      var_v(0) := '1';
      var_done := '1';
```

```
      CASE var_op IS
         --(0)add
         WHEN ADD =>
            var_r := var_a + var_b;
         --(1)sub
         WHEN SUBB =>
            var_r := var_a - var_b;
         --(2)mul
         WHEN MUL =>
            var_mult := var_a * var_b;
            var_r := var_mult(LEN_WIDTH-5 DOWNTO 0); -- Expressão tem 64
elementos, mas deve ter 32
         --(3)div
         WHEN DIV =>
            IF (to_integer(var_b) = 0) THEN
               var_e := "10";
               var_r := (OTHERS => 'U');
            ELSE
               var_r := var_a / var_b;
            END IF;
         --(4)not
         WHEN NOTT =>
            var_r := NOT(var_b);
         --(5)and
         WHEN ANDD =>
            var_r := var_a AND var_b;
         --(6)or
         WHEN ORR =>
            var_r := var_a OR var_b;
         --(7)xor
         WHEN XORR =>
            var_r := var_a XOR var_b;
         --(8)srl
         WHEN RSHIFT =>
            var_r := signed(std_logic_vector(unsigned(var_a) srl
to_integer(unsigned(var_b))));

            var_o := signed(std_logic_vector(unsigned(var_r) sll
to_integer(unsigned(var_b))));
            IF (var_o /= var_a) THEN
               var_e := "01"; -- set exception if occurs overflow
               var_r := (OTHERS => 'U');
            END IF;
         --(9)srl
         WHEN LSHIFT =>
            var_r := signed(std_logic_vector(unsigned(var_a) sll
to_integer(unsigned(var_b))));
```

```
            var_o := signed(std_logic_vector(unsigned(var_r) srl
to_integer(unsigned(var_b))));
            IF (var_o /= var_a) THEN
               var_e := "01"; -- set exception if occurs overflow
               var_r := (OTHERS => 'U');
            END IF;
         WHEN OTHERS =>
            var_r := (OTHERS => '0');
            var_v(0) := '0';
            var_done := '0';
      END CASE;

      IF(var_v(0) = '1') THEN
         -- set zero bit if result equals zero
         IF(to_integer(var_r) = 0 AND std_logic_vector(var_r) /=
("UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU")) THEN
            var_z(0) := '1';
         END IF;

         -- set negative bit if result less than zero
         IF (std_logic_vector(var_r) = ("UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU"))
THEN
            var_n(0) := '0';
         ELSE
            var_n(0) := var_r(LEN_WIDTH-5);
         END IF;

         -- assign variables to output signals
         sig_result(LEN_WIDTH-1 DOWNTO LEN_WIDTH-4) :=
STD_LOGIC_VECTOR(CTRL_OPERAND);
         sig_result(LEN_WIDTH-5 DOWNTO 0) := STD_LOGIC_VECTOR(var_r);
         sig_except := STD_LOGIC_VECTOR(var_e);
         sig_zero := var_z(0);
         sig_neg := var_n(0);
         sig_valid := var_v(0);
      END IF;
      CASE sel IS
         WHEN P_NORTH =>
            result_north <= sig_result;
            validResult(P_NORTH) <= sig_valid;
            exception_north <= sig_except;
            zero(P_NORTH) <= sig_zero;
            neg(P_NORTH) <= sig_neg;
            done <= '1';

            result_south <= (OTHERS => '0');
            validResult(P_SOUTH) <= '0';
            exception_south <= (OTHERS => '0');
            zero(P_SOUTH) <= '0';
```

```
    neg(P_SOUTH) <= '0';
    result_west  <= (OTHERS => '0');
    validResult(P_WEST) <= '0';
    exception_west  <= (OTHERS => '0');
    zero(P_WEST) <= '0';
    neg(P_WEST) <= '0';
    result_east  <= (OTHERS => '0');
    validResult(P_EAST) <= '0';
    exception_east  <= (OTHERS => '0');
    zero(P_EAST) <= '0';
    neg(P_EAST) <= '0';

