UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

IMPLEMENTAÇÃO DE UMA REDE INTRA-CHIP COM TOPOLOGIA MALHA BIDIMENSIONAL E TRIDIMENSIONAL

TRABALHO DE CONCLUSÃO DE CURSO

Iaçanã Ianiski Weber

Santa Maria, RS, Brasil

2015

IMPLEMENTAÇÃO DE UMA REDE INTRA-CHIP COM TOPOLOGIA MALHA BIDIMENSIONAL E TRIDIMENSIONAL

Iaçanã Ianiski Weber

Trabalho de conclusão de curso apresentado como requisito parcial para obtenção do grau de **Engenheiro de Computação**.

Orientador: Prof. Dr. Everton Alceu Carara

Santa Maria, RS, Brasil

2015

Universidade Federal de Santa Maria
Centro de Tecnologia
Graduação em Engenharia de Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Conclusão de Curso

IMPLEMENTAÇÃO DE UMA REDE INTRA-CHIP COM TOPOLOGIA MALHA BIDIMENSIONAL E TRIDIMENSIONAL

elaborado por
Iaçanã Ianiski Weber

COMISSÃO EXAMINADORA:

_____
Prof. Dr. Everton Carara (UFSM)
(Presidente/orientador)

_____
Prof. Dr. Mateus Beck Rutzig (UFSM)

_____
Prof. Dr. Carlos Barriquello (UFSM)

Santa Maria, 15 de dezembro de 2015.

RESUMO

Trabalho de Conclusão de Curso
Graduação em Engenharia de Computação
Universidade Federal de Santa Maria

IMPLEMENTAÇÃO DE UMA REDE INTRA-CHIP COM TOPOLOGIA MALHA BIDIMENSIONAL E TRIDIMENSIONAL
AUTOR: IAÇANÃ IANISKI WEBER
ORIENTADOR: EVERTON CARARA

Sistemas intra-chip (SoCs – *Systems-on-Chip*) são o atual paradigma utilizado na implementação de sistemas embarcados. O poder computacional destas plataformas possibilita a execução simultânea de diversas aplicações com diferentes requisitos. Neste cenário, o emprego de redes intra-chip (NoCs – *Networks-on-chip*) como infraestrutura de comunicação é uma realidade em pesquisas acadêmicas e projetos industriais. NoCs normalmente são vistas como alternativas à comunicação via barramento, oferecendo como principais vantagens a escalabilidade e suporte a comunicações paralelas.

Muitos trabalhos têm sido conduzidos na última década na área de NoCs, tratando da capacidade e potencial que esta infraestrutura de comunicação pode garantir aos SoCs. Devido à recente introdução do paradigma de circuitos integrados 3D e 2.5D, diversas pesquisas apontam a topologia tridimensional das NoCs como uma grande solução para SoCs que necessitam de centenas de núcleos de processamento. Motivados pela escassez de NoCs de código aberto que possuem nativamente a topologia de malha tridimensional e que sejam de fácil acesso a estudantes e desenvolvedores, o projeto da NoC Arke busca oferecer uma NoC estável, que dá suporte às topologias malha bidimensional e malha tridimensional, com diversos aspectos parametrizáveis, de fácil entendimento e independente de *frameworks* para geração automatizada de diferentes instâncias.

A NoC Hermes foi o principal objeto de estudos em relação ao funcionamento de uma NoC e ponto de partida para a implementação da NoC Arke. Ambas NoCs possuem muitas características em comum. Durante a criação da Arke buscou-se superar o modelo no qual ela havia sido moldada. Aspectos relacionados a facilidade de utilização, legibilidade do código, independência de *framework*, área ocupada pelo circuito, potência dissipada e tempo de roteamento foram pontos onde a Arke mostrou-se mais eficiente.

Palavras chave: SoC, NoC, 3D e comunicação intra-chip.

ABSTRACT

Course Completion Assignment
Graduate Degree in Computer Engineering
Federal University of Santa Maria

BIDIMENSIONAL AND TRIDIMENSIONAL MESH TOPOLOGY NETWORK-ON-CHIP IMPLEMENTATION

AUTHOR: IAÇANÃ IANISKI WEBER
COUNSELLOR: EVERTON CARARA
Date and Location of Defence: Santa Maria, 2015.

Systems-on-Chip (SoCs) are the current paradigm used in the implementation of embedded systems. The computational power of these platforms enables the simultaneous execution of multiple applications with distinct requirements. In this scenario the use of Networks-on-Chip (NoCs) as the communication infrastructure is already a reality in both academic research and industrial projects. NoC are typically seen as an alternative for bus communication, offering as main advantages the scalability and the support of parallel communication.

Many studies have been conducted over the past decade in the NoC field, showing the ability and potential that this communication infrastructure can ensure to SoCs. Following the recent introduction of 3D and 2.5D integrated circuit paradigm, several studies have pointed to three-dimensional topology NoCs as a great solution for SoCs that require hundreds of processing cores. Motivated by the lack of an open source NoC which natively supports the three-dimensional mesh topology and is easily accessible to students and developers, this project seeks to offer the Arke NoC: a stable NoC which enables both two-dimensional and three-dimensional mesh topologies, with many configurable features, yet still easy to understand and independent of frameworks to generate different instances.

The Hermes NoC was the main object of study in relationto the operation of a NoC and starting point for the implementation of the NoC Arke. Both NoCs have many features in common. Since the time of its creation, Arke has sought to overcome the model in which it was shaped: aspects related to ease of use, readability of code, area occupied by the circuit, power dissipation and routing time are points where the Arke has proven to be more efficient.

Key-words: SoC, NoC, 3D and intra-chip communication.

LISTA DE SIGLAS

ASIC - Application Specific Integrated Circuit
DAMQ – Dynamically Allocated Multi-Queue
EOP – End of Package
FIFO – First-in first-out
FPGA - Field Programmable Gate Array
HOL – Head-of-Line blocking
IC – Integrated Circuit
IP – Intellectual Property
LUT - Look-Up Tables
NoC – Network-on-Chip
RTL – Register transfer level
SAFC – Statically Allocated Fully Connected
SAMQ – Statically Allocated Multi-Queue
SoC – System-on-Chip
SSI – Stacked Silicon Interconnect
TSV – *Through-Silicon Via*
VHDL - Very High Speed Integrated Circuits Hardware Description Language

ÍNDICE DE FIGURAS

Figura 1 – Representação de uma NoC de topologia de malha bidimensional (4 x 4) (FEERO & PANDE, 2007). 2
Figura 2 - Virtex®-7 2000T FPGA Enabled by SSI Technology (SABAN, 2012). 4
Figura 3 – Circuito Integrado Tridimensional. 5
Figura 4 – Representação de um SoC utilizando uma NoC de Topologia 3D de dimensões 3x3x3 (XxYxZ) (FEERO & PANDE, 2007). 5
Figura 5 – SoC realizando a transmissão de um pacote entre dois IPs através de uma NoC. 7
Figura 6 – Redes diretas: (a) malha 2D; (b) toróide 2D; (c) malha 3D (ZEFERINO, 2003). 9
Figura 7 - Etapas do *Circuit Switching.* (a) O cabeçalho avança pela rede, alocando os canais desde sua fonte até seu destino; (b) Uma informação de reconhecimento é enviada para o nodo fonte pelo caminho de retorno do circuito estabelecido. (c) Inicia-se a transferência dos dados da mensagem e o *tail flit* desaloca os canais por onde passa (PALMA J. , 2003). 12
Figura 8 – Arquitetura do roteador da Hermes. B – *Input Buffers* (MORAES, 2004). 15
Figura 9 – Roteador da rede Arke. SC – *Switch Control.* 17
Figura 10 – Organização dos pacotes na rede Arke. 18
Figura 11 – Interface entre dois roteadores conectados através das portas *East* e *West*. 19
Figura 12 – Esquemático completo do roteador da rede Arke. 20
Figura 13 - Interface do *Input Buffer*. 21
Figura 14 – Simulação para demonstração do funcionamento do *Input Buffer*. 21
Figura 15 – Interface do *Switch Control* e estrutura da *routingTable* 22
Figura 16 – Malha tridimensional 3x3x3. O caminho destacado exemplifica o funcionamento do algoritmo de roteamento XYZ. 23
Figura 17 – Simulação do *switch control* recebendo uma solicitação de roteamento 24
Figura 18 – (a) Interface do *Crossbar*. (b) Ligação entre o *Input Buffer* e a porta de saída *east* feita no *Crossbar*. 25
Figura 19 – *Crossbar* realizando a conexão entre um *Input Buffer* e uma porta de saída. 26
Figura 20 - Roteador da rede Arke. Em destaque, os componentes que participam do envio do pacote exemplo. SC – *Switch Control.* 27
Figura 21 – Forma de onda mostrando a entrada e saída de um pacote em um roteador. 28
Figura 22 – Diagrama do pipeline de transmissão de pacotes do roteador da Arke. R0, R1 e R2 – Roteadores; BW – *Buffer Write*; RR – *Routing Request*; SA – *Switch Allocation*; CT – *Crossbar Traverse* 29
Figura 23 - Diagrama do pipeline de transmissão de pacotes do roteador da Arke. Considerando a arquitetura de 3 ciclos do *Input Buffer*. R0, R1 e R2 – Roteadores; BW – *Buffer Write*; RR – *Router Request*; SA – *Switch Allocation*; CT – *Crossbar Traverse* 31
Figura 24 – Gráfico do limiar entre de desempenho entre os pipelines de 3 e 4 ciclos para as implementações em FPGA XC6SLX45 do roteador 2D da Arke. 35
Figura 25 – Distribuição da área ocupada pelos componentes dos roteadores da Arke. 36
Figura 26 – Crescimento da área dos roteadores em função do aumento da largura dos *flits.* 36
Figura 27 – Crescimento da potência dissipada pelo roteador em função do aumento da largura dos *flits.* 38
Figura 28 – Representação do sistema prototipado em FPGA da NoC Arke. 44
Figura 29 – Imagens da exibição dos *flits*, no FPGA, que foram transmitidos pela NoC. 45

Índice de Tabelas

Tabela 1: Classificação dos algoritmos de roteamento 10
Tabela 2 – Comparativo de desempenho entre as topologias oferecidas pela Arke. 32
Tabela 3 – Quantidades de recursos do FPGA Spartan-6 XC6SLX45 utilizados na implementação de cada um dos componentes do roteador Arke. 33
Tabela 4 – Quantidades de recursos do FPGA Spartan-6 XC6SLX45 utilizados na implementação dos roteadores Arke. 34
Tabela 5 – Comparação entre *Input Buffer* da Arke e da Hermes 40
Tabela 6 - Comparação entre os *Switch Controls* da Arke e da Hermes. 41
Tabela 7 – Comparação entre o *Crossbar* da Arke e da Hermes. 41
Tabela 8 - Comparação entre o Roteador 2D (5 portas) da Arke e o Roteador da Hermes. 42
Tabela 9 – Comparação entre as frequências máximas alcançadas nos roteadores da Arke e Hermes (ASIC). 43
Tabela 10 – Resumo da comparação entre as NoCs Arke e Hermes. 47

ÍNDICE

1 INTRODUÇÃO 1

1.1 INTEGRAÇÃO 3D 3
1.1.1 2.5D 4
1.1.2 3D ICs 4
1.2 OBJETIVOS 5

2 NETWORKS-ON-CHIP 7

2.1 TOPOLOGIA 8
2.2 ROTEAMENTO 9
2.3 CHAVEAMENTO 11
2.3.1 Chaveamento por circuito (*Circuit Switching*) 11
2.3.2 Chaveamento por Pacote (*Packet Switching*) 12
2.4 MEMORIZAÇÃO 14
2.5 CONTROLE DE FLUXO 14
2.6 ARBITRAGEM 14
2.7 HERMES 15

3 A REDE ARKE 17

3.1 *INPUT BUFFER* 20
3.2 *SWITCH CONTROL* 22
3.3 *CROSSBAR* 25
3.4 ROTEADOR 27

4 RESULTADOS 29

4.1 DESEMPENHO 29
4.2 ÁREA 32
4.3 ARKE *VS* HERMES 39
4.4 PROTOTIPAÇÃO EM FPGA 44

5 CONCLUSÃO 47

6 REFERÊNCIAS BIBLIOGRÁFICAS 51

APÊNDICE 1 – ARKE STARTER GUIDE 55

1 INTRODUÇÃO

Impulsionados pela grande demanda do mercado consumidor de tecnologia, empresas de semicondutores e pesquisadores do ramo investiram tempo e dinheiro em aperfeiçoamentos na arquitetura e desempenho de processadores. Porém, cientes das limitações agregadas com o aumento da frequência de *clock*, começaram as buscas por alternativas para substituir o paradigma de um único processador. Foi em meados de 2004 que as companhias líderes do mercado de computadores de propósito geral e sistemas embarcados confirmaram a nova tendência, *chips* com múltiplos processadores (ABDALLAH, 2013). Gordon Moore, em abril de 1965, havia publicado um artigo na *Electronics magazine*, onde ele previu que a cada dezoito meses a quantidade de transistores em um chip duplicaria (MOORE, 1965), hoje, conhecemos esta previsão como Lei de Moore. E é graças ao cumprimento dela que foi possível dar suporte ao novo paradigma proposto.

Seguindo a tendência do aumento da complexidade dos sistemas e forçados pelo sedento mercado que demanda de um curto *time-to-market,* os projetistas de circuitos integrados aderiram a uma nova metodologia. A qual é baseada na utilização de núcleos de *hardware*, denominados núcleos de propriedade intelectual ou simplesmente propriedade intelectual (*Intellectual Property* - IP) (GUPTA, 1997). A nova metodologia de projeto exigie que os projetistas desenvolvam, testem e implementem individualmente cada um dos IPs, e estes, uma vez funcionais e validados, podem ser reutilizados em outros projetos. Eles podem ser desenvolvidos pela própria empresa responsável pelo projeto do sistema ou adquiridos de terceiros. Desta forma, os projetistas podem se concentrar nas características do sistema como um todo, sem ter que se preocupar com o desempenho ou funcionamento interno de componentes individuais a cada novo projeto (PALMA J. C., 2007).

Um projeto que contenha diversos núcleos de hardware integrados em um único chip, é conhecido como sistema intra-chip (*System-on-Chip* – SoC), e é baseado no reuso de IPs (GUPTA, 1997). Porém, os núcleos sozinhos não constituem um SoC, é necessária uma estrutura de interconexão capaz de realizar a comunicação e interface entre os IPs (MARTIN & CHANG, 2001). Em sistemas onde a quantidade de núcleos é pequena, a utilização de barramentos dedicados ou compartilhados entre IPs é viável e efetiva. Entretanto, quando utilizado o barramento dedicado a quantidade de ligações cresce rapidamente com o aumento da complexidade do SoC, sendo algumas dezenas o limite da escalabilidade. No caso do barramento compartilhado, todos os IPs se conectam diretamente com um único barramento. Onde apenas uma comunicação pode ocorrer no mesmo instante de tempo. Isto acarreta o aumento na latência do sistema. (KUMAR, 2002).

Neste contexto, surgem as redes intra-chip (*Network-on-Chip* - NoC), uma rede de roteadores interconectados entre si de acordo com uma determinada topologia, sendo que cada roteador se

conecta a um IP (BENINI & MICHELI, 2002). A Figura 1 ilustra um exemplo de IPs interconectados por uma NoC com topologia malha bidimensional. Dentre algumas características da NoC, podemos citar: eficiência energética, confiabilidade, escalabilidade, reusabilidade e decisão de roteamento distribuída (BENINI & MICHELI, 2002) (GUERRIER & GREINER, 2000). As NoCs possuem um alto grau de confiabilidade quando comparadas a barramentos, visto que caso apresentem algum problema na ligação entre dois roteadores, apenas uma parte do sistema encontrará falha de comunicação. No caso de um barramento compartilhado apresentar falha em algum dos fios, a comunicação de todo o sistema pode ser comprometida. Além disso, se a rede for projetada de maneira parametrizável ela pode ser expandida facilmente, facilitando a reutilização em projetos com diferentes quantidades de IPs.

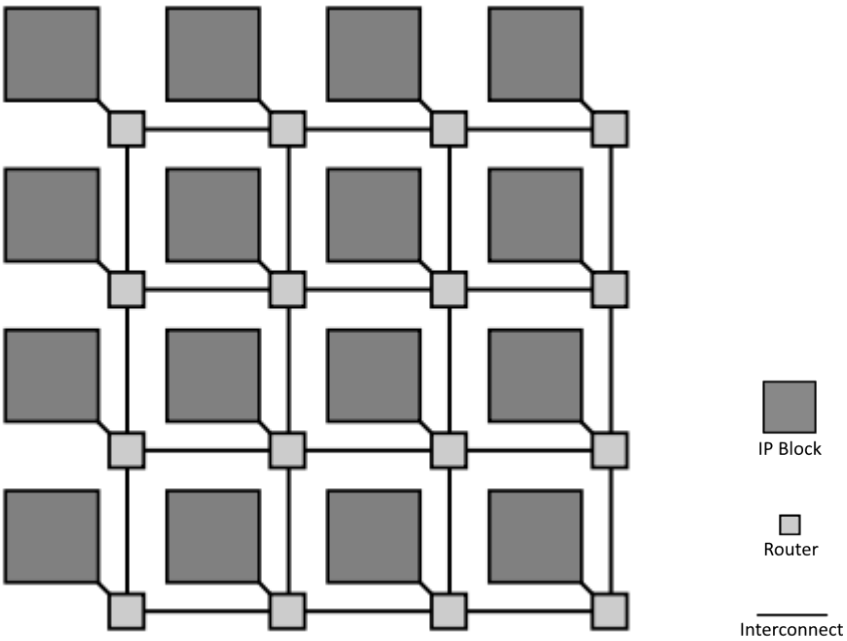


Figura 1 – Representação de uma NoC de topologia de malha bidimensional (4 x 4) (FEERO & PANDE, 2007).

A topologia da NoC possui relevância em seu desempenho. A Figura 1 mostra como é a organização usualmente empregada em NoCs. Cada IP possui uma ligação direta com um roteador, este, por sua vez, conecta-se com seus vizinhos formando uma malha bidimensional (NoC 2D). Além da topologia apresentada existem outras formas de realizar a conexão entre os roteadores, ficando a cargo do projetista escolher a melhor estrutura para seu SoC. Constantemente novas topologias vêm sendo estudadas, sendo uma delas a malha tridimensional (NoC 3D). As malhas tridimensionais são basicamente compostas por malhas bidimensionais empilhadas e verticalmente conectadas (ABDALLAH, 2013). Nessa nova topologia cada roteador pode receber até mais duas novas conexões, para realizar a ligação entre as camadas. Feero e Pande demonstraram que utilizando uma NoC 3D é possível reduzir em 40% a latência e o consumo de energia por pacote, através da redução da quantidade de saltos. Pavlidis e Friedman comparam NoCs com diferentes topologias, mas com a

mesma quantidade de roteadores. Eles demonstraram que até 62% de redução de energia pode ser alcançada com a utilização de NoCs 3D ao invés de 2D (PAVLIDIS & FRIEDMAN, 2007). Os benefícios da utilização de topologias 3D em comparação com as 2D podem ser explicados pela menor quantidade de saltos que os pacotes precisam fazer para cumprir sua rota, visto que o diâmetro da rede diminui para a mesma quantidade de roteadores. O diâmetro da rede é definido pela quantidade de saltos que um pacote tem que fazer percorrendo o maior caminho mínimo entre dois roteadores (ABDALLAH, 2013).

Principalmente pela grande capacidade de escalabilidade e por ter se mostrado um eficiente meio de realizar comunicação intra-chip em SoCs com dezenas de IPs, as NoCs continuam atraindo a atenção e investimentos tanto do cenário acadêmico quanto da indústria (PAVLIDIS & FRIEDMAN, 2007). Atualmente, com o avanço da tecnologia de circuitos integrados tridimensionais, as topologias 3D vêm ganhando espaço e oportunidades (ABDALLAH, 2013). Em circuitos integrados 3D, NoCs com topologia malha têm atingido ganhos significativos, taxa de transferência, latência e dissipação de energia (FEERO & PANDE, 2009). Desde 1998 no mercado, a empresa Sonics possui mais de 130 patentes registradas e constantemente vem investindo em aperfeiçoamento e desenvolvimento de novas tecnologias. Se auto intitulando a "verdadeira líder em redes intra-chip". Porém já há alguns anos, vem competindo contra a Arteris, fundada em 2003, que possui menos tempo de mercado, mas oferece maiores facilidades para o desenvolvimento de produtos utilizando sua tecnologia e como clientes, possui algumas gigantes tais como: Samsung, Altera, Freescale e Spreadtrum. Apostando neste mercado, a Intel vem investindo recursos no projeto Teraflops, o primeiro chip a alcançar um trilhão de cálculos de ponto flutuante por segundo. Esse chip é constituído de 80 núcleos interconectados através de uma NoC. Em alguns resultados para o chip foram alcançadas até frequências de 5.7GHz resultando em um trafego de dados de 2.92 Terabits/s tudo isso agregado à pouca energia necessária e baixa dissipação de calor.

1.1 Integração 3D

Novas funcionalidades são oferecidas constantemente pela indústria de semicondutores. O crescimento destas empresas nos últimos 30 anos está diretamente relacionado a esta capacidade de inovação constante (DENG & MALY, 2005). A tecnologia dos transistores está chegando ao seu limite físico em termos de tamanho e isto força os pesquisadores a buscarem outro paradigma de integração. O mercado deve estar preparado para continuar inovando ou sofrerá uma estagnação quando o tamanho mínimo do transistor for atingido. Dois novos paradigmas de integração de circuitos se destacam, são eles: integração 2.5-dimensional (2.5D) e integração tridimensional (3D).

1.1.1 2.5D

A ideia por trás da integração 2.5-D é fabricar diversos *dies*, cada um contendo partes de um único SoC e conecta-los através de uma camada inferior. A camada inferior além de atuar como um meio de interconexão para os *dies* também os acomoda sobre ela. A utilização deste tipo de método de integração fornece diversas vantagens, tais como: redução do tamanho das vias de comunicação dentro do chip, oportunidade de reuso de IPs em nível de *die*, a dissociação entre a escolha da tecnologia (cada *die* pode ter uma tecnologia de fabricação diferente) e o aumento da funcionalidade (DENG & MALY, 2005).

