UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Iaçanã Ianiski Weber

EXPLORAÇÃO DE COMUNICAÇÃO FIM-A-FIM ASSÍNCRONA ATRAVÉS DE UMA NOC SÍNCRONA

Santa Maria, RS
2019

Iaçanã Ianiski Weber

EXPLORAÇÃO DE COMUNICAÇÃO FIM-A-FIM ASSÍNCRONA ATRAVÉS DE UMA NOC SÍNCRONA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Ciência da Computação**.

Orientador: Prof. Dr. Everton Alceu Carara

Santa Maria, RS

2019

Weber, Iaçanã Ianiski

Exploração de comunicação fim-a-fim assíncrona através de uma NoC síncrona / por Iaçanã Ianiski Weber. – 2019.
80 f.: il.; 30 cm.

Orientador: Everton Alceu Carara
Dissertação (Mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Pós-Graduação em Ciência da Computação , RS, 2019.

1. Rede intra Chip. 2. Bypass de Buffer. 3. Fila Circular Assíncrona. 4. Comunicação Assíncrona. I. Carara, Everton Alceu. II.Exploração de comunicação fim-a-fim assíncrona através de uma NoC síncrona.

E-mail: iacanaw@gmail.com

Iaçanã Ianiski Weber

EXPLORAÇÃO DE COMUNICAÇÃO FIM-A-FIM ASSÍNCRONA ATRAVÉS DE UMA NOC SÍNCRONA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Ciência da Computação**.

Aprovado em 21 de fevereiro de 2019:

Everton Alceu Carara, Dr. (UFSM)
(Presidente/Orientador)

Fernando Gehm Moraes, Dr. (PUCRS) (*videoconferência*)

Mateus Beck Rutzig, Dr. (UFSM)

Santa Maria, RS

2019

AGRADECIMENTOS

Esta dissertação não poderia chegar ao bom ponto em que se encontra sem o precioso apoio de várias pessoas. Em primeiro lugar, gostaria de deixar meu profundo agradecimento ao meu orientador, Professor Everton Alceu Carara, por todo o empenho e paciência com que sempre me orientou neste e em todos os outros trabalhos ao decorrer da nossa jornada de graduação e mestrado. Muito obrigado por ter me corrigido, e olha que não foi pouco, quando necessário sem nunca me desmotivar.

Desejo igualmente agradecer a todos os meus amigos, especialmente a Luana Palma, Michel Duarte, Michael Jordan, Mateus Milano, Juliana Brondani, Rafael Faccenda e Karin Mïbö cujo apoio e amizade estiveram sempre presentes em todos os momentos. Agradeço também a todos os funcionários e professores do Centro de Tecnologia da UFSM bem como todos os colegas do Grupo de Microeletrônica, que muito bem me acolheram em 2011 e continuaram sempre dispostos a ajudar até os dias atuais. Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro prestado no decorrer do desenvolvimento deste trabalho.

Por último e mais importante, quero agradecer à minha família, em especial ao meu pai, Jandir Melquior Weber, e minha mãe, Olga Ianiski Weber, sem os quais toda essa jornada não teria sido possível.

“Penso que não cegamos, penso que estamos cegos, Cegos que veem, Cegos que, vendo, não veem.”

(JOSÉ SARAMAGO)

RESUMO

EXPLORAÇÃO DE COMUNICAÇÃO FIM-A-FIM ASSÍNCRONA ATRAVÉS DE UMA NOC SÍNCRONA

AUTOR: IAÇANÃ IANISKI WEBER
ORIENTADOR: EVERTON ALCEU CARARA

Atualmente Sistemas intra-Chip (*System on-Chip* - SoC) com um grande número de núcleos vêm adotando Redes intra-Chip (*Network on-Chip* - NoC) como infraestrutura de comunicação devido sua alta capacidade de escalabilidade. Nestes SoCs com dezenas de núcleos a dificuldade de realizar a distribuição de um *clock skew-free* por toda a dimensão do chip é elevada em tecnologias de fabricação atuais. Além disso, o consumo de energia dos transistores não se manteve proporcional ao aumento da densidade de integração, decorrente de aperfeiçoamentos na tecnologia de integração, e por consequência, hoje não é possível manter todos os núcleos em funcionamento simultâneo e ainda manter-se dentro dos limites de consumo de energia, a esse fenômeno foi dado o nome de *Dark Silicon*. Portanto, projetistas de hardware vêm adotando um paradigma de desenvolvimento conhecido como globalmente assíncrono, localmente síncrono (GALS). Cada núcleo possui seu próprio domínio de *clock* e para que ocorram comunicações entre núcleos são necessárias sincronizações de forma a evitar metaestabilidade dos dados. A NoC possui seu próprio domínio de *clock*, no qual as comunicações são estabelecidas de forma síncrona. Com o intuito de reduzir o consumo de energia foi implementado a técnica de realizar *bypass* sobre os *buffers* da NoC, desta forma, enquanto uma comunicação ocorre os *buffers* podem ser desligados, pois não realizam armazenamentos temporários, foi dado o nome de comunicação assíncrona para o protocolo que utiliza o *bypass*. A comunicação assíncrona conecta diretamente o roteador transmissor até o receptor, desta forma os dados inseridos na entrada do roteador pelo núcleo transmissor, são imediatamente transmitidos, na velocidade de propagação da via, até o núcleo receptor, à essa característica damos o nome de comunicação fim-a-fim. Os núcleos atendidos pela NoC possuem domínios de *clock* próprios, o que faz com que seja necessário realizar sincronização entre os núcleos e a NoC. Uma técnica amplamente utilizada é a de fila bissíncrona. Porém, o desempenho da comunicação assíncrona combinado com a fila bissíncrona não foi satisfatório, criando um *overhead* de latência impeditivo. Desta forma, optou-se por utiliza uma técnica de sincronização de borda, juntamente com o desenvolvimento de uma fila circular assíncrona. Quando empregados juntos, é possível alcançar reduções do consumo de energia (até 52%) sob um custo de latência (16% até 30%) e área (21%) quando comparada a uma NoC referência.

Palavras-chave: Rede intra Chip. Bypass de Buffer. Fila Circular Assíncrona. Comunicação Assíncrona.

ABSTRACT

EXPLORING ASYNCHRONOUS END-TO-END COMMUNICATION THROUGH A SYNCHRONOUS NOC

AUTHOR: IAÇANÃ IANISKI WEBER
ADVISOR: EVERTON ALCEU CARARA

Systems on-Chip (SoC) with a large number of cores adopt Networks on-chip (NoC) as the communication infrastructure due to its scalability. The complexity to distribute a skew-free synchronous clock signal over the entire chip increases in current fabrication technologies due to the process variability. The transistors energy consumption hasn't remained proportional to the increase in integration density, breaking the Dennard's scaling, as a consequence, today it is not possible to keep every core in full operation whitout breaking the limits of energy consumption, this phenomenon is called as Dark Silicon. Thus, designers may choose among asynchronous and Globally Asynchronous, Locally Synchronous (GALS) NoCs. This work proposes an intermediate solution. Each Intellectual Property (IP) core may have its clock domain, and the NoC supports both synchronous and asynchronous communication. The asynchronous communication is implemented in the NoC using a technique called bypass over internal buffers. During runtime each router in the path between the transmitter and the receiver has its internal buffers bypassed, creating a direct connection between each IP and allowing them to communicate without the NoC clock domain interference, this is called end-to-end communication. The asynchronous communication reduces the switching activity inside the NoC because router buffers are bypassed. The communication between IPs and NoC requires some synchronization technique that must be applied to contain the metastability in data transmission between clock domains. However the most traditional technique to make the synchronization between NoC and IP is a bisynchronous FIFO which proved to be unsatisfactory due to high latency penalty when associated to the asynchronous communication protocol. To work around this problem the bisynchronous FIFO has been changed by the border synchronization, which makes individual sinchronizations when a control signal is crossing to another clock domain. This technique associated with an asynchronous circular FIFO proved satisfactory in terms of energy reduction (up to 52%) under latency (16% to 30%) and area (21%) overhead.

Keywords: Network on Chip. Buffer Bypass. Asynchronous Circular FIFO. Asynchronous Communication.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABC	Asynchronous Bypass Channels
CPU	Central Processing Unit
DVFS	Dynamic Voltage and Frequency Scaling
EG	Equivalent Gate
EVC	Express Virtual Channels
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
GALS	Globally Asynchronous, Locally Synchronous
GMICRO	Grupo de Microeletrônica
IP	Intellectual Property
NI	Network Interface
NoC	Network-on-Chip
NRZ	Non Return to Zero
NOCHI	NoC with Hybrid Interconnect
ReNoC	Reconfigurable NoC
SMART	Single cycle Multihop Asynchronous Repeated Traversal
SoC	System-on-Chip
SSR	SMART-hop Setup Request
UFSM	Universidade Federal de Santa Maria
VC	Virtual Channel
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VIP	Virtual Point-to-Point

SUMÁRIO

1 INTRODUÇÃO

Gordon Moore, em abril de 1965, publicou um artigo onde ele previu que a cada dezoito meses a quantidade de transistores em um chip duplicaria (Moore, 1965). Essa previsão, conhecida como Lei de Moore, foi a força motriz por trás do desenvolvimento da microeletrônica nas últimas décadas, porém, o crescimento de um recurso físico nunca pode durar para sempre (Moore, 2003).

Circuitos integrados são compostos basicamente de dois tipos de elementos. Os construídos a partir de transistores, mais conhecidos e discutidos, são os elementos lógicos e de armazenamento. O outro tipo de elemento é conhecido como via. As vias são responsáveis pelas interconexões dos elementos construídos a partir de transistores. Elas são basicamente fios de metal que transportam a informação digital por todo o do chip.

Por décadas, os transistores dominaram o custo associado aos circuitos integrados. Não apenas o custo monetário, mas também o atraso, o consumo de energia e a área do circuito. Devido ao desenvolvimento dos processos de fabricação, os transistores se tornaram mais rápidos, energeticamente mais eficientes e menores.

Enquanto as diferenças entre o atrasos das vias em relação ao atraso das portas lógicas era desprezível, a tendência da Lei de Moore se manteve estável. Porém, isto começou a mudar quando as propriedades físicas das vias de roteamento (e.g impedâncias complexas das vias de escala nanométrica) começaram a se manifestar de forma significativa sobre os elementos lógicos. Ao longo do tempo, os recursos de roteamento tornaram-se progressivamente os principais responsáveis pelo custo, novamente, sendo o custo no sentido que englobe, atrasos, dissipação de energia e área ocupada. Neste contexto, independente do quão rápido um transistor pode chavear, se a via que carrega o sinal determina o atraso mínimo, então, a frequência do *clock* não pode ultrapassar esta barreira.

A Figura 1 mostra a evolução de diversos aspectos de unidades centrais de processamento (*Central Processing Unit* - CPU), ao longo de 42 anos. Neste gráfico pode-se observar que a frequência deixou de aumentar desde meados de 2005. Isto revela que um limite assintótico da velocidade do *clock* parece ter sido alcançado, este limite está intimamente relacionado à velocidade com que os sinais podem ser transmitidos dentro do *chip*. Além disso, é cada vez mais difícil, controlar efeitos de *jitter* (porção de uma unidade de intervalo em que o estado lógico de um sinal é indeterminado (Texas, 2000)) e *skew* (intervalo de tempo entre a recepção

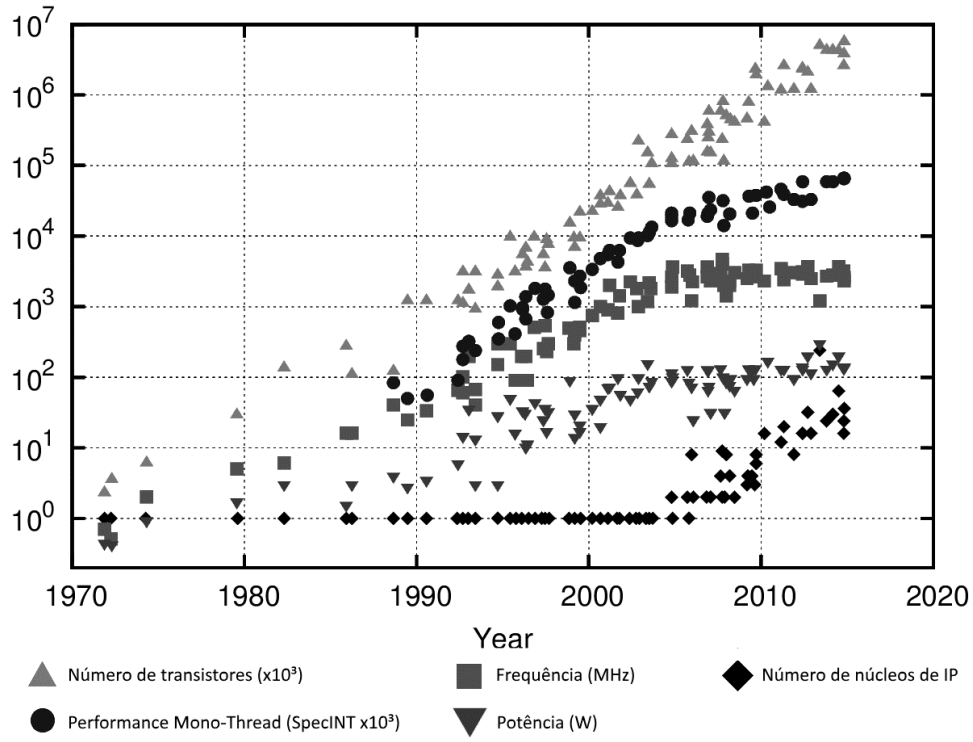


Figura 1 – Evolução dos microprocessadores ao longo de 42 anos (Rupp, 2018).

de um mesmo sinal por dois elementos distintos (Texas, 2000)) no sinal do *clock*.

Devido a essa estagnação da frequência do *clock*, processadores de propósito geral com uma única CPU parecem ter alcançado seu limite tecnológico. Porém, o número de transistores dentro de uma mesma área continua aumentando. Isto levou com que, em meados de 2004 as companhias líderes do mercado de computadores de propósito geral e sistemas embarcados confirmassem a nova tendência, chips com múltiplos processadores (Abdallah, 2013). Criar múltiplas instanciações de processadores dentro do mesmo *chip*, mostrou-se uma alternativa para manter o aumento da capacidade computacional. Outro ponto que podemos observar no gráfico da Figura 1, é o aumento do número de núcleos nos processadores. Núcleos começaram a ser adicionados aos microprocessadores no mesmo momento em que a frequência começa a estagnar.

Seguindo nessa direção, os projetistas aderiram uma nova metodologia de projeto, baseada na utilização de núcleos de hardware, denominados núcleos de propriedade intelectual ou simplesmente propriedade intelectual (*Intellectual Property* - IP) (Gupta and Zorian, 1997). Um projeto que contenha diversos IPs em um único chip, interconectados através de uma infraestrutura de comunicação, é conhecido como Sistema intra-Chip (*System-on-Chip* – SoC). Este paradigma deu início ao incessante crescimento da complexidade dos chips, permitindo

que dentro de um mesmo *die* estejam, núcleos de processamento, núcleos gráficos, núcleos de comunicação, núcleos de lógica programável, entre outros.

Porém, os IPs sozinhos não constituem um SoC, é necessária uma estrutura de interconexão capaz de realizar a comunicação e interface entre os IPs (Martin and 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 (ponto-a-ponto) 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 e apenas uma comunicação pode ocorrer no mesmo instante de tempo. Isto acarreta aumento na latência do sistema, devido a contenção dos demais IPs enquanto ocorre uma comunicação. (Kumar et al., 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 and Micheli, 2002). A Figura 2 ilustra um exemplo de IPs interconectados por uma NoC com topologia malha bidimensional. Dentre algumas características da NoC, podemos citar: suporte a comunicações simultâneas, confiabilidade, escalabilidade, reusabilidade e decisão de roteamento distribuída (Benini and Micheli, 2002) (Guerrier and Greiner, 2000). As NoCs possuem um alto grau de confiabilidade quando comparadas aos 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, facilitando a reutilização em projetos com diferentes quantidades de IPs.

Com o crescente número de IPs instanciados em um mesmo chip, e o tamanho dos transistores diminuindo, é difícil manter o *clock* sincronizado ao longo do *die* e, ainda sim, manter o desempenho acompanhando a Lei de Moore. Uma das mais promissoras soluções para o problema da distribuição do *clock* é o paradigma chamado Globalmente Assíncrono, Localmente Síncrono (*Globally Asynchronous, Locally Syncrhonous* – GALS). Este paradigma propõe que cada instância de IP deve ser considerada como uma "ilha síncrona", possuindo seu próprio domínio de *clock*. Ou seja, os IPs são localmente síncronos, porém, ao observar uma imagem global do sistema, cada IP estará operando em uma frequência distinta, isto é, o sistema é globalmente assíncrono (Krstic et al., 2007).

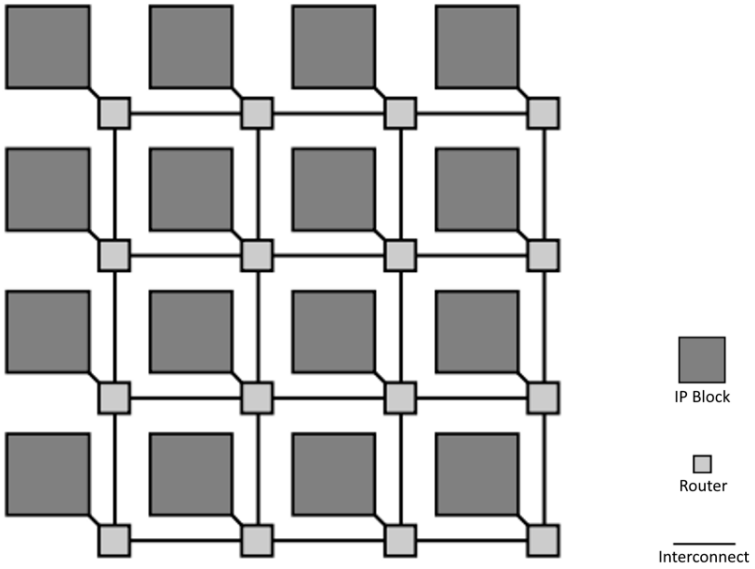


Figura 2 – Representação de uma NoC de topologia de malha bidimensional (4 X 4) (Feero and Pande, 2007).

A proposta do paradigma GALS remete aos anos 80 (Chapiro, 1984). Porém, o interesse dos projetistas aumentou apenas no final dos anos 1990 e começo dos anos 2000, com as primeiras propostas práticas (Yun and Donohue, 1996)(Muttersbach et al., 2000). Desde então, propostas de GALS SoCs têm abordado diversos aspectos de aprimoramentos como, redução no consumo de energia, aumento da velocidade de operação, aumento da taxa de transferência (Kessels et al., 2003) (Krstic et al., 2006). Em sistemas GALS, o maior desafio é a criação de uma interface de *handshake* que seja confiável em relação aos problemas de metaestabilidade, que podem ocorrer durante a transmissão de dados entre distintos domínios de *clock*.

Existem diversos métodos para solucionar o problema de transferência de dados de forma segura e confiável em um sistema GALS. Dentre eles, três estratégias se destacam, (i) *pausible-clock generators*, onde é aplicado um *clock* (*pausible, stretchable* ou *data-driven*) que torna o circuito imune a metaestabilidade, uma vez que é garantido que não haverão pulsos de *clock* enquanto o dado é transferido, (ii) *first in, first out* (FIFO) *buffers*, são utilizados entre distintos domínios de *clock*, onde ocorre a sincronização (e.g. *buffer* FIFO bisíncronos) e (ii) sincronização de borda, que realiza sincronização de sinais que estão atravessando uma determinada ilha síncrona, sem que seu funcionamento seja interrompido (Krstic et al., 2007).

1.1 OBJETIVOS

Esta dissertação tem por objetivo explorar o paradigma GALS associado à utilização de redes intra-chip em SoCs. Esse objetivo é almejado visando a possibilidade de futuramente explorar técnicas de DVFS na NoC, com o intuito de reduzir o consumo da infraestrutura de comunicação e minimizar o impacto no desempenho devido à redução da frequência/tensão. Nossa proposta é a criação de uma NoC síncrona capaz de conectar dois IPs diretamente um com outro, como se estivessem conectados por vias dedicadas, cada um com seu próprio domínio de *clock*, provendo uma interface de comunicação baseada em um protocolo assíncrono (*handshake*). Uma vez que há um caminho reservado para o pacote ser transmitido, os *flits* atravessam, via canal de *bypass*, todos os *buffers* dos roteadores que compõe o caminho entre os IPs fonte e destino. Esta capacidade foi alcançada através da capacidade de realizar *bypass* dos *buffers*, transformando um roteador de *packet switching* em um roteador híbrido *packet/circuit switching*. Para que este objetivo seja alcançado, os seguintes objetivos específicos devem ser cumpridos:

- Desenvolvimento da NoC Arke, uma rede de malha 2D/3D, parametrizável, descrita em VHDL sintetizável e que suporta tanto a comunicação utilizando o tradicional *packet switching*, quanto a possibilidade de fazer *bypass* dos *buffers*, gerando um *circuit switching*;

- Desenvolvimento da Interface de Rede (*Network Interface* – NI), responsável pelo gerenciamento da conexão. Esta NI é equipada com uma fila circular assíncrona que visa maximizar o *throughput* de dados.

- Avaliação das implementações, onde serão comparados a área, potência e desempenho de ambos os protocolos suportados pela Arke, o síncrono e o assíncrono. Sendo que o síncrono requer o NI com FIFO bisíncrono e o assíncrono o NI com a fila circular assíncrona.

1.2 CONTRIBUIÇÕES

Como principais contribuições, esta dissertação apresenta a NoC Arke e uma NI com fila circular assíncrona, ambas descritas em VHDL sintetizável e de código aberto. Quando combinadas em um SoC GALS, este conjunto de NoC e NI, provê comunicação de alta vazão

para IPs que estejam operando em uma frequência superior a da NoC. Além disso apresentam vantagens em relação ao consumo de energia ao utilizarem comunicação fim-a-fim baseada em *hanshake*.

2 TRABALHOS RELACIONADOS

Este capítulo reúne trabalhos que ajudam a descrever o estado da arte de NoCs que fazem uso de técnicas para *bypass* dos *buffers* internos dos roteadores, conectando a entrada do roteador diretamente com a saída. Desta forma, evitando o armazenamento temporário dos *flits* em cada roteador que compõe o caminho da comunicação, este tipo de técnica é conhecida como *bypass* do *buffer*.

2.1 PROPOSTA SMART

Em (Chen et al., 2013, 2014), os autores apresentam a NoC SMART (*Single Cycle Multihop Asynchronous Repeated Traversal*) é uma rede capaz de enviar dados através de vários roteadores em um único ciclo, evitando o armazenamento temporário nos *buffers* dos roteadores que estão no caminho e eliminando o atraso que estes armazenamentos gerariam para a transmissão. Os autores propõem a utilização de um circuito repetidor assíncrono que permite a propagação dos dados através de um longo caminho combinacional em um ciclo. A Figura 3 apresenta a microarquitetura da NoC SMART.

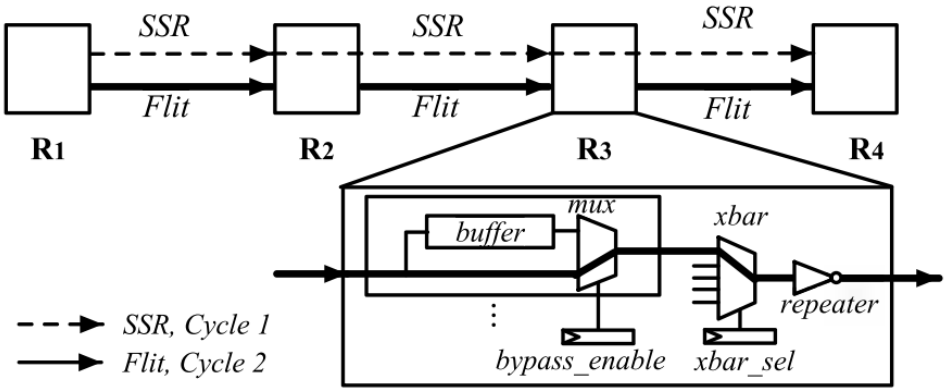


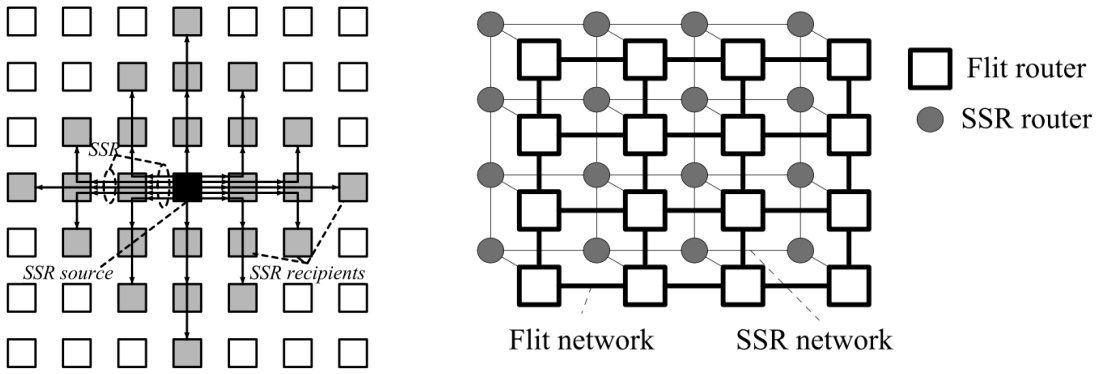
Figura 3 – Representação da SMART NoC, com ênfase na arquitetura do roteador (Chen and Jha, 2016).

