UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DÉBORA DA SILVA MOTTA MATOS

# Interfaces Parametrizáveis para Aplicações Interconectadas por uma Rede-em-Chip

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Altamiro Amadeu Susin
Orientador

Porto Alegre, abril de 2010.

**CIP – CATALOGAÇÃO NA PUBLICAÇÃO**

Matos, Débora da Silva Motta

Interfaces Parametrizáveis para Aplicações Interconectadas por uma Rede-em-Chip/

Débora da Silva Motta Matos – Porto Alegre: Programa de Pós-Graduação em Computação, 2010.

143 p.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2010. Orientador: Altamiro Amadeu Susin.

1. Interfaces de Rede. 2. Redes-em-Chip 3. Decodificador de Vídeo H.264 4. Sistemas em Chip. I. Susin, Altamiro Amadeu. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Aldo Bolten Lucion
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenador do PPGC: Prof. Álvaro Freitas Moreira
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

# AGRADECIMENTOS

Como não podia deixar de ser, inicio meus agradecimentos à pessoa mais especial que já conheci, que está sempre ao meu lado, me incentivando e me motivando a dar o próximo passo: meu amado esposo Cristiano. A ti um agradecimento especial, pela tua eterna compreensão em todos os momentos, pelo teu carinho e por aceitar e entender a minha "falta" de finais de semana e feriados, que se tornaram uma constância nos últimos meses. Prometo que no doutorado vai ser pior, mas que vou procurar não torna-lo tão desgastante para nós. Obrigada por fazer parte da minha vida!

Agradeço Àquele que está presente em todos os momentos da minha vida, guiando minhas atitudes e me inspirando a encontrar soluções aos meus problemas ou, até mesmo, a encontrar problemas que valorizam a continuação do meu trabalho: Deus.

Ao meu orientador, Prof. Altamiro Susin, pelo seu profissionalismo e amizade. Obrigada pela contribuição no desenvolvimento do trabalho e por sempre estar disponível para um conversa ou um auxílio às minhas dúvidas.

Ao meu amigo e professor, Luigi Carro, que inúmeras vezes têm me ajudado com ideias de trabalho, fazendo com que elas sempre fossem ao encontro dos meus objetivos acadêmicos e profissionais, mesmo não sendo meu co-orientador no mestrado. Obrigada pela revisão de trabalhos e artigos, pelas várias contribuições e dicas e ainda por me instigar a pensar como uma cientista.

À minha mãe, Dulce, que é uma das responsáveis pela pessoa que sou e pelos valores e princípios que trago comigo, que sempre esteve disposta a me ajudar e compreender os meus momentos de ausência. Agradeço também a minha sogra, pela compreensão e por sempre se mostrar disposta a me ajudar com as demais tarefas para que me sobrasse mais tempo no desenvolvimento das minhas atividades acadêmicas.

Um agradecimento especial ao meu amigo Cláudio, pelas incontáveis dicas, ideias e experiências repassadas durante todo o período de mestrado.

Um forte agradecimento a minha amiga e colega Caroline Concatto, com quem realizei diversos trabalhos e publicações e que sempre esteve disposta a trocar uma ideia sobre o desenvolvimento do meu trabalho ou somente para jogar conversa fora. Agradeço também ao meu amigo Márcio Kreutz, que mesmo agora distante, sempre esteve por perto para ajudar e contribuir na elaboração de artigos, que junto com outras pessoas, pudemos desenvolver nesse período.

Aos colegas Rafael Leite, Paulo Cesar, Cezar Reinbrecht e Miklécio Costa que trabalharam mais diretamente comigo em assuntos relacionados ao desenvolvimento dessa dissertação. Agradeço a todos os meus colegas do laboratório que ajudaram a descontrair nos momentos em que isso foi preciso, permitindo que, depois de uma conversa divertida, com boas risadas, se pudesse dar continuidade ao trabalho com mais ânimo. Dentre estes estão os colegas Bruno, Thaísa, Vagner, Dieison, Guilherme Côrrea, Zanetti, Fábio Ramos, André (Alemão), Franco, Max, Marcelo e alguns outros que devo ter esquecido agora, mas que não tiveram menos importância nessa fase. Agradeço também a Anelise Kologeski e a Mariza que estiveram sempre por perto para trocar ideias e discutir propostas de trabalho. Agradeço ainda aos colegas Fábio Pereira,

André Borin e Alexsandro Bonatto que me ajudaram na compreensão dos códigos do decodificador de vídeo utilizados no desenvolvimento dessa dissertação.

Aos demais professores do PPGC que contribuíram na minha formação: Fernanda Kastensmidt, Erika Cota, Marcelo Lubaszewski e Sergio Bampi. Obrigada!

Aos demais amigos que continuaram meus amigos mesmo eu não estando ultimamente tão presente e aos demais colegas e parentes que estão sempre torcendo por mim para que eu alcance meus objetivos. A todos os colegas e amigos que formei neste período e a todas as pessoas brilhantes que conheci desde que ingressei nesta Universidade. Um muito obrigada!

Ao CNPq, por financiar meu período de estudos durante o mestrado através de uma bolsa.

A todos, minha sincera gratidão!

# SUMÁRIO

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ASIP | *Application-Specific Instruction Processor* |
| AVC | *Advanced Video Coding* |
| AXI | *Advanced eXtensible Interface* |
| BAD | *Binary Arithmetic Decoding* |
| bop | *begin of-packet* |
| BRAM | *Block RAM* |
| CABAC | *Context-based Adaptive Binary Arithmetic Coding* |
| CABAD | *Context-based Adaptive Binary Arithmetic Decoding* |
| CAVLD | *Context-based Adaptive Variable Length Coding* |
| CAVLD | *Context-based Adaptive Variable Length Decoding* |
| Cb | *Chrominance blue* |
| CIF | *Common Intermediate Format* |
| Cr | *Chrominance red* |
| CS | *Circuit Switching* |
| DAMQ | *Dynamically Allocated Multi-Queue* |
| DC | *Direct Current* |
| DOR | *Dimension Ordered Routing* |
| DSP | *Digital Signal Processing* |
| DTL | *Device Transaction Level* |
| eop | *end-of-packet* |
| EP | Elementos de Processamento |
| FCFS | *First-Come, First-Served* |
| *FIFO* | *First-In, First-Out* |
| FSM | *Finite State Machine* |
| IDCT | *Inverse Discrete Cosine Transform* |
| INTER | Predição Inter-Quadros |
| INTRA | Predição Intra-Quadro |
| IP | *Intellectual Property* |
| IQ | *Inverse Quantization* |

| | |
|---|---|
| IT | *Inverse Transforms* |
| ITIQ | *Inverse Transforms and Inverse Quantization* |
| LRS | *Least Recently Served* |
| LUT | *Look-Up Tables* |
| ME | *Motion Estimation* |
| MPEG | *Motion Picture Experts Group* |
| MPSoC | *Multiprocessor SoC* |
| NAL | *Network Abstraction Layer* |
| NI | *Network Interface* |
| NoC | *Network-on-Chip* |
| OCP | *Open Core Protocol* |
| POC | *Picture Order Count* |
| QCIF | *Quarter Common Intermediate Format* |
| QoS | *Quality of Service* |
| QP | *Quantization Parameter* |
| RAM | *Random Access Memory* |
| RASoC | *Router Architecture for System-on-Chip* |
| RGB | *Red, Green, Blue* |
| RISC | *Reduced Instruction Set Computer* |
| SAF | *Store-and-Forward* |
| SAFC | *Statically Allocated, Fully Connected* |
| SAMQ | *Statically Allocated Multi-Queue* |
| SoC | *System-on-Chip* |
| SoCIN | *System-on-Chip Interconnection Network* |
| TDM | *Time-Division Multiplexing* |
| UFRGS | Universidade Federal do Rio Grande do Sul |
| VC | *Virtual Channels* |
| VCI | *Virtual Components Interface* |
| VCL | *Video Coding Layer* |
| VCT | *Virtual Cut-Through* |
| VGA | *Video Graphics Array* |
| VHDL | *VHSIC Hardware Description Language* |
| VHSI | *Very High Speed Integrated Circuit* |
| VLC | *Variable Length Coding* |
| VLD | *Variable Length Decoding* |

| | |
|---|---|
| VLIW | *Very Long Instruction Word* |
| VLSI | *Very Large Scale Integration* |
| Y | *Luminance* |
| YCbCr | *Luminance, Chrominance blue, Chrominance red* |
| WH | *Wormhole* |

# LISTA DE FIGURAS

# LISTA DE TABELAS

# RESUMO

As redes-em-chip (NoCs) surgiram como uma alternativa aos atuais problemas de interconexão decorrentes da redução da escala de tecnologia de fabricação de circuitos integrados. O desenvolvimento de transistores com nanômetros de largura tem permitido a inserção de sistemas altamente complexos em uma única pastilha de silício. Dessa forma, os SoCs (Systems-on-Chip) passaram a constituir inúmeros elementos de processamentos (EPs) e as NoCs têm se apresentado como uma opção eficiente no provimento da interconexão dos mesmos, permitindo maior escalabilidade e paralelismo ao sistema. No entanto, esta conexão não é realizada de forma direta. Todo sistema conectado por uma NoC necessita de interfaces de rede (NIs) para intermediar a conexão dos elementos de processamento aos roteadores da rede.

O objetivo desse trabalho é apresentar soluções arquiteturais de interfaces de rede para NoCs que atendam diferentes aplicações de forma genérica. Neste trabalho foram desenvolvidas interfaces de redes reutilizáveis e parametrizáveis, e para atender a estas características, as interfaces de rede possibilitam a configuração de diversos parâmetros arquiteturais, como largura da palavra de dados dos EPs, profundidade das FIFOs das interfaces, profundidade das FIFOs da NoC e largura de dados da rede, possibilitando prover a interconexão de qualquer aplicação com um mínimo de reprojeto. As interfaces de rede, juntamente com a NoC, são responsáveis pelo desempenho da comunicação da aplicação e, para isso, o projeto de uma NI deve ser capaz de atender aos requisitos do sistema, por isso, a importância de se obter um projeto de NIs flexível.

Para validar as arquiteturas das NIs desenvolvidas, os módulos do decodificador de vídeo no contexto do padrão H.264 foram conectados à NoC através das interfaces projetadas. A partir dessa implementação, puderam-se levantar diversas necessidades que devem ser atendidas pelas NIs. Por fim, foram analisados os resultados de síntese das NIs para diferentes configurações. Também foram verificados os resultados de síntese e desempenho do decodificador de vídeo H.264 conectado pelas NIs à NoC. Com relação aos resultados de síntese em FPGA, a implementação do decodificador de vídeo com NoC e NIs não apresentou um grande aumento em área quando comparada a implementação com conexão ponto-a-ponto. Além disso, para diferentes configurações das NIs, a NoC pode ser utilizada atendendo aos requisitos de desempenho exigidos pela aplicação, sem a necessidade de operar na sua máxima taxa de operação para a resolução QCIF.

**Palavras-Chave:** Interfaces de rede, redes-em-chip, decodificador de vídeo H.264, soluções de interconexão, elementos de processamento, sistemas em chip.

**Configurable Interfaces for Applications Interconnected by a Network-on-Chip**

# ABSTRACT

Networks-on-Chip (NoCs) have emerged as an alternative to the current interconnection problems arising from the scaling technology for manufacturing integrated circuits. The development of transistors with nanometer-wide has enabled the integration of highly complex systems on a single silicon wafer. Thus, SoCs (Systems-on-Chip) have integrated numerous processing elements (EPs) and NoCs have been presented as an effective option in providing the interconnection of these elements, allowing scalability and parallelism to the system. However, this connection is not done directly. Every system connected by NoC needs network interfaces to intermediate the connection of processing elements to network routers.

The goal of this thesis is to present architectural solutions for network interfaces for applications in general. In this work we developed a generic, reusable and configurable network interface. The proposed network interface enables the configuration of several architectural parameters, such as data width of the packets, FIFOs depth of the interfaces, FIFO depth and data width of the NoC, and thus, being able to provide the interconnection of any application with a minimal redesign. Network interfaces, together with the NoC, are responsible for application performance and, therefore, the design of an NI should be able to support the system requirements.

To validate the architecture of developed NI, the modules of H.264 decoder were connected to NoC through designed interface. From this implementation, one could raise several needs that must be supported by the NIs. Finally, we analyzed the results of synthesis of the NIs for different configurations. It was also analyzed the results of synthesis and performance of H.264 video decoder connected by NIs to NoC. According to results for FPGA synthesis, the implementation of video decoder with NoC and NIs did not show a large increase in area when compared with the implementation of peer-to-peer. Moreover, for different configurations, the NoC can be used according to time requisitions required by the application, without the need to operate at its maximum operation frequency for QCIF resolution.

**Keywords:** Network interfaces, network-on-chip, H.264 video decoder, interconnection solutions, processing elements, system-on-chip.

# 1 INTRODUÇÃO

O desenvolvimento de transistores com nanômetros de largura de canal, o aumento na complexidade de sistemas microeletrônicos e o advento de novas tecnologias de VLSI (*Very Large Scale Integration*) possibilitou o surgimento dos SoCs (*System-on-Chips*), que proporcionaram a integração de bilhões de transistores em um único chip (BENINI, 2001), (KUMAR, 2002). SoCs permitem agrupar em uma única pastilha de silício sistemas computacionais completos, com um número maior de funcionalidades a um elevado nível de complexidade (BENINI, 2002). Os MPSoCs (Multiprocessor SoCs) podem conter diversos tipos de dispositivos, como processadores RISC, VLIW, DSP, ASIP ou ainda hardwares dedicados (IPs - *Intellectual Property*), que genericamente, podem ser chamados também de elementos de processamento (EPs). No entanto, pesquisadores têm se preocupado em obter soluções eficientes para a interconexão entre os módulos destes sistemas. Até então, apesar de suas conhecidas limitações, vinha-se fazendo o uso de barramentos para estas interconexões. Os barramentos já foram alvo de inúmeras pesquisas e várias soluções e melhorias foram propostas para aumento de desempenho dos mesmos (LAHTINEN, 2003), (RYU, 2001), (AMBA, 1999), (LEE, 2004). Porém, soluções tradicionais de interconexão apresentam limites físicos que comprometem a escalabilidade, o desempenho, além de apresentarem um elevado consumo de potência, dentre outros fatores, quando utilizados para os atuais projetos de SoCs ou MPSoCs.

Quando se faz o uso de um barramento, à medida que mais módulos são conectados a ele, o desempenho do sistema como um todo é degradado (REGO, 2006), já que a largura de banda do barramento é dividida por todos os dispositivos conectados a ele (GUERRIER, 2000). Além disso, a cada módulo extra conectado ao barramento, a capacitância do mesmo é aumentada, fazendo com que o sistema dissipe mais energia. Devido a estes fatores, um barramento comporta apenas alguns módulos para que não ocorra uma queda no desempenho global do sistema (ZEFERINO, 2002). O desafio é estabelecer uma estrutura de comunicação capaz de atender a requisitos de desempenho e consumo de energia para diferentes aplicações (BENINI, 2001). Uma alternativa em substituição ao uso de barramentos veio das redes de computadores. A partir do momento em que se tem um sistema computacional inteiro em um único chip, por que não interconectá-los também em uma rede em chip? Partindo desta premissa, surgiu o conceito de Rede-em-Chip (do inglês, *Network-on-Chip* - NoC) (BENINI, 2002). Em redes-em-chip os problemas encontrados em barramentos não acontecem por que as conexões são do tipo ponto-a-ponto, e, dessa forma, o desempenho é independente do número de módulos conectados à rede. Com o aumento do número de EPs interligados aos atuais sistemas e com o avanço no uso de MPSoCs, torna-se cada vez mais interessante usar uma rede de interconexão no lugar de um barramento (SINGH, 2006),

(BENINI, 2001) (DALLY, 2001), já que utilizando uma NoC é possível obter uma comunicação escalável e de alto desempenho.

No entanto, para que uma rede-em-chip possa ser utilizada como solução de interconexão, é necessário que interfaces de rede (NI, do inglês, *network interfaces*) sejam aplicadas na conexão dos EPs à rede. As interfaces de redes são responsáveis por adaptar o protocolo dos EPs ao protocolo da rede, empacotar e desempacotar as mensagens transferidas entre a NoC e os EPs, realizar o controle de fluxo dos dados, garantir que os dados sejam entregues corretamente, sem perdas de informações, adicionar informações referentes ao roteamento no cabeçalho da mensagem, adicionar sinais de controle referente ao tipo de pacote, dentre outras funções (DE MICHELI, 2006), (BHOJWANI, 2003).

Na literatura, porém, obtêm-se inúmeros trabalhos que focam em arquiteturas de redes-em-chip, propondo soluções para topologias de rede, algoritmos de roteamento, reutilização dos recursos da NoC, dentre outros. Contudo, numa proporção muito menor, encontram-se trabalhos que abordam soluções de interfaces de rede. Mesmo considerando esse número limitado de trabalhos que apresentam propostas de interfaces de rede, ainda existem várias situações que não foram abordadas na literatura e que ocorrem frequentemente em aplicações utilizadas como *benchmarks* de NoCs.

Além disso, muitas das propostas de redes-em-chip foram validadas a partir de geradores de tráfegos, sendo que, mesmo utilizando as taxas de comunicação dos módulos que compõem uma aplicação, muitas vezes, estes simuladores de tráfego não contemplam todas as situações que envolvem de fato uma aplicação, e que precisam ser consideradas nos projetos de interfaces de rede. Situações estas como dependência de dados de outras fontes para consideração de início do processamento de um EP, módulos com comportamento de tráfegos irregulares, necessidade de sincronismo de pacotes e consideração da latência no envio de dados da rede para os núcleos quando os EPs apresentam indisponibilidade de recebimento de novos dados. Outros fatores ainda precisam ser considerados quando se tem uma aplicação interconectada por NoC, como inconstância no tempo de processamento dos EPs em função dos dados de entrada recebidos e indisponibilidade do recebimento de novos dados devido aos recursos de armazenamento das interfaces de rede estarem cheios, dentre outras várias situações que nem sempre são consideradas pelas ferramentas de simulação de NoCs.

O desafio do projeto de interfaces de rede é obter uma solução em que a maioria dos recursos do circuito possa ser compartilhada entre os EPs que necessitam enviar ou receber dados para/da NI. Além disso, a NI, assim como a NoC, tem a responsabilidade de garantir um alto desempenho de comunicação para a aplicação. Uma NI pode ainda requerer uma lógica de controle complexa para adaptar o protocolo do IP ao protocolo da NoC, e as decisões tomadas nesse sentido estão diretamente relacionadas às características de desempenho de comunicação da aplicação.

Neste trabalho serão apresentadas propostas de interfaces de redes que satisfazem aplicações em geral. No desenvolvimento desse trabalho foram projetadas interfaces de rede genéricas, que podem ser utilizadas para a interconexão de qualquer módulo à rede com um mínimo de reprojeto. Além disso, as NIs projetadas apresentam módulos que podem ser reutilizados por qualquer EP que envie ou receba dados para/do roteador na qual a interface está conectada.

Como estudo de caso, para validar as interfaces de redes projetadas e comprovar a flexibilidade da implementação das mesmas, utilizou-se a aplicação de um

decodificador de vídeo baseado no padrão H.264 (PEREIRA, 2009) conectado à rede SoCIN (ZEFERINO, 2003) através das interfaces de rede desenvolvidas. Utilizando-se o decodificador de vídeo para esta interconexão, puderam-se analisar as reais necessidades que as interfaces de rede precisam atender e tomou-se conhecimento de diversas situações que ocorrem ou podem ocorrer na comunicação entre os EPs de uma aplicação. Situações estas que precisam ser analisadas e consideradas nos atuais projetos de interfaces de rede. A partir dessa implementação, pôde-se também explorar diversas características de configuração da NoC e das interfaces de rede.

As interfaces respeitam as parametrizações projetadas para a NoC SoCIN, como largura do canal e profundidade das FIFOs utilizadas nos canais de entrada da rede. Além disso, as interfaces de rede também permitem configuração da largura do pacote de dados do EP e da profundidade das FIFOs que as constituem, já que a profundidade ideal dependerá das exigências de cada aplicação. Uma única interface de rede também possibilita o recebimento e envio de pacotes com tamanhos variados, pois cada EP pode apresentar pacotes de dados com tamanhos específicos, tanto no recebimento dos dados, como no envio dos mesmos. Com estas possibilidades de configuração, os resultados de latência, vazão, área e frequência da implementação do decodificador de vídeo em NoC foram analisados e comparados com a aplicação que apresenta conexão direta entre os módulos. Analisaram-se ainda as características de outras propostas de NIs encontradas na literatura, considerando-se vantagens e desvantagens de cada trabalho e comparando-as com o trabalho desenvolvido.

Não é o foco desse trabalho discutir de forma aprofundada os motivos para os quais foram definidas as divisões dos módulos do decodificador de vídeo H.264 para conexão aos roteadores da rede, nem justificar o uso da NoC para esta aplicação. O decodificador de vídeo foi utilizado como estudo de caso com o objetivo de obter núcleos reais e, para que assim, se tomasse conhecimento das reais necessidades de interconexão quando se faz o uso de uma NoC.

## 1.1 Descrição do Problema

A diminuição das escalas de tecnologia e, consequentemente, a redução dos tamanhos dos dispositivos têm levado a um aumento considerável na densidade do chip. No entanto, associado a isto, aparecem também os problemas de interconexão, já que os sistemas têm se tornado mais complexos, e passaram a ser constituídos por numerosos elementos de processamento (BENINI, 2002). Os EPs podem ser conectados por soluções de interconexão como barramentos, estruturas de conexão *ad-hoc*, *crossbar* e mais recentemente, as redes-em-chip. Como solução a estes problemas, as NoCs têm sido uma excelente alternativa (SINGH, 2006).

No entanto, a conexão dos EPs que constituem um sistema à NoC não ocorre de forma direta. Para isso, é necessário o uso de circuitos conhecidos como interfaces de rede que fazem a conexão dos módulos aos roteadores da NoC. A principal função de uma interface de rede é adaptar a comunicação entre os núcleos e os roteadores, conforme está ilustrado na figura 1.1. A figura 1.1 apresenta parte de uma aplicação com alguns módulos conectados aos roteadores de uma NoC. Como se pode observar nesta figura, cada núcleo apresenta uma necessidade diferente em termos de frequência de operação e de largura de dados para o envio e recebimento dos pacotes pela rede.

Sendo assim, para que a correta transmissão das mensagens entre os diferentes nodos seja realizada, é indispensável o uso de interfaces de redes na conexão dos núcleos á NoC.

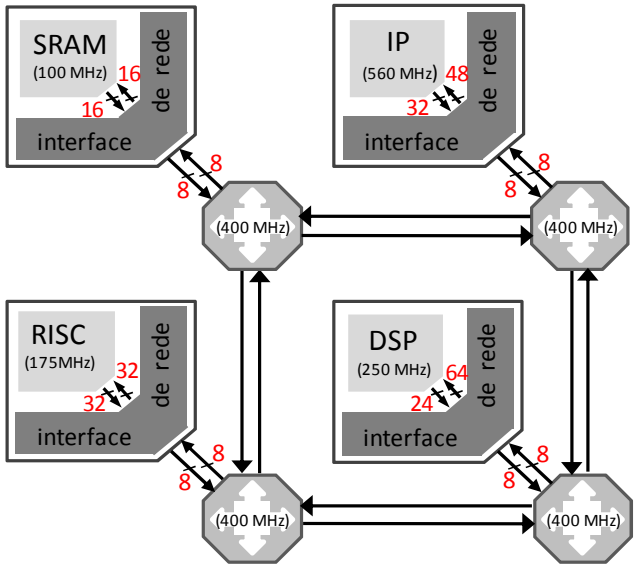



Figura 1.1: Exemplo da necessidade de NIs quando se utiliza uma NoC para a interconexão dos núcleos que compõem uma aplicação.

As interfaces de redes têm revelado diversas situações que precisam ser solucionadas e, além disso, muitos problemas ainda não estão totalmente tratados na literatura, já que poucos projetos em nível arquitetural têm sido propostos, embora estes circuitos sejam imprescindíveis quando se faz a utilização de uma NoC.

Além disso, para que essa conexão ocorra apropriadamente, é requerido que tais interfaces de rede sejam bem definidas e abstraiam os detalhes da interconexão (BHOJWANI, 2003), ou seja, quanto mais direta for a conexão dos EPs às NIs, menor é o tempo de projeto e mais independente se torna ao projetista a implementação de um sistema interconectado por uma NoC.

Neste trabalho foram levantadas algumas necessidades de interfaces de rede e a partir destas análises, algumas soluções arquiteturais para atender a estas exigências foram implementadas, sendo que, algumas delas ainda não tinham sido apresentadas na literatura.

## 1.2 Caracterização do Escopo do Trabalho

Embora existam vários estudos que relatam que o uso de soluções antigas de interconexão não sustentam os atuais sistemas (HANSSON, 2007), (GUERRIER, 2000) e que as NoCs têm tomado este espaço como principal solução a esta necessidade, ainda é muito recente o uso de redes-em-chip em aplicações reais. No entanto, já se percebe algum interesse da indústria neste tipo de solução e isso é mostrado pela existência de algumas propostas de NoCs (ARTERIS), (INoCs), (SILISTIX) que têm sido adotadas por algumas empresas de semicondutores. Dentro deste contexto, estão também as interfaces de redes, foco deste trabalho, que se fazem necessárias quando a solução por uma conexão utilizando NoC é definida para o projeto.

## 1.3 Objetivos e Contribuições

A contribuição mais relevante deste trabalho é a obtenção uma proposta de interface de rede onde a maioria das funcionalidades de interfaceamento é incorporada a qualquer EP que precise ser conectado a um roteador de uma NoC. Dessa forma, diferentes núcleos podem ser conectados à rede com o mínimo de esforço de projeto. Neste trabalho, propuseram-se interfaces de rede simples, genéricas e reutilizáveis, de forma que seja possível adaptá-las para a interconexão de qualquer outra aplicação por uma rede-em-chip. Dessa forma, foram desenvolvidas unidades que realizam o empacotamento e desempacotamento dos dados trafegados pela rede, em que a instanciação de uma única unidade permite o envio de dados para diferentes destinatários, e o recebimento de dados de diferentes origens. Para que estas características sejam atendidas, as interfaces de rede projetadas apresentam uma série de parâmetros que podem ser configurados conforme as exigências da aplicação.

## 1.4 Considerações

Nesse capítulo foram mencionados diversos motivos pelos quais as NoCs têm sido alvo de estudo para a interconexão de SoCs. No entanto, quando se utiliza uma NoC necessita-se de interfaces de rede para possibilitar essa conexão e estas arquiteturas são o alvo de estudo deste trabalho. Neste capítulo apresentou-se a necessidade da proposta de trabalho e as contribuições que serão obtidas com o mesmo. Cada uma das três arquiteturas que foram utilizadas nesse trabalho (NoC, decodificador de vídeo e NIs) será detalhada nos capítulos que seguem. No próximo capítulo, primeiramente são apresentados alguns conceitos de redes-em-chip e, posteriormente, é apresentada a rede SoCIN, que foi a NoC utilizada no desenvolvimento deste trabalho.

