PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
ENGENHARIA DE COMPUTAÇÃO

PROJETO DE ROTEADOR PARA REDE INTRA-CHIP UTILIZADO A ARQUITETURA ROUNDABOUT

HENRIQUE MARTINS MEDINA

Monografia apresentada como requisito parcial à obtenção do grau de Engenheiro de Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre
2019

PROJETO DE ROTEADOR PARA REDE INTRA-CHIP UTILIZADO A ARQUITETURA ROUNDABOUT

RESUMO

Atualmente as interconexões podem limitar a desempenho dos sistemas digitais. Nesse sentido, redes intra-chip (NoC) representam uma opção a barramentos por duas razões principais: paralelismo na comunicação e escalabilidade. NoCs são compostas por roteadores e enlaces. Roteadores em sua maioria composta por uma arquitetura baseada em buffers de entrada. Estes buffers são responsáveis pela maior parte do consumo de energia dos roteadores, e na maioria das vezes são subutilizados, dado que muito raramente há fluxos de pacotes cruzando todos os buffers. NoC com arquitetura *roundabout*, inspiradas nas rotatórias de trânsito são uma proposta que visa diminuir a quantidade de buffer subutilizados e aumentar o desempenho da rede. Nesta arquitetura, os recursos do roteador serão compartilhados por demanda. Este TCC tem por objetivo propor uma implementação no nível RTL para a arquitetura de roteador roundabout adicionando novas características.

Palavras-Chave: NoC, infraestrutura de comunicação, arquitetura roundabout, Elastic Buffer .

INTRA-CHIP NETWORK ROUTER DESIGN USING ROUNDABOUT ARCHITECTURE

ABSTRACT

Interconnections may currently limit the performance of digital systems. In this sense, intra-chip (NoC) networks represent an option to buses for two main reasons: parallelism in communication and scalability. NoCs are composed of routers and links. Routers mostly consist of an architecture based on input buffers. These buffers are responsible for most of the power consumption of the routers, and most of the time they are underutilized, since very rarely there are packet flows crossing all the buffers. NoC with *roundabout* architecture, inspired by the traffic roundabouts are a proposal that aims to decrease the amount of underutilized buffer and increase network performance. On this architecture, router resources will be shared on demand. This TCC aims to propose an RTL-level implementation for the roundabout router architecture by adding new features.

Keywords: NOC, communication infrastructure, roundabout architecture.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Atualmente, sistemas digitais são onipresentes na sociedade. Computadores e *smartphones* estão em diversas tarefas do cotidiano, ajudando na realização de simulações, gerência de banco de dados, produção de documentos, entretenimento e em outras atividades [Dally and Towles, 2003].

De acordo com [Dally and Towles, 2003], um sistema digital pode ser dividido em três blocos básicos, sendo eles: lógica, memória e comunicação. A lógica é responsável pela transformação e combinação de dados, a memória por armazenar informação para eventual uso, e por último a comunicação tem o papel de mover dados entre elementos de processamento, regiões de memorias e armazenamento externo. Esses sistemas deram origem às CPUs e Circuitos Integrados (CI) da atualidade, possibilitando a execução das tarefas cotidianas.

Contudo, por mais que o desempenho de tais componentes seja extraordinário, as interconexões podem limitar o desempenho de tais sistemas. Grande parte da energia consumida por sistemas de ponta é devido à atividade de chaveamento nos fios das interconexões. Este problema acentua-se com a quantidade das interconexões, principalmente em estruturas como barramento [Dally and Towles, 2003].

Desta forma, o presente trabalho busca estudar estruturas de interconexão do tipo NoC (*network-on-chip*), que reduzem o comprimento de interconexões, e melhoram o desempenho de sistemas digitais complexos. Em particular, o foco do estudo é em arquitetura de roteador inspirada em uma rotatória de veículos (*roundabout* NoC).

1.1 Motivação

A motivação para o desenvolvimento desse trabalho origina dos conhecimentos adquiridos durante o curso. Em particular, da experiência obtida com MPSoCs (*Multiprocessor Sytem-on-Chip*) ao longo de bolsas de iniciação cientifica. Um MPSoC é um sistema multi-processador (*many-core*), que possui vários elementos de processamentos (PE). Os PEs são conectados através de uma NoC, com o propósito de trocar dados necessários para concluir tarefas.

Os trabalhos [Ruaro et al., 2017a, Ruaro et al., 2017b, Oliveira et al., 2018a, Oliveira et al., 2018b, Sant'ana et al., 2019] forneceram uma visão de arquiteturas MPSoC, e seu protocolos de comunicação. Sendo assim, a oportunidade de continuar o projeto de doutorado de Charles Effiong [Effiong, 2018], na área de NoCs proporciona uma chance de aplicar o conhecimento adquirido ao longo do curso, enquanto traz novos desafios de implementação.

1.2 Objetivos

Este TCC tem por objetivo geral propor uma revisão da arquitetura do roteador *roundabout*, proposta originalmente por Charles Effiong em seu Doutorado [Effiong, 2018]. Nesse sentido, a nova implementação pretende agregar novas características a rede intra-chip, tornando-a mais dinâmica e parametrizável.

Os objetivos específicos para atingir o objetivo geral incluem:

1. Estudar a Tese de Charles Effiong [Effiong, 2018], compreendendo o trabalho realizado, as dificuldades enfrentadas, e os resultados obtidos;
2. Analisar a implementação da NoC e os resultados apresentados em [Effiong et al., 2017];
3. Implementar um roteador baseado na proposta *roundabout*, com otimizações como quantidade de buffers utilizados, avaliando seu desempenho.

1.3 Organização do documento

O presente TCC está organizado em 4 Capítulos. Este Capítulo apresentou a motivação e os objetivos do presente TCC. O Capítulo 2 apresenta conceitos básicos de redes intra-chip (NoCs). O Capítulo 3 apresenta trabalhos relacionados a este TCC, em particular o conceito de buffers elásticos e a arquitetura *roundabout*. O Capítulo 4 apresenta a implementação proposta neste TCC, e finalmente no Capítulo 5 conclui este trabalho.

2. CONCEITOS DE REDES INTRA-CHIP

Estre Capítulo apresenta conceitos relacionados a redes Intra-chip (NoC). Para isso, utilizou-se o livro “On-Chip Communication Architectures: System on Chip Interconnect”, escrito por Sudeep Pasricha e Nikil Dutt como base para o conteúdo deste Capítulo [Pasricha and Dutt, 2010].

Sendo assim, de acordo com [Pasricha and Dutt, 2010], NoC é um modelo de comunicação baseado em conceitos de redes de computadores. Esse modelo é aplicado em sistemas multiprocessados em chip com a intenção de substituir a comunicação por barramentos por uma baseada em pacotes. Tais pacotes são formados por datagramas utilizados para rotear os dados da sua origem até o seu destino.

2.1 Topologia

Topologias de NoCs são classificadas em redes diretas, indiretas e irregulares. No presente trabalho utilizou-se apenas as redes diretas, visto que essa topologia fornece conexões ortogonais, entre os roteadores, torna o roteamento simples, de fácil implementação e eficiente. A Figura 2.1 apresenta uma topologia malha bidimensional (*mesh* 2-D). Esta topologia é uma das mais populares, pois os enlaces possuem o mesmo comprimento, havendo pesquisas mostrando que esta topologia apresenta baixa latência, possui baixo consumo de energia e é de fácil implementação, comparando a outras topologias [Kaushal and Singh, 2014].

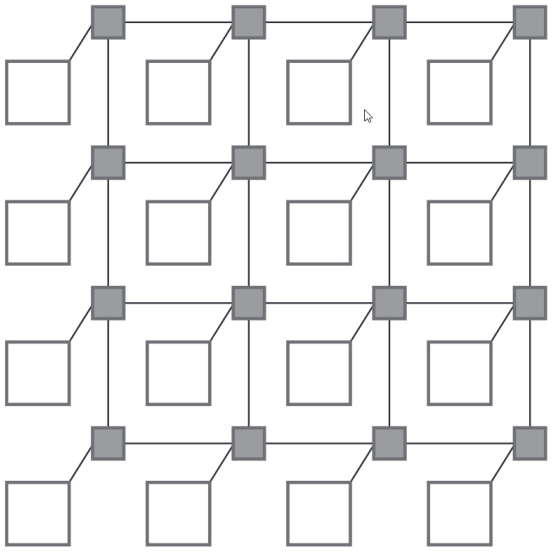

Figura 2.1 – Exemplo de uma NoC interconectada por uma *mesh* 2-D. [Pasricha and Dutt, 2010].

2.2 Estratégias de Chaveamento

O chaveamento é usado para determinar como os dados passam pelos roteadores e qual será a granularidade da transferência de dados. Diante disto, PEs geram mensagens que são divididas em vários pacotes de dados (*packets*), como apresentado na Figura 2.2.

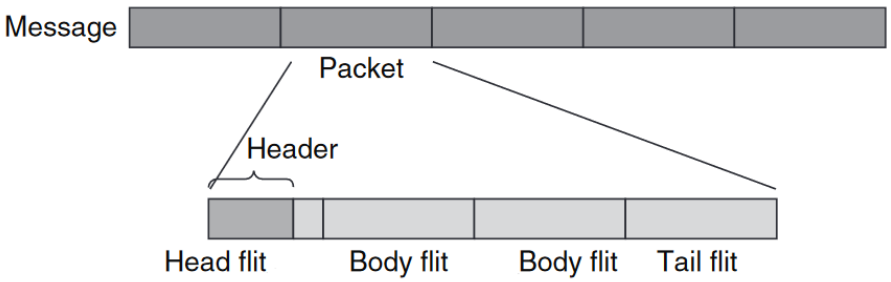

Figura 2.2 – Estrutura de Mensagem. [Pasricha and Dutt, 2010]

.

Os pacotes são subdivididos em flits, que é uma unidade de controle de fluxo, usada no roteamento do pacote e na sua restauração no PE de destino.