Para que os dados possam atravessar sem armazenamento os *buffers* dos roteadores é adicionado um canal de *bypass*. O caminho entre origem e destino é pré-estabelecido antes da aplicação iniciar a execução. O programador deve realizar a pré-configuração nos ciclos iniciais da aplicação, através de comandos de acesso à memória. Uma vez que o caminho esteja alocado, os dados podem ser transmitidos em um ciclo. Caso ocorram colisões entre rotas SMART, é necessário fazer o armazenamento temporário de pacotes nos *buffers* dos roteadores

intermediários. Isto obriga os roteadores a possuírem *buffers* capazes de armazenar um pacote inteiro. Além disso, os repetidores são projetados para que seja possível transmitir os dados dentro do período de *clock* até uma determinada distância. Para transmissões que excedam a distância máxima estabelecida no projeto do repetidor, também é necessário o armazenamento em roteadores intermediários.

Há dois principais custos associados à SMART, o primeiro é a necessidade de sinais de controle, denominadas *SMART-hop Setup Request* (SSR). Os SSRs configuram os roteadores intermediários, para que formem um caminho de *bypass* pelo qual os *flits* serão transmitidos, um por ciclo. Quanto maior for a quantidade máxima de roteadores transpassados por ciclo, maior será o tamanho da rede SSR. Além disso, mais de uma requisição de *bypass* pode chegar no mesmo instante a um mesmo roteador. Isto faz com que seja necessário um arbitro responsável por alocar estas requisições, e, novamente, com o aumento das dimensões da NoC, maior será a complexidade deste árbitro.

Em (Chen and Jha, 2016), os autores fazem uma análise da malha SSR e propõem uma solução para a sobrecarga de sinais de controle. A proposta se baseia na utilização de uma rede auxiliar para a distribuição do SSRs aos roteadores.



(a) Rede de *flits* e barramentos de distribuição de SSR.

(b) Rede de *flits* e rede auxiliar de distribuição de SSR.

Figura 4 – Comparação da distribuição do SSR via barramentos dedicados e rede auxiliar (Chen and Jha, 2016).

Na Figura 4(a) pode-se observar uma NoC SMART de malha bidimensional, onde estão destacadas os sinais SSR que partem do roteador central. Cada roteador possui o sinal SSR para todos os roteadores que estão dentro do raio que contempla o número máximo de saltos por ciclo. A Figura 4(b) representa a proposta de uma rede auxiliar para a transmissão dos SSR. Com esta proposta os autores conseguem reduzir em até 12,2 vezes a sobrecarga de sinais de

controle em determinadas configurações. Isto se traduz em uma redução de até 63,7% em área e de 15,7% da potência dinâmica.

2.2 PROPOSTA *BYPASS ROUTER*

Em (Noghondar and Reshadi, 2015), os autores apresentam uma proposta para solucionar o problema dos sinais de *broadcast* de SSRs utilizadas pela NoC SMART. Como descrito na Seção 2.1, estes sinais são utilizadas para configurar os caminhos por onde o *bypass* será realizado. O *Bypass Router* apresenta um método alternativo para configurar os caminhos de *bypass*, de forma a reduzir o *overhead* de consumo de energia e área quanto comparados aos roteadores SMART.

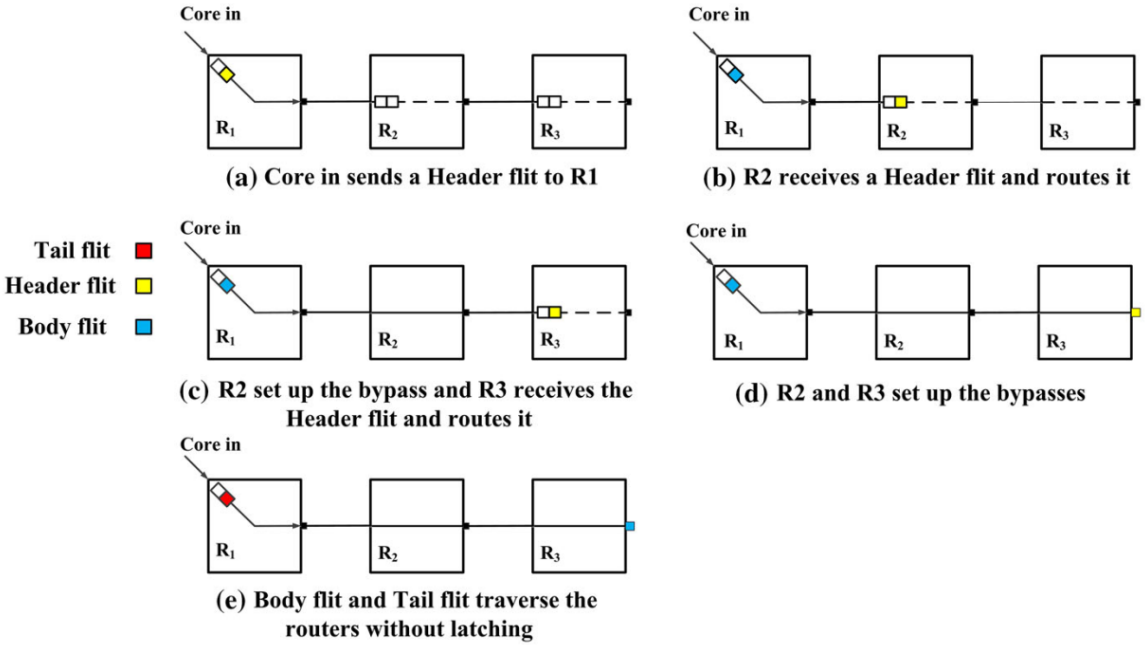


Figura 5 – Funcionamento do *bypass* Router (Noghondar and Reshadi, 2015).

A proposta apresenta a utilização do flit de cabeçalho para estabelecer o canal de *bypass*, ao invés dos sinais de SSR, desta forma, o flit de cabeçalho estabelece o *bypass* nos roteadores intermediários e os *payload flits* atravessam sem a necessidade de armazenamento temporário dentro de cada roteador que compõe o caminho. A Figura 5 mostra um exemplo de um pacote sendo transmitido utilizando o sistema proposto. Na Figura 5(a) o *Core in* envia o flit de cabeçalho para o roteador R_1 que realiza o roteamento e o envia para o roteador R_2. Na Figura 5(b) o roteador R_2 recebe e roteia o flit de cabeçalho, além de configurar o *bypass*. Na Figura 5(c) o roteador R_3, assim como no R_2, roteia o flit de cabeçalho e configura o *bypass*, e uma vez que o

caminho está estabelecido os *payload flits* atravessam os roteadores R_2 e R_3 sem a necessidade de serem armazenados temporariamente, como é visto nas Figuras 5(d) e 5(e).

A proposta apresentada eliminou o *overhead* de área criado pela da malha SSR da SMART. Todavia, os repetidores ainda oferecem a limitação de que os dados precisam atravessar o caminho de *bypass* dentro de um ciclo de *clock*. Desta forma, permanece o problema associado aos saltos com distâncias superiores às suportadas pelo repetidor, exigindo o armazenamento temporário dos dados em roteadores intermediários durante transmissões com distâncias máximas superiores às suportadas pelos repetidores.

2.3 PROPOSTA RENOC

Em (Stensgaard and Sparsø, 2008), os autores apresentam uma NoC com topologia reconfigurável chamada de ReNoC (*Reconfigurable* NoC) com o objetivo de adequar a topologia de acordo com a aplicação a ser executada. Cada roteador da NoC é envolto por um hardware chamado de *topology switch*, o qual é implementado utilizando circuitos de chaveamento semelhantes ao de FPGAs. A Figura 6 mostra uma NoC onde cada nodo da rede consiste de um roteador envolto por um *topology switch*.

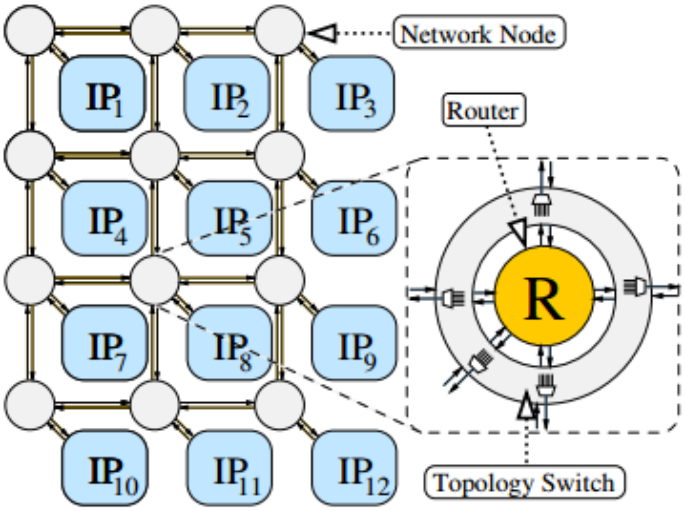


Figura 6 – Uma NoC de malha 2D onde cada nodo da rede consiste de um roteador envolto por um *Topology Switch* (Stensgaard and Sparsø, 2008).

Os *topology switches* são utilizados para criar diferentes topologias lógicas sobre uma mesma topologia física através de canais de *bypass* sobre os roteadores, como mostra a Figura 7. Desta forma, após a fase de inicialização, a NoC fica configurada de forma a minimizar a comutação de pacotes. Uma vez que o processo de reconfiguração é atrelado exclusivamente ao

topology switch, pode-se utilizar arquiteturas diversas nos roteadores. Sempre que os roteadores ficam ociosos (os que sofrem *bypass*) depois de uma reconfiguração, são utilizadas técnicas de *clock gating* para reduzir a potência dinâmica e de *power gating* para reduzir a potência de *leakage*.

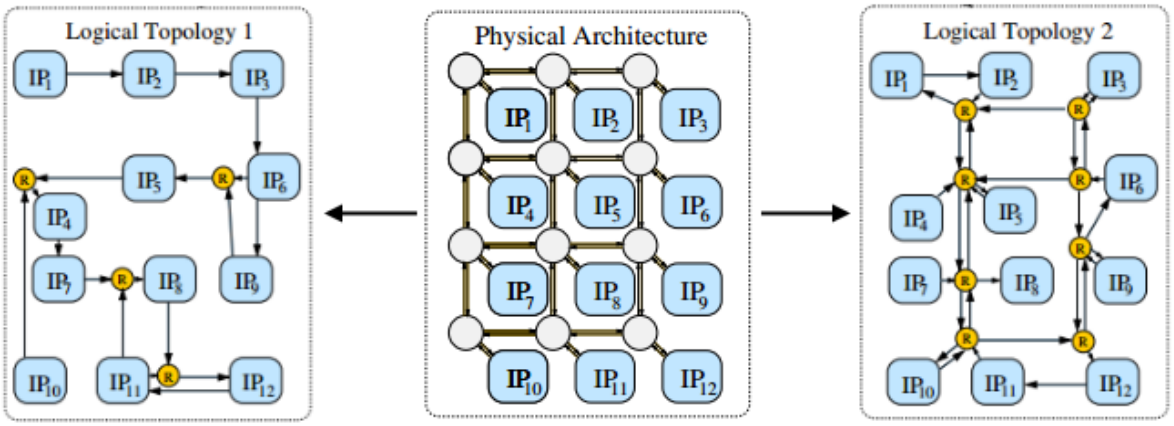


Figura 7 – Duas possíveis configurações da arquitetura física através da configuração apropriada dos *Topology Switches* (Stensgaard and Sparsø, 2008).

A ReNoC apresenta até 56% de redução em relação ao consumo, quando comparada a uma NoC malha 2D estática, ao custo de um aumento de área de 10%. Porém, é necessário um conhecimento prévio da aplicação para que o programador configure a topologia lógica desejada antes do início da execução.

Em (Jackson and Hollis, 2010), os autores apresentam uma arquitetura baseada na ReNoC, chamada *Skip-links*. Nesta proposta os autores automatizam o processo de configuração da topologia, o qual ocorre de forma dinâmica durante a execução das aplicações. Cada nodo da rede analisa o fluxo local e determina se deve ser estabelecido um *skip-link* entre um determinado par entrada-saída de maneira que o roteador sobra *bypass*. Quando vários nodos adjacentes formam *skip-links* em sequência, é formado o que é chamado de *skip-chain*. O conjunto de *skip-chains* tende a otimizar a topologia para a aplicação durante o seu tempo de execução. Desta forma, o principal inconveniente da ReNoC é solucionado e a topologia passa a ser transparente para a aplicação. Quando comparada a uma NoC de malha 2D estática a redução na energia consumida e no número de roteadores atravessados pelos pacotes (*hop count*) é em torno de 10%.

2.4 PROPOSTA *ASYNCHRONOUS BYPASS CHANNELS*

Em (Jain et al., 2010), os autores tomam como ponto de partida um SoC projetado com o paradigma GALS, onde cada par IP-roteador opera em sem próprio domínio de *clock*. Quando um sinal precisa atravessar diferentes domínios de *clock*, é necessário realizar sincronização. Com a finalidade de eliminar o tempo de sincronização entre roteadores vizinhos, os autores propõem a adição de canais assíncronos chamados de *Asynchronous Bypass Channels* (ABCs) dentro dos roteadores.

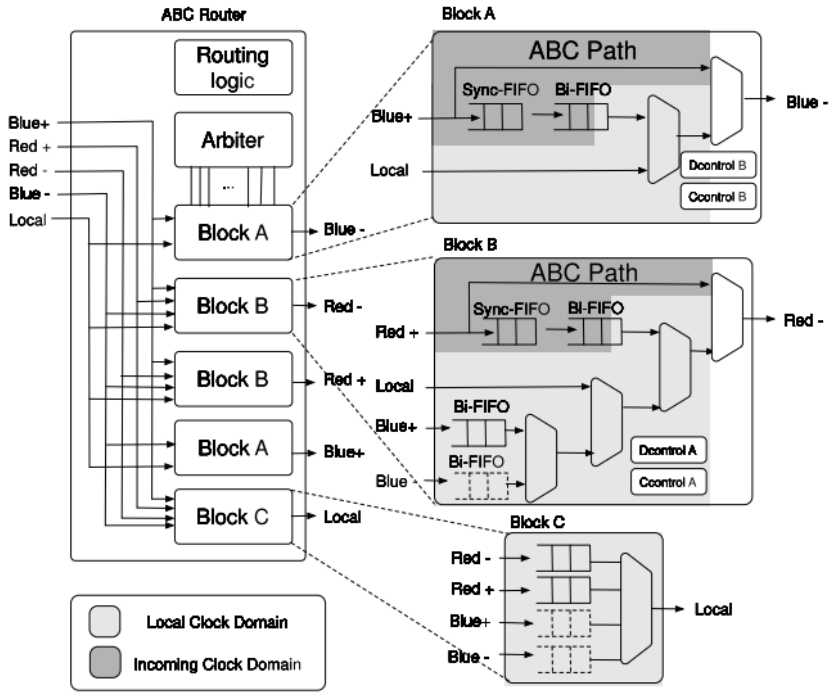


Figura 8 – Microarquitetura do roteador ABC (Jain et al., 2010).

A Figura 8 apresenta a microarquitetura do roteador, destacando a aplicação dos ABCs. Como pode ser observado, os dados que entram no roteador podem utilizar os ABCs para realizar *bypass* nos *buffers* de sincronização. Uma vez que o roteamento é estabelecido, o roteador mantém o chaveamento e os dados fluem diretamente sem nenhum tipo de sincronização entre roteadores vizinhos. Ampliando a perspectiva, vê-se formando um caminho de roteadores entre o transmissor e o receptor, o qual pode ser considerado como um barramento. Desta forma, os IPs realizam a comunicação como se estivessem conectados de forma direta, obedecendo seus próprios protocolos de comunicação e nos próprios domínios de *clock*.

Como resultado, quando comparada a um design síncrono, o design proposto melhora em até 26% o desempenho para baixas cargas e em 50% a vazão. Existem algumas limitações

na implementação, por exemplo, não é possível utilizar o ABC *Path* quando o pacote faz uma mudança de direção (e.g. leste para norte) dentro da rede, o que se traduz em um aumento na latência. Além disso, o tamanho máximo do pacote deve ser de quatro *flits* menor que o tamanho dos *buffers* dos roteadores, a fim de evitar perdas de dados durante congestionamentos.

2.5 PROPOSTA DE NOC COM CONEXÃO HÍBRIDA

No artigo (Krishna et al., 2008), os autores propõem a utilização de uma NoC hibrida de duas camadas, uma para realizar a transmissão dos dados e uma segunda para prover a rede de informações sobre o trafego, com o objetivo de aumentar a eficiência do controle de fluxo. A NoC proposta possui um controle de fluxo do tipo *Express Virtual Channels* (EVCs). A ideia principal por trás do EVC é prover vias virtuais na rede, de forma que os *flits* possam realizar *bypass* sobre os roteadores intermediários. O *bypass* evita a o armazenamento temporário dos *flits*, que são transmitidos adiante assim que chegam em um roteador intermediário. Isto acarreta em redução de latência, potência dinâmica (pois o acesso ao *buffer* é evitado), e potência de *leakage* (pois o número de *buffers* necessários para manter a mesma banda de transmissão é menor). Os autores propõem a utilização das *Global Interconnect Lines* (G-lines), responsáveis por realizar a difusão dos sinais de controle através de todo o sistema. Algumas diferenciações são necessárias para adaptar o esquema EVC, para que seja compatível com as G-lines. A junção do controle de fluxo EVC com as G-lines, proposta pelos autores, recebe o nome de *Network-on-Chip With Hybrid Interconnect* (NOCHI).

Na Figura 9 podemos observar como se dá a distribuição das G-lines na NOCHI. Devido a adição das G-lines, o sistema fica apto a difundir um determinado sinal através de todo o sistema. Neste exemplo, 14 vias de um bit são instanciadas, para cada direção (i.e. N-S, S-N, L-O e O-L). As G-lines são divididas em dois conjuntos, aquelas que transmitem a disponibilidade de *buffer* e as que transmitem informação sobre canais virtuais livres. É através da leitura destes sinais que os roteadores transmissores sabem se o roteador destino está apto a receber o pacote. Desta forma, desde que as condições estejam favoráveis, um pacote pode ser transmitido, fazendo *bypass*, por toda a amplitude da rede em uma direção (no caso da Figura 9, é possível transmitir por até 6 roteadores antes de precisar ser armazenado).

Os autores propuseram um cenário onde comparam o desempenho da NOCHI-EVC com com uma NoC EVC convencional. Neste cenário a NOCHI-EVC foi capaz de reduzir a latência em até 44% próximo do ponto de saturação da EVC, e em 9,4% para uma rede vazia.

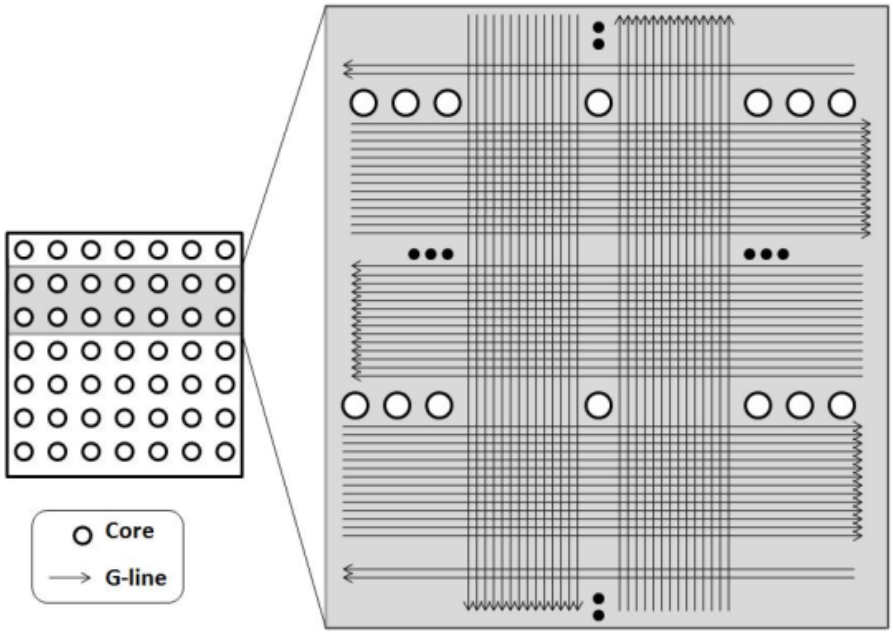


Figura 9 – NoC 7x7 com 14G-lines por direção por linha/coluna (Krishna et al., 2008).

Isto porque a NOCHI-EVC possibilita o *bypass* para distâncias maiores, que proporcionam em média 53,7% de *bypass* em comparação aos 41,3% atingidos pela NoC EVC convencional. Em relação a consumo, Tushar Krishna, que também é colaborador da NoC SMART, et al., afirmam que podem alcançar uma redução de 54.6mW por na potência consumida por cada roteador, o que se traduz em 29,2% quando comparado a NoC EVC convencional.

2.6 PROPOSTA *VIRTUAL POINT-TO-POINT LINKS*

Neste trabalho, (Modarressi et al., 2008), os autores apresentam um método para criação de uma conexão *virtual point-to-point* (VIP) entre roteadores de uma NoC com chaveamento por pacotes. Como pode ser visto na Figura 10, um canal virtual de cada canal físico do roteador é designado à realizar as conexões VIP. Neste canal virtual o *buffer* é substituído por um *latch* (1-*flit buffer*) que é conectado a saída do roteador através de uma conexão reconfigurável. A saída do roteador consiste na multiplexação de *flits* advindos de duas fontes, a primeira, que seleciona as conexões VIP, é o resultado da multiplexação de qual das entradas deve ser direcionada a determinada porta de saída. A segunda fonte, é a saída do *crossbar*, que trata dos *flits* que trafegam utilizando o tradicional chaveamento por pacote. Através da configuração prévia dos multiplexadores, o *flit* salvo no *latch* é transmitido para a saída sem que seja necessário passar pelos estágios de escrita e leitura no *buffer*, roteamento, arbitragem e traves-

sia de *crossbar*. Desta forma, uma comunicação que utilize as conexões VIP é estabelecida através da conexão dos *latches*, criando um caminho entre os roteadores fonte e destino. Para isso, os multiplexadores nos roteador fonte e intermediários são configurados de maneira que os *latches* em cada roteador se conecte à saída, criando um caminho direto ao longo da NoC. A arquitetura proposta, em média, reduz em 45% o consumo de energia quando comparada a uma *packet-switching* NoC, porém, há um *overhead* de área de 25% devido a adição de *hardware* para execução do *bypass*.

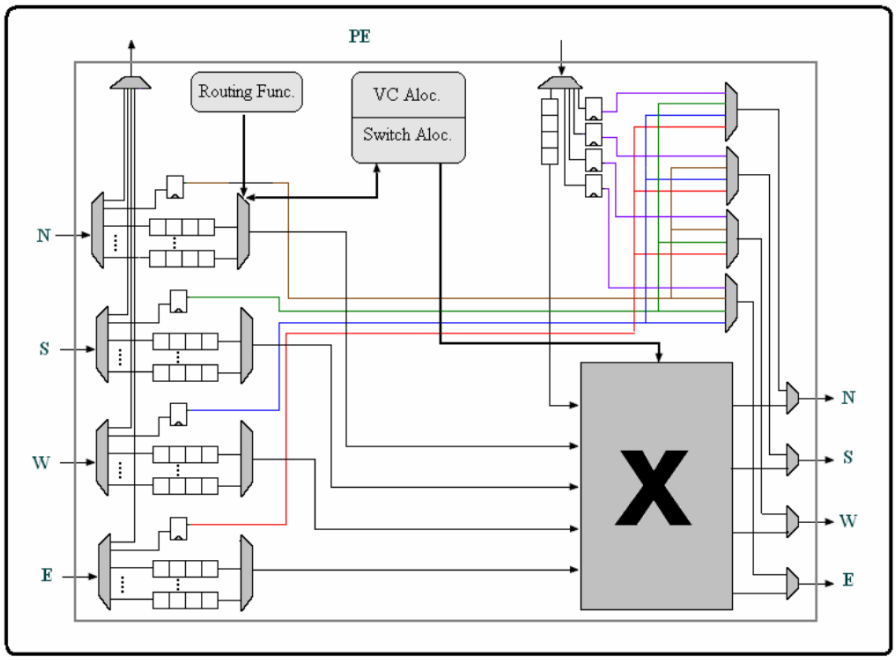


Figura 10 – Arquitetura proposta para o roteador VIP (Modarressi et al., 2008).

2.7 CONSIDERAÇÕES SOBRE A REVISÃO BIBLIOGRÁFICA

Os trabalhos apresentados nesta seção reúnem aplicações de técnicas de *bypass* sobre os *buffers* dos roteadores. Tal técnica será base fundamental para o desenvolvimento do modo de comunicação assíncrono que foi implementado na NoC Arke. A proposta SMART, apresentada na Seção 2.1, introduz os canais de *bypass* tal como serão implementados na Arke, todavia o funcionamento da NoC continua sendo síncrono, fato que obrigou os autores a desenvolver um repetidor capaz de entregar o sinal respeitando o perídio do *clock*. Como solução para o problema da entrega do sinal, respeitando o período de *clock*, surgiu a ideia de trabalhar com um protocolo assíncrono, desta forma, não precisando respeitar o período de *clock* da NoC durante as transmissões. Ademais, a NoC SMART realiza a configuração dos caminhos de *bypass* antes

da execução de uma tarefa, desta forma, é necessário o conhecimento prévio do comportamento das comunicações além de uma malha de configuração (SSR). Estes problemas foram levados em consideração na Seção 2.2, onde é apresentada a proposta *Bypass Router*. O modo de comunicação assíncrono que foi desenvolvido baseou-se no mesmo principio apresentado, possuindo uma etapa de estabelecimento onde ocorre a configuração do caminho de *bypass*, em tempo de execução. Assim como a NoC Arke, a proposta *Asynchronous Bypass Channels*, apresentada na Seção 2.4, tem por objetivo atuar em um sistema desenvolvido em um paradigma de integração GALS. Todavia, a NoC empregada nesse sistema possui cada roteador dentro do domínio de *clock* do seu IP local. O problema que se destaca ao tratar a NoC dessa forma, são as sucessivas sincronizações necessárias quando não é possivel estabelecer um caminho de comunicação assíncrono. A NoC Arke, apresentada neste trabalho, por outro lado, possui um domínio de *clock* único para todos seus roteadores, ou seja, a NoC é tratada como um IP responsável pela comunicação. Caso seja necessário realizar uma transmissão síncrona tradicional, não ocorre o problema de sincronizações sucessivas entre roteadores. Por fim, as propostas de NoC com conexão híbrida e *virtual point-to-point links*, apresentadas nas Seções 2.5 e 2.6 respectivamente, apresentam mais exemplos de pesquisas realizadas com a técnica de *bypass* dos roteadores, demonstrando o potencial da técnica em obter redução de consumo e latência devido ao não armazenamento dos *flits* em roteadores intermediários, sob pena de aumento na área do circuito. O capítulo seguinte irá apresentar a NoC Arke, assim como seus modos de comunicação síncrono e assíncrono, detalhando aspectos arquiteturais de sua implementação.