# 2 CONCEITOS DE REDES-EM-CHIP

As redes-em-chip, embora sejam motivo de grande estudo entre alguns pesquisadores, é ainda um assunto que está sendo analisado pela indústria, com muitos conceitos e ideias até então em discussão. As NoCs apresentam várias vantagens sobre outros métodos, permitindo a interconexão de centenas ou até de milhares de módulos (GUERRIER, 2000), (DALLY, 2001), (AHMAD, 2005). Nas seções que seguem, serão comentadas as principais características das NoCs e as vantagens e desvantagens no uso de redes-em-chip e o porquê seu uso tem sido fortemente indicado por tantos pesquisadores.

Este capítulo apresentará primeiramente as principais características de redes intrachip, comentando sobre como seu uso tem estado em destaque como solução de conexão de sistemas complexos. Na próxima seção serão descritos os principais componentes e algoritmos que compõem uma NoC, tais como topologia, roteamento, chaveamento, memorização, arbitragem e controle de fluxo. Ainda serão relacionadas, de forma breve, algumas técnicas de estado da arte utilizadas em redes-em-chip. Por fim, a rede SoCIN será apresentada dentro dos conceitos de NoCs apresentados na próxima seção, que foi a NoC utilizada neste trabalho.

## 2.1 Características Gerais de Redes-em-Chip

O uso de NoCs tem como objetivo obter alternativas para minimizar os problemas encontrados pelas demais arquiteturas de interconexão existentes, como barramentos, fios dedicados, *crossbar* e outras. Dentre as principais vantagens no uso de NoCs, pode-se citar:

- ✓ Paralelismo (BJERREGAARD, 2006): devido à multiplicidade de caminhos possíveis em uma NoC;
- ✓ Estruturação e o gerenciamento de conexões em tecnologias sub-micrônicas (BENINI, 2002), (DALLY, 2001): utilização de fios mais curtos, ponto-a-ponto, com menor capacitância parasita;
- ✓ Compartilhamento de fios (BOLOTIN, 2004), (DALLY, 2001): possibilitando sua utilização de maneira mais eficiente;
- ✓ Confiabilidade (VELLANKI, 2004): possibilitando o uso de várias técnicas que permitem aumentar a confiabilidade do sistema;
- ✓ Escalabilidade (GUERRIER, 2000), (BJERREGAARD, 2006): permitindo adicionar mais módulos a rede sem comprometer o desempenho do sistema;
- ✓ Reusabilidade (BENINI, 2002): possibilitando aproveitar a mesma estrutura de comunicação e os núcleos já existentes em aplicações distintas;

- ✓ Decisões de roteamento distribuídas (BENINI, 2002), (GUERRIER, 2000): permitindo um maior balanceamento na utilização dos recursos da rede.

Além das vantagens acima relacionadas, as NoCs têm como característica a possibilidade de reinstanciar os roteadores para qualquer tamanho de NoC, apresentam largura de banda escalável e permitem o uso de BISTs (*Built-In Self-Test*) dedicados. Já, os barramentos apresentam largura de banda limitada e dividida para todos os módulos conectados (TANENBAUM, 1999), além de não apresentarem boa testabilidade. No entanto, como qualquer proposta arquitetural, as NoCs também podem apresentar algumas desvantagens quando comparadas a outras opções. Sendo assim, algumas possíveis desvantagens no uso de NoCs são (GUERRIER, 2000):

- ✓ Considerável consumo de área de silício e potência em relação aos barramentos, o que se deve em função do uso de roteadores e interfaces que não são necessários em outras possibilidades de conexões;
- ✓ Necessidade do uso de interfaces de rede: os núcleos conectados à rede necessitam de adaptadores de protocolo para a comunicação;
- ✓ Latência (tempo para transmissão de pacotes): causada por contenções na rede e em função dos roteadores;
- ✓ Complexidade de projeto.

No entanto, os requisitos de área, potência e a latência são considerados desvantagens quando comparados a outras alternativas de conexão dependendo da aplicação em questão, ou seja, conforme o número de núcleos presentes na rede (ZEFERINO, 2002) e em função da necessidade de comunicação dos EPs.

Uma NoC é uma estrutura composta por um conjunto de roteadores e enlaces (ou *links*). Os roteadores são interconectados entre si através dos *links* e são responsáveis por verificarem um possível caminho para a transferência de dados. Normalmente estes dados são divididos em pacotes e enviados pela rede. Os *links* são responsáveis pela conexão entre os roteadores. Cada roteador pode possuir um núcleo acoplado, sendo que um núcleo pode ser um processador, uma memória, um cluster composto por processadores e/ou memória ou ainda blocos dedicados (MOLLER, 2006), também chamados de IPs ou EPs.

As redes-em-chip são classificadas quanto a sua topologia, quanto ao roteamento das mensagens, quanto a arbitragem, quanto ao chaveamento, quanto ao controle de fluxo utilizado, dentre outras características. A figura 2.1 apresenta uma NoC 3x3 com a identificação de algumas partes que constituem uma rede-em-chip. No modelo exemplificado, a bufferização é feita somente nos canais de entrada e o roteador utilizado é um *crossbar*.

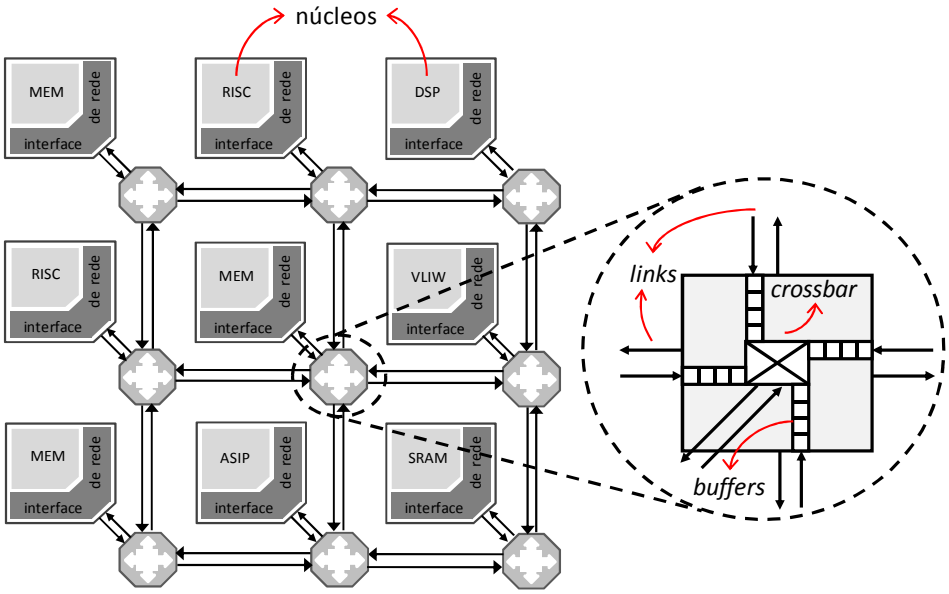



Figura 2.1: Exemplo de uma NoC 3x3 e as partes que a constituem.

### 2.1.1 Roteador

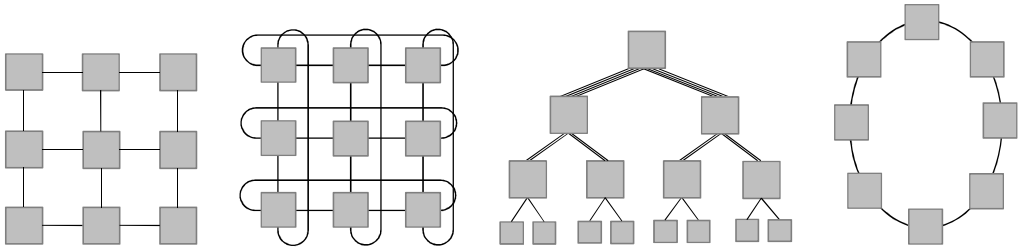
Um conjunto de roteadores conectados por *links* constituem uma rede-em-chip. Um roteador é formado por um núcleo de chaveamento (*crossbar*), um árbitro, um controle de roteamento e portas que são constituídas de canais de entrada e saída. Os canais normalmente possuem *buffers* para armazenamento temporário de mensagens.

### 2.1.2 Topologia

A definição de topologia de uma rede intrachip é determinada pela forma como os seus roteadores estão interconectados. Quanto aos tipos de topologias, pode-se ter um roteador associado a cada EP ou este pode estar conectado a uma rede de roteadores e, neste caso, somente alguns destes roteadores estão conectados a um EP (ZEFERINO, 2003). As diferentes topologias de NoCs existentes estão associadas também ao número de canais que cada roteador possui e como estes estão conectados aos demais roteadores. Nesse caso, a topologia definida também pode influenciar no tipo de roteamento atribuído para a rede, já que quando uma mensagem é enviada para um determinado nodo, se este não for o destinatário da mesma, o algoritmo de roteamento deve indicar o caminho que este pacote deve seguir na rede para que o mesmo chegue ao seu destinatário. Em função disto, a vizinhança de cada roteador determinará os possíveis fluxos de dados pela rede.

Dentre as topologias de rede-em-chip mais comuns, pode-se citar a grelha (*mesh*, do inglês), toróide (*torus*, do inglês), em anel e em árvore. Nas topologias de grelha e toróide os roteadores podem estar dispostos em duas dimensões (2-D) ou em três dimensões (3-D). A topologia grelha é a mais utilizada na literatura. Nesta topologia, cada nodo é conectado a quatro nodos vizinhos (exceto nas extremidades), chamados de leste, oeste, norte e sul e todos os canais possuem a mesma largura de dados. A topologia toróide é muito similar à topologia grelha com a diferença de que os canais de uma das extremidades são conectados aos canais da extremidade oposta. A topologia em anel é a mais simples dentre as topologias, mas, no entanto, apresenta escalabilidade limitada já que o desempenho decresce à medida que mais nodos são adicionados à

rede. Na topologia em árvore, o modelo mais utilizado é a *fat-tree*, onde os EPs estão conectados somente nas folhas e a largura dos canais aumenta quanto mais próximo eles estiverem da raiz, já que é onde há maior concentração de tráfego na rede (PASRICHA, 2008). A figura 2.2 apresenta algumas das topologias de NoC acima mencionadas.



(a) (b) (c) (d)

Figura 2.2: Exemplos de topologias de redes: grelha 2-D (a), toróide 2-D (b), árvore (c), anel (d).

Também tem sido muito frequente o estudo de topologias de NoC irregulares e adaptáveis conforme a necessidade da aplicação (STENSGAARD, 2008), (CHAN, 2008). Quando se faz o uso de uma topologia customizada para uma aplicação, obtém-se resultados de eficiência e desempenho melhores do que quando se opta por uma topologia regular (JERGER, 2009). As topologias irregulares têm como objetivo reduzir a distância entre os nodos com relação às demais topologias. Para isso, normalmente alguns roteadores são removidos da rede para realização de uma conexão ponto-a-ponto entre os elementos de processamento que apresentam altas taxas de comunicação. Já, em outros casos, os roteadores podem apresentar um número irregular de canais, sendo que alguns canais podem ser removidos e outros adicionados quando se compara a uma topologia regular (STENSGAARD, 2008). A figura 2.3 apresenta um exemplo de topologia irregular, conforme descrito acima.

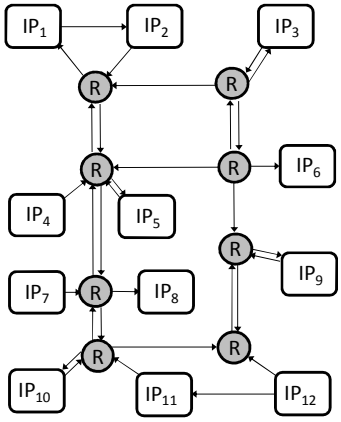



Figura 2.3: Exemplo de topologia irregular ideal para uma determinada aplicação (STENSGAARD, 2008).

Fazendo-se uso de uma topologia customizada, tem-se uma redução na conectividade da rede justificável para uma aplicação especifica, já que, neste caso, uma NoC regular com todas as suas conexões não iria aumentar o desempenho da aplicação, além de apresentar um aumento em área e potência. Certamente que essa solução é ideal tratando-se de uma aplicação especifica, pois se for necessário executar diferentes aplicações utilizando-se a mesma rede ou se uma mesma aplicação puder ser atualizada

ou alterada após a sua implementação, uma solução customizada dificilmente será a ideal em termos de QoS (qualidade de serviço, do inglês, *Quality of Service*), além da rede apresentar perda de escalabilidade. No entanto, também têm sido propostas arquiteturas de topologias irregulares que permitem reconfiguração, ou seja, a topologia é definida conforme a necessidade da aplicação em execução (STENSGAARD, 2008).

### 2.1.3 Roteamento

Em função da definição do tipo de topologia de rede adotada é que são analisados os possíveis algoritmos de roteamento que podem ser aplicados à NoC. O algoritmo de roteamento irá decidir qual o caminho que uma mensagem irá percorrer pela rede, do seu transmissor até atingir o seu receptor. O objetivo do algoritmo de roteamento é distribuir o tráfego entre os caminhos fornecidos pela topologia da rede evitando os chamados *hotspots* (canais da rede muito congestionados), minimizando a contenção da rede e melhorando os resultados de vazão e latência da mesma (JERGER, 2009). Os algoritmos de roteamento podem ser divididos em inúmeras categorias, tais como estáticos ou dinâmicos, distribuídos, centralizados ou fontes, determinísticos ou adaptativos (PASRICHA, 2008) (ZEFERINO, 2003).

Inúmeros algoritmos de roteamento têm sido propostos nos últimos anos, embora os algoritmos mais utilizados ainda sejam os de ordenamento por dimensão (DOR – *dimension-ordered routing*) devido à sua simplicidade. Estes algoritmos de roteamento também são classificados como determinístico, ou seja, uma mensagem enviada do nodo A para o nodo B percorrerá sempre o mesmo caminho, e são algoritmos frequentemente utilizados em redes de topologia grelha e toróide (2-D e 3-D). Um exemplo de algoritmo determinístico muito presente na literatura é o X-Y (BOBDA, 2005). Esse algoritmo primeiramente envia pacotes na direção de X e somente quando não precisa percorrer mais nessa direção é que passa a percorrer na direção Y, até atingir o seu destinatário. Uma vez que o pacote tenha deixado de percorrer na direção X, ele não pode tomar mais essa direção. Embora se trate de um algoritmo simples, ele é frequentemente utilizado em arquiteturas de NoC porque é livre de *deadlock.* Uma situação de *deadlock* em redes ocorre quando existe uma dependência cíclica entre múltiplas mensagens. No entanto, a obtenção desta garantia acarreta em uma desvantagem pelo fato de que este algoritmo reduz o número de caminhos livres a serem acessados por outros pacotes quando uma mensagem está ocupando um determinado caminho.

Uma outra possibilidade de algoritmos de roteamento são os adaptativos. Estes algoritmos têm sido amplamente estudados na literatura (BOBDA, 2005), (MING, 2006), (HAIBO, 2007), e a definição da rota por este tipo de algoritmo é decidida conforme a situação de tráfego nos canais ou conforme alguma outra condição estabelecida, como por exemplo, a detecção de *links* falhados na rede. Dessa forma, os algoritmos de roteamento adaptativo podem optar por um caminho ou outro com o objetivo de diminuir a latência e aumentar a vazão da rede (MING, 2006), ou para tolerar falhas que tenham sido detectadas em algum canal da NoC (HAIBO, 2007). No entanto, quando um algoritmo totalmente adaptativo é escolhido para o roteamento de mensagens na rede, situações de *deadlock* podem ser um problema. Alguns trabalhos propõem soluções para contornar estas situações, e este é um assunto que há algum tempo tem sido discutido na literatura (DUATO, 1993), (DUATO, 1995).

Outro desafio com relação ao uso dos roteamentos totalmente adaptativos é preservar o ordenamento dos flits no receptor, já que cada flit pode percorrer uma rota diferente até o destinatário. Mecanismos de reordenamento já foram propostos como solução para este problema (KIM, 2006).

Um outro algoritmo de roteamento muito utilizado é o baseado em tabelas. Esse tipo de algoritmo utiliza uma tabela em cada roteador, e com base nessa informação é definido qual será o próximo *hop* da mensagem. Alguns trabalhos utilizam essa técnica para alterar os valores de roteamento pré-definidos na tabela conforme a disponibilidade da rede, tornando o roteamento adaptativo (JERGER, 2009). A figura 2.4 apresenta os tipos de rotas mínimas possíveis para enviar uma mensagem de um nodo A para um nodo B. Conforme se pode observar nas rotas apresentadas, o algoritmo X-Y, como é do tipo determinístico, apresenta somente um caminho de A para B. No entanto, ao total, 6 rotas mínimas são possíveis para esse caso quando se faz o uso de um algoritmo totalmente adaptativo.

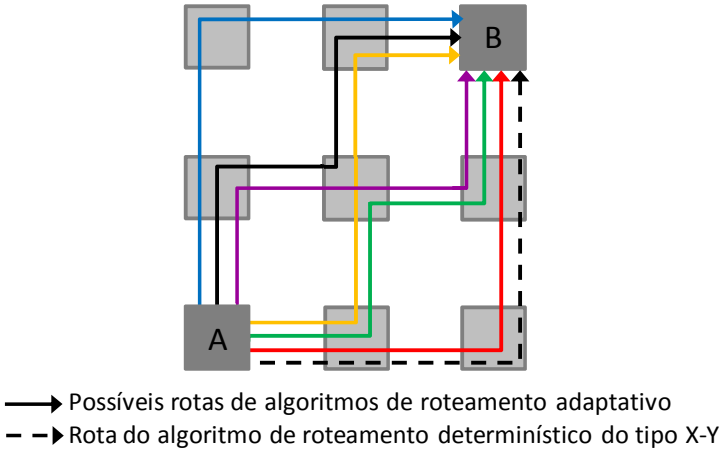



Figura 2.4: Exemplos de algoritmos de roteamento e suas rotas para enviar uma mensagem do nodo A ao nodo B.

### 2.1.4 Chaveamento

O chaveamento é responsável por definir como os dados fluem na rede, ou seja, como um dado recebido por um canal de entrada é repassado para um dos seus canais de saída. Para a melhor compreensão dos tipos de chaveamento, é preciso compreender a granularidade das transferências de dados em uma rede. Uma mensagem é quebrada em pacotes e cada pacote pode ser ainda particionado em unidades menores, denominadas de flits (*flow control unit*). Cada flit é formado por um ou mais phits, sendo que cada phit tem a largura do canal físico de dados. A figura 2.5 apresenta a estrutura de uma mensagem para facilitar o entendimento destas definições.

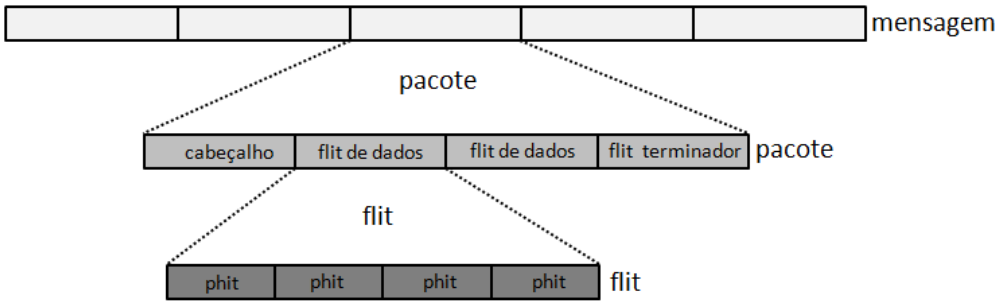



Figura 2.5: Composição de uma mensagem por pacotes, flit e phits (figura alterada a partir de PASRICHA, 2008).

Quanto às estratégias de chaveamento, há três métodos de transferência de pacotes mais utilizados:

- Chaveamento por Circuitos (*circuit-switching* – CS): neste tipo de chaveamento um caminho entre o nodo receptor e o nodo transmissor é reservado antes das mensagens começarem a ser enviadas. É estabelecido um caminho físico direto entre fonte e destinatário para a transferência das mensagens que deve ser mantido até o término da comunicação, sem que haja contenção na rede. O caminho reservado só é liberado durante o avanço do terminador da mensagem até o receptor (CHANG, 2006).

- Chaveamento por Pacotes: neste chaveamento os pacotes são transmitidos sem a necessidade de reservar o caminho pelo qual o pacote deve seguir. Quando os *links* estiverem livres, os pacotes que estavam armazenados nos buffers são encaminhados para os roteadores adjacentes ou para um núcleo destinatário. Nesse caso, três métodos de chaveamento podem ser utilizados:

    - Armazena e repassa (*store-and-foward* - SAF): este método é utilizado quando as mensagens enviadas entre os nodos da rede são curtas e frequentes. Nesse caso, os pacotes possuem um cabeçalho com as informações de roteamento. Cada pacote reserva o caminho necessário até o seu destinatário e os pacotes são repassados através de um protocolo entre os buffers presentes nos canais.

    - Transpasse Virtual (*virtual cut-through* - VCT): este método apresenta o mesmo conceito de mensagens por pacotes, no entanto, neste caso, o pacote não precisa ser inteiramente armazenado. Quando o canal de saída desejado estiver disponível, o restante do pacote é repassado diretamente para o canal, sem a necessidade de ser armazenado nos buffers.

    - Wormhole (WH): nesse tipo de chaveamento, o envio dos dados pela rede é feito por unidades de flits. Neste método, todos os flits seguem o caminho reservado pelo cabeçalho e somente após o envio do flit terminador é que o caminho reservado é liberado. A vantagem desta técnica é que os canais de entrada não precisam armazenar um pacote inteiro e com isso, profundidades menores de *buffers* na entrada dos canais podem ser definidas. *Buffers* muito profundos

inserem um aumento considerável de área e potência na rede. Com essa técnica, é possível definir uma profundidade de *buffer* que justifique seu uso em termos de custo e desempenho. Esse método apresenta como desvantagem a não possibilidade de intercalar flits de diferentes pacotes, já que somente após o envio de todo o pacote é que o caminho alocado é liberado.

- Chaveamento por Canais Virtuais (*virtual channels* – VC): Esta técnica permite o envio de flits de diferentes pacotes intercalados em um mesmo link. Dessa forma, soluciona-se o problema de bloqueio de um canal por um determinado pacote na rede.

### 2.1.5 Memorização

A memorização em NoCs é necessária sempre que o tipo de chaveamento for por pacote, pois se precisa garantir que não ocorrerão perdas das mensagens que trafegam pela rede. Quando um pacote é enviado pela rede e este precisa acessar um canal que já está em uso por outro pacote, precisa-se de algum esquema de memorização para que ele possa ser enviado ao seu destinatário após a liberação do canal. O armazenamento dos flits pode ser utilizado tanto nos canais de entrada como nos canais de saída ou em ambos, podendo-se optar por uma solução centralizada, independente ou compartilhada.

Quando a memorização ocorre na entrada, tem-se um espaço de memória alocado para cada canal de entrada do roteador, conforme está ilustrado na figura 2.6. A implementação mais comum para esse armazenamento é a utilização de *buffers* independentes para cada canal. Uma utilização muito frequente é utilizar *buffers* FIFO (*First-In First-Out*) por ser uma alternativa muito simples de ser implementada. No entanto, a utilização de uma única FIFO por canal de entrada afeta o desempenho da rede, já que somente um pacote pode trafegar por vez em um determinado canal, impedindo o tráfego de outros pacotes que também desejem utilizar o mesmo canal. Desta forma, existem outras alternativas para os tipos de *buffers* a serem empregados, como *buffers* SAFC, *buffers* SAMQ e *buffers* DAMQ (ZEFERINO, 2003).

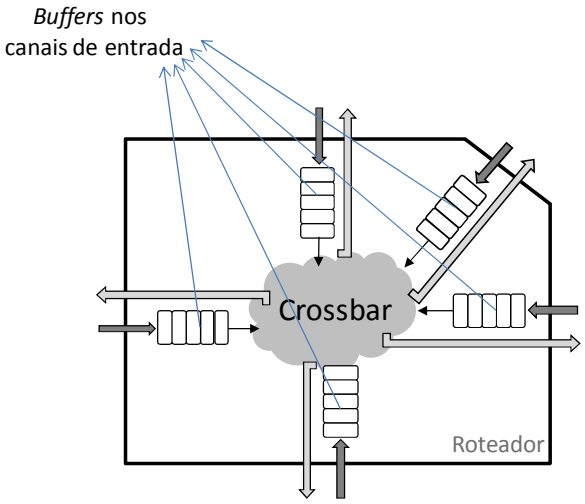



Figura 2.6: Exemplo de uso de FIFOs na entrada dos canais de um roteador para memorização dos pacotes.

Na memorização centralizada, cada canal tem um buffer que pode ser particionado para todos os demais canais do roteador, podendo apresentar algum controle para que este particionamento ocorra em função dos tráfegos entre os canais. Essa solução evita o bloqueio do canal quando mais de um pacote acessa um mesmo caminho.

Muitos trabalhos abordam soluções de memorização utilizando técnicas de adaptabilidade com o objetivo de aumentar o desempenho da rede ou reduzir o consumo de potência (NICOPOULOS, 2006), (AL FARUQUE, 2008), (MATOS, 2009a), (MATOS, 2009b). A adaptabilidade nesse caso está associada à seleção da profundidade de *buffers* conforme a necessidade requerida por cada canal. Os *buffers* presentes em uma rede-em-chip são os principais responsáveis pelo consumo de potência (XUNING, 2003), no entanto, *buffers* mais profundos asseguram um maior desempenho da aplicação. Sendo assim, as soluções anteriormente citadas procuram prover uma distribuição de *buffers* conforme a taxa de comunicação requerida para o canal. Dessa forma, reduz-se a profundidade dos *buffers* dos canais menos utilizados, encontrando-se um equilíbrio entre o consumo de potência e o desempenho da rede.

### 2.1.6 Arbitragem

A arbitragem é responsável por selecionar o canal de entrada que deve receber um determinado canal de saída. Esse mecanismo é importante quando pacotes em diferentes canais de entrada disputam um mesmo canal de saída. A arbitragem pode ser centralizada ou distribuída. Na arbitragem distribuída, cada canal de saída possui um árbitro. Existem vários algoritmos que são utilizados neste tipo de arbitragem, dentre eles pode-se citar o *Round-Robin*, que apresenta prioridade rotativa, ou seja, o último a requisitar o canal é o que apresenta menor prioridade na próxima alocação do canal. Outros algoritmos de escalonamento também podem ser utilizados, como, por exemplo, por idade (FCFS, LRS), por prioridades estáticas, dentre outros (ZEFERINO, 2003).

### 2.1.7 Controle de Fluxo

O controle de fluxo define como os recursos da rede são alocados para os pacotes que atravessam a NoC. Quando múltiplos pacotes disputam pelos mesmos recursos da rede, algumas técnicas precisam ser consideradas. A forma mais comum de controle de fluxo é a utilização de *buffers* nos canais de entrada para armazenamento dos flits. Nesse caso, um controle baseado em *handshake* define quando um flit pode ser armazenado nos buffers. São utilizadas duas linhas para esse controle, uma linha de validação enviada pelo transmissor, indicando a intenção de enviar um dado ao receptor (*val*) e uma linha de reconhecimento enviada pelo receptor, indicando a disponibilidade de espaço nos buffers (*ack*). A figura 2.7 ilustra como funciona o protocolo de *handshake*.