WHEN P_SOUTH =>
    result_south <= sig_result;
    validResult(P_SOUTH) <= sig_valid;
    exception_south <= sig_except;
    zero(P_SOUTH) <= sig_zero;
    neg(P_SOUTH) <= sig_neg;
    done <= '1';

    result_north <= (OTHERS => '0');
    validResult(P_NORTH) <= '0';
    exception_north <= (OTHERS => '0');
    zero(P_NORTH) <= '0';
    neg(P_NORTH) <= '0';
    result_west  <= (OTHERS => '0');
    validResult(P_WEST) <= '0';
    exception_west  <= (OTHERS => '0');
    zero(P_WEST) <= '0';
    neg(P_WEST) <= '0';
    result_east  <= (OTHERS => '0');
    validResult(P_EAST) <= '0';
    exception_east  <= (OTHERS => '0');
    zero(P_EAST) <= '0';
    neg(P_EAST) <= '0';

WHEN P_WEST =>
    result_west  <= sig_result;
    validResult(P_WEST) <= sig_valid;
    exception_west  <= sig_except;
    zero(P_WEST) <= sig_zero;
    neg(P_WEST) <= sig_neg;
    done <= '1';

    result_north <= (OTHERS => '0');
    validResult(P_NORTH) <= '0';
    exception_north <= (OTHERS => '0');
    zero(P_NORTH) <= '0';
    neg(P_NORTH) <= '0';
```

```
    result_south <= (OTHERS => '0');
    validResult(P_SOUTH) <= '0';
    exception_south <= (OTHERS => '0');
    zero(P_SOUTH) <= '0';
    neg(P_SOUTH) <= '0';
    result_east  <= (OTHERS => '0');
    validResult(P_EAST) <= '0';
    exception_east  <= (OTHERS => '0');
    zero(P_EAST) <= '0';
    neg(P_EAST) <= '0';

WHEN P_EAST =>
    result_east  <= sig_result;
    validResult(P_EAST) <= sig_valid;
    exception_east  <= sig_except;
    zero(P_EAST) <= sig_zero;
    neg(P_EAST) <= sig_neg;
    done <= '1';

    result_north <= (OTHERS => '0');
    validResult(P_NORTH) <= '0';
    exception_north <= (OTHERS => '0');
    zero(P_NORTH) <= '0';
    neg(P_NORTH) <= '0';
    result_south <= (OTHERS => '0');
    validResult(P_SOUTH) <= '0';
    exception_south <= (OTHERS => '0');
    zero(P_SOUTH) <= '0';
    neg(P_SOUTH) <= '0';
    result_west  <= (OTHERS => '0');
    validResult(P_WEST) <= '0';
    exception_west  <= (OTHERS => '0');
    zero(P_WEST) <= '0';
    neg(P_WEST) <= '0';

WHEN OTHERS =>
    result_north <= (OTHERS => '0');
    exception_north <= (OTHERS => '0');
    result_south <= (OTHERS => '0');
    exception_south <= (OTHERS => '0');
    result_west  <= (OTHERS => '0');
    exception_west  <= (OTHERS => '0');
    result_east  <= (OTHERS => '0');
    exception_east  <= (OTHERS => '0');

    validResult <= (OTHERS => '0');
    zero <= (OTHERS => '0');
    neg <= (OTHERS => '0');
    done <= '0';
```

```
      END CASE;
    END IF;
  END PROCESS;

END behaviour;
```

- *Buffer Results*

```
LIBRARY ieee, work;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE work.ipnosys_package.all;
-------------------------
ENTITY buffer_results IS
-------------------------
  PORT(
    -- System signals
    clk : IN  STD_LOGIC;  -- clock
    rst : IN  STD_LOGIC;  -- reset

    -- Interfaces
    data_in      : IN STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);       -- input
data channel
    addressWrite : IN STD_LOGIC_VECTOR((LEN_RESULT*2)-1 DOWNTO 0);
    wr         : IN STD_LOGIC;                                    -- command
to write a data
    addressRead  : IN STD_LOGIC_VECTOR(LEN_RESULT-1 DOWNTO 0);
    rd           : IN STD_LOGIC;                                    -- command
to read a data

    data_out    : OUT STD_LOGIC_VECTOR(LEN_WIDTH-1 DOWNTO 0);       -- output
data channel
    valid_data  : OUT STD_LOGIC);
END buffer_results;