3 A REDE INTRA-CHIP ARKE

Nossa proposta foi intitulada de Arke, tal nome é dado em referência à mensageira dos Titãs, da mitologia grega, pois, assim como uma mensageira, a NoC tem por objetivo entregar as mensagens que são enviadas através dela. A NoC Arke é descrita em VHDL sintetizável e parametrizável. Suporta topologia de malha bidimensional ou tridimensional. Possui dois modos de comunicação, (*i*) síncrono e (*ii*) assíncrono. Independente de *framework* para gerar instâncias, sendo necessário apenas a alteração de parâmetros no arquivo de configuração para que seja possível mudar as dimensões da malha, tamanho de *buffers* e largura dos *flits*.

A Arke implementa chaveamento por pacotes do tipo *wormhole*. Este método exige que a mensagem a ser transmitida seja dividida em pequenas partes denominadas de *flits* (*flow control units*). Junto do conjunto de *flits* é adicionado um cabeçalho que contém informações quanto ao tipo de comunicação que será estabelecida e o endereço destino do pacote, ou seja, as coordenadas X, Y e Z do roteador destino, Figura 11. O cabeçalho é dividido em quatro campos (um para definição do tipo de comunicação que será estabelecido e outros três para cada uma das coordenadas) cujos tamanhos são ajustados automaticamente a partir das dimensões que foram determinadas para a NoC. Este ajuste é feito de maneira que cada coordenada ocupe o mínimo de bits necessário para representar as dimensões da NoC. Por exemplo, uma NoC tridimensional com dimensões 4x2x5 (XxYxZ) terá um campo de dois bit para a coordenada X, um bit para a coordenada Y e três bits para a coordenada Z. Enquanto as coordenadas ocupam a parte menos significativa do cabeçalho, a seleção do tipo de comunicação que será estabelecida faz uso do bit mais significativo. Para que seja estabelecida uma comunicação síncrona, o bit mais significativo deve ser '0', por outro lado, para estabelecer uma comunicação assíncrona, o bit mais significativo deve ser '1'. Em relação ao *payload* do pacote, não há restrições quanto ao seu tamanho, de modo que uma mensagem qualquer do nível de IP pode ser transmitida em um único pacote da NoC. A liberação dos canais alocados pelo pacote é feita durante a transmissão do último *flit*, o qual é indicado através de um sinal de controle específico chamado *EoP* (*End-of-Package*).

A NoC Arke é construída a partir da múltipla instanciação de um bloco básico, o roteador, ilustrado na Figura 12. O número de portas varia 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* (*Input Buffer*) de en-

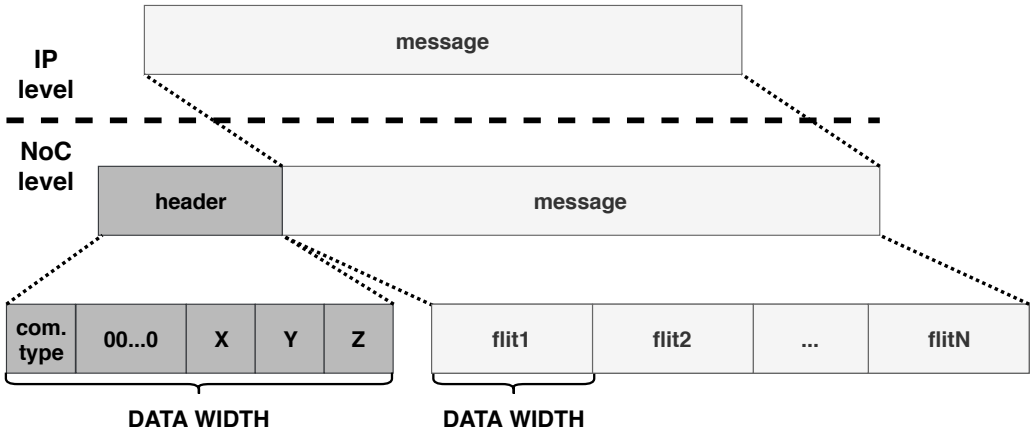


Figura 11 – Formato do pacote da NoC Arke.

trada 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 de roteamento centralizada (*Switch Control*) que implementa o algoritmo de roteamento, faz a arbitragem das requisições e configura o *Crossbar* que realiza a conexão entre as portas de entrada e saídas. A instanciação dos roteadores é feita de forma automática através do código VHDL. Com base em parâmetros dimensionais definidos no arquivo de configuração 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

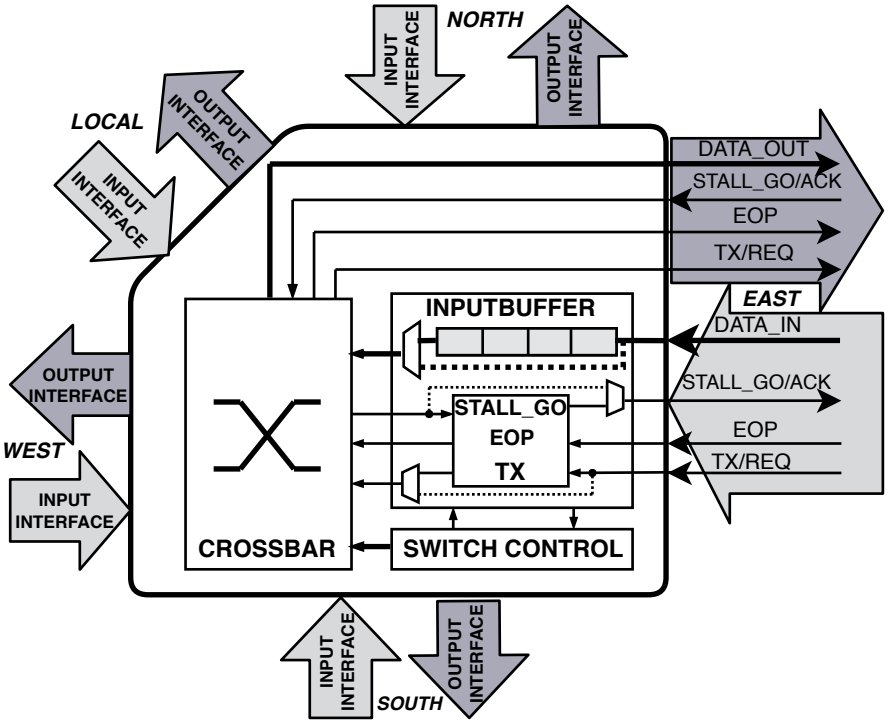


Figura 12 – Organização simplificada do Roteador Arke (Weber et al., 2018).

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.

A arbitragem implementada no *Switch Control* determina qual pacote deve ser roteado quando mais de um cabeçalho chega no roteador em um mesmo instante de tempo. Para garantir que nenhum pacote sofra de *starvation*, o árbitro é dinâmico rotativo e implementa *Round-Robin* baseado em um codificador de prioridade programável. O árbitro atende uma solicitação de roteamento por vez. Assim 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. 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*.

3.1 O ROTEADOR ARKE

A Figura 13 servirá como um guia para o restante deste capítulo. Ela mostra detalhadamente a organização interna do roteador. Como pode ser observado, cada direção é identificada por um índice: 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 utilizando ambos os modos de comunicação, síncrono e assíncrono. 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.

3.1.1 *Input Buffer*

Este componente é a porta de entrada dos pacotes no roteador e é responsável por executar o controle de fluxo dos *flits* que trafegam entre roteadores. Possui um *buffer* parametrizável, tanto na largura quanto na profundidade. 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 para o *Input Buffer*, informando que uma porta de saída está alocada, *Crossbar* configurado e os dados podem ser transmitidos adiante.

Na Figura 14 pode ser observado a interface com o *Switch Control*, entrada e saída. Além disso, um esquema simplificado de sua organização interna, onde destaca-se os canais de *bypass*

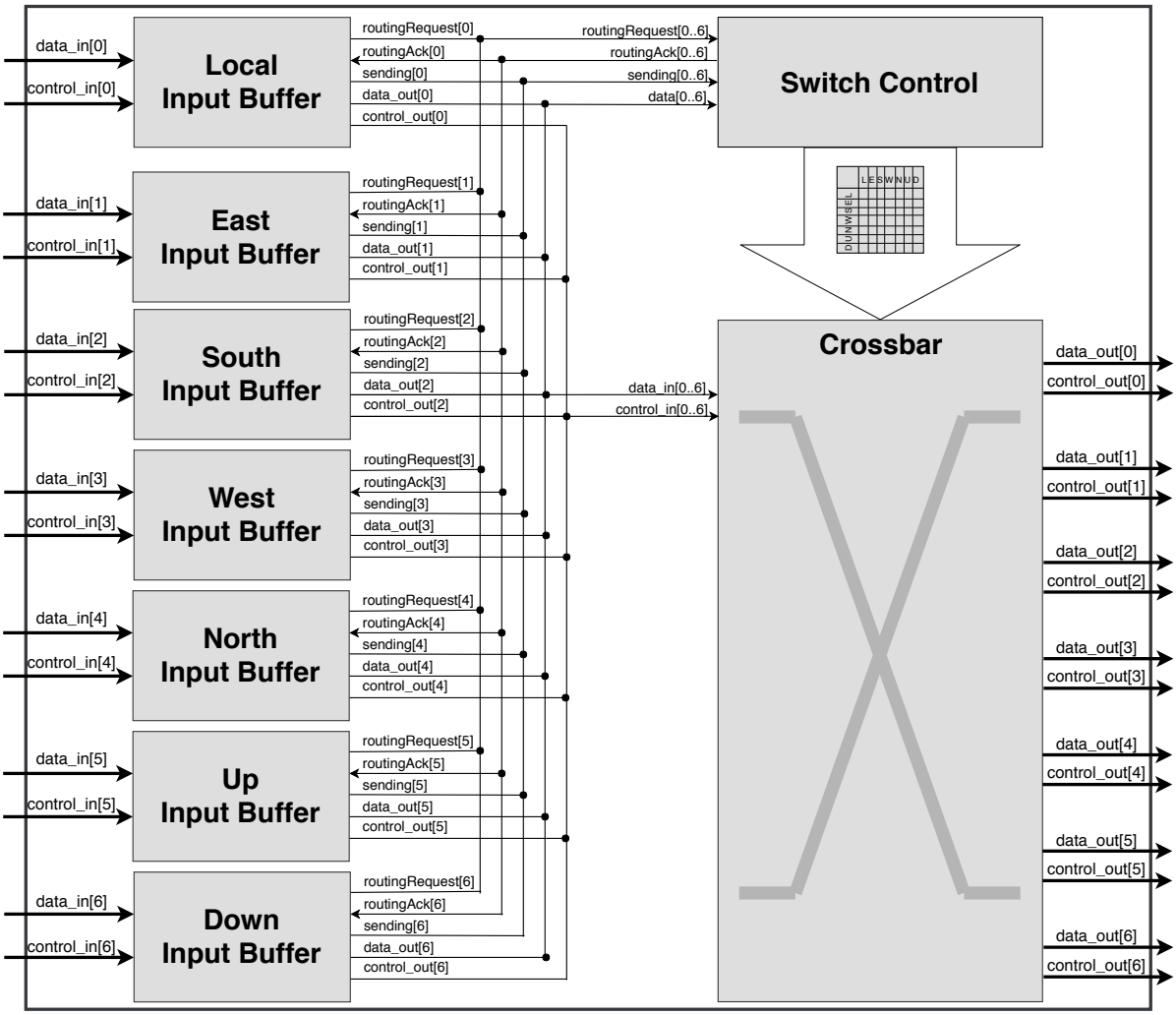


Figura 13 – Organização do Roteador Arke.

(pontilhado) por onde fluem os flits, sem armazenamento temporário, durante a comunicação fim-a-fim assíncrona. Para compreender melhor o funcionamento do *Input Buffer*, dividiremos o seu funcionamento em duas subseções que melhor detalharão o funcionamento dos dois modos de comunicação suportados pela Arke, Comunicação Síncrona (Seção 3.1.1.1) que emprega o tradicional chaveamento por pacotes e Comunicação Assíncrona (Seção 3.1.1.2) que possui três estágios, formando um híbrido de chaveamento por pacote com chaveamento de circuito.

3.1.1.1 Comunicação Síncrona

A comunicação síncrona emprega o tradicional chaveamento por pacotes, o qual implementa o método *stall and go* como controle de fluxo. A interface entre dois roteadores pode ser observada na porta leste do roteador da Figura 12. Nesta interface há duas portas (saída e entrada). E cada porta é formada por dois barramentos, o de dados (data_in/out) e o de con-

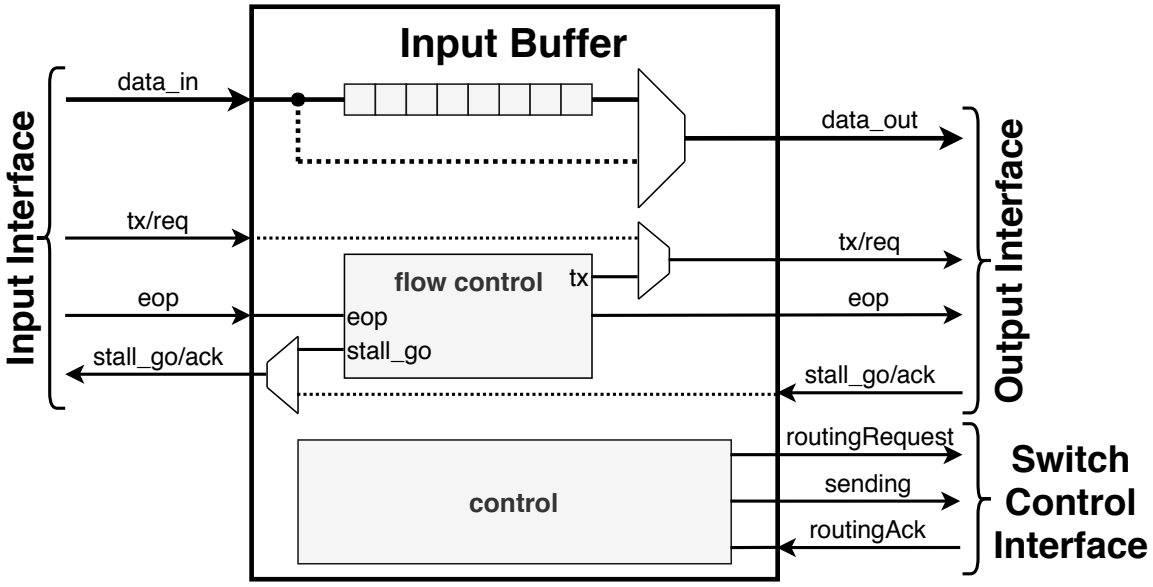


Figura 14 – Organização do *Input Buffer* da Arke.

trole (*stall_go - eop - tx*), onde cada bit tem uma função específica. Este método consiste na utilização de uma fila FIFO (*First-In First-Out*) de entrada no receptor e uma linha de retorno (*stall_go*) ao transmissor para informar se há espaço disponível nesta file. 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 Figura 15 mostra um diagrama envolvendo os sinais necessários para realizar a transmissão de um pacote utilizando o protocolo de comunicação síncrono. Inicialmente (*idle*) o sinal de controle *tx* é mantido em nível lógico zero, informando à NoC que não há *flit* válido para ser transmitido. No momento que um novo pacote fica disponível para o envio (*sendFlit*), um novo *flit* deve ser disponibilizado no barramento *data_out*, bem como o sinal *tx* é alterado para nível lógico alto, indicando que um *flit* válido está disponível. Como não se trata do último *flit* do pacote, o sinal *eop* deve ser mantido em nível lógico baixo. O transmissor deve verificar o sinal de retorno *stall_go*, se o nível estiver baixo, indicando que a fila à frente está cheia, o dado deve ser mantido, assim como os sinais *tx* e *eop* (*hold*). Uma vez que o *stall_go* indique que há espaço, um novo *flit* pode ser enviado (*sendFlit*). Quando o último *flit* for enviado, deve-se alterar o estado do sinal *eop* para nível lógico alto (*sendEop*), desta forma a conexão interna do roteador será finalizada, liberando os recursos para que uma nova conexão interna possa ser estabelecida.

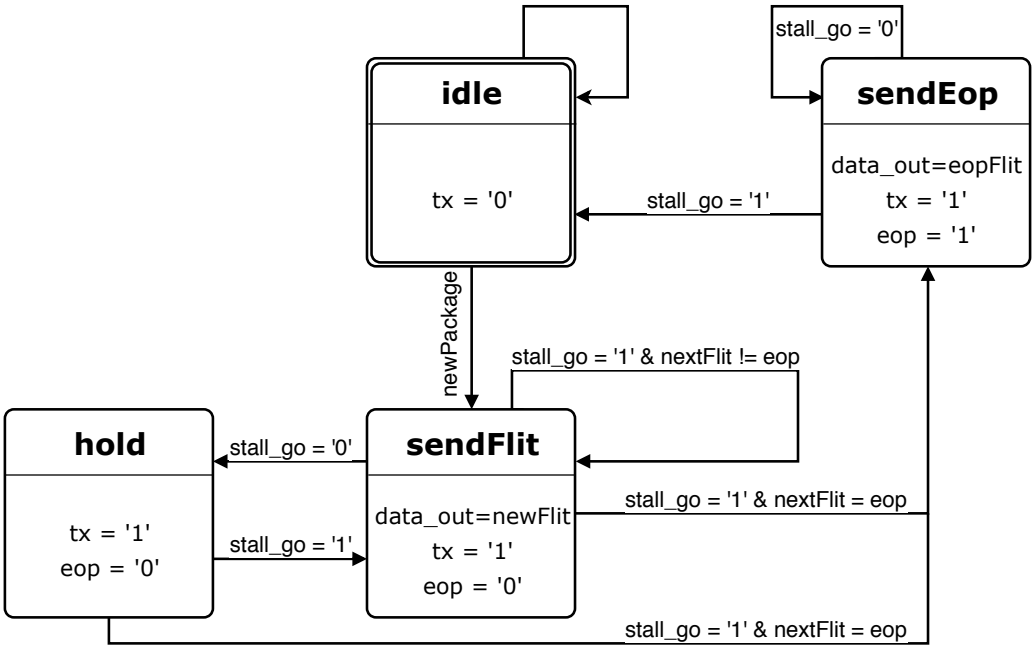


Figura 15 – Diagrama do funcionamento da Comunicação Síncrona, sob a perspectiva de um IP transmissor.

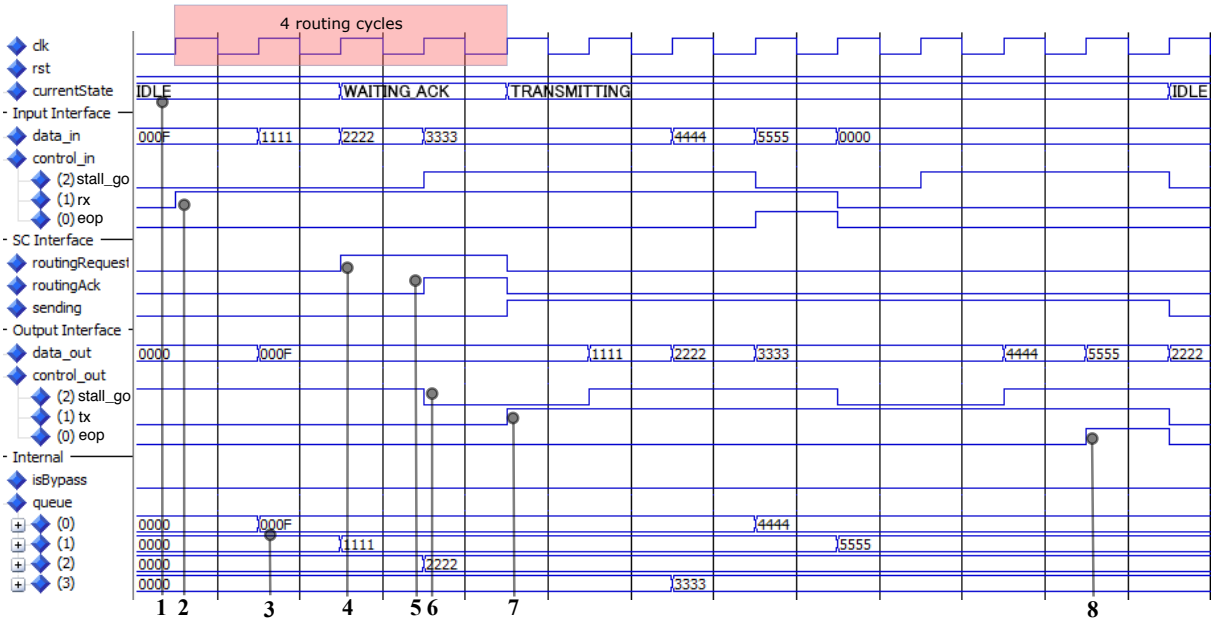


Figura 16 – Exemplo de um pacote sendo recebido e transmitido pelo *Input Buffer* utilizando o protocolo síncrono.

Na Figura 16 é possível visualizar o *Input Buffer local* recebendo um pacote com seis *flits* (0x000F, 0x1111, 0x2222, 0x3333, 0x4444 e 0x5555) e em seguida sendo transmitido ao roteador vizinho, utilizando a comunicação síncrona. Nesta figura pode-se observar o período de roteamento dura quatro ciclos, destacados na imagem, compreendidos entre a chegada do *control_in(rx)* e a sinalização do *sending* para o *Switch Control*. No primeiro ciclo ocorre o

armazenamento do *flit* de cabeçalho, no segundo a requisição de roteamento, seguido de um ciclo de espera para que o *Switch Control* realize aplique o protocolo de roteamento e configure o *Crossbar* e no último ciclo ocorre a travessia do *Crossbar*.

1. Inicialmente o *Input Buffer* começa no estado *idle* (*currentState*) e permanece aguardando até que um novo *flit*, que será um cabeçalho, seja disponibilizado.

2. O cabeçalho (0x000F) fica disponível para o roteador através do barramento 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 fila (*queue*) um ciclo após ser disponibilizado, caso haja espaço para que isso ocorra. Neste caso a fila encontra-se vazio e o *flit* é armazenado. Enquanto o sinal *stall_go* do *Input Buffer* indica que há espaço para o armazenamento de *flits*, o transmissor continua transmitindo um novo *flit* a cada ciclo.

4. No ciclo seguinte, o estado do *Input Buffer* passa para *waiting_ack*. Neste estado é feita a requisição de roteamento para o *Switch Control* (*routingRequest*). Baseado no *flit* de cabeçalho presente na saída *data_out*, o *Switch Control* irá executar o algoritmo de roteamento e determinará a porta de saída do roteador que será utilizada na transmissão do pacote.

5. Ao finalizar o roteamento, o *Switch Control* sinaliza ao *Input Buffer* através do sinal *routingAck* que o *Crossbar* está configurado e pode-se iniciar a transmissão do pacote.

6. Neste momento a fila (*queue*) ficou cheia, o sinal *stall_go = 0* indicará que não é possível armazenar um novo *flit*, desta forma o transmissor deve manter o *flit* aguardando no barramento (*data_in*) até que haja espaço na fila.

7. O sinal *tx* da interface de saída é mantido ativo enquanto existirem *flits* do pacote armazenados. O *flit* disponibilizado na saída (*data_out*) é atualizado a cada ciclo se o próximo receptor estiver com o sinal *stall_go* alto. Neste momento o estado se encontra em *transmitting* e o sinal *sending* é mantido ativo, sinalizando ao *Switch Control* que a porta de saída alocada ainda está sendo utilizada.

8. O processo de transmissão de um pacote utilizando o protocolo síncrono é encerrado quando último *flit* do pacote é transmitido. O sinal *eop* indica que o *flit* presente no

barramento de 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.

3.1.1.2 Comunicação Assíncrona

A comunicação assíncrona, tem por objetivo fornecer aos IPs uma opção de comunicação na qual o impacto do domínio de frequência da NoC seja mínimo no desempenho da comunicação. O protocolo assíncrono possui três estágios, são eles, (*i*) estabelecimento, (*ii*) transmissão e (*iii*) encerramento. Durante o estabelecimento, a NoC realiza a conexão entre o IP transmissor e o receptor. Neste estágio o cabeçalho é transmitido de forma síncrona, reservando um caminho dentro da NoC. Uma vez que o cabeçalho é transmitido adiante, o roteador configura os multiplexadores para executar o *bypass* do *buffer* de entrada e os sinais de controle de fluxo *tx* e *stall_go* (os sinais que permitem o *bypass* estão pontilhados na Figura 12). Desta forma a porta de saída fica diretamente conectada com a de entrada. Após a passagem do cabeçalho pela NoC, reservando o caminho e aplicando o *bypass* nos *buffers*, forma-se uma conexão direta fim-a-fim. Alterações nos sinais na entrada irão se propagar pela caminho reservado, como se fosse um fio, chegando na saída sem que ocorram armazenamentos intermediários. Neste momento se dá inicio a segunda fase, transmissão. Neste ponto, os IPs estão diretamente conectados e podem fazer uso desta conexão utilizando um protocolo assíncrono. Ficam disponíveis como meio de controle de comunicação os sinais de *tx* e *stall_go*. O protocolo empregado fica à escolha do projetista, neste trabalho foi utilizado o protocolo de *handshake* de duas fases *non-return-to-zero* (NRZ). Nesta seção não será abordado o funcionamento do protocolo de comunicação assíncrono empregado na etapa de transmissão, pois ele será amplamente discutido no Capítulo 4, uma vez que está intimamente relacionado ao funcionamento da *Network Interface* que foi desenvolvida. A terceira e última etapa, encerramento, é iniciada quando o IP envia realiza o envio do último *flit* do pacote. Assim que o roteador detecta o sinal nível alto do bit *eop* ele envia para o próximo roteador do caminho um sinal de *eop* e libera a conexão interna. Este processo se repete até que todos os roteadores que formavam o caminho tenham liberado suas conexões internas que formavam a conexão fim-a-fim.