Para que os flits sejam transmitidos corretamente entre os roteadores empregam-se métodos de controle de fluxo, como disponibilidade de espaço no buffer vizinho (*créditos*) ou *handshake*.

Uma característica dos roteadores é sua largura de banda. Este parâmetro informa a quantidade de bits que a rede pode transmitir em um ciclo de relógio (unidade de tempo usada para sincronizar hardware). A largura de banda é função da frequência de operação da NoC e da largura (em bits) dos flits. Ambos parâmetros afetam o custo da NoC, em termos de desempenho e energia consumida.

Os dois modos principais de transporte de flits são circuit-switching e packet-switching.

2.2.1 Circuit-Switching

No circuit-switching (CS), um caminho físico entre a origem da mensagem e o destino deve ser previamente reservado antes da transmissão dos dados. Esse processo ocorre através da alocação de uma sequência de enlaces, que são as conexões entre roteadores. Após estabelecer a conexão, os pacotes podem ser enviados, e a utilização dos enlaces alocados fica bloqueada, até que o PE origem inicie o procedimento de desconexão do caminho.

2.2.2 Packet Switching

Neste modo não há reserva prévia de enlaces. Sendo assim um roteador no packet switching (PS) exige a utilização de um esquema de chaveamento, que define a forma como os pacotes passam através dos roteadores. Em outras palavras, para CS a conexão é estabelecida e mantida enquanto a origem dos dados desejar. Para PS, a conexão é realizada a cada pacote.

Existem três modos de encaminhamento de dados quando se utiliza PS:

1. *Store and Forward* – SAF: um pacote é transmitido de um roteador ao seguinte somente se este tiver espaço no buffer para o pacote inteiro. Os roteadores só encaminham o pacote após ele ter sido completamente recebido.
2. *Virtual Cut Through* – VCT: o primeiro flit de um pacote é enviado assim que houver espaço para o pacote inteiro no próximo buffer. Em uma comparação com SAF, VCT reduz a latência de encaminhamento do pacote.
3. *Wormhole* – WH: nesta técnica, os requisitos de buffer são reduzidos. Um flit de um pacote é encaminhado para o roteador receptor desde que haja espaço para este flit no roteador receptor.

2.3 Algoritmos de Roteamento

Os algoritmos de roteamento podem ser classificados segundo vários critérios, em diferentes categorias, como roteamento estático ou dinâmico, roteamento distribuído ou de origem e roteamento mínimo, ou não mínimo.

2.3.1 Roteamento estático ou dinâmico

No roteamento estático, caminhos pré-determinados são utilizados para transferir dados entre dois roteadores. Uma vantagem deste roteamento é a sua facilidade de implementação, pois requer pouca lógica no roteador.

O roteamento estático também permite que os pacotes sejam divididos entre múltiplos caminhos entre uma origem e um destino, de maneira pré-determinada. Se apenas um caminho for usado, o roteamento estático geralmente garante a entrega de pacotes de dados em ordem, eliminando a necessidade de adicionar bits aos pacotes, para identificá-los corretamente e reordená-los no destino, se necessário. Este comportamento adaptativo, no

entanto, custa recursos adicionais para monitorar continuamente o estado da rede e alteram dinamicamente os caminhos de roteamento.

O roteamento dinâmico pode distribuir melhor o tráfego em uma rede, e consegue utilizar caminhos alternativos quando certos trajetos ficam congestionados.

2.3.2 Roteamento distribuído ou na origem

No roteamento distribuído, cada pacote carrega o endereço de destino. Quando um pacote chega ao roteador, o destino é extraído do *header* flit e as decisões de roteamento são tomadas, podendo ser através de uma pesquisa em uma tabela de roteamento ou executando uma função de hardware.

Quando a decisão de roteamento é tomada na origem, o roteador ou PE origem, define a rota e insere esta no cabeçalho do pacote, com base no endereço de destino. Cada pacote carrega as opções de roteamento em seu cabeçalho para cada salto em seu caminho. Diferentemente do roteamento distribuído, o roteamento na origem não requer um endereço de destino em seu pacote, e nenhuma tabela de roteamento intermediária ou funções são necessárias para calcular a rota.

2.3.3 Roteamento mínimo ou não mínimo

Um roteamento é mínimo se o comprimento do caminho de roteamento da origem até o destino for o menor possível entre os dois PEs. Este comprimento mínimo é a distância Manhattan entre a origem e o destino do pacote, aplicada somente em redes utilizando *mesh*. [Pasricha and Dutt, 2010]

Um algoritmo de roteamento não mínimo não possui essas restrições e pode usar caminhos mais longos se um caminho mínimo não estiver disponível. Ao permitir caminhos não mínimos, o número de caminhos alternativos aumenta, o que pode ser útil por um lado para evitar congestionamentos, mas, por outro lado pode levar a *livelock* (pacote percorrendo a NoC indefinidamente sem chegar ao seu destino).

2.3.4 Algoritmo de roteamento XY

No presente trabalho utilizou-se o algoritmo de roteamento XY. Esse algoritmo é classificado como estático, distribuído e mínimo. Os flits são roteados inicialmente ao longo do eixo X até alcançarem a coluna do roteador destino. Posteriormente os flits são

roteados no eixo Y. Quando o endereço do roteador for igual ao endereço de destino, o flit é encaminhado para a porta local.

Se a porta de saída escolhida estiver ocupada, o fluxo de flits do pacote é bloqueado. A solicitação de roteamento para este pacote permanecerá ativa até a liberação da porta de saída [Kaushal and Singh, 2014].

3. TRABALHOS RELACIONADOS

Este Capítulo apresenta 2 trabalhos que o presente TCC usa de base para a implementação do roteador *roundabout*.

3.1 Elastic-Buffer

Os *Elastic Buffers* (EB) permitem reduzir a potência e a área em reteadores devido aos buffers, utilizando um projeto baseado latch [Michelogiannakis and Dally, 2011] e [Michelogiannakis et al., 2010], como apresentado na Figura 3.1. Os buffers de entrada normalmente empregados nos roteadores podem ser substituídos por EBs diretamente nos enlaces, formando um pipeline distribuído entre reoteadores vizinhos.

Os EB são implementados por uma latch mestre e uma escravo, controlada pela lógica de controle EB. A latch mestre é habilitada quando o clock está baixo, e a latch escravo quando o clock está alto.

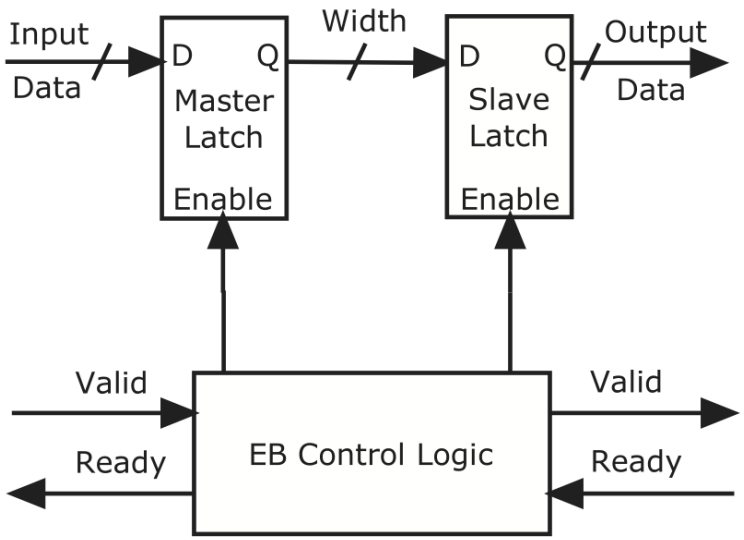

Figura 3.1 – Elastic-Buffer(EBs). Fonte: [Michelogiannakis and Dally, 2011].

Os autores detalham a operação dos EBs. Eles usam um mecanismo de *handshake* para validar a comunicação, usando os sinais *ready* e *valid*. O sinal de *ready* indica que o EB possui pelo menos uma latch para armazenar o flit. O sinal *valid* indica que o flit atualmente armazenado é válido para consumo.

3.2 Distributed and Dynamic Shared-Buffer Router for High-Performance Interconnect

O artigo de Effiong et al. [Effiong et al., 2017], traz a implementação de um roteador baseado em *elastic buffers*, que promete reduzir em 80% a área e simplificar o projeto da implementação quando comparado com uma implementação síncrona de roteador.

3.2.1 A Arquitetura Roundabout

Roundabout é o nome dado por Effiong et al. para a sua arquitetura de roteador. Essa designação surge devido à inspiração de implementar um roteador contendo múltiplas faixas como se fosse uma rotatória de carros.

A Figura 3.2 exemplifica o modo de operação da arquitetura *roundabout*. O roteador é particionado em faixas (*lanes*) primárias e secundárias. Na Figura 3.2, as *lane* 0 e 1 são denominadas primárias, enquanto a 2 e a 3 são as secundárias. A diferença entre essas *lanes* dá-se na distribuição das portas de entrada, ou seja, entradas só tem acesso ao roteador através das *lanes* primária, *lanes* 0 e 1, enquanto as secundárias são utilizadas apenas quando ocorre congestionamento de uma *lane* primária pois possuem prioridade para acessar recursos compartilhados.