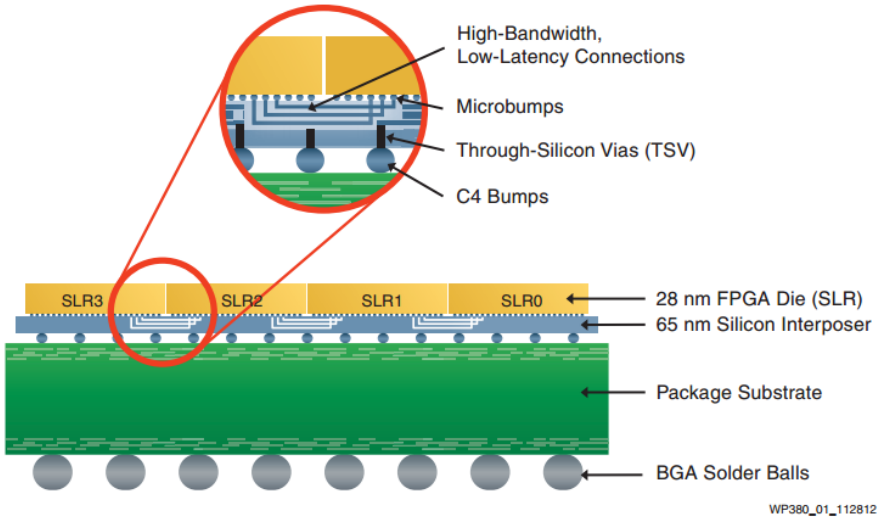


Figura 2 - Virtex®-7 2000T FPGA Enabled by SSI Technology (SABAN, 2012).

Na Figura 2, pode-se observar um esquema que representa a tecnologia utilizado no FPGA Virtex®-7 2000T fabricado pela Xilinx. Chamada pela empresa de *Stacked Silicon Interconnect* (SSI) *Technology*, este é um exemplo de integração 2.5D. Os *dies* são fabricados e otimizados visando desempenho e/ou custo individualmente (DENG & MALY, 2005). Eles são postos lado a lado, sobre uma camada, chamada *interposer*, que foi fabricada de maneira a acomoda-los e interconecta-los. No projeto Virtex®-7 a empresa utilizou tecnologia de 65 nm para fabricar a camada *interposer*, e segundo a empresa esta tecnologia é suficiente para prover dezenas de milhares de conexões *die-to-die.* Ainda segundo a empresa, a tecnologia SSI elimina alguns problemas relacionados a confiabilidade e potência associados ao empilhamento dos *dies* um sobre o outro (SABAN, 2012). Esta proposta acaba sendo muito interessante para a topologia de NoCs 3D, uma vez que cada *die* pode ser projetado para conter uma camada de roteadores e seus respectivos IPs.

1.1.2 3D ICs

A tecnologia 2.5D surgiu da ideia de empilhar diversas camadas de componentes para formar um *chip* 3D, porem a metodologia empregada é dispor diversos *dies* sob um meio que realiza a comunicação entre eles, desta forma, eles estão virtualmente empilhados. Já os circuitos integrados

tridimensionais (3D ICs) são de fato uma aplicação da ideia original, ou seja, os *dies* são colocados uns sobre os outros.

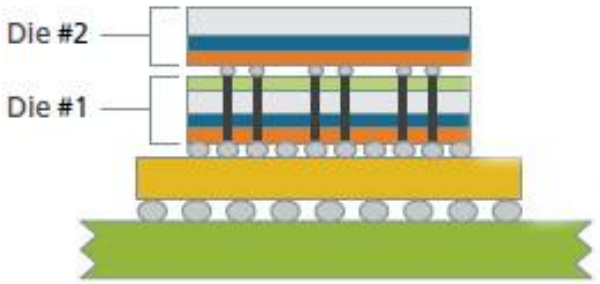


Figura 3 – Circuito Integrado Tridimensional.

A Figura 3 mostra dois *dies* fisicamente empilhados e interconectados através de vias verticais de metal (TSV – *Through-Silicon Via*). Os ICs 3D podem oferecer grandes oportunidades para uma NoC 3D. A utilização de NoCs 3D neste paradigma pode ser feita de maneira semelhante à citada na integração 2.5D. Cada camada da NoC 3D pode ser projetada a ocupar uma das camadas do circuito 3D. A Figura 4 mostra uma representação de um SoC que utiliza uma NoC 3D para interconectar seus IPs. A integração 3D pode viabilizar a fabricação deste SoC da maneira como está representado, onde as camadas superiores (*dies*) se conectam com as inferiores através das TSVs.

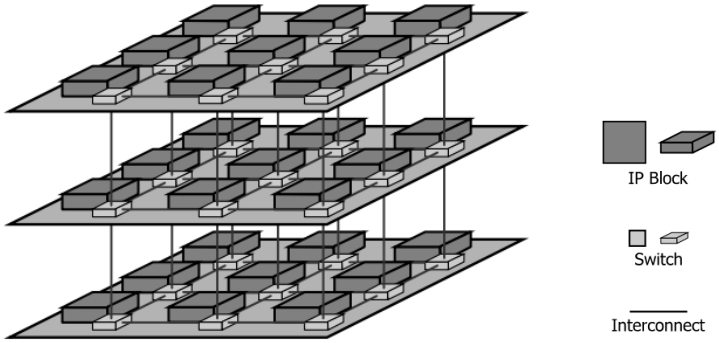


Figura 4 – Representação de um SoC utilizando uma NoC de Topologia 3D de dimensões 3x3x3 (XxYxZ) (FEERO & PANDE, 2007).

1.2 Objetivos

Ficou claro que com o aumento da quantidade de componentes que é possível integrar em um único *chip*, os sistemas estão ficando cada vez mais complexos. Os módulos de *hardware* que facilmente são reutilizados, chamados IPs, são a tendência no quesito de agilidade na hora de colocar um novo projeto em prática. O reuso dos IPs é essencial para que as empresas consigam entregar um produto de uma nova geração antes de sua concorrente. Porém os IPs normalmente não trabalham sozinhos, eles são responsáveis por algumas tarefas que exigem trocar informações com outros IPs para o processamento de uma aplicação complexa. Em um SoC que possua centenas de IPs, todos fazendo requisições para realizar comunicação, fica inviável conecta-los através de um barramento

ou ainda criar um barramento dedicado entre cada IP. Neste cenário uma possível solução, para o problema da comunicação é a utilização de NoCs. A busca da solução do problema da comunicação eficiente dentro dos SoCs é o principal objetivo das NoCs.

A ideia da criação da NoC Arke, surgiu da necessidade de ter a disposição de projetistas, desenvolvedores e pesquisadores da área, uma NoC acessível de forma gratuita, parametrizável e de código aberto. A NoC Hermes, desenvolvida pela PUC-RS, é uma NoC que atende a estes objetivos, porém não possui suporte a topologia 3D e além disso, necessita um *framework* dedicado a automatizar a geração de diferentes instâncias da NoC. Então, alguns objetivos específicos foram almejados para que estes problemas fossem solucionados.

- Implementação RTL de uma NoC com topologia 2D e 3D;
- Código VHDL simples, claro e parametrizável em diversos aspectos;
- Descrição sintetizável (FPGA/ASIC);
- Independência de *framework* para geração automatizada de diferentes instâncias;
- Desempenho semelhante ou superior a Hermes;
- Código aberto e distribuição gratuita.

2 NETWORKS-ON-CHIP

Neste capítulo serão apresentados conceitos básicos sobre NoCs. Ressaltando diversas características das redes, porém o conteúdo aqui apresentado não se trata de uma síntese absoluta, mas sim, um guia para o leitor do trabalho. Diversos termos e aspectos serão explicados nas seções deste capítulo, os quais facilitarão a leitura e compreensão dos conteúdos subsequentes. Ademais, a rede Hermes será apresentada, pois serviu como ponto de partida para o desenvolvimento deste trabalho, além de servir como referência para comparações em termos de área e desempenho.

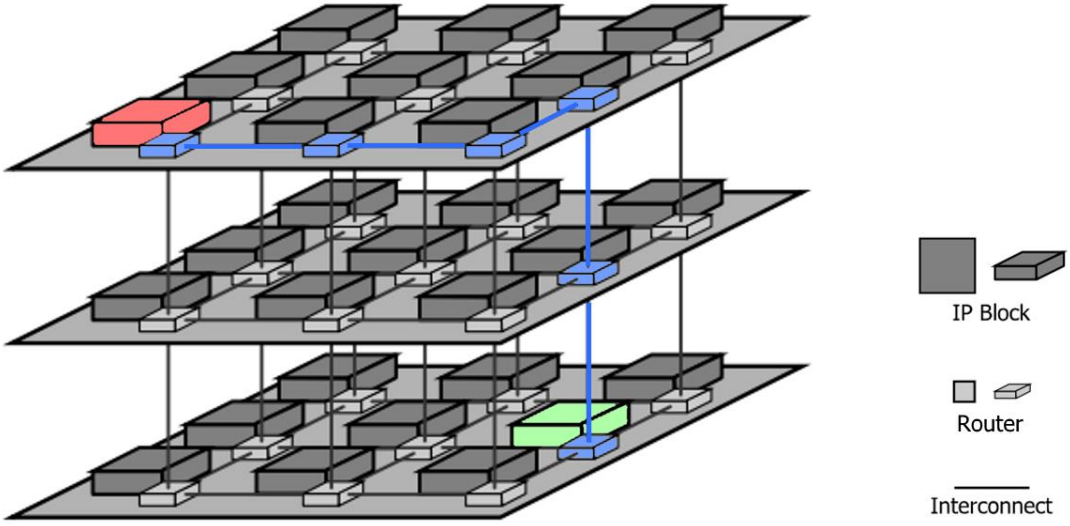


Figura 5 – SoC realizando a transmissão de um pacote entre dois IPs através de uma NoC.

A comunicação de um SoC que utiliza uma NoC como meio de comunicação entre seus diversos dispositivos é realizada seguindo algumas etapas, que podem ser acompanhas pela Figura 5. Inicialmente um dispositivo A possui um conjunto de dados que precisa ser enviado para o dispositivo B, a este conjunto de dados é dado o nome de mensagem. As mensagens não podem ser enviadas diretamente para o roteador, antes é necessário transformá-la em um ou vários pacotes. Um pacote é uma mensagem, ou parte dela, organizada de forma a possibilitar a sua transmissão pela NoC. O pacote é constituído por uma sequência de *flits*. Um *flit* é a menor unidade de dados sobre a qual é realizado o controle de fluxo, normalmente possuindo o tamanho de um *phit* (número de *bits* transmitidos simultaneamente). A organização dos *flits* do pacote é feita da seguinte forma: inicialmente um *flit* carrega o endereço do destino, chamado cabeçalho, seguido por *flits* de dados que compõe a mensagem, sendo que o ultimo destes, intitulado *tail flit*, que carrega uma informação sinalizando o final do pacote, necessária para liberar as rotas que foram alocadas dentro da NoC. Somente então o IP pode transmitir o pacote para o roteador. Assim que o roteador recebe o pacote, é iniciada a transmissão que ocorre através do envio dos dados para um roteador vizinho até que eles

cheguem no roteador destino. Lá o roteador passa o pacote para o IP local, que remove os *flits* auxiliares ficando com a mensagem que lhe foi enviada.

Para entender melhor o funcionamento deste processo, é necessário abordar aspectos relativos à construção da NoC, ela pode ser caracterizada quanto à topologia, estratégias de roteamento, controle de fluxo, chaveamento e arbitragem. Todas essas características serão abordadas nas seções seguintes.

2.1 Topologia

A maneira de como os roteadores são interligados define a topologia da NoC. As topologias podem ser classificadas em dois grupos principais, redes diretas e redes indiretas. Nas redes diretas, cada roteador possui um IP associado, e esse par pode ser visto como um único elemento dentro do sistema, onde tipicamente é referenciado pela palavra "nodo". Cada nodo possui enlaces diretos com um determinado número de nodos vizinhos. Um enlace é a conexão física unidirecional dos sinais da porta de saída de um roteador com a porta de entrada do vizinho. Desta forma, os canais que compõem os roteadores são bidirecionais, formados por dois enlaces. As mensagens de um nodo são destinadas a outro nodo e o roteador do primeiro deve repassá-la a um de seus vizinhos para que a mensagem avance em direção a seu destinatário. Uma mensagem trocada entre dois nodos não-vizinhos, obrigatoriamente deve passar por um ou mais nodos intermediários. É importante ressaltar que durante as trocas de mensagem entre nodos, apenas os roteadores dos nodos intermediários são envolvidos no processo, sem qualquer interferência no processamento dos IPs.

Quanto à conectividade, a rede direta ideal é completamente conectada, ou seja, cada nodo possui um enlace com cada outro nodo da rede. Desta forma, a escalabilidade fica comprometida e acaba limitada a uma rede com um número moderado de nodos. De forma a manter uma boa relação entre a escalabilidade e o custo, diversas implementações foram propostas, sendo a grande maioria modelos de topologia ortogonal. Uma rede possui topologia ortogonal se, e somente se, seus nodos podem ser arranjados em um espaço *n*-dimensional e cada enlace entre nodos vizinhos produz deslocamento em uma única dimensão. As topologias de redes diretas ortogonais mais conhecidas são a malha *n*-dimensional e o toróide, exemplificados na Figura 6.

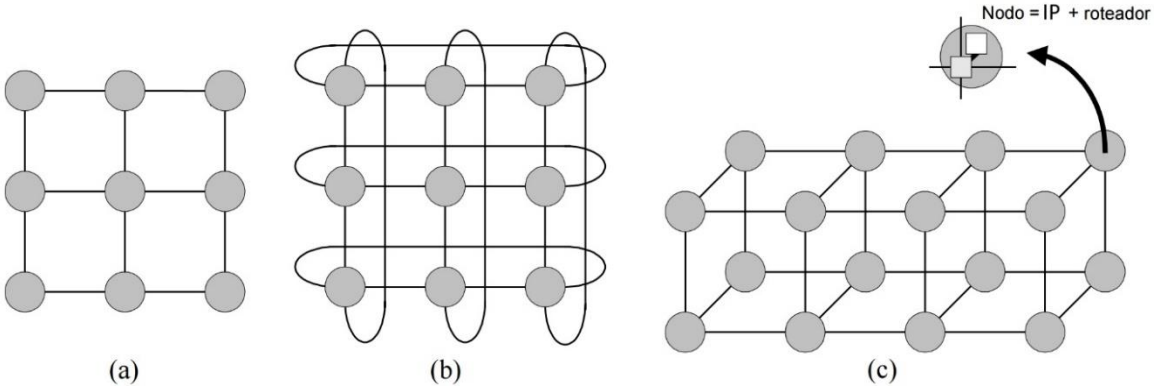


Figura 6 – Redes diretas: (a) malha 2D; (b) toróide 2D; (c) malha 3D (ZEFERINO, 2003).

2.2 Roteamento

É denominado roteamento o método usado pelos roteadores para determinar o caminho através da rede pelo qual um pacote deve ser transmitido. O desempenho da NoC depende fortemente do algoritmo de roteamento utilizado. Em geral, o algoritmo de roteamento visa atender a alguns objetivos específicos, os quais têm consequência direta sobre algumas propriedades da rede, como:

a) Conectividade – capacidade de rotear pacotes de qualquer nodo origem para qualquer nodo destino.

b) Liberdade de *deadlock* e *livelock* – capacidade de garantir que nenhum pacote ficará bloqueado (*deadlock*) ou circulando pela rede sem atingir o seu destinatário (*livelock*).

c) Adaptabilidade – capacidade de rotear pacotes através de caminhos alternativos quando ocorrer congestionamento ou falha em algum componente do caminho em uso.

d) Tolerância a falhas – capacidade de rotear pacotes na presença de falhas em componentes. A tolerância a falhas pode ser obtida sem a adaptabilidade, roteando-se um pacote em duas ou mais fases.

Existem vários algoritmos de roteamento propostos para atender a requisitos distintos. Estes algoritmos podem ser classificados de acordo com a Tabela 1.

Tabela 1: Classificação dos algoritmos de roteamento

Critério	Classificação	Descrição
Quanto ao número de destinos	*Unicast*	Os pacotes têm um único destino.
	Multicast	Os pacotes podem ser roteados para múltiplos destinos.
Quanto ao lugar onde as decisões de roteamento são tomadas	Centralizado	Os caminhos são estabelecidos por um controlador central.
	Fonte	O nodo emissor define o caminho a ser seguido pelo pacote antes de injeta-lo na rede.
	Distribuído	O roteamento é realizado pelos roteadores enquanto o pacote atravessa a rede.
Quanto a implementação	Baseado em tabela	O roteamento é feito a partir de uma consulta a uma tabela em memória.
	Baseado em máquina de estados	O roteamento é realizado a partir da execução de um algoritmo implementado em *software* ou em *hardware*.
Quanto a adaptatividade	Determinístico	O algoritmo de roteamento fornece sempre o mesmo caminho entre um determinado par origem-destino.
	Adaptativo	O algoritmo de roteamento utiliza alguma informação a respeito do tráfego da rede e/ou do estado dos canais para evitar regiões congestionadas ou com falhas.

Fonte**:** (PALMA J. , 2003)

Os algoritmos adaptativos ainda podem ser sub classificados quanto à:

1) Progressividade
   a) Progressivo: Se o cabeçalho do pacote sempre avança pela rede, reservando um novo canal a cada passo de roteamento
   b) Regressivo (*backtracking*): Se o cabeçalho do pacote pode retornar pela rede, liberando canais previamente reservados.
2) Minimalidade:
   a) Mínimo (*profitable*): Se o algoritmo de roteamento pode selecionar apenas caminhos com o número mínimo de roteadores entre a origem e o destino do pacote.
   b) Não-mínimo (*misrouting*): Se o algoritmo de roteamento pode selecionar caminhos mais longos que os caminhos mínimos.
3) Número de caminhos:
   a) Completo: Se o algoritmo de roteamento pode utilizar todos os caminhos disponíveis.
   b) Parcial: Se o algoritmo de roteamento pode utilizar apenas um subconjunto de todos os caminhos possíveis.

2.3 Chaveamento

O método de chaveamento determinará de que forma os dados são transmitidos de uma origem para um destino através dos roteadores. Os roteadores devem assumir uma política de chaveamento para executar as transmissões. Duas políticas de chaveamento são utilizadas em NoCs e são baseadas ou no estabelecimento de um caminho completo entre o nodo origem e o destino da mensagem (circuito) ou na divisão das mensagens em pacotes, que reservam seus caminhos dinamicamente na medida em que avançam em direção ao destino.

2.3.1 Chaveamento por circuito (*Circuit Switching*)

No método de chaveamento por circuito, um caminho completo entre a origem e o destino da mensagem é estabelecido antes da transmissão. Este caminho é mantido até o termino da transmissão e outras requisições para estabelecer comunicação, envolvendo os canais alocados, são negadas. O processo é executado em duas etapas, como mostra a Figura 7. Na primeira, o IP origem envia para a rede um cabeçalho de roteamento contendo o endereço do destino e informações de controle. Esse cabeçalho avança pela rede, alocando canais físicos para o estabelecimento do circuito. Caso algum canal desejado esteja sendo utilizado por outra transmissão, o cabeçalho fica bloqueado aguardando

a liberação do canal. Quando o cabeçalho atinge o seu destino, uma informação de reconhecimento é enviada para a origem através do caminho de retorno do circuito alocado. Ao receber o reconhecimento inicia-se a segunda etapa da comunicação, ou seja, a transferência dos dados. O circuito é desfeito através do avanço do *tail flit*, que ao passar por cada um dos roteadores sinaliza que o canal pode ser liberado.

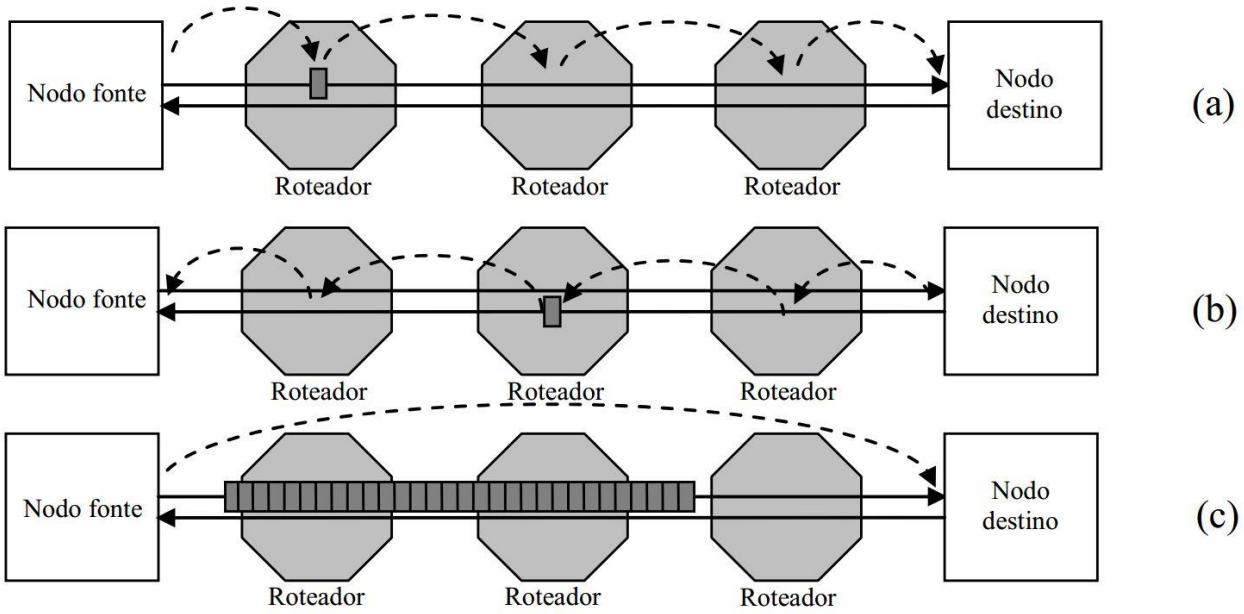


Figura 7 - Etapas do *Circuit Switching*. (a) O cabeçalho avança pela rede, alocando os canais desde sua fonte até seu destino; (b) Uma informação de reconhecimento é enviada para o nodo fonte pelo caminho de retorno do circuito estabelecido. (c) Inicia-se a transferência dos dados da mensagem e o *tail flit* desaloca os canais por onde passa (PALMA J. , 2003).