-------------------------------------------
ARCHITECTURE behaviour OF buffer_results IS
-------------------------------------------
TYPE STOREAGE_TYPE IS ARRAY(RESULT_SIZE-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(LEN_WIDTH DOWNTO 0);
SIGNAL registers : STOREAGE_TYPE;

SIGNAL addrR1 : INTEGER RANGE 0 TO RESULT_SIZE-1;

BEGIN
```

```
------------
PROCESS(rst, clk)
------------
VARIABLE addrW1 : INTEGER;
VARIABLE addrW2 : INTEGER;

  BEGIN
   IF (rst = '1') THEN
    FOR i IN 0 TO (RESULT_SIZE-1) LOOP
      registers(i) <= (OTHERS => '0');
      END LOOP;
    ELSIF rising_edge(clk) THEN
      IF (wr = '1') THEN
      addrW1 := to_integer(unsigned(addressWrite(B_RESULT_1 DOWNTO
E_RESULT_1)));
      addrW2 := to_integer(unsigned(addressWrite(B_RESULT_2 DOWNTO
E_RESULT_2)));
      IF (addrW1 = addrW2) THEN
        IF (addrW1 <= RESULT_SIZE-1) THEN
          registers(addrW1)(LEN_WIDTH-1 DOWNTO 0) <= data_in;
          registers(addrW1)(LEN_WIDTH) <= '1';
        END IF;
      ELSE
        IF (addrW1 <= RESULT_SIZE-1) THEN
          registers(addrW1)(LEN_WIDTH-1 DOWNTO 0) <= data_in;
          registers(addrW1)(LEN_WIDTH) <= '1';
        END IF;

        IF (addrW2 <= RESULT_SIZE-1) THEN
          registers(addrW2)(LEN_WIDTH-1 DOWNTO 0) <= data_in;
          registers(addrW2)(LEN_WIDTH) <= '1';
        END IF;
      END IF;
      ELSIF (rd = '1') THEN
      registers(addrR1)(LEN_WIDTH) <= '0';
       END IF;
    END IF;
END PROCESS;

  addrR1 <= to_integer(unsigned(addressRead));
  data_out <= registers(addrR1)(LEN_WIDTH-1 DOWNTO 0);
  valid_data <= registers(addrR1)(LEN_WIDTH);

END behaviour;
```

- Componentes Auxiliares

```
LIBRARY ieee, work;
```

```
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE work.ipnosys_package.all;
------------------------
------------------------
ENTITY counter IS
------------------------
------------------------
  GENERIC (startCount: INTEGER := 1);
  PORT(
    -- System signals
    clk   : IN  STD_LOGIC;  -- clock
    rst   : IN  STD_LOGIC;  -- reset

    -- interface
    increment : IN  STD_LOGIC;  -- command to increment the counter
    decrement : IN  STD_LOGIC;  -- command to decrement the counter

    -- Control interface
    used : OUT INTEGER := startCount     -- current counter state
  );
END counter;

----------------------------------------
----------------------------------------
ARCHITECTURE behaviour OF counter IS
----------------------------------------
----------------------------------------
SIGNAL current_state   : INTEGER := startCount; -- current state
SIGNAL next_state      : INTEGER; -- next state
BEGIN
  ------------- next state
  p_next_state:PROCESS(current_state,increment, decrement)
  ------------------------
  BEGIN
    -- EMPTY
    IF (current_state = 1) THEN
      IF (increment='1') THEN
        next_state <= current_state+1;
      ELSE
        next_state <= current_state;
      END IF;

    ELSE
      IF (increment='1') THEN
        IF (decrement='1') THEN
          next_state <= current_state;
        ELSE
          next_state <= current_state+1;
```

```
        END IF;
      ELSIF (decrement='1') THEN
        next_state <= current_state-1;
      ELSE
        next_state <= current_state;
      END IF;
    END IF;
  END PROCESS p_next_state;