A Figura 17 mostra um diagrama envolvendo os sinais necessários para realizar a transmissão de um pacote utilizando o protocolo de comunicação assíncrono. Este diagrama é apresentado sob a perspectiva de um IP transmissor. Como se trata de um protocolo assíncrono, os sinais *tx* e *stall_go* passam a ser tratados como *request* (*req*) e *acknowledgement* (*ack*), respecti-

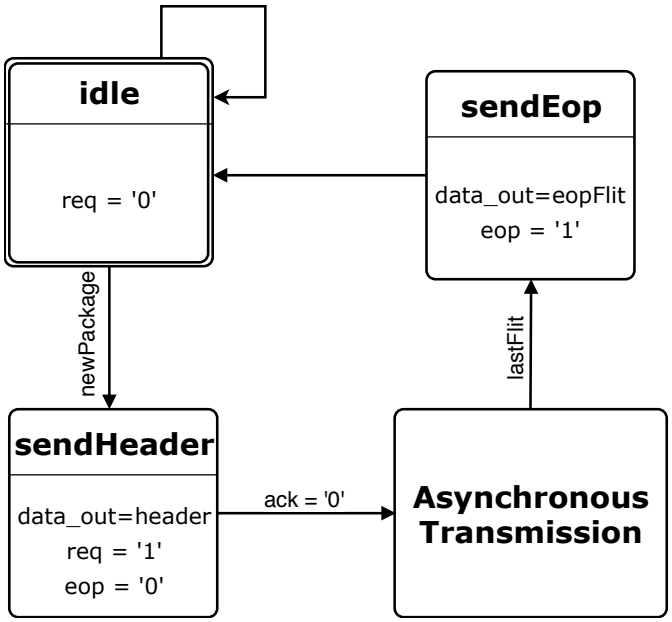


Figura 17 – Diagrama do funcionamento da Comunicação Assíncrona, sob a perspectiva de um IP transmissor.

vamente, como é de costume na comunicação assíncrona. Inicialmente (*idle*) o sinal de controle *req* é mantido em nível lógico zero, informando à NoC que não há *flit* válido para ser transmitido. No momento que um novo pacote fica disponível para o envio (*sendHeader*), o cabeçalho é disponibilizado no barramento *data_out*, bem como o sinal *req* é alterado para nível lógico alto, indicando que um *flit* válido está disponível. Como não se trata do último *flit* do pacote, o sinal *eop* deve ser mantido em nível lógico baixo. O transmissor deve aguardar uma transição no sinal de retorno, *ack*, que se manterá em nível alto até que a conexão esteja estabelecida. Uma vez que ocorra esta transição, tem-se a indicação de que o cabeçalho foi recebido pelo receptor, neste momento se dá inicio a etapa de transmissão que irá utilizará um protocolo de comunicação assíncrono, definido pelos IPs. Quando o último *flit* for enviado, deve-se alterar o estado do sinal *eop* para nível lógico alto (*sendEop*), desta forma a conexão será finalizada, desfazendo os canais de *bypass* e liberando os recursos da NoC para que uma nova conexão possa ser estabelecida.

Na Figura 18 é possível visualizar o *Input Buffer* local (ligado ao IP transmissor) recebendo o cabeçalho de um pacote (0x8001, 0x1111, 0x2222, 0x3333, 0x4444 e 0x5555) que demanda comunicação assíncrona. Neste exemplo pode-se observar as três etapas da comunicação assíncrona.

1. Inicialmente o *Input Buffer* começa no estado *idle* (*currentState*) e permanece aguardando até que um novo *flit*, que será um cabeçalho, seja disponibilizado.

2. O cabeçalho fica disponível para o roteador através do barramento de entrada (*data_in*), este dado é confirmado como válido através do sinal de controle *rx*. Devido ao fato do bit mais significativo do cabeçalho estar em nível lógico alto (0x8001), trata-se de uma requisição de comunicação assíncrona. Desta forma, tratamos esta etapa por estabelecimento do *bypass* (destacado em vermelho na Figura 18).

3. O *flit* presente em *data_in* é armazenado na fila um ciclo após ser disponibilizado, caso haja espaço para que isso ocorra. Neste caso a fila (*queue*) encontra-se vazio e o *flit* é armazenado. Por se tratar de uma comunicação assíncrona, o transmissor envia apenas o cabeçalho e aguarda até que o *bypass* esteja estabelecido em todos os roteadores do caminho para que a transmissão do payload possa iniciar.

4. No ciclo seguinte, o estado do *Input Buffer* passa para *waiting_ack*. Neste estado é feita a requisição de roteamento para o *Switch Control* (*routingRequest*). Baseado no *flit* de cabeçalho presente na saída *data_out*, o *Switch Control* irá executar o algoritmo de roteamento e determinará a porta de saída do roteador que será utilizada na transmissão do pacote.

5. Ao finalizar o roteamento, o *Switch Control* sinaliza ao *Input Buffer* através do sinal *routingAck* que o *Crossbar* está configurado, e pode-se iniciar a transmissão do cabeçalho para o próximo roteador que formará o caminho assíncrono.

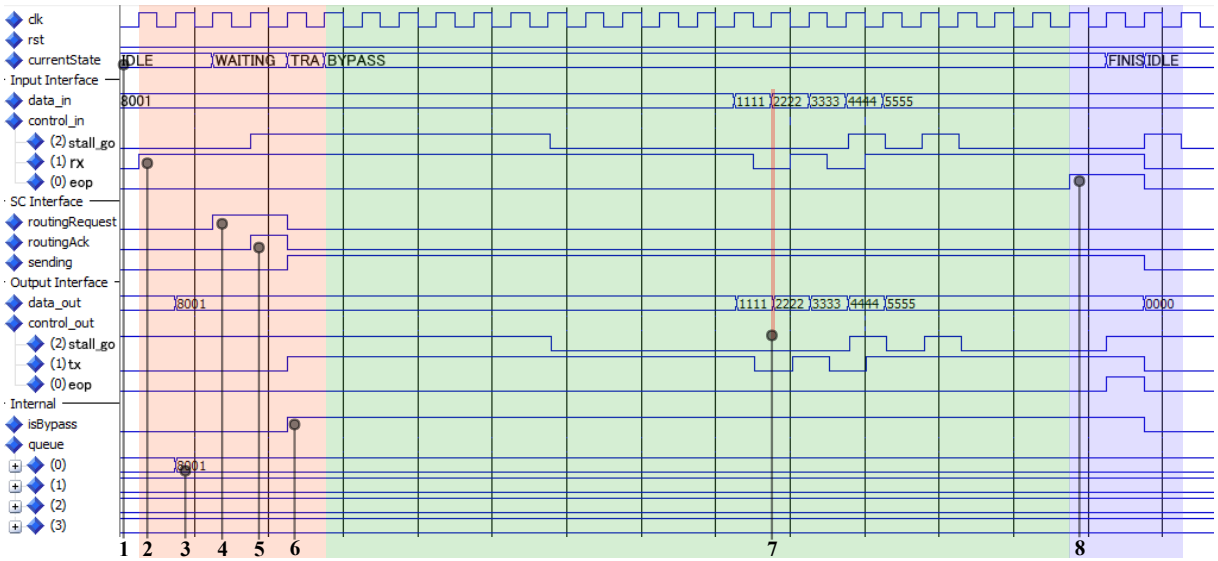


Figura 18 – Exemplo de um pacote sendo recebido e transmitido pelo *Input Buffer* utilizando o protocolo assíncrono.

6. O sinal *isBypass* é ativado, indicando que o caminho de *bypass* deve ser utilizado. No próximo ciclo, o estado é alterado para *bypass*. A partir deste momento se dá inicio a fase de *transmitting* (destacado em verde na Figura 18). Nesta fase o bypass sobre o *buffer* está estabelecido e os *flits* fluem diretamente do barramente de entrada para a saída, sem armazenamento temporário. Detalhes do funcionamento do protocolo de transmissão assíncrono utilizado nesta etapa serão vistos durante o capítulo que trata da *Network Interface*.

7. Aqui é destacado que a transmissão dos dados do barramento de entrada (*data_in*) para a saída (data_out) não é instantânea, pois há o atraso de propagação lógica intrínseco do *Input Buffer*, gerado pelos multiplexadores e pelo *Crossbar*.

8. O processo de transmissão do pacote utilizando o protocolo assíncrono é encerrado quando último *flit* do pacote é transmitido. O sinal *eop* indica que o *flit* presente no barramento de saída, *data_out*, é o último do pacote. A esta etapa, damos o nome de encerramento *bypass* (destacado em azul na Figura 18). Ao detectar o sinal *eop* o *Input Buffer* troca para o estado de *finish_bypass* onde ele envia para o próximo roteador a informação que o último *flit* está sendo transmitido. Só então o sinal *sending* é desativado, sinalizando ao *Switch Control* que a porta de saída alocada pode ser liberada.

3.1.2 *Switch Control*

Outro componente que constitui os roteadores Arke é o *Switch Control*. Ele é o responsável por arbitrar a utilização das portas de saída do roteador, aplicar o algoritmo de roteamento sobre os endereços destinos, configurar as conexões do *Crossbar* e não permitir que os pacotes sofram de *starvation*. Possui interface de conexão com cada um dos *Input Buffers* e com o *Crossbar*, como pode ser observado na Figura 19 e na Figura 13.

A arbitragem implementada no *Switch Control* é dinâmica rotativa (*Round-Robin*), implementada através de um codificador de propriedade 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.

O *Switch Control* implementa um algoritmo de roteamento determinístico. A partir da

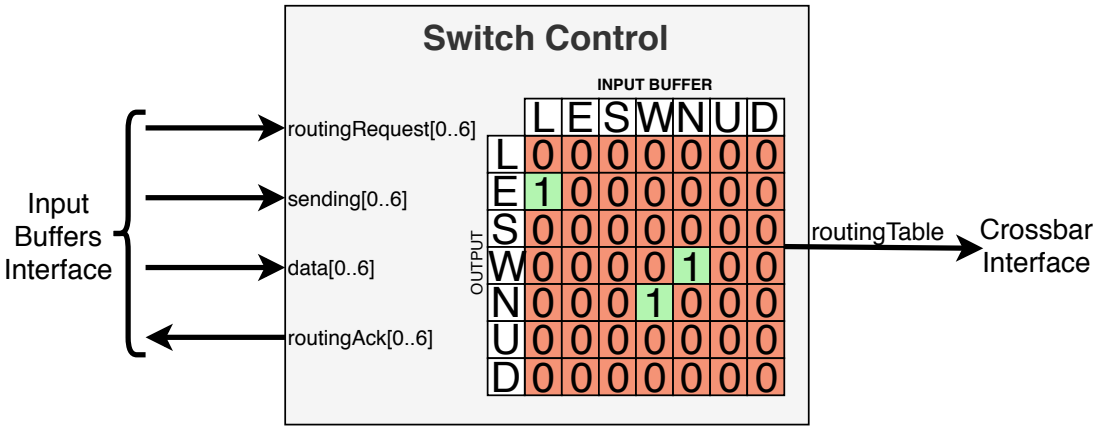


Figura 19 – Interface do *Switch Control* do Roteador Arke e estrutura da *routingTable*.

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 XY ou XYZ, dependendo da configuração da NoC, que é dado devido à maneira de como 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 20 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.

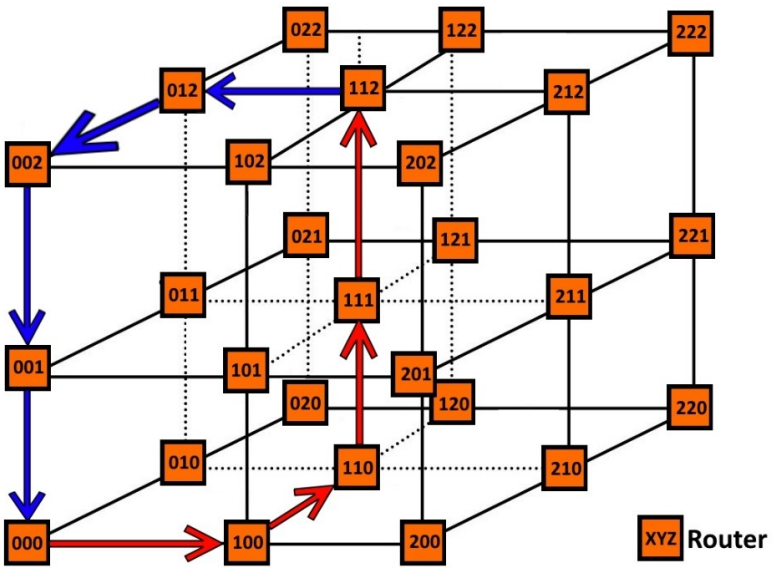


Figura 20 – Malha tridimensional 3x3x3. O caminho destacado exemplifica o funcionamento do algoritmo de roteamento XYZ.

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*. A *routingTable* (19) é estruturada da seguinte forma, cada linha da tabela corresponde a uma porta de saída e cada coluna representa um *Input Buffer*. Quando um *Input Buffer* precisa ser conectado a uma determinada porta de saída, basta setar a intersecção e o *Crossbar* irá realizar a conexão. Intersecções com valor '0' indicam que não há conexão estabelecida. Na Figura 19 o *Input Bufer* local é conectado à saída leste.

Para exemplificar o funcionamento do *Switch Control*, a Figura 21 apresenta uma simulação com uma solicitação de roteamento sendo realizada. Esta solicitação corresponde ao passo 4 da Figura 18 e Figura 16.

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*. Neste exemplo a requisição é feita pelo *Input Buffer local* (*routingRequest(0)*). Quando uma requisição chega, o codificador de prioridades determina com base na última porta atendida, 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 é a porta leste que estava livre. Então o *Switch Control* seta a *routingTable*, na intersecção da porta de saída selecionada com o *Input Buffer* solicitante (*table(1)(0) <= '1'*). Desta forma o *Crossbar* sabe quais portas devem ser conectadas.

4. O estado muda para *routing ack*. 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*.

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

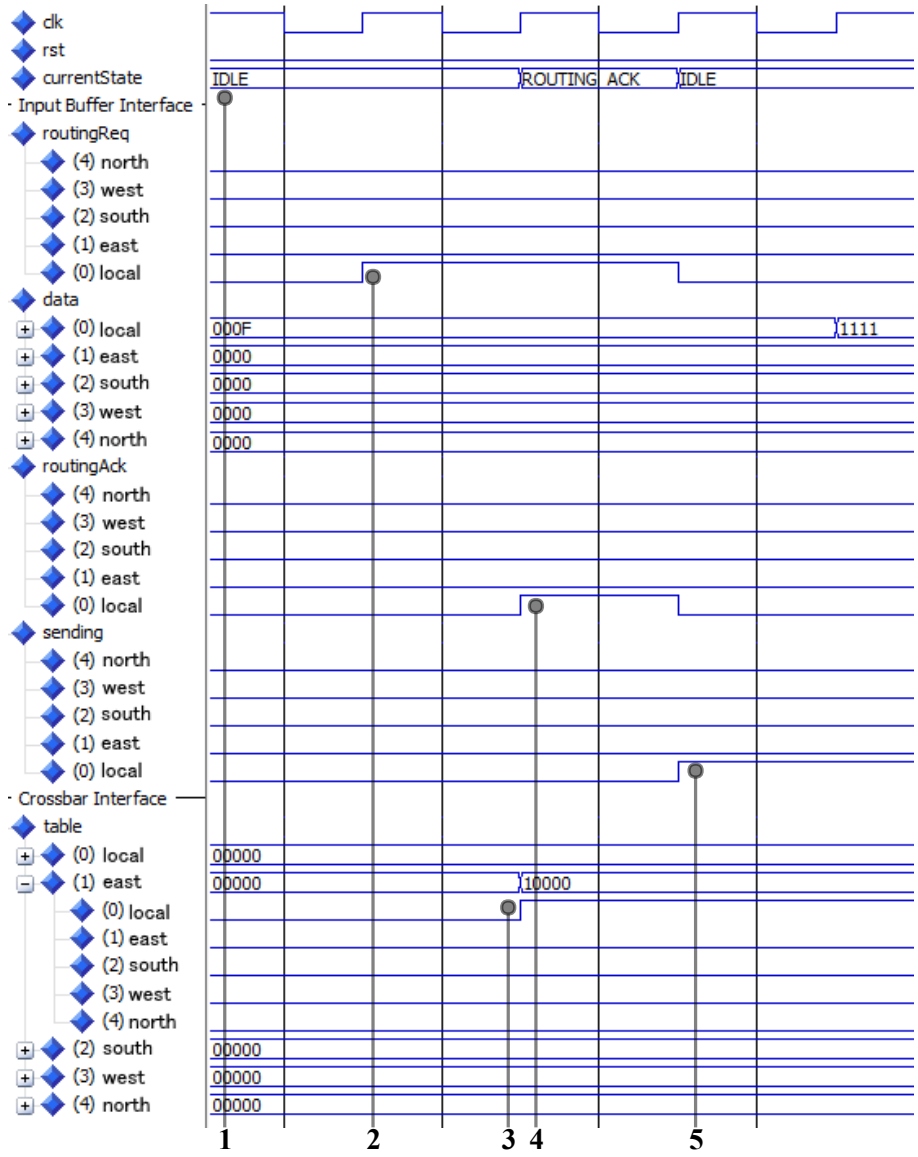


Figura 21 – Simulação do *Switch Control* atendendo a uma solicitação de roteamento.

3.1.3 *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 *routingTable*, a qual é gerada pelo *Switch Control*. Na Figura 22 é apresentada a interface do *Crossbar*.

Neste capítulo foi detalhado o funcionamento da NoC Arke, onde foi coberto o funcionamento do roteador e de cada um dos componentes que o compõe (*Input Buffer*, *Switch Control* e *Crossbar*). Além disso, foram introduzidos os dois modos de comunicação, síncrono

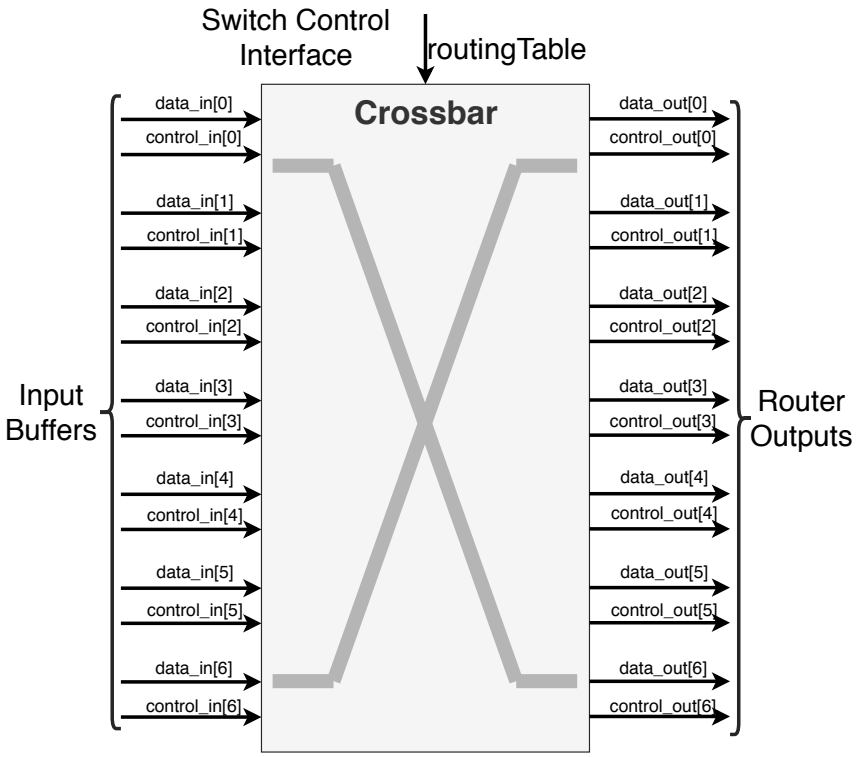


Figura 22 – Interface do *Crossbar*.

e assíncrono. Todavia, ainda há a necessidade de compreender o funcionamento das *Network Interfaces*, pois, elas são responsáveis por gerenciar as conexões que seus respectivos IPs estão utilizando. No próximo capítulo iremos explorar dois tipos de *Network Interface*, uma para cada protocolo de comunicação.

4 NETWORK INTERFACE

A Network Interface possui expressiva importância no desempenho que será extraído da comunicação intra-chip oferecido pela NoC. Uma NI bem projetada é não-obstrutiva, permitindo que o IP faça uso de toda a banda oferecida pela NoC com a menor latência possível. Infelizmente muitas NIs não são tão transparentes. Uma NI mal projetada acaba gerando um gargalo na comunicação e aumentando significativamente a latência da comunicação (Dally and Towles, 2003).

Um sistema que implementa o paradigma GALS integra em um mesmo chip diversos IPs que foram desenvolvidos individualmente utilizando ferramentas de CAD para sistemas síncronos (daí o termo "localmente síncrono"). Todavia, a operação de todos os IPs não é sincronizada e a comunicação entre os blocos de IP síncronos ocorre de forma assíncrona (daí o termo "globalmente assíncrono"). Para permitir a comunicação de um IP com a NoC deve-se transferir os dados que estão indo de um domínio de *clock* para outro. Esta transferência de domínio de *clock* ocorre no nível da NI de maneira transparente à NoC e ao IP. A literatura apresenta possíveis alternativas para realizar a sincronização entre os distintos domínios de *clock*. Três técnicas se apresentaram adequadas de serem aplicadas, são elas:

i. *Pausible-clock*: Onde se aplica um *clock* local (*pausible, stretchable*, ou *data-driven*) garantindo que não hajam pulsos de *clock* enquanto o dado está sendo transferido, dessa forma, evitando metaestabilidade dos dados (Yun and Donohue, 1996) (Muttersbach et al., 2000) (Krstic et al., 2006);

ii. Sincronização de borda: As sincronizações ocorrem individualmente nos sinais que atravessam as bordas de uma região síncrona sem que seja necessário parar o funcionamento local do bloco síncrono (Dobkin et al., 2004) (Bjerregaard et al., 2005);

iii. Fila FIFO: Faz uso de filas FIFO bissíncronas entre os blocos síncronos, escondendo o problema da sincronização (Chelcea and Nowick, 2000) (Beigne and Vivet, 2006).

De forma a atender o principio de não obstrutividade e baixa latência, foram desenvolvidas duas NIs, (*i*) NI de fila bissíncrona e (*ii*) NI de fila circular assíncrona. Sendo cada uma delas especializada na utilização de um dos modos de comunicação fornecido pela Arke. As seções seguintes apresentarão ambas as NIs, detalhando suas arquiteturas e funcionamentos.

4.1 NI DE FILA BISSÍNCRONA

Esta NI foi desenvolvida para operar com ambos os protocolos da Arke, ou seja, utilizando o controle de fluxo *stall and go* ou o protocolo de *handshake* assíncrono. O principal objetivo desta NI é realizar a adaptação dos domínios de *clock* entre o IP e a NoC, mantendo a máxima vazão que o protocolo síncrono permite, de um *flit* por ciclo de *clock* da NoC. Desta forma, foi escolhido uma fila FIFO por oferecer a melhor capacidade vazão dentre as três técnicas citadas anteriormente, além de ser uma solução simples e de ampla utilização em sistemas GALS. A fila bissíncrona utilizada nesta NI é apresentada em (Panades and Greiner, 2007). Segundo os autores, para garantir máxima vazão em um sistema onde os os *clocks* de leitura e escrita na fila são similares é necessário utilizar seis posições na fila. Veja na Figura 23 a interface e a organização simplificada do NI com fila bissíncrona.

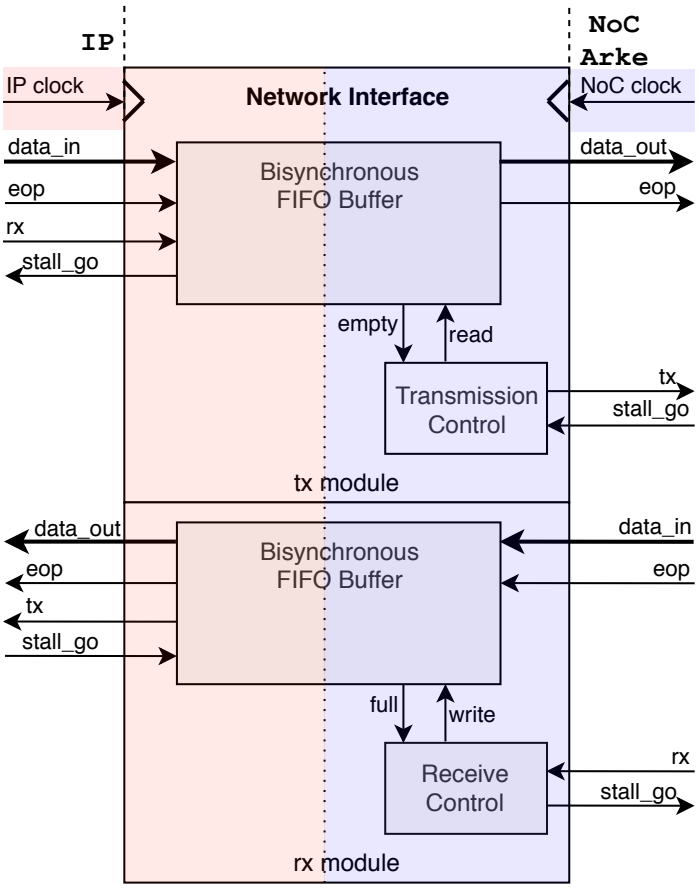


Figura 23 – Interface e organização simplificada da NI bissíncrona.