A grande vantagem deste tipo de chaveamento está no fato de que, uma vez estabelecido o caminho entre dois IPs origem e destino, a mensagem é transferida sem bloqueios. Isto faz com que os roteadores não precisem de filas para armazenar *flits* das mensagens. Apenas pequenos *buffers* podem ser utilizados com capacidade de armazenar o cabeçalho enquanto um canal desejado estiver ocupado. Por outro lado, a desvantagem deste método é a degradação do desempenho do sistema como um todo, pois o caminho entre a origem e o destino fica alocado durante toda a fase de transmissão de dados, não podendo ser utilizado por outro circuito.

2.3.2 Chaveamento por Pacote (*Packet Switching*)

A utilização de chaveamento por circuito em casos onde as mensagens trocadas entre os IPs da rede são curtas e frequentes, pode aumentar a contenção na rede e o tempo para estabelecer um circuito. Uma alternativa é quebrar as mensagens em pacotes que são transmitidos pela NoC. Cada pacote possui um cabeçalho com as informações necessárias para realizar o seu roteamento. Estes pacotes são transmitidos um a um, reservando somente o necessário de recursos para avançar de nodo em nodo. A maior vantagem da utilização deste método é que o canal permanece ocupado apenas

enquanto o pacote está sendo transferido. E como desvantagens, (*i*) obriga que cada roteador possua *buffers* com capacidade de armazenar temporariamente um pacote inteiro ou parte dele e (*ii*) o roteamento deve ser realizado para cada pacote. As técnicas de chaveamento por pacote mais utilizadas são *store-and-forward, virtual cut-through* e *whormhole* (RIJPKEMA, 2001) (MOHAPATRA, 1998).

2.3.2.1 *Store-and-forward*

Neste método, um roteador recebe um pacote, armazena-o temporariamente em seu *buffer*, seleciona a porta de saída e repassa o pacote adiante. Daí o nome da técnica, armazena e repassa (*store-and-forward*). Uma das desvantagens dá-se pelo fato de que um pacote só pode ser repassado após ter sido completamente recebido, o que aumenta a latência da comunicação. Além disso são necessários *buffers* em todos os roteadores de tamanho suficiente para acomodar um pacote completo.

2.3.2.2 *Virtual cut-through*

Como aperfeiçoamento do método *store-and-forward*, foi proposto o método *virtual cut-through*. Sua vantagem em relação ao método anterior é que ele só armazena o pacote inteiro no roteador quando o canal de saída desejado está ocupado. Isso reduz o problema de latência da comunicação quando o canal não está ocupado, pois neste caso o pacote é imediatamente repassado sem a necessidade de aguardar o armazenamento total. Porém o *virtual cut-through* ainda possui a desvantagem de que os roteadores necessitem de *buffers* de tamanho suficiente para acomodar um pacote completo, pois em caso de encontrar um canal bloqueado, ainda é necessário que o *buffer* armazene todo o pacote.

2.3.2.3 Wormhole

Criado como uma variação do *virtual cut-through*, o chaveamento *wormhole* visa reduzir o tamanho dos *buffers* necessários para manter pacotes bloqueados na rede. Neste método os *flits* do pacote vão sendo transmitidos através da rede, caso o *flit* de cabeçalho seja bloqueado, os *flits* sequentes ficam armazenados nos *buffers* dos roteadores intermediários. A vantagem deste método se encontra na redução dos *buffers* dentro dos roteadores, que não precisam mais armazenar um pacote inteiro, apenas alguns *flits*. A desvantagem da técnica é a contenção de recursos causada pelo bloqueio do pacote, pois agora um pacote pode estar distribuído em diversos roteadores ao longo do caminho. Atualmente o chaveamento *wormhole* é o mais utilizado por oferecer um melhor *trade-off* em relação à utilização da rede e ao custo dos roteadores.

2.4 Memorização

Em redes que utilizam chaveamento por pacote, os roteadores devem ser capazes de armazenar os pacotes destinados às saídas que estejam sendo utilizadas por outros pacotes e realizar o controle de fluxo para evitar a perda de dados. Para isto, é necessário um esquema de memorização para manutenção dos pacotes bloqueados no roteador. Uma das formas de implementar isto, é através da utilização de *buffers* independentes nas portas de entrada do roteador. A estratégia mais simples e de menor custo é chamada FIFO (*First-in, First-out*), nela cada *buffer* possui um espaço fixo de memória onde os dados são lidos na mesma ordem em que são escritos. O grande inconveniente da implementação FIFO é o problema de bloqueio HOL (*Head-of-Line blocking*), ou seja, um pacote bloqueado na saída do *buffer* impede o avanço de outro pacote que esteja atrás dele e que possui a saída desejada disponível. Para solucionar o problema de bloqueio HOL diversas outras estratégias são propostas, tais como, o *Statically Allocated, Fully Connected* (SAFC), o *Statically Allocated, Multi-Queue* (SAMQ) e o *Dynamically-Allocated, Multi-Queue* (DAMQ). Tais estratégias tem em comum a possibilidade de leitura em qualquer posição do buffer, o que permite o avanço de pacotes não bloqueados.

2.5 Controle de Fluxo

Os pacotes que trafegam pela NoC competem pelos canais e *buffers* na medida que avançam em direção a seus destinos. Quando um pacote necessita de um recurso que está sendo utilizado por outro pacote, diz-se que houve uma colisão. Quando ocorre uma colisão, é necessário utilizar uma política para decidir o que irá acontecer com o pacote. Esta política é chamada de controle de fluxo e lida com a alocação dos canais e dos *buffers* para a transmissão de um pacote pela rede (NI & McKINLEY, 1993).

O controle de fluxo nas NoCs é realizado em nível de enlace. Geralmente os roteadores possuem *buffers* para armazenamento dos dados em transferência. Quando o *buffer* de entrada do receptor está cheio, o transmissor deve manter o dado a ser enviado em um *buffer* local até que o receptor esteja apto a recebê-lo. Para isto é necessário um mecanismo de controle que envie uma informação do receptor ao transmissor indicando se o primeiro está pronto ou não para receber novos dados.

2.6 Arbitragem

Por fim, a arbitragem é responsável por definir qual porta de entrada poderá utilizar uma determinada porta de saída em um determinado momento. Este mecanismo é essencial para resolver conflitos causados pela existência de múltiplos pacotes competindo por uma mesma porta de saída.

O árbitro deve ser capaz de resolver esses conflitos, selecionando um dos pacotes com base em algum critério e sem levar qualquer pacote a sofrer *starvation*, ou seja, ficar indefinidamente esperando uma oportunidade para avançar em direção ao nodo destino.

2.7 Hermes

Como um exemplo de NoC, a rede Hermes (MORAES, 2004), que foi desenvolvida pelo Grupo de Apoio ao Projeto de Hardware (GAPH) da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), foi escolhida como ponto de partida para o desenvolvimento deste trabalho. Ela possui código aberto e um *framework* para a geração automatizada de NoCs com diversas configurações e características.

A Hermes é uma NoC construída a partir da múltipla instanciação do seu bloco básico, o roteador, ilustrado na Figura 8. Esse roteador possui lógica de controle centralizada além de cinco portas bidirecionais: *East*, *South*, *West*, *North* e *Local*. Cada porta possui um *buffer* de entrada para armazenamento temporário dos *flits*. A porta *Local* estabelece a ligação entre o IP e o roteador, enquanto as demais portas são conectadas com roteadores vizinhos. A lógica de controle implementa um algoritmo de roteamento e o método de arbitragem.

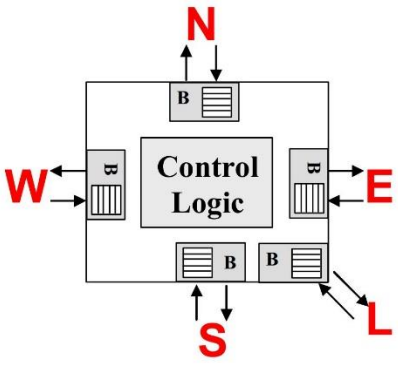


Figura 8 – Arquitetura do roteador da Hermes. B – *Input Buffers* (MORAES, 2004).

A topologia implementada na rede Hermes é a malha 2D, a utilização desta topologia é justificada por facilitar o posicionamento dos roteadores e núcleos no *layout*, bem como o roteamento dos canais entre os roteadores, além de simplificar o algoritmo de roteamento que está implementado na lógica de controle.

Dente os métodos de chaveamento apresentados na seção 2.3, o método *wormhole* foi escolhido por oferecer diversas vantagens: necessidade de *buffers* pequenos para a memorização dos dados, fornece baixa latência na comunicação, pode multiplexar mais de um canal virtual em um canal físico. Como descrito anteriormente, o chaveamento por *wormhole* demanda a divisão do pacote em *flits*. Cada pacote possui um cabeçalho de 2 *flits* com informações necessárias para o seu roteamento. O primeiro *flit* carrega o endereço destino, ou seja, as coordenadas X e Y do roteador

destino. O segundo *flit* indica a quantidade de *flits* que formam o *payload* (área de dados) do pacote. Visto que o tamanho de *flit* é parametrizável na rede Hermes, o tamanho máximo de *payload* é limitado em $2^{(\text{largura de } flit, \text{ em bits})}$. Por exemplo, em uma NoC com largura de *flit* igual a 8 bits, o tamanho máximo para o *payload* é de $2^8 = 256$ *flits*.

Quanto ao roteamento, diversos algoritmos estão disponíveis para a rede Hermes, são eles: XY, *West-first minimal*, *North-last minimal* e *Negative-first minimal*. Os algoritmos são parcialmente adaptativos, exceto o XY que é determinístico, roteando o pacote primeiro na direção X e depois na direção Y. Em caso de a porta de saída escolhida estiver ocupada, todos os *flits* subsequentes deste pacote ficam bloqueados, até que a conexão com a porta desejada seja estabelecida.

Em relação ao controle de fluxo, duas opções estão disponíveis: *handshake* e *credit-based*. O método *credit-based* consiste na utilização de um *buffer* FIFO de entrada no receptor, e uma linha de retorno ao transmissor para informar se há espaço disponível neste *buffer*. Essa informação é interpretada pelo transmissor como sendo um crédito e ele só envia um dado se tiver crédito disponível. No método *handshake* o transmissor informa ao receptor a existência de um dado a ser enviado através de uma linha (*tx*) e o receptor confirma a disponibilidade para receber este dado através de uma linha de reconhecimento (*acknowledge*).

O roteador da Hermes possui um árbitro para determinar qual o pacote deve ser roteado quando mais de um cabeçalho chega ao roteador em um mesmo instante de tempo. O árbitro é dinâmico rotativo e possui uma implementação através de uma fila circular (*Round-Robin*) baseado em um codificador de prioridade programável. Isto garante que todas as requisições sejam atendidas evitando que ocorra *starvation*. O algoritmo de roteamento demanda quatro ciclos de *clock* para tratar uma requisição de roteamento. Somente após este período o árbitro volta a atender outras solicitações.

A rede Hermes pode ser gerada automaticamente através da ferramenta MAIA (MELLO, 2005), incorporada ao ambiente ATLAS (AMORY, 2015). Este ambiente automatiza os vários processos relacionados ao fluxo de projeto e avaliação de desempenho da rede Hermes.

3 A REDE ARKE

Nossa proposta, intitulada Arke, surgiu através do estudo da rede Hermes visando algumas melhorias principalmente em termos de código VHDL. Como principais diferenciais destaca-se o suporte à criação de redes com topologia malha tridimensional e a independência de qualquer tipo de *framework* para a geração de instâncias da NoC. A Arke não busca a extinção de *frameworks* que deem suporte à NoC, apenas a independência deles na geração de instâncias. É notável a importância deste tipo de ferramenta na geração de tráfego e análise de desempenho das NoCs. Uma vez independente, com a alteração de alguns parâmetros, no arquivo de configuração, é possível mudar as dimensões da malha, tamanho de *buffers* e largura dos *flits*. Além disso, durante a codificação da rede, uma das principais preocupações foi manter a legibilidade do código, facilitando o entendimento do funcionamento da rede.

A NoC Arke é construída a partir da múltipla instanciação de um bloco básico, o roteador, ilustrado na Figura 9. O número de portas não é fixo, variando de acordo com a posição do roteador na malha e a topologia (2D ou 3D). Desta forma, o roteador pode ter até sete portas bidirecionais: *East*, *South*, *West*, *North*, *Up*, *Down* e *Local*. Cada porta possui um *buffer* de entrada para armazenamento temporário dos *flits.* A porta *Local* estabelece a ligação entre o IP e o roteador, enquanto as demais portas se conectam com roteadores vizinhos. Além dos *buffers*, os roteadores possuem uma lógica de controle centralizada que implementa o algoritmo de roteamento e faz a arbitragem das conexões entre as portas de entrada e saída.

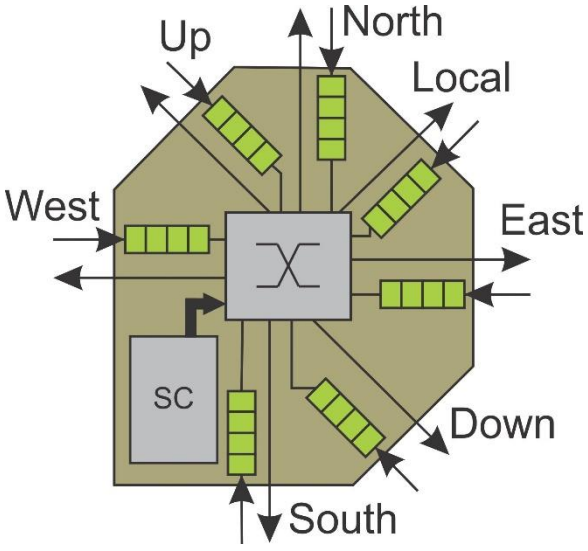


Figura 9 – Roteador da rede Arke. SC – *Switch Control.*

A instanciação dos roteadores é feita de forma automática através do código VHDL. Com base em parâmetros dimensionais, definidos no pacote da NoC, a rede pode possuir uma das seguintes topologias: malha 2D ou malha 3D. A utilização destas topologias é justificada por facilitarem o

posicionamento dos roteadores e núcleos no *layout*, assim como o roteamento dos canais entre os roteadores, além de simplificar o algoritmo de roteamento que é implementado na lógica de controle.

Assim como a NoC Hermes, o método de chaveamento da Arke é *wormhole*. Que foi escolhido por oferecer diversas vantagens como necessidade de *buffers* pequenos para a memorização dos dados e baixa latência na comunicação. Como descrito na seção 2.3.2.3, o método *wormhole* demanda que o pacote seja dividido em *flits*. Cada pacote possui um *flit* de cabeçalho contendo apenas o endereço destino, ou seja, as coordenadas X, Y e Z do roteador destino. O cabeçalho é dividido em três campos (um para cada coordenada) cujos tamanhos são ajustados automaticamente a partir das dimensões que foram determinadas para a NoC. Por exemplo, uma NoC tridimensional com dimensões 2x4x5 (XxYxZ) terá um campo de um *bit* para a coordenada X, dois *bits* para a coordenada Y e três *bits* para a coordenada Z. O restante dos *bits* do cabeçalho são preenchidos por zeros. Em relação ao *payload* do pacote, não há restrições quanto ao seu tamanho. A liberação dos canais alocados pelo pacote é feita conforme o último *flit* é transmitido. O último *flit* de um pacote é indicado através de um sinal de controle específico (*eop –End-of-Package*). Como foi apresentado no capítulo anterior, o cabeçalho da Hermes possui um *flit* que informa a quantidade total de *flits* que formam o pacote e isto acaba limitando o tamanho deles. Devido ao fato da Arke utilizar o sinal *eop*, os pacotes não têm limitação de tamanho. Desta forma, um IP que possui um conjunto de *bytes* (mensagem), que deve ser enviado a outro IP, pode escolher entre dividir a mensagem em pacotes menores ou envia-la inteira, em um único pacote. A forma como os pacotes são organizados pode ser observado na Figura 10.

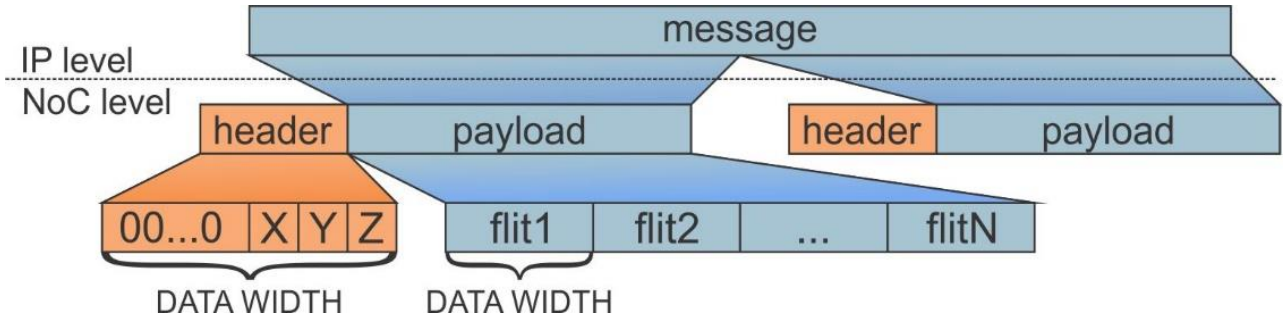


Figura 10 – Organização dos pacotes na rede Arke.

Quanto ao controle de fluxo, é implementado o método *stall and go*. Este método consiste na utilização de um *buffer* FIFO de entrada no receptor, e uma linha de retorno (*control_in(stall_go)*) ao transmissor para informar se há espaço disponível neste *buffer*. Esta informação é interpretada pelo transmissor como uma sinaleira, se estiver em "*go*" (*stall_go* = '1') os dados podem continuar a serem enviados, se estiver em "*stall*" (*stall_go* = '0') os dados devem parar e aguardar a liberação. Para informar quanto a validade do dado presente no *data_out*, o transmissor envia um sinal de controle "*tx*" ao receptor. A interface entre dois roteadores pode ser observada na Figura 11. Na interface entre

os roteadores há duas portas (saída e entrada). E cada porta é formada por dois barramentos, o de dados (*data_in/out*) e o de controle (*control_in/out*), onde cada bit tem uma função específica.

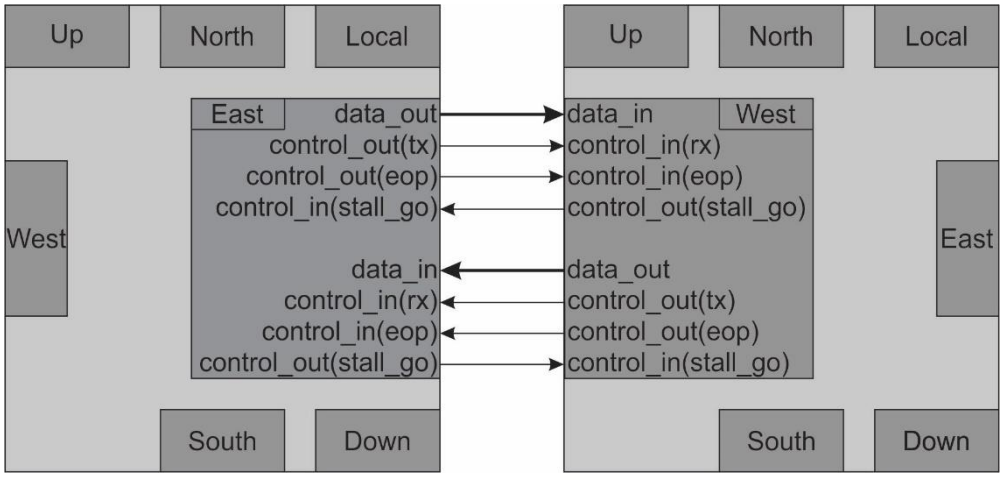


Figura 11 – Interface entre dois roteadores conectados através das portas *East* e *West*.

O roteador da Arke possui um árbitro para determinar qual o pacote deve ser roteado quando mais de um cabeçalho chega ao roteador em um mesmo instante de tempo. Para garantir que nenhum pacote sofra de *starvation*, o árbitro é dinâmico rotativo e é implementado através de uma fila circular (*Round-Robin*) baseado em um codificador de prioridade programável. O árbitro atende uma solicitação de roteamento por vez. Uma vez que a solicitação é atendida e o chaveamento é realizado o arbitro pode atender a novas solicitações. Desta forma, diversas transmissões podem ocorrer simultaneamente dentro do roteador, havendo *delay* apenas no aguardo para que ocorra o roteamento de *Input Buffers* com maiores prioridades. Além disto, se uma porta de entrada está recebendo um pacote de um roteador vizinho, a porta de saída correspondente pode estar enviando um pacote para este mesmo roteador vizinho, ou seja, a conexão entre os roteadores é *full-duplex*.

Como supracitado, não é necessária a utilização de nenhum tipo de *framework* para a geração de códigos da NoC Arke. Todos os componentes foram projetados de maneira parametrizável e estão descritos em VHDL. Através da alteração dos parâmetros no pacote da Arke é possível transitar entre as possíveis formatações da rede. A Figura 12 servirá como um guia para as próximas seções. Ela mostra detalhadamente a organização interna do roteador. Como pode ser observado, cada direção é identificada por um número: *Local* (0), *East* (1), *South* (2), *West* (3), *North* (4), *Up* (5) e *Down* (6). Nas próximas seções, será discutido o funcionamento de cada um dos componentes que formam o roteador da Arke, além disso, será apresentada a transmissão completa de um pacote dentro do roteador. Começando pela sua chegada ao *Input Buffer*, a requisição de roteamento ao *Switch Control*, a configuração do *Crossbar* e o envio para o próximo roteador.