  -- current state
  p_current_state:PROCESS(clk,rst)
  ----------------
  BEGIN
    IF (rst='1') THEN
      current_state  <= startCount;
    ELSIF (clk'EVENT AND clk='1') THEN
      current_state  <= next_state;
    END IF;
  END PROCESS p_current_state;

  ---------
  -- OUTPUT
  ---------
  used <= current_state;
END behaviour;


-- ------------------------------------------------------------------------
--
-- Copyright (c) 2009 Benjamin Krill <benjamin@krll.de>
--
-- Permission is hereby granted, free of charge, to any person obtaining a
copy
-- of this software and associated documentation files (the "Software"), to
deal
-- in the Software without restriction, including without limitation the
rights
-- to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
-- copies of the Software, and to permit persons to whom the Software is
-- furnished to do so, subject to the following conditions:
--
-- The above copyright notice and this permission notice shall be included in
-- all copies or substantial portions of the Software.
--
-- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
-- IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
-- FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
-- AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
```

```
-- LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
-- OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
-- THE SOFTWARE.
-- ------------------------------------------------------------------------
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity round_robin is
   generic ( CNT : integer := 4 );
   port (
      rst   : in    std_logic;
      clk   : in    std_logic;

      req   : in    std_logic_vector(CNT-1 downto 0);
      done  : in    std_logic;
      grant : out   std_logic_vector(CNT-1 downto 0)
   );
end;

architecture round_robin of round_robin is
   signal grant_q  : std_logic_vector(CNT-1 downto 0);
   signal pre_req  : std_logic_vector(CNT-1 downto 0);
   signal sel_gnt  : std_logic_vector(CNT-1 downto 0);
   signal isol_lsb : std_logic_vector(CNT-1 downto 0);
   signal mask_pre : std_logic_vector(CNT-1 downto 0);
   signal win      : std_logic_vector(CNT-1 downto 0);
begin
   grant    <= grant_q;
   mask_pre <=      req and not (std_logic_vector(unsigned(pre_req) - 1) or
pre_req); -- Mask off previous winners
   sel_gnt  <= mask_pre and      std_logic_vector(unsigned(not(mask_pre)) +
1);       -- Select new winner
   isol_lsb <=      req and      std_logic_vector(unsigned(not(req)) + 1);
-- Isolate least significant set bit.
   win      <= sel_gnt when mask_pre /= (CNT-1 downto 0 => '0') else isol_lsb;

   process (clk, rst)
   begin
   if rst = '1' then
      pre_req <= (others => '0');
      grant_q <= (others => '0');
   elsif rising_edge(clk) then
      grant_q <= grant_q;
      pre_req <= pre_req;
      if grant_q = (CNT-1 downto 0 => '0') or done = '1' then -- grant_q = ack
         if win /= (CNT-1 downto 0 => '0') then
```

```
            pre_req <= win;
         end if;
         grant_q <= win;
      end if;
   end if;
   end process;