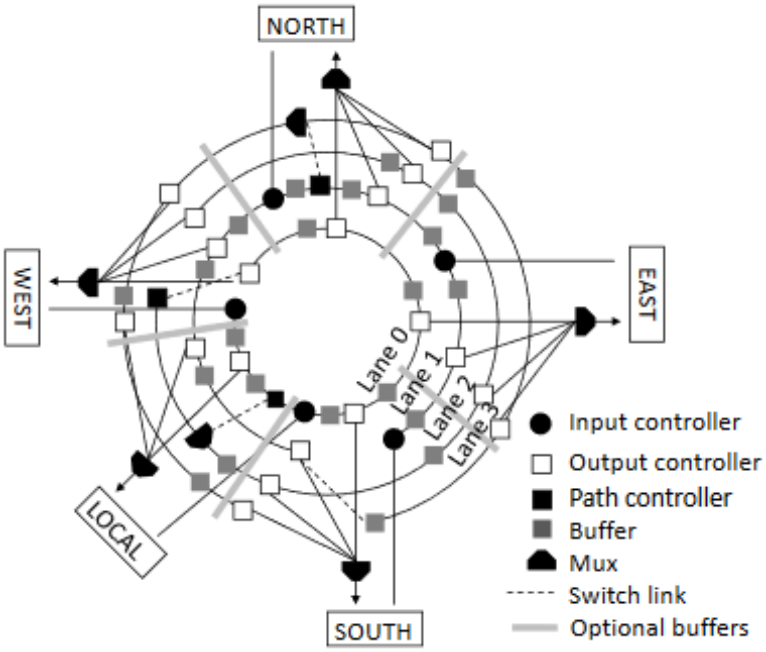

Figura 3.2 – Arquitetura Roundabout [Effiong et al., 2017].

.

Na arquitetura da Figura 3.2, as portas de entrada são distribuídas da seguinte maneira. A *lane* 0 recebe as portas de entrada local (L) e oeste (W), podendo trocar para a *lane* secundária 2, enquanto a *lane* 1 recebe as portas leste (E), sul (S) e norte (N), podendo trocar para a *lane* 3. Essa distribuição das portas de entrada foi realizada a partir do algoritmo de roteamento XY, levando em consideração as portas de saída às quais as portas de entrada podem se conectar, por exemplo, um pacote oriundo da porta norte tende a ir para as portas sul ou local. Com isto, coloca-se na mesma *lane* a porta de entrada norte e as portas de saída sul e local.

Quando um pacote que trafega em uma faixa primária é bloqueado ou não recebe acesso à porta de saída, ele alterna para uma faixa secundária através de um multiplexador. A Figura 3.3(b) ilustra um dado entrando pela porta leste (E - *east*), destinado à porta de saída local (L). O pacote avança pela *lane* até um ponto onde ocorre um bloqueio devido a outro pacote que está usando a mesma lane. Ao perceber esse congestionamento, o roteador envia o pacote bloqueado para uma *lane* secundária. Este procedimento evita que pacotes sejam enfileirados em uma determinada faixa sempre que vários fluxos competirem pelos recursos de caminho mais curto.

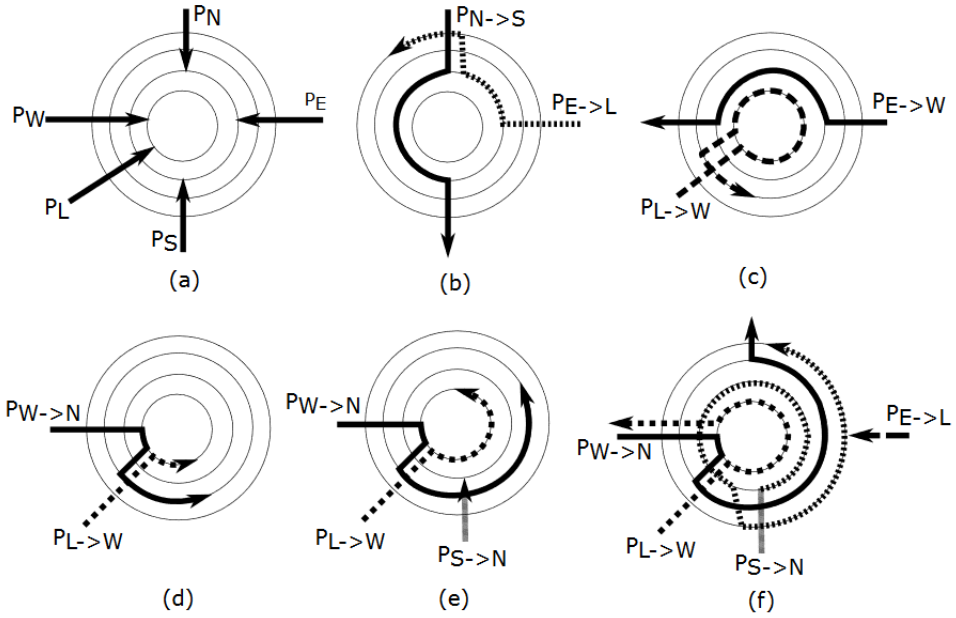

Figura 3.3 – Cenários de fluxo de pacotes [Effiong et al., 2017].

.

Na figura 3.3 podemos observar outros cenários de fluxo. A Figura 3.3(a) representa as entadas dos pacotes nas lanes; (c) troca de *lane* quando a porta de saída desejada está ocupada; (d) cenário onde um pacote da porta local é acessa do *lane* primária, obrigando o pacote da west usar a *lane* secundária; (e)-(f) outros recursos das *lanes* estão sendo usados.

3.2.2 Implementação

Para implementar o roteador Roundabout, o autor utilizou *handshaking* entre blocos divididos em *lanes*, configurados em uma lista FIFO que usa de EB pra controle de fluxo.

O formato de pacote adotado consiste em um pacote de 32 bits, contendo a informação de inicio de pacote (BoP) e de fim de pacote (EoP). Isso permite alocar EBs durante o percurso do pacote, e liberar ao chegar no final do pacote.

Na Figura 3.4, é apresentada a implementação da *lane* 0, desenvolvida pelo autor. A implementação das demais *lanes* seguem a mesma infraestrutura. Os flits chegam ao " input controller", e são encaminhados para o bloco de " Path Computation". Esse bloco é responsável por computar a porta de saída do pacote usando o roteamento XY. Os outros flits do pacote seguem o caminho reservado para o header.

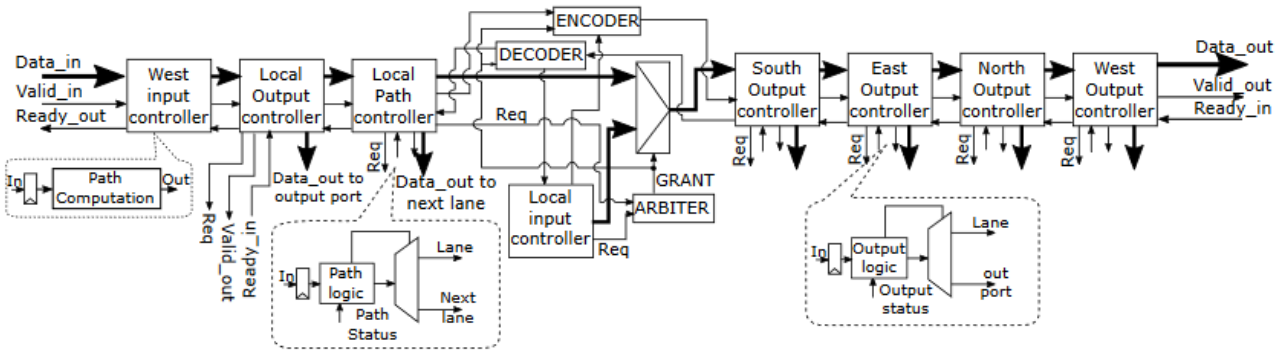

Figura 3.4 – Microarquitetura da *lane* 0 do Roundabout [Effiong et al., 2017].

Na Figura 3.4, são apresentados os outros dois blocos lógicos da arquitetura, sendo eles o "Path Controller" e o "Output Controller". O "Path Controller" é utilizado quando há a necessidade de trocar de lane. Dessa forma, verificado se o caminho principal está livre, e com base nisso, o bloco toma a decisão de trocar de *lane* ou não. O "Output Controller" tem o papel de fazer a interface com uma porta de saída. Quando o flit de header chega neste módulo são verificadas as informações contidas em seu interior, e com isso, decide-se se o pacote vai continuar pela *lane* ou sairá do reteador.

3.3 Observações Finais

A implementação de Charles Effiong traz uma ideia interessante de arquitetura de roteador, no entanto, a implementação não é tão eficiente, como podemos notar nos resultados do próprio artigo.

No artigo, é realizado comparações com o roteador Hermes [Moraes et al., 2004]. Nessas comparações o próprio autor informa que o desempenho do roudabout C0, a estrutura *baseline* do roteador com 4 *lanes*, 2 primárias e 2 secundárias, é inferior ao desempenho do roteador Hermes, que oferece melhor taxa de transferência quando ocorre a saturação da rede. Esse resultado pode ser observado na figura 3.5.