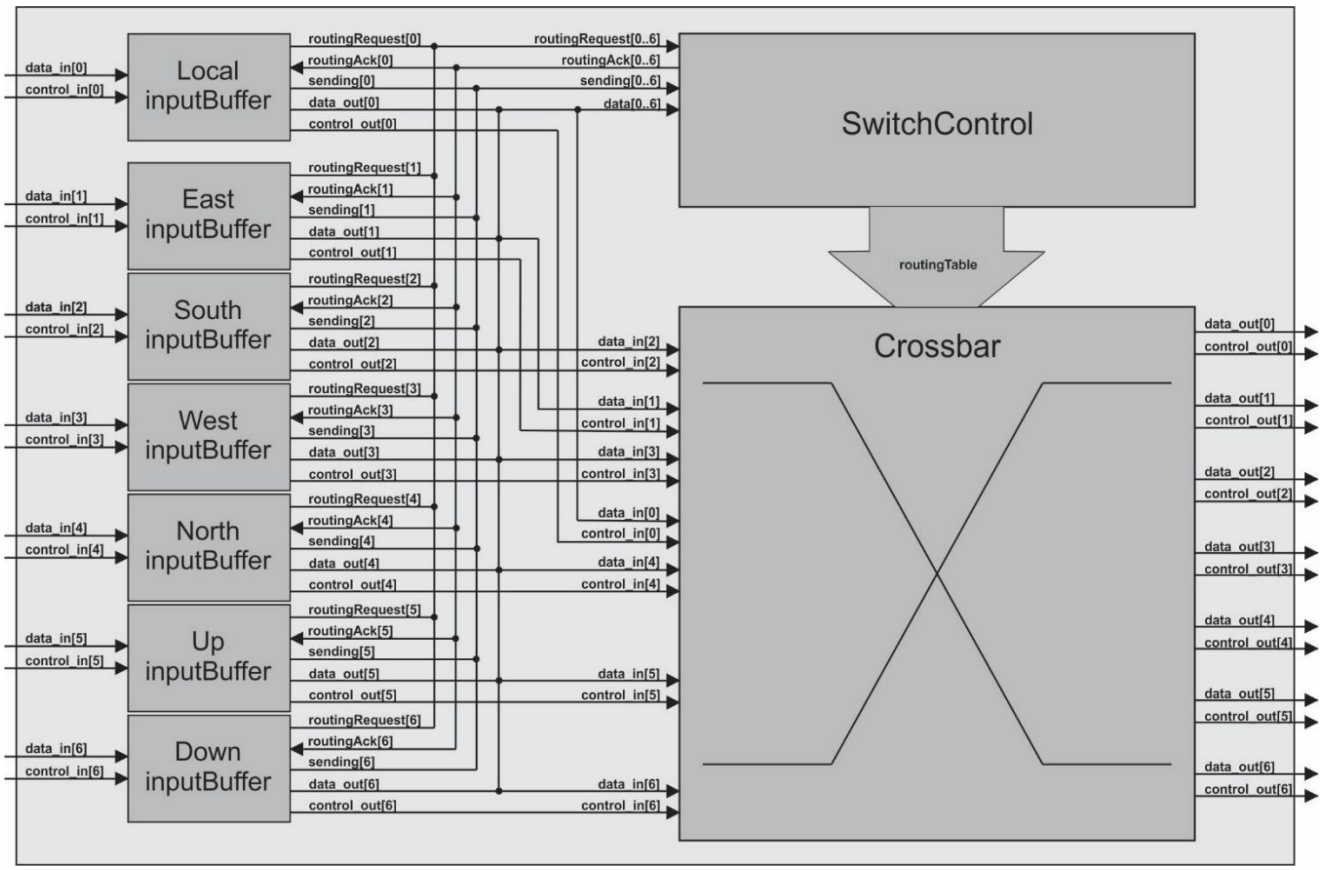


Figura 12 – Esquemático completo do roteador da rede Arke.

3.1 *Input Buffer*

Este componente é fundamental para o funcionamento da NoC e além de ser a porta de entrada dos pacotes é responsável por executar o controle de fluxo dos *flits* que trafegam entre roteadores. Assim como outros aspectos da NoC, é parametrizável, tanto na largura quanto na profundidade. Para realizar o controle de fluxo dos *flits* o *Input Buffer* segue o protocolo *stall and go*. Assim que um cabeçalho é recebido pelo *Input Buffer* ele faz uma requisição de roteamento para o *Switch Control*. Cabe ao *Switch Control* enviar uma confirmação (*acknowledgement*) para o *Input Buffer*, informando que uma porta de saída está alocada e os dados podem ser transmitidos adiante.

Na Figura 13 pode ser observado o esquema de portas de entrada e saída do *Input Buffer*. Através da Figura 14 é possível visualizar o *Input Buffer local* recebendo um pacote com cinco *flits* (0x0022, 0x1111, 0x2222, 0x3333 e 0x4444) e em seguida sendo transmitido ao roteador vizinho.

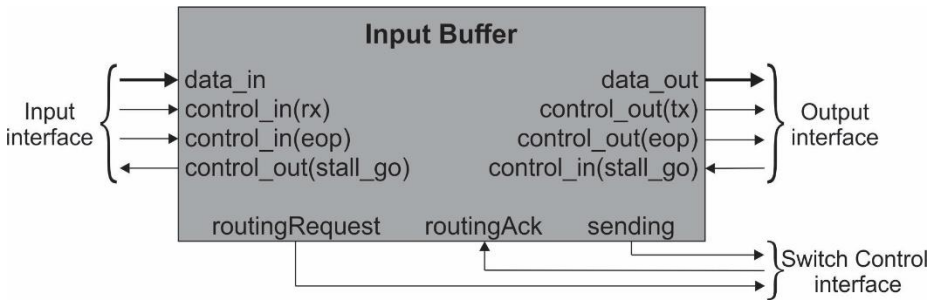


Figura 13 - Interface do *Input Buffer*.

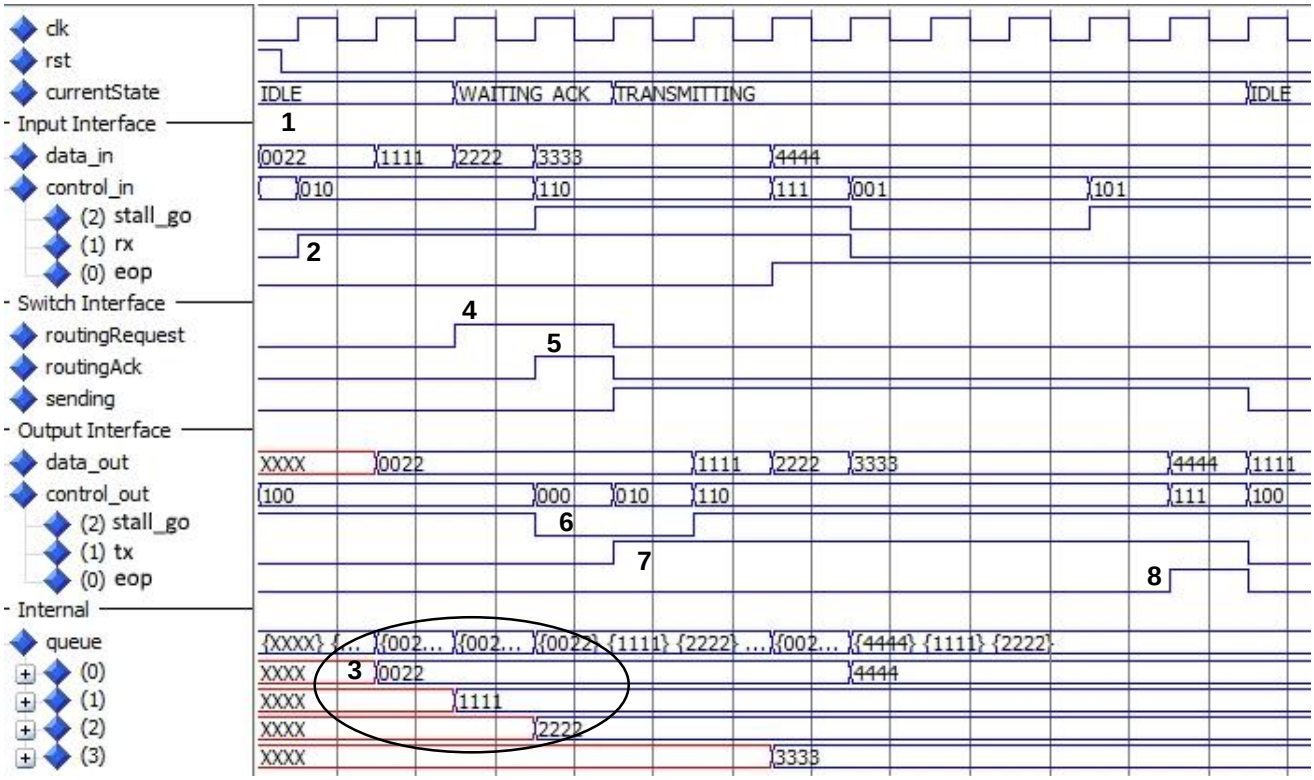


Figura 14 – Simulação para demonstração do funcionamento do *Input Buffer*.

1. Inicialmente o *Input Buffer* começa no estado *Idle* (*currentState*) e permanece aguardando até que um novo *flit* seja armazenado na *queue* (sinais mais abaixo na Figura 14).

2. Um novo dado está disponível para o roteador através da interface de entrada (*data_in*), este dado é confirmado como válido através do sinal de controle *rx.*

3. O *flit* presente em *data_in* é armazenado na *queue* um ciclo após ser disponibilizado, caso haja espaço. Visto que o *buffer* se encontrava vazio, o novo *flit*, correspondente ao endereço destino do pacote, e é imediatamente disponibilizado na interface de saída (*data_out*). Uma vez que o sinal *stall_go* do *buffer* está indicando que há espaço para o armazenamento de *flits* o transmissor continua a transmitir novos *flits* a cada ciclo.

4. No ciclo seguinte, o estado do *buffer* passa para *waiting_ack*, onde é feita a requisição de roteamento para o *Switch Control*. Baseado no *flit* de cabeçalho presente na saída *data_out*, o *Switch Control* executa o algoritmo de roteamento e determina a porta de saída do roteador a

ser utilizada na transmissão do pacote.

5. Ao finalizar o roteamento, o *Switch Control* sinaliza ao *Input Buffer* através do sinal *routingAck* que este pode iniciar a transmissão do pacote.
6. Caso em algum momento a *queue* fique cheia o sinal *stall_go* indicará que não é possível armazenar um novo *flit*. Ao iniciar a transmissão, espaço na *queue* será liberado e novos *flits* poderão ser armazenados.
7. O sinal *tx* da interface de saída é mantido ativo enquanto existirem *flits* do pacote armazenados. O dado de saída (*data_out*) é atualizado a cada ciclo se o receptor estiver com o sinal *stall_go* ativo. Neste momento o estado se encontra em *Transmitting* e o sinal *sending* é mantido ativo, sinalizando ao *Switch Control* que os recursos que foram alocados ainda estão sendo utilizados.
8. O processo de transmissão de um pacote é encerrado quando último *flit* do pacote é transmitido. O sinal *eop* indica que o *flit* presente na saída *data_out* é o último do pacote. Só então o sinal *sending* é desativado, sinalizando ao *Switch Control* que a porta de saída alocada pode ser liberada.

3.2 *Switch Control*

Outro componente que constitui os roteadores é o *Switch Control*. Este é o responsável por arbitrar a utilização das portas de saída do roteador, aplicar o algoritmo de roteamento sobre os endereços destinos e não permitir que os pacotes sofram de *starvation*. Possui interface de conexão com cada um dos *Input Buffers* e com o *Crossbar*, o que pode ser observado na Figura 15 e na Figura 12.

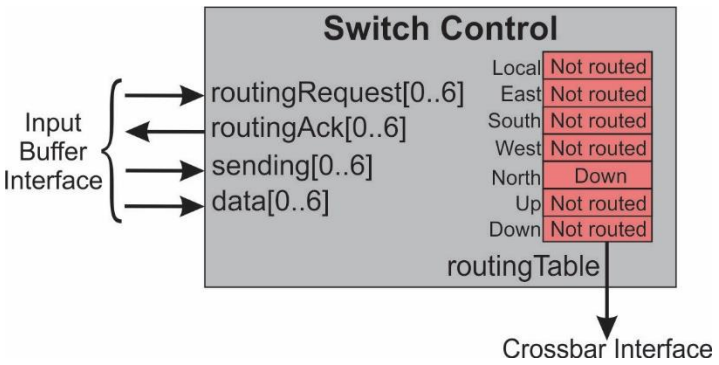


Figura 15 – Interface do *Switch Control* e estrutura da *routingTable*

O *Switch Control* possui um árbitro dinâmico rotativo, implementado através de uma fila circular (*Round-Robin*) baseado em um codificador de prioridade programável. Desta forma, sempre que mais de uma requisição de roteamento (*routingRequest*) é feita ao mesmo tempo, o primeiro da fila é atendido. O algoritmo de roteamento é executado e, caso a porta de saída necessária esteja

disponível, ela é alocada e um sinal (*routingAck*) é enviado para o *Input Buffer*. Caso contrário, aquela requisição passar a ser a última da fila e a próxima é atendida.

A rede Arke possui um algoritmo de roteamento determinístico. A partir da comparação entre o endereço do roteador atual com o endereço destino (carregado pelo cabeçalho) é determinada uma porta de saída. O algoritmo tem o nome de XYZ, que é dado devido à maneira de como o este escolhe a porta de saída. Primeiramente o pacote é encaminhado na direção X até atingir a mesma coordenada X do destino. Em seguida o pacote é encaminhado na direção Y até atingir a mesma coordenada Y do destino. Por fim o pacote é encaminhado na direção Z até atingir o destino. A Figura 16 exemplifica o caminho que um pacote segue dentro de uma rede, utilizando o algoritmo XYZ. Dada uma malha tridimensional com dimensões de 3x3x3, um pacote é enviado do roteador "000" para o roteador "112" seguindo o caminho destacado pelas setas vermelhas. O caminho contrário, do roteador "112" para o roteador "000", está destacado pelas setas azuis.

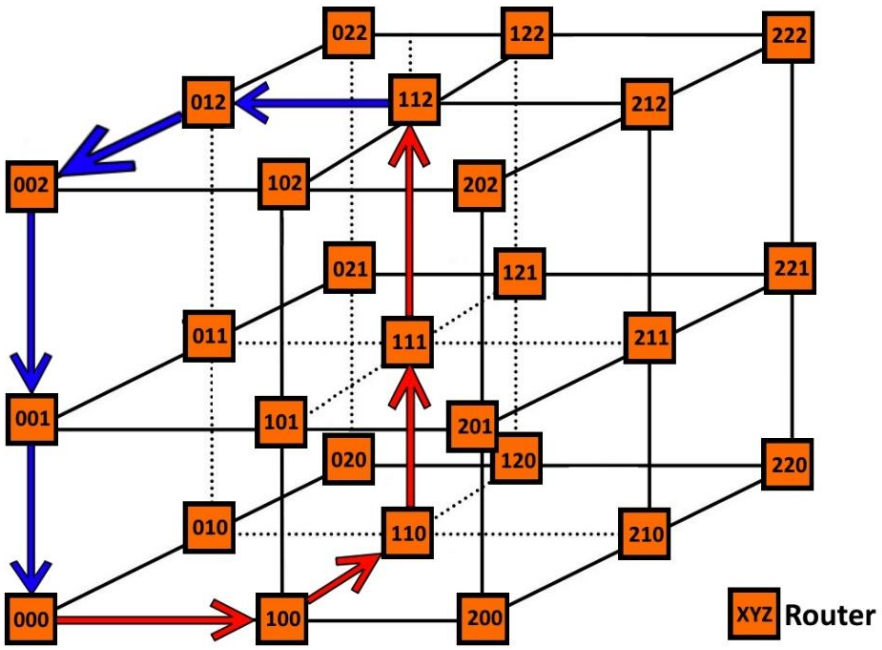


Figura 16 – Malha tridimensional 3x3x3. O caminho destacado exemplifica o funcionamento do algoritmo de roteamento XYZ.

O *Crossbar* é o responsável por executar a ligação entre os *Input Buffers* e as portas de saída do roteador. Para informar sobre quais devem ser as ligações a serem estabelecidas o *Switch Control* fornece uma tabela de roteamento (*routingTable*) para o *Crossbar*, Figura 12. A estrutura da *routingTable* pode ser observada na Figura 15. Cada linha da tabela corresponde a um dos *Input Buffers*. O conteúdo de dada linha corresponde ao número da porta de saída a qual um determinado *Input Buffer* deve ser conectado. Na tabela que está representada na Figura 15, somente a quinta linha (correspondente ao *Input Buffer North*) está conectada a uma porta de saída, a porta *Down*. As linhas referentes a *Input Buffers* que não estão conectados a nenhuma porta de saída ficam preenchidos com valores que não codificam nenhuma porta válida (*Not routed*).

Para exemplificar o funcionamento do *Switch Control*, a Figura 17 apresenta uma simulação com uma solicitação de roteamento sendo realizada. Esta solicitação corresponde ao passo 4 da Figura 14.

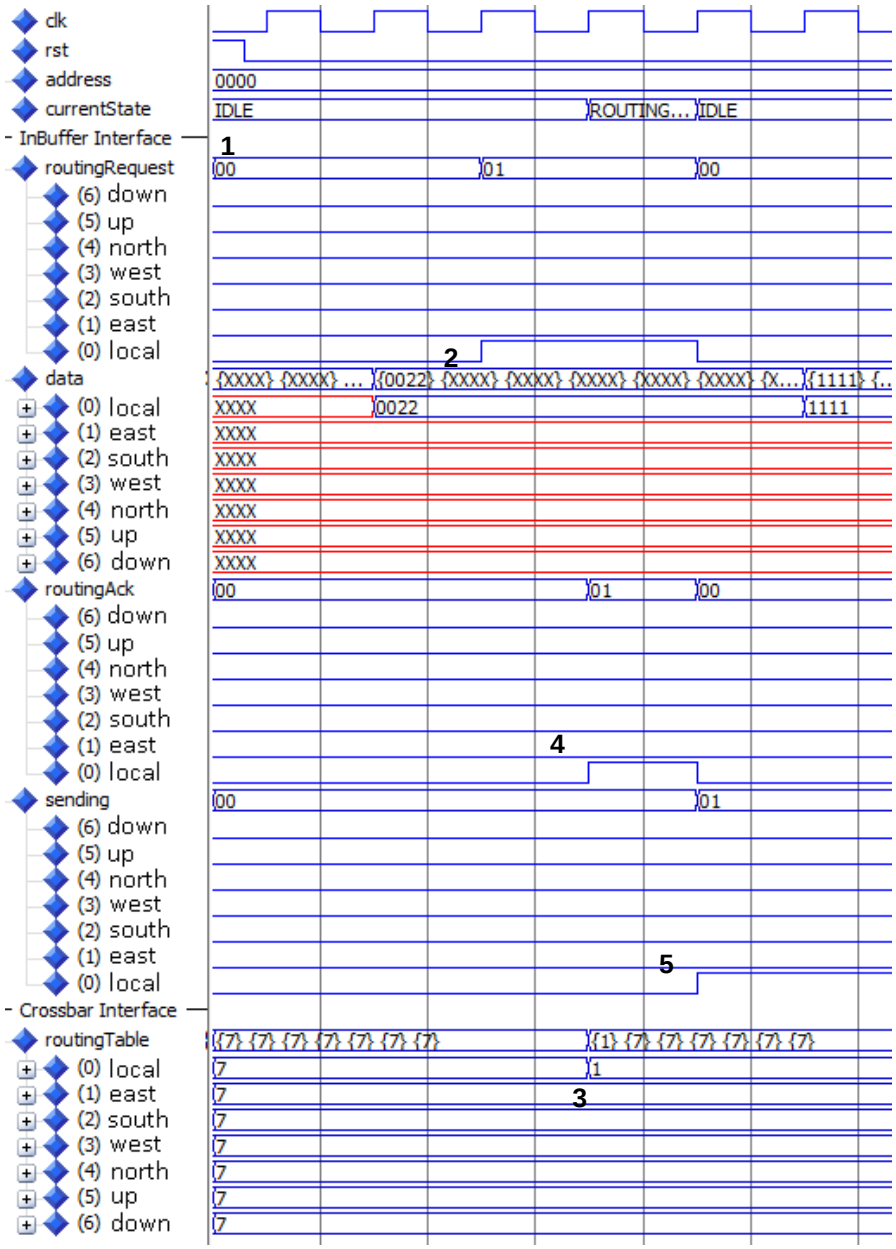


Figura 17 – Simulação do *switch control* recebendo uma solicitação de roteamento

1. Inicialmente no estado *idle*, o *Switch Control* aguarda por uma requisição de roteamento de algum *Input Buffer*.

2. As requisições de roteamento feitas pelos *Input Buffers* chegam através da entrada *routingRequest*. Na Figura 17 a requisição é feita pelo *Input Buffer local* (*routingRequest*(0)), a qual corresponde ao passo 4 da Figura 14. Quando uma requisição chega, o codificador de prioridades determina com base na última porta roteada, que possui a menor prioridade, a porta que deve ser atendida naquele instante. Como esta era a única requisição, foi atendida

imediatamente.

3. No caso desta requisição, a porta de saída selecionada pelo algoritmo de roteamento estava livre. Então o *Switch Control* escreve na *routingTable*, na linha correspondente ao *Input Buffer* que fez a solicitação, a porta de saída que deve ser utilizada. Desta forma o *Crossbar* sabe quais portas devem ser conectadas.

4. O estado muda para *routing ack* (*currentState*), neste estado um pulso de confirmação (*routingAck*) é enviado para o *Input Buffer* que teve requisição de roteamento atendida, informando que a transmissão pode ser iniciada. O pulso dura um ciclo e depois o sistema volta ao estado *idle*. Esse pulso é visto chegando no *Input Buffer* na Figura 14 passo 5.

5. Enquanto aguarda novas solicitações, no estado *idle*, o *Switch Control* fica constantemente monitorando o sinal *sending*, que é gerado pelos *Input Buffers* (Figura 14 passo 7). Se o *sending* correspondente a algum *Input Buffer* deixar de ficar alto, o *Switch Control* pode desalocar a porta de saída correspondente.