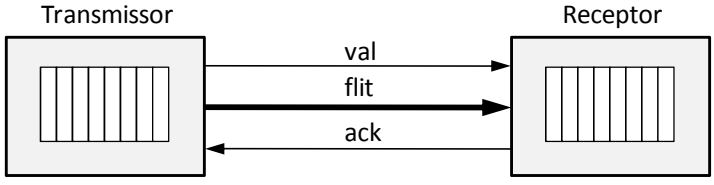



Figura 2.7: Controle de fluxo baseado em *handshake*.

## 2.2 Rede SoCIN

A rede-em-chip utilizada nesse trabalho foi a SoCIN (*System-on-Chip Interconnection Network*) (ZEFERINO, 2003). A rede SoCIN pode ser configurada na topologia *mesh* 2-D ou toróide 2-D, possui controle de fluxo por *handshake*, memorização de entrada por FIFOs e roteamento do tipo X-Y.

A rede SoCIN utiliza chaveamento por pacotes do tipo *wormhole.* Um flit da rede SoCIN corresponde à largura do canal físico da rede e cada flit possui tamanho de *n+2* bits, sendo 2 bits utilizados para indicar o tipo de flit: cabeçalho (*header*), carga útil (*payload*) ou terminador (*trailer*) e *n* corresponde ao número de bits referente aos dados do pacote. Os 2 bits que definem o flit são chamados de *bop* (*begin of-packet*), indicando início do pacote, quando igual a 1, e *eop* (*end-of-packet)* indicando fim do pacote, quando igual a 1. O primeiro flit do pacote é o cabeçalho e os demais flits seguem o cabeçalho pela rede em modo de *pipeline*. Como o pacote é dividido em flits, a informação de roteamento consta no flit de cabeçalho, que tem como função alocar o caminho pelo qual os demais flits seguirão. Neste tipo de chaveamento, não é possível intercalar flits de pacotes diferentes pelo mesmo canal lógico, sendo assim, um pacote com todos os seus flits deve atravessar o canal antes de liberá-lo para outro pacote.

O esquema de arbitragem utilizado na SoCIN é dinâmico e distribuído e, para tanto, faz uso do algoritmo *Round-Robin*, que garante também que nenhum pacote fique bloqueado permanentemente na rede (situação conhecida como *starvation).*

A técnica de controle de fluxo utilizada na rede SoCIN é baseada no protocolo de *handshake*, em que o emissor informa ao receptor a intenção de enviar um dado e este responde se existe *buffers* disponíveis para recebê-lo. Esse reconhecimento é feito através de uma linha de controle de fluxo denominada *val* e *ack.*

O roteador utilizado na SoCIN é chamado de RASoC e possui 5 portas bidirecionais chamadas de L (*Local*), N (*North*), E (*East*), S (*South*) e W (*Weast*). Cada porta possui dois canais unidirecionais denominados canal de entrada e canal de saída. O roteador RASoC utiliza um *crossbar* para conectar as entradas às saídas. Este roteador possui 3 dimensões configuráveis: largura dos canais de comunicação, profundidade dos buffers e largura da informação de roteamento no cabeçalho do pacote.

## 2.3 Considerações

Neste capítulo apresentou-se brevemente alguns conceitos de redes-em-chip que são fundamentais para a compreensão de como as interfaces de rede foram projetadas para serem utilizadas juntamente com a NoC. Apresentou-se as principais características da rede SoCIN, que foi a rede intrachip utilizada nesse trabalho. Além disso, comentou-se sobre algumas técnicas de roteamento, memorização, topologias e chaveamento em NoCs que têm sido amplamente discutidas na literatura. No próximo capítulo serão comentadas algumas funções das interfaces de redes e serão apresentadas algumas necessidades que devem ser previstas nos projetos das mesmas.

# 3 INTERFACES DE REDE

As interfaces de rede (NIs, do inglês, network interfaces) são indispensáveis quando se faz o uso de uma rede-em-chip. A figura 1.1 ilustrou um exemplo genérico da necessidade do uso das NIs na interconexão de uma aplicação por uma NoC e agora na figura 3.1 são apresentados os principais módulos que compõem uma interface de rede.

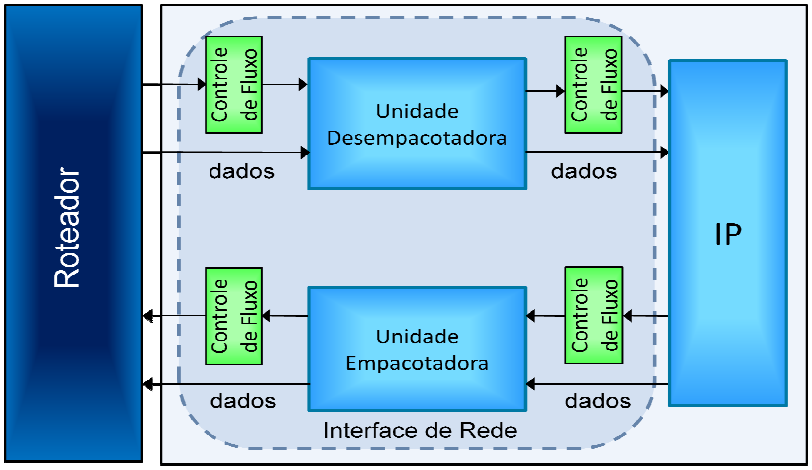



Figura 3.1: Principais módulos que constituem uma interface de rede.

A partir dos módulos apresentados na figura 3.1, serão detalhadas, neste capítulo, as responsabilidades que são atribuídas a uma interface de rede e as várias situações que devem estar incorporadas ao projeto de uma NI. Algumas das funções de uma interface de rede são (DE MICHELI, 2006):

- realizar o controle de fluxo dos dados, ou seja, garantir que os dados sejam entregues corretamente tanto para a rede como para o EP, sem perdas de informações (módulos de *controle de fluxo* na figura 3.1);
- adaptar o envio e recebimento de mensagens entre os EPs e a rede-em-chip, adequando a largura de dados e o protocolo de comunicação entre núcleo e rede (módulos de *controle de fluxo* na figura 3.1);
- particionar a mensagem que deve ser enviada pela rede, estruturando-a em pacotes ou flits. Aos pacotes ou flits são adicionados todos os controles necessários para que os mesmos cheguem ao núcleo destinatário. Tais controles podem conter informações do roteamento na rede, endereço de destinatário, identificação de origem, tipo de flit/pacote, dentre outras (módulo da *unidade empacotadora* na figura 3.1);
- receber os flits da rede e reconstituir a palavra de dados a ser transmitida ao EP, removendo informações de roteamento e todas as demais identificações inseridas ao pacote, que foram utilizadas apenas para controle da

transmissão do mesmo (módulo da *unidade desempacotadora* na figura 3.1);

O processo de empacotamento e desempacotamento realizado pelas NIs é crítico com relação à latência da rede e algumas implementações têm sido exploradas com o objetivo de obter resultados mais eficientes (DE MICHELI, 2006), (EBRAHIMI, 2009), (RADULESCU, 2004). Para que se permita uma redução de latência no envio e recebimento das mensagens, algumas características podem ser consideradas no projeto de NoCs e NIs. Para isso, as NIs devem permitir a configuração de alguns parâmetros com o objetivo de obter uma implementação adequada às exigências da aplicação. No entanto, uma parametrização apropriada para a NI deve ser verificada para cada caso, pois, definindo-se uma NI com alto desempenho, provavelmente o consumo de potência será maior, bem como a área obtida. Sendo assim, o ideal é utilizar a configuração mínima que atenda as exigências de uma aplicação, sem ocasionar desperdícios de recursos e sem comprometer o consumo de potência do sistema.

Como possibilidades de parametrização, uma NI pode ser projetada para permitir configuração da profundidade das FIFOs, da largura dos pacotes, da largura dos canais da NoC, dente outras. Os tamanhos dos pacotes/flits devem ser projetados para que a aplicação atinja um desempenho elevado ou que obtenha os parâmetros de QoS necessários para a aplicação (BOLOTIN, 2004).

Uma interface de rede deve ser capaz de receber e enviar os pacotes independentemente da topologia de rede utilizada. Existem algumas propostas de NoC em que o formato do pacote é muito dependente da topologia e isso pode comprometer a generalidade de uma interface de rede. Na SoCIN, o roteamento da rede é definido pelo cabeçalho. Como o tipo de roteamento da NoC está diretamente relacionado ao tipo de topologia, no caso das interfaces de rede projetadas para a SoCIN, se a informação de roteamento estiver sempre contida no flit de cabeçalho, o projeto da NI não será afetado, mesmo que seja utilizada uma outra topologia.

Em algumas situações, para cumprir com os requisitos de latência da aplicação, a NoC deve operar a uma frequência bem acima da frequência de operação da aplicação e o projeto de NoC deve prover que tais condições sejam atendidas (LIU, 2002). No entanto, o projeto das interfaces de rede deve atender a estes mesmos requisitos de frequência, pois do contrário, todo o esforço no projeto da NoC será desperdiçado.

Dependendo da aplicação, as NIs podem ter que agregar alguns serviços. Um serviço frequentemente necessário e muito discutido na literatura (BEIGNÉ, 2006), (SHEIBANYRAD, 2007), (THONNART, 2009), (NING, 2009) é prover uma adaptação para diferentes domínios de relógio, ou seja, quando os EPs apresentam um domínio de relógio diferente da rede. Uma solução muito usada para a sincronização dos domínios de relógio é utilizar FIFOs com relógios de leitura e escritas distintos, conforme o relógio definido para a NoC e para o EP. As soluções que têm sido discutidas para esse problema estão relacionadas ao uso de propostas que utilizem GALS (globalmente assíncronos, localmente síncronos, do inglês, *globally asynchronous and locally synchronous*).

Alguns outros serviços podem ser requeridos por uma interface de rede. Por exemplo, em roteamentos adaptativos, como não se conhece o caminho que cada pacote/flit pode percorrer na rede, a NI deve ser capaz de reordenar os pacotes/ flits. Um exemplo de NI com esta solução foi apresentada em (KIM, 2006), onde um esquema de rótulos é utilizado para a identificação de cada pacote/ flit e a partir dessa identificação

é possível reordená-los. Outros exemplos de serviços que podem ser atribuídos as NIs são as técnicas de correções de erros no recebimento dos dados, o uso de soluções de baixo de consumo de potência para aplicações com este tipo restrição, dentre outras (DE MICHELI, 2006).

Algumas das necessidades que uma NI precisa atender, conforme anteriormente levantadas, serão detalhadas na próxima seção. Estas necessidades foram consideradas nas NIs implementadas neste trabalho e na seção a seguir, serão comentadas qual a importância de cada uma destas definições e como estas características impactam na implementação de uma aplicação interconectada por uma NoC através de NIs.

## 3.1 Características necessárias aos projetos de interfaces de rede

### 3.1.1 Reusabilidade

A reusabilidade em NIs pode ser tratada em duas situações. A primeira proposta de reusabilidade em NIs é considerada dentro de uma mesma aplicação. Nessa situação, considera-se que os módulos que compõem uma NI devem ser reutilizáveis, ou seja, tanto a unidade de empacotamento, como a unidade de desempacotamento e os recursos de armazenamento definidos na NI devem permitir que qualquer EP possa utilizá-los. Por exemplo, uma unidade desempacotadora pode receber pacotes vindos de diferentes origens, sendo que, cada pacote recebido da rede pode apresentar um tamanho diferente. Nesse caso, a unidade desempacotadora deve ser capaz de receber os pacotes enviados por diferentes transmissores, identificá-los e restituir a palavra de dados que deve ser enviada ao EP. O mesmo ocorre com a unidade empacotadora, que nesse caso, tem a tarefa de permitir que pacotes sejam enviados para diferentes destinos. Nesse caso, cada pacote a ser enviado para a rede por uma mesma NI também pode apresentar tamanho variado, alterando também o número de flits a serem enviados para o roteador. Sendo assim, a unidade empacotadora identifica para quem os dados devem ser transmitidos e encapsula os flits, adicionando o cabeçalho apropriado a cada pacote, conforme o destinatário, e identificando quando deve ser enviado o flit terminador, que vai depender da largura de dados a ser enviada para cada nodo e da largura da rede.

A segunda tratativa de reusabilidade em NIs refere-se à possibilidade de utilizar a mesma descrição de NI para conectar os módulos de qualquer aplicação a uma NoC. Nesse caso, esta segunda proposta de reusabilidade em NIs refere-se à generalidade do projeto de uma NI, permitindo que a mesma implementação possa ser utilizada em diferentes aplicações. Sendo assim, o projeto de uma NI deve prever a possibilidade de configuração de alguns parâmetros para que os mesmos possam ser ajustados conforme os requisitos da aplicação. No entanto, muitas vezes, projetos de módulos genéricos não conseguem atingir um desempenho tão elevado quando comparado ao projeto específico. Por exemplo, se uma NI fosse projetada para atender a qualquer protocolo que possa ser utilizado por um EP, provavelmente essa generalidade ocasionaria desvantagens sobre outros aspectos importantes em projetos de hardware, como desempenho, consumo de potência ou área. Dessa forma, é necessário encontrar a melhor relação entre eficiência e reusabilidade.

O projeto de uma NI configurável permitirá a implementação de uma aplicação em NoC com um mínimo de reprojeto, reduzindo significativamente o tempo de desenvolvimento da aplicação. Quando a maioria dos recursos de uma interface de rede é reutilizada, é possível obter-se uma redução em área e no consumo de potência do

circuito. A utilização de uma NI configurável é também vantajosa quando se considera o uso de EPs também reutilizáveis.

### 3.1.2 Empacotamento e Desempacotamento

Uma das principais tarefas da interface de rede é empacotar e desempacotar as mensagens transmitidas via NoC. Na etapa de empacotamento, a NI recebe os dados do EP, os organiza em flits, acrescenta cabeçalho, códigos de identificação do tipo de flit, código de origem do módulo transmissor e demais controles que forem necessários. O número de flits a serem enviados é definido conforme a largura do canal da rede e do tamanho do pacote. Na etapa de desempacotamento, todos os controles adicionados ao pacote são removidos e agrupados de forma a reconstituir a palavra de dados original que deve ser transmitida ao EP.

### 3.1.3 Configuração da largura da palavra de dados

Conforme comentado anteriormente, uma NI deve assegurar que os seus módulos possam ser reutilizados e, para que isso seja possível, é imprescindível que a NI permita receber ou enviar pacotes com largura variada da palavra de dados. O tamanho do pacote depende do tipo de núcleo e do tipo de mensagem que o mesmo deseja enviar. Considerando-se que uma NI irá receber e enviar pacotes de/para diferentes elementos de processamento, o projeto da interface deve permitir o correto empacotamento e desempacotamento das mensagens para diferentes larguras de palavra de dados. Quando se implementa uma aplicação utilizando uma NoC como dispositivo de interconexão, muito provavelmente os diferentes nodos mapeados na NoC apresentarão largura da palavra de dados variada, já que um nodo pode ser constituído tanto de processadores, como de módulos de DSP, hardwares dedicados ou ainda de módulos de memória, dentre outros. Dessa forma, a NI deve permitir o envio e recebimento de pacotes de qualquer tamanho, sem necessitar acrescentar a NI módulos específicos para cada caso.

### 3.1.4 Configuração da largura do canal da NoC

Considerando-se que o número de bits de um phit corresponde à largura do canal da rede e que, nesse trabalho, um flit tem a mesma largura de um phit, a definição de qual tamanho de phit é mais adequada para aplicação vai depender do tamanho das mensagens que os EPs precisam enviar para a rede, da taxa com que os dados são gerados e da exigência de vazão requerida pela aplicação. Conforme o tamanho da mensagem, definindo-se a largura do canal da rede, sabe-se o número de flits que serão enviados para o roteador por um determinado núcleo. Algumas decisões de projeto podem ser definidas conforme os requisitos da aplicação. Por exemplo, a opção por flits menores pode ser preferida, pois, com isso, reduz-se a largura dos buffers e o consumo de potência da rede. Por outro lado, o uso de flits maiores diminui o número de flits totais a serem enviados, reduzindo a latência da rede. Por esses motivos, esse parâmetro não deve ser fixo e a parametrização do mesmo ocorrerá conforme a necessidade da aplicação. Normalmente as redes-em-chip apresentam possibilidade de configuração desse parâmetro. Nestes casos, a NI também deve ser capaz de permitir parametrização em função das necessidades acima mencionadas e para permitir tal configuração, algumas características de projeto devem ser consideradas na implementação das NIs para que a mesma apresente essa possibilidade.

No entanto, quando as informações de roteamento são inseridas no flit de cabeçalho, a largura mínima que o flit pode assumir deve ser suficiente para contemplar

todas as informações necessárias para o roteamento. Tais informações podem depender do tamanho da rede e da distância entre os nodos que precisam se comunicar.

### 3.1.5 Armazenamento de pacotes e configuração da profundidade das FIFOs

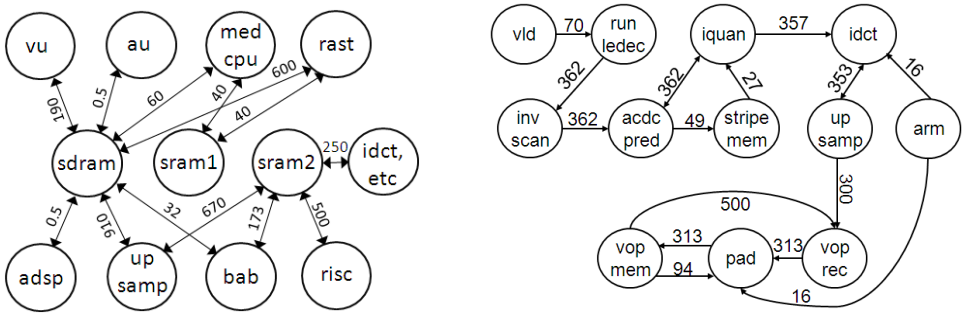
Uma das necessidades em projetos de interfaces de rede é o uso de recursos de armazenamento dos dados que trafegam entre os EPs e a NoC. Existem pelo menos 3 situações que justifica o uso de FIFOs nas NIs, conforme descritas a seguir:

- quando, em determinadas situações, a taxa de envio dos dados pelo EP for maior do que a taxa de envio do pacote pela rede. Nesse caso, dependendo do tamanho do pacote e da largura dos canais da rede, o tempo de processamento do EP para gerar novos dados pode ser menor do que o tempo de envio de cada pacote pela rede. Por exemplo, imaginando-se que um bloco gere dados na saída contendo ao todo 128 bits e que a largura do canal da rede seja de 16 bits. Nesse caso, tem-se 128/16 = 8 flits de dados a serem enviados pela rede mais o flit de cabeçalho, totalizando 9 flits a serem enviados pela NoC. Para o envio de todo esse pacote necessita-se considerar o tempo de envio dos 9 flits, mais o tempo de *handshake* entre os canais para cada flit e ainda o número de roteadores que o pacote precisa percorrer. Sendo assim, se no pior caso, considerando a mesma frequência de operação para toda a aplicação, se o EP gerar dados a cada 8 ciclos, não será possível garantir o envio do pacote nessa mesma taxa. Como solução, necessita-se armazenar os dados em uma FIFO para que os mesmos não sejam perdidos e também para que se consiga reduzir o atraso no envio das mensagens assim que elas puderem ser transmitidas.
- quando a taxa de envio dos dados ocorrer por rajadas ou sob uma condição de tráfego desconhecida. Tanto nesse caso, como na situação anterior, quando o tempo de processamento do núcleo variar em função do dado de entrada, não se consegue garantir a mesma taxa de processamento do EP para envio dos dados pela rede. Essa situação ainda é piorada quando a saída entrega os dados em rajadas, sendo indispensável o uso de uma solução de armazenamento, como FIFOs.
- quando vários pacotes vindos de diferentes núcleos de origem utilizarem o mesmo canal para fazer o roteamento. Essa é uma característica da NoC, ou seja, permitir o reuso dos canais da rede por diferentes EPs para realizar o roteamento dos pacotes. No entanto, como não se sabe quando os dados poderão ser enviados pela rede, já que nem sempre os canais utilizados no roteamento estarão disponíveis, um dispositivo de armazenamento para envio e/ou recebimento dos dados deve ser utilizado.

### 3.1.6 Envio de pacotes para múltiplos destinatários e recebimento de pacotes de múltiplas origens

O objetivo no uso de uma NoC como solução de interconexão é prover paralelismo e escalabilidade entre as comunicações e para isto um mesmo elemento de processamento pode se comunicar com diversos outros EPs, e essa é justamente a razão pela qual se justifica o uso de uma rede de interconexão ao invés de outras soluções. Em uma aplicação com NoCs, cada módulo pode se comunicar com vários outros EPs. A figura 3.2 apresenta o grafo de duas aplicações muito frequentemente utilizadas na literatura como *benchmarks* para análise de NoCs (BERTOZZI, 2005). Como se pode observar,

os EPs recebem dados de diferentes núcleos e cada núcleo pode também enviar dados para outros vários.




(a) (b)

Figura 3.2: Grafos das aplicações MPEG4 (a) e VOPD(b).

Para que se possa utilizar uma única interface de rede para o envio de pacotes para diferentes destinatários, a unidade empacotadora precisa identificar para quem deve ser enviado o dado e selecionar o cabeçalho que conterá as informações de roteamento na rede para que o dado chegue ao destinatário desejado.

Assim como o envio de dados para diferentes destinatários, o recebimento de pacotes de diferentes origens é uma das funções de uso de uma rede-em-chip. Da mesma forma, o objetivo é que se utilize uma única interface de rede para o recebimento de dados de diferentes transmissores e uma mesma unidade desempacotadora deverá reconstituir as palavras de dados recebidas de núcleos distintos. Para que isso seja possível, o pacote necessita ter uma identificação do núcleo transmissor que o enviou para que assim o dado seja reconstituído e repassado apropriadamente ao EP.

### 3.1.7 Sincronização de pacotes

A sincronização de pacotes pode ser necessária em uma NI quando um EP, para realizar uma determinada computação, depende simultaneamente de pacotes recebidos de nodos transmissores distintos. Dessa forma, é necessária a utilização de um circuito que identifique e organize os pacotes recebidos e os retransmita para o EP apropriadamente. Neste trabalho, duas soluções para esse problema foram abordadas e largamente discutidas no capítulo 6. Uma delas utiliza FIFOs para o armazenamento dos dados concorrentes, conforme o código do nodo transmissor, ou seja, nesse caso, cada pacote que precisa ser associado possui uma FIFO especifica. A NI realiza o controle de escrita dos dados na FIFO conforme a procedência do mesmo e verifica quando cada FIFO apresenta pelo menos um dado armazenado para proceder com o envio dos mesmos para o nodo receptor. Como nesse estudo todos os flits do pacote utilizam o mesmo roteamento (roteamento determinístico XY), é garantido que a sequência de pacotes recebida pela interface já está ordenada. Sendo assim, a NI apenas precisa assegurar que todos os pacotes provenientes de nodos distintos tenham chegado à interface de rede destinatária. A outra estratégia utiliza rótulos para identificar pacotes que apresentem a mesma sequência com que foram gerados. Assim a NI verifica o rótulo de cada pacote e retransmite os dados que precisam estar associados ao nodo receptor.

### 3.1.8 Controle de fluxo de dados

As NIs precisam ainda apresentar técnicas de controle de fluxo para o envio e recebimento dos dados. Um controle de fluxo muito frequentemente utilizado em redes-em-chip é o *handshake*. Esse mecanismo permite realizar o controle de *overflow* (estouro dos *buffers*) nas FIFOs presentes tanto na NoC como nas NIs. No caso da SoCIN, a NI só pode enviar dados para a rede se houver espaço disponível nas FIFOs dos canais de entrada da NoC, da mesma forma que o EP só pode enviar um novo dado se houver espaço disponível na FIFO da NI. No entanto, os flits são propagados pela rede e quando o núcleo receptor não puder recebê-los, somente após todos os buffers presentes nos canais utilizados para esse roteamento estiverem cheios é que os flits deixam de ser enviados. Esta estratégia de armazenamento pode reduzir significativamente a latência na entrega dos flits, já que os flits são armazenados ao longo de todo o trajeto percorrido pelo pacote (DE MICHELI, 2006).

Relacionado ao controle de fluxo, a interface de rede tem como função ainda traduzir o protocolo de rede baseado em pacotes para a interface requerida pelo IP. Os EPs podem ter sido projetados para os mais variados tipos de protocolos. No entanto, uma interface de rede não deve reproduzir todo o protocolo utilizado, por exemplo, em um barramento, pois dessa forma o objetivo de uso da NoC não estará sendo atingido. A interface de rede deve prover uma fácil interconexão dos núcleos à rede mesmo quando a NI tiver que ser adaptada para um padrão de protocolo diferente para o qual ela foi projetada.

Algumas interfaces de rede utilizam protocolos conhecidos para manter a mesma compatibilidade com os EPs, tais como, AMBA, AXI, OCP, VCI e DTL, já que muitos processadores disponíveis já foram implementados para a conexão a partir destes protocolos de comunicação. No entanto, obtendo-se uma interface de rede genérica, é possível de forma mais fácil, incorporar a interface de rede suporte para novos protocolos de comunicação (EBRAHIMI, 2009). Sendo assim, para o correto interfaceamento do protocolo do EP à NoC, dependo da complexidade do protocolo utilizado, muitas vezes pode ser necessário o uso de um *wrapper* juntamente a NI. Esse *wrapper* tem como função fazer a correta adaptação do protocolo utilizado pelo EP. Alguns trabalhos propõem propostas de *wrappers* integrados à NI para se adequarem a protocolos já conhecidos e utilizados pelos dispositivos (FERRANTE, 2008), (ATTIA, 2009), (EBRAHIMI, 2009), (RADULESCU, 2004). No entanto, não significa que uma interface de rede, para ser genérica, precise suportar os diversos protocolos existentes. O ideal é ter o projeto de uma NI genérica configurável e de fácil conexão, para que, quando a conexão a um EP exigir maiores restrições de protocolo, faz-se o uso de *wrappers* específicos que atendem a estas necessidades. Além disso, nem todos os nodos da rede necessitarão deste tipo de interface. Dessa forma, não justifica o uso de uma NI adaptada para o uso de um protocolo específico em todos os nodos da rede, pois isso resultaria em desperdício de recursos que nem sempre se fazem necessários.

## 3.2 Considerações

Nesse capítulo foram comentadas diversas situações que podem ser abordadas nos projetos de interfaces de rede. As características que foram detalhadas nesse capítulo foram exploradas e consideradas nas arquiteturas de NIs desenvolvidas nesse trabalho. A seguir serão apresentadas algumas propostas de interfaces de rede encontradas na

literatura, onde algumas destas serão utilizadas na comparação com a proposta de NI implementada.

# 4 TRABALHOS RELACIONADOS

Neste capítulo serão comentados alguns trabalhos encontrados na literatura que apresentam propostas de interfaces de redes para permitir a conexão de uma aplicação a uma NoC. Posteriormente, no capítulo 6, estas mesmas propostas serão comparadas com a arquitetura apresentada nesse trabalho, levantando vantagens e desvantagens no uso de cada uma delas.