A NI de fila bissíncrona é formada por dois módulos, um de transmissão e outro de recepção. Cada um dos módulos possui uma fila bissíncrona que é responsável por realizar a sincronização dos dados que chegam nela através do domínio de *clock* do IP, no caso do módulo

de transmissão, e da NoC, no caso do módulo de recepção. Além das filas, foi necessário adicionar lógica de controle para fazer a adaptação de protocolo para suportar a comunicação assíncrona.

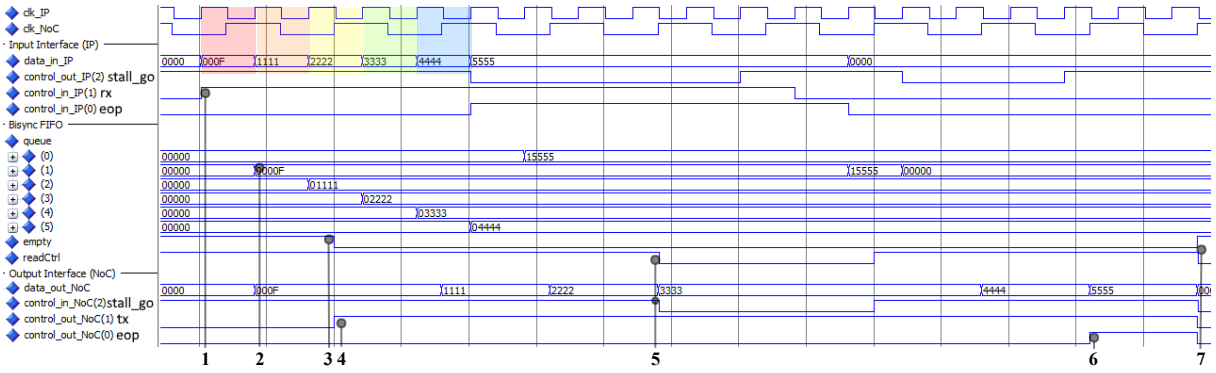


Figura 24 – Simulação demonstrando a operação da NI bissíncrona no modo de transmissão síncrono da Arke.

A Figura 24 mostra uma simulação demonstrando o funcionamento do modulo de transmissão da NI de fila bissíncrona. Nessa simulação ocorre o envio de um pacote (0x000F, 0x1111, 0x2222, 0x3333, 0x4444 e 0x5555) utilizando o modo de transmissão síncrono da Arke.

1. O IP que está conectado à NI bissíncrona começa a enviar os *flits*. Observe que cada novo *flit* é disponibilizado em um novo ciclo de *clock* do IP (destacado em cores na imagem).

2. Cada *flit* é armazenado na fila (*queue*) para aguardar a sincronização para o domínio de *clock* da NoC.

3. Uma vez que ocorre a sincronização do primeiro *flit* , o sinal *empty* indica ao controle de transmissão que a fila não está mais vazia.

4. O controle de transmissão lê o cabeçalho (0x000F) e observa o bit mais significativo para determinar qual o modo de comunicação da Arke deve ser utilizado. Neste caso o bit mais significativo é ‘0’, então inicia-se a transmissão em modo síncrono, utilizando o protocolo tradicional *stall and go*.

5. Neste momento, ocorre uma saturação no *Input Buffer* do roteador da NoC e o sinal *stall_go* baixa sinalizando que não possui mais espaço para armazenar *flits*. Essa informação é passada à fila bissíncrona. Uma vez que o *input buffer* tem espaço para um novo *flit*, o sinal *stall and go* retorna para nível alto e a transmissão continua.

6. Neste instante o último *flit* é transmitido e para indicar para a NoC que se trata do último *flit* do pacote, o sinal *eop* fica em '1'.

7. Assim que ocorre a transmissão do pacote, e nenhum outro pacote foi enviado pelo IP para a NI, o sinal *empty* indica que não há nada mais a ser transmitido nesse momento.

Uma vez que a operação da NI bissíncrona está detalhada para o protocolo síncrono, pode-se definir uma equação para determinar o tempo mínimo para transmissão de um pacote considerando a ausência de colisões dentro da NoC. Como pode ser visto na Equação 4.1, o tempo mínimo de transmissão (*Tempo*$_{min}$) é equivalente a soma do tempo de sincronização necessário dentro da NI transmissora (*Tempo*$_{NItx}$), com o tempo de transmissão do pacote dentro da NoC (*Tempo*$_{NoC}$) e o tempo de sincronização no NI receptor (*Tempo*$_{NIrx}$).

$$Tempo_{min} = Tempo_{NItx} + Tempo_{NoC} + Tempo_{NIrx} \tag{4.1}$$

A Equação 4.2 define o tempo de transmissão de um pacote dentro da NoC utilizando o modo síncrono. Ela é composta por duas parcelas, sendo que a primeira diz respeito ao tempo de transmissão do cabeçalho através dos roteadores que compõem o caminho entre os IPs transmissor e receptor, enquanto que a segunda parcela trata da transmissão do *payload* do pacote.

$$Tempo_{NoC} = 4N_{roteadores} + (N_{flits} - 1) \tag{4.2}$$

Uma vez que as filas bissíncronas trabalham com a sincronização em pipeline, o atraso na sincronização (*Tempo*$_{NItx}$ e *Tempo*$_{NIrx}$) ocorre apenas uma vez, pois enquanto o primeiro *flit* está sendo sincronizado o segundo chega. Durante a sincronização do segundo, o primeiro está sendo lido e o terceiro chega. Esse processo se repete até o final do pacote. Ou seja, se a transmissão mantiver uma taxa de um *flit* por ciclo, apenas um ciclo de sincronização será levado em consideração em cada uma das filas bissíncronas. Desta forma, o tempo mínimo para realizar a transmissão de um pacote, utilizando o protocolo síncrono, em ciclos de *clock* da NoC, por um IP que possua frequência igual ou superior a da NoC é definido pela Equação 4.3.

$$Tempo_{min_{sync}} = 4N_{roteadores} + N_{flits} + 1 \tag{4.3}$$

A Figura 25 mostra uma simulação que demonstra o funcionamento do modulo de transmissão da NI bissíncrona, utilizando o modo de transmissão assíncrono da Arke. Foi utilizado o

protocolo de *handshake* de duas fases *non-return-to-zero* (NRZ). Neste protocolo cada transição de um sinal de controle (*ack* e *req*) representa uma interação.

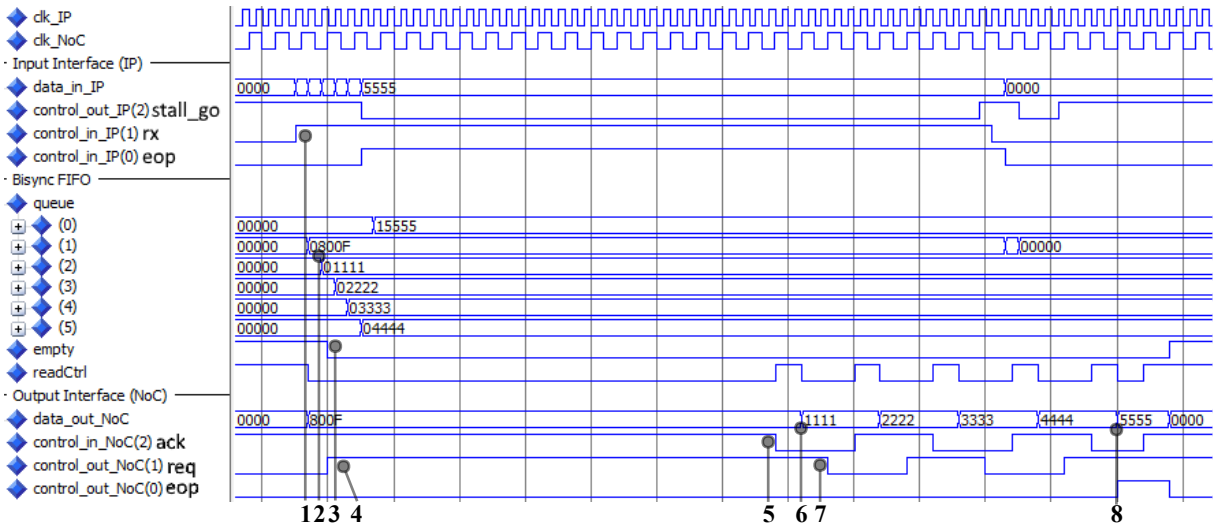


Figura 25 – Simulação demonstrando a operação da NI bissíncrona no modo de transmissão assíncrono da Arke.

1. O IP que está conectado a NI bissíncrona começa a enviar os *flits*.

2. Cada *flit* é armazenado na fila (*queue*) para aguardar a sincronização para o domínio de *clock* da NoC.

3. Uma vez que ocorre a sincronização do primeiro *flit* , o sinal *empty* indica ao controle de transmissão que a fila não está mais vazia.

4. O controle de transmissão lê o cabeçalho (0x800F) e observa o bit mais significativo para determinar qual o modo de comunicação da Arke deve ser utilizado. Neste caso o bit mais significativo é ‘1’, então inicia-se a transmissão em modo assíncrono, utilizado o protocolo de *handshake* de duas fases NRZ. O primeiro *request* (*req*) contendo o cabeçalho é feito. O cabeçalho irá atravessar a NoC, estabelecendo o caminho de *bypass* e conectando diretamente a NI transmissora com a NI receptora.

5. Neste instante o primeiro *acknowledgement* (*ack*) é recebido. Informando que o cabeçalho foi recebido pela NI receptora.

6. Uma vez que o *ack* é recebido, a NI transmissora faz a leitura na fila bissíncrona, alterando o *flit* no barramento de saída (*data_out*). Por se tratar de uma comunicação assíncrona, é necessário setar primeiro o dado antes do *req*, pois devemos garantir que haja tempo

suficiente para o sinal do barramento se estabilizar antes de realizar o *req*, o qual em um sistema assíncrono é responsável por validar o dado.

7. Após aguardar um ciclo, o *req* é feito, indicando à receptora que o *flit* no barramento é válido. Este processo se repete, até que todos os *flits* tenham sido enviados de forma assíncrona.

8. O último *flit* do pacote precisa trafegar de forma síncrona dentro da NoC, de maneira a encerrar as conexões de *bypass* que foram estabelecidas nos roteadores. Desta forma, o último *flit* não realiza *request*, apenas indica que é válido utilizando o bit de controle *eop*.

Assim como no protocolo síncrono, pode-se definir uma equação para determinar o tempo mínimo de transmissão de um pacote (T_{min}). Assim como no modo de transmissão síncrona, o tempo mínimo é definido pela Equação 4.1. Todavia, o tempo mínimo de transmissão de um pacote dentro da NoC (T_{NoC}) difere da situação anterior e é descrito pela Equação 4.4 dada a alteração de protocolos.

$$Tempo_{NoC} = (4N_{roteadores} + 2) + 3(N_{flits} - 2) + N_{roteadores} \tag{4.4}$$

A Equação 4.4 é composta por três parcelas de uma soma. A primeira delas se refere ao tempo de transmissão do cabeçalho pela NoC, este tempo é dado pela quantidade de ciclos que o cabeçalho fica em cada roteador além de dois ciclos que são adicionados devido a transmissão utilizando o protocolo assíncrono de duas fases NRZ. A segunda parcela engloba a transmissão dos *flits* de *payload*, exceto o último deles, são três ciclos para cada (um para *req*, um para colocar o dado no barramento e outro para o *ack*). A terceira parcela diz respeito a transmissão do último *flit* do pacote, que é transmitido um roteador por vez desconectando o *bypass* estabelecido. Combinando a Equação 4.4 na Equação 4.1 e rearranjando os termos, obtemos a Equação 4.5 que define o tempo mínimo de transmissão utilizando a NI bissíncrona com o protocolo assíncrono.

$$Tempo_{min_{async}} = 5N_{roteadores} + 3N_{flits} - 2 \tag{4.5}$$

As equações que definem o tempo mínimo de transmissão utilizando a NI com a fila bissíncrona foram estabelecidas. Com isso, é possível comparar o desempenho dos modos de

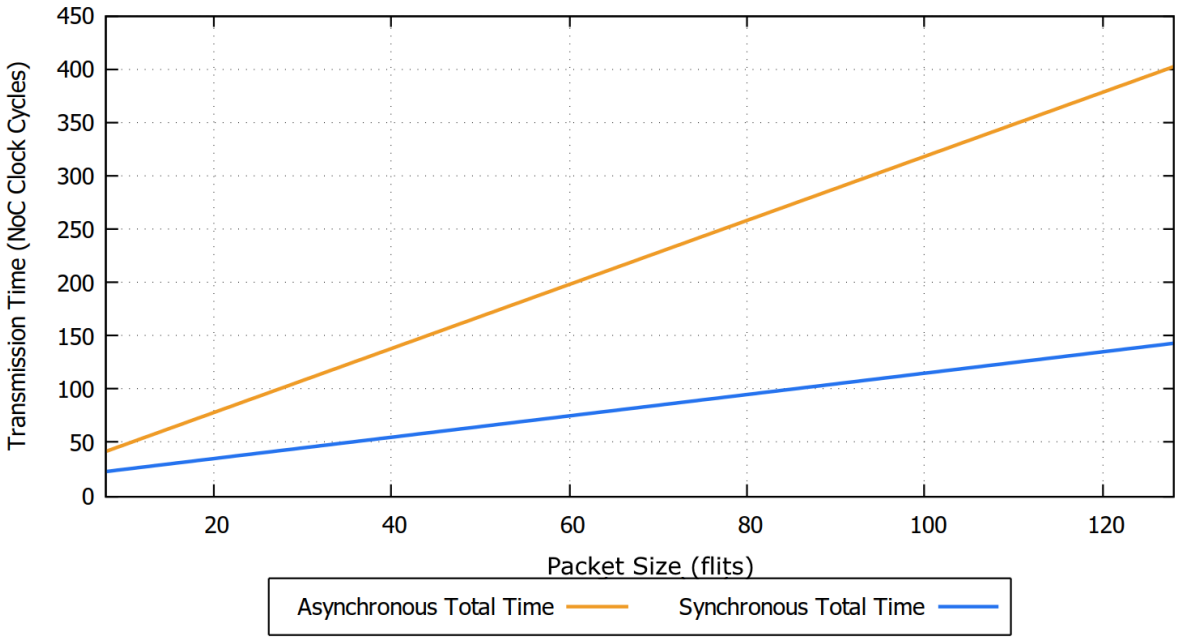


Figura 26 – Tempo mínimo de transmissão de pacotes em função do tamanho do pacote utilizando a NI bissíncrona.

comunicação quando operados pela NI de fila bissíncrona. Isso pode ser visto na Figura 26, que apresenta a variação do tempo mínimo de transmissão em função do tamanho do pacote.

Na Figura 26 observa-se que o modo de comunicação assíncrono tem um desempenho inferior em comparação ao modo de comunicação síncrono. Isso ocorre pois, como visto nesta seção, por mais que a fila bissíncrona consiga fornecer um *flit* por ciclo da NoC, o protocolo assíncrono, quando executado entre os NIs transmissor e receptor, demanda três ciclos para transmitir um *flit*. A utilização do modo de comunicação assíncrono pela NI com fila bissíncrona é sempre desvantajosa. Com intuito aproveitar a conexão fim-a-fim que a Arke em modo assíncrono oferece, foi desenvolvido uma NI com uma fila circular assíncrona, com o intuito de extender o protocolo handshake assíncrono até os IPs.

4.2 NI DE FILA CIRCULAR ASSÍNCRONA

A NI com fila circular assíncrona surge como uma proposta para aproveitar de maneira mais eficiente a conexão fim-a-fim que a NoC Arke provê quando operando em modo assíncrono. Esta NI foi desenvolvida com capacidade de operar apenas com o modo assíncrono. A ideia por trás dessa fila surgiu após observar que ao realizar a transmissão utilizando o protocolo assíncrono dentro do domínio de *clock* da NoC ocorreria um aumento de latência impeditivo. O objetivo desta NI é permitir que os IPs possam, através da conexão fim-a-fim da Arke, se

comunicarem sem a interferência do domínio de *clock* da NoC. Para atingir esse objetivo foi criada uma fila circular assíncrona. A utilização de uma fila especificamente circular, é necessária devido ao modo como a sincronização será realizada nessa NI (sincronização de borda) e ela é assíncrona para que a comunicação fim-a-fim possa ocorrer entre IPtx-IPrx e não entre NItx-NIrx (como acontecia na NI de fila bissíncrona). A fila circular assíncrona é construída a partir de *N* estágios MOUSETRAP, Figura 27, e dois contadores Johnson assíncronos de *N*-bits, como pode ser visto na Figura 28. Cada estágio MOUSETRAP gera uma posição da fila. Cada um dos estágios da fila possui um sinal (*slotFree* que indica se o estágio está cheio ou vazio, é o conjunto destes sinais (um bit de cada estágio) que será sincronizado. Dessa forma, cada posição da fila é sincronizada individualmente, de forma paralela e os IPs origem e destino necessariamente precisam respeitar a ordem da fila. Assim, o IP origem escreve na posição 0, seguido da 1, 2, até a enésima posição, enquanto o IP destino lê destas posições. Desta forma, como o estado de cada posição da fila é sincronizado individualmente, o IP origem pode realizar diversas escritas enquanto aguarda os *acknowledgements* oriundos da leitura do IP destino, isso irá permitir à NI de fila circular assíncrona obter vantagens de desempenho em relação à NI de fila bissíncrona que necessariamente precisava aguardar o *acknowledgement* de cada *flit* antes de realizar o próximo *request*.

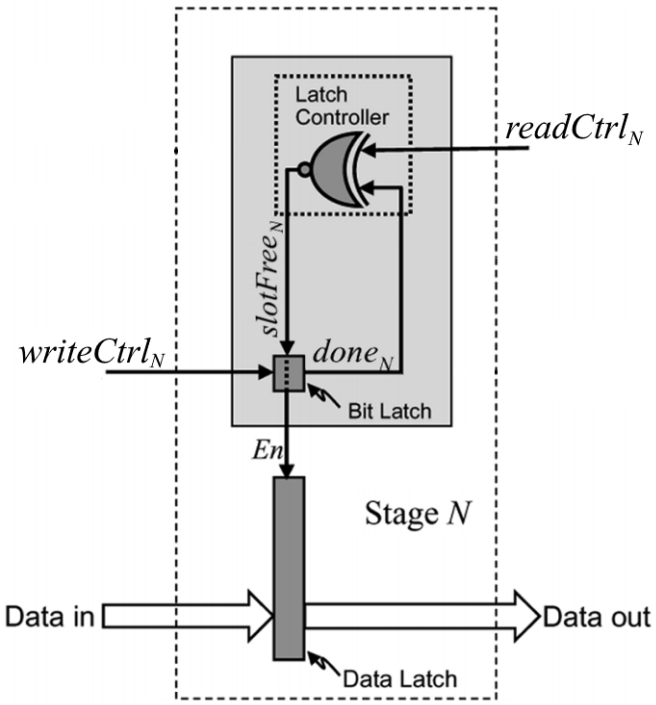


Figura 27 – Estágio MOUSETRAP (Singh and Nowick, 2007).

A fila emprega o mesmo protocolo assíncrono que foi utilizado na NI de fila bissín-

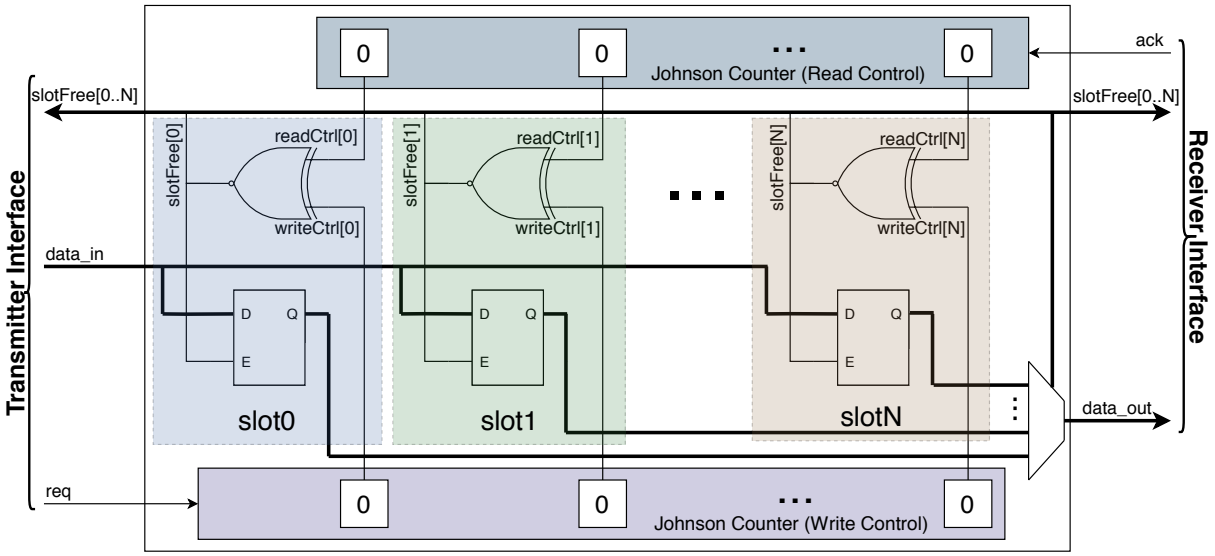


Figura 28 – Interface e organização da fila circular assíncrona.

crona, o *handshake* de duas fases NRZ. Ao iniciar uma escrita, o transmissor deve observar se a primeira posição da fila está vazia. Ele tem acesso a um mapeamento de todas as posições da fila através do sinal *slotFree*. Caso a primeira posição esteja livre (*slotFree*[0] = '1') o transmissor disponibiliza o dado no barramento de entrada (*data_in*) seguido de uma transição do *req*. Todos os *latches* recebem o dado simultaneamente, pois estão conectados em paralelo, porém, a escrita não é controlada diretamente pelo sinal *req*. O *req* aciona um contador Johnson responsável por controlar as escritas. O contador Johnson é assíncrono e sensível a transições, cada transição da entrada incrementa sua saída. Cada bit do contador Johnson é responsável por controlar a escrita em uma posição da fila. Após o incremento do controle de escrita, os bits *writeCtrl* e *readCtrl* estarão diferentes, que por sua vez irá alterar o estado do bit *slotFree* referente a posição que foi ocupada (*slotFree*[0] = '0'). O transmissor pode continuar realizando escritas na fila enquanto houverem posições livres. A liberação de uma posição ocorre após o receptor realizar a leitura do dado no barramento de saída (*data_out*) e enviar uma transição no sinal *ack*. Assim como na escrita, a leitura também é controlada por um contador Johnson assíncrono sensível a transições. Após o incremento do controle de leitura, os sinais *readCtrl* e *writeCtrl* voltarão a ser iguais, resultando na habilitação para escrita na posição. A Figura 29 mostra uma simulação do funcionamento da fila circular assíncrona.

1. Neste exemplo temos uma fila circular assíncrona de sete posições. Antes de realizar uma escrita na fila circular assíncrona o transmissor deve verificar se o estado da primeira posição está livre. Neste caso, por se tratar da primeira escrita na fila, a primeira posição

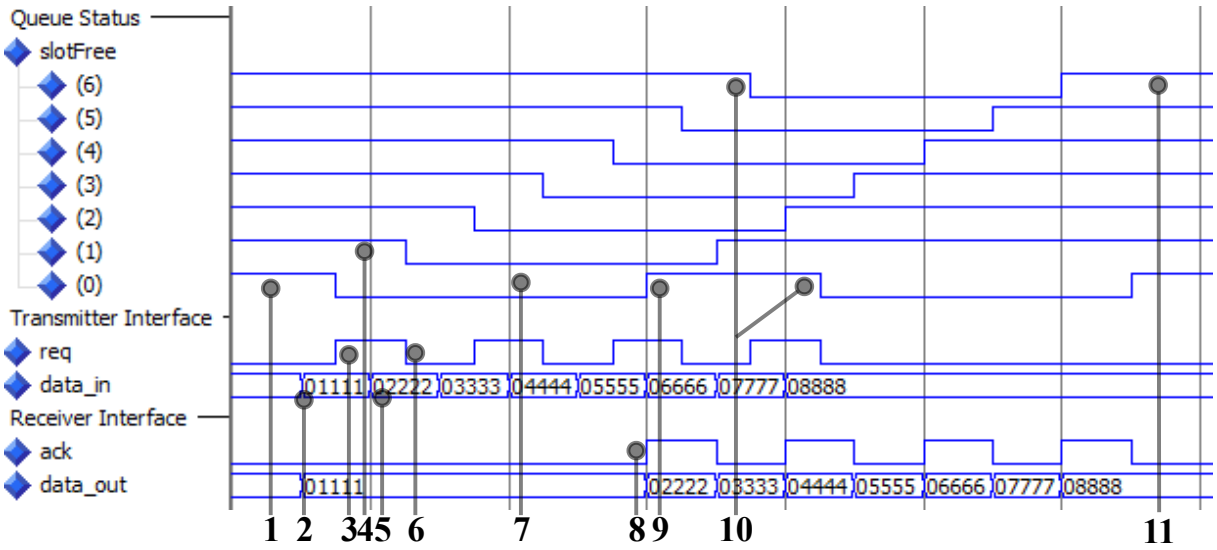


Figura 29 – Simulação demonstrando o funcionamento da fila circular assíncrona.

é a de número zero.

2. Após verificar que a posição na qual deseja-se escrever está livre, o transmissor insere no barramento o dado que será transmitido (*data_in* ← x“01111”).