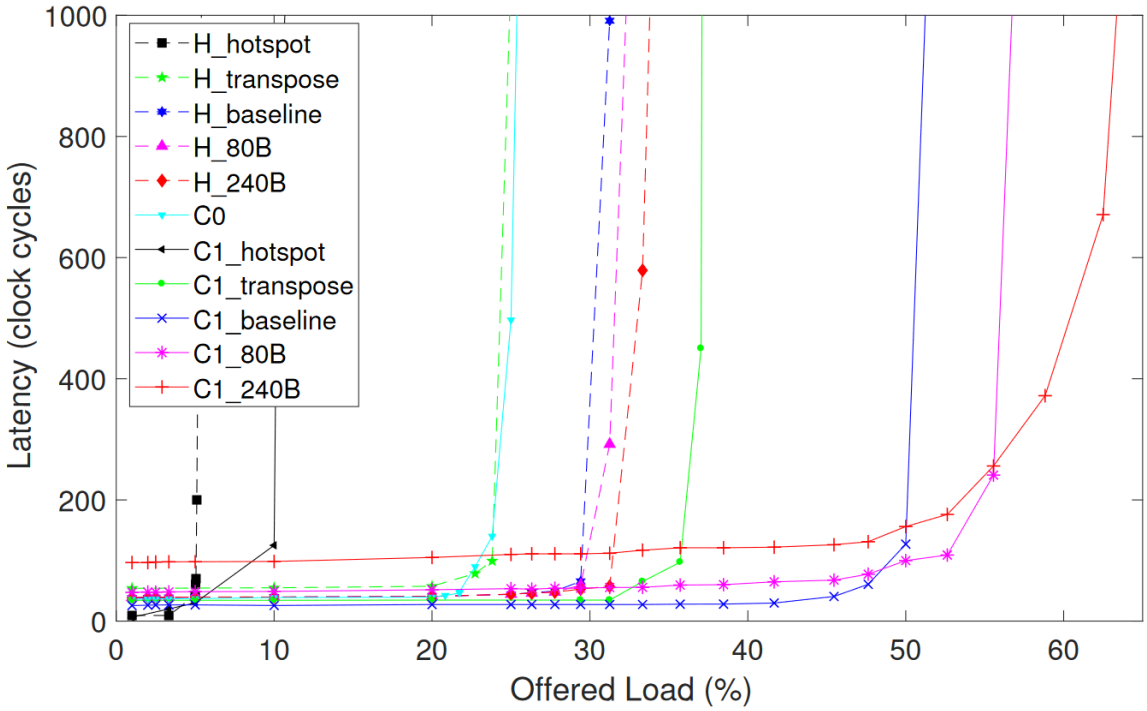

Figura 3.5 – Comparação de desempenho entre e o roteador Roundabout e Hermes. XB denota roteador *baseline* com X *buffers* adicionais [Effiong et al., 2017]

Charles Effiong afirma que o motivo desse queda de desempenho é devido à inabilidade do roteador Roudabout de suportar vários fluxos de dados simultâneos, decorrente das disputas por recursos que pode ocorrer nas *lanes* do roteador, devido aos diferentes fluxos observado na Figura 3.3.

Isso motivou o Autor a pesquisar uma solução, que foi a adição de mais *lanes* no roteador. O número de *lanes* subiu para 9, e essas foram distribuídas em diferentes combinações de *lanes* primária e secundárias, além de diferentes níveis de paralelismo. A Tabela 3.1 mostra quais foram as configurações usadas pelo autor.

Quando ocorre o fluxo apresentado na Figura 3.3(d), a *lane* principal fica reservada devido a um pacote da porta local, obrigando o pacote da porta oeste a trocar de lane. A Figura 3.6 apresenta a implementação das *lanes* 0 e 2 do roteador roundabout, onde podemos observar que a arbitragem das *lanes* não tem um papel tão relevante. Quando ocorre o fluxo descrito acima, as *lanes* se comportam como dois papelines individuais, onde ambos possuem as mesmas saídas e aumentam a quantidade de multiplexadores para uma determinada porta.

Tabela 3.1 – Configurações dos roteadores roundabout (P/S denota a razão entre faixas primárias e secundárias. PL denota nível de paralelismo) [Effiong et al., 2017].

Config.	Lane depth	PL	P/S (%)	No. of lanes
C0	2	2	50	4
C1	3	5	56	9
C2	3	4	44	
C3	4	3	33	
C4	3	3	33	
C5	2	3	44	

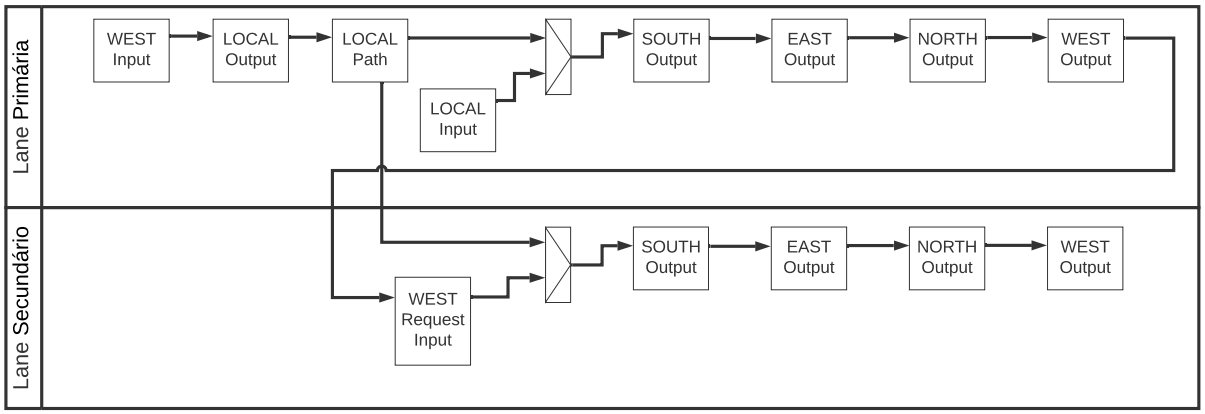

Figura 3.6 – Microarquitetura da *lane* 0 e 2 do roteador roundabout configurado para C0.

4. DESENVOLVIMENTO

Neste Capítulo é apresentada a implementação do roteador *roundabout* (*R-NoC*). O R-NoC é uma implementação de roteador utilizando a ideia proposta em [Effiong, 2018], a fim de acrescentar funcionalidades que não foram exploradas na arquitetura original. Sendo assim, o desenvolvimento parte da ideia de substituir os buffers por elastic-buffer distribuídos em diversas *lanes*, que os pacotes podem percorrer. Diante disso, a implementação da R-NoC pretende trazer uma nova arquitetura para o projeto, alterando mecânicas de arbitragem e conexão entre as lane.

A implementação iniciou-se com a definição da arquitetura do roteador, com o objetivo de avaliar seu comportamento e a distribuição do recursos. A Figura 4.1 mostra a proposta da arquitetura do roteador *R-NoC*. A Figura 4.1 apresenta uma arquitetura com 4 lanes, 5 buffers de chaveamento (formado por *AdaptiveMux* e *elastic-buffer* – retângulos brancos), e duas entradas e saídas para cada porta do roteador. Esses parâmetros podem ser facilmente modificados, tornando a arquitetura parametrizável.

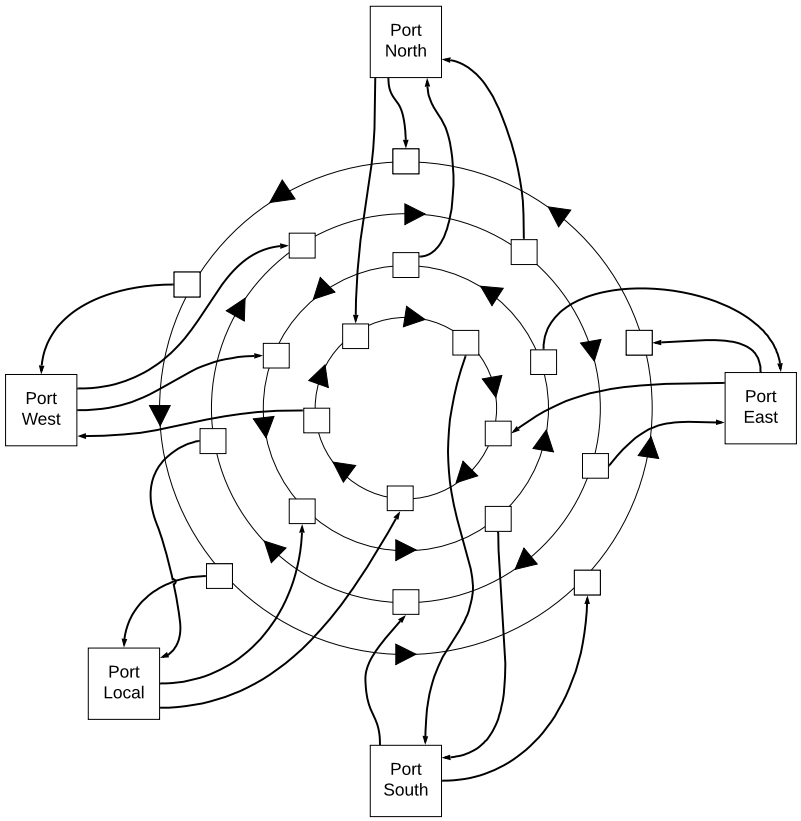

Figura 4.1 – Microarquitetura de uma instância de um roteador para a *R-NoC*.

Uma das ideia propostas na implementação do roteador é a adição de múltiplas *lanes* “girando”em sentidos opostos (setas pretas), essa abordagem está sendo avaliada para propor um caminho mais curto entre uma porta de entrada e a porta de saída do R-NoC.

A Figura 4.2 apresenta o que seriam os “ giros”nas *lanes*. Podemos reparar que a diferença entre uma *lane* girar no sentido anti-horário para o horário é apenas a posição em que as portas se encontram. Assim, temos a percepção que o pacote esta “ girando", pois em determinadas *lanes* o pacote alcança portas diferentes em comparação com outras *lanes*. Isso permite que o pacote entre em uma *lane* e troque de sentido para alcançar uma porta, por um caminho mais curto, evitando cenários em que necessitaria de um giro completo para alcançar a saída.

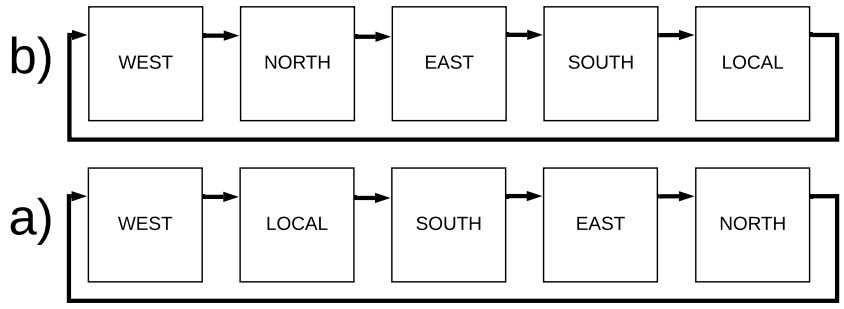

Figura 4.2 – Representação do “ giro”da *lane*, (a) corresponde uma *lane* girando no sentido anti-horário (b) corresponde uma *lane* girando no sentido horário.