## 4.1 Propostas de Interfaces de Redes

### 4.1.1 Interface de Rede proposta por Bhojwani

Bhojwani (BHOJWANI, 2003) apresentou uma primeira análise com relação às possibilidades de projetos de interfaces de rede. Nesse estudo, o autor considerou três alternativas de projetos que realizam a etapa de empacotamento e para isso, as seguintes situações foram consideras para esta tarefa: uma implementação em software, uma implementação em hardware, através de um processador configurável e uma implementação utilizando um módulo de hardware específico. Este trabalho não teve como objetivo apresentar uma proposta de interface de rede vantajosa sobre algum aspecto. A análise desenvolvida com esta proposta serviu apenas para que se obtivesse uma comparação no uso de recursos versus o desempenho obtido para cada uma das três situações acima consideradas. Somente a etapa de empacotamento de dados foi considerada nessa análise, as demais etapas providas por uma interface de rede não foram levantadas nesse trabalho. Como conclusão desse estudo, verifica-se que quando se utiliza uma NoC para a interconexão de uma aplicação, as interfaces de rede precisam ser implementadas em hardware, com o objetivo de apresentarem uma latência mínima na transmissão de dados entre EPs e NoC. A implementação em software para essa tarefa apresentou um atraso muito grande para envio dos dados pela rede, podendo prejudicar muito o desempenho da aplicação. Conforme o estudo apresentado por Bhojwani, a etapa de empacotamento em software necessitou de 47 ciclos, sendo que em hardware, após a obtenção de todos os flits, 1 ou alguns poucos ciclos são necessários para envio da palavra de dados ao EP.

### 4.1.2 Interface de Rede proposta por Radulescu

Radulescu (RADULESCU, 2004) propôs uma interface de rede com protocolo baseado em transação (*transaction-based protocol*) para permitir compatibilidade com outros protocolos de barramentos já existentes (AXI, OCP, DTL). No protocolo utilizado, os EPs mestres solicitam mensagens que são executadas pelos módulos

escravos, e estes respondem com uma mensagem de resposta. Na interface de rede proposta por Raduslescu, o número e o tipo de portas utilizadas podem ser definidos em tempo de projeto (configuração, mestre ou escravo). Nesta proposta o autor dividiu a interface de rede em dois blocos principais: *NI kernel* e o *NI shells*. O bloco *NI kernel* realiza as funções de empacotamento das mensagens, controle de fluxo dos dados e sincronização de relógio, já o bloco *NI shells* implementa as adaptações necessárias para atender ao protocolo do EP. A figura 4.1 ilustra a arquitetura do bloco *NI kernel*. Como apresentado na figura 4.1, várias portas podem ser acopladas a uma única interface de rede. No entanto, assim como no recebimento dos pacotes, um único *link* é utilizado para envio das mensagens para a NoC.

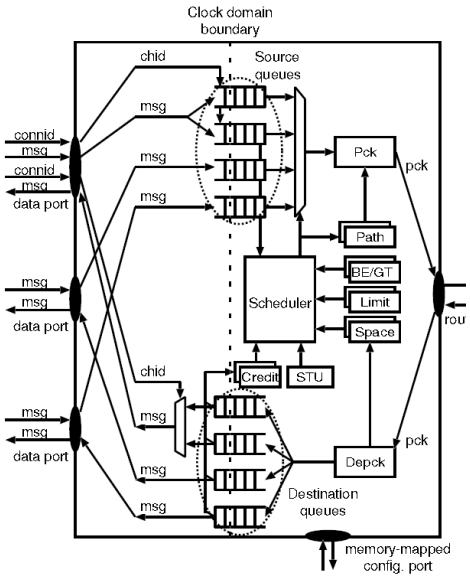



Figura 4.1: Arquitetura da interface de rede (*NI Kernel*) proposta por Radulescu (RADULESCU, 2004).

Segundo o autor, a geração dos códigos descritos em VHDL para as NIs, bem como para a NoC são obtidos através de uma descrição XML que gera a implementação automaticamente. Dessa forma, os adaptadores de protocolos que formam as NIs podem ser especificados e gerados em tempo de projeto de maneira rápida. Além disso, essa proposta apresenta um módulo de configuração que é conectado a NoC como um escravo. Sendo assim, mensagens de configuração podem ser enviadas para outras interfaces conectadas na NoC. As configurações do sistema significam manter conexões abertas ou fechadas entre os canais.

Essa solução apresenta como vantagem a possibilidade de permitir reuso dos EPs e oferece ordenamento de mensagens. No entanto, pode-se considerar como uma desvantagem dessa proposta uma penalização no desempenho, pois apresenta uma latência muito alta comparada a outras arquiteturas de interfaces de rede, podendo essa arquitetura apresentar uma latência de até 10 ciclos.

### 4.1.3 Interface de Rede proposta por Singh

Uma proposta de interface de rede encontrada na literatura que tem como objetivo ser genérica e permitir uma rápida conexão foi apresentada por Singh (SINGH, 2006). Singh propôs uma interface de rede com os seguintes blocos: *Generic Core Interface*

(GCI), *Packet Maker* (PM) e *Packet Disassembler* (PD), duas memórias e duas FIFOs assíncronas. A figura 4.2 apresenta o modelo de arquitetura proposto por Singh. O módulo GCI fica situado entre a interface de rede e um *wrapper* específico conectado ao EP. Dessa forma, o GCI é responsável por realizar um controle no envio dos dados juntamente com o *wrapper* conectado ao EP.

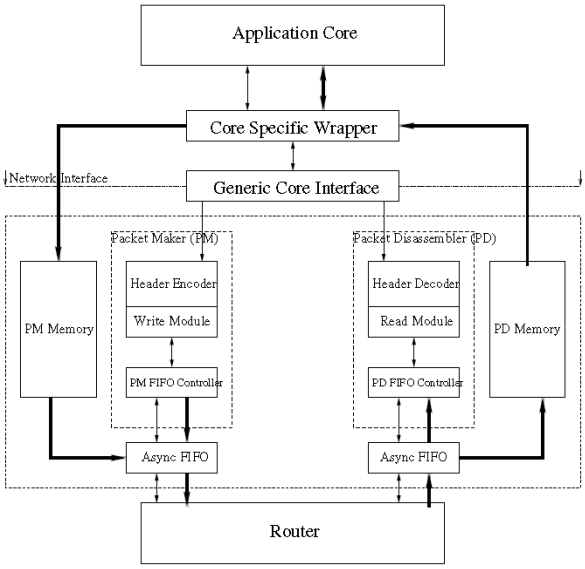



Figura 4.2: Arquitetura da interface de rede proposta por Singh (SINGH, 2006).

O bloco PM tem por função criar o cabeçalho da mensagem, adicionando informações de roteamento e controle ao pacote, particionar o pacote em unidades menores a serem enviadas pela rede, escrever os pacotes de dados na memória PM e convertê-los em flits, antes de repassá-los para a FIFO assíncrona. O cabeçalho é definido por este bloco de acordo com as informações enviadas pelo GCI que inclui endereçamento do núcleo destinatário e informações de controle. O endereço dos nodos destinatários é mantido em uma tabela. Tanto o cabeçalho, como os pacotes de dados são escritos na memória.

O bloco PD realiza o processo inverso ao bloco PM. Primeiramente os dados da FIFO assíncrona são lidos e todas as informações de controle são extraídas a fim de obter a mensagem total. Após esse processo, os dados são salvos em uma memória para posteriormente serem entregues ao EP.

O interessante dessa proposta de NI é que ela foi projetada pensando na conexão com *wrappers* específicos que possam ser necessários na conexão de alguns módulos. Sendo assim, essa proposta não apresenta um adaptador para um único protocolo, como ocorre em algumas propostas de NIs, e dessa forma, permite que diferentes *wrappers* possam ser acoplados a NI.

No entanto, essa arquitetura utiliza um número muito grande de recursos de armazenamento, sendo que são esses recursos os principais responsáveis pelo consumo de potência de uma arquitetura. Essa arquitetura ainda necessita armazenar nas memórias o cabeçalho do pacote, sendo este armazenamento desnecessário, já que o cabeçalho poderia ser gerado diretamente no momento de envio do pacote. No recebimento do cabeçalho, apenas as informações ainda necessárias pela interface de

rede poderiam ser armazenadas em registradores, descartando as demais informações que foram utilizadas apenas para o roteamento dos pacotes, e que não são mais úteis nessa fase.

Outra questão é com relação à falta de clareza da figura 4.2 quanto à comunicação entre os próprios blocos desenvolvidos. Segundo o autor, o bloco responsável por dividir o pacote salvo na memória em flits é bloco *PM FIFO Controller*, no entanto, esse bloco não apresenta conexão com a memória PM para recebimento do pacote armazenado, tornando um pouco confuso o entendimento da mesma.

A figura 4.3 apresenta o formato do cabeçalho utilizado por essa arquitetura. Neste trabalho, o cabeçalho apresenta 48 bits, embora, segundo o autor, os pacotes possam ser enviados em flits com largura parametrizável.

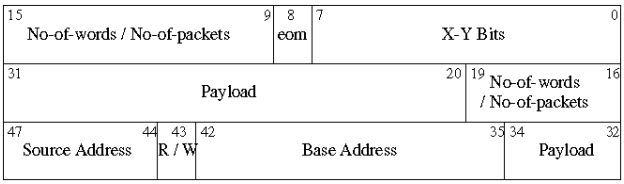



Figura 4.3: Formato do cabeçalho do pacote da arquitetura de NI proposta por Singh (SINGH, 2006).

Nessa arquitetura o número de bits de dados presente no pacote é informado no campo de *payload* do cabeçalho, sendo que, em um único pacote podem ser enviados até 32767 bits de dados. Além disso, como a NoC utilizada pelo autor apresenta roteamento X-Y, um pacote com essa profundidade de dados reservaria uma rota da rede por muito tempo, impossibilitando que outros pacotes, que utilizam os mesmos canais, possam ser enviados e, dessa forma, prejudicando o desempenho do sistema.

As interfaces de rede desenvolvidas nesse trabalho foram utilizadas na conexão de um codificador JPEG e para isso 4 módulos foram utilizados: um processador, um módulo JPEG, uma unidade de memória e um módulo de UART.

Por fim, essa arquitetura não apresentou resultados de síntese, impossibilitando uma comparação mais direta com a arquitetura desenvolvida.

### 4.1.4 Interface de Rede proposta por Ferrante

A proposta apresentada por Ferrante (FERRANTE, 2008) levanta a possibilidade de dividir os recursos de uma interface de rede com mais de um EP, reutilizando também os recursos de armazenamento, que são os que mais impactam na área e potência de uma NI. Esta proposta utiliza NIs adaptadas ao protocolo OCP, e para isso, a interface é dividida em dois sub-módulos: um para as mensagens de solicitação e o outro para as mensagens de resposta. Porém, como os recursos são divididos entre mais de um EP, é necessário se fazer o uso de um árbitro juntamente às interfaces para a definição de quem pode acessar a NI. A figura 4.4 apresenta o árbitro utilizado nessa arquitetura. No entanto, fica a dúvida, nesta proposta, de quão vantajoso é dividir os recursos da NI da forma como apresentada por Ferrante, já que um conjunto de circuitos a mais são necessários para permitir o correto envio e armazenamento dos dados. Além disso, conforme apresentado na figura 4.4, se muitos EPs puderem ser conectados a uma mesmo canal do roteador, então talvez o uso de uma NoC não seja necessário e, neste caso, o uso de um barramento possa ser justificável. Partindo do princípio de utilização

de uma NoC como dispositivo de interconexão, a divisão dos recursos de uma interface de rede para vários elementos de processamento, como apresentado na figura 4.4, dificilmente se justificaria em termos de desempenho.

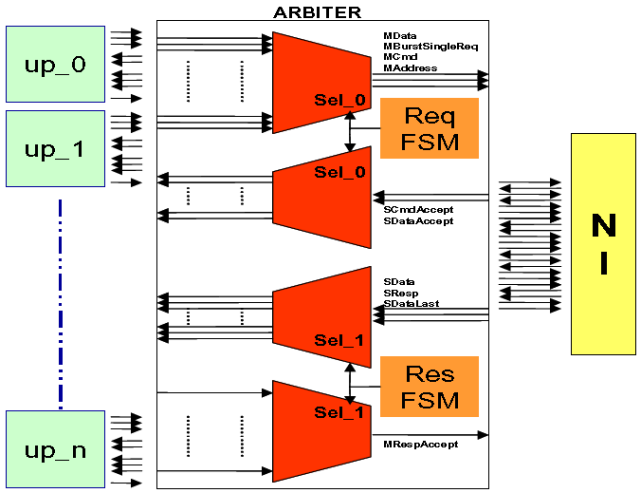



Figura 4.4: Árbitro utilizado para definir EP que utilizam os recursos da NI proposto por Ferrante (FERRANTE, 2008).

### 4.1.5 Interface de Rede proposta por Lee

Lee (LEE, 2008) propôs uma interface de rede e, para o desenvolvimento da mesma, o autor utilizou o processador OpenRISC como EP. Esta proposta apresenta unidade de empacotamento, unidade de desempacotamento e circuito para permitir a adaptação a diferentes protocolos de comunicação. O autor definiu uma memória para o armazenamento de todos os registradores de controle que podem ser utilizados na adaptação de um protocolo de barramento utilizado pelo EP.

A unidade de empacotamento constrói o cabeçalho do pacote e converte os dados armazenados na memória em flits. Nos dados enviados para a rede há um flit de controle com registradores e controles de DMA. O cabeçalho é formado a partir de informações fornecidas pelos registradores, onde é informado o endereço de destino, identificação do dado, número de flits e nível de serviço. O pacote também pode enviar um sinal de interrupção para o OpenRISC.

Esta proposta, dentre as demais apresentadas, pareceu ser a mais genérica, já que prevê a utilização de diferentes tipos de controles para NoCs e possibilita a conexão de diversos *wrappers* que façam a adaptação de protocolo do EP à rede. No entanto, esta generalidade tem um preço alto a pagar. Para permitir essas possibilidades de uso da interface, o autor utiliza um número grande de registradores para armazenar todas as características necessárias para permitir a generalidade na comunicação entre os EPs. Além disso, é necessário definir diversos parâmetros nas tabelas utilizadas juntamente as NIs. Nestas tabelas estão previstas informações como tipo de EP, se ele é mestre ou escravo, sinais de controle de entrada, sinais de *status* vindo do EP, sinal de modo, que definem o tipo de transmissão, sinais de interface de entrada e saída do EP que serão utilizadas na comunicação entre EPs e interface de rede, dentre outros.

### 4.1.6 Interface de Rede proposta por Attia

Attia (ATTIA, 2009) também apresentou uma interface de rede para NoCs. No entanto, essa interface também foi desenvolvida para conexão a partir do protocolo OCP. O uso de uma interface de rede específica para um protocolo apresenta como desvantagem o fato de nem todos os módulos utilizarem essa mesma interface, representando custos de recursos desnecessários nestes casos.

Nesta proposta, dois adaptadores para NoCs foram desenvolvidos, um para as interfaces mestres e outro para as interfaces escravas. Neste trabalho foram apresentadas propostas de interfaces de rede distintas para mensagens de solicitação e de resposta entre mestres e escravos.

Essa proposta apresenta como desvantagem o fato de apresentar largura de pacote da rede fixa. A largura de dados do canal definida para essa arquitetura é igual a 32 bits, no entanto, os primeiros dois flits são utilizados como cabeçalho. Os 16 primeiros bits do primeiro flit definem o roteamento da mensagem e os demais bits são utilizados para permitir a adaptação ao protocolo OCP. A figura 4.5 apresenta o formato do pacote utilizado nesse trabalho.

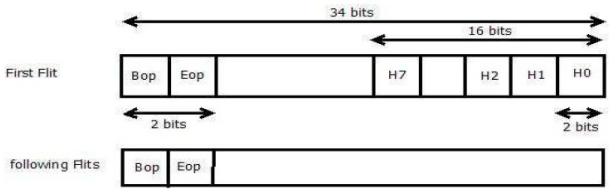



Figura 4.5: Formato do pacote da arquitetura de NI proposta por Attia (ATTIA, 2009).

Essa arquitetura apresenta também como desvantagem o uso de adaptadores complexos e específicos para cada caso, podendo prejudicar o desempenho da aplicação e comprometer os objetivos principais de uso de uma rede-em-chip.

### 4.1.7 Interface de Rede proposta por Ebrahimi

Ebrahimi (EBRAHIMI, 2009) também propôs uma interface de rede e a sua proposta é baseada no protocolo AXI. Este trabalho apresenta as mesmas limitações da proposta apresentada por Attia, já que a interface de rede só pode ser utilizada em EPs que utilizem o mesmo protocolo definido nos projetos das NIs. Neste protocolo os núcleos também são classificados como mestres e escravos. Os EPs mestres iniciam transações de leitura e escrita e os escravos recebem e executam cada transação.

A principal motivação desta proposta é fornecer uma alocação dinâmica de buffer conforme a variação do tamanho do pacote, já que mensagens de resposta e solicitação podem apresentar tamanhos diferentes. Além disso, esta NI propõe um circuito de reordenamento dos pacotes, permitindo a utilização de algoritmos de roteamento dinâmico. No entanto, a arquitetura foi validada utilizando o roteamento determinístico X-Y. O buffer utilizado nessa proposta apresenta a mesma largura de flits definido pela rede e cada flit recebido é armazenado a partir de uma tabela de reordenamento. No entanto, esta proposta apresenta forte relação com o protocolo AXI. Como comentado para as demais propostas que se baseiam no uso de um protocolo específico, o uso da NI pode ser comprometido quando o EP apresentar um protocolo diferente ao apresentado pela proposta de NI que não exija as funções do circuito de NI por apresentar uma interface mais simples. Dessa forma, o uso de recursos a mais fornecidos pela NI pode

não contribuir na conexão de determinados EPs, além de apresentar elevado consumo de potência e área. Além disso, esta proposta não foi validada com núcleos reais. Para verificar o funcionamento da NIs foram utilizados apenas simuladores de processadores e memórias a partir de tráfegos sintéticos. Também não foram informados valores de síntese da arquitetura impedindo uma comparação com a arquitetura proposta.

## 4.2 Considerações

Este capítulo apresentou algumas propostas de interfaces de rede encontradas na literatura. A partir desse estudo, pôde-se perceber diferentes implementações para atender a diferentes necessidades. No entanto, a maioria das propostas encontradas na literatura foi validada utilizando pouquíssimos EPs, normalmente com pouca comunicação entre os núcleos, ou seja, com o envio de pacotes apenas de um receptor para um transmissor específico. Sendo assim, algumas situações podem não ter sido analisadas, como, por exemplo, o envio de pacotes por um mesmo núcleo para diferentes nodos da rede, bem como o recebimento de pacotes por um núcleo vindos de vários outros nodos da rede.

No próximo capítulo será apresentado o decodificador de vídeo, que foi a aplicação utilizada como estudo de caso deste trabalho. Tendo-se conhecimento da arquitetura da NoC SoCIN e das arquiteturas desenvolvidas para compor o decodificador de vídeo segundo o padrão H.264, é possível definir as interfaces de rede necessárias para conectar estas arquiteturas.

No capitulo 6 as interfaces de rede descritas neste capítulo serão comparadas com a implementação de NI desenvolvida nesse trabalho.

# 5 CONCEITOS DE COMPRESSÃO DE VÍDEO E O PADRÃO H.264/AVC

Neste capítulo serão introduzidos alguns conceitos sobre compressão de vídeo digital e a decodificação de vídeo no contexto do padrão H.264/AVC, dos quais os módulos que foram utilizados para compor este trabalho estão inseridos. Uma abordagem mais extensa sobre compressão de vídeo pode ser encontrada em (RICHARDSON, 2003) e (PURI, 2004). Entretanto, uma descrição mais detalhada sobre o padrão H.264/AVC pode ser obtida através da instituição que o definiu (ITU, 2007).

## 5.1 Representação do Vídeo Digital

O vídeo digital é representado por uma sequência de imagens capturadas a intervalos regulares de tempo. A captura de cada imagem define a amostragem espacial do vídeo. A resolução de uma imagem é definida de acordo com o número de pontos (*pixels*) que uma imagem é representada. Sendo assim, quanto mais pixels uma imagem apresentar, maior será a sua resolução. As imagens são organizadas em alguns formatos pré-definidos, tais como o CIF (*Common Intermediate Format*) com 352x288 *pixels,* o QCIF (*Quarter Common Intermediate Format*), com 176x144 *pixels*, o SQCIF (*Sub* QCIF) com 128x96 pixels e o 4CIF com 704x576 *pixels*.

A amostragem temporal do vídeo é definida de acordo com o intervalo de captura de cada imagem. Sendo assim, quanto maior for a taxa de captura, mais suave será o movimento da cena de um quadro para o outro. Porém, neste caso, é requerido que mais amostras sejam capturadas e armazenadas. As taxas de captura usualmente utilizadas estão na faixa de 15 a 60 imagens por segundo (RICHARDSON, 2003).

Cada *pixel* de uma imagem de vídeo é representado por três componentes de cor. O espaço de cores mais utilizado para representar imagens digitais é o RGB (*Red, Green, Blue*) que utiliza três matrizes distintas para representar cada uma das cores primárias captadas pelo sistema visual humano: vermelho, verde e azul. No entanto, existe um outro espaço de cores que é mais apropriado para a compressão de vídeo, chamado de YCbCr (*Luminance, Chrominance blue, Chrominance red)*. Os sistemas de compressão de vídeo adotam esse espaço de cores por que as informações de cor (crominância) estão completamente separadas da informação de brilho da imagem (luminância), permitindo a aplicação de técnicas de compressão distintas para luminância e crominância, o que não ocorre quando se usa o espaço de cores RGB.

Outro fator determinante no uso de técnicas de compressão no espaço de cores YCbCr é a possibilidade de utilizar sub-amostragens nas amostras de crominância. Essa alternativa é utilizada pelo fato de que o sistema visual humano é mais sensível ao

brilho do que à crominância, desta forma, o uso de sub-amostragens não representa impacto visual de qualidade da imagem. Além disso, o uso desta alternativa permite que se obtenha uma redução significativa dos dados para representação do vídeo.

Um exemplo de sub-amostragem frequentemente utilizada é 4:2:0, o que significa que a cada 4 amostras de luminância estão associadas uma amostra de Cb e uma de Cr. Utilizando-se essa sub-amostragem já é obtida uma compressão de 50% em relação à 4:4:4 (RICHARDSON, 2003).

## 5.2 Conceitos Básicos de Compressão de Vídeo

As técnicas de compressão de vídeo classificam-se como sem perdas ou com perdas de informação. Na codificação sem perdas de informação, os dados podem ser reconstruídos de forma idêntica aos dados originais. No entanto, tais técnicas atingem baixas taxas de compressão, sendo seu uso somente viável quando não for admissível nenhuma perda de informação (RICHARDSON, 2003).

Já, quando se faz uso de técnicas de compressão com perdas de informação, na reconstrução de dados codificados, é possível obter alguns dados que diferem do original. No entanto, essas técnicas atingem altas taxas de compressão. Como o olho humano apresenta limitações visuais, essas técnicas são utilizadas para compressão de imagens e vídeos de forma que a reconstrução da imagem tenha uma diferença visual imperceptível ao olho humano em comparação com a imagem original.

Os vídeos digitais também possibilitam o uso de técnicas de compressão que estão relacionadas às redundâncias de informações existentes em sua representação. Existem três tipos principais de redundância: espacial, temporal e entrópica:

- Redundância Espacial: representa a correlação existente entre os *pixels* presentes em um mesmo quadro.
- Redundância Temporal: representa a correlação existente entre os *pixels* dos quadros temporalmente próximos em um vídeo.
- Redundância Entrópica: representa à probabilidade de ocorrência dos símbolos codificados. É possível utilizar técnicas que representam os símbolos do vídeo codificado com um número menor de bits, reduzindo o volume de informações.

Os padrões de compressão de vídeo combinam todas estas técnicas para permitir que elevadas taxas de compressão sejam atingidas. Na próxima seção serão apresentadas as técnicas que compõem o decodificador de vídeo conforme o padrão H.264/AVC.

## 5.3 Introdução ao padrão H.264/AVC

### 5.3.1 Padrão H.264/AVC

O padrão H.264/AVC apresenta a possibilidade de utilizar quatro tipos de perfis: *Baseline*, *Main, Extended* e *High*. Cada um dos perfis apresenta um conjunto de funções de codificação e especifica as características que o codec precisa suportar em cada uma delas (ITU, 2003).

O perfil *Baseline* é direcionado a aplicações como videotelefonia, videoconferência e vídeo sem fio. O perfil *Main* é adequado para transmissão de televisão e armazenamento de vídeo, já o perfil *Extended* é mais focado para aplicações em *streaming* de vídeo e melhora a robustez a erros de transmissão. Os perfis *Baseline*, *Main* e *Extended* apresentam a resolução entre os elementos de cores fixas e igual a 4:2:0 para o espaço de cores YCbCr. Os perfis *High* possuem algumas características com suporte para vídeos com resoluções mais elevadas (SULLIVAN, 2004). As funções suportadas por cada um destes perfis estão resumidas na Figura 5.1. Maiores informações de cada perfil podem ser obtidas em (ITU, 2003), (SULLIVAN, 2004).

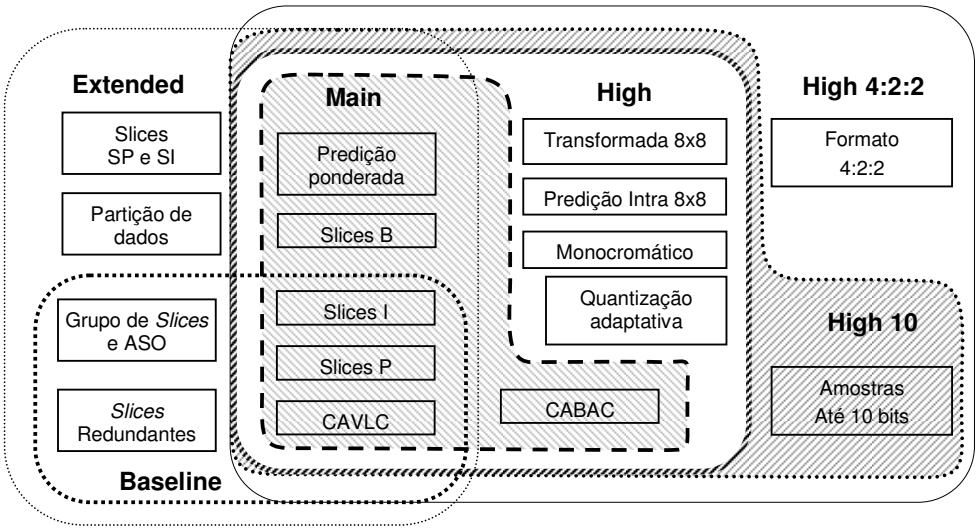



Figura 5.1: Perfis do H.264/AVC (ROSA, 2009b).

A cada quadro é associado um número que define sua ordem de exibição, chamado de *Picture Order Count* (POC). A codificação dos quadros não precisa seguir a mesma ordem de exibição. Cada quadro pode ser codificado a partir de um *slice* ou de vários *slices*. Um *slice* é um conjunto inteiro de macroblocos ordenados sequencialmente da esquerda para a direita e de cima para baixo dentro de uma imagem (ITU, 2007). Cada macrobloco contém 16x16 amostras de luminância (Y) e duas componentes de crominância associadas (Cb e Cr). Cada componente de crominância é formado por um bloco de 8x8 amostras na sub-amostragem de cor 4:2:0, sendo que, cada amostra, tanto de luminância como de crominância, possui 8 bits.