3. Em seguida o transmissor transiciona o sinal *req*, indicando a validade do dado. Ao acionar a escrita o dado é armazenado no *latch* e o primeiro estágio é declarado ocupado (*slotFree*[0] = ‘0’).

4. O transmissor avalia se o próximo estágio da fila está disponível através do sinal *slotFree*[1].

5. Insere o novo dado no barramento (*data_in* ← x“02222”).

6. Faz uma transição no sinal *req*, indicando a validade do dado. O processo de verificar se há espaço livre através do *slotFree*, disponibilizar o novo dado em *data_in* e transicionar o *req* se repete enquanto houver posições livres, ou até que não haja mais dados para serem enviados.

7. O receptor deve monitorar o estado do primeiro estágio da fila, aguardando uma indicação de que aquele estágio está ocupado. Ao detectar o *slotFree*[0] = ‘0’, o receptor lê o dado que está no barramento de saída da fila circular assíncrona (*data_out*).

8. Após realizar a leitura do dado, o receptor transiciona o sinal de *ack* indicando que realizou a leitura da primeira posição da fila.

9. Recebendo o *ack* do receptor, o controle do estágio MOUSETRAP indicará que o estágio está livre, rotacionado a fila. Desta forma, a primeira posição passa a ser a próximo estágio (neste caso o de número um).

10. Quando o último estágio (neste exemplo, o de número seis) é ocupado, o próximo a ser ocupado será o primeiro estágio, por isso é denomina-se "circular".

11. O processo de leitura, que consiste em observar o status do primeiro estágio e enviar o *ack* após reconhecer que ele foi ocupado, permanece se repetindo até que a fila esteja vazia.

Como pode ser observado, a fila circular assíncrona não possui a mesma interface que os roteadores da NoC. Desta forma, para utilizar infraestrutura fornecida pela NoC sem realizar modificações em sua interface, a NI é composta por dois módulos, um de transmissão e outro de recepção. O módulo de transmissão possui apenas os controles de escrita e leitura (contadores Johnson). Já o módulo de recepção possui a fila circular assíncrona completa. Desta forma, a interface entre a NoC e a NI é preservada, sendo composta pelo barramento de dados e os três bits de controle (*tx/req, stall_go/ack* e *eop*). Todavia, a interface entre a NI e o IP foi alterada. Os IPs devem utilizar o protocolo que foi apresentado durante a descrição do funcionamento da fila circular assíncrona. Os IPs observam o estado da fila (através dos sinais *FIFO_slot_free_tx e FIFO_slot_free_rx*) e realizam *requests* para confirmar a validade dos dados e *acknowledgments* para indicar a leitura dos dados. A Figura 30 apresenta a organização simplificada da NI de fila circular assíncrona.

Como visto na Seção 3.1.1.2, a comunicação assíncrona precisa ser estabelecida e após a transmissão, finalizada. O processo de estabelecimento e encerramento de conexão é feito no domínio de *clock* da NoC, ou seja, o controle da NI de fila circular assíncrona opera na frequência da NoC. A comunicação fim-a-fim assíncrona, que envolve a fila circular assíncrona, ocorre durante a etapa de transmissão do modo de comunicação assíncrono da Arke. Uma vez que a conexão assíncrona é estabelecida, é criado um caminho que conecta a entrada do primeiro roteador à saída do último roteador. A NI de fila circular assíncrona consegue explorar essa capacidade da Arke para permitir que os IPs se comuniquem com uma taxa de transmissão até mesmo superior à um *flit* por ciclo, que é a taxa máxima que a NI de fila bissíncrona consegue alcançar operando em modo síncrono. Isso só pode ocorrer porque ao realizar o *bypass* dos *Input Buffers* dos roteadores os sinais podem trafegar pela NoC de forma direta. Ou seja, quando a comunicação assíncrona entra em fase de transmissão o *request* que o IP transmis-

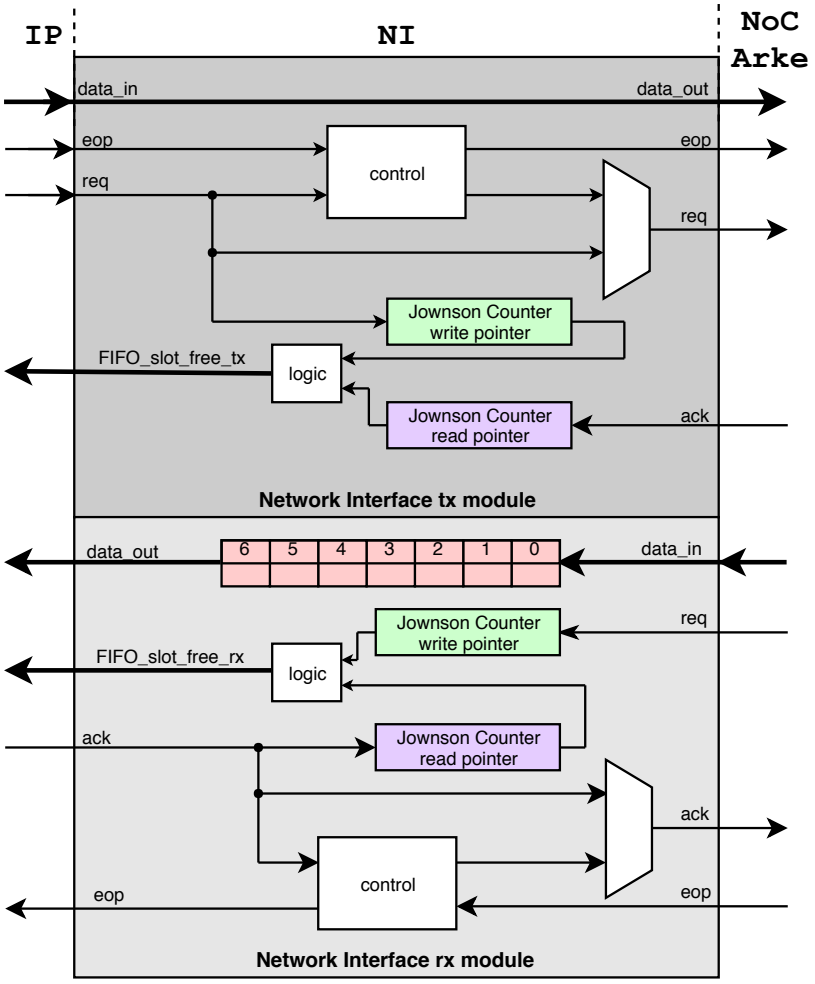


Figura 30 – Interface e organização da NI de fila circular assíncrona.

sor envia para o módulo de transmissão da NI é repassado diretamente para a entrada da NoC e é transmitido sem armazenamentos intermediários pelos roteadores que formam o caminho até alcançar o módulo de recepção da NI receptora. Desta forma, o *request* aciona ambos os controles de escrita (no módulo de transmissão da NI transmissora e no módulo de recepção da NI receptora), Figura 30. O *flit* é armazenado na fila circular assíncrona dentro do módulo de recepção e ficar disponível para leitura do IP receptor. Após ler o *flit*, o IP receptor envia o *acknowledgment* que irá acionar o controle de leitura no módulo de recepção, liberando a posição da fila. O *ack* também é repassado diretamente para a NoC por onde ocorre a transmissão sem armazenamentos intermediários. Ao alcançar o módulo de transmissão da NI transmissora, também irá acionar o controle de leitura, que por sua vez indicará ao IP transmissor que o *flit* foi lido, Figura 30. Esse é o processo utilizado nas execuções das comunicações assíncronas que utilizam a fila circular assíncrona.

Esta é uma NI desenvolvida para atuar em um cenário GALS, ou seja, com cada IP

operando em seu próprio domínio de *clock*. De forma a evitar metaestabilidade nos dados que atravessam esses domínios de *clock* é necessária a utilização de alguma técnica de sincronização. A NI de fila bissíncrona possui a própria fila que é desenvolvida para realizar adaptação dos domínios de *clock*. A fila circular assíncrona não possui essa capacidade, portanto, foi necessário empregar outra técnica para realizar a adaptação entre os domínios de *clock* da NoC e dos IPs. Optou-se pela utilização da sincronização de borda, principalmente em razão de sua simplicidade e baixo custo de área. A sincronização de borda ocorre individualmente nos sinais que atravessam de um domínio de *clock* para outro, sem que seja necessário parar o funcionamento local do bloco síncrono. Desta forma, todos os sinais de controle devem ser sincronizados durante a transmissão de um pacote. Para que um IP possa enviar um *flit* a cada ciclo de *clock* sem interrupções, é necessário uma fila que tenha posições suficientes para armazenar tantos *flits* quanto forem os ciclos necessários para que um *flit* seja lido pelo IP receptor. A este tempo, de enviar o *request* até receber o *acknowledgment*, se dá o nome de *round trip* do *flit*. No caso da implementação adotada, o *round trip* possui seis ciclos de *clock*, que é compreendido da seguinte forma: (ciclo *I*) o IP transmissor envia o *flit* seguido do *request* que fará a escrita na fila; (ciclos *II* e *III*) tempo necessário para realizar a sincronização de borda para o domínio de *clock* do IP receptor; (ciclo *IV*) envio do *acknowledgment* informando a leitura do dado para o IP transmissor; (ciclos *V* e *VI*) sincronização de borda para o domínio de *clock* do IP transmissor. Desta forma, a fila circular assíncrona foi projetada com sete posições, de forma que contemple o *round trip* em um caso onde IPs de frequências semelhantes estejam se comunicando. Veja na Figura 31 e na Figura 32 o funcionamento da NI sob a perspectiva do módulo de transmissão.

1. O IP deve constantemente avaliar a situação do bit menos significativo do estado da fila circular assíncrona (*FIFO_slotFree*). Como visto anteriormente, a fila está implementada apenas no módulo receptor. Desta forma, quando um novo estabelecimento de conexão estiver apto a ser iniciado a NI irá informar que apenas a primeira posição da fila está livre (*FIFO_slotFree* = 0b0000001), permitindo o envio somente do cabeçalho.

2. Após detectar a disponibilidade da NI para estabelecer uma nova conexão o IP disponibiliza o cabeçalho através do barramento *data_in*, seguido de um *request*, que informará à NI a validade do *flit*. Por se tratar de um sinal de controle gerado em um domínio de *clock* do IP, é necessário realizar sincronização do sinal *req* para evitar metaestabilidade.

3. Neste instante a NI recebe o *request* sincronizado.

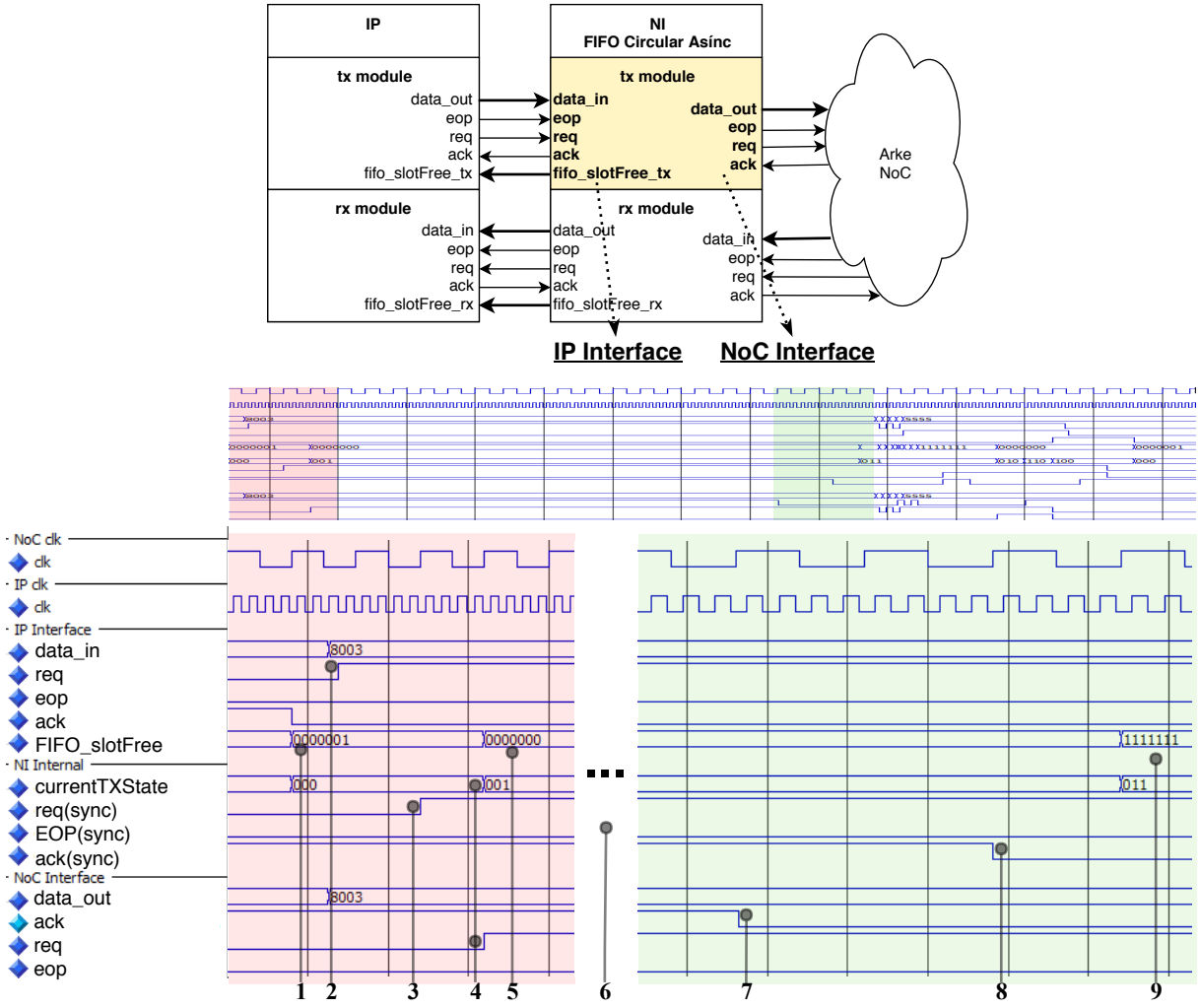


Figura 31 – Simulação demonstrando a operação do módulo de transmissão da NI de fila circular assíncrona. Cobrindo a fase de estabelecimento do modo de comunicação assíncrono da Arke.

4. Após receber o *request*, o estado da NI passa de *waiting* (“000”) para *waitAck* (“001”). Neste estado será enviado para a NoC o *request* gerado no domínio de *clock* da NoC e aguardar-se-a o recebimento do *ack*, enviado pelo módulo receptor da NI receptora.

5. No estado *waitAck* a NI mantém o estado da fila como totalmente ocupado (*FIFO_slotFree* = 0b0000000), pois o caminho assíncrono está sendo alocado e o módulo transmissor não possui uma fila para armazenamento temporário dos *flits*.

6. Neste ponto os sinais observados estão estáveis pois a NoC está realizando a transmissão do cabeçalho e o alocando os canais de *bypass* até a NI receptora. Foi omitido esta etapa da simulação pois já foi coberta com detalhes na Seção 3.1.1.2.

7. Após o cabeçalho chegar na NI receptora o caminho assíncrono formado pelos *bypasses* está formado. Após detectar o *request* o módulo de recepção envia o *acknowledgment*.

Por mais que esse *ack* tenha sido gerado de maneira síncrona no domínio de *clock* da NoC, ele será transmitido diretamente pelo caminho assíncrono. Por isso, necessita de sincronização ao chegar no módulo transmissor da NI.

8. Neste ponto o módulo de transmissão da NI detecta o *ack*, que foi sincronizado, gerado pela NI receptora.

9. Após receber o *ack* que confirmou o estabelecimento da conexão, o estado da NI transmissora passa para *asyncTransmittion* (“011”). Neste estado os sinais que indicam a situação da fila circular assíncrona (*FIFO_slotFree*) passam a ser gerados pelos controles de escrita e leitura assíncronos (Figura 30), indicando a situação da fila circular assíncrona do módulo de recepção da NI receptora.

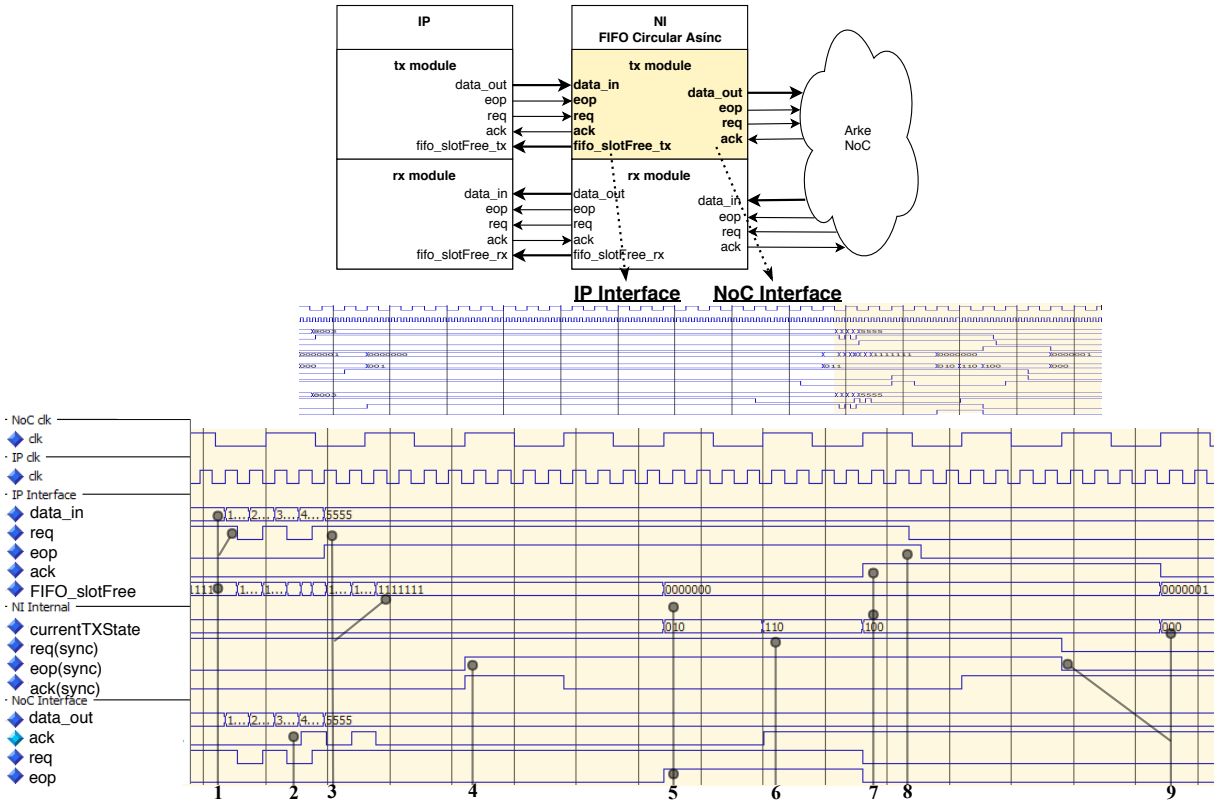


Figura 32 – Simulação demonstrando a operação do módulo de transmissão da NI de fila circular assíncrona. Cobrindo a fase de transmissão e encerramento do modo de comunicação assíncrono da Arke.

1. Os sinais de estado da fila circular assíncrona (*FIFO_slotFree*) gerados de maneira assíncrona pelos controladores de escrita e leitura da fila circular assíncrona precisam passar por sincronização (para o domínio de *clock* do IP) para serem lidos de forma correta e evitar metaestabilidade. Assim que o IP detecta que há posições livres na fila começa a realizar transmissões colocando o *flit* no barramento seguido de uma transição do *req*. Cada transição do *req* executa uma escrita na fila circular assíncrona.

2. Os *flits* acompanhados dos seus respectivos *requests* são transmitidos até o módulo de recepção da NI receptora onde são armazenados e aguardam pela leitura do IP local (destino do pacote). Após a leitura o IP envia um *ack* que libera a posição da fila no módulo de recepção da NI receptora, além disso é transmitido até o módulo de transmissão da NI origem onde aciona o controle de leitura e libera uma posição da fila circular assíncrona. Desta forma, cada *request* e *acknowledgment* muda o estado da fila circular assíncrona.

3. Após disponibilizar o último *flit* do pacote o IP sinaliza apenas o bit *eop* e não o *request*, como manda o protocolo. Novamente, por se tratar de um sinal de controle gerado no domínio de *clock* do IP, é necessário realizar sincronização. Além de aguardar a sincronização do bit indicador de *eop* a NI precisa garantir que a fila circular assíncrona está vazia. Para isso, o sinal *FIFO_slotFree* também é sincronizado.

4. Neste instante ocorre a sincronização do bit indicativo de *eop*.

5. Após detectar que a fila circular assíncrona foi totalmente lida e que o *flit eop* está disponível, a NI muda para o estado de *sendEOP* (“010”). Neste estado a NI envia para a NoC o sinal indicativo de *eop*, gerado no domínio de *clock* da NoC. Além disso, a NI toma controle do *FIFO_slotFree*, indicando que não há mais espaço na fila circular assíncrona.

6. O próximo estado (*holdEOP* - 0b“110”) apenas mantém os sinais estáveis na entrada da NoC. Isso é necessário pois o roteador leva um ciclo para desfazer a conexão assíncrona, e qualquer alteração nos sinais de controle poderia refletir em uma escrita inapropriada na fila circular assíncrona.

7. O último estado do módulo de transmissão da NI é *closeConnection* (“100”). Neste estado a NI informa ao IP que o *flit eop* foi transmitido para a NoC e que a transmissão foi finalizada. Isto é comunicado ao IP através do sinal *ack*.

8. Uma vez que o IP detecta o *ack* (que deve ser sincronizado, pois é gerado no domínio de *clock* da NoC), retorna os sinais de controle para o estado inicial *req* = '0' e *eop* = '0'.

9. Após detectar que o IP recebeu a informação de que a conexão foi finalizada, através do *eop* sincronizado, o estado do módulo de transmissão da NI retorna para o estado inicial (*waiting* - "000") onde esperará por um novo pacote para ser transmitido.

Para esclarecer o funcionamento completo da NI será apresentada a mesma comunicação sob a perspectiva do módulo de recepção. Observe a Figura 33.

1. Após o cabeçalho do pacote atravessar a NoC formando o caminho assíncrono, ele chega no módulo receptor da NI receptora. O *req* aciona a NI receptora, informando que uma nova comunicação está sendo iniciada.

2. O estado da NI receptora passa de *waiting*, onde aguardava-se uma nova comunicação, para *asyncReicieving*. Neste estado o IP local lê os dados que estão sendo escritos pelo IP transmissor em sua fila circular assíncrona, enquanto isso a NI apenas mantém-se aguardando o final da comunicação.

3. Neste ponto vemos a comunicação assíncrona ocorrendo. Cada *request* enviado pelo IP transmissor faz uma escrita na fila circular assíncrona, enquanto cada *acknowledgment* enviado pelo IP receptor libera uma posição. Observe que as posições livres permanecem transparentes, pois a fila é formada por *latches*.

4. Após receber a sinalização de *eop* a NI fará a escrita do último *flit* de forma síncrona na fila circular assíncrona. O primeiro passo é dado ao trocar o estado para *dataDisponibilization*. Neste estado a NI mantém o último *flit* na entrada da fila circular assíncrona. Vale

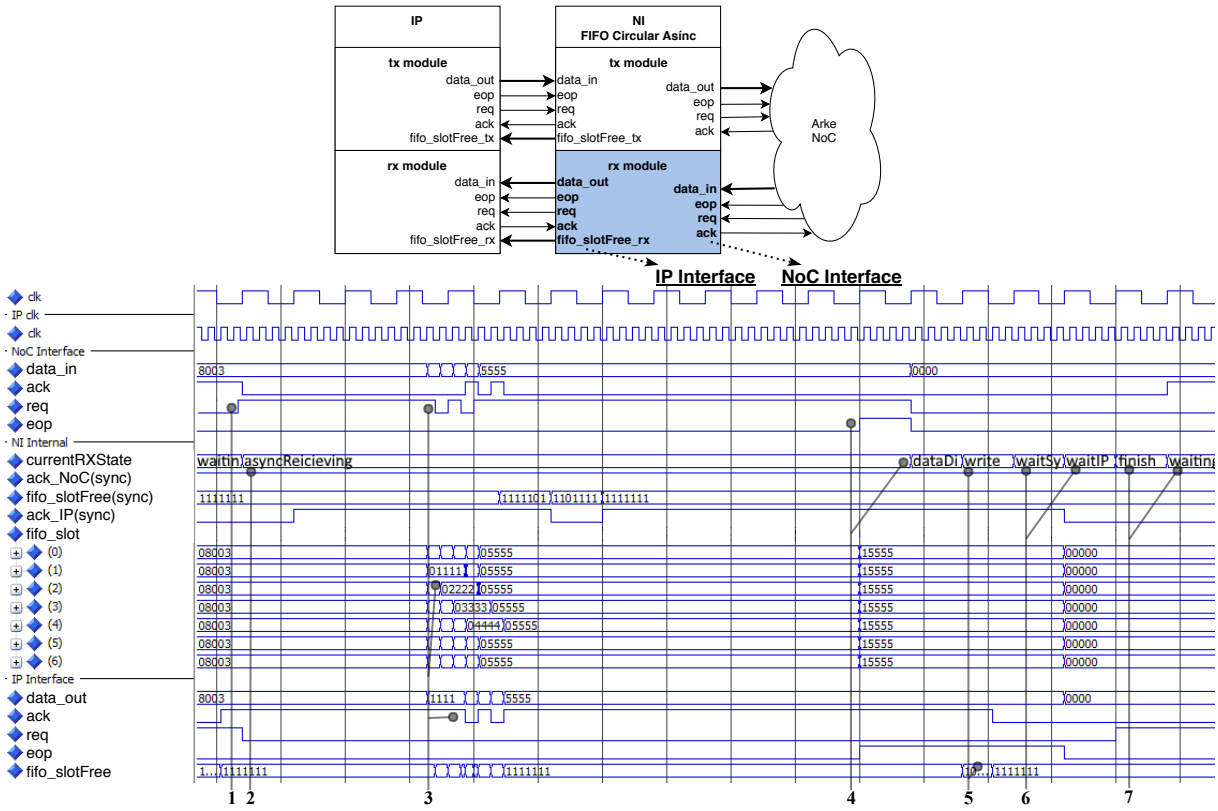


Figura 33 – Simulação demonstrando a operação do módulo de recepção da NI de fila circular assíncrona. Cobrindo a fase de transmissão e encerramento do modo de comunicação assíncrono da Arke.

ressaltar que neste ponto o roteador que se conecta à NI já desfez a comunicação e não está mais com o dado (0x5555) no barramento *data_in* da NI.