3.3 *Crossbar*

O *Crossbar* é responsável por realizar o chaveamento entre os *Input Buffers* e as portas de saída do roteador. As conexões são definidas pelo *Switch Control*, e enviadas para o *Crossbar*. O funcionamento deste módulo é totalmente combinacional e baseado na entrada de controle *table*, a qual é gerada pelo *Switch Control* (*routingTable*). Na Figura 18(a) é apresentada a interface do *Crossbar*, bem como na Figura 12.

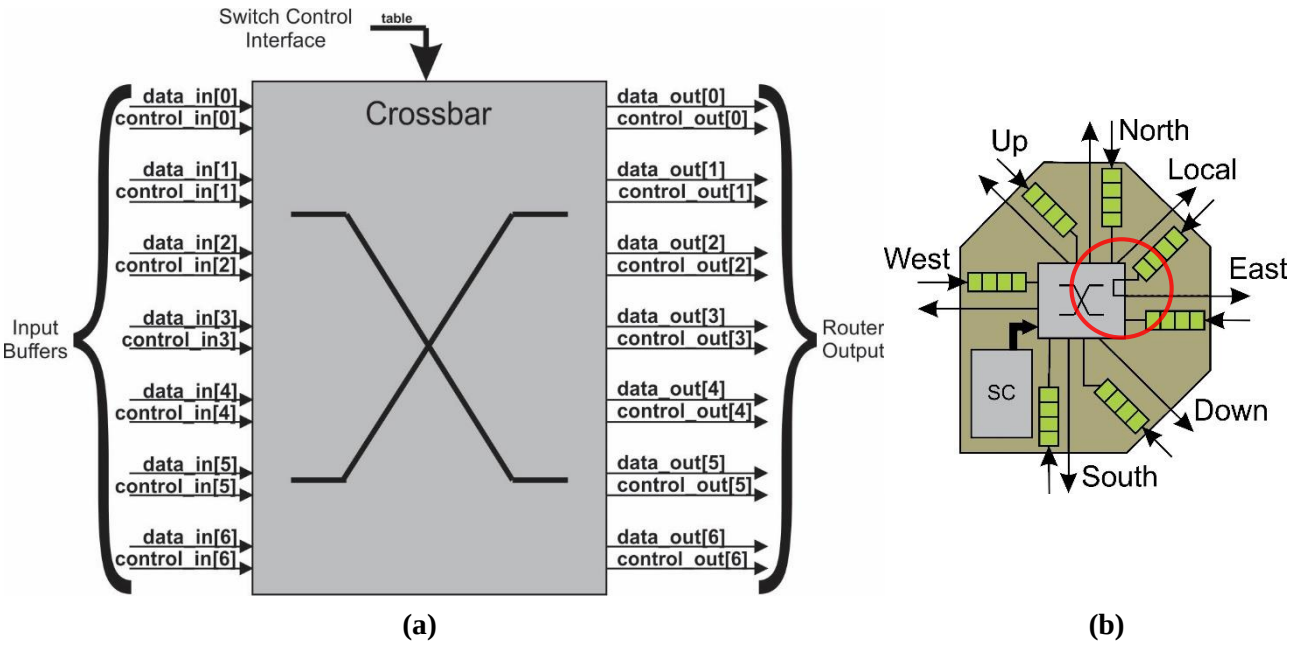


Figura 18 – (a) Interface do *Crossbar*. (b) Ligação entre o *Input Buffer* e a porta de saída *east* feita no *Crossbar*.

Na

Figura 18(b), pode-se ver a ligação entre a saída do *Input Buffer local* e a porta de saída *east*, ela é realizada pelo *Crossbar* através do controle que recebe do *Switch Control*. Esta conexão é mostrada na forma de onda apresentada na Figura 19 e em seguida é comentada.

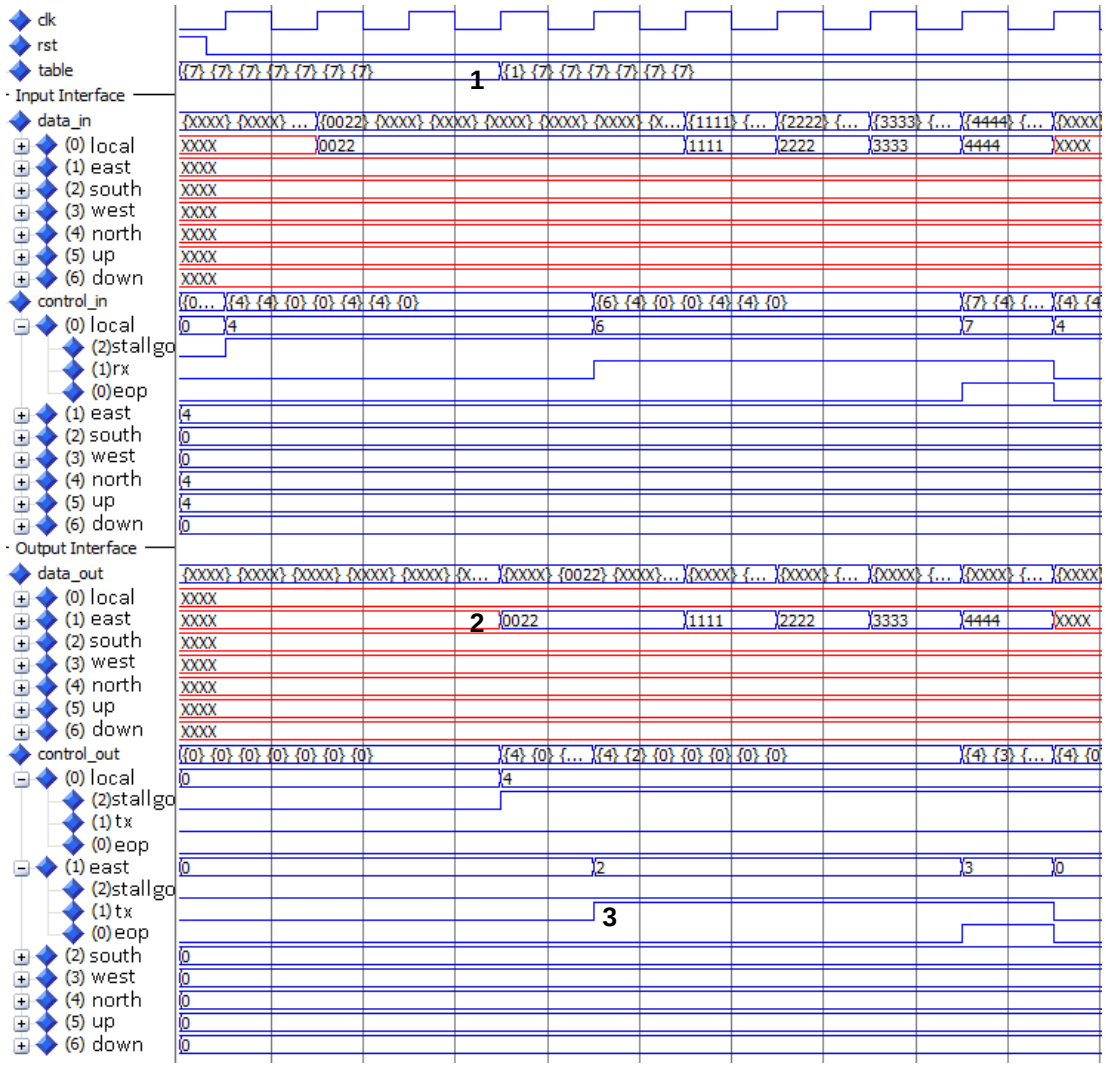


Figura 19 – *Crossbar* realizando a conexão entre um *Input Buffer* e uma porta de saída.

1. Devido ao fato do *Crossbar* ser um circuito combinacional, assim que a entrada *table* é alterada as conexões entre *Input Buffers* e portas de saída do roteador são realizadas imediatamente.

2. O dado que está chegando na interface de entrada do *Crossbar*, oriundo do *Input Buffer local* (*data_in*(0)), aparece na porta de saída *east* (*data_out*(1)).

3. Os sinais de controle *tx* e *eop* gerados pelo *Input Buffer local* aparecem na saída de controle da porta *east* (*control_out*(1)). Já o sinal *stall_go* segue fluxo contrário (porta de saída → *Input Buffer*), pois é responsável por informar se há ou não espaço para armazenar os *flits* que estão sendo transmitidos. É enviado do receptor para o transmissor.

3.4 Roteador

O roteador da Arke, é a peça fundamental na construção da NoC. É através da múltipla instanciação dos roteadores que se constrói a malha (2D ou 3D) que servirá como meio de comunicação para um SoC. Como foi mostrado na Figura 12, o roteador é composto a partir de três componentes: *Input Buffer*, *Switch Control* e *Crossbar*. A quantidade de portas que possui varia de acordo com a topologia e a sua posição na malha. Por exemplo, em uma NoC 3D de dimensões 3x3x3, como na Figura 16, um roteador na posição 000 (XYZ) possuirá somente quatro portas (*Local*, *East*, *North* e *Up*). Porém, um roteador central, como o 111 (XYZ), terá todas as suas sete possíveis portas. Agora, dada uma malha bidimensional, pelo fato da mudança na topologia, duas portas são removidas, *Up* e *Down*. Essas particularidades dos roteadores, bem como sua instanciação são tratadas no código VHDL da Arke, assim, não é necessário um *framework* dedicado a geração de código para a criação de uma NoC. Na Figura 21 o envio do pacote através do roteador, que foi detalhado nas seções anteriores, é mostrado sob a perspectiva do roteador. Pode-se acompanhar na Figura 20 (as ligações destacadas em vermelho), os dados chegam no roteador enviados do IP através da porta local e entram no *Input Buffer*. A requisição é feita ao *Switch Control* que configura o *Crossbar*, e o *Input Buffer* envia os dados que saem pela porta de saída *east.*

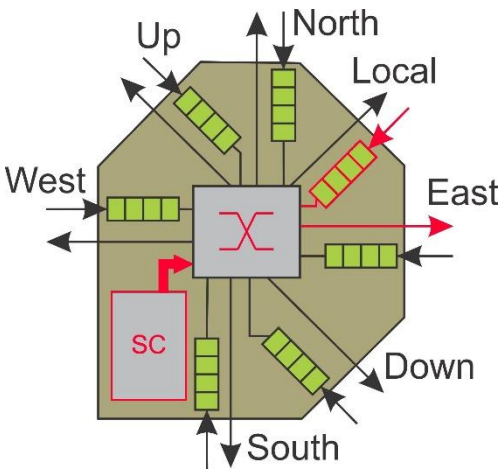


Figura 20 - Roteador da rede Arke. Em destaque, os componentes que participam do envio do pacote exemplo. SC – *Switch Control.*

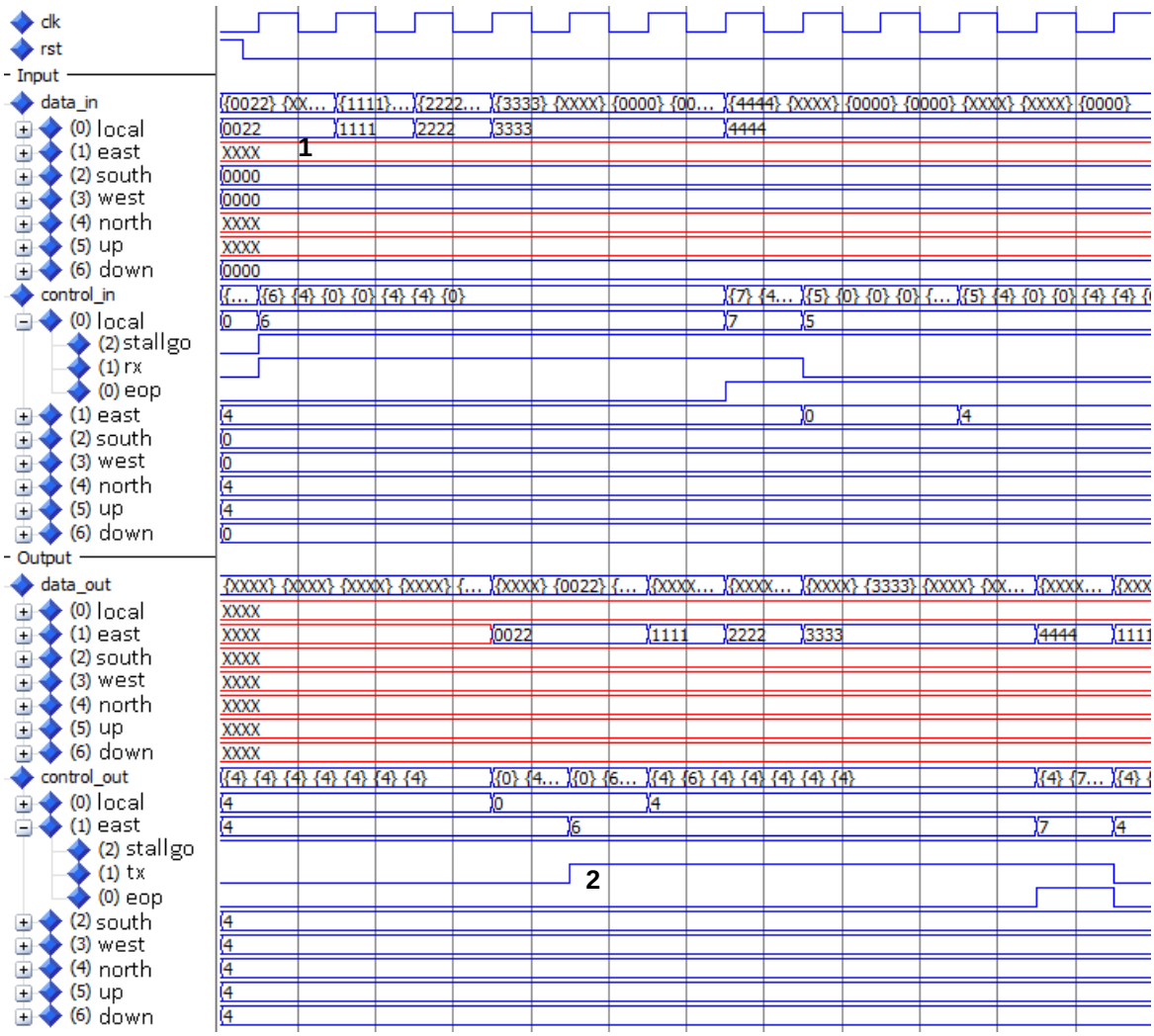


Figura 21 – Forma de onda mostrando a entrada e saída de um pacote em um roteador.

1. Chegada do pacote, enviado do IP para a NoC através da porta *Local* do roteador (*data_in*(0)) e (*control_in*(0)). Os *flits* serão armazenados no *Input Buffer* (passo 2 da Figura 14), e todo o processo de roteamento explicado nas seções anteriores se inicia.

2. Após a conclusão do roteamento, os dados passam a ser transmitidos para o roteador vizinho pela porta *east* (*data_out*(1) e *control_out*(1)). O *Input Buffer west* do roteador vizinho receberá estes dados e efetuará o mesmo processo que foi descrito nas seções anteriores. Isso se repetirá até que o pacote tenha chego no IP destinatário, completando assim a função da NoC.

4 RESULTADOS

A fim de determinar o desempenho da Arke, bem como compará-la com a Hermes, diversas sínteses, testes e simulações foram realizadas. As sínteses para o FPGA XC6SLX45T englobam reportes de recursos utilizados e frequências máximas atingidas pelos componentes. O FPGA XC6SLX45T possui tecnologia CMOS de 45 nanômetros, 43661 elementos lógicos e 190 portas de entrada/saída. As sínteses ASIC estimam o consumo de área, dissipação de potência e frequência máxima para as diversas configurações de roteadores e componentes da Arke e da Hermes. As simulações proporcionam a possibilidade de analisar aspectos referentes ao desempenho das diferentes topologias da Arke. As comparações entre a Arke e a Hermes foram feitas através dos roteadores e seus componentes, para que esta comparação fosse justa, os roteadores da Arke utilizados foram todos bidimensionais, uma vez que a Hermes possui apenas esta topologia.

4.1 Desempenho

Um fator importante atrelado a comunicação em NoCs é a latência, ou seja, o tempo que se passa entre o pacote ser disponibilizado pelo IP transmissor e o pacote ser entregue até o IP receptor. Analisando o comportamento da Arke e supondo um cenário onde nenhum pacote esteja sendo transmitido pela NoC, pode-se determinar o tempo mínimo para o envio de um pacote (latência minima) a partir do comportamento ilustrado na Figura 22.

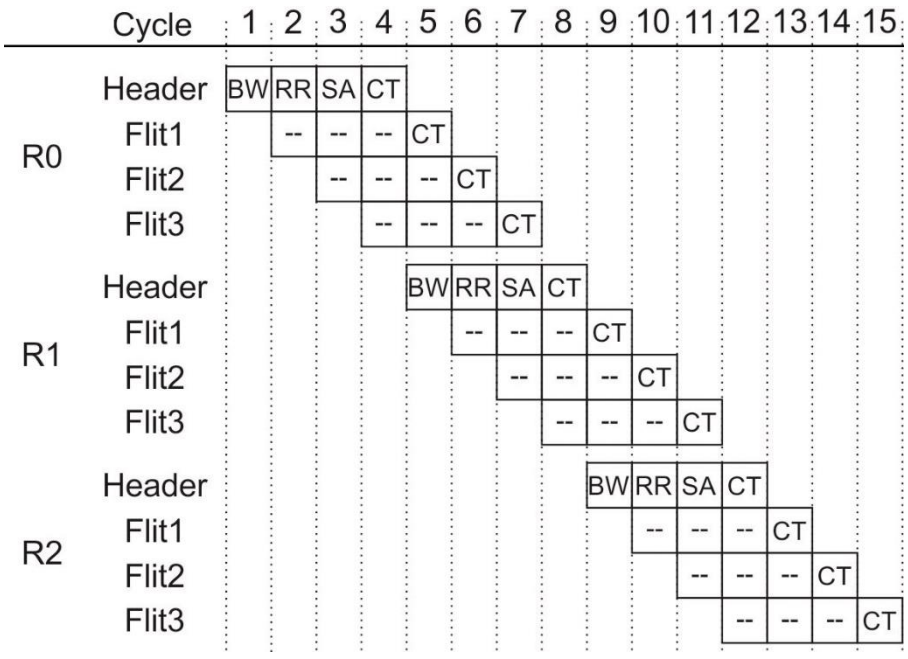


Figura 22 – Diagrama do pipeline de transmissão de pacotes do roteador da Arke. R0, R1 e R2 – Roteadores; BW – *Buffer Write*; RR – *Routing Request*; SA – *Switch Allocation*; CT – *Crossbar Traverse*

No exemplo apresentado, um pacote chega ao roteador R0 e no primeiro ciclo o cabeçalho é armazenado no *Input Buffer* (BW – *Buffer Write*). Em seguida, ocorre a requisição de roteamento

(RR – *Routing Request*) juntamente com a chegada do primeiro *flit* de *payload* do pacote. A cada ciclo de *clock* um novo *flit* de *payload* é armazenado enquanto o processo de roteamento ocorre. No terceiro ciclo uma porta de saída é alocada pelo *Switch Control* (SA – *Switch Allocation*) encerrando o processo de roteamento. No quarto ciclo ocorrer a transmissão do *flit* de cabeçalho para o roteador vizinho R1 e nos ciclos seguintes os *flits* que compõe o *payload* e estão armazenados no *Input Buffer* começam a ser transmitidos. Nos demais roteadores o mesmo processo se repete até que o pacote chegue ao destino. Através da Figura 22, pode-se determinar a Equação 1, que representa o tempo mínimo, em ciclos de *clock*, para enviar um pacote através da NoC.

$$T = (T_{BW} + T_{RR} + T_{SA} + T_{CT}) \cdot N_{rot} + (N_{flits} - 1) \cdot T_{tx} \qquad \textbf{Equação 1}$$

- T: tempo total de envio em ciclos de *clock*;
- T_{BW}: tempo necessário para que ocorra o armazenamento do dado na *queue*;
- T_{RR}: tempo necessário para que ocorra o envio da requisição de roteamento;
- T_{SA}: tempo de roteamento e alocação do caminho para o pacote;
- T_{CT}: tempo necessário para que um *flit* seja transmitido para o roteador vizinho;
- N_{rot}: quantidade de roteadores por onde o pacote irá passar;
- N_{flits}: quantidade de *flits* que forma o pacote, incluindo o cabeçalho;
- $N_{flits} - 1$: quantidade de *flits* que forma o *payload* do pacote;
- T_{tx}: tempo de envio de um *flit*.

Substituído os tempos T_{BW}, T_{RR}, T_{SA}, T_{CT} e T_{tx} pelo número de ciclos que cada um leva, Equação 2, e simplificando obtém-se a Equação 3.

$$T = (1 + 1 + 1 + 1) \cdot N_{rot} + ((N_{flits} - 1) \cdot 1) \qquad \textbf{Equação 2}$$

$$T = 4 \cdot N_{rot} + (N_{flits} - 1) \qquad \textbf{Equação 3}$$

A implementação que foi explorada até agora, possui quatro ciclos no pipeline do roteador (BW, RR, AS e CT). Originalmente esta não era a implementação principal, ela foi criada visando diminuir o caminho crítico do circuito, e consequentemente aumentar a frequência de operação da rede. Uma implementação alternativa do *Input Buffer* reduz em um ciclo o pipeline do roteador, pois engloba duas operações em um único ciclo. A Figura 23 mostra o diagrama do pipeline de três ciclos do roteador utilizando a implementação alternativa do *Input Buffer*.