4.1 Formato do Pacote

Na *R-NoC*, os pacotes são compostos por um “*header*", usado para reservar o caminho entre os roteadores de origem e destino, seguido pelos flits de dados, e por fim um flit de “*tail*", usado para liberar o caminho reservado. A Figura 4.3 ilustra o formato do pacote utilizado pela *R-NoC*. Note que na Figura 4.3 possuímos um campo denominado *type_flit*. Esse campo é utilizado como um *flag*, sendo ele o responsável por informar quando um pacote inicia ("*header*") e termina ("*tail*"), desta forma, reduzindo a necessidade de acrescentar lógica de controle de “*payload*". Em contrapartida, ele acrescenta 2 bits no pacote.

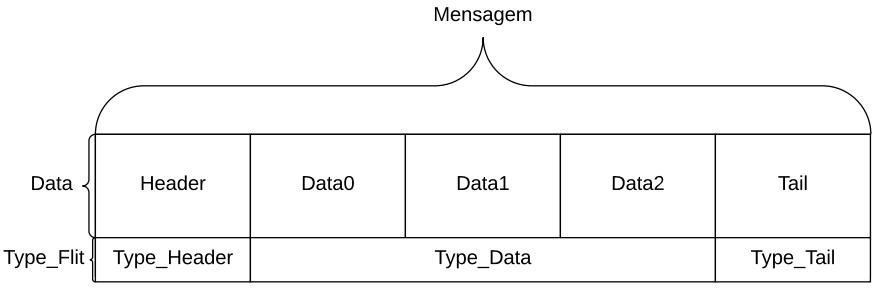

Figura 4.3 – Formato do pacote da *R-NoC*.

4.1.1 Sistema de chaveamento

A *R-NoC* utiliza o algoritmo de roteamento XY para conectar a origem ao destino. Esse algoritmo é executado na interface de entrada do roteador, através de uma lógica combinacional no “Elastic-buffer” de entrada. Essa lógica faz com que os bits mais significativos do header sejam modificados com o propósito de informar às *lanes* a porta de saída para aquele flit. A Figura 4.4 ilustra o flit de cabeçalho modificado, de tal forma que os 5 bits mais significativos indiquem a porta de saída do roteador.

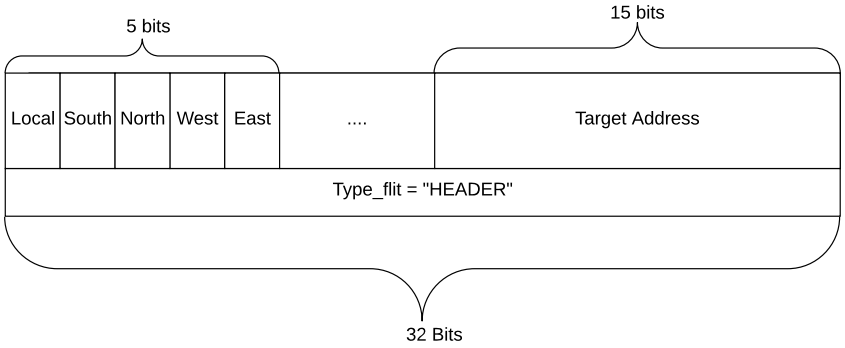

Figura 4.4 – Estrutura do flit de cabeçalho na *R-NoC*.

Com o flit de header configurado, o pacote passa por uma lógica que verifica se alguma das *lanes* conectadas àquela porta possui uma saída válida, ou seja, que corresponde à que foi adicionada no header. Caso tenha uma porta de saída, a interface irá enviar um *request* para a *lane* e aguardará por um retorno. Na Figura 4.5 temos uma representação das interfaces de entrada. Devido à parametrização ela pode ter acesso a múltiplas *lanes*, mas só se comunica com uma de cada vez.

Quando o header alcança a porta de saída é realizado o processo inverso, ou seja, os bits mais significativos são zerados para não influenciar na lógica do próximo roteador. Essa etapa de remoção também ocorre nos EB da interface de saída. A lógica da saída segue o mesmo estilo da entrada, porém espelhada. As portas de entrada podem ter múltiplas saídas na *lane*, porém apenas uma de cada vez pode se comunicar.

4.2 Lanes

As *lanes* são compostas por buffers de chaveamento, que é uma união entre um *AdaptiveMux* e um *elastic-buffer*. Na figura 4.6 é apresentada uma *lane* com 3 buffers de chaveamento.

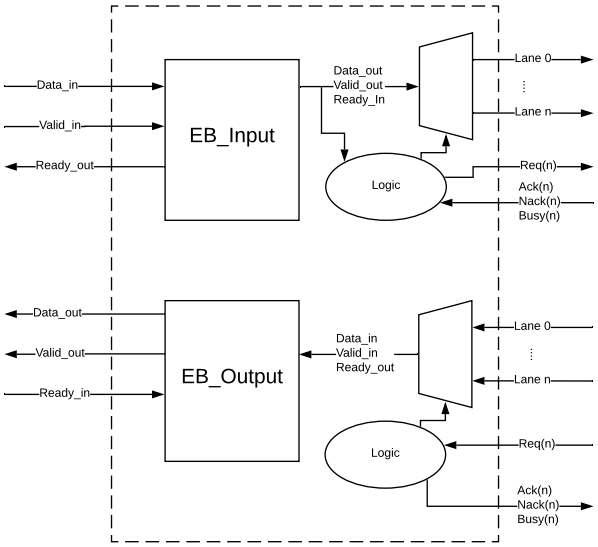

Figura 4.5 – Interface de comunicação do R-NoC.

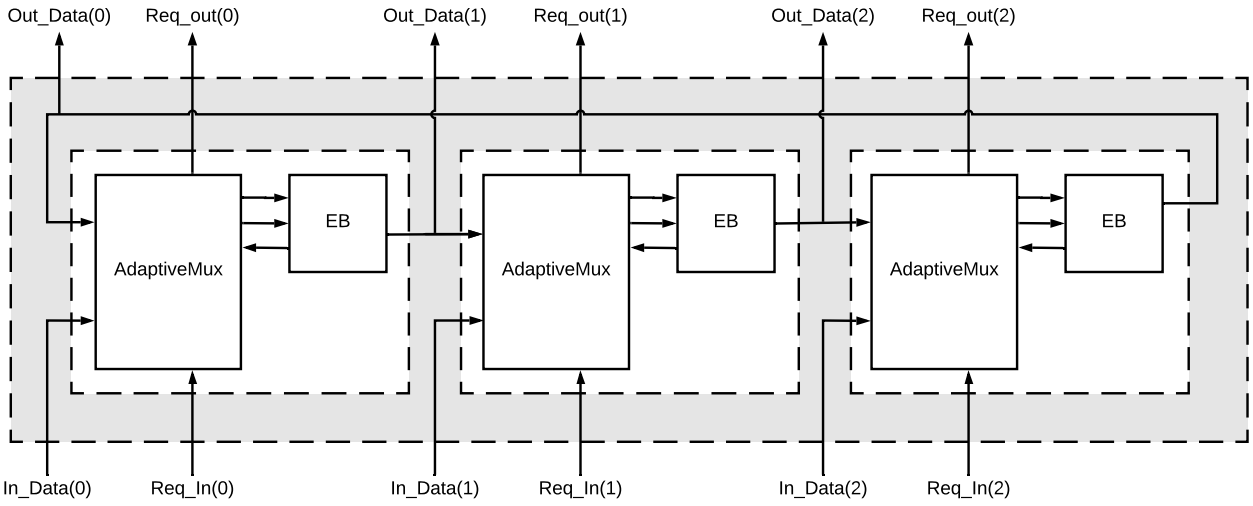

Figura 4.6 – Diagrama de blocos de uma *lane* com 3 buffers de chaveamento.

Cada buffer de chaveamento, possui duas interfaces de entrada, uma destinada a conectar os próprios elementos da *lane*, formando um anel, e a outra utilizada para comunicar-se com outras *lanes* e portas de entrada e saída. A Figura 4.7 detalha a infraestrutura de cada buffer de chaveamento.

4.3 AdaptiveMux

O *AdaptiveMux* é um módulo que implementa a lógica de decisão do roteador. A Figura 4.8 mostra a sua arquitetura. A decisão do roteamento interno, entre as *lanes*, é realizada com base nas informações contidas no header flit.

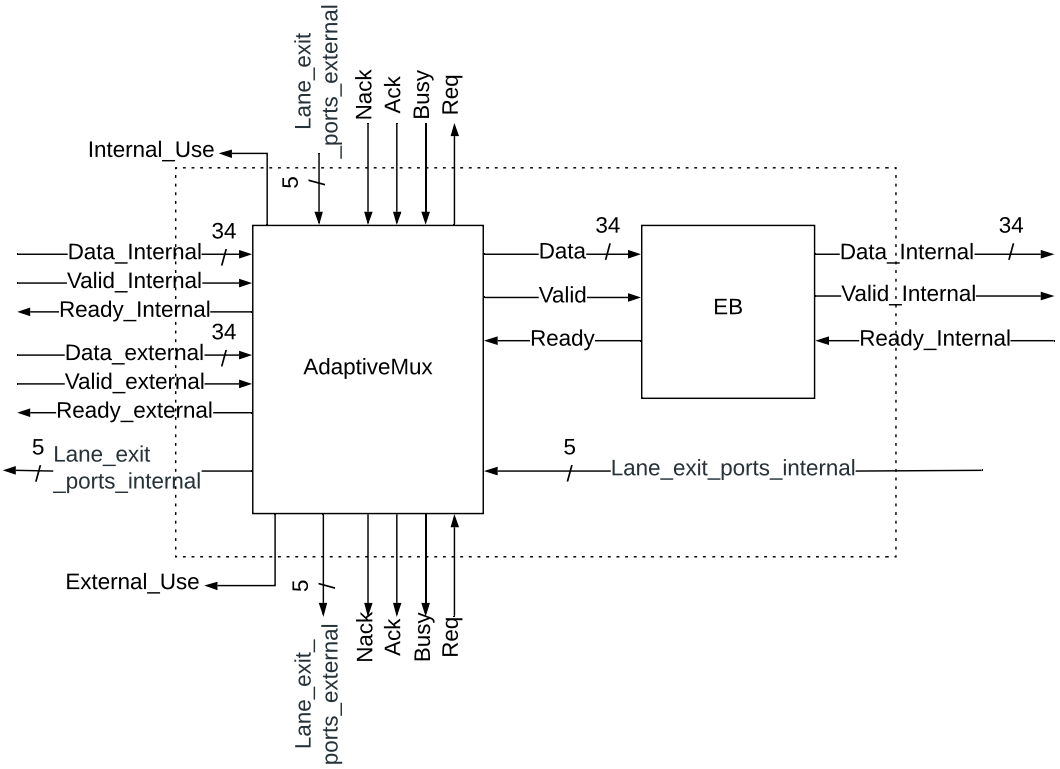

Figura 4.7 – Diagrama de blocos do buffer de chaveamento da *lane*.