5. Em seguida a NI passa para o estado *write* onde ocorre a escrita do *flit* na fila circular assíncrona. Pode-se observar que através do sinal *fifo_slotFree* que ocorerreu uma escrita, seguida de uma leitura.

6. O próximos dois estados (*waitSynchronization* e *waitIP*) são necessários apenas para uma situação onde o IP opere em uma frequência inferior à da NoC. O estado *waitSynchronization* fornece tempo para que ocorra a sincronização do estado da fila circular assíncrona, de forma que fique visível à NI que existe uma posição ocupada na fila (posição que foi ocupada pelo último *flit*). Enquanto que o estado *waitIP* é necessário para que a NI aguarde a leitura do último *flit* que foi escrito na fila circular assíncrona.

7. Após garantir que o IP receptor recebeu o último *flit*, a NI finaliza a conexão retornando os sinais de controle para o estado original, bem como restabelece a fila para que a próxima escrita seja na posição inicial, isso ocorre no estado *finish*, logo em seguida a NI passa

para o estado *waiting* onde irá aguardar por uma nova comunicação.

4.2.1 Hazard - o problema

Durante as simulações da NI de fila circular assíncrona combinado com o protocolo de comunicação assíncrono da Arke foram encontrados problemas relacionados a *hazard*. Um *hazard* é uma alteração lógica transitória que ocorre na saída de um circuito devido a alterações em suas entradas. Este é um efeito que causa poucas consequências em sistemas totalmente síncronos, pois o tempo para que os sinais se propaguem e estabilizem adequadamente são englobados pelo período do *clock*. Todavia, em sistemas que possuem elementos assíncronos, como é o caso do sistema aqui apresentado, que possui uma fila circular assíncrona dentro da *Network Interface*, tais perturbações em sinais de controle repercutem em funcionamento incorreto do dispositivo. Existem três tipos de *hazard*, (*i*) estático, (*ii*) dinâmico e (*iii*) funcional. O *hazard* estático ocorre quando a saída oscila momentaneamente para um valor lógico incorreto após uma entrada ter sido alterada. O método mais comum para eliminar o *hazard* estático é a adição de redundâncias lógicas. O *hazard* dinâmico ocorre quando sucessivas oscilações lógicas ocorrem na saída do circuito em reflexo a uma alteração em uma das entradas. A correção do *hazard* dinâmico pode ser complexa, porém, se todos os *hazards* estáticos tiverem sido removidos do circuito o *hazard* dinâmico não ocorre. Por fim, o *hazard* funcional ocorre quando mais de uma entrada é alterada, resultando em oscilações na saída. Não existe uma única solução especifica para eliminar este tipo de problema. Uma solução seria prevenir que mais de uma entrada sofresse alterações simultaneamente, o que não é uma opção para determinados casos.

Foram detectados dois tipos de *hazard* advindos do *Input Buffer* da Arke. Em primeiro lugar, o multiplexador responsável pelo sinal de controle *stall_go* (e *ack* do protocolo assíncrono) estava gerando *hazard* estático. Para solucionar este problema, adicionamos uma porta lógica que cria uma camada de redundância e impede que se forme um *hazard* durante a alteração de apenas uma entrada. A solução empregada pode ser observada na Figura 34. O multiplexador anti-*hazard* possui uma parcela redundante (i0 AND i1), responsável por garantir que quando ambas as entradas (i0 e i1) estejam em ‘1’ uma mudança no bit seletor não crie uma oscilação para ‘0’ na saída.

Outro multiplexador que apresentou problemas de *hazard* foi o multiplexador responsável pelo sinal de *tx* (e *req* no protocolo assíncrono). Como este é um multiplexador de quatro

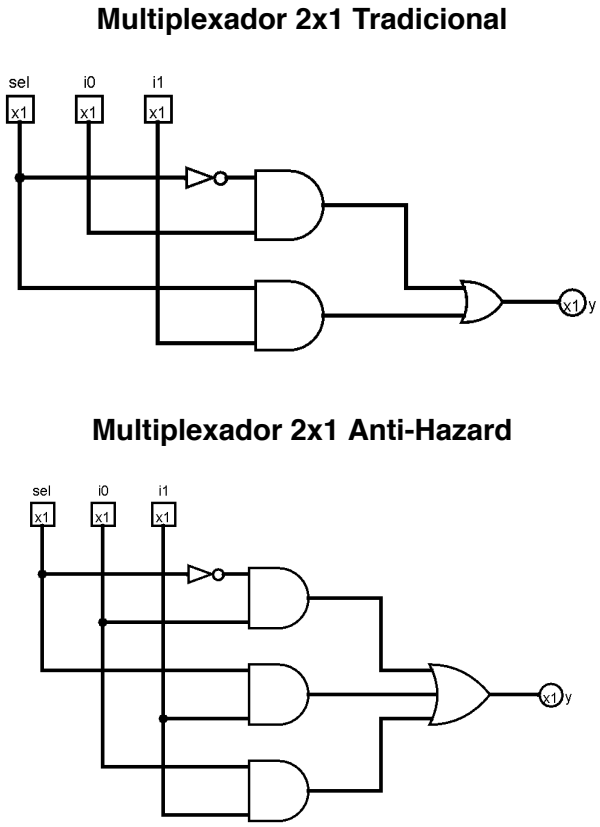


Figura 34 – Implementação do multiplexador Anti-Hazard utilizado na Arke.

entradas, construímos a partir de dois multiplexadores *anti-hazard* dois para um, que foi utilizado na solução do primeiro problema. Desta maneira eliminamos o problema de *hazard* estático. Porém, este multiplexador, sendo de quatro entradas, utiliza dois bits como seletores. Havia momentos em que ambos os bits eram alterados, gerando agora *hazard* funcional. Para solucionar o problema de *hazard* funcional, foi alterada a codificação dos estados do *Input Buffer* (visto que o controle do multiplexador depende do estado), garantindo que todas as alterações fossem de apenas um bit, codificação do tipo Gray, impedindo que esta forma de *hazard* ocorresse.

4.2.2 Análise Algébrica de Desempenho

A última etapa para obtermos a descrição completa da NI de fila circular assíncrona é a definição do tempo mínimo de transmissão. Este tempo engloba todas as etapas da transmissão, cada parcela que contribui para o tempo de transmissão pode ser observado na Tabela 1.

Somando todos os tempos, organizando os termos e convertendo os tempos referentes aos períodos dos IPs em ciclos de *clock* da NoC, obtemos a Equação 4.6, que define o tempo mínimo de transmissão utilizando o modo de comunicação assíncrono com a NI de fila circular

Tabela 1 – Tempo de cada etapa da transmissão utilizando a NI de fila circular assíncrona

Tempo	**Descrição**
2 $Ciclos_{NoC}$	Sincronização do *request*
$4N_{roteadores}$ $Ciclos_{NoC}$	Estabelecimento da conexão assíncrona pela NoC
2 $Ciclos_{NoC}$ + 2 $Período_{IPrx}$	Ativação do modo assíncrono
2 $Período_{IPtx}$	Sincronização do *FIFO_slotFree*
(N_{flits}-2)min($Período_{IPtx}$,$Período_{IPrx}$)	Transmissão assíncrona
2 $Período_{IPrx}$	Sincronização do *FIFO_slotFree*
2 $Ciclos_{NoC}$	Sincronização do *FIFO_slotFree*
2 $Ciclos_{NoC}$	Sincronização do *eop*
$N_{roteadores}$ $Ciclos_{NoC}$	Transmissão do *eop*
6 $Ciclos_{NoC}$ + 2 $Período_{IPrx}$	Finalização da comunicação

assíncrona. As variáveis T_{IPtx}, T_{IPrx} e T_{NoC} representam período, medido em segundos.

$$Tempo_{min} = 14+5N_{roteadores}+\frac{2T_{IPtx} + 6T_{IPrx} + (N_{flits} - 2)\min(T_{IPrx}, T_{IPtx})}{T_{NoC}} \tag{4.6}$$

Considerando que ambos os IPs possuam um *clock* com o mesmo período (T_{IPrx} = T_{IPtx}), podemos simplificar e reescrever a equação da seguinte forma:

$$Tempo_{min_{async}} = 14 + 5N_{roteadores} + \frac{T_{IP}(6 + N_{flits})}{T_{NoC}} \tag{4.7}$$

Observe que ao contrário da Equação 4.3 (que define o tempo mínimo de transmissão utilizando o modo síncrono e a fila bissíncrona) e da Equação 4.5 (que define o tempo mínimo de transmissão utilizando o modo assíncrono e a fila bissíncrona), a Equação 4.7 leva em consideração as frequências de operação dos IPs e da NoC. Ou seja, o desempenho da comunicação assíncrona utilizando a NI de fila circular assíncrona varia em função da razão entre a frequência da NoC e dos IPs. Observe a Figura 35.

Na Figura 35 é apresentado um gráfico que mostra o tempo mínimo de transmissão do modo síncrono, em verde, utilizando a NI de fila bissíncrona (Equação 4.3) e do modo assíncrono, em azul, utilizando a NI de fila circular assíncrona (Equação 4.7). A linha formada pela intersecção dos planos define o momento em que utilizar a comunicação assíncrona com a NI de fila circular assíncrona começa a ser vantajoso em termos de latência. Esta linha pode ser definida matematicamente da seguinte forma:

$$Tempo_{min_{sync}} = Tempo_{min_{async}} \tag{4.8}$$

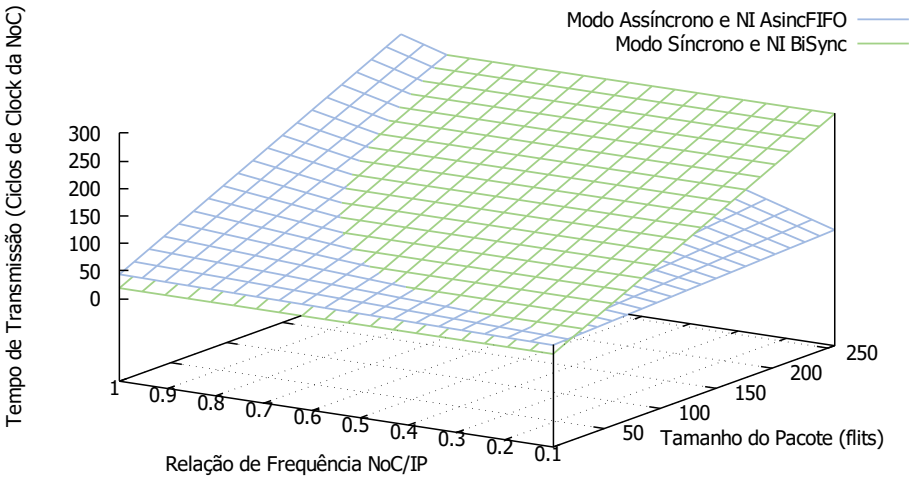


Figura 35 – Gráfico mostrando o tempo mínimo de transmissão do modo síncrono com a NI de fila bissíncrona e do modo assíncrono com a NI de fila circular assíncrona em função do tamanho do pacote e da razão de frequência NoC/IP.

Igualando os tempos de transmissão, Equação 4.8, obteremos o momento em que a comunicação assíncrona passa a ser mais eficiente do que a síncrona. Substituindo as equações e simplificando obtemos a seguinte relação:

$$15 + N_{roteadores} - N_{flits} + \frac{T_{IP}(6 + N_{flits})}{T_{NoC}} = 0 \tag{4.9}$$

Fixando, por exemplo, o número de roteadores em quatro, podemos definir uma relação direta entre o número de *flits* do pacote e a razão entre a frequência da NoC e dos IPs através da Equação 4.10.

$$N_{flits} = \frac{19 + 6\frac{T_{IP}}{T_{NoC}}}{1 - \frac{T_{IP}}{T_{NoC}}} \tag{4.10}$$

A curva gerada pela equação, Figura 36, apresenta graficamente o limite do desempenho da comunicação síncrona. A região abaixo da curva (em azul) são configurações (tamanho de pacote e razão da frequência da NoC e IP) onde a comunicação ocorrerá de forma vantajosa para o modo síncrono utilizando a NI de fila bissíncrona, enquanto que configurações na região acima da curva (em verde) apresentarão desempenho vantajoso para a comunicação assíncrona utilizando a NI de fila circular assíncrona. Por exemplo, se os IPs estiverem operando em uma frequência duas vezes superior à da NoC (NoC/IP = 0.5), o tamanho mínimo do pacote

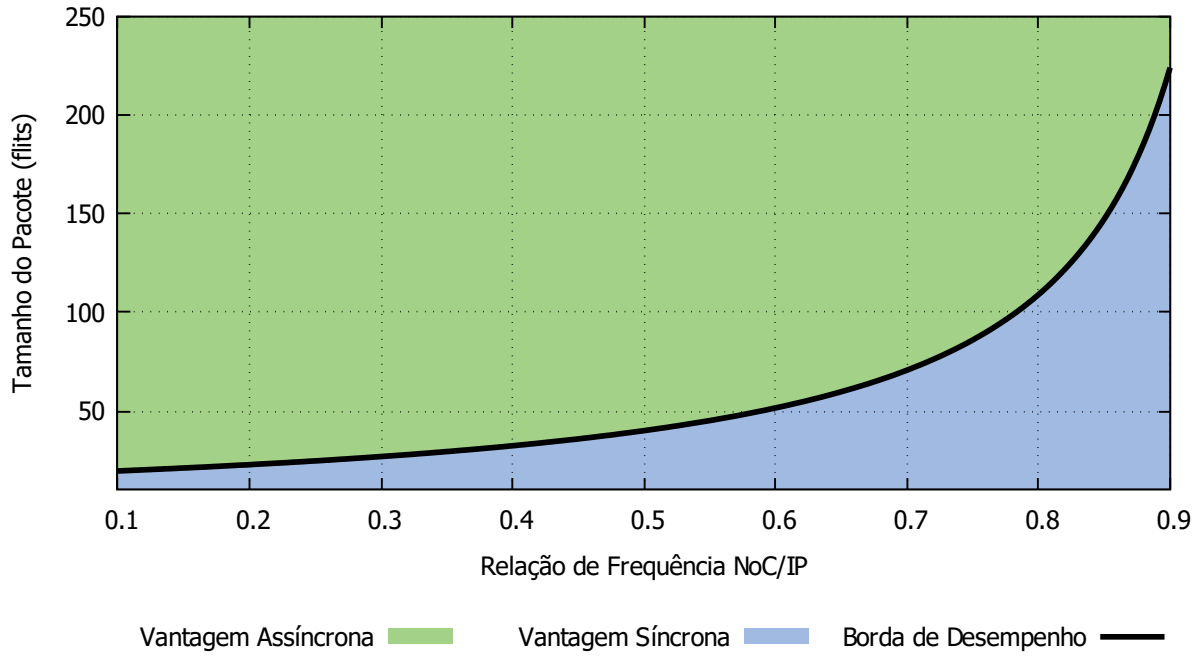


Figura 36 – Borda de desempenho entre o modo síncrono utilizando NI de fila bissíncrona e o modo assíncrono utilizando NI de fila circular assíncrona.

para que a comunicação assíncrona apresente desempenho igual ao do modo síncrono seria de aproximadamente 40 *flits*. Para transmissões que excedam esse valor mínimo de quarenta *flits* o modo de comunicação assíncrono apresentará vantagem no desempenho.

5 EXPERIMENTOS E RESULTADOS

Este capítulo tem por objetivo demonstrar o impacto do uso da comunicação assíncrona, para tal são realizados experimentos em dois cenários. Os experimentos comparam a comunicação assíncrona, que foi apresentada neste trabalho, com a comunicação síncrona, que é amplamente utilizada e bem definida na literatura. Desta forma, o capítulo está dividido em três seções que cobrem as seguintes características, a área dos circuitos, o seu desempenho na comunicação e o consumo energético.

5.1 ÁREA

Todas as sínteses apresentadas nesta seção foram feitas utilizando as ferramentas da CADENCE disponibilizadas pelo Grupo de Microeletrônica (GMICRO) da UFSM. As sínteses utilizaram uma biblioteca de células padrão da IBM de 180 nm, também disponibilizada pelo GMICRO.

Para avaliarmos o impacto na área relativo ao modo de transmissão assíncrono da Arke, tomamos uma rede com as mesmas características porém sem o suporte ao modo assíncrono como NoC referência. Foi realizada a síntese lógica em ambas as NoCs (Arke e NoC referência) e os resultados podem ser observados na Tabela 2.

Tabela 2 – Área do Roteador, IBM 180nm. EG - *equivalent gate* (NAND2).

Módulo	**NoC referência** (μm^2)	**Arke** (μm^2)
Roteador - *buffer* 2 posições	59.019 (2.566 EG)	81.964 (3.564 EG)
Roteador - *buffer* 4 posições	86.025 (3.740 EG)	114.836 (4.993 EG)

Foram feitas sínteses para rotadores com largura de *flit* fixa em 16 bits e tamanhos de *buffers* de duas e quatro posições. Isto porque como visto no Capítulo 3, sabe-se que a comunicação assíncrona não se beneficia dos *buffers*, pois é necessário apenas o espaço para realizar o armazenamento temporário do cabeçalho durante a fase de estabelecimento. Nestas avaliações o roteador da Arke apresentou aumento de área de de 38.88% ao utilizar *buffers* de duas posições, enquanto que o com quatro posições o aumento foi de 33.49%. O aumento da área do roteador ocorre devido à necessidade de adicionar lógica de controle para o modo assíncrono e circuitos anti-*hazard* no roteador da Arke. Todavia, por se tratar de uma NoC projetada para trabalhar em um sistema GALS, tem-se na NI o suporte a comunicações entre distintos domí-

nios de *clock*. Desta forma, também é necessário levar em consideração a NI, cujos resultados de área são apresentados na Tabela 3.

Tabela 3 – Área da NI, IBM 180nm. EG - *equivalent gate* (NAND2).

Módulo	**Área** (μm^2)
NI Fila Bissíncrona	31.190 (1.356 EG)
NI Fila Circular Assíncrona	27.207 (1.183 EG)

Levando em consideração que cada roteador está acoplado a uma NI que é responsável por realizar a adaptação do domínio de *clock* e do protocolo de comunicação, pode-se observar o aumento considerando o par roteador/NI. Utilizando os roteadores Arke e a NI de fila circular assíncrona o aumento da área é de cerca de 21% quando comparado com os roteadores linha de base combinados com a NI de fila bissíncrona.

5.2 DESEMPENHO

Além das avaliações de área, alguns cenários foram propostos para avaliar o impacto no desempenho. Nesta seção serão comparados dois pares operacionais, o primeiro é composto pela Arke operando em modo síncrono juntamente com a NI de fila bissíncrona e, em segundo lugar, a Arke operando em modo assíncrono com a NI de fila circular assíncrona. Não será considerado o par composto pela Arke operando de forma assíncrona com a NI de fila bissíncrona, pois, como visto no Capítulo 4, esse conjunto sempre está em desvantagem em relação ao primeiro. Desta forma, fica definido que a comunicação assíncrona implica na utilização da NI de fila circular assíncrona e a comunicação síncrona implica na utilização da NI de fila bissíncrona.

Inicialmente, avaliamos o impacto na latência de transmissão de pacotes em um cenário onde a frequência dos IPs é mantida constante e a frequência da NoC diminui. Esse cenário é justificado pela constante necessidade de diminuir o consumo de elementos do sistema de forma a atender o limite de utilização, para atender essas demanda, uma técnica utilizada é a *dynamic voltage and frequency scaling* - (DVFS), onde certos elementos do sistema têm sua frequência e tensão reduzidos em tempo de execução de forma a reduzir seu consumo. Para tal, mediu-se o tempo de transmissão transmissão de pacotes (de 8 a 512 *flits*) utilizando o modo de comunicação síncrono com NoC e IPs operando a uma frequência de 50 MHz. Posteriormente, alterou-se o modo de comunicação para assíncrono e mediu-se o tempo de transmissão dos pacotes com

a NoC operando a 50 MHz, 25 MHz, 20 MHz, 12.5 MHz e 10 MHz, que respectivamente correspondem a razão de frequência NoC/IP de 1.00 (IPs a 1 vez *clock* NoC), 0.50 (IPs a 2 vezes *clock* NoC), 0.40 (IPs a 2.5 vezes *clock* NoC), 0.25 (IPs a 4 vezes *clock* NoC) e 0.20 (IPs a 5 vezes *clock* NoC). Neste cenário há apenas um IP transmissor e um receptor, ambos operando com capacidade de transmitir e receber um *flit* por ciclo de *clock*, o cenário está ilustrado na Figura 37. Os resultados podem ser observados na Figura 38.

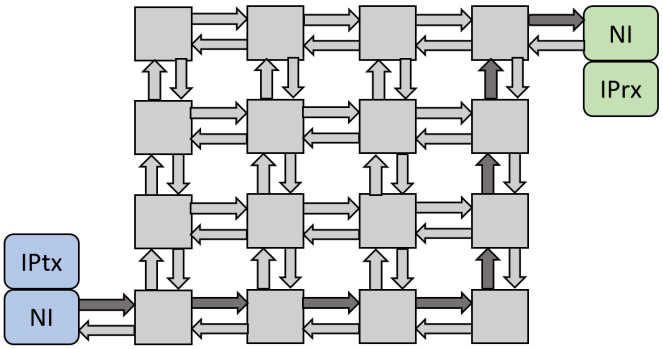


Figura 37 – Configuração do primeiro cenário, onde avalia-se o impacto na latência devido ao decréscimo da frequência da NoC.

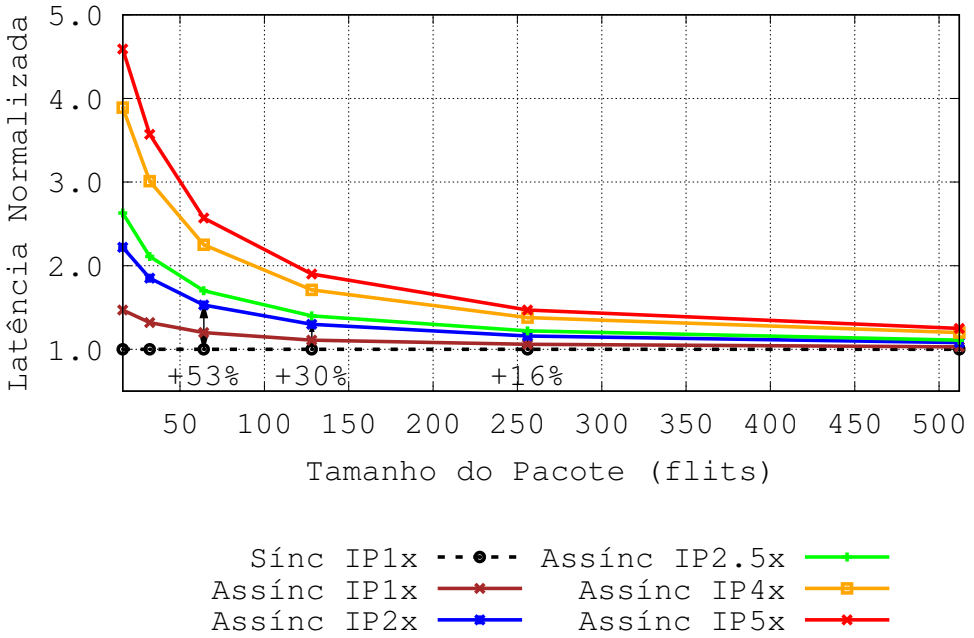


Figura 38 – Desempenho da comunicação síncrona e assíncrona, variando o tamanho do pacote e a frequência. Resultados normalizados em relação Arke operando em modo síncrono com a NI de fila bissíncrona.

A latência de transmissão dos pacotes foi normalizada em relação a latência da transmissão síncrona, representada pela linha preta pontilhada. Cada uma das outras linhas representa uma transmissão assíncrona ocorrida na NoC operando a uma frequência inferior aos

IPs. Observe que em transmissões de pacotes pequenos o impacto é maior. A latência têm um comportamento assintótico em relação ao tamanho do pacote. Para pacotes menores o impacto do estabelecimento/encerramento da conexão assíncrona com a NoC operando em frequências baixas, tem maior relevância do que para pacotes grandes. Por exemplo, quando a Arke está operando na metade da frequência dos IPs (*IP2x*), o aumento da latência varia de 53%, 30% e 16% para pacotes de 64, 128 e 256 *flits* respectivamente. Resumindo, o impacto na latência de transmissão da Arke operando na metade de sua frequência nominal (IP2x) varia de 55% à 10%, dependendo do tamanho do pacote. O comportamento assintótico é explicado devido ao fato de que quanto maior for o pacote, menor é o impacto do estabelecimento/encerramento da conexão na latência, visto que este processo ocorre na frequência da NoC. Uma vez estabelecida a conexão, a transmissão ocorre na freqência dos IPs. O modo assíncrono apresenta vantagem perante o modo síncrono quando a NoC opera em uma frequência mais baixa que os IPs, pois se fosse utilizado o modo síncrono com a NoC operando na metade de sua frequência nominal, o impacto na latência seria de 100%.