end round_robin;
```

- Constantes

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_arith.all;

------------------------
PACKAGE ipnosys_package IS
------------------------

--/************************************************
-- * PARAMETERS TO SIMULATION
-- ************************************************/
CONSTANT UNRESERVED     : INTEGER := 4; -- Útil na ALU e SU.
CONSTANT N_LANES        : INTEGER := 1;  -- number of lanes for virtual
channel (lane 0: control packets; others: regular packets)
CONSTANT RESULT_SIZE    : INTEGER := 256; --2048;
CONSTANT BUFFER_SIZE    : INTEGER := 10; -- 30; -- 100;
CONSTANT SIDEE          : INTEGER := 4;
CONSTANT N_MAUS         : INTEGER := 4;
CONSTANT IONODE_REQUEST_TABLE_SIZE : INTEGER := 1000;
CONSTANT MAX_MEMARBITER : INTEGER := 2; --//number of components that uses
MemArbiter

--************************************************
-- TAGS TO BEGIN AND THE END OF THE PACKET'S FIELD
--************************************************
-- Bit control (4 bits)
CONSTANT B_CONTROL : INTEGER := 35;
CONSTANT E_CONTROL : INTEGER := 32;
-- Target (8 bits)
CONSTANT B_TARGET  : INTEGER := 31;
CONSTANT E_TARGET  : INTEGER := 24;
-- Source (8 bits)
CONSTANT B_SOURCE  : INTEGER := 23;
CONSTANT E_SOURCE  : INTEGER := 16;
-- Quantity of instructions (8 bits)
CONSTANT B_N_INSTRUCTIONS : INTEGER := 15;
```

```
CONSTANT E_N_INSTRUCTIONS : INTEGER := 8;
-- Re-routing (6 bits)
CONSTANT B_REROUTE : INTEGER := 7;
CONSTANT E_REROUTE : INTEGER := 2;
-- Type of packet (2 bits)
CONSTANT B_TYPE    : INTEGER := 1;
CONSTANT E_TYPE    : INTEGER := 0;

-- The packet's identifier (16 bits)
CONSTANT B_PROCESS_NUM  : INTEGER := 31;
CONSTANT E_PROCESS_NUM  : INTEGER := 16;
-- The packet's number (16 bits)
CONSTANT B_PACKET_NUM   : INTEGER := 15;
CONSTANT E_PACKET_NUM   : INTEGER := 0;

-- MAU's addresss that inject the packet first time (8 bits)
CONSTANT B_PROCESS_ADDR : INTEGER := 31;
CONSTANT E_PROCESS_ADDR : INTEGER := 24;
-- Instruction per RPU (8 bits)
CONSTANT B_INST_RPU : INTEGER := 23;
CONSTANT E_INST_RPU : INTEGER := 16;
-- Pointer (16 bits)
CONSTANT B_POINTER  : INTEGER := 15;
CONSTANT E_POINTER  : INTEGER := 0;

-- Instruction identifier (8 bits)
CONSTANT B_INSTRUCTION : INTEGER := 31;
CONSTANT E_INSTRUCTION : INTEGER := 24;
-- Quantity of operands (2 bits)
CONSTANT B_N_OPERANDS  : INTEGER := 23;
CONSTANT E_N_OPERANDS  : INTEGER := 22;
-- Result 1 (11 bits)
CONSTANT B_RESULT_1 : INTEGER := 21;
CONSTANT E_RESULT_1 : INTEGER := 11;
-- Result 2 (11 bits)
CONSTANT B_RESULT_2 : INTEGER := 10;
CONSTANT E_RESULT_2 : INTEGER := 0;

-- Operand (32 bits)
CONSTANT B_OPERAND  : INTEGER := 31;
CONSTANT E_OPERAND  : INTEGER := 0;

-- End of the packet (32 bits
CONSTANT B_END : INTEGER := 31;
CONSTANT E_END : INTEGER := 0;


--**********************************************
-- LENGHT OF THE FIELDS IN THE PACKET
```

```
--************************************************
CONSTANT LEN_WIDTH          : INTEGER := 36;
CONSTANT LEN_EXCEPTION      : INTEGER := 2;
CONSTANT LEN_HEADER         : INTEGER := 32;
CONSTANT LEN_CONTROL        : INTEGER := (B_CONTROL - E_CONTROL + 1);
CONSTANT LEN_TARGET         : INTEGER := (B_TARGET - E_TARGET + 1);
CONSTANT LEN_SOURCE         : INTEGER := (B_SOURCE - E_SOURCE + 1);
CONSTANT LEN_N_INSTRUCTIONS : INTEGER := (B_N_INSTRUCTIONS - E_N_INSTRUCTIONS