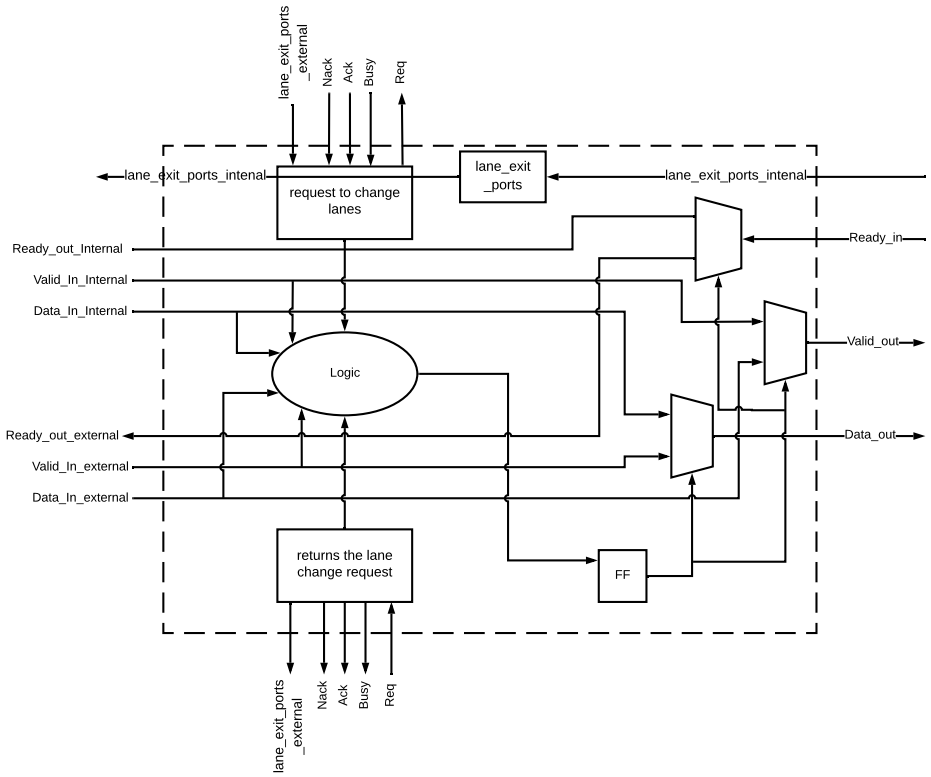

Figura 4.8 – Diagrama de blocos do AdaptiveMux.

A inicialização do AdaptiveMux ocorre em três etapas:

1. Durante o reset, os AdaptiveMux que estão conectados às portas de saída, configuram um registrador chamado de *Lane_exit_ports*. Esse registrador informa qual é a porta de saída que ele está associado, seguindo o padrão onde cada bit do registrador corresponde a uma porta de saída, ou seja, LOCAL, NORTH, SOUTH, WEST e EAST respectivamente.

2. Com o valor do registrador configurado e ainda no reset, os AdaptiveMux trocam essa informação entre si, dentro da lane, até todos os AdaptiveMux terem a mesma informação sobre as portas de saída da lane.

3. Posteriormente a esse processo de sincronização, os AdaptiveMux que tiverem uma comunicação com outras *lanes* compartilham o *Lane_exit_ports*. Isso é necessário para que o pacote que estiver na lane saiba se ele possui uma saída válida, ou se é possível realizar uma troca de lane.

4.4 Decisão de Chaveamento

Com o header do pacote percorrendo a *lane*, pode ser necessário verificar se existe a necessidade de trocar de *lane*. Para isso, é considerada a informação do header com as informações do *Lane_exit_ports* (registrador contendo informação sobre quais são as portas de saída dá *lane*). A Figura 4.9 ilustra a lógica que verifica se determinada lane possui a porta de saída do pacote. Basicamente é realizada uma lógica “and” bit a bit, e verificado se alguma das portas retornaram verdadeiro.

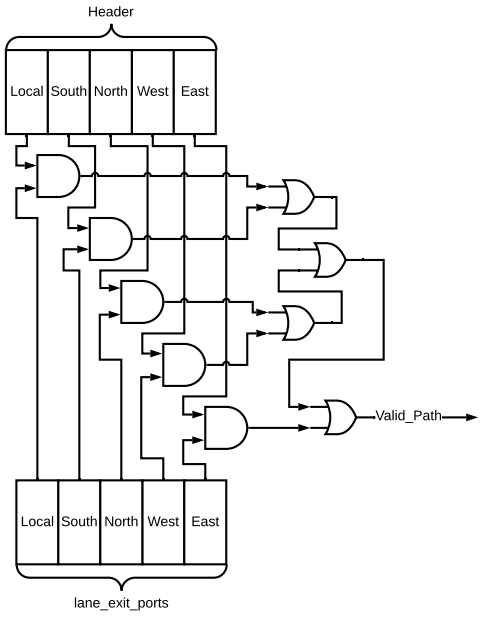

Figura 4.9 – Lógica de verificação caminho válido.

Dessa forma o pacote tende a ir para a *lane* que possui a porta de saída igual à contida no header, e caso nenhuma das possíveis *lanes* tenham essa porta, o flit trocará de *lane* para tentar alcançar a porta de saída através de outra *lena*. O fluxograma da figura 4.10 apresenta o passo a passo de cada interação que o pacote realizará, durante sua percurso entre uma porta de entrada e saída do roteador.

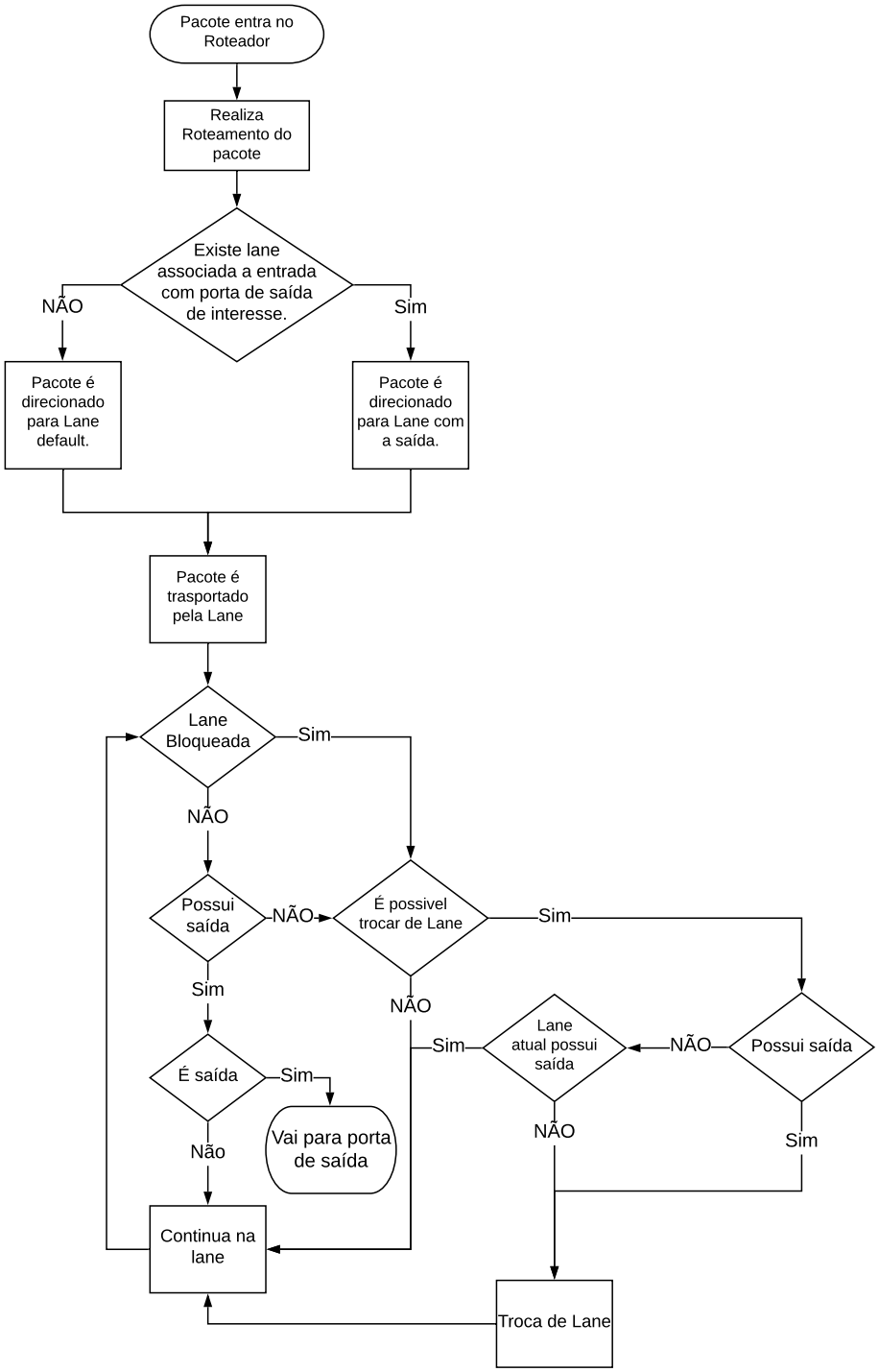

Figura 4.10 – Fluxograma de decisão para um pacote no roteador da R-NoC.