O segundo experimento é semelhante ao primeiro, no entanto os resultados não foram normalizados em relação a transmissão síncrona. Neste cenário um IP fonte envia um pacote pela NoC em direção a um IP destino (de 16 à 512 *flits*) utilizando os dois modos de comunicação. Para realizar este experimento, fixou-se a frequência da NoC em 20 MHz enquanto utilizou-se pares de IPs comunicantes com frequências de 20 MHz, 40 MHz, 60 MHz, 80 MHz e 100 MHz que correspondem, respectivamente, a razão de frequência NoC/IP de 1.00 (IP1x), 0.50 (IP2x), 0.67 (IP3x), 0.25 (IP4x) e 0.20 (IP5x). Veja na Figura 39 os resultados.

Cada linha da Figura 39 representa a latência da transmissão dos pacotes para um par de IPs comunicantes em uma frequência diferente. É importante lembrar que a comunicação síncrona utiliza a NI de fila bissíncrona enquanto que a comunicação assíncrona faz uso da NI de fila circular assíncrona. Como esperado, a latência da comunicação em modo síncrono (linha pontilhada) não varia com o aumento da frequência dos IPs, pois a NoC operando em modo síncrono torna-se um gargalo transmitindo um *flit* por ciclo na sua frequência de operação. Além disso, a latência da comunicação assíncrona de IPs que operam na mesma frequência da NoC (IP1x Asínc) é sempre maior devido às fases de estabelecimento e encerramento do modo assíncrono, as quais geram sobrecarga com relação ao modo síncrono. Como pode ser visto no gráfico, essa diferença é constante, e para sendo neste experimento 18 ciclos de *clock* da NoC. Contudo, conforme a frequência do par de IPs comunicantes aumenta, a sobrecarga na

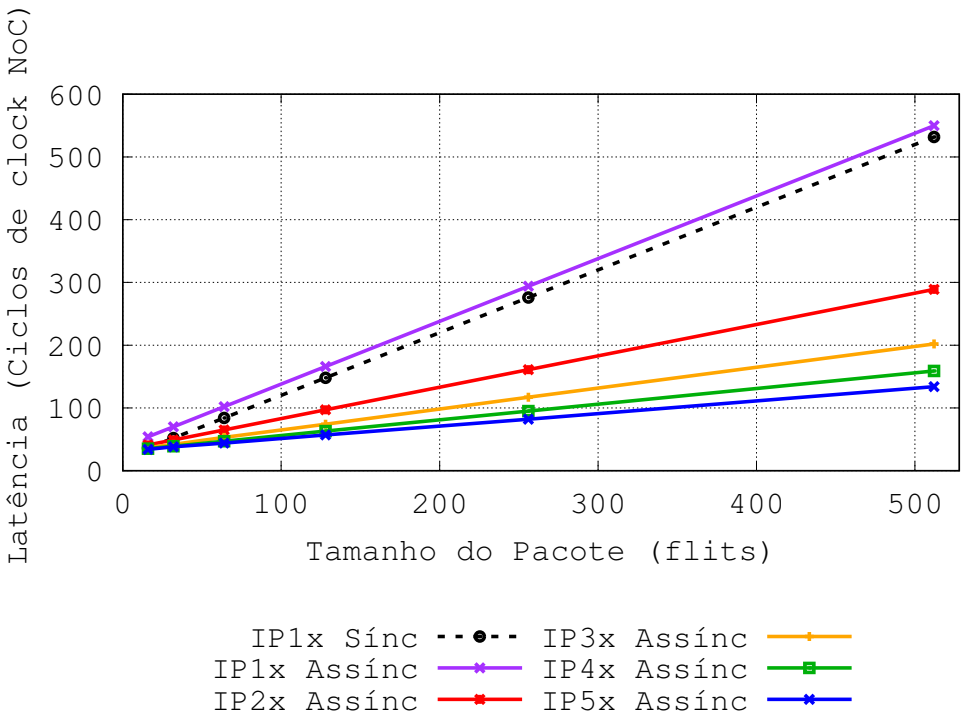


Figura 39 – Desempenho da comunicação síncrona e assíncrona na transmissão de pacotes individuais (até 512 *flits)*, variando-se a frequência dos IPs em relação à da NoC.

latência devido ao gerenciamento da conexão é amortizada e a comunicação assíncrona supera a síncrona. A comunicação assíncrona apresenta maior latência apenas para pacotes pequenos. Para analisar melhor a intersecção entre as comunicações síncrona e assíncrona observe a Figura 40. Resumindo, observa-se que o tempo total de transmissão aumenta linearmente em função do tamanho dos pacotes, enquanto diminui assintoticamente em relação ao aumento do *clock* dos IPs. Esse comportamento é descrito pela Equação 4.10 no Capítulo 4.

Em um segundo cenário, é realizada a terceira avaliação de desempenho, onde ambos os modos de comunicação são comparados com vários IPs comunicantes. O cenário consiste em uma NoC 4x4 que interconecta 16 IPs, como mostra a Figura 41. Cada IP injeta 200 pacotes, totalizando 3200 pacotes. O tamanho de cada pacote varia de 18 a 512 *flits* e é enviado para um destinatário aleatório. A frequência da NoC é mantida constante enquanto que para cada rodada de simulação, os IPs são substituídos por IPs mais rápidos, como no cenário anterior, cobrindo as frequências de 1 até 5 vezes a da NoC. É computado o tempo total utilizado para realizar o envio de todos os pacotes, medido em ciclos de *clock* da NoC. A Figura 42 apresenta os resultados do experimento.

A linha pontilhada representa o desempenho da comunicação síncrona. Como esperado, o tempo total para transmissão dos pacotes não é impactado pela frequência dos IPs no modo síncrono. Por outro lado, as transmissões assíncronas levam mais tempo para serem finalizadas apenas quando a frequência dos IPs é igual à da NoC, principalmente pelo fato da sobrecarga

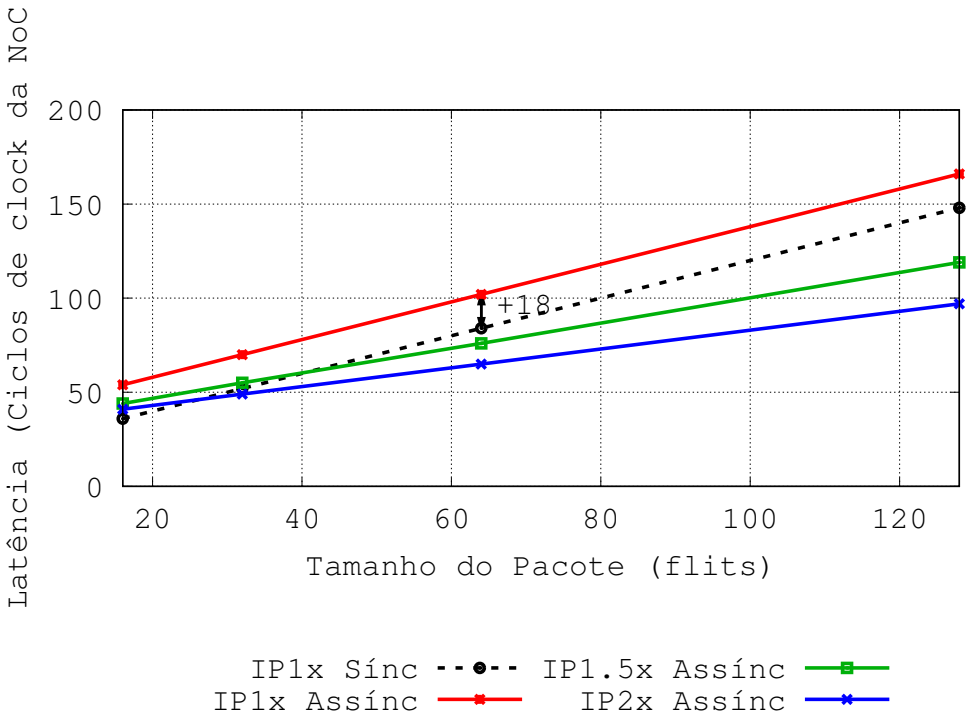


Figura 40 – Desempenho da comunicação síncrona e assíncrona na transmissão de pacotes individuais (até 128 *flits*), variando-se a frequência dos IPs em relação à da NoC.

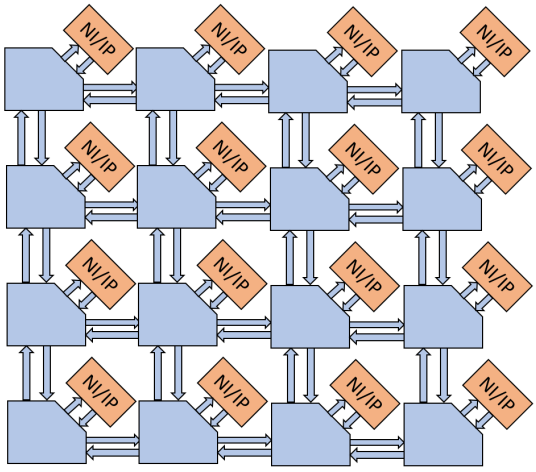


Figura 41 – Configuração do segundo cenário, onde avalia-se o impacto na latência devido ao decréscimo da frequência da NoC

gerada pelo gerenciamento da conexão do modo assíncrono. Vale ressaltar que para cada pacote transmitido, uma nova conexão é aberta e fechada. O desempenho é melhor ao utilizar transmissões em modo rajada, mantendo a conexão ativa para enviar vários pacotes. Observe que para IPs operando ao dobro da frequência da NoC a transmissão síncrona leva 1.77 vezes mais tempo que a assíncrona, esse número chega a 3.73 vezes quando os IPs estão operando a 5 vezes a frequência da NoC.

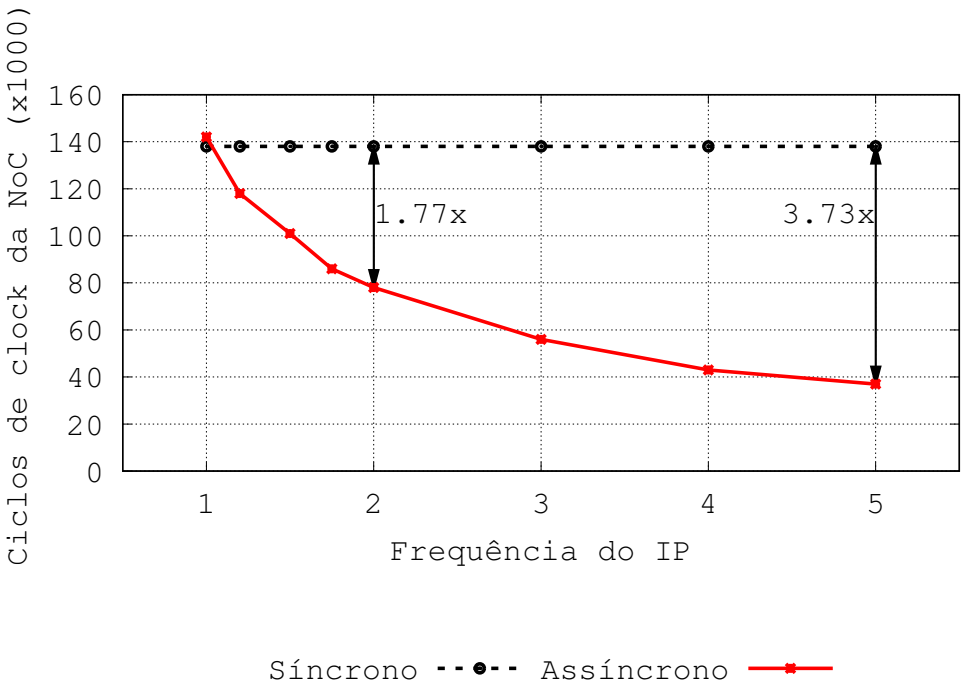


Figura 42 – Desempenho da comunicação síncrona e assíncrona na transmissão de um conjunto de pacotes.

5.3 ENERGIA

O próximo experimento avalia a energia consumida no mesmo cenário do experimento anterior, Figura 41, formado por uma NoC 4x4, com 16 IPs enviando pacotes (16 a 512 *flits*) para destinos aleatórios, com IPs operando na frequência da NoC e em frequências superiores (até 4.5 vezes). Os valores de energia foram obtidos através da leitura do arquivo de chaveamento gerado após a simulação das transmissões dos pacotes. As simulações foram conduzidas com uma instância da NoC Arke sintetizada e com os atrasos lógicos. Veja na Figura 43 os resultados.

A linha pontilhada preta corresponde a Arke operando em modo síncrono com *Input Buffers* contendo quatro posições. Neste caso a energia consumida pela NoC não muda mesmo após a substituição dos IPs lentos pelos IPs rápidos. Isso ocorre pois as transmissões são realizadas no domínio de *clock* da NoC, independente da frequência dos IPs. Utilizando a comunicação assíncrona apresentam-se dois resultados, referentes a roteadores com *Input Buffers* de duas e de quatro posições. A energia consumida pela NoC diminui com a redução do tamanho do *Input Buffer* e com o aumento da frequência dos IPs em relação à NoC. A redução da profundidade dos *Input Buffers* não afeta o desempenho, pois durante a transmissão eles sofrem *bypass* e durante o estabelecimento só é necessário espaço apenas para o armazenamento temporário do cabeçalho. IPs com frequência superior à da NoC permite que a transmissão ocorra

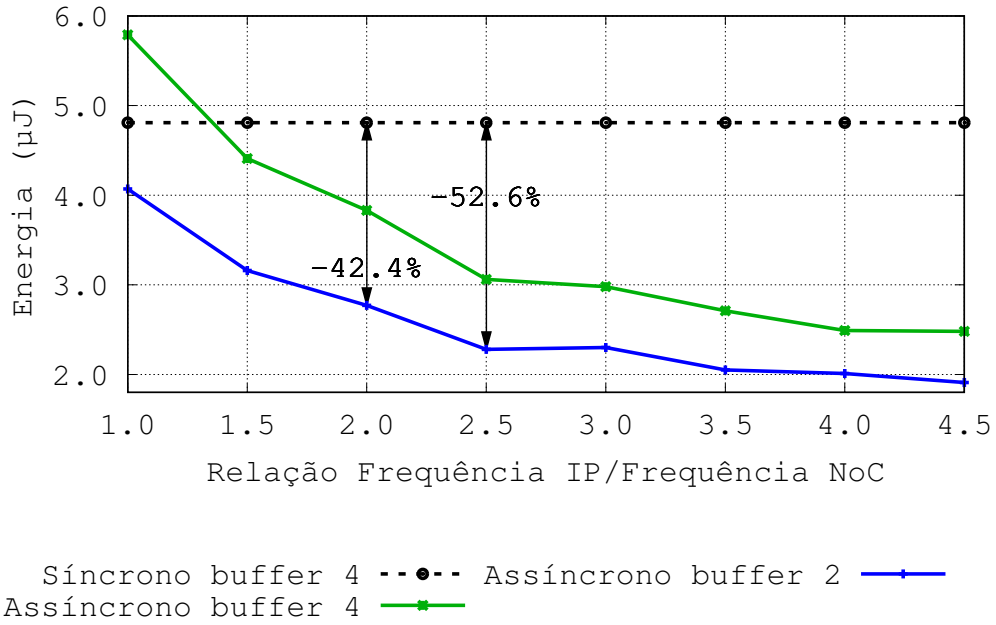


Figura 43 – Consumo de energia na transmissão de pacotes utilizando modo de comunicação síncrono e assíncrono, variando a frequência dos IPs em relação a NoC.

em menos tempo e por consequência reduz o consumo de energia para enviar os pacotes, dado que a energia é o produto da potência que o circuito gasta pelo tempo que ele está funcionando, além de diminuir a atividade de chaveamento dos *Input Buffers* em *bypass*. O roteador com *buffers* de duas posições apresenta redução de 42.4% e 52.6% considerando IPs operando a 2 e 2.5 vezes a frequência da NoC.

6 CONCLUSÃO

A ideia deste trabalho originou-se de duas necessidades em SoCs, remover o gargalo criado pela NoC em sistemas GALS e reduzir o consumo de energia da NoC, que varia entre 10% (Cheng et al., 2015) a 30% (Hoskote et al., 2007) do consumo do SoC. Inicialmente implementou-se a técnica de *bypass* nos *Input Buffers* na NoC Arke, visando reduzir o consumo de energia devido ao fato de que os *Input Buffers* não realizam armazenamentos enquanto estão sob *bypass*. Uma vez que o *bypass* foi implementado, observou-se que o aumento na latência da comunicação seria impeditivo quando combinado com a NI de fila bissíncrona, devido às constantes sincronizações necessárias para evitar a metaestabilidade durante a troca de domínios de *clock*. Para resolver este problema, foi necessário utilizar outra técnica para transferência de domínio de *clock*, pois a fila bissíncrona gerava um gargalo. Desta forma, optou-se pela utilização de sincronização de borda, que quando combinado com uma fila circular assíncrona, fornece a possibilidade de paralelizar as sincronizações de cada uma das posições da fila, permitindo a transmissão de sucessivos *flits* enquanto houverem posições livres na fila. A fila circular assíncrona quando combinada com a capacidade da comunicação assíncrona da Arke, permite alcançar vantagens de desempenho aliadas a redução do consumo do circuito. Desta forma, neste trabalho foi apresentada a NoC Arke, bem como duas NI que fazem adaptação de protocolo e auxiliam na adaptação do domínio de *clock*, oferecendo uma infraestrutura de comunicação eficiente para sistemas GALS.

Buscou-se apresentar de forma detalhada os processos de operação dos modos de comunicação síncrono e assíncrono implementados pela Arke, apresentando detalhadamente a arquitetura de todas as estruturas que compõe a NoC. Além da NoC foi apresentada a arquitetura das NI de fila bissíncrona e de fila circular assíncrona. Além disso, foram apresentadas equações que permitem ao usuário determinar qual modo de operação irá fornecer menor latência dependendo da situação (tamanho do pacote, frequência dos IPs e da NoC e distância da comunicação). A Arke e as NI foram descritas em VHDL parametrizável e estão disponíveis para *download* em: *www.github.com/iacanaw/Arke*.

Avaliou-se três aspectos da NoC, área, desempenho e consumo de energia. O aumento da área do roteador estava previsto, uma vez que foi adicionado um novo modo de comunicação na rede. A Arke dá suporte simultaneamente a ambos os modos de comunicação, síncrono e assíncrono, ao contrário da NoC referência que suporta apenas o modo síncrono. Em relação

ao desempenho, a comunicação síncrona é a melhor opção para a comunicação entre IPs que estejam operando na mesma frequência que a NoC. Porém, caso os IPs estejam operando em uma frequência superior, o modo assíncrono sempre poderá superar o modo síncrono, desde que o tamanho do pacote seja suficientemente grande. A redução no consumo de energia pode alcançar mais de 50%, se os IPs estiverem operando em frequências maiores que a da NoC.

6.1 TRABALHOS FUTUROS

A Arke pode se beneficiar de diversas melhorias, dentre elas destacam-se as técnicas de *power* e *clock gating*. Os *buffers* internos dos *Input Buffers* podem ser desligados durante a comunicação assíncrona, pois eles sofrem *bypass* e não são utilizados durante a fase de transmissão. Estas tecnicas poderiam aumentar a vantagem energética da comunicação assíncrona, seguindo uma tendência de *dark silicon*. Ademais, uma avaliação no nível de síntese física poderia trazer resultados mais verossimilhantes à realidade, uma vez que todas as análises neste trabalho foram realizadas considerando apenas os atrasos oriundos da síntese lógica, ou seja, sem levar em consideração os atrasos dos fios.

Atualmente existe um trabalho em andamento com NIs hibridas, que deem suporte à comunicação entre IPs assíncronos, o que torna a Arke uma alternativa à NoCs assíncronas, além de uma NI capaz de realizar comunicação entre IPs síncronos (na sua ilha síncrona) e IPs assíncronos. Uma NI hibrida poderia implementar em seu controle uma análise para determinar qual é o modo que maximiza o desempenho do sistema para um determinado tamanho de pacote pacote, destino e frequências de operação dos IPs envolvidos.

REFERÊNCIAS

Abdallah, A. B. (2013). *Multicore Systems On-Chip: Practical Software/Hardware Design*. Atlantis Press, Japão. Seccond Edition.

Beigne, E. and Vivet, P. (2006). Design of on-chip and off-chip interfaces for a GALS NoC architecture. In *Proc. 12th IEEE Int'l Symp. Asynchronous Circuits and Systems (ASYNC 06)*, pages 172–181.

Benini, L. and Micheli, G. D. (2002). Networks on chip: A new paradigm for systems on chip design. France. Design, Automation and Test in Europe Conference and Exhibition, IEEE.

Bjerregaard, T., Mahadevan, S., Olsen, R., and Sparso, J. (2005). An OCP Compliant Network Adapter for GALS-based SoC Design Using the MANGO Network-on-Chip. In *Proc. Int'l Symp. System-on-Chip (SoC 05)*, pages 171–174.

Chapiro, D. M. (1984). *Globally-asynchronous locally-synchronous systems*. Doutorado em computação, Stanford University, Stanford, CA, USA.

Chelcea, T. and Nowick, S. (2000). Low-latency asynchronous FIFO's using token rings. In *Proc. 6th Int'l Symp. Advanced Research in Asynchronous Circuits and Systems (ASYNC 00)*, pages 210–220.

Chen, C.-H. O., Krishna, T., Kwon, W.-C., and Peh, L.-S. (2014). Smart: Single-cycle multihop traversals over a shared network on chip. *IEEE Micro*, 34(3):43–56.

Chen, C.-H. O., Park, S., Krishna, T., Subramanian, S., Chandrakasan, A. P., and Peh, L.-S. (2013). Smart: A single-cycle reconfigurable noc for soc applications. France. Design, Automation & Test in Europe Conference & Exhibition, IEEE.

Chen, X. and Jha, N. K. (2016). Reducing wire and energy overheads of the smart noc using a setup request network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(10):3013–3026.

Cheng, H.-Y., Zhan, J., Zhao, J., Xie, Y., Sampson, J., and Irwin, M. J. (2015). Core vs. uncore: The heart of darkness. In *DAC*.

Dally, W. and Towles, B. (2003). *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, Burlington, Massachusetts. First Edition.

Dobkin, R., Ginosar, R., and Sotiriu, C. (2004). Data Synchronization Issues in GALS SoCs. In *Proc. 10th IEEE Int'l Symp. Asynchronous Circuits and Systems (ASYNC 04)*, pages 170–179.

Feero, B. and Pande, P. P. (2007). Performance evaluation for three-dimensional networks-on-chip. Brazil. Computer Society Annual Symposium on VLSI, IEEE.

Guerrier, P. and Greiner, A. (2000). A generic architecture for on-chip packet-switched interconections. France. Design, Automation and Test in Europe Conference and Exhibition, IEEE.

Gupta, R. K. and Zorian, Y. (1997). Introducing core-based system design. *IEEE Design & Test of Computers*, 14:15–25.

Hoskote, Y., Vangal, S., Singh, A., Borkar, N., and Borkar, S. (2007). A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro*, 27(5):51–61.

Jackson, C. and Hollis, S. J. (2010). Skip-links: A dynamically reconfiguring topology for energy-efficient nocs. Finland. International Symposium on System on Chip, IEEE.

Jain, T. N. K., Gratz, P. V., Sprintson, A., and Choi, G. (2010). Asynchronous bypass channels: Improving performance for multi-synchronous nocs. France. International Symposium on Networks-on-Chip, IEEE.

Kessels, J., Peeters, A., Wielage, P., and Kim, S.-J. (2003). Clock synchronization through handshake signalling. *Microprocessors and Microsystems*, pages 447–460.

Krishna, T., Kumar, A., Chiang, P., Erez, M., and Peh, L.-S. (2008). Noc with near-ideal express virtual channels using global-line communication. USA. Symposium on High Performance Interconnects, IEEE.

Krstic, M., Grass, E., Gürkaynak, F. K., and Vivet, P. (2007). Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design & Test of Computers*, 24(5):430–441.

Krstic, M., Grass, E., Stahl, C., and Piz, M. (2006). System integration by request-driven gals design. *IEE Proceedings - Computers and Digital Techniques*, 153(5):362–372.

Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Oberg, J., K., T., and Hemani, A. (2002). A network on chip architecture and design methodology. USA. Computer Society Annual Symposium on VLSI, IEEE.

Martin, G. and Chang, H. (2001). System on chip design. China. International Conference on ASIC, IEEE.

Modarressi, M., Sarbazi-Azad, H., and Tavakkol, A. (2008). Virtual point-to-point links in packet-switched nocs. France. IEEE Computer Society Annual Symposium on VLSI, IEEE.

Moore, G. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8):114.

Moore, G. E. (2003). No exponential is forever: but "forever" can be delayed! [semiconductor industry]. USA. International Solid-State Circuits Conference, IEEE.

Muttersbach, J., Villiger, T., and Fichtner, W. (2000). Practical design of globally-asynchronous locally-synchronous systems. Israel. International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE.

Noghondar, A. F. and Reshadi, M. (2015). A low-cost and latency bypass channel-based on-chip network. *The Journal of Supercomputing*, 71(10):3770–3786.

Panades, I. M. and Greiner, A. (2007). Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures. In *First International Symposium on Networks-on-Chip (NOCS'07)*.

Rupp, K. (2018). 42 years of microprocessor trend data. Acesso em 17 set. 2018.

Singh, M. and Nowick, S. M. (2007). MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 684–698.

Stensgaard, M. B. and Sparsø, J. (2008). Renoc: A network-on-chip architecture with reconfigurable topology. UK. International Symposium on Networks-on-Chip, IEEE.

Texas, I. I. (2000). Skew definition and jitter analysis. *Analog Applications Journal*, pages 29–32.

Weber, I. I., Oliveira, L. L. d., Moraes, F. G., and Carara, E. A. (2018). Exploring asynchronous end-to-end communication through a synchronous NoC. In *SBCCI*.

Yun, K. Y. and Donohue, R. P. (1996). Pausible clocking: A first step toward heterogeneous systems. USA. International Conference on Computer Design: VLSI in Computers and Processors, IEEE.