### 5.3.2 Formato do Vídeo Codificado

O padrão H.264/AVC classifica a informação do vídeo em dois níveis de abstração, chamados de camada de vídeo codificado (VCL – *Vídeo Coding Layer*) e camada de abstração de rede (NAL – *Network Abstration Layer*) (RICHARDSON, 2003).

No processo de codificação de vídeo, os dados de saída estão na camada VCL e eles são formados por uma sequência de bits que representam os dados do vídeo codificado. Estes dados são mapeados para unidades NAL antes da transmissão ou armazenamento. Uma sequência de vídeo codificado é representada por uma sequência de unidades NAL que podem ser transmitidas sobre uma rede baseada em pacotes, via um link de transmissão de *bitstream* ou ainda armazenados em um arquivo (PURI, 2004).

## 5.4 Decodificador H.264/AVC

O decodificador H.264/AVC é composto pelos blocos de predição intra-quadros (INTRA), predição inter-quadros (INTER), transformadas inversas (IT), quantização inversa (IQ), filtro redutor do efeito de bloco e decodificação de entropia, conforme apresentado no diagrama de blocos da figura 5.2. Os demais módulos apresentados na figura 5.2 são blocos que estão implícitos ao processo de decodificação de vídeo e que são necessários para permitir o envio correto dos dados ou sinais de controle entre os demais módulos. Na figura 5.2, as linhas pontilhadas indicam sinais de controle e as linhas contínuas indicam dados. Um multiplexador (*SEL*) indica o tipo de predição que foi utilizada na etapa de codificação de vídeo e este controle é informado no *bitstream*. Além destes módulos é necessário utilizar um módulo *parser* que é responsável por enviar dados de controle aos demais módulos. Cada um dos módulos que compõe o decodificador de vídeo será brevemente apresentado nas próximas subseções.

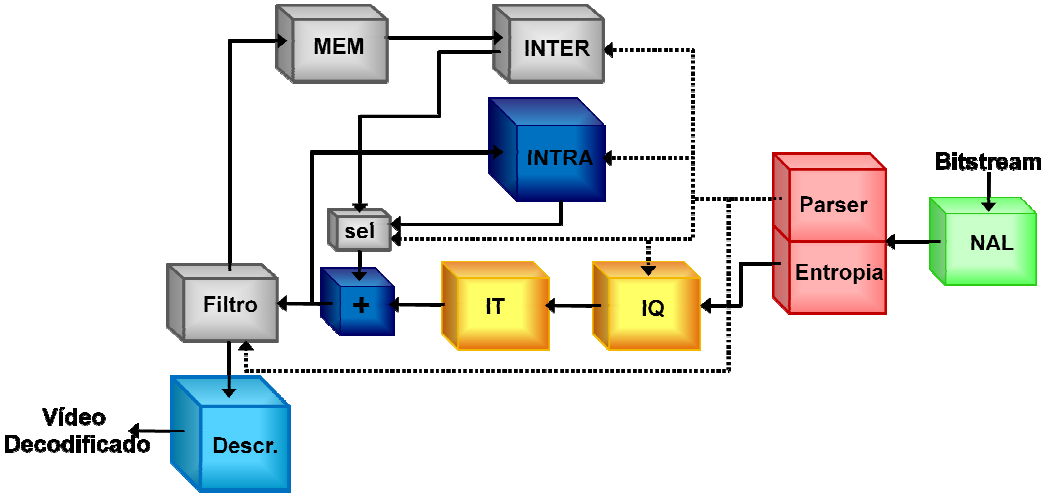



Figura 5.2: Diagrama em blocos de um decodificador H.264/AVC.

Para maior entendimento do funcionamento do decodificador de vídeo será visto primeiramente como ocorre o processo inverso, ou seja, a codificação de vídeo. De forma resumida, o funcionamento de um codificador de vídeo H.264/AVC ocorre da seguinte forma: cada macrobloco a ser codificado é subtraído de um macrobloco predito pelo módulo INTRA ou pelo módulo INTER. A definição pelo tipo de predição utilizada (INTER ou INTRA) é definida por um modo de decisão. Um macrobloco de resíduos é gerado como resultado da subtração entre os macroblocos original e o predito. Este resultado é particionado em blocos 4x4 e processado pelos blocos de transformada e quantização, gerando os coeficientes quantizados. Os coeficientes quantizados, bem como as demais informações de controle (indicadores de modo de predição e vetores de movimento, por exemplo) são enviados para a etapa de codificação de entropia, que explora a redundância estatística destas informações e gera uma sequência de bits que forma o macrobloco codificado. O conjunto de todos os macroblocos codificados e organizados em seus quadros forma o vídeo codificado (uma revisão sobre este processo pode ser vista em DINIZ, 2009).

A sequência de bits que representa o vídeo codificado é denominada *bitstream*. No decodificador, primeiramente, o *bitstream* codificado é recebido pelo decodificador de entropia e *parser*. No *bitstream* estão contidas informações de controle essenciais para predição e reconstrução do vídeo, tais como o tipo do macrobloco, modos de predição e vetores de movimento. Os resíduos quantizados são processados pelos blocos de quantização inversa (IQ) e pelas transformadas inversas (IT). A predição sinalizada no *bitstream* é realizada pelos blocos de INTER ou INTRA, sendo que o macrobloco predito é somado ao resíduo resultante da etapa das transformadas inversas. A imagem reconstruída é posteriormente filtrada para ser exibida e armazenada para referências futuras. A figura 5.3 apresenta a formação do quadro reconstruído pela adição dos resíduos à predição.

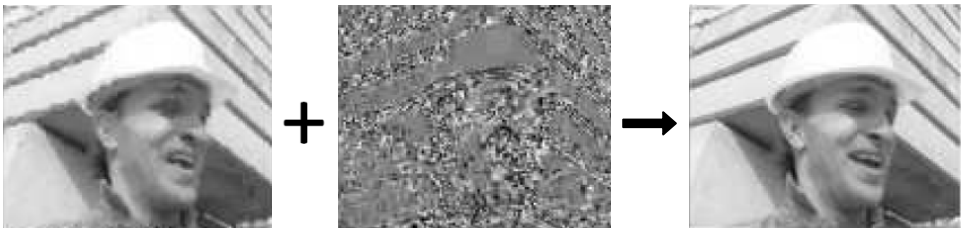



Figura 5.3: Reconstrução de um quadro obtido pela adição do resultado de resíduo ao quadro predito (PEREIRA, 2009).

### 5.4.1 Unidades NAL

As unidades NAL, conforme comentado anteriormente, apresentam os dados de vídeo encapsulados para permitir a transmissão do *bitstream* via pacotes de rede. O bloco NAL controla o recebimento dos *bitstreams*, remove os bits extras utilizados para completar o pacote e as bordas dos pacotes (identificando início e fim de cada elemento sintático) e os armazena para serem lidos pelo decodificador de entropia e *parser*.

### 5.4.2 Decodificação de Entropia e *Parser*

Na decodificação de entropia, dois métodos são possíveis: a decodificação por códigos de comprimento variável (VLD, do inglês *Variable Length Decoding*) e a decodificação binária aritmética (BAD, do inglês *Binary Arithmetic Decoding*). Na etapa de codificação, cada bloco gera um conjunto de informações para posterior decodificação, como, por exemplo, modos de predição, tipo do macrobloco e os valores dos resíduos quantizados. Para cada uma destas informações é gerado um elemento sintático. O bloco de codificação de entropia trabalha sobre os elementos sintáticos para extrair estatísticas que correlacionam estes dados para representá-los com menor número de bits. Na decodificação de entropia, o processo inverso é realizado, restituindo os elementos sintáticos que contém informações necessárias aos demais blocos.

O método VLD combina códigos de comprimento fixo, códigos *Exp-Golomb* e decodificação de comprimento variável adaptativa ao contexto (CAVLD). O CAVLD é aplicado somente aos blocos de resíduos quantizados (RICHARDSON, 2003). Ao invés do CAVLD, outro método que utiliza decodificação aritmética binária adaptativa ao contexto (CABAD) pode ser empregado. O CABAD decodifica os resíduos quantizados e todas as informações relativas à codificação de macroblocos (modos de predição,

vetores de movimento, dentre outros). Os códigos *Exp-Golomb* são códigos de comprimento variável que atribuem sequências de bits menores a elementos mais frequentemente utilizados. O *Exp-Golomb* é usado para a decodificação dos dados de controle e predição, enquanto que o CAVLD é usado para a decodificação dos resíduos.

A decodificação aritmética binária adaptativa ao contexto (CABAD) é baseada em uma decodificação associada à seleção de modelos de probabilidade de ocorrência para cada elemento sintático de acordo com o contexto de codificação de cada elemento. O CABAD é uma alternativa ao CAVLD e *Exp-Golomb* para o processo de codificação de entropia.

O *parser* interpreta as unidades NAL, composto pelo cabeçalho do s*lice* e os pacotes de dados, identifica e decodifica os elementos sintáticos, separando os dados referentes ao módulo de predição e os dados referentes aos resíduos. Os elementos sintáticos do *bitstream* podem incluir códigos fixos ou codificados com *Exp-Golomb,* CAVLC ou CABAC. A decodificação do *slice* pelo *parser* compreende no processo de determinação da ordem de contagem das imagens para a exibição, gerenciamento e marcação dos quadros de referência e sequenciamento do processo de decodificação. Este último refere-se ao particionamento das informações de controle para os módulos de predição e para o módulo de quantização e transformadas inversas. Um resumo sobre estes conceitos e sobre o funcionamento do CABAD pode ser encontrado em (DEPRÁ, 2009) e uma explicação simplificada sobre o CAVLD e *parser* podem ser obtidas em (PEREIRA, 2009). A descrição completa destes módulos e a definição da norma são encontradas em (ITU, 2007).

### 5.4.3 Predição Inter-Quadros

Na codificação, um módulo de estimação de movimento (ME) localiza e associa uma área, a partir dos quadros de referência, para a geração de um vetor de movimento. Este vetor indica qual macrobloco, nos quadros de referência, mais se assemelha ao macrobloco atual. Associada a essa decisão, conforme o vetor de movimento, a predição inter-quadros localiza os blocos utilizados para estimação. A localização dos blocos é feita a partir de uma memória que contém os quadros que estão sendo utilizados como referência. A predição inter-quadros deve ser capaz ainda de tratar múltiplos tamanhos de partições de macrobloco, interpretar e tratar corretamente os vetores da predição, reconstruir os macroblocos com uma precisão de até ¼ de pixel, dentre outras funções cabíveis a este módulo. No processo de decodificação de vídeo, a predição inter-quadros receberá o vetor de movimento definido na codificação de vídeo e encontrará, nos quadros de referência, o macrobloco equivalente. Posteriormente, esse macrobloco é adicionado aos resíduos para formar o vídeo reconstruído (uma revisão sobre o módulo de predição INTER pode ser obtida em AZEVEDO, 2006).

### 5.4.4 Predição Intra-Quadros

Esta predição é atribuída tanto para as amostras de luminância (Y) quanto para as amostras de crominância (Cb e Cr) e a predição de um macrobloco é gerada a partir de amostras já reconstruídas que estão nas fronteiras do macrobloco em predição. Na predição intra-quadros dois tipos de predição para as amostras de luminância são possíveis: predição de um macrobloco (16x16 amostras), onde 4 modos são permitidos e predição de um bloco 4x4, onde 9 modos são previstos. A predição das amostras de crominância é realizada diretamente sobre blocos 8x8 e, neste caso, 4 modos de predição são possíveis.

Estes modos procuram reproduzir diferentes padrões espaciais contidos em um quadro do vídeo. Quando um macrobloco predito é muito similar ao macrobloco original, pode-se reduzir significantemente o valor dos resíduos a serem codificados e, por consequência, o número de bits de codificação do macrobloco em questão. A predição intra-quadros é aplicada no domínio espacial e ocorre com base nas amostras vizinhas. Uma revisão sobre este módulo pode ser encontrada em (STAEHLER, 2006) e em (DINIZ, 2009).

### 5.4.5 Quantização Inversa (IQ)

Os cálculos do bloco de quantização dependem do tipo de predição e se o elemento é de luminância ou crominância. Em resumo, a quantização realiza uma operação de multiplicação da amostra de entrada por uma constante. Após esta etapa, é efetuada uma soma deste resultado por uma outra constante e por útlimo, é realizado um deslocamento desta soma controlado por uma terceira constante.

A etapa de quantização inversa no decodificador é aplicada após a etapa de decodificação de entropia. Com relação aos módulos do padrão, este é o único que gera perdas no processo de codificação do vídeo. O fator de quantização controla a amplitude desta perda, variando a relação taxa-distorção do vídeo na saída que depende do parâmetro de quantização QP (*Quantization Parameter*). Os valores de QP variam de 0 a 51. Um resumo sobre o módulo de quantização pode ser obtido em (AGOSTINI, 2007).

### 5.4.6 Transformadas Inversas (IT)

O padrão H.264/AVC define o uso da transformada *Hadamard* inversa para blocos 4x4 formados por coeficientes DC de luminância quando o macrobloco tiver sido predito usando o modo Intra 16x16. A transformada *Hadamard* inversa também é utilizada para blocos 2x2 formados por coeficientes DC de crominância. Para os demais blocos 4x4, é aplicada a transformada inversa baseada na IDCT 2-D (*Inverse Discrete Cosine Transform*).

Quando a predição escolhida for Intra 16x16, a transformada Hadamard 4x4 inversa é aplicada sobre os coeficientes DC dos blocos de luminância, enquanto que, para os blocos de crominância, é aplicada a Hadamard 2x2 inversa sobre os blocos 2x2 de coeficientes DC de crominância, tanto para Cb quanto para Cr. Uma revisão sobre a aplicação das transformadas utilizadas pelo padrão H.264 pode ser obtida em (AGOSTINI, 2007).

### 5.4.7 Filtro Redutor de Efeitos de Bloco

Como o processo de decodificação é realizado em blocos, é possível perceber-se, na imagem reconstruída, efeitos de borda dos blocos em função do processamento realizado. O objetivo deste filtro é suavizar o efeito de borda nos blocos dos quadros reconstruídos antes deles serem usados para a predição de um macrobloco pelo módulo INTER e antes deles serem exibidos. A operação de filtragem é feita sobre as bordas verticais e horizontais dos blocos 4x4 de um macrobloco já reconstruído (RICHARDSON, 2003). Com este filtro obtém-se uma melhora subjetiva na qualidade da imagem reconstruída, principalmente para vídeos com baixa taxa de bits.

## 5.5 Decodificador de Vídeo Mínimo

Na seção anterior, foram apresentados os módulos que compõem um decodificador de vídeo conforme as especificações do padrão H.264/AVC. O grupo de pesquisa de TV digital da UFRGS tem desenvolvido soluções arquiteturais para os diversos módulos que compõem o decodificador de vídeo (PEREIRA, 2009), (AZEVEDO, 2006), (STAEHLER, 2006), (AGOSTINI, 2007), (SILVA, 2009), (DEPRÁ, 2008), (ROSA, 2009). Alguns destes módulos foram integrados, constituindo uma versão mínima do decodificador de vídeo H.264. Nesta versão, foram utilizados somente os módulos conforme apresentado na figura 5.4 (PEREIRA, 2009). Todos os módulos desenvolvidos foram descritos em VHDL. Na sequência deste trabalho, planeja-se adicionar os demais módulos que compõem o decodificador de vídeo H.264 a esta primeira versão arquitetural. O objetivo do grupo de pesquisa de TV digital é obter um set-top box ao final do projeto.

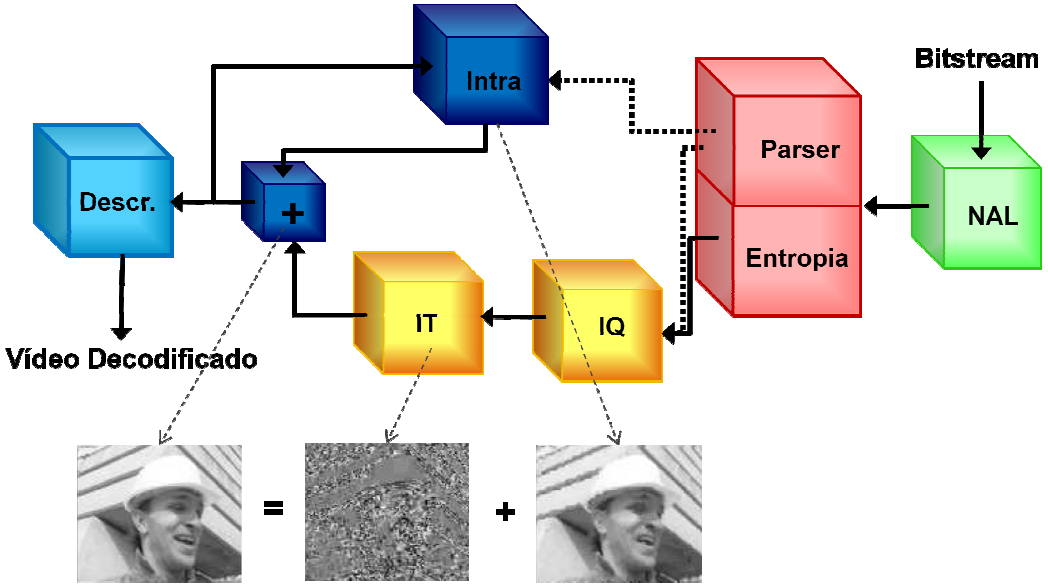



Figura 5.4: Decodificador de vídeo H.264 mínimo utilizado neste trabalho.

O decodificador de vídeo mínimo suporta decodificação de entropia CAVLD, quantização inversa, transformadas inversas e predição intra-quadros. Os demais módulos, como CABAD, predição inter-quadros, e filtro redutor de efeito de blocos serão posteriormente incorporados à arquitetura.

Conforme já comentado, inicialmente o *bitstream* decodificado é recebido em pacotes de um *byte* pelo módulo da unidade NAL. Os limites de cada pacote são identificados e os elementos sintáticos são armazenados a cada 32 bits em uma FIFO que posteriormente será lida pelos blocos *parser* e decodificação de entropia. O *parser* decodifica os elementos sintáticos e envia os resíduos e informações de predição para os seus respectivos buffers. Os dados que são armazenados pelo *parser* na FIFO de predição contém 71 bits que incluem informações sobre o tamanho do bloco utilizado para predição (blocos de tamanho 4 ou 16), o modo usado se a predição for por

macroblocos, o modo usado caso a predição seja feita por blocos 4x4, e ainda o modo de predição para os elementos de crominância.

Os resíduos, juntamente com o Qp e a informação de tamanho do bloco de predição, são decodificados pelo CAVLD e repassados para o bloco de Quantização Inversa e Transformadas Inversas (ITIQ). A predição intra-quadros gera as informações de predição enquanto que o bloco ITIQ decodifica os resíduos. Por último, as amostras preditas são adicionadas aos resíduos, obtendo-se o vídeo reconstruído. Para a predição das próximas amostras, o vídeo reconstruído deve ser enviado para o módulo INTRA que servirá como vizinhança para a predição seguinte.

O somador é um módulo totalmente combinacional, composto de 4 somadores de 9 bits. As amostras são reduzidas a 8 bits sem sinal e 4 amostras válidas por vez são geradas. As amostras reconstruídas, antes de serem utilizadas para exibição do vídeo, ainda são enviadas para o módulo de *descrambler*. O *descrambler* tem como função reordenar as amostras, pois na saída do somador elas estão em ordem de duplo Z. Para a exibição do vídeo, elas precisam ser ordenadas formando uma linha de *pixels*. O *descrambler* armazena duas linhas de macroblocos e reordena-as em linhas de pixels a serem exibidas como vídeo final. Cada linha de macrobloco possui 16 linhas de pixels, e dessa forma, é utilizada uma memória para armazenar até 32 linhas de macrobloco. Para reordenamento das amostras em linha de pixel, pelo menos uma linha de macroblocos precisa estar armazenada na memória do *descrambler* para que as linhas de pixel da imagem possam ser ordenadas corretamente (PEREIRA, 2009).

## 5.6 Considerações

Neste capítulo foram apresentados os conceitos básicos de compressão de vídeo e de decodificação de vídeo no contexto do padrão H.264/AVC. Como não é o objetivo deste trabalho aprofundar os estudos de compressão de vídeo, as funções de cada módulo foram vistas brevemente, tendo-se apenas como objetivo apresentar a dependência entre os módulos e como os mesmos se comunicam, fornecendo uma compreensão geral do funcionamento do decodificador de vídeo. A partir destas informações, pode-se verificar algumas necessidades de conexão dos módulos à NoC, possibilitando compreender também algumas soluções de projeto das interfaces de rede para SoCs.

No próximo capítulo serão apresentadas as arquiteturas de interfaces de rede desenvolvidas, considerando-se as características anteriormente comentadas e discutindo as soluções que foram implementadas.

# 6 ARQUITETURAS DAS INTERFACES DE REDES DESENVOLVIDAS

Neste capítulo, serão abordadas as propostas de interfaces de redes projetadas para a conexão de módulos de hardware em redes em chip. As soluções arquiteturais utilizadas para o desenvolvimento das interfaces de rede desse trabalho podem ser reproduzidas para qualquer aplicação e exploram diversos cenários encontrados em aplicações reais. Como estudo de caso, utilizou-se as interfaces de rede desenvolvidas para a conexão do decodificador de vídeo segundo o padrão H.264 à rede SoCIN.

Os módulos que compõem uma aplicação apresentam heterogeneidade entre si sobre diferentes aspectos, seja por possuírem largura da palavra de dados distintas, ou por apresentarem diferentes taxas de comunicação, ou pelo fato de alguns EP apresentarem tempo de processamento em função dos dados de entrada ou ainda pelo motivo de cada núcleo comunicar-se com um número diferente de EPs. Considerando-se tais situações, o ideal seria poder reutilizar a mesma descrição de interface de rede para interconectar qualquer núcleo a um roteador de uma NoC, com um mínimo de reprojeto possível. Nesse trabalho, procurou-se desenvolver alguns módulos básicos na constituição das interfaces de rede que podem ser utilizados como solução de NIs em diferentes situações. A arquitetura de interface de rede básica para conexão dos núcleos projetada nesse trabalho está baseada no esquema apresentado na figura 6.1.

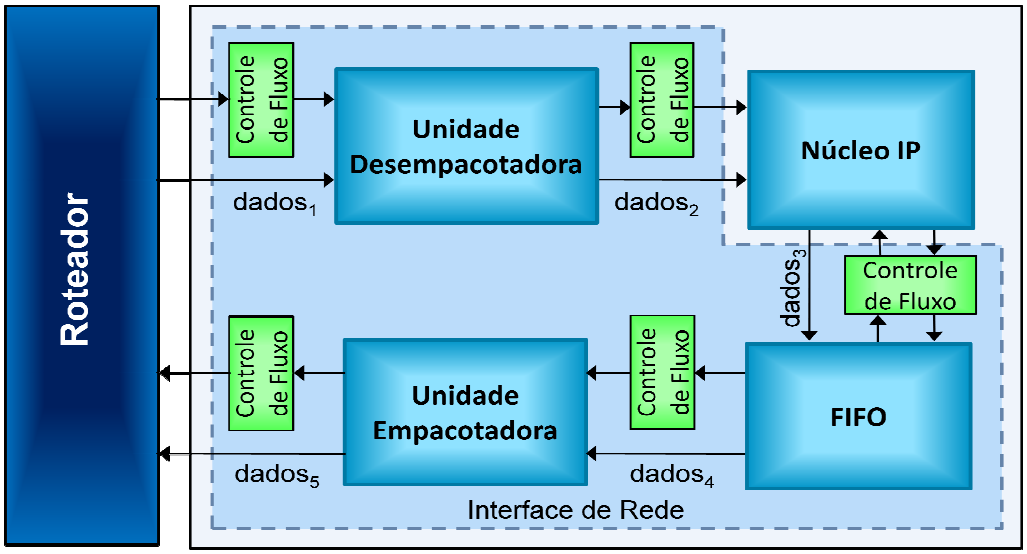



Figura 6.1: Estrutura básica da interface de rede desenvolvida.

A figura 6.1 apresenta uma Unidade Desempacotadora que tem como função receber os flits pela rede ($dados_1$) e compor os dados para serem enviados para a entrada do núcleo IP ($dados_2$). O núcleo IP, por sua vez, processará os dados recebidos e entregará dados na saída que são armazenados em uma FIFO ($dados_3$). O uso dessa FIFO se faz necessário em inúmeras situações e a sua profundidade dependerá da taxa de processamento de dados, conforme já comentado na subseção 5.1.5. Quando a Unidade Empacotadora estiver disponível, um novo dado é lido da FIFO ($dados_4$) e empacotado para a NoC ($dados_5$).

Conforme a arquitetura de interface de rede apresentada na figura 6.1, caso algum EP apresente um protocolo específico, a arquitetura proposta permite adicionar um *wrapper* específico para realizar a comunicação do núcleo com as NIs desenvolvidas através de um simples controle de *handshake* já incorporados às interfaces propostas.

As próximas seções detalharão as arquiteturas de interfaces de rede desenvolvidas e serão apresentados os resultados de síntese para algumas configurações.

## 6.1 Modelos de Interfaces de Rede

Nesta seção, serão apresentadas propostas de interfaces de rede que podem ser facilmente adaptadas na conexão de qualquer EP a uma rede-em-chip. Serão apresentadas algumas necessidades de NIs que frequentemente se enquadram as aplicações atuais e que também foram encontradas na implementação do decodificador de vídeo utilizada neste trabalho. Conforme será detalhado nas próximas subseções, é possível observar que as interfaces são genéricas já que permitem a configuração sobre diferentes aspectos, adotando as mesmas parametrizações da rede SoCIN e ainda permitindo configuração da profundidade das FIFOs da NI e largura do pacote. Além disso, pode-se observar que o protocolo utilizado pela NI para a conexão dos IPs à rede se adapta a outras implementações de interface de IPs, como será discutido a seguir. Nas próximas subseções serão apresentados como os blocos que constituem a NI se comunicam com a NoC e com o Núcleo IP, e ainda serão detalhadas as Unidades Desempacotadora e Empacotadora.

### 6.1.1 Interface de Rede Genérica

O modelo de NI apresentado nesta subseção permite a conexão de um EP à rede SoCIN. Para isso, os EPs (também chamado de núcleo IP neste capítulo) precisam apresentar uma simples estrutura de comunicação, conforme apresentado na figura 6.2.