Na *lane*, um flit pode sofrer bloqueios devido a outros pacotes que estejam compartilhando a mesma. Neste caso, o pacote terá a escolha de mudar a *lane*, ou continuar na mesma e aguardar a liberação do caminho, como ilustrado na Figura 4.10. Quando o pacote troca de lane, ele não pode retornar à mesma para evitar *livelock*. A lógica apresentada na Figura 4.9 é usada para encontrar portas válidas entre *lanes*, conjuntamente com uma função para validar se o pacote deve ou não trocar de *lane*.

A Tabela 4.1 apresenta as condições possíveis para que o pacote continue ou troque de *lane*. A coluna “Possui Caminho Interno Válido" e a coluna “Possui Caminho Externo Válido" são geradas a partir da lógica apresentada na Figura 4.9. “Caminho interno bloqueado" e "Caminho externo bloqueado" é um bit que informa se o elastic-buffer associado ao AdaptiveMux está ocupado por outro fluxo de pacote. Às duas últimas colunas informam se o pacote deve seguir na *lane* (Seguir Caminho Interno) ou mudar de *lane* (Seguir Caminho Externo).

Tabela 4.1 – Tabela Verdade do AdaptiveMux

Possui Caminho Interno Válido	Caminho interno bloqueado	Possui Caminho Externo Válido	Caminho externo bloqueado	Seguir Caminho Interno	Seguir Caminho Externo
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	1	1

4.5 Validação

A validação foi realizada a partir de simulações em VHDL. Cada elemento recebeu um test-bench para avaliar a lógica pertencente àquele hardware. Desta forma espera-se ter uma cobertura maior da funcionalidade de cada componente desenvolvido.

A Figura 4.11 apresenta a simulação do *Adaptivemux*. No primeiro retângulo vermelho, temos o pacote percorrendo a *lane*. Esse pacote entra no *Adaptivemux* pela porta interna ("*int_data_i*") e segue para a porta de saída do *Adaptivemux*, continuando o caminho pela *lane*. Isso ocorre porque o pacote possui uma saída válida na *lane* que se encontra. Desta forma não há a necessidade de realizar uma troca de *lane*.

No segundo retângulo vermelho temos um pacote percorrendo a *lane*, e alcançando o *Adaptivemux* pela porta interna. Dado que o pacote não possui uma saída valida pela *lane* atual, o *Adaptivemux* solicita conexão com outra *lane* que possua a saída válida.

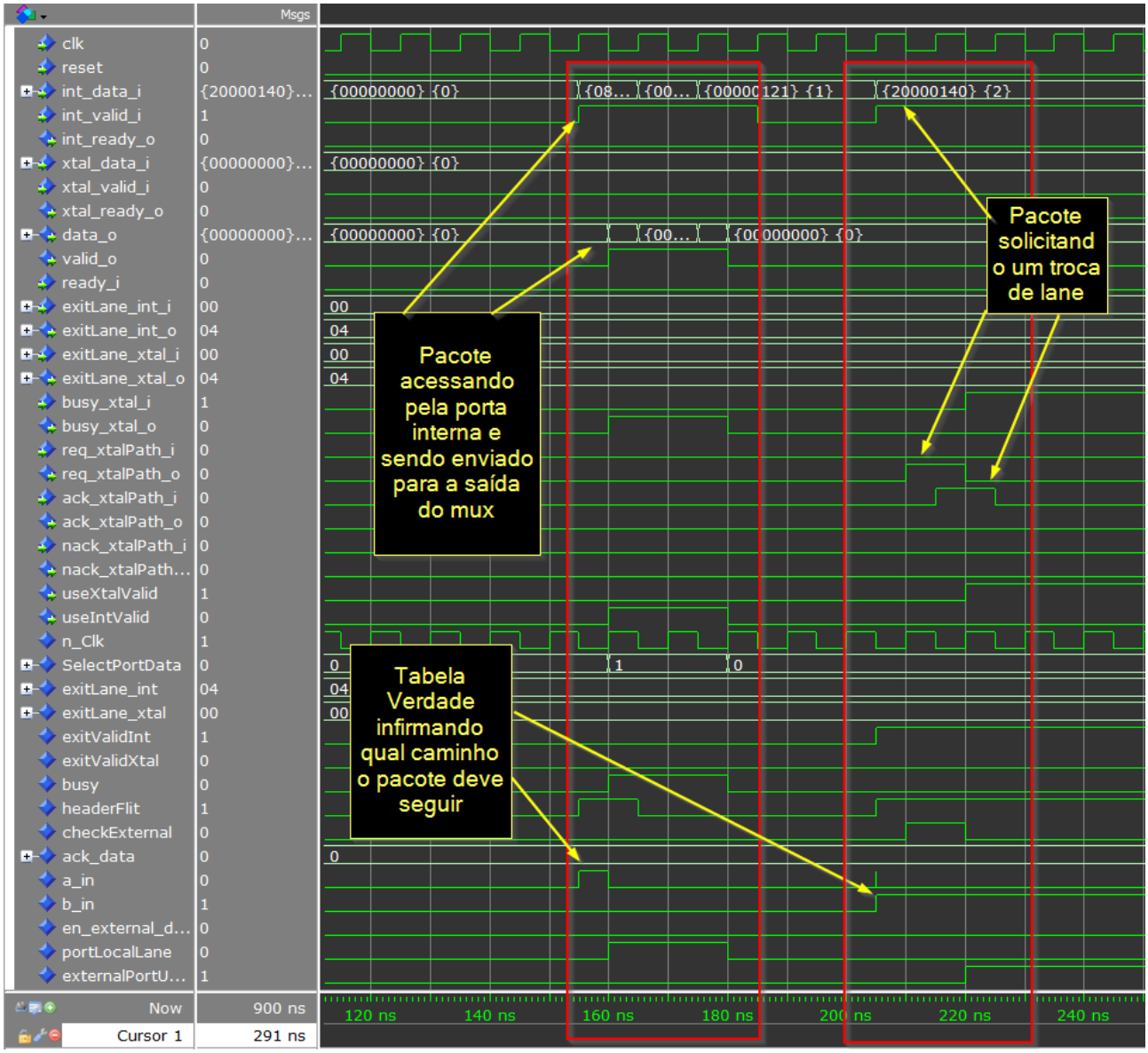

Figura 4.11 – Forma de onda da resposta do AdaptiveMux.

Na Figura 4.12 é testado o caminho interno da *lane*. Neste teste o pacote acessa a *lane* pelo primeiro *Adaptivemux*, e segue avançando pelo caminho até retornar ao ponto de partida. Para evitar deadlock, o *Adaptivemux* move o pacote para outra *lane*, evitando este problema.

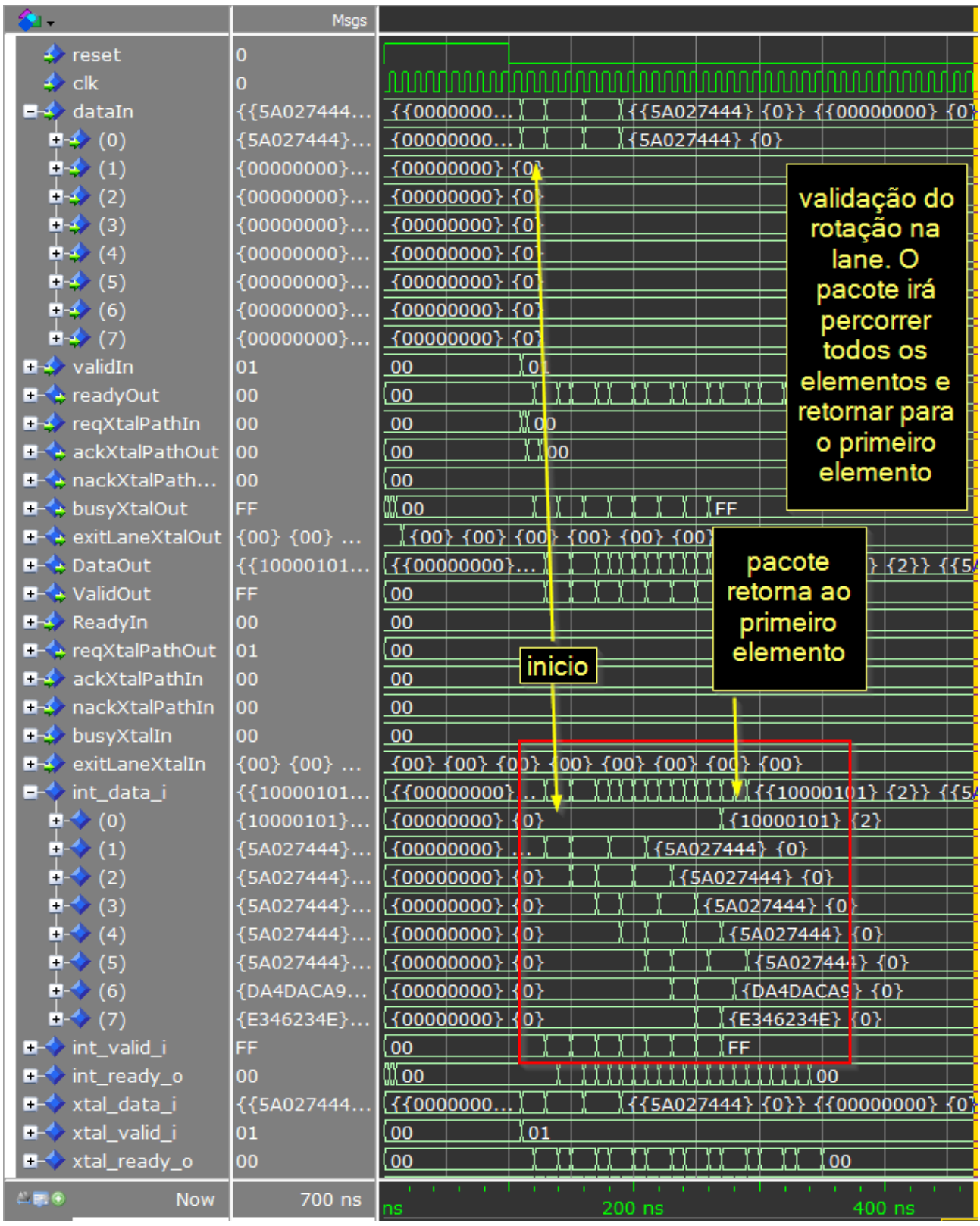

Figura 4.12 – Forma de onda da resposta do Lane.