+ 1);
CONSTANT LEN_REROUTE        : INTEGER := (B_REROUTE - E_REROUTE + 1);
CONSTANT LEN_TYPE           : INTEGER := (B_TYPE - E_TYPE + 1);
CONSTANT LEN_PROCESS_NUM    : INTEGER := (B_PROCESS_NUM - E_PROCESS_NUM + 1);
CONSTANT LEN_PACKET_NUM     : INTEGER := (B_PACKET_NUM - E_PACKET_NUM + 1);
CONSTANT LEN_PROCESS_ADDR   : INTEGER := (B_PROCESS_ADDR - E_PROCESS_ADDR +
1);
CONSTANT LEN_INST_RPU       : INTEGER := (B_INST_RPU - E_INST_RPU + 1);
CONSTANT LEN_POINTER        : INTEGER := (B_POINTER - E_POINTER + 1);
CONSTANT LEN_INSTRUCTION    : INTEGER := (B_INSTRUCTION - E_INSTRUCTION + 1);
CONSTANT LEN_N_OPERANDS     : INTEGER := (B_N_OPERANDS - E_N_OPERANDS + 1);
CONSTANT LEN_RESULT         : INTEGER := (B_RESULT_1 - E_RESULT_1 + 1);
CONSTANT LEN_OPERAND        : INTEGER := (B_OPERAND - E_OPERAND + 1);
CONSTANT LEN_END            : INTEGER := (B_END - E_END + 1);

--************************************************
-- VALUES OF THE CONTROL BITS
--************************************************
CONSTANT CTRL_HEADER      : STD_LOGIC_VECTOR := "1000"; -- 8
CONSTANT CTRL_INSTRUCTION : STD_LOGIC_VECTOR := "0010"; -- 2
CONSTANT CTRL_OPERAND     : STD_LOGIC_VECTOR := "0001"; -- 1
CONSTANT CTRL_END         : STD_LOGIC_VECTOR := "0100"; -- 4

--************************************************
-- TYPE OF PACKETS
--************************************************
CONSTANT PKT_CONTROL     : STD_LOGIC_VECTOR := "00"; --INTEGER := 0;       --
codigo para uma MAU especifica
CONSTANT PKT_REGULAR     : STD_LOGIC_VECTOR := "01"; --INTEGER := 1;       --
codigo a ser executado em qualquer RPU
CONSTANT PKT_INTERRUPTED : STD_LOGIC_VECTOR := "10"; --INTEGER := 2;       --
espera por E/S
CONSTANT PKT_CALLER      : STD_LOGIC_VECTOR := "11"; --INTEGER := 3;       --
espera o fim de uma funcao que chamou


--************************************************
-- CODES TO THE ROUTES
--************************************************
CONSTANT SPIRAL_1 : STD_LOGIC_VECTOR := "000010"; -- INTEGER := 2;
CONSTANT SPIRAL_2 : STD_LOGIC_VECTOR := "000111"; -- INTEGER := 7;
```

```
CONSTANT SPIRAL_3 : STD_LOGIC_VECTOR := "000001"; -- INTEGER := 1;
CONSTANT SPIRAL_4 : STD_LOGIC_VECTOR := "000100"; -- INTEGER := 4;


--************************************
-- OPCODES
--************************************

-- //// ARITMETICAS /////
-- soma
CONSTANT ADD : INTEGER := 0;
-- subtracao
CONSTANT SUBB : INTEGER := 1;
-- multiplicacao
CONSTANT MUL : INTEGER := 2;
-- divisao
CONSTANT DIV : INTEGER := 3;

-- //// LOGICAS /////
-- negacao
CONSTANT NOTT : INTEGER := 4;
-- E logico
CONSTANT ANDD : INTEGER := 5;
-- OU logico
CONSTANT ORR  : INTEGER := 6;
-- OU exclusivo
CONSTANT XORR : INTEGER := 7;

-- deslocamento para direita
CONSTANT RSHIFT : INTEGER := 8;
-- deslocamento para esquerda
CONSTANT LSHIFT : INTEGER := 9;

-- //// ACESSO A MEMORIA /////
-- carrega da memoria
CONSTANT LOAD    : INTEGER := 10;
-- armazena na memoria
CONSTANT STORE   : INTEGER := 11;
-- EXECUTE: ordena a injecao (execucao) de um pacote da memoria no sistema
CONSTANT EXEC    : INTEGER := 12;
-- SINCRONIZED EXECUTE: ordena a injecao de um pacote depois de receber todos
os sinais de sincronizacao
CONSTANT SYNEXEC : INTEGER := 13;
-- confirmacao de sincronizacao, enviado para que o SINEXEC seja executado
CONSTANT SYNC    : INTEGER := 14;
-- retorno do valor carregado pelo LOAD