A figura 6.2 apresenta um detalhamento da figura 6.1 com relação ao controle de fluxo dos dados da interface de rede. Explorando estes controles de fluxo, verifica-se que o recebimento de um novo flit pela interface de rede a partir de um roteador da NoC ocorrerá quando o sinal de *val_unpack* for igual a 1, indicando que existe um dado válido na saída do canal local daquele roteador. Conforme o *handshake* adotado pela NoC SoCIN, a interface de rede envia o sinal chamado de *ack_unpack* para o roteador, indicando que um novo flit pode ser recebido. Para que o sinal *ack_unpack* seja ativado, é necessário que o sinal de *val_unpack* seja igual a 1 e o sinal *data_ok* igual a 0, conforme apresentado na figura 6.2. Quando o sinal *data_ok* for igual a 0 significa que não existe nenhum dado pendente na Unidade Desempacotadora. Sempre que o sinal *data_ok* for igual a 1 é porque que um pacote anterior foi reconstituído a partir dos flits recebidos pela rede mas o Núcleo IP ainda não pôde recebê-lo. O sinal de *data_ok* passa a ser igual a 0 quando o pacote for transmitido para o Núcleo IP. Enquanto o sinal *rd* do Núcleo IP não for igual a 1, significa que o Núcleo IP não pode receber um novo

pacote, porque um dado anterior está em processamento, e dessa forma, o novo pacote é mantido em um registrador na Unidade Desempacotadora até que o mesmo possa ser enviado para o Núcleo IP, liberando esta unidade para o recebimento de novos flits. Pode ser entendido ainda que enquanto o sinal *data_ok* é igual a 0, um pacote em constituição ainda não recebeu o flit terminador ou que a máquina de estados da Unidade Desempacotadora está ociosa.

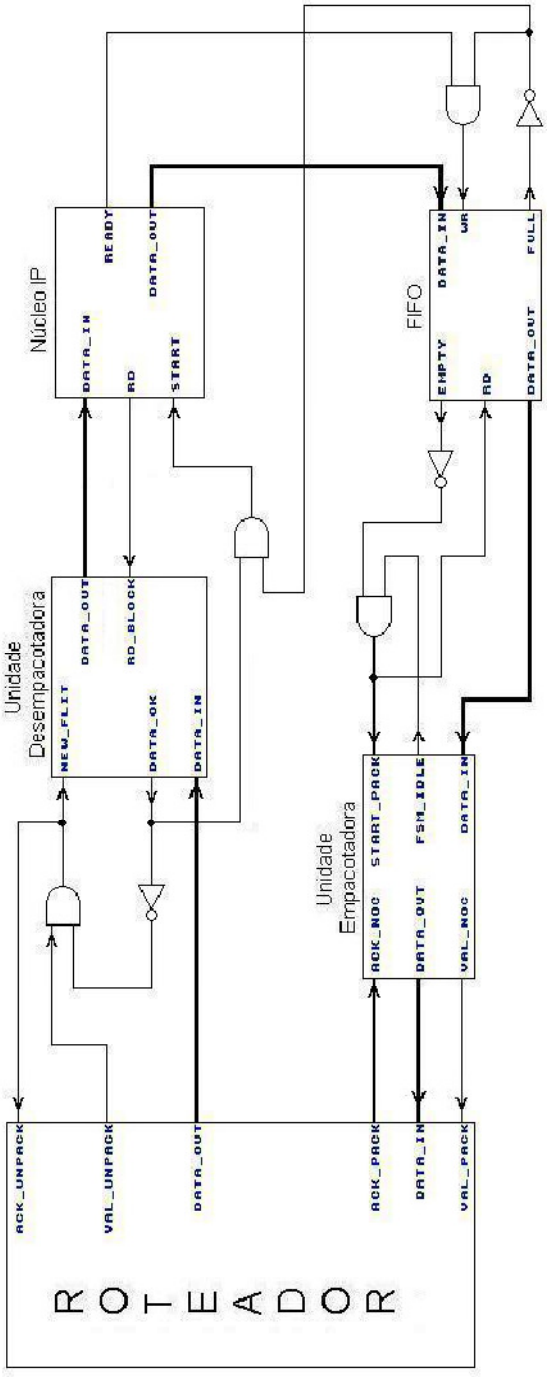



Figura 6.2: Arquitetura básica de uma interface de rede.

Normalmente um EP apresenta algum protocolo para controle do recebimento e envio de dados. Alguns dos módulos empregados neste trabalho que constituem o decodificador de vídeo apresentavam sinais de controle para interface com FIFOs, conforme apresentado na figura 6.3. Sendo assim, adaptou-se esse mesmo protocolo de conexão com as FIFOs para a utilização das NIs. No entanto, se os EPs apresentarem outro tipo de interface, o sinal de *rd* do Núcleo IP pode ser diretamente substituído por qualquer outro sinal que indique que o IP pode receber um novo dado de entrada, mantendo o correto funcionamento da interface de rede. O sinal *start* enviado para a entrada do IP também pode ter outra função, ele apenas indica que um novo pacote foi formado pela Unidade Desempacotadora. Como exemplo de uma adaptação deste sinal para uma outra necessidade pode-se citar o caso dos módulos do decodificador utilizados neste trabalho. Como os módulos realizam uma interface através de FIFOs, o IP lê os dados de uma FIFO, os processa e os envia para uma segunda FIFO que recebe os dados de saída do IP. Sendo assim, para que o IP possa ler os dados da primeira FIFO e ativar o sinal de *rd*, ele precisa receber a informação de que esta FIFO não está vazia e assim poder receber um dado válido. A adaptação que foi feita nesse caso para o sinal de *start* foi utilizar a saída invertida do *data_ok* para fazer às vezes de FIFO vazia. Sendo assim, quando a Unidade Desempacotadora ainda não tiver todo o pacote reconstituído pelo fato de que ainda não recebeu todos os flits que o compõem, o sinal *data_ok* será igual a 0 e dessa forma, a inversão desse valor reflete a situação de FIFO vazia (o sinal de *empty* que seria recebido de uma FIFO).


Figura 6.3: Protocolo de conexão de IPs através do uso de FIFOs.

O sinal de *ready* enviado pelo IP indica que um novo dado de saída está disponível e é este sinal que habilita a escrita na FIFO de saída sempre que a mesma não estiver cheia (sinal *full*). Da mesma forma, o sinal de *ready* foi adaptado para o protocolo com FIFOs na implementação da aplicação do decodificador de vídeo, já que nesse caso, o IP já apresentava um sinal de escrita na FIFO.

A Unidade Empacotadora, por sua vez, é acionada sempre que a FIFO não estiver vazia (sinal *empty*) e quando não houver flits sendo enviados para a rede (Unidade Empacotadora ociosa). Estas duas condições são necessárias para ativar a Unidade Empacotadora. A indicação de que esta unidade está ociosa é sinalizada pelo sinal *fsm_idle*. Como a Unidade Empacotadora recebe todo o pacote da FIFO e o divide em flits, é possível que o tempo de envio de todos os flits que compõem o pacote seja maior que o tempo de processamento do Núcleo IP, sendo que o tempo de envio do pacote pela rede dependerá do tamanho do mesmo, da largura do canal definida para a NoC e da disponibilidade da rede de receber novos flits. O ideal é que o tempo de envio de todos os flits do pacote seja aproximadamente o mesmo tempo do processamento do IP e um dos fatores que pode ser ajustado para que isso ocorra e definir a largura do canal da rede mais apropriada.

Existem vários aspectos que tornam necessário o uso de uma FIFO com largura de dados igual à palavra de dados recebida ou transmitida pelo IP. Esta FIFO poderia estar disposta ou na entrada do IP ou na saída do IP, o importante é que uma vez definida esta disposição, todas as NIs apresentem o mesmo padrão. Dessa forma, quando um caminho da rede estiver congestionado por muito tempo é garantido que todos os dados do fluxo estarão armazenados apropriadamente nas FIFOs das NIs e nas FIFOs da NoC.

Neste trabalho definiu-se o uso de uma FIFO sempre na saída do IP, antes da interface de rede realizar o empacotamento dos dados. Sendo assim, se em um determinado momento, um núcleo receptor não puder receber novos dados, somente após todos os buffers presentes na rota percorrida pelo pacote estarem cheios é que os flits deixam de ser enviados para o nodo destinatário. Para que o núcleo transmissor deixe de produzir dados, todas as FIFOs utilizadas no trajeto de envio do pacote precisarão estar preenchidas, desde a FIFO utilizada na NI do módulo transmissor até as FIFOs presentes nos canais de entrada da NoC. Sendo assim, quando o núcleo receptor novamente estiver disponível para receber novos dados, os flits são rapidamente repassados, sem prejudicar o desempenho da aplicação. Nessa situação, a Unidade Desempacotadora terá um pacote totalmente reconstituído, aguardando o sinal de leitura enviado pelo IP. Esta estratégia de armazenamento pode reduzir significativamente a latência na entrega da mensagem, já que os flits são armazenados ao longo de todo o trajeto percorrido pelo pacote (DE MICHELI, 2006). A utilização de uma FIFO com a largura de dados do IP é ainda justificada nas seguintes situações:

- Quando o IP puder gerar novos dados, mesmo que nem todos os flits pertencentes ao pacote anterior tenham sido enviados para a rede. Sendo assim, é necessário um dispositivo de armazenamento para não prejudicar o desempenho da aplicação. Esta situação pode ocorrer quando o tempo de processamento do IP varia ou conforme a indisponibilidade da rede em receber os flits referentes ao pacote anterior. Mesmo que na média, o envio dos flits seja igual à taxa de processamento do IP, se a taxa variar, torna-se necessário o uso de um dispositivo de armazenamento.
- Se uma FIFO de flits fosse utilizada ao invés de uma FIFO com a largura da palavra de dados do IP, da mesma forma, o desempenho da aplicação seria prejudicado, já que o IP só poderia gerar um novo dado após todo o pacote ter sido enviado para a rede. Como os canais da NoC podem ser utilizados por pacotes enviados por diferentes nodos, a disponibilidade destes canais não é sempre garantida. Da mesma forma que IPs apresentam taxas irregulares para a geração de dados na saída, a disponibilidade da rede também não ocorre em tempos constantes. Sendo assim, utilizando-se uma FIFO de flits, o IP só poderia iniciar o processamento de novos dados quando a Unidade Empacotadora estivesse ociosa e esse intervalo de tempo pode trazer grandes perdas de desempenho. Com o uso desta FIFO, a NI irá enviar os pacotes para a rede e simultaneamente novos dados estarão sendo processados pelo IP. Porém, se o caminho para envio dos flits para o IP subsequente não estiver disponível, o desempenho da rede não é muito prejudicado, pois os dados já estarão armazenados nas FIFOs e prontos para o envio, assim que a rede permitir transmiti-los.

As FIFOs presentes nas NIs podem ser utilizadas para o armazenamento de dados distintos gerados pelo IP de origem e/ou que serão enviados para diferentes IPs destinatários. Quando não houver nenhuma restrição na geração destes dados pelos IPs,

uma mesma FIFO pode ser utilizada para armazenamento de dados. No entanto, uma identificação de cada palavra de dados armazenada e para quem a mesma deve ser enviada deve ser concatenada aos dados salvos na FIFO para posterior identificação pela Unidade Empacotadora.

Como as FIFOs estarão dispostas sempre após o IP em todas as interfaces, o correto armazenamento dos dados é garantido quando houver indisponibilidade no recebimento dos dados pelos canais da NoC ou por algum IP. Como esta FIFO apresenta largura de dados igual à gerada pelo IP, não foi necessário utilizar uma FIFO também na entrada do IP, pois dessa forma se garante a taxa de envio dos pacotes pela rede, bem como o correto armazenamento dos dados ao longo de todo o caminho. Com relação ao desempenho, o uso de uma FIFO na entrada do IP provavelmente não apresentaria muita vantagem, além de representar um consumo maior de potência. No entanto, pode ser requerido o uso de mais FIFOs na NI para atender a outras necessidades da aplicação, como será discutido nas próximas subseções. Além disso, pode-se utilizar uma FIFO também na entrada do IP, como solução para a adaptação de diferentes relógios utilizados pelo EP e a NoC. Inúmeros trabalhos apresentam a utilização de FIFOs para essa função. Da mesma forma, a FIFO utilizada pode também ser utilizada como solução nestes casos, já que apresenta ponteiros de leitura e escrita com domínios de relógios distintos, o que permitiria fazer a sincronização de relógios entre os dispositivos. No entanto, deixou-se esta análise para um trabalho futuro.

Tanto a Unidade Empacotadora como a Unidade Desempacotadora foram projetadas para se adaptarem ao controle de fluxo da NoC. A figura 6.4 apresenta o roteador RASoC que compõe a rede SoCIN e os sinais utilizados como controle de fluxo pelo canal local. Como se pode observar pela figura 6.4, o canal local aciona o sinal *val_unpack* quando tiver um flit a ser entregue para a NI. Se a Unidade Desempacotadora puder receber o flit, o sinal de *ack_unpack* é acionado. Para que o envio de um flit da NI para o roteador possa ser realizado, da mesma forma que no recebimento dos flits, a NI precisa primeiramente enviar um sinal de *val_pack* e aguardar o recebimento do *ack_pack* igual a 1. Unidade Empacotadora aciona o sinal de *val_pack* sempre que um novo flit estiver disponível para o envio. Quando houver espaço disponível na FIFO do canal local, então o envio do flit será permitido.

Nesta subseção, procurou-se apresentar uma interface de rede genérica que possui conexões com um roteador da NoC e com um IP e foi ilustrado como ocorre a conexão deste modelo de NI quando os IPs de uma aplicação já apresentam um protocolo de comunicação entre eles, como é o caso do decodificador de vídeo utilizado como estudo de caso neste trabalho. Isto prova que a NI pode ser adaptada com um mínimo de reprojeto para outras implementações de interfaces entre os IPs. As NIs projetadas consideram sinais de controle de entrada e saída dos dados normalmente implementados em módulos de HW e são de fáceis adaptações quando os IPs apresentam algum outro controle. Nas próximas subseções serão apresentadas as necessidades de uma NI para algumas situações específicas.

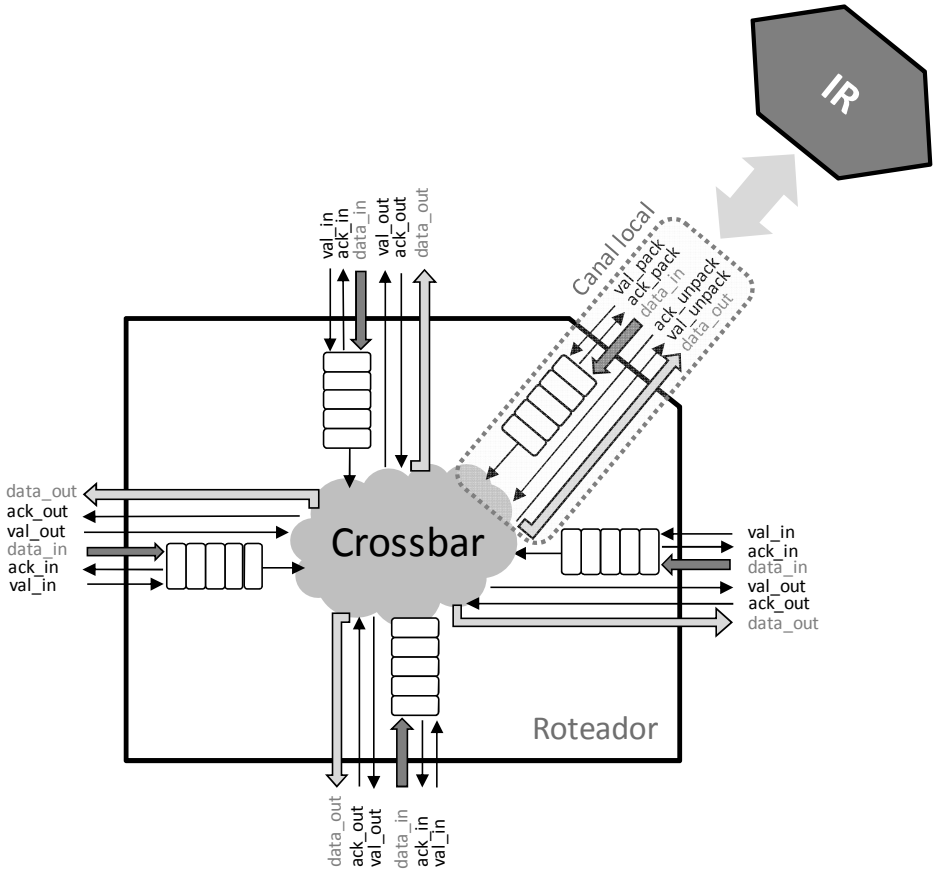



Figura 6.4: Roteador RASoC com destaque para o controle de fluxo do canal local.

### 6.1.2 Interface de rede com recebimento de pacotes de múltiplos núcleos

Conforme já comentado anteriormente, em aplicações complexas, muitos núcleos precisam se comunicar com um mesmo IP, da mesma forma que um único IP pode necessitar enviar dados para diferentes núcleos. Existem distintas maneiras dessa comunicação ocorrer e dependerá das exigências de processamento de cada IP. Por exemplo, se um IP realiza o mesmo tipo de processamento para dados de diferentes IPs, então o esquema apresentado na figura 6.2 pode ser implementado, utilizando somente uma unidade de FIFO na saída do Núcleo IP e sem a necessidade de utilização de FIFOs na entrada do IP. No entanto, quando o IP necessita de dados de diferentes IPs de origem para iniciar o processamento, um esquema como apresentado na figura 6.5 é requerido. A abordagem sugerida por (FERRANTE, 2008) alega a possibilidade de utilizar somente uma unidade de FIFO para a comunicação de diferentes IPs, no entanto a possibilidade de necessitar de dados de diferentes IPs simultaneamente por um outro EP não foi levantada, sendo assim, a vantagem de divisão de recursos de HW levantadas pelo autor só é válida para a situação anteriormente comentada, o que também se aplica a implementação de NI proposta nesse trabalho.

A utilização de uma única FIFO pela NI não é possível quando um IP, para realizar o processamento, necessita de dados de diferentes núcleos simultaneamente. Essa impossibilidade ocorre porque não se garante que o recebimento de cada pacote pela rede se dará em ordem intercalada, sendo que um determinado IP pode ter enviado, por exemplo, 4 pacotes para a NI, enquanto que outro módulo pode ter enviado apenas 1

pacote. Sendo assim, nesse caso, uma das soluções para o sincronismo destes dados é utilizar uma FIFO para cada IP, como apresentado no esquema da figura 6.5. A diferença da proposta apresentada na figura 6.2 para o esquema ilustrado na figura 6.5 ocorre após a etapa de desempacotamento dos dados. Após a reconstituição da palavra de dados através dos flits recebidos pela Unidade Desempacotadora, cada palavra de dados tem uma FIFO de destino, conforme o IP que enviou o pacote. Para isso, a Unidade Desempacotadora precisa receber o endereço do IP de origem de cada pacote e a partir dessa identificação, é verificada para qual FIFO a palavra de dados será enviada. Como apresentado na figura 6.5, a escrita na FIFO depende de 3 situações que são:

- a FIFO não pode estar cheia (indicado pelo sinal de *full* na FIFO);
- o sinal *data_ok* da NI, indicando que tem um pacote pronto, deve estar ativo;
- o endereço de origem do IP deve coincidir com a FIFO definida para armazenamento do mesmo.

Atendendo a estas situações, cada FIFO terá os dados armazenados conforme o IP que o enviou. Quando o IP depende dos dados de todas as FIFOs para desenvolver a computação, é necessário verificar se elas contêm dados válidos e essa indicação é feita pelo sinal de *empty* de cada FIFO. Quando nenhuma FIFO estiver vazia (sinal de *empty* de todas as FIFOs igual a 0) e o Núcleo IP puder receber os dados (*rd* do Núcleo IP igual a 1), os dados na saída da FIFO serão lidos e o processamento do IP iniciado. Esta dependência de dados foi uma das necessidades levantadas na implementação do decodificador de vídeo que será melhor detalhada na subseção 7.1.4.

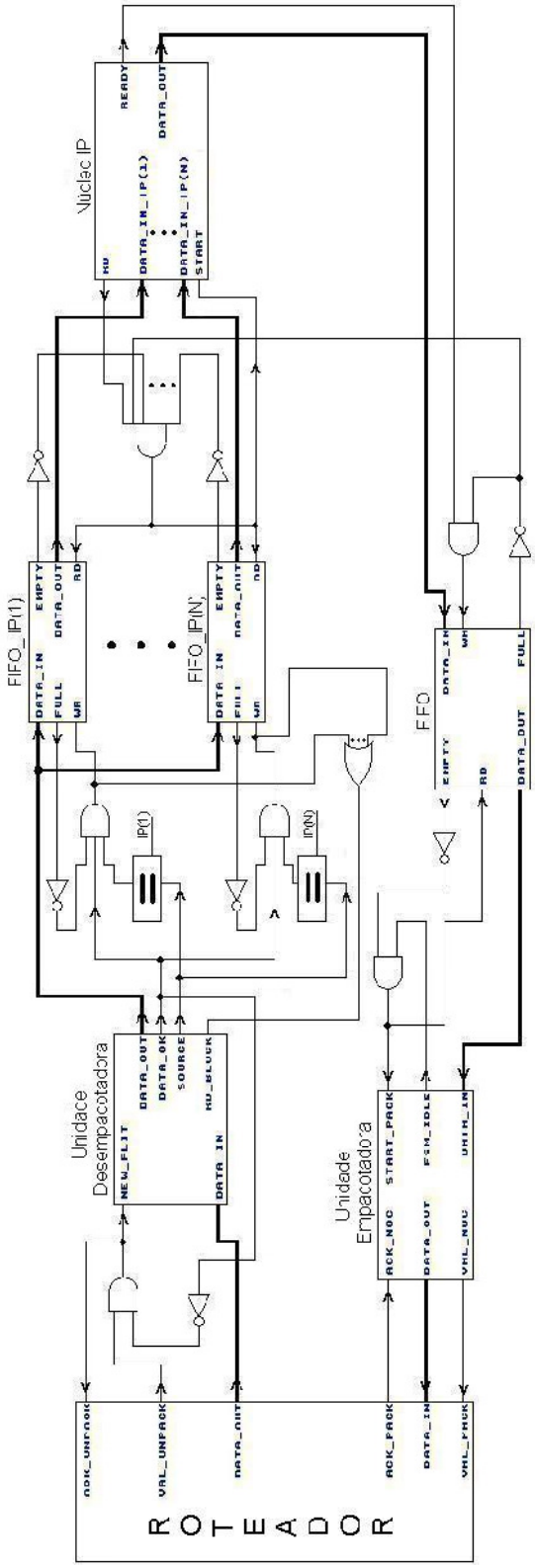



Figura 6.5: Arquitetura de uma interface de rede quando um IP recebe dados de múltiplas fontes.

### 6.1.3 Interface de rede com transmissão de pacotes para múltiplos núcleos

Outra necessidade levantada no estudo de caso do decodificador de vídeo é o envio de dados distintos para múltiplos IPs. Neste caso, como comentado na seção anterior, não é possível utilizar uma mesma FIFO para envio dos dados, primeiramente por que cada IP pode apresentar uma largura diferente da palavra de dados, o que poderia acarretar em desperdício de recursos caso definisse-se a largura de dados da FIFO conforme o dado que apresenta maior largura. Algumas condições precisam ser avaliadas para a utilização de uma única FIFO na saída do núcleo IP. Por exemplo, se o envio dos dados pela rede puder seguir a mesma sequência com que eles são gerados pelo núcleo, pode ser utilizada uma única FIFO, mesmo que o IP gere palavras com largura da palavra de dados diferentes. No entanto, nessa situação, é necessário que a largura de dados da FIFO seja igual a maior largura da palavra de dados entregue pelo IP. Nesse caso, a Unidade Empacotadora precisa selecionar a largura de dados da palavra em função do endereço do IP destinatário. Para isso, é necessário armazenar na FIFO, juntamente com o dado que será repassado para a rede, o endereço do IP destinatário. Dessa forma, quando a Unidade Empacotadora for enviar o pacote para a rede o cabeçalho apropriado é selecionado em função do IP destinatário.

Porém, quando os dados não puderem ser enviados conforme a sequência com que são gerados ou quando eles puderem ser gerados simultaneamente, é preciso utilizar dispositivos de armazenamentos distintos, além de alguma política de arbitragem para o envio dos pacotes pela NI. O processo de escrita nas FIFOs ocorre da mesma forma apresentada na primeira proposta de NI, ou seja, sempre que um dado estiver pronto na saída do IP e a sua FIFO respectiva não estiver cheia, este dado pode ser escrito na mesma. A diferença aqui é que o Núcleo IP pode gerar os dados a serem salvos nas FIFOs em tempos distintos, então pode ser necessário ter um controle independente para cada um dos dados gerados, informando quando o mesmo está pronto na saída. Se os dados sempre forem gerados ao mesmo tempo, então somente um sinal de *ready* é necessário e a escrita nas FIFOs ocorre simultaneamente.

Com relação à decisão de arbitragem para envio dos dados das FIFOs para a Unidade Empacotadora, pode ser necessário definir uma lógica de prioridade para o envio dos pacotes pela rede. Uma solução semelhante foi necessária na implementação do decodificador de vídeo, no entanto, esta análise será melhor detalhada no capítulo 7, onde serão apresentados exemplos reais destas necessidades. Aqui apenas será ilustrado um exemplo genérico da necessidade desta lógica. A figura 6.6 apresenta uma NI onde o IP gera dados para múltiplos núcleos e estes são armazenados em FIFOs independentes. Entretanto, os dados a serem enviados para um mesmo núcleo podem ser gerados a um tempo de processamento variado e, além disso, a diferença do tempo de processamento de cada IP pode ser muito grande. Sendo assim, se nenhuma lógica de arbitragem for definida, é possível que os dados para um determinado IP levem muito tempo para serem enviados, prejudicando o desempenho da aplicação. Esta situação pode ocorrer se, por exemplo, o IP gera dados para o núcleo X a uma taxa muito maior do que para o núcleo Y. Se o envio dos dados se der primeiramente pela verificação da FIFO com dados para o núcleo X, como raramente esta FIFO estará vazia, o envio de dados para o núcleo Y pode ser muito postergado ou até mesmo não ocorrer, gerando uma situação conhecida como *starvation*. Na implementação do decodificador de vídeo priorizou-se o envio dos dados para o núcleo que apresenta menor taxa de processamento e esta decisão será melhor discutida na subseção 7.1.2.

Como se pode observar na figura 6.6, os dados são lidos da FIFO e repassados para a Unidade Empacotadora conforme a prioridade definida para seleção do pacote a ser enviado (sinal *priority*). De acordo com o núcleo para o qual será enviado o pacote, a Unidade Empacotadora define o cabeçalho do pacote a partir de uma tabela com cabeçalhos pré-definidos para cada caso. Dessa forma, conforme a definição de prioridade, é verificado para qual núcleo o pacote deve ser enviado e em função deste código de destino, é feita a seleção do cabeçalho na tabela. São utilizados multiplexadores para a seleção dos dados, cabeçalhos e ativação do sinal de *rd* da FIFO que será lida. Os demais controles apresentados na figura 6.6 são os mesmos apresentados para a interface de rede genérica, conforme ilustrado na figura 6.2.