Na Figura 4.13 apresentamos a simulação completa do roteador. Nesta etapa a simulação se torna mais complexa, devido à quantidade de caminhos possíveis para um

único pacote. Devido à limitação de tempo do TCC, não foi possível realizar uma exploração completa do roteador.

A Figura 4.13 traz um pacote entrando pela porta WEST, e saindo pela porta EAST. Esta simulação corresponde a implementa a arquitetura de ligação das portas de entradas e saída apresenta na figura 4.1.

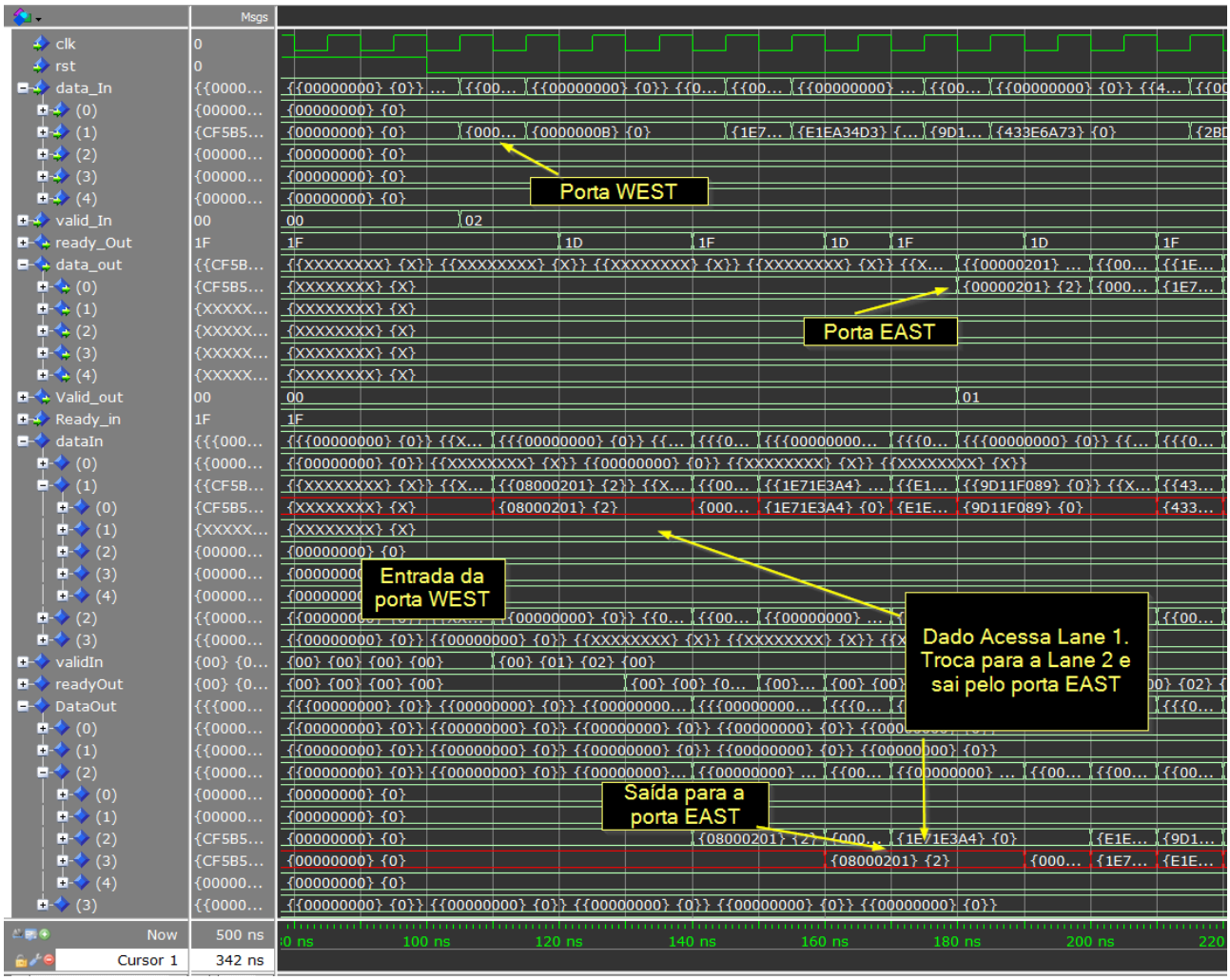

Figura 4.13 – Forma de onda da resposta do Router.

Sendo assim, quando o pacote, chega na porta WEST, ele pode tomar duas decisões possíveis. Na primeira, o pacote escolhe a *lane* 1 como entrada e a segunda opção é o pacote entrar pela *lane* 2. Ambas *lanes* possuem saída para a porta EAST. Na Figura 4.13 observamos que o pacote entra pela *lane* 1 e em seguida realiza uma troca para a *lane* 2. O pacote segue até a porta de saída EAST pela *lane* 2.

Esse comportamento mostra que possuímos algumas falhas na distribuição das portas de entra e saída, e que devem ser sanadas. Por falta de tempo de implementação, estes problemas ainda não foram resolvidos.

5. CONCLUSÃO

O presente TCC complementou o curso de Engenharia de Computação nas disciplinas de sistemas digitais e arquiteturas de computadores. O Autor deste TCC já possuía conhecimento prévio em NoCs, mas não na implementação de roteadores. Assim, o estudo da arquitetura *roundabout* e a correspondente implementação de uma arquitetura similar mostrou-se um grande desafio técnico.

Nesse trabalho foi explorado o conceito de um roteador contendo faixas compartilhadas por várias portas de entrada e saída, de modo a melhorar a utilização de recursos do roteador. A implementação inicial mostrou-se operacional, porém não totalmente validada. A arquitetura do roteador *roundabout* apresenta-se parcialmente validada no nível de *lane*. As etapas seguintes do trabalho correspondem à integração e simulação das *lanes*, para posterior avaliação de uma topologia completa de NoC.

Resultados preliminares da implementação proposta no Capítulo 4, apontam para um consumo de área da R-NoC inferior à rede Hermes em 25%, e o consumo de energia também é reduzido em 31%. Trabalhos futuros também incluem uma avaliação mais precisa de consumo de energia e desempenho deste roteador.

REFERÊNCIAS BIBLIOGRÁFICAS

[Dally and Towles, 2003] Dally, W. J. and Towles, B. (2003). *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc.

[Effiong, 2018] Effiong, C. (2018). *Exploration of multicore systems based on silicon integrated communication networks*. PhD thesis, Université de Montpellier.

[Effiong et al., 2017] Effiong, C., Sassatelli, G., and Gamatié, A. (2017). Distributed and Dynamic Shared-Buffer Router for High-Performance Interconnect. In *NOCS*, pages 2:1–2:8.

[Kaushal and Singh, 2014] Kaushal, A. and Singh, S. (2014). Network on Chip Architecture and Routing Techniques : A survey. *International Journal of Research in Engineering and Science (IJRES)*, 2(6):65–79.

[Michelogiannakis et al., 2010] Michelogiannakis, G., Becker, D., and Dally, W. (2010). Evaluating elastic buffer and wormhole flow control. *IEEE Transactions on Computers*, 60(6):896–903.

[Michelogiannakis and Dally, 2011] Michelogiannakis, G. and Dally, W. J. (2011). Elastic buffer flow control for on-chip networks. *IEEE Transactions on computers*, 62(2):295–309.

[Moraes et al., 2004] Moraes, F., Calazans, N., Mello, A., Möller, L., and Ost, L. (2004). Hermes: an infrastructure for low area overhead packet-switching networks on chip. *INTEGRATION, the VLSI journal*, 38(1):69–93.

[Oliveira et al., 2018a] Oliveira, B. S., Medina, H. M., Sant'Ana, A. C., and Moraes, F. G. (2018a). Secure Environment Architecture for MPSoCs. In *SBCCI*, pages 1–6.

[Oliveira et al., 2018b] Oliveira, B. S., Reusch, R., Medina, H., and Moraes, F. (2018b). Evaluating the cost to cipher the NoC communication. In *LASCAS*, pages 1–4.

[Pasricha and Dutt, 2010] Pasricha, S. and Dutt, N. (2010). *On-chip communication architectures: system on chip interconnect*. Morgan Kaufmann.

[Ruaro et al., 2017a] Ruaro, M., Medina, H. M., and Moraes, F. G. (2017a). SDN-Based Circuit-Switching for Many-Cores. In *ISVLSI*, pages 385–390.

[Ruaro et al., 2017b] Ruaro, M., Medina, H. M., and Moraes, F. G. (2017b). SDN-Based Circuit-Switching for Many-Cores. In *ISVLSI*, pages 385–390.

[Sant'ana et al., 2019] Sant'ana, A., Boucinha, K., Medina, H., and Moraes, F. G. (2019). Lightweight Security Mechanisms for MPSoCs. In *SBCCI*, pages 1–6.