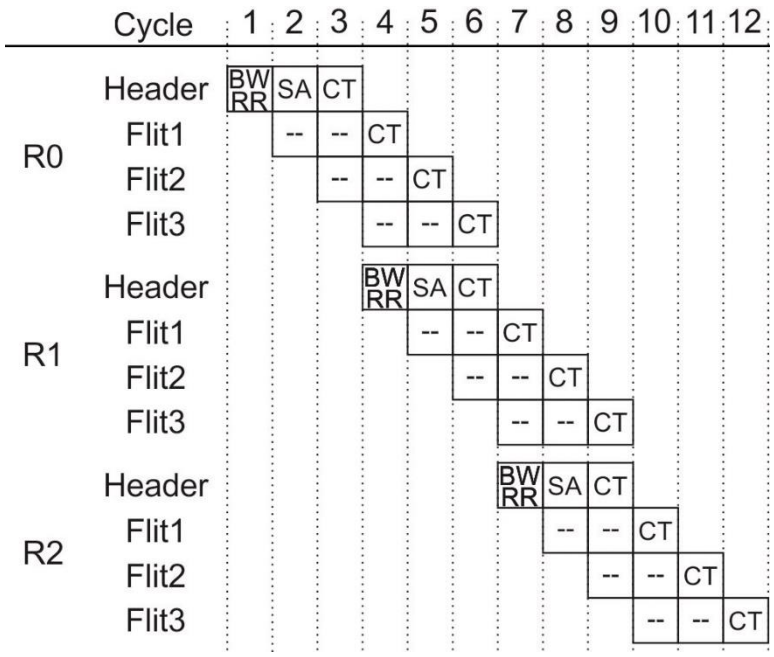


Figura 23 - Diagrama do pipeline de transmissão de pacotes do roteador da Arke. Considerando a arquitetura de 3 ciclos do *Input Buffer*. R0, R1 e R2 – Roteadores; BW – *Buffer Write*; RR – *Router Request*; SA – *Switch Allocation*; CT – *Crossbar Traverse*

Como nesta implementação um ciclo foi removido a equação que determina a latência mínima para a transmissão de um pacote deve ser alterada, uma vez que a escrita no *Input Buffer* e a requisição de roteamento acontecem no mesmo ciclo. Ao reescrever a equação, obtêm-se a Equação 4, substituindo e simplificando resulta na Equação 5.

$$T = \left(T_{BW/RR} + T_{SA} + T_{CT}\right) \cdot N_{rot} + (N_{flits} - 1) \cdot T_{tx} \qquad \textbf{Equação 4}$$

$$T = 3 \cdot N_{rot} + (N_{flits} - 1) \qquad \textbf{Equação 5}$$

Ambas as implementações, de quatro e três ciclos, estão disponíveis no código VHDL da Arke. Fica a cargo do projetista escolher a opção que melhor se adapte ao seu projeto.

Outro fator importante em relação ao desempenho da comunicação é a topologia escolhida para a NoC. A Arke oferece duas topologias de malha, bidimensional e tridimensional. Com o intuito de comparar o desempenho das topologias, foram criados dois cenários: (*i*) 64 IPs interconectados por uma NoC 2D 8x8 e (*ii*) 64 IPs interconectados por uma NoC 3D (4x4x4). Cada IP envia o mesmo número de pacotes, com a mesma quantidade de *flits* cada, para endereços aleatórios da rede. O envio dos pacotes é feito a uma taxa de injeção de 100%, ou seja, a cada ciclo um novo *flit* é injetado pelo IP na NoC, caso houver espaço para armazena-lo. Neste experimento a largura dos *flits* foi configurada em 16 *bits* e os *Input Buffers* possuíam filas de 8 posições. Depois de cada simulação foi obtido o tempo total para o envio de todos os pacotes. Diversas simulações foram feitas, de forma a reduzir as influências causadas pela aleatoriedade dos endereços destinos. Então o tempo médio para transmitir todos os pacotes em ambas as redes foi comparado, como mostra a Tabela 2.

Tabela 2 – Comparativo de desempenho entre as topologias oferecidas pela Arke.

Topologia	Tempo médio de transmissão (ciclos de *clock*)	% melhora do desempenho
2D	28458	36,93
3D	17948	

Com a mudança da topologia da rede para tridimensional, foi obtida uma redução no tempo da comunicação de 36,93%. Isso era esperado, uma vez que as redes tridimensionais se beneficiam de possuírem maior número de portas nos roteadores e menor diâmetro em relação as bidimensionais. Com até duas portas a mais (*UP* e *DOWN*), os roteadores tridimensionais possuem uma banda de transmissão maior que os roteadores bidimensionais. Em relação ao diâmetro da rede, por exemplo, em uma NoC 8x8 um pacote que percorre a distância máxima da rede (entre os roteadores de endereço 00 e 77) têm de ultrapassar 15 roteadores. Já em uma rede tridimensional com o mesmo número de roteadores (4x4x4), um pacote que percorre a distância máxima (entre os roteadores de endereço 000 e 333) necessita atravessar apenas 10 roteadores. Isso resulta em uma redução de trajeto de 33,33%. A soma destes dois fatores, aumento da banda de transmissão e menor diâmetro da rede, resulta na significativa melhora do tempo de transmissão que foi observado. Porém, existe um *trade off* que deve ser analisado, pois a opção de topologia 3D é tem um maior custo de área, visto que, os roteadores 3D possuem até mais duas portas em relação aos 2D.

4.2 Área

A fim de verificar o custo de implementação da Arke, foi feito um levantamento dos recursos que os componentes que formam a NoC ocupam. Inicialmente através do *framework PlanAhead* 14.7 da Xilinx, fornecido gratuitamente através de licença de estudante. Foi implementado cada um dos componentes que formam o roteador da Arke. Todas as sínteses foram feitas para o FPGA Spartan-6 XC6SLX45. E os resultados podem ser conferidos na Tabela 3.

Tabela 3 – Quantidades de recursos do FPGA Spartan-6 XC6SLX45 utilizados na implementação de cada um dos componentes do roteador Arke.

	Componente					
	***Input Buffer* 4 ciclos**	***Input Buffer* 3 ciclos**	***Switch Control* 3D**	***Switch Control* 2D**	***Crossbar* 3D**	***Crossbar* 2D**
Registradores	23	21	49	29	-	-
LUTs	45	45	183	128	432	200
Frequência máxima (MHz)	243,96	243,96	111,525	123,509	-	-

Na Tabela 3 pode ser vista a frequência máxima de operação e consumo de recursos do FPGA dos componentes que formam o roteador da Arke. A largura de *flit* de todos os componentes sintetizados estava configurada para 16 *bits* e os *Input Buffers* possuíam profundidade de 8 posições. Observa-se que há diferença entre os *Switch Controls* de 2D e 3D devido ao número de portas manipuladas por cada um deles. O *framework* PlanAhead não reporta frequência máxima para circuitos combinacionais, como o caso do *Crossbar*. Com o aumento das portas do roteador 2D em relação ao 3D (5 para 7 respectivamente, 40% de aumento), vê-se que a quantidade de LUTs aumenta cerca de 116%. Isto ocorre, porque, o *Crossbar* faz a ligação de cada *Input Buffer* com cada porta de saída, resultando em um crescimento não linear do circuito, em função do número de portas. Isto é esper Entre as duas arquiteturas de *Input Buffer* aparecem pequenas diferenças em relação aos recursos ocupados, apenas dois registradores. Isso porque na implementação de 4 ciclos há o armazenamento de um sinal em um registrador para aumenta o pipeline do circuito, como foi apresentado na Figura 22.

Em relação as duas arquiteturas de *Input Buffers*, quando avaliadas como circuitos isolados, suportam a mesma frequência máxima. No entanto, quando interconectados com os demais componentes do roteador, a frequência máxima suportada pelo roteador depende da interação entre o *Input Buffer* e *Switch Control*. Isto pode ser observado nos resultados de síntese de um roteador completo apresentados na Tabela 4.

Tabela 4 – Quantidades de recursos do FPGA Spartan-6 XC6SLX45 utilizados na implementação dos roteadores Arke.

	Roteadores			
	Roteador 3D (pipeline 4 ciclos)	**Roteador 3D (pipeline 3 ciclos)**	**Roteador 2D (pipeline 4 ciclos)**	**Roteador 2D (pipeline 3 ciclos)**
Registradores	164	144	114	104
LUTs	885	869	501	490
Frequência máxima (MHz)	104,875	87,281	123,847	93,557

Na Tabela 4 pode ser visto os resultados referentes a implementação de quatro roteadores Arke, com as duas implementações de *Input Buffers* para cada uma das duas topologias oferecidas. Os *flits* possuíam largura de 16 *bits* e os *Input Buffers* foram configurados em filas de 8 posições. Observe que os roteadores que utilizam a implementação de 4 ciclos se beneficiam do maior pipeline, que resulta em um aumento na frequência de operação. Este é o motivo pelo qual foi desenvolvida a implementação alternativa do *Input Buffer*. Dadas as frequências máximas reportadas pelo sintetizador do FPGA e as equações de latência mínima, é possível determinar o tempo em segundos para a transmissão de um pacote. Este tempo em segundos permite determinar qual pipeline se torna mais vantajoso em termos de latência mínima para o envio de pacotes através da rede. Por exemplo, em uma NoC que possua grandes distâncias entre os roteadores origem e destino (mais de 60 roteadores) o pipeline de três ciclos é mais eficiente para o envio de pequenos pacotes (menos de 5 *flits*); para pacotes maiores (mais de 5 *flits*) o pipeline de 4 ciclos já se mostra mais eficiente. A reta do gráfico da Figura 24 mostra o limiar de desempenho entre os pipelines de 3 e 4 ciclos.

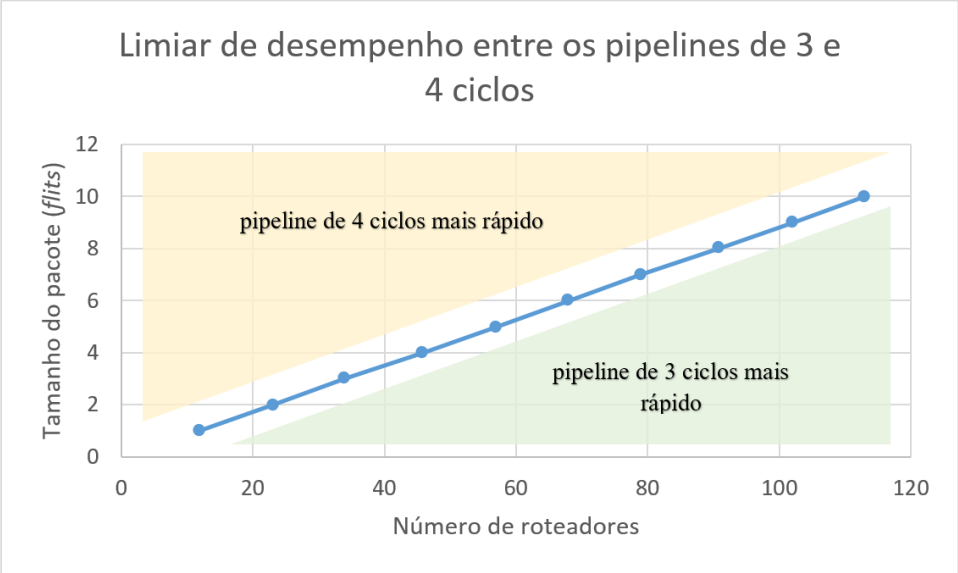
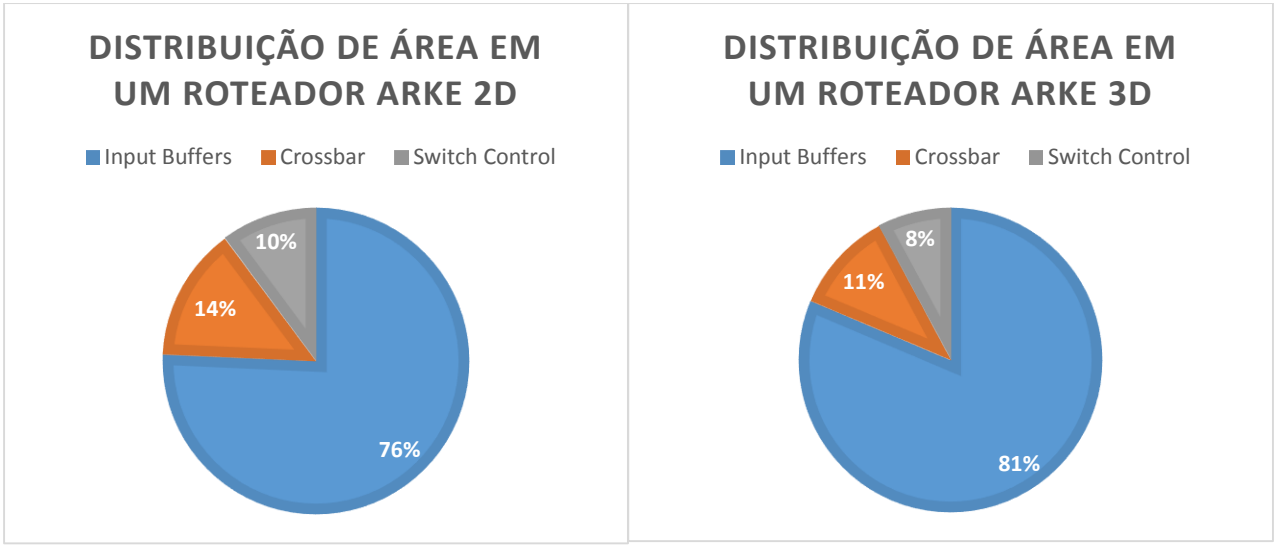


Figura 24 – Gráfico do limiar entre de desempenho entre os pipelines de 3 e 4 ciclos para as implementações em FPGA XC6SLX45 do roteador 2D da Arke.

O gráfico da Figura 24 foi criado com o objetivo demonstrar a exploração do espaço de projeto cabível na implementação de uma NoC. Cada ponto do gráfico é o limiar de desempenho entre as implementações, de 3 e 4 ciclos, do roteador. Por exemplo, dada uma quantidade de 46 roteadores o pipeline de 3 ciclos oferece desempenho superior para pacotes de até 4 *flits*. Desta forma, pode-se interpretar o gráfico da seguinte forma: a área superior a reta representa os casos onde a utilização do pipeline de 4 ciclos é mais vantajosa, já a área inferior à reta representa os casos onde a utilização do pipeline de 3 ciclos é mais eficiente. É importante frisar que este gráfico foi feito com base no tempo, medido em segundos, necessário para enviar os pacotes, calculados a partir das frequências reportadas pelo sintetizador do FPGA XC6SLX45, e que caso estas frequências se alterem a relação apresentada no gráfico sofrerá mudanças. Porém, se o projetista já tenha determinado uma frequência de operação para o SoC que seja suportada por ambas implementações (e.g. 50MHz), é mais conveniente escolher um pipeline menor.

Na Figura 25 pode ser observada a quantidade de área ocupada por cada um dos componentes que compõe o roteador da Arke. Nota-se que há diferença na distribuição dos componentes entre os roteadores 2D e 3D. Esta diferença é explicada pela adição de dois *Input Buffers*, responsáveis pelo recebimento e encaminhamento de pacotes na nova dimensão.



(a) (b)

Figura 25 – Distribuição da área ocupada pelos componentes dos roteadores da Arke.

A área dos roteadores não varia somente em função do número de portas. Uma vez que o roteador é feito de maneira a suportar o armazenamento temporário e transmissão de *flits*, espera-se que ele aumente linearmente de tamanho em função do crescimento da largura do *flit*. Afim de demonstrar esta característica foram realizadas diversas sínteses ASIC com roteadores de diferentes larguras de *flits*. Os resultados são apresentados no gráfico da Figura 26. As sínteses foram realizadas utilizando o pacote *standard cells* de 180 nanômetros da IBM e frequência de 155 MHz em todos os testes. O *software* utilizado para realizar as sínteses foi o *RTL Compiler* da Cadence.

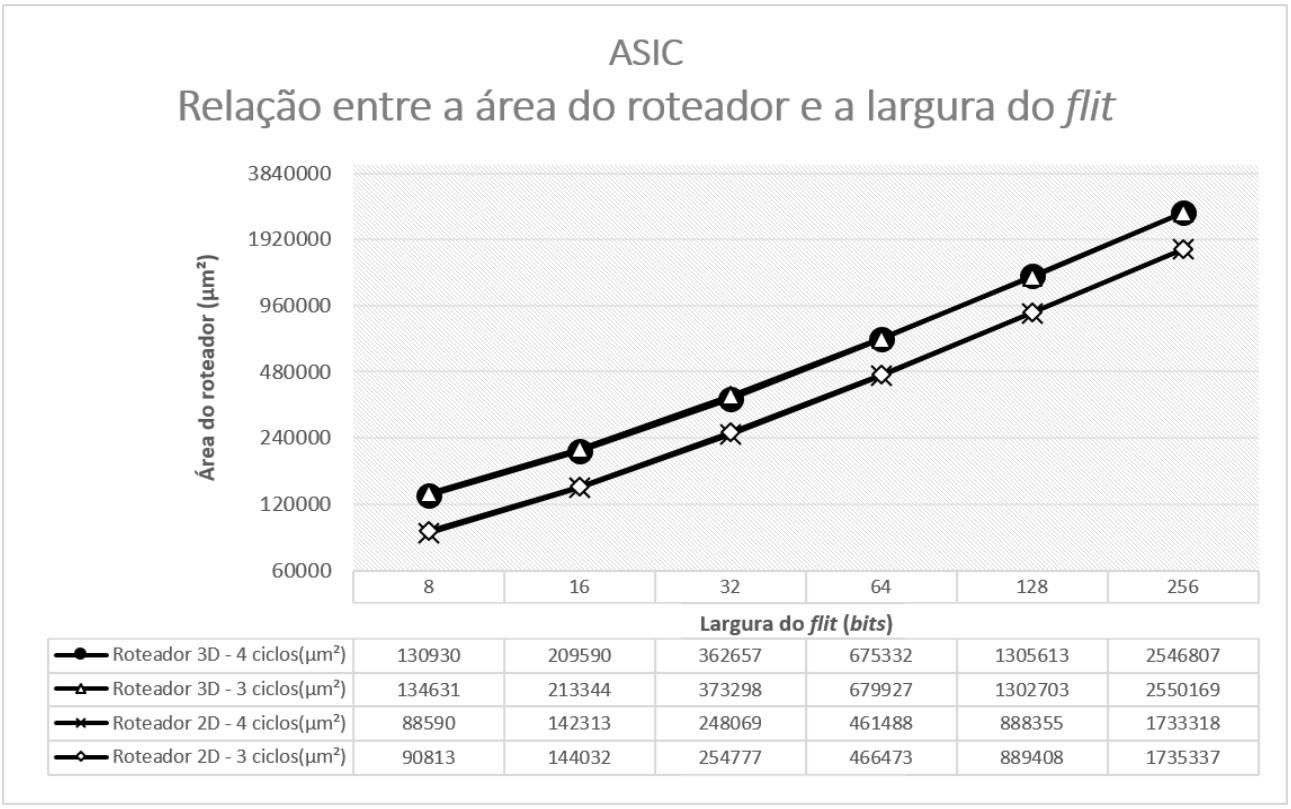


Figura 26 – Crescimento da área dos roteadores em função do aumento da largura dos *flits*.

No gráfico da Figura 26, observa-se a relação entre o aumento da largura dos *flits* e a área do roteador. No eixo das abcissas tem-se a largura dos *flits*, dada em bits, aumentando de forma exponencial de base dois. No eixo das ordenadas está representada a área, em micrometros quadrados, também crescendo exponencialmente com base dois. Representado pelas marcações de circulos, a variação da área dos roteadores tridimensionais que utilizam o pipeline de 4 ciclos no *Input Buffer*, já pelas marcações de xis tem-se a área dos roteadores bidimensionais que utilizam a arquitetura de 4 ciclos de pipeline no *Input Buffer*. É interessante observar que as áreas crescem de maneira linear com o aumento da largura dos *flits*, assim como previsto. Outro fato que pode ser constatado é que a proporção entre os roteadores tridimensionais e bidimensionais se mantem a mesma com o aumento da largura dos *flits*. Proporcionalmente os roteadores 3D são em média 47% maiores que os 2D, isto também é esperado, uma vez que a diferença entre os dois está no acréscimo de duas portas ao roteador tridimensional, o que resulta em um aumento de 40% do número de portas. Destacado através dos triângulos no gráfico, a área dos roteadores 3D que utilizam a implementação do pipeline de 3 ciclos no *Input Buffer*, mostra que a diferença entre as duas implementações é muito pequena, correspondente apenas a um registrador e algumas poucas operações lógicas. Isto ocorre da mesma maneira para os roteadores 2D que utilizam a implementação do pipeline de 3 ciclos do *Input Buffer*, marcados no gráfico com os losangos.

Além do crescimento da área, o aumento da largura de *flit* influência também no aspecto da potência dissipada. Uma vez que a dissipação de potência é diretamente relacionada a quantidade de chaveamentos dos transistores. Se aumentarmos a quantidade de *bits* dos *flits*, consequentemente *buffers* e *portas* necessitarão de mais transistores que gerarão maior chaveamento e por consequência dissiparão mais potência. O fluxo ASIC dá a oportunidade de estimar a potência dissipada para as diferentes larguras de *flits*. O gráfico da Figura 27 mostra os resultados obtidos através da síntese.