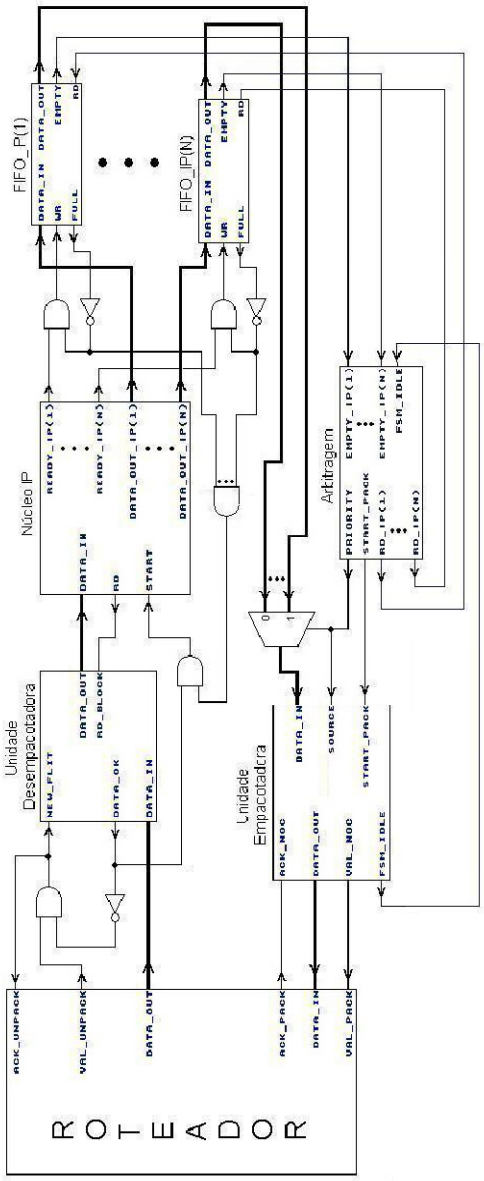



Figura 6.6: Arquitetura de uma interface de rede quando um IP envia dados para múltiplos destinatários.

### 6.1.4 Unidade Empacotadora

A Unidade Empacotadora é constituída de uma máquina de estados, de multiplexadores, registradores e um contador. A máquina de estados (FSM) implementada para esta unidade está apresentada na figura 6.7. Cada estado válido tem a função de despachar um determinado flit, podendo ser flit de cabeçalho, de dados ou um flit terminador. A condição para a passagem do estado *header* para o *tail* ou *payload* ou do *payload* para o *tail* ocorrerá se o flit referente ao estado atual puder ser enviado para a rede. Para passar para o estado de *tail* ainda é requerida a condição de o próximo flit ser o último do pacote que está sendo enviado. O último flit é controlado em função da largura da palavra de dados e da largura do canal da rede. Foi implementado um controle que verifica se o número de bits de dados enviados se aproxima da largura da palavra de dados do IP e, a partir dessa informação e da definição da largura do canal da NoC, é sabido quando o flit corresponde ao último do pacote. Conforme visto anteriormente, a NoC SoCIN possui um controle de fluxo a partir de um protocolo de *handshake*. Sendo assim, sempre que um flit precisar ser enviado para o canal local do roteador, a interface de rede envia um sinal de flit válido (*val_pack*) e aguarda o recebimento de um sinal de *acknowledgement* (*ack_pack*), conforme ilustrado na figura 6.4. Se o sinal de *ack_pack* for igual a '1' significa que existe espaço disponível no buffer de entrada do canal local e que o flit pode ser enviado. Caso não haja espaço nos buffers do canal local, o flit a ser enviado aguarda no estado que está e este valor é registrado até que o seu envio seja permitido.

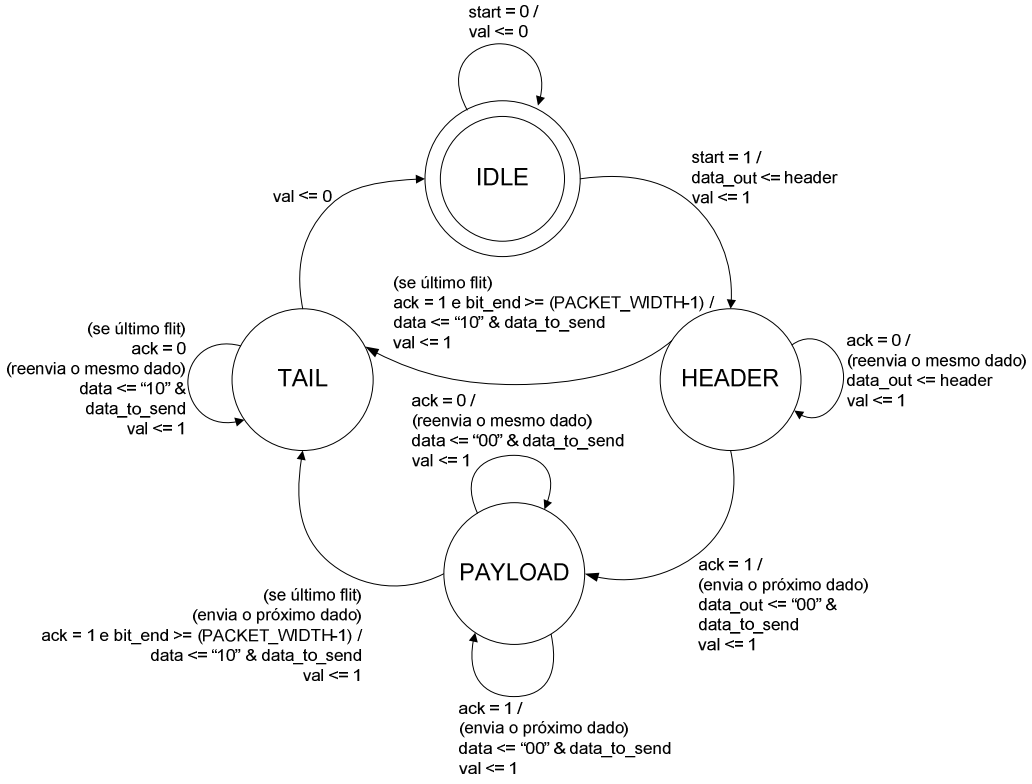



Figura 6.7: Máquina de Estados da Unidade Empacotadora.

Utilizou-se um multiplexador para definir o tipo de flit a ser enviado conforme o estado na FSM da figura 6.7, definindo os bits de enquadramento do pacote (*bop* e *eop*)

e concatenando-os aos dados que compõem o flit. É utilizado ainda um registrador que contém a mensagem completa a ser transmita e um outro registrador que armazena o flit quando este não puder ser enviado para a rede.

Quando a NI necessita enviar dados para mais de um módulo é possível utilizar-se uma única Unidade Empacotadora para o envio de pacotes para diferentes destinatários. No entanto, para estas situações, necessita-se adicionar um multiplexador para definir para quem deve ser enviando o pacote, conforme já foi comentado anteriormente. A mesma unidade de empacotamento é utilizada para envio dos pacotes para diferentes nodos. Porém, no nodo destinatário, o Núcleo IP pode receber dados de mais de um módulo, como é o caso do módulo de predição INTRA que recebe dados de dois módulos (ITIQ e *parser*). Sendo assim, é necessário identificar qual foi o nodo que enviou o pacote e, nestes casos, foi preciso adicionar mais um estado à FSM para a identificação do nodo que transmitiu o pacote.

Quando um pacote chega ao nodo destinatário, nenhuma informação de qual módulo o enviou é recebida no modelo de pacotes utilizado pela SoCIN. Sendo assim, definiu-se que um dos flits será utilizado para informações extras, como endereço do nodo que enviou o dado, além de outras informações que precisam ser enviadas em uma outra proposta de NI comentada na seção 6.2. Embora eventualmente fosse possível utilizar os bits que não foram utilizados no cabeçalho para a definição do roteamento, preferiu-se aderir a uma solução mais genérica. Dessa forma, não se precisa levar em consideração a largura do canal definida versus o tamanho da rede, já que estas são informações que impactam no número de bits necessários para a informação de roteamento no cabeçalho. Sendo assim, dependendo da aplicação e da largura do canal, não se garante que sempre existirão bits disponíveis no flit de cabeçalho para a indicação do módulo transmissor. Além disso, a solução de utilizar o segundo flit para a indicação do núcleo transmissor não necessitou fazer qualquer alteração na implementação da rede e esse flit também pode ser utilizado para envio de rótulos, conforme proposto nesse trabalho como uma das soluções de sincronização que será detalhado na seção 6.2.

A máquina de estados com a adição do estado de *SOURCE* que indica o código do módulo transmissor está mostrada na figura 6.8. Na implementação do decodificador de vídeo utilizou-se tanto a máquina de estados apresentada na figura 6.7 como a da figura 6.8, embora fosse possível utilizar somente a máquina de estados da figura 6.8. Essa decisão foi tomada em função de que alguns módulos enviam somente pacotes para um destinatário e este receptor recebe pacotes somente de um mesmo transmissor. Nesse caso, como não havia a necessidade do envio do flit extra e como cada flit enviado pela rede apresenta uma determinada latência, decidiu-se utilizar a máquina de estados da figura 6.8 somente quando fosse necessário.

A figura 6.9 ilustra como o pacote definido para as NIs implementadas é formado. A definição do formato do pacote é a mesma utilizada pela rede SoCIN, com a diferença de que se adicionou um flit reservado utilizado para identificação do transmissor do pacote e outras informações utilizadas na solução de sincronização necessárias para o funcionamento da rede e que diferem da carga útil. O flit reservado será sempre o flit que vem logo a seguir ao cabeçalho. Somente após o recebimento deste flit é que a rede começa a enviar os dados referentes à carga útil. O cabeçalho (*header*) indica início do pacote e o flit terminador (*tail*) indica fim do pacote.

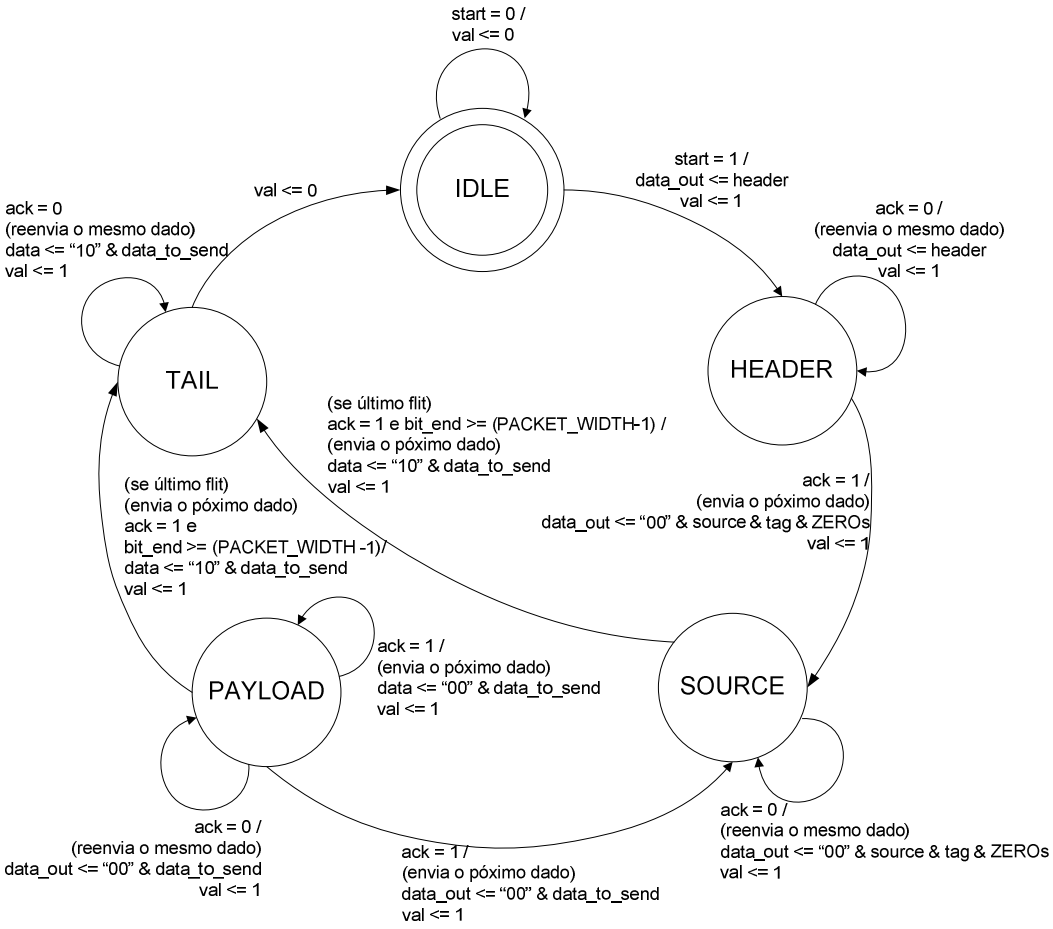



Figura 6.8: Máquina de Estados da Unidade Empacotadora com a definição do código do IP transmissor.

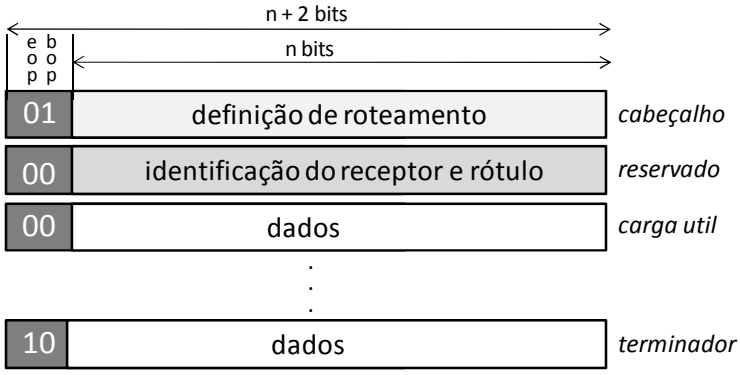



Figura 6.9: Formato do pacote definido a partir da rede SoCIN adaptado para prover informações adicionais.

O formato do pacote definido na figura 6.9 foi utilizado sempre que a identificação do receptor se fez necessária, quando não foi preciso passar essa informação, o pacote apresentou apenas os flits de cabeçalho, carga útil e terminador, conforme definido pela rede SoCIN (ZEFERINO, 2003).

Utilizou-se um contador com a função de controlar o envio dos flits de dados. O contador é incrementado sempre que um novo flit de dados é enviado para a rede e este incremento só ocorre após a confirmação do envio do flit, que se dará conforme a disponibilidade da rede. A arquitetura do contador está ilustrada na figura 6.10. Como se pode observar, a decisão de contagem é feita em função do estado e da disponibilidade da NoC. Conforme apresentado na figura 6.10, os sinais que controlam o contador são: o sinal de *ack_pack*, que é recebido da rede, os *flags* (*flag_payload*, *flag_header*), que são acionados conforme o estado da FSM, e o sinal *fsm_idle*, que indica que a Unidade Empacotadora não está enviando nenhum pacote para a rede. Sempre que a FSM estiver ociosa (*fsm_idle* igual a '1'), o contador é zerado. Quando se faz necessário o envio do flit contendo a identificação do código do IP transmissor, o contador só é incrementado após o envio do flit reservado (nesse caso, ao invés do *flag_header* é utilizado o *flag_source* na arquitetura apresentada na figura 6.10).

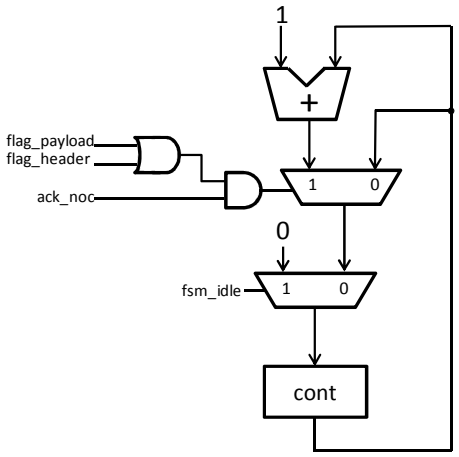



Figura 6.10: Contador utilizado na arquitetura da Unidade Empacotadora.

Na Unidade Empacotadora, um sinal interno concatena todos os dados que devem ser transmitidos pela rede, formando um pacote de dados que foi chamado de *data_to_send*, conforme apresentado pela figura 6.11. A partir deste pacote de dados, duas possibilidades de indexação dos pacotes na Unidade Empacotadora foram implementadas. A indexação dos pacotes diz respeito à forma como os flits são divididos a partir do pacote formado por *data_to_send*. Sendo assim, para manter a generalidade da aplicação e permitir que uma mesma interface de rede possa ser utilizada para diferentes tamanhos de pacotes e que atenda a diferentes configurações da largura do canal da NoC, algumas soluções de indexação para a formação dos flits a partir do conjunto de dados agrupados no pacote foram estudadas. Neste caso, a indexação refere-se a forma como os flits são particionados utilizando-se a linguagem de descrição VHDL a partir dos dados em *data_to_send.*

Na primeira solução, quando a NI for enviar o último flit do pacote e o número de bits restantes referentes ao dado válido for menor do que a largura do canal da rede, é utilizada a técnica de preenchimento de zeros (*zeros-extended*) para montar o último flit. Na Unidade Desempacotadora, os zeros extras são removidos antes de o dado ser repassado para o IP. O dado formado por *data_to_send* é empacotado em flits de acordo com os valores calculados para *bit_end* e *bit_start.* Como apresentado na figura 6.11, os valores de *bit_end* e *bit_start* são concatenados com um conjunto de bits para formar o índice correto em função da largura do canal definida para rede. Os sinais

*bit_end* e *bit_start* indexam a partição do dado a ser enviado no flit e também são utilizados para a definição do tipo de flit. Por exemplo, se a largura do canal da rede for definida para 16 bits, nesse caso o valor do contador é concatenado com "1111" para formar o *bit_end* e "0000" para formar o *bit_start*. O número de "zeros" e "uns" necessários na concatenação para a obtenção dos valores de *bit_end* e *bit_start* será sempre igual a *n*, sendo $2^n$ igual à largura do canal.

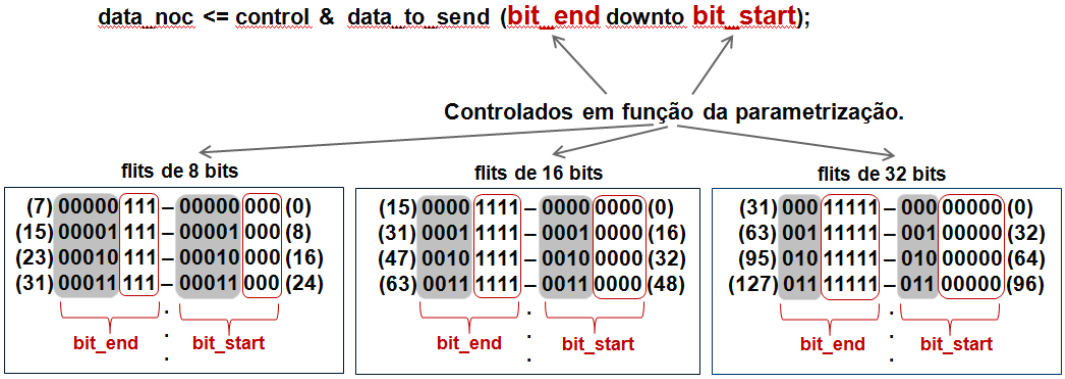



Figura 6.11: Formação dos índices para a indexação dos flits.

No entanto, essa solução só é sintetizável em FPGA. Na síntese em *standard cells* para as ferramentas Design Compiler e Leonardo Spectrum não são aceitos variáveis como índices de sinais. Por esse motivo, implementou-se uma segunda versão para o empacotamento dos dados que será apresentada na sequência e que é sintetizável tanto para FPGA, quanto para *standard cells*. Mesmo utilizando-se outra alternativa de indexação para a palavra de dados, o esquema apresentado na figura 6.11 para a obtenção do *bit_end* precisou ser utilizado, pois, em função deste valor, define-se quando a máquina de estados deve passar para o estado de *TAIL*. Sendo assim, sempre que o valor de *bit_end* for maior ou igual ao tamanho da palavra de dados do IP (*PACKET_WIDTH,* conforme identificado na máquina de estados da figura 6.7 ou figura 6.8), a FSM passa para o estado de *TAIL* e o último flit do pacote é enviado. Dessa forma, permite-se parametrizar a largura da palavra de dados do pacote apenas alterando-se seu valor a partir de uma tabela com a definição da largura da palavra de dados necessária para cada tipo de mensagem enviada pelo EP nas diferentes NIs utilizadas na implementação.

Como na síntese em *standard cells* para as ferramentas anteriormente citadas só são aceitas constantes para indexação de um sinal, a definição para essa segunda proposta ocorreu baseada nas constantes que já tinham sido definidas na arquitetura. Como as NIs projetadas são genéricas, uma das definições que se necessita indicar por tabela é a largura da palavra de dados do pacote produzido por cada IP. Baseado nessa informação, a NI sabe quantos bits de dados deve enviar e é a partir dessa constante que o flit terminador é identificado na Unidade Empacotadora. Sendo assim, quando pacotes com largura de palavra de dados variadas precisam ser enviados para a NoC pela NI, é possível utilizar a mesma Unidade Empacotadora. Neste caso, um MUX fará a seleção da constante que determina o tamanho do pacote conforme o endereço do destinatário.

A segunda proposta de indexação do pacote baseou-se nas duas constantes que já tinham sido definidas na arquitetura: a largura do canal da rede (*DATA_WIDTH*) e o tamanho do pacote (*PACKET_WIDTH*). Nesta proposta, a ideia é deslocar os dados com a mesma largura de dados definida para rede. Sendo assim, a formação do flit a ser

enviado pela rede estará sempre disposta no início do *data_to_send*. A figura 6.12 apresenta de forma mais clara como ocorre a formação dos flits para serem enviados na rede. A cada flit que é enviado para a rede, o dado é deslocado, posicionando o próximo flit para envio.

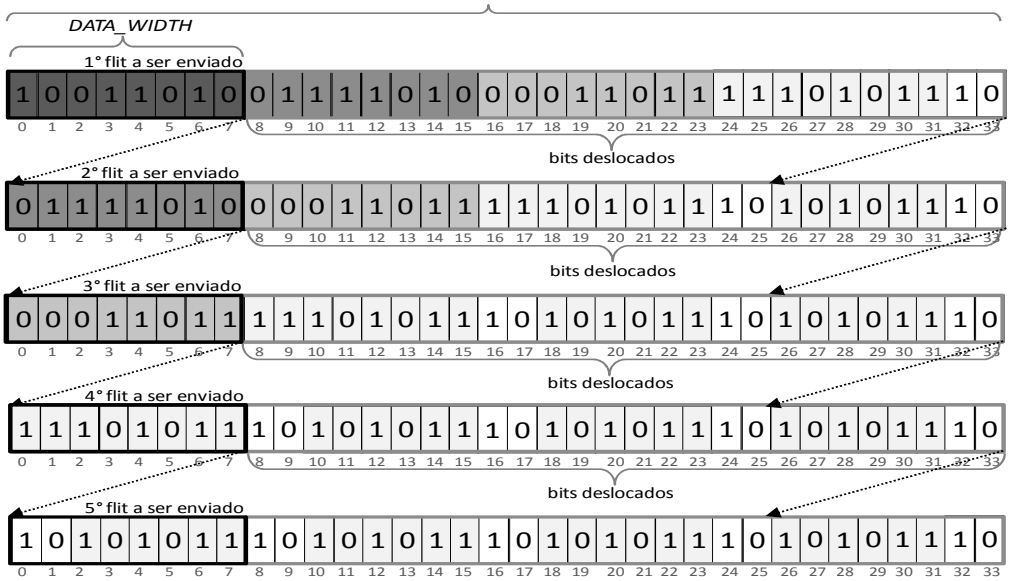



Figura 6.12: Exemplo de deslocamento dos dados para a formação dos flits a serem enviados na NoC.

No exemplo da figura 6.12, a largura do pacote é de 34 bits e a do canal de dados da rede é de 8 bits. Sendo assim, cada flit é composto de 8 flits, mas como a largura do pacote não é um múltiplo deste valor, neste caso específico, restam dois bits válidos para serem enviados no último flit. Juntamente com os dois bits válidos, o último flit é formado com bits que restaram no processo de deslocamento.