CONSTANT RELOAD  : INTEGER := 15;

-- //// COMANDO CONDICIONAL /////
```

```
-- BRANCH ON EQUAL: desvia o fluxo se os valores comparados sao iguais
CONSTANT BE   : INTEGER := 16;
-- BRANCH ON NOT EQUAL: desvia o fluxo se os valores comparados sao diferentes
CONSTANT BNE  : INTEGER := 17;
-- BRANCH ON LESS: desvia o fluxo se o 1o. valor comparado é menor que o 2o.
CONSTANT BL   : INTEGER := 18;
-- BRANCH ON GREATER: desvia o fluxo se o 1o. valor comparado é maior que o
2o.
CONSTANT BG   : INTEGER := 19;
-- BRANCH ON LESS OR EQUAL: desvia o fluxo se o 1o. valor comparado é menor ou
igual ao 2o.
CONSTANT BLE  : INTEGER := 20;
-- BRANCH ON GREATER OR EQUAL: desvia o fluxo se o 1o. valor comparado é maior
ou igual ao 2o.
CONSTANT BGE  : INTEGER := 21;
-- INCONDITIONAL BRANCH: desvia o fluxo incondicionalmente
CONSTANT JUMP : INTEGER := 22;

-- //// AUXILIARES /////
-- copia um valor para outra palavra do pacote
CONSTANT COPY : INTEGER := 23;
-- Nao faz nada
CONSTANT NOP  : INTEGER := 24;
-- envia um valor para ser usado em um pacote que sera carregado da memoria
CONSTANT SEND : INTEGER := 25;

-- //// ENTRADA E SAIDA /////
-- Entrada
CONSTANT INN  : INTEGER := 26;
-- Saida
CONSTANT OUTT : INTEGER := 27;

-- //// INTERRUPCAO /////
-- Reinjetar um pacote que foi interrompido para esperar E/S
CONSTANT WAKEUP : INTEGER := 28;

-- //// NOTIFICACAO /////
-- Envia uma notificacao. Ex: aconteceu uma interrupcao e um pacote foi
armazenado em outra MAU
CONSTANT NOTIFY : INTEGER := 29;

-- //// CHAMADA DE FUNCAO /////
-- Chama a funcao
CONSTANT CALL : INTEGER := 30;
-- Retorna da funcao
CONSTANT RETURNN : INTEGER := 31;


--**********************************
```

```
-- General Values
--************************************
CONSTANT WORD       : INTEGER := 36;          -- Word Size (in bits) =
control(4)+word(32)
CONSTANT WORD_BYTES : INTEGER := (WORD/8);    -- Bytes in a Word
CONSTANT ADDR       : INTEGER := LEN_SOURCE;  -- Address Size (in bits)
CONSTANT SIZE       : INTEGER := 32;          -- Size of the buffer
CONSTANT EOP        : STD_LOGIC_VECTOR := "00000000000000000000000000000000";
--INTEGER := 0;           -- value of the end of packet
CONSTANT WL         : INTEGER := WORD;
CONSTANT IWL        : INTEGER := 20;


--************************************
--NoC Values
--************************************
CONSTANT PCK_WIDTH : INTEGER := WORD;
CONSTANT P_NORTH   : INTEGER := 0;
CONSTANT P_SOUTH   : INTEGER := 1;
CONSTANT P_WEST    : INTEGER := 2;
CONSTANT P_EAST    : INTEGER := 3;
CONSTANT P_MOD     : INTEGER := 4;  -- local port - used to store control
packet buit by SU

CONSTANT TORUS     : INTEGER := 0;


--************************************
-- ObTree Values
--************************************
CONSTANT BUFFER_ADDR_WIDTH : INTEGER := 8;
CONSTANT FCFS              : INTEGER := 1;  --First Come First Served
CONSTANT ARBITRATION       : INTEGER := FCFS;


--************************************
-- Mesh Values
--************************************
CONSTANT ROUTER_PORTS : INTEGER := 4;


--************************************
-- VHDL Values
--************************************
CONSTANT FIFO_TYPE  : STRING  := "RING";
CONSTANT DEPTH      : INTEGER := 4; -- number of positions
CONSTANT LOG2_DEPTH : INTEGER := 2; -- log2 of the number of positions

END ipnosys_package;
```