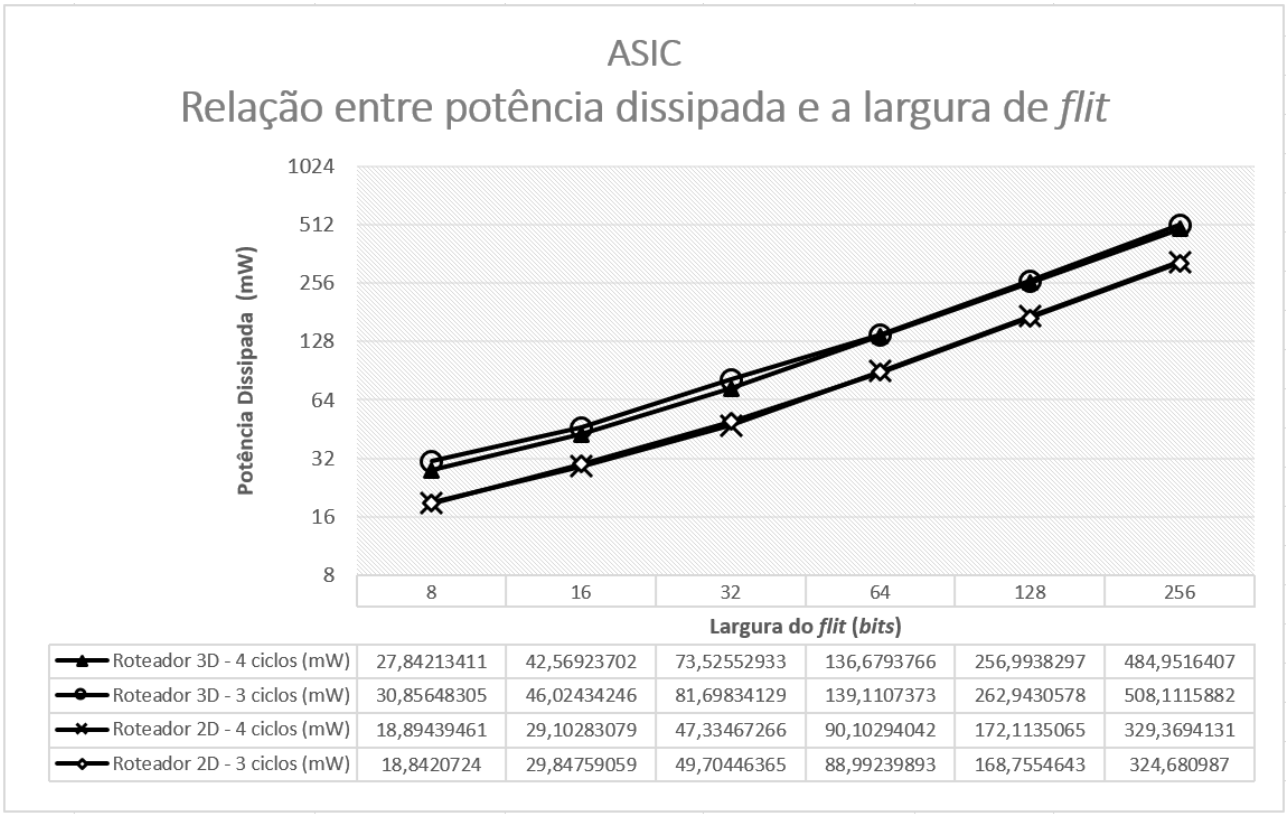


Figura 27 – Crescimento da potência dissipada pelo roteador em função do aumento da largura dos *flits.*

No gráfico da Figura 27, observa-se a relação entre o aumento da largura dos *flits* e a potência dissipada pelo roteador. No eixo das abcissas tem-se a largura dos *flits*, dada em *bits*, aumentando de forma exponencial de base dois. No eixo das ordenadas está representada a potência dissipada, medida em miliwatts, também crescendo exponencialmente com base dois. A diferença entre as implementações do roteador 3D é muito pequena no quesito da potência dissipada, assim como havia sido na área, e isto ocorre da mesma maneira para os roteadores 2D. É interessante lembrar que a potência gerada pela síntese é dada através de uma taxa de chaveamento das portas. Esta taxa, por padrão vem fixada em 20% da frequência do *clock* no RTL Compiler da Cadence e foi mantida assim para realizar estas simulações. A potência total que está representada no gráfico é dada pela soma da potência dinâmica e da potência estática das células que foram associadas ao projeto. A potência dinâmica vem de um processo de carga e descarga de capacitâncias de carga e depende da frequência, tensão, capacitância e fator de atividade. A potência estática vem da corrente de *leakage* e de curto circuito (LONDERO, 2015). Assim como a área, a potência dissipada também cresce linearmente em função do aumento da largura dos *flits*. Quando a largura dos *flits* aumenta, são necessários mais transistores para formar os *buffers* e realizar as operações de roteamento. Desta forma, podemos relacionar o aumento da potência diretamente com o aumento da área do circuito. Quando comparados os gráficos de aumento de área e do aumento da potência, nota-se que a taxa de crescimento das retas é muito semelhante, em torno de 55%, isto mostra a relação direta entre o consumo e a área do circuito.

4.3 Arke *vs* Hermes

Foi partir da NoC Hermes que teve início o projeto da NoC Arke. Além de adicionar uma nova topologia, gerar um código limpo e simples de ser compreendido e facilitar a geração de uma NoC parametrizável sem a utilização de *frameworks*. Manteve-se a ambição de criar algo que fosse equivalente ou superior ao original. Com o objetivo de verificar o desempenho da Arke em relação à Hermes, foram feitas algumas comparações entre Hermes e Arke. Inicialmente, de maneira a determinar o desempenho em relação ao tempo mínimo necessário para enviar um pacote através das NoCs. Foi feita a comparação entre as equações de latência mínima. A equação de latência mínima da Hermes foi obtida a partir da simulação da sua descrição VHDL, que é disponibilizado pelos desenvolvedores gratuitamente.

$$T = 3 \cdot N_{rot} + (N_{flits} - 1) \qquad \textbf{Equação 6}$$

$$T = 4 \cdot N_{rot} + (N_{flits} - 1) \qquad \textbf{Equação 7}$$

$$T = 6 \cdot N_{rot} + (N_{flits} - 1) \qquad \textbf{Equação 8}$$

A Equação 6 é referente a Arke com implementação de 3 ciclos do *Input Buffer*, a Equação 7 a Arke com implementação de 4 ciclos do *Input Buffer* e a Equação 8 é referente a Hermes. Considerando ambas NoCs operando na mesma frequência, tem-se a Arke apresentando uma latência mínima menor que a hermes graças ao menor tempo de roteamento. A Arke reduz até 3 ciclos por roteador no caso da utilização da implementação de 3 ciclos do *Input Buffer*. Se a implementação escolhida for a de 4 ciclos do *Input Buffer*, a Arke caba tendo 2 ciclos por roteador de vantagem em relação a Hermes. Isto representa uma redução entre 50% e 33% de tempo de roteamento, caso o projetista opte pela NoC Arke para executar a comunicação de seu SoC.

Outro fator importante e que é determinante para a escolha da NoC é o custo em relação a área ocupada por ela. Por este motivo, foram feitas implementações da Hermes em ASIC e FPGA a fim de comparar com os resultados de área obtidos. Devido ao fato da Hermes possuir somente a topologia 2D, as comparações realizadas são com componentes de topologia 2D da Arke.

Tabela 5 – Comparação entre *Input Buffer* da Arke e da Hermes

	Componente		
	***Input Buffer* Arke (3 ciclos)**	***Input Buffer* Arke (4 ciclos)**	***Input Buffer* Hermes**
Síntese FPGA – Spartan 6 XC6SLX45			
Registradores	23	21	26
LUTs	45	45	64
Frequência Máxima (MHz)	243,96	243,96	193,424
Síntese ASIC - *Standard Cells* IBM 180nm			
Área (µm²)	13175	13666	15301
Potência (mW)	2,084	2,161	3,124

A Tabela 5 apresenta os resultados de síntese FPGA e síntese ASIC para os *Input Buffers* da Arke e da Hermes, considerando 8 posições e 16 *bits* (largura do *flit*). A síntese ASCI foi realizada utilizando a frequência fixa de 165,56 MHz para todos os componentes, pois a mudança da frequência implica em variações de área e potência devido a utilização de células distintas. Em relação aos recursos utilizados pelo FPGA ambos *Imput Buffers* da Arke possuem vantagem em relação ao da Hermes, ocupando até 19,23% menos registradores e 30% menos LUTs. Já em relação a área medida através da síntese ASIC, o *Input Buffer* da Arke ocupa até 13,9% menos área para a implementação de 3 ciclos e 10,7% para a implementação de 4 ciclos. Além da área, nas implementações para FPGA, ambos os *Input Buffers* da Arke alcançam frequência de operação superior ao da Hermes. Em relação a potência estimada na síntese ASIC ambos os *Input Buffers* da Arke consomem menos, chegando até a 33,3% menos potência dissipada.

Tabela 6 - Comparação entre os *Switch Controls* da Arke e da Hermes.

	Componente	
	***Switch Control* Arke**	***Switch Control* Hermes**
Sintese FPGA – Spartan 6 XC6SLX45		
Registradores	29	57
LUTs	128	90
Frequência Máxima (MHz)	123,509	91,706
Síntese ASIC – *Standard Cells* IBM 180nm		
Área (µm²)	9145	12309
Potência (mW)	2,141	2,088

A Tabela 6 apresenta os resultados da implementação em FPGA e da síntese ASIC do *Switch Control* de ambas as NoCs. A frequência máxima de operação em FPGA do *Switch Control* da Arke é maior que a reportada para a Hermes. Em relação aos recursos utilizados pelo FPGA, a Arke utiliza 49% menos registradores, porém acaba fazendo uso de 42,22% mais LUTs. Na síntese ASIC pode ser vista que a vantagem no uso das LUTs pela Hermes não supera o ganho que a Arke possui em relação aos registradores, visto que a área utilizada pelo *Switch Control* da Arke acaba sendo 25,7% menor que o da Hermes. Porém, em relação a potência dissipada, o *Switch Control* da Hermes acaba dissipando 2,5% menos potência que o da Arke.

Tabela 7 – Comparação entre o *Crossbar* da Arke e da Hermes.

	Componente	
	Crossbar Arke	**Crossbar Hermes**
Sintese FPGA – Spartan 6 XC6SLX45		
LUTs	200	174
Síntese ASIC – *Standard Cells* IBM 180nm		
Área (µm²)	12774	7922
Potência (mW)	0,546	0,278

Na Tabela 7 é apresentada a comparação entre as implementações em FPGA e sínteses ASIC para o componente *Crossbar*. O *Crossbar* é um componente combinacional, por este motivo o sintetizador não reporta frequência máxima de operação e nem registradores na implementação em FPGA. Porém, em relação as LUTs utilizadas do FPGA a Hermes acaba ocupando 13% menos em relação a Arke. Por fim, através da síntese ASIC, fica claro que o *Crossbar* da Hermes realmente ocupa uma área menor, chegando a 38% de diferença em relação ao da Arke. Além da área maior, o *Crossbar* da Arke também acaba dissipando mais potência que o da Hermes. A diferença observada nas áreas dos componentes se deve ao fato de que o Crossbar da Hermes é incompleto, não possibilita a conexão de uma mesma entrada e saída, por exemplo, não há conexão entre a porta de entrada *East* e a porta de saída *East*. A remoção desse tipo de conexão na Hermes foi feita com a finalidade de otimização visto que o algoritmo de roteamento a proíbe. Na Arke, dois motivos principais levaram a decisão de deixar estas ligações. A primeira, que buscando legibilidade do código, sempre que possível foram utilizadas estruturas de repetição para gerar código parametrizável. No caso do *Crossbar* da Arke, para que fosse possível ser gerado por esse tipo de estrutura (*for ... generate*), foi necessário deixar esse tipo de conexão. O segundo motivo é o fato de que existem algoritmos de roteamento que utilizam este tipo de conexão. Como a Arke foi concebida para que outros projetistas também possam contribuir futuramente no desenvolvimento da NoC ou até mesmo realizarem pesquisas utilizando ela, a opção de deixar o *Crossbar* completo foi escolhida, não sendo necessária fazer nenhuma mudança para comportar novos algoritmos de roteamento.

Tabela 8 - Comparação entre o Roteador 2D (5 portas) da Arke e o Roteador da Hermes.

	Componente		
	Roteador Arke 2D (3 ciclos)	**Roteador Arke 2D (4 ciclos)**	**Roteador Hermes**
Sintese FPGA – Spartan 6 XC6SLX45			
Registradores	104	114	187
LUTs	490	501	580
Frequência Máxima (MHz)	93,557	123,847	95,389
Síntese ASIC – *Standard Cells* IBM 180nm			
Área (µm²)	89083	88590	98287
Potência (mW)	19,855	18,417	21,045

A Tabela 8 apresenta as implementações para FPGA e as sínteses ASIC para os roteadores 2D da Arke e o roteador da Hermes. Nesta implementação fica claro o motivo da implementação alternativa do *Input Buffer* com pipeline de 4 ciclos. A frequência máxima alcançada pelo roteador utilizando tal implementação é 32,37% maior em comparação ao de 3 ciclos. Porém há o acréscimo de registradores e LUTs, mas continua não ultrapassando a Hermes que ocupa até 79,8% mais registadores e 18,37% mais LUTs. Já nas sínteses ASIC, um fato inesperado foi constatado, a implementação de 3 ciclos acabou ocupando uma área maior que a de 4 ciclos. Isto não era esperado, porém, algumas optimizações foram feitas pelo sintetizador, que acabou gerando um circuito menor. Da mesma forma a potência dissipada para o roteador de 3 ciclos foi ligeiramente maior que a estimativa para o roteador de 4 ciclos. Na comparação final, ambas as arquiteturas da Arke acabam consumindo menor área de circuito e dissipando menos potência que o roteador da Hermes. Chegando a uma diferença de 10% de área e 12,5% de potência dissipada em relação ao roteador da Hermes.

Todas as sínteses ASIC feitas do início desta seção até aqui, foram feitas em uma frequência fixa, 165,56 MHz. Isto porque a variação da frequência força o sintetizador a escolher células diversas o que faria com que as comparações de área não fossem justas. Então, para verificar o desempenho em relação a frequência de operação que os roteadores podem alcançar mantendo o mesmo circuito (conjunto de células), outras sínteses foram realizadas, desta vez, forçando o sintetizado até a máxima frequência possível para o circuito. As máximas frequências que foram alcançadas para os roteadores da Arke e Hermes, utilizaram o pacote *Standard Cells* 180 nanometros da IBM, através do RTL Compiler, podem ser visualizadas na Tabela 9.

Tabela 9 – Comparação entre as frequências máximas alcançadas nos roteadores da Arke e Hermes (ASIC).

	Componente		
	Roteador Arke 2D (3 ciclos)	**Roteador Arke 2D (4 ciclos)**	**Roteador Hermes**
Frequência Máxima (MHz)	198,491	200,000	170,126

Todos os roteadores que foram sintetizados, da Arke e da Hermes, possuíam largura de *flit* de 32 *bits*, 8 posições no *Input Buffer* e 5 portas. Nota-se que o ganho na frequência ao adotar a implementação de 4 ciclos do *Input Buffer* não chega a 1%, diferente do que foi apresentado pela implementação em FPGA, que resultava em até 24,4% de incremento da frequência de operação. Ambas versões do roteador da Arke apresentam uma frequência aproximadamente 15% superior ao da Hermes.

4.4 Prototipação em FPGA

A fim de prototipar em FPGA a NoC Arke, foi criado um pequeno sistema utilizando uma rede de dimensões 3x3x3. A placa de prototipação utilizada foi a Digilent Atlys com o FPGA Spartan-6 XC6SLX45. O sistema é basicamente composto de um emissor de *flits* (*Data Manager - Sender*) que é acionado por um botão da placa de prototipação e um receptor de *flits* (*Data Manager - Receiver*) que recebe os *flits* da NoC e exibe o último *flit* recebido através dos LEDs da placa de prototipação. O esquema do sistema está representado na Figura 28.

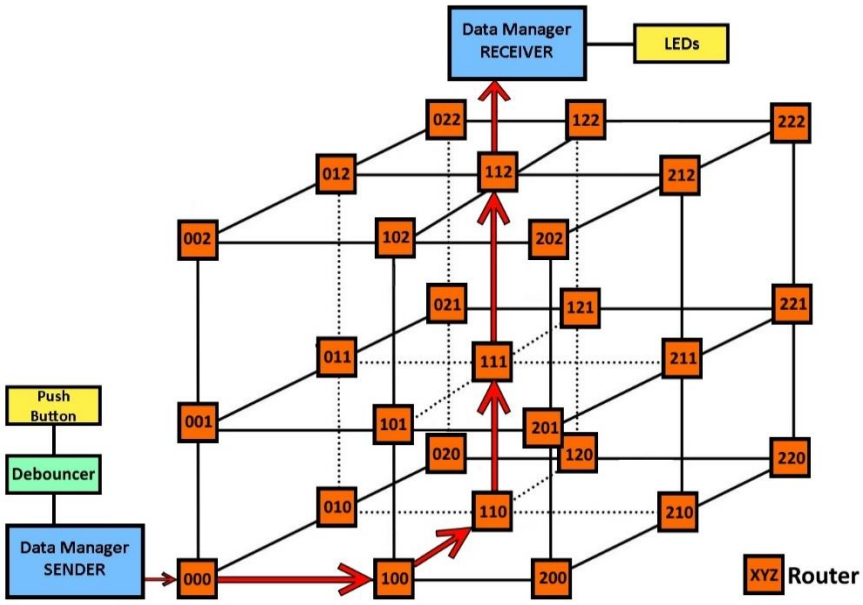


Figura 28 – Representação do sistema prototipado em FPGA da NoC Arke.

Os módulos *Data Manager* possuem dois modos de operação, um onde ele faz o envio de *flits* para um roteador (*sender*) e outro onde ele recebe os *flits* do roteador (*receiver*). O *Data Manager* conectado ao roteador '000' foi configurado de forma a trabalhar como *Sender*, acionado através do clique de um botão da placa de prototipação. O botão está conectado a um módulo de *debouncer* para filtrar o ruído que é gerado pelo clique. O pacote enviado é sempre o mesmo e é formado por dezesseis *flits* (8 *bits* de largura), onde o primeiro *flit* é o cabeçalho que contém o endereço destino do pacote ("00010110" → "00-XX-YY-ZZ" → X=1, Y=1 e Z=2) e os demais *flits* (*payload*) tem os seguintes valores: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE e 0xFF. Quando configurado em modo *Receiver* o *Data Manager* recebe os *flits* através da porta *Local* do roteador a qual ele está conectado e os exibe em sua saída. Na prototipação, o *Data Manager* conectado ao roteador '112' foi configurado como *Receiver* e o sinal de saída que informa o último *flit* recebido foi conectado aos LEDs da placa.

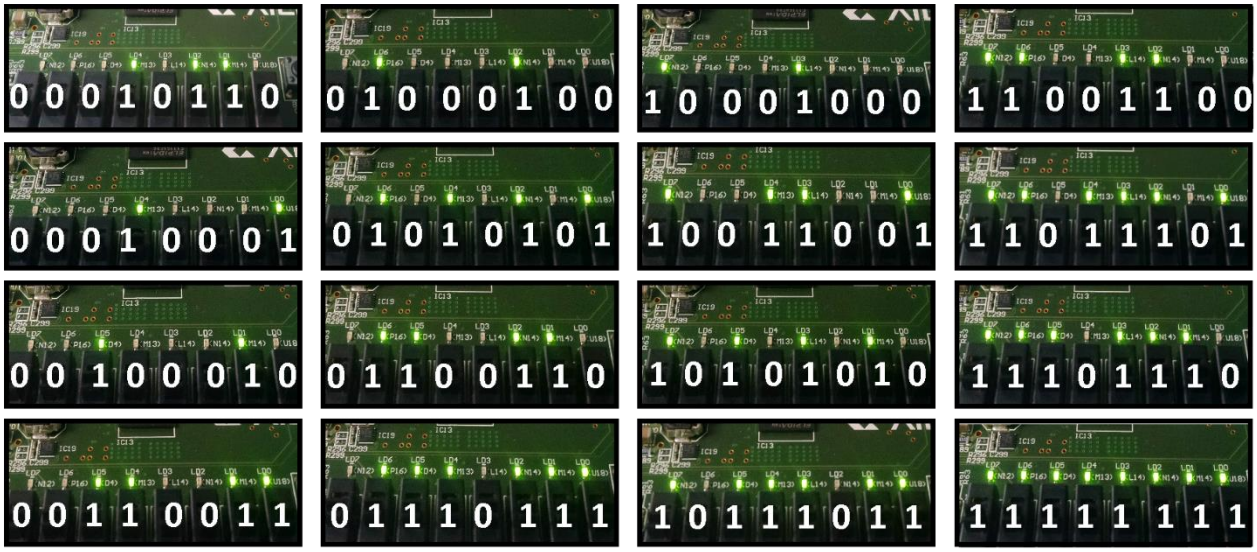


Figura 29 – Imagens da exibição dos *flits*, no FPGA, que foram transmitidos pela NoC.

Na Figura 29 é mostrado o resultado da prototipação, onde cada quadro da imagem mostra um *flit* do pacote que foi enviado do *Data Manager Sender* para o *Data Manager Receiver* pela NoC Arke.

5 CONCLUSÃO

Este trabalho apresentou o projeto, a implementação e a avaliação da NoC Arke. Uma das vantagens do emprego de SoCs para integrar diversos núcleos de processamento está na velocidade de troca de informações entre os núcleos, que é superior se comparada com a comunicação entre diferentes sistemas implementados em múltiplos circuitos integrados. Para realizar esta integração é necessária uma infraestrutura de comunicação entre os IPs e o emprego de NoCs oferece diversas vantagens em relação a outras estruturas.

Com relação ao desempenho e características, foi feito um resumo comparativo entre os roteadores da Arke (2D) e a Hermes, como pode ser visto na Tabela 10.

Tabela 10 – Resumo da comparação entre as NoCs Arke e Hermes.

		Arke	**Hermes**	**Relação**
Área FPGA (Spartan-6)	**Registradores**	114	187	39,04% ↓
	LUTs	501	580	13,62% ↓
Frequência máxima FPGA (Spartan-6) (MHz)		123,847	95,389	29,83% ↑
Área ASIC (IBM 180nm) (μm^2)		88590	98287	9,86% ↓
Frequência máxima ASIC (IBM 180nm) (MHz)		200,000	170,126	17,56% ↑
Potência ASIC (IBM 180nm) (mW)		18,418	21,046	12,48% ↓
Numero de ciclos de roteamento		3 ou 4	6	50% ↓ a 33,33% ↓
Topologia		2D e 3D	2D	
Geração Automatizada		Através de parâmetros da descrição VHDL	Baseada em *framework*	

Uma das ambições almejadas durante a descrição VHDL da Arke era que ela possuísse características superiores ou equivalentes às da Hermes. Em todos os quesitos avaliados na Tabela 10 a Arke atingiu satisfatoriamente os objetivos. Isto ocorreu devido ao fato de que a Arke é produto de

estudos da Hermes, estes levaram a constatação de diversos pontos não otimizados. Através da criação de uma nova NoC muitos aspectos foram otimizados, que se mostam evidentes na Tabela 10.

Um ponto interessante, que foi percebido através dos resultados, é que por mais que a potência dissipada pelos roteadores tridimensionais seja maior que os bidimensionais, o tempo de permanência dos pacotes dentro das redes 3D é em média menor. Como se sabe, a energia consumida é determinada pela potência dissipada em um determinado período de tempo. Devido a esta relação é esperado que ao utilizar uma rede 3D para enviar uma determinada quantidade de pacotes a quantidade de energia consumida seja menor do que em uma rede 2D.

É importante ressaltar que todos os roteadores instanciados, através da descrição, possuem o número máximo de portas referentes a sua dimensão (5 portas para os 2D e 7 portas para os 3D). Porém, visto os testes em FPGA e ASIC foi possível constatar que as portas aterradas, que não possuem conexão com roteadores vizinhos, são removidas pelos sintetizadores. Desta forma, é deixado por conta do sintetizador estas otimizações, graças a isso é possível a criação de uma NoC a partir de uma única implementação do roteador, ao contrário da Hermes, que possui uma implementação de roteador para cada tipo de posição que o roteador pode assumir na rede.

As principais contribuições do presente trabalho foram: (*i*) desenvolvimento da NoC Arke, funcional para reuso em outros projetos; (*ii*) análise de recursos utilizados na implementação em FPGA; (*iii*) análise de área e potência através de síntese ASIC; (*iv*) comparação de desempenho entre a NoC Arke e a NoC Hermes; (*v*) desenvolvimento de um guia para a utilização da NoC Arke, que está no Apêndice 1 deste trabalho. Além disto, este trabalho proporcionou aos envolvidos uma grande experiência em arquitetura de computadores, redes de comunicação, projeto e implementação de sistemas embarcados. A descrição em VHDL da Arke bem como o *starter guide* se encontram disponíveis no sistema de controle de versões *guithub* (guithub.com/iacanaw/Arke) e podem ser acessados, utilizados e replicados de maneira livre.

A partir deste trabalho, podem ser enumeradas algumas direções para pesquisas futuras, dentre as quais:

- Estudos de novos algoritmos de roteamento para topologias tridimensionais;
- Realização de testes de dissipação de potência dinâmica, desta vez, considerando uma atividade de chaveamento real;
- Estudar técnicas de compactação de pacotes em tempo de transmissão, e com isso, aumentar o desempenho da rede;
- Prototipação de uma NoC tridimensional em um FPGA 2.5D, como por exemplo a Virtex-7

2000T;

- Modelagem de um sistema com propósito específico que utilize a NoC Arke como infraestrutura de comunicação e posterior design e fabricação deste.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ABDALLAH, A. B. (2013). *Multicore Systems On-Chip: Pratical Software/Hardware Design.* Japão: Atlantis Press.

AMORY, A. (30 de 11 de 2015). Fonte: Atlas - Hermes - GAPH PUCRS: https://corfu.pucrs.br/redmine/projects/atlas

BENINI, L., & MICHELI, G. d. (2002). Networks on Chip: A New Paradigm for Systems on Chip Design. *IEEE Computer*, 70-78.

CULLER, D., GUPTA, A., & SINGH, J. (1998). *Parallel Computer Architecture: a Hardware Software Approach.* Los Altos, California: Morgan Kaufmann.

DENG, Y., & MALY, W. (2005). 2.5-Dimensional VLSI System Integration. *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 13, NO. 6,* 668-677.

FEERO , B., & PANDE, P. P. (2007). Performance Evaluation for Three-Dimensional Networks-On-Chip. *IEEE Computer Society Annual Symposium on VLSI* (pp. 305 - 310). Porto Alegre: IEEE.

FEERO, B. S., & PANDE, P. P. (2009). Networks-on-Chip in a Three-Dimensional. *IEEE TRANSACTIONS ON COMPUTERS*, 32-45.

FENG, T. (1981). A Survey of Interconnection Networks. *Computer V14,n12*, 12-27.

GUERRIER, P., & GREINER, A. (2000). A generic architecture for on-chip packet-switched interconections. *Design Automation and Test in Europe*, (pp. 250-256).

GUPTA, R. K. (1997). Introducing Core-Based. *IEEE*, 15-25.

HWANG, K. (1993). Advanced Computer Architecture: Parallelism, Scalability, Programmability. *McGraw-Hill*, 770.

KUMAR, S. (2002). A Network on Chip Architecture and Design Methodology. *IEEE Computer Society Annual Symposium on VLSI*, (pp. 105-112).

LONDERO, L. (2015). *Notas de Aula: Projeto de Sistemas Digitais Integrados - Timing and Power.* Santa Maria: UFSM.

MARTIN, G., & CHANG, H. (2001). System on Chip Design. *9th International Symposium on Integrated Circuits, Devices & Systems*, (p. Tutorial 2).

MELLO, A. (2005). MAIA - A Framework for Network on Chip Generation and Verification.

ASIA AND SOUTH PACIFIC - DESIGN AUTOMATION CONFERENCE, ASP-DAC. Piscataway, NJ: IEEE, 2005 (a). v.1.

MOHAPATRA, P. (September de 1998). Whormhole Routing Techniques for Directly Connected Multicomputer Systems. *ACM Computing Surveys, v.30, n.3*, p. 374.

MOORE, G. E. (1965). Cramming more components onto integrated circuits. *Electronics Magazine*.

MORAES, F. (2004). HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip. *Integration, the VLSI Journal*, 69–93.

NI, L., & McKINLEY, P. (1993). A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer Magazine, [S.1], v.26, n.2*, 62-76.

NUGENT, S. (1988). The iPSC/2 Direct Connect Communicaions Technology. *Conferece on Hypercube Concurrent Computers and Applications, 3* (pp. 51-59). Pasadena: New York : ACM Press.

PALMA, J. (2003). *Um Estudo Sobre Redes de Conexão Intra-Chip.* Porto Alegre: Trabalho Individual (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS.

PALMA, J. C. (2007). *Reduzindo o Consumo de Potência em Redes Intra-Chip através de Esquemas de Codificação de Dados.* Porto Alegre: UFRGS.

PAVLIDIS, V. F., & FRIEDMAN, E. G. (2007). 3-D Topologies for Networks-on-Chip. *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, 1081-1090.

PAVLIDIS, V., & FRIEDMAN, E. (2007). 3-D Topologies for Networks-on-Chip. *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS* (pp. 1081-1090). Porto Alegre: IEEE.

RASMUSSEN, M. (2006). *Network-on-Chip in Digital Hearing Aids.* Denmark: DTU - IMM.

RIJPKEMA, E. (2001). A Router Architecture for Network on Silicon. *In: Workshop Progress, 2001*. Netherlands.

SABAN, K. (11 de Dezembro de 2012). Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency. *XILINX: Virtex-7 FPGAs - WP380 (v1.2)*.

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

APÊNDICE 1 – ARKE STARTER GUIDE

Arke Netwok-on-Chip / Starter Guide

Universidade Federal de Santa Maria – UFSM – Grupo de Micro Eletrônica - GMICRO
iacana.weber@ecomp.ufsm.br; carara@ufsm.br
www.github.com/iacanaw/Arke
v1.0

Este projeto foi desenvolvido pelos alunos Iaçanã Ianiski Weber e Michel Duarte, sob orientação e aconselhamento do Professor Doutor Everton Alceu Carara.

Características

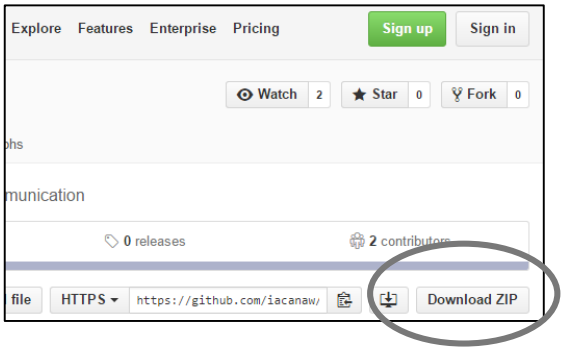
- NoC estável
- Chaveamento por *wormhole*
- Descrita em VHDL
- Código de fácil entendimento
- Suporte a topologias de malha 2D e 3D
- Dimensões parametrizáveis
- Largura de *flit* parametrizável
- Profundidade dos *buffers* parametrizável
- Independência de framework para gerar código
- Baixo tempo de roteamento
- *Crossbar* completo, pronto para aceitar outros algoritmos de roteamento

Onde encontrar

Atualmente pode ser encontrada no GitHub através da seguinte URL:

www.github.com/iacanaw/Arke

Basta clicar em "Download ZIP" e descompactar.



Conhecendo as opções de parametrização

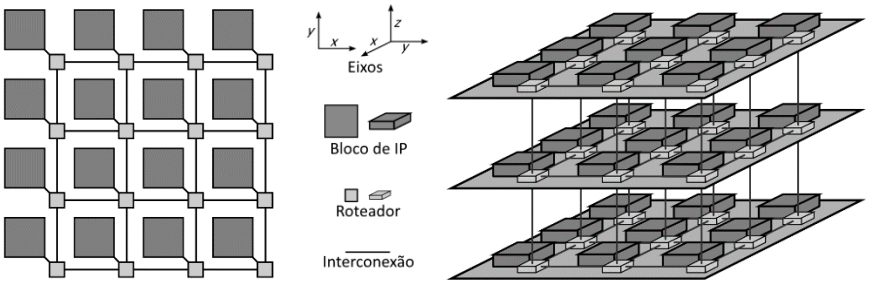
No arquivo **Arke_pkg.vhd** estão definidas todas as entidades, funções e constantes necessárias para o funcionamento da Arke. A alteração destas constantes faz com que a NoC seja sintetizada da maneira como o projetista desejar.

A Arke suporta duas topologias, malha bidimensional (malha 2D) e malha tridimensional (malha 3D). Para alternar entre estas topologias e ainda definir as dimensões da NoC as seguintes constantes devem ser alteradas:

```
-- Exemplo de configuração para criar uma NoC 3D 3x3x3
26    constant DIM_X    : integer := 3;
27    constant DIM_Y    : integer := 3;
28    constant DIM_Z    : integer := 3;
```

No trecho de código há três constantes que podem ser alteradas para determinar a topologia e as dimensões da rede. Onde `DIM_X` determina quantos roteadores devem haver no eixo X, `DIM_Y` determina quantos roteadores devem haver no eixo Y e `DIM_Z` determina quantos roteadores devem haver no eixo X. É importante ressaltar que para a criação de uma rede 2D `DIM_Z` deve ser setado para 1 (um) e não 0 (zero), pois, na realidade uma rede 2D é uma rede 3D de uma única camada.

```
-- Exemplo de configuração para criar uma NoC 2D 4x4
26    constant DIM_X    : integer := 4;
27    constant DIM_Y    : integer := 4;
28    constant DIM_Z    : integer := 1;
```



As imagens acima representam as redes que foram criadas a partir dos códigos exemplos.
Outro parâmetro que pode ser alterado de acordo com a necessidade do projetista é a profundidade do *buffer*. Esta profundidade representa quantos espaços para armazenamento temporário dos *flits* estarão disponíveis em cada porta de entrada dos roteadores. No exemplo abaixo, a profundidade do buffer foi ajustada para 16 posições.

```
31    constant BUFFER_DEPTH : integer := 16;
```

Esta configuração possui restrições duas restrições, são elas:

- A profundidade do buffer deve ser maior que 1 (um);
- A profundidade do buffer deve ser uma potência de dois (por exemplo: 2, 4, 8, 16, 32, ...).

A última etapa da parametrização da NoC é o ajuste da largura do *flit* e a adição de bits de controle. Fica a cargo do projetista definir a largura de *flit* do seu sistema, isto porque, os IPs que ele for utilizar devem respeitar esta configuração. Em relação aos *bits* de controle ajustes avançados no *Crossbar* devem ser feitos caso o projetista necessite adicionar mais algum. Existem restrições quanto ao tamanho destas duas constantes, são elas:

- `DATA_WIDTH` deve possuir o tamanho mínimo necessário para endereças todos os roteadores. Para determinar o tamanho mínimo, basta realizar a seguinte conta:

$$\text{DATA\_WIDTH} \geq \log_2 \text{DIM\_X} + \log_2 \text{DIM\_Y} + \log_2 \text{DIM\_Z}$$

- `CONTROL_WIDTH` deve ser igual ou maior que três, pois existem três sinais de controle necessários para o funcionamento correto da NoC.

```
-- Exemplo de configuração para criar uma NoC com largura de flit
-- de 16 bits e 3 sinais de controle (padrão)
34    constant DATA_WIDTH     : integer := 16;
35    constant CONTROL_WIDTH  : integer := 3;
```

Atenção ao realizar alterações no `CONTROL_WIDTH`, pois elas demandam alterações no *Crossbar*.

A interface

A interface da NoC Arke é formada pelo conjunto das portas (de saída e entrada) locais de cada um dos roteadores que fazem parte da malha e dos sinais de *clock* e *reset*.
O código VHDL que define a interface da NoC é ***seguinte:***

```
port(
    clk         : in std_logic;
    rst         : in std_logic;
    data_in     : in Array3D_data(0 to DIM_X-1, 0 to DIM_Y-1, 0 to DIM_Z-1);
    control_in  : in Array3D_control(0 to DIM_X-1, 0 to DIM_Y-1, 0 to DIM_Z-1);
    data_out    : out Array3D_data(0 to DIM_X-1, 0 to DIM_Y-1, 0 to DIM_Z-1);
    control_out : out Array3D_control(0 to DIM_X-1, 0 to DIM_Y-1, 0 to DIM_Z-1)
    );
end NoC;
```

- O sinal de clock do sistema deve ser passado para a NoC através do sinal ***clk***.
- O sinal de reset do sistema deve ser passado para a NoC através do sinal ***rst***.
- O acesso a interface Local de cada um dos roteadores que formam a malha é através dos demais sinais. Todos estes sinais possuem três indexadores, cada um referente a uma dimensão da NoC. Por exemplo, para se referir a porta de entrada de dados local do roteador 120, basta indexar o sinal da seguinte forma: `data_in(1)(2)(0)`. Este tipo de indexação permite que a Arke gere instanciamento automatizado de estruturas.

Para explicar o funcionamento da interface de comunicação entre os IPs e a porta Local dos roteadores iremos utilizar o DataManager, que se encontra dentro do diretório ".../simulation". Este componente é um IP desenvolvido para fins de experimentos com a Arke. Ele possui duas funcionalidades, são elas: (i) ler pacotes de um arquivo .txt e envia-lo pela NoC; (ii) receber pacotes da NoC e escreve-los em um arquivo de saída.

O cabeçalho do pacote é responsável por carregar o endereço do roteador destino. O formato do flit de cabeçalho segue o seguinte padrão:

0...00	X	Y	Z

Onde os campos X, Y e Z, possuem a largura mínima para acomodar os possíveis endereços das respectivas coordenadas. Caso nem todos os bits sejam necessários para fazer o endereçamento, a parte alta é preenchida com zeros. Por exemplo, uma rede com largura de flit de 16 bits e dimensões de 3x8x17 terá os seguintes campos:

```
    header(4 downto 0) <= Z; -- 5 bits (para endereçar 17 posições)
    header(7 downto 5) <= Y; -- 3 bits (para endereçar 8 posições)
    header(9 downto 8) <= X; -- 2 bits (para endereçar 3 posições)
    header(15 downto 10) <= others('0'); -- preenchido com zeros
```

Caso a rede seja bidimensional, o campo Z não é endereçado. Desta forma, o cabeçalho seguirá o seguinte padrão:

0...00	X	Y

O arquivo utilizado para fazer esta pequena demonstração será: fileIn0_0_0.txt que neste momento possui o seguinte conteúdo:

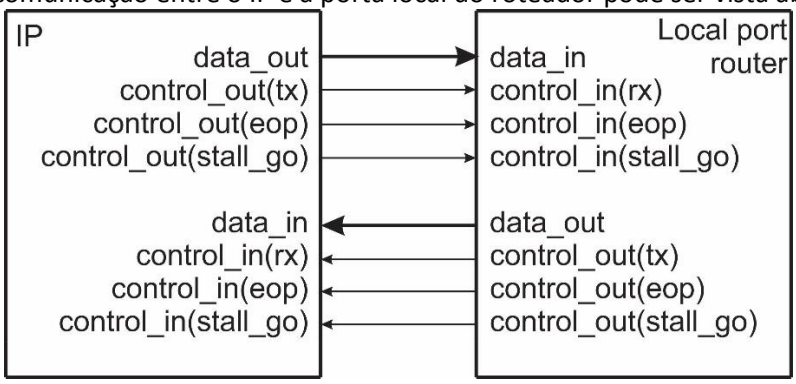
```
0002A
01111
02222
13333
```

Este é o formato no qual o arquivo deve ser escrito. Seguindo a seguinte ordem:

EOP + FLIT DE CABEÇALHO
EOP + PAYLOAD
EOP + PAYLOAD
...
EOP + PAYLOAD

Onde EOP é equivalente ao primeiro digito de cada linha, este digito informa ao DataManager o final do pacote. Na primeira linha os dígitos que acompanham o EOP são referentes ao flit de cabeçalho, onde é enviado o endereço destino a qual o pacote está destinado. Nas linhas seguintes o conteúdo que acompanha o EOP é referente ao payload, que são os dados que estão sendo enviados para outro IP. Quando o flit de payload for enviado o bit de EOP deve ser colocado em um lógico para sinalizar que o pacote chegou ao fim.

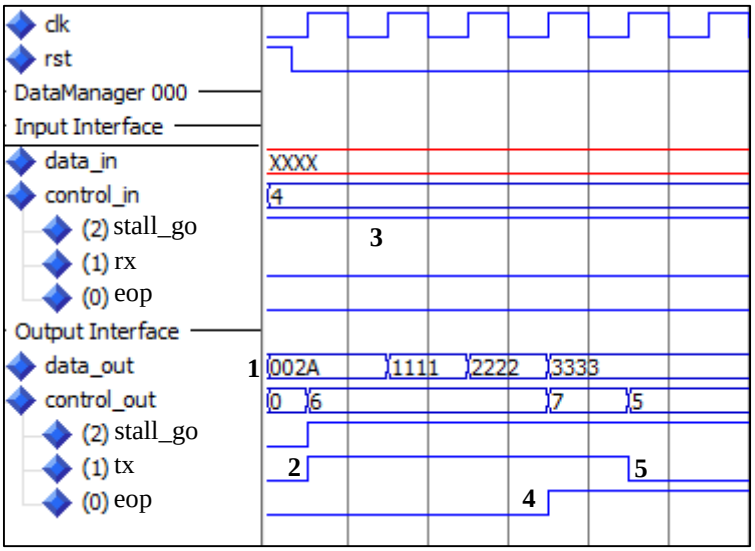
A interface de comunicação entre o IP e a porta local do roteador pode ser vista abaixo:



O protocolo de comunicação segue estas regras:

- Quando há um *flit* para ser enviado ele deve ser posto na porta de saída, `data_out`.
- Um dado na porta de saída só é válido quando o bit de controle `tx` está em valor lógico alto.
- O transmissor sabe que o dado foi transmitido com sucesso se o bit de controle `stall_go` estiver alto no momento da transmissão. Caso contrário, o *flit* deve continuar disponível e o sinal de transmissão alto até que o `stall_go` fique alto, isto confirmará a recepção.
- O bit de `eop` fica em estado lógico alto somente quando o *flit* que está sendo transmitido é o último do pacote.

A imagem abaixo retrata o envio do pacote exemplo para o roteador *Local*.



Nesta imagem, apenas a interface do DataManager é mostrada.

1) O primeiro flit (cabeçalho) fica disponível na porta de saída do DataManager;
2) O sinal `tx`, informa que o *flit* pode ser recebido pela NoC;
3) O sinal `stall_go` indica que o roteador possui espaço para armazenar um *flit.*
4) O sinal `eop` indicando que o flit 0x3333 é o último deste pacote.
5) O sinal `tx` indicando que não há mais *flits* para serem transmitidos.

Configurando uma NoC 3D

Para criar uma NoC 3D, basta abrir o Arke_pkg.vhd e redefinir algumas constantes:
É necessário informar quantos roteadores são necessários em cada coordenada. Por exemplo, uma NoC que possua 5 roteadores na direção X, 3 roteadores na direção Y e 2 roteadores na direção Z, será configurada da seguinte forma:

```
26   constant DIM_X     : integer := 5;
27   constant DIM_Y     : integer := 3;
28   constant DIM_Z     : integer := 2;
```

Agora, deve-se decidir quantas posições devem ter os *Input Buffers*, por exemplo, para definir 4 posições:

```
31   constant BUFFER_DEPTH : integer := 4;
```

Por último, a largura do *flit*, esta largura deve obedecer a equação do tamanho mínimo, então:

$$\text{DATA\_WIDTH} \geq \log_2 5 + \log_2 3 + log_2 2 \geq 4.9$$

Depois de verificado qual o tamanho mínimo, basta definir um tamanho de *flit*, igual ou maior que a resposta da equação (note que o tamanho do *flit* é dado em bits, então o número deve se SEMPRE arredondado para cima). Por exemplo, definir 8 bits.

```
34   constant DATA_WIDTH     : integer := 8;
```

Configurando uma NoC 2D

Para criar uma NoC 2D, o processo é muito semelhante ao da criação da 3D, basta abrir o Arke_pkg.vhd e redefinir algumas constantes:

É necessário informar quantos roteadores são necessários em cada coordenada. Por exemplo, uma NoC que possua 5 roteadores na direção X, 8 roteadores na direção Y, será configurada da seguinte forma:

```
26   constant DIM_X     : integer := 5;
27   constant DIM_Y     : integer := 8;
28   constant DIM_Z     : integer := 1;
```

Agora, deve-se decidir quantas posições devem ter os *Input Buffers*, por exemplo, para definir 16 posições:

```
31   constant BUFFER_DEPTH : integer := 16;
```

Por último, a largura do *flit*, esta largura deve obedecer a equação do tamanho mínimo, observe que agora a equação não leva em conta a dimensão Z uma vez que ela não é indexada em endereços 2D, então:

$$\text{DATA\_WIDTH} \geq \log_2 5 + \log_2 8 \geq 5{,}3$$

Depois de verificado qual o tamanho mínimo, basta definir um tamanho de *flit*, igual ou maior que a resposta da equação. Por exemplo, definir 8 bits.

```
34   constant DATA_WIDTH     : integer := 8;
```