Com a solução apresentada na figura 6.12, os bits de dados são enviados para a NoC conforme a linha de código indicada em (1):

$$data\_noc <= control\_code \,\&\, data\_to\_send( DATA\_WIDTH - 1 \text{ downto } 0 ); \quad (1)$$

Dessa forma, a indexação em (1) depende apenas da definição de *DATA_WIDTH*. O sinal de *control_code* refere-se aos sinais de *bop* e *eop* para a definição do tipo de flit enviado para a rede. Com os deslocamentos, garante-se que o próximo flit estará sempre disponível no início da mensagem de dados. Para o deslocamento dos dados, utilizou-se a linha de código como em (2):

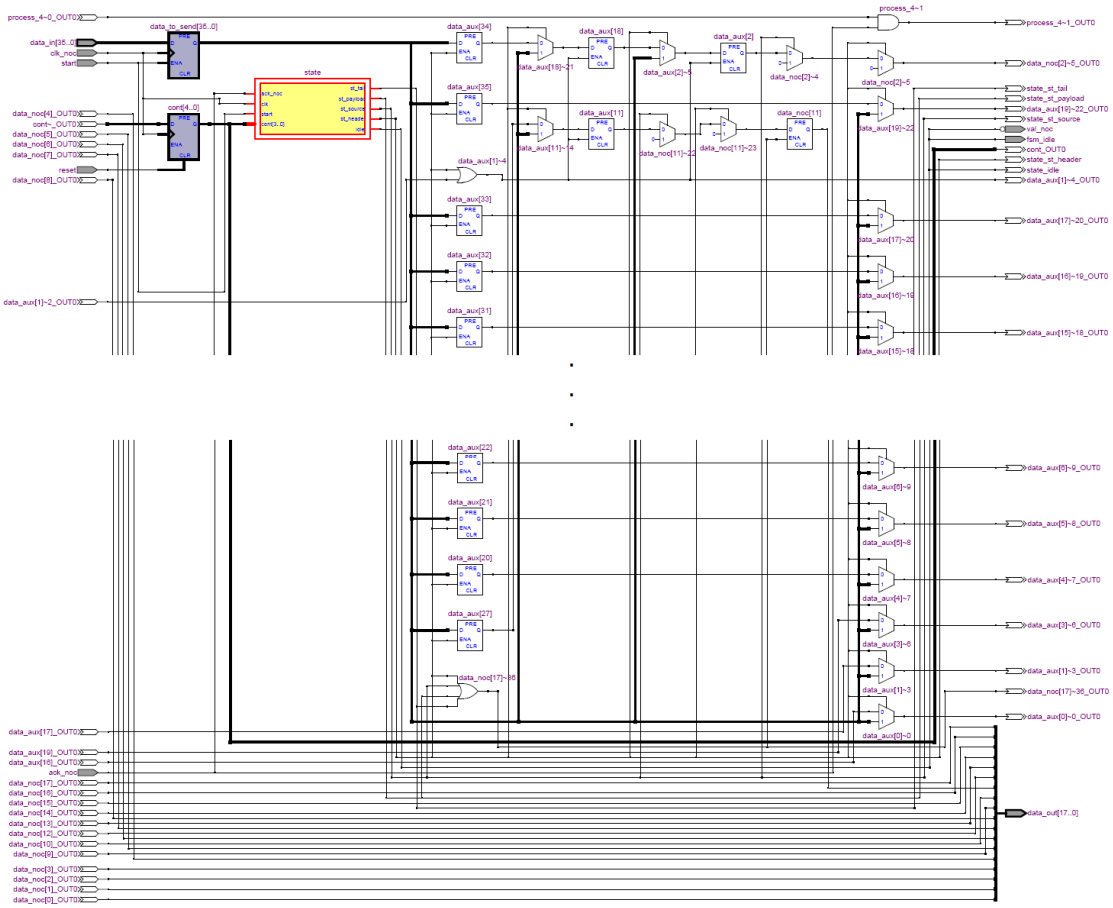
$$data\_to\_send( PACKET\_WIDTH - DATA\_WIDTH \text{ downto } 0 ) <= data\_to\_send( PACKET\_WIDTH \text{ downto } DATA\_WIDTH ); \quad (2)$$

Para a linha de código em (2), apenas as constantes anteriormente comentadas foram utilizadas (*DATA_WIDTH* e *PACKET_WIDTH*) e com esse esquema, garante-se facilmente a parametrização da NoC e das NIs com relação a largura do canal e a largura de dados do pacote. A única diferença das duas propostas acima comentadas para a formação dos flits foi com relação à indexação, no entanto, não houve alteração

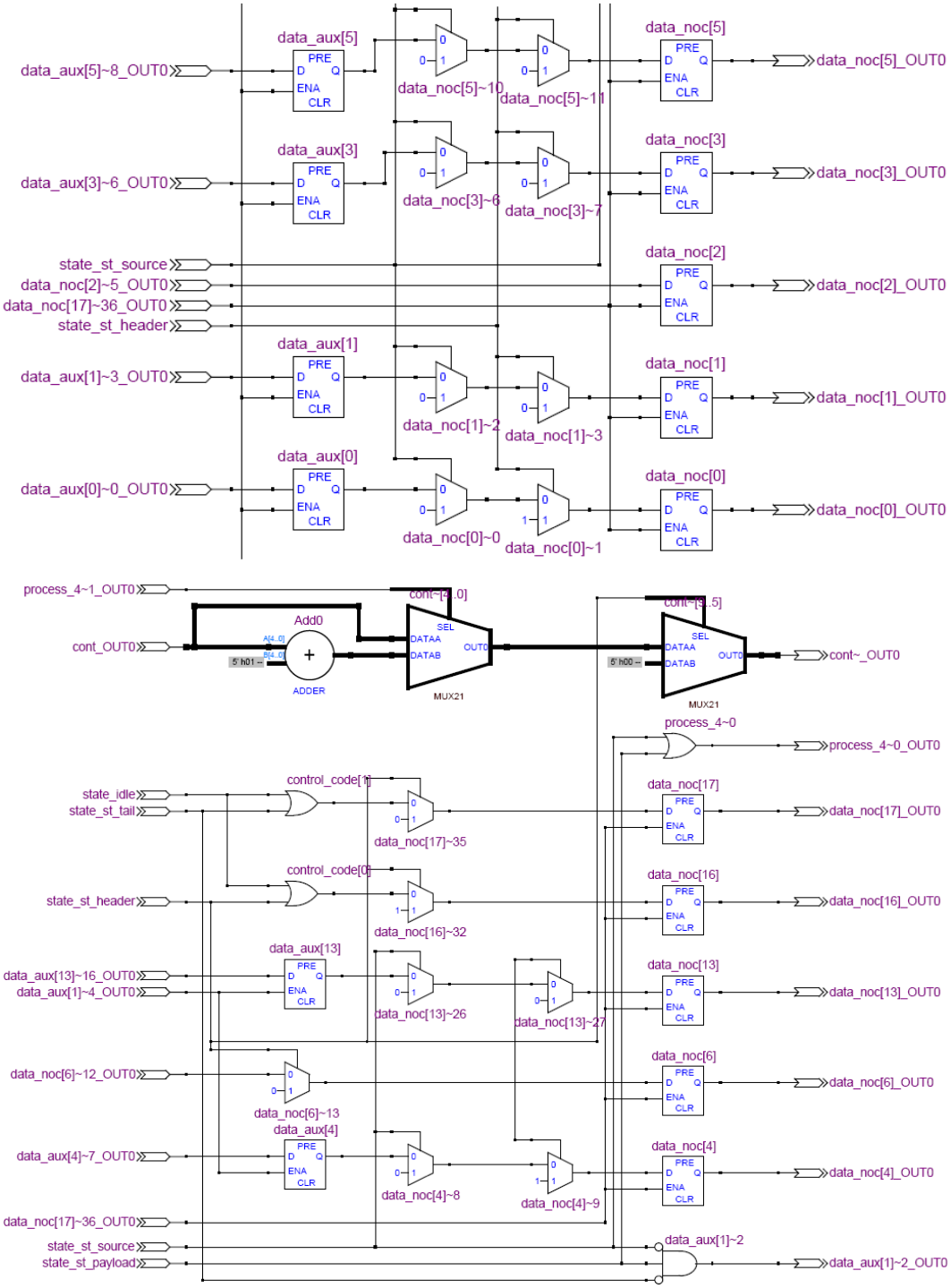
nas máquinas de estados anteriormente apresentadas nem na implementação do contador que é necessário para a definição do último flit. A garantia de parametrização com relação à profundidade dos *buffers* dos canais da NoC é concedida simplesmente verificando-se quando os flits podem ser enviados para rede, e essa decisão depende apenas da disponibilidade de espaço nos *buffers*.

As arquiteturas projetadas para a Unidade Empacotadora permitem que sejam utilizados qualquer tamanho de pacote, para qualquer profundidade de *buffers* dos canais de entrada da NoC, para qualquer profundidade de FIFOs definidas para a NI e com flexibilidade quanto a configuração da largura do canal da rede.

Por fim, na figura 6.13 é apresentada a arquitetura completa da Unidade Empacotadora desenvolvida para a solução de indexação dos flits apresentada na figura 6.12. O diagrama desta figura foi extraído a partir da ferramenta MAXPLUS II da Altera. Utilizando-se uma ferramenta de síntese, como a da Altera, normalmente são feitas algumas simplificações na lógica projetada a partir do código VHDL.



(a)




(b)

Figura 6.13: Diagrama Esquemático da Unidade Empacotadora.

### 6.1.5 Unidade Desempacotadora

A Unidade Desempacotadora tem por função receber os dados da rede e agrupá-los de forma a reconstituir o dado originado pelo módulo transmissor. Quando os flits chegam ao destinatário, as informações referentes ao cabeçalho, endereço do módulo transmissor e código de controle do flit são removidas e somente os bits referentes a dados são repassados para o módulo receptor. Similarmente a Unidade Empacotadora, a Unidade Desempacotadora possui uma máquina de estados e cada estado está relacionado ao tipo de flit que está sendo recebido. A máquina de estados utilizada para esta implementação está ilustrada na figura 6.14. Também foi utilizado um registrador para manter os flits salvos na palavra de dado a ser repassada para o bloco receptor até que todo o pacote tenha sido recebido.

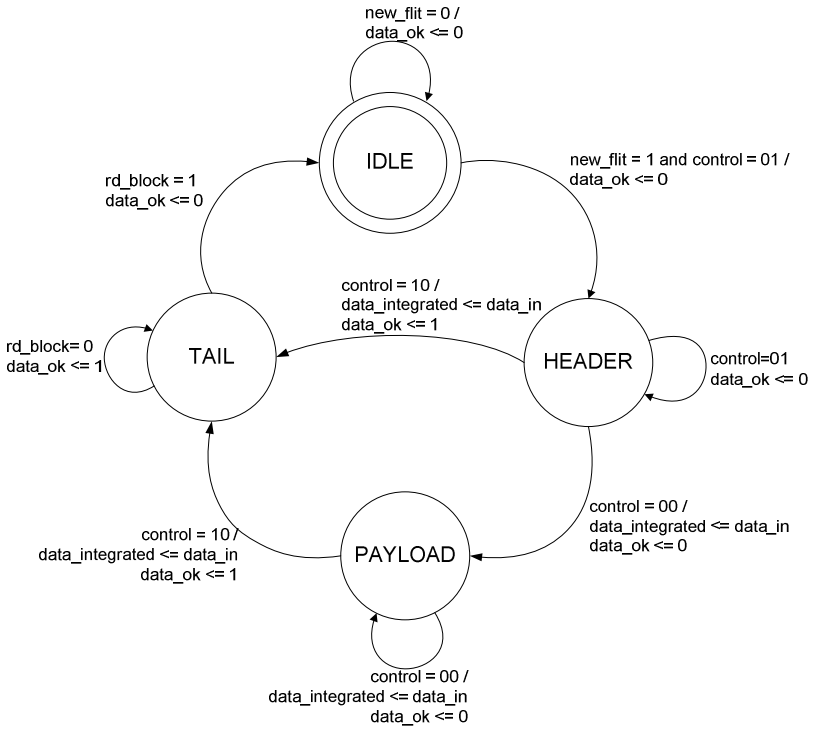



Figura 6.14: Máquinas de Estados da Unidade Desempacotadora.

A Unidade Desempacotadora também pode apresentar o estado de *SOURCE* quando for necessário identificar quem enviou o pacote para o núcleo. Nesse caso, o segundo flit indica o código do transmissor e de acordo com essa sinalização, o dado recebido é repassado para os respectivos sinais de entrada do módulo receptor. A FSM com a adição deste estado está ilustrada na figura 6.15.

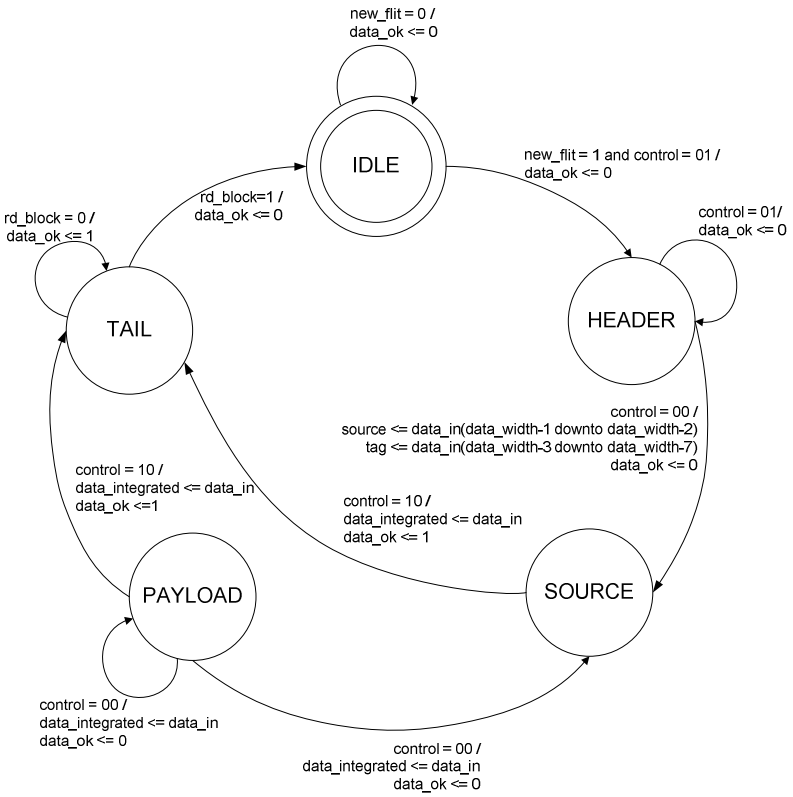



Figura 6.15: Máquinas de Estados da Unidade Desempacotadora com a definição do código do transmissor.

Sempre que a máquina de estados da Unidade Empacotadora possuir o estado de *SOURCE*, a máquina de estados da Unidade Desempacotadora também deve ter este estado para que a correta reconstituição do dado ocorra.

Da mesma forma que na Unidade Empacotadora, duas versões de indexação para a constituição da mensagem original foram implementadas e as mesmas ideias de indexação foram utilizadas nesta unidade. Na versão que utiliza variáveis para indexar a palavra de dados reconstituída utilizou-se um contador que define a posição de agrupamento dos dados recebidos no flit para compor o dado novamente. O contador utilizado nessa implementação está apresentado na figura 6.16. Este contador é atualizado sempre que um flit de dado for recebido e para isso duas condições são verificadas: se um novo flit for recebido (pelo sinal *new_flit* da figura 6.16) e se este se refere a um flit de dado do pacote (pelo sinal *flag_data*, como mostrado na figura 6.16). Atendendo a estas duas condições, o contador é incrementado e a partição de dado recebida pelo flit é adicionada a palavra de dados a ser enviada para o IP, até o recebimento do último flit do pacote.

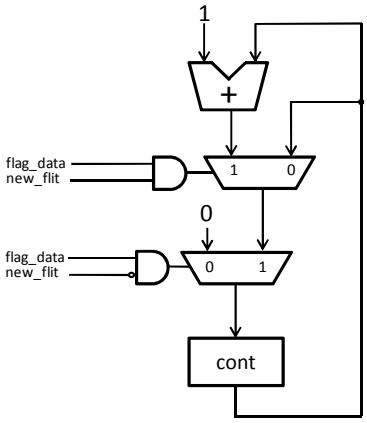



Figura 6.16: Contador utilizado na arquitetura da Unidade Desempacotadora.

Na versão que utiliza deslocamentos, quando o pacote não apresenta tamanho múltiplo da largura de dados da rede, a dificuldade foi saber quantos bits presentes no flit terminador correspondem à palavra de dados a ser reconstituída, apenas utilizando-se as constantes já definidas na arquitetura. Considerando-se as mesmas constantes utilizadas na indexação dos flits pela Unidade Empacotadora, a solução encontrada foi observar os bits da constante *PACKET_WIDTH*. Para esclarecer qual a estratégia utilizada, a figura 6.17 ilustra um exemplo da solução.

Conforme o esquema apresentado na figura 6.17, utilizou-se, como exemplo, largura do pacote igual a 34 bits, que em binário é igual a "100010". Considerando-se a largura de dados do canal igual a 8 bits, o tamanho do pacote não representa um múltiplo da largura do canal. Dessa forma, sendo a largura do canal igual a 8 bits, o seu valor de potência na base dois será igual 3 ($2^3$=8). Então, o valor formado pelos últimos 3 bits do *PACKET_WIDTH* representam o número de bits que são válidos no último flit. O valor da potência na base dois da largura do canal já era utilizado para definir os sinais *bit_end* e *bit_start* e é uma constante definida na arquitetura juntamente com a largura do canal. No exemplo da figura 6.17, nos últimos 3 bits para *PACKET_WIDTH* igual a 34, o número de bits válidos obtidos no último flit são 2 bits.

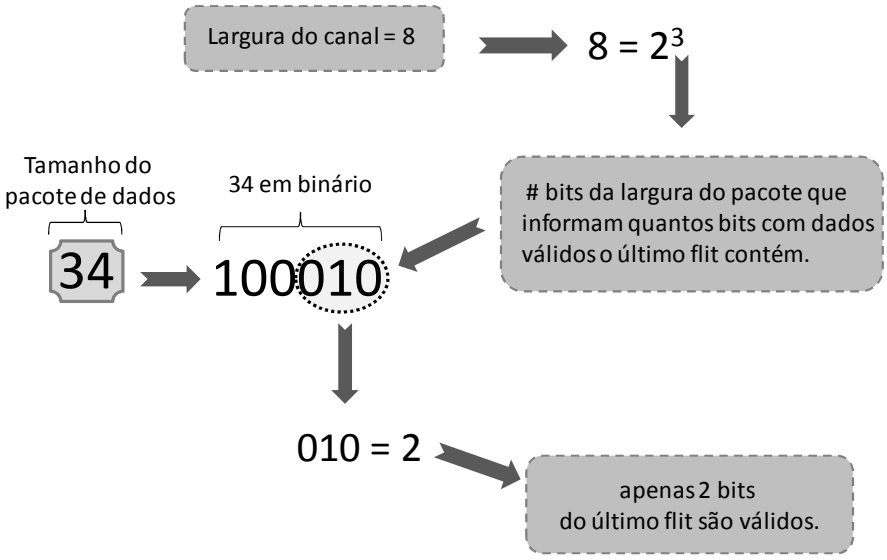



Figura 6.17: Exemplo da definição dos bits válidos no último flit.

A tabela 6.1, apresenta outros exemplos de como é realizado o cálculo para tamanhos aleatórios de pacotes a fim de se saber quantos bits do último flit são válidos. Conhecendo-se o número de bits válidos no último flit (que é uma constante definida a partir de *DATA_WIDTH*) e tendo-se os valores das constantes *DATA_WIDTH e PACKET_WIDTH* é possível reconstruir a mensagem original, conforme ilustrado na figura 6.18. A mesma proposta de deslocamentos dos flits é utilizada aqui, no entanto, neste caso, conforme eles são recebidos pela Unidade Desempacotadora, os dados são posicionados corretamente na variável que compõe a palavra de dados final. A posição inicial de recebimento dos flits é que precisa ser calculada em função das constantes acima comentadas. É utilizada uma variável genérica com um total de 128 bits (valor este que também é parametrizável) para a composição da mensagem original e que permite o recebimento de pacotes de qualquer tamanho dentro deste total. No entanto, no processo de síntese, só são considerados o número de bits que formam os pacotes de dados utilizados pela NI e que corresponde a maior palavra de dados enviada ao IP.

Tabela 6.1: Exemplos do cálculo utilizado para definir quantos bits válidos devem ser considerados no último flit.

| Tamanho do pacote | Valor em binário | Largura do canal da NoC | Valor da potência de 2 da largura do canal | Número de bits válidos no flit terminador decimal (binário) |
|---|---|---|---|---|
| 67 | 1000**011** | 8 bits | 3 | 3 (011) |
| 155 | 1001**1011** | 16 bits | 4 | 11 (1011) |
| 90 | 10**11010** | 32 bits | 5 | 26 (11010) |
| 231 | 11**100111** | 64 bits | 6 | 39 (100111) |

Quando a largura da palavra de dados de um módulo não for um valor múltiplo da largura de dados do canal da rede, para o recebimento e processo de deslocamento dos flits a fim de compor a mensagem original, precisa-se considerar a menor largura de dados que seja um múltiplo da largura do canal da NoC e que contemple todos os bits que correspondem a mensagem original. A figura 6.18 apresenta um exemplo desse caso. Neste exemplo, para um pacote com 34 bits e largura de dados da rede igual a 8, a largura de dados total que deve ser considerada para a reconstituição da mensagem deve ser igual a 40 bits. Por fim, apenas a mensagem composta de bits válidos, ou seja, dos bits que formam a mensagem original, é selecionada através da constante *PACKET_WIDTH*, conforme apresentado na última linha da figura 6.18.

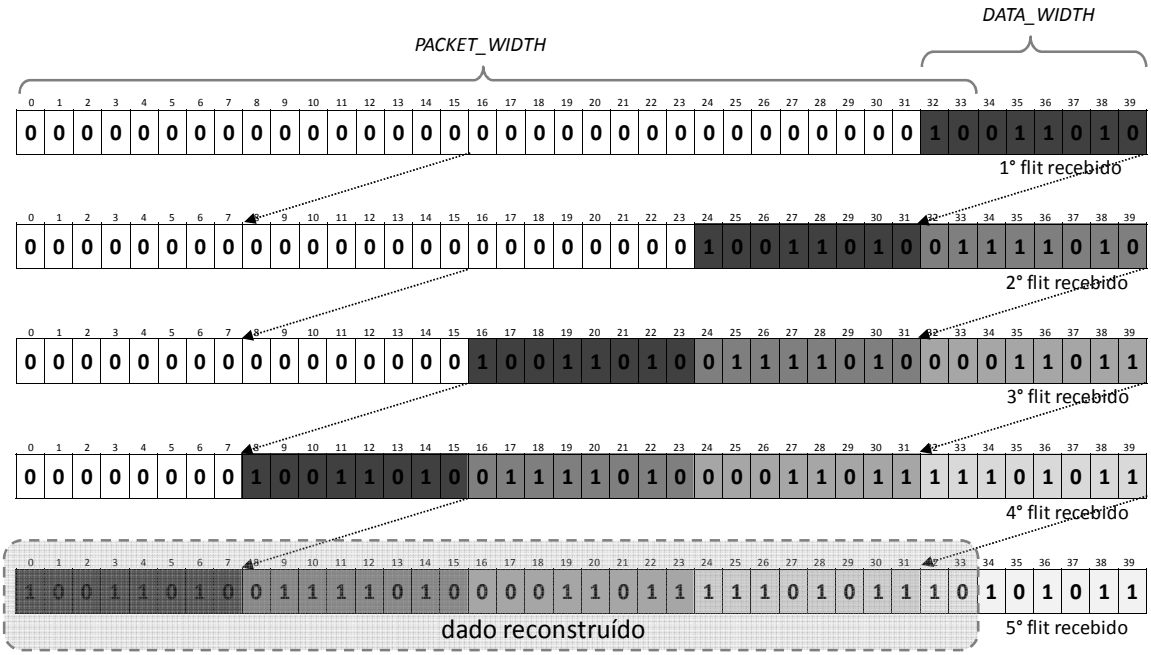



Figura 6.18: Exemplo de deslocamento dos dados para a formação da palavra de dados reconstruída.

Da mesma forma como apresentado na Unidade Empacotadora, a figura 6.19 apresenta a arquitetura da Unidade Desempacotadora desenvolvida para a solução de indexação dos flits apresentada na figura 6.18. O diagrama desta figura também foi extraído a partir da ferramenta MAXPLUS II da Altera. Como se pode observar neste diagrama, a arquitetura apresenta uma máquina de estados, um registrador de deslocamento e mais algumas portas lógicas que fazem o controle desse registrador de deslocamento, constituindo uma arquitetura bem mais simplificada que a obtida pela Unidade Empacotadora.

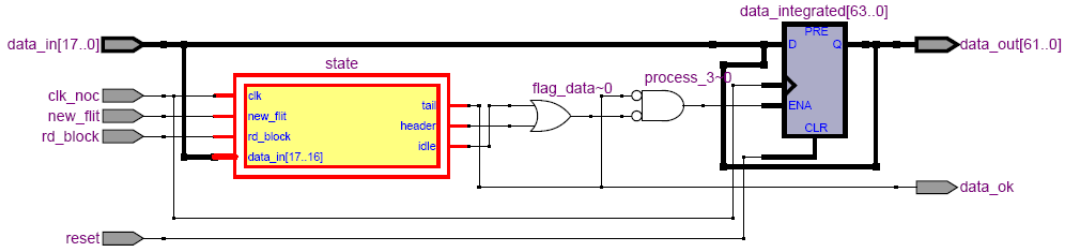



Figura 6.19: Diagrama Esquemático da Unidade Desempacotadora.

### 6.1.6 Resultados

As interfaces de redes foram desenvolvidas em VHDL e simuladas utilizando-se a ferramenta ModelSim. Após a verificação do funcionamento das mesmas, obtiveram-se os resultados de síntese para uma interface de rede genérica e verificaram-se as variações destes resultados quando alguns parâmetros de configuração foram alterados. A tabela 6.2 apresenta os resultados utilizando duas ferramentas de síntese: Design Compiler, da Synopsys e Leonardo Spectrum, da Mentor. Ambas as ferramentas apresentam síntese *standard cells* para a tecnologia 0,18um, no entanto, a diferença está

na apresentação dos resultados de área, sendo os resultados dados em um² para a primeira ferramenta e em *gates* (portas lógicas equivalentes a NANDs de 2 entradas), para a segunda.

Os resultados apresentados na tabela 6.2 foram obtidos para largura do canal da NoC igual a 16 bits e largura do pacote de dados igual a 32 bits. A definição destes parâmetros para a obtenção dos resultados deve-se ao fato de muitos processadores de uso geral apresentarem largura da palavra de dados igual a 32 bits. Além disso, na implementação do decodificador de vídeo, a largura da palavra de dados na saída do módulo INTRA e entrada do módulo *descrambler* também é de 32 bits, já que são obtidas 4 amostras de 8 bits por vez. Para os resultados de síntese obtidos, considerou-se o envio do flit com a definição do código do transmissor, sendo assim, a FSM, tanto da Unidade Empacotadora como da Unidade Desempacotadora, apresentam um estado a mais para identificação deste código.

Tabela 6.2: Resultados de síntese para *standard cells* de uma interface de rede genérica.

| | **Synopsys Design Compiler TSMC 0,18um** | | | | **Leonardo Spectrum TSMC 0,18um** | |
|---|---|---|---|---|---|---|
| **Profundidade da FIFO da NI** | **Área (um²)** | **Máxima Frequência** | **Potência @ 100MHz (mW)** | **Potência @ Max. Freq. (mW)** | **Área (gates)** | **Máxima Frequência** |
| **sem FIFO** | 21.339 | 407 | 0,77 | 3,11 | 882 | 701 |
| **4** | 54.930 | 412 | 2,30 | 9,46 | 2198 | 584 |
| **8** | 80.974 | 412 | 3,46 | 14,25 | 3227 | 553 |
| **16** | 132.522 | 412 | 5,46 | 22,48 | 5293 | 536 |

Conforme apresentado na tabela 6.2, os resultados de frequência obtidos com a ferramenta da Synopsys foram calculados através da inversão do tempo obtido para o caminho crítico. No entanto, quando se comparam os resultados obtidos com as duas ferramentas, observa-se que os valores de frequência apresentam diferenças de ferramenta para ferramenta. Conforme se pode observar nos resultados obtidos pela ferramenta Leonardo Spectrum, à medida que se aumenta a profundidade das FIFOs utilizadas na NI, obtém-se uma pequena redução nos valores de máxima frequência do circuito.

Apenas como comparação do tamanho de um roteador da SoCIN em relação às interfaces de rede desenvolvidas, tem-se que a área utilizada para um roteador com largura do canal da rede igual a 16 e profundidade das FIFOs do canal de entrada igual a 4 é de 115.215 um². Comparando-se a área da interface de rede desenvolvida com profundidade da FIFO igual a 4 na NI, a área obtida é aproximadamente igual a metade do tamanho de um roteador da NoC SoCIN.

Para a obtenção dos resultados de potência pela ferramenta da Synopsys realizaram-se duas análises, uma considerando-se o tempo obtido para o caminho crítico da implementação (nesse caso, utilizou-se para a obtenção dos resultados de potência a máxima frequência obtida para o circuito) e outra, considerando-se a frequência de 100MHz. A definição do valor de 100 MHz para a obtenção dos resultados de potência foi considerado em função de que, se uma aplicação for conectada por NoCs e NIs e todo esse sistema estiver operando sob a mesma frequência, dificilmente será utilizada a

máxima frequência obtida para as NIs. Um exemplo disso é o próprio decodificador de vídeo utilizado como estudo de caso neste trabalho, no entanto, ainda podem-se citar diversas outras aplicações que rodam a uma frequência de aproximadamente 100 MHz para a tecnologia de 0,18um, como é o caso do processador ARM7EJ-S (ARM, 2010) e que poderia, por exemplo, compor um nodo da NoC.

Conforme será detalhado no próximo capítulo, verificou-se que as profundidades das FIFOs utilizadas nas NIs variam conforme a necessidade da aplicação. Por esse motivo, analisaram-se os resultados para diferentes profundidades de FIFOs, a fim de verificar o impacto nos resultados de síntese para diferentes valores de profundidade. Obtiveram-se também os resultados de síntese para uma NI sem FIFO, ou seja, considerando-se uma NI com conexão a uma FIFO externa. Esta hipótese foi considerada, pois a FIFO pode fazer parte da própria implementação do IP, podendo ter sido adicionada ao módulo para facilitar a conexão com os demais IPs da aplicação. Além disso, os resultados de síntese da NI genérica com ou sem FIFO contribuem para que se obtenha uma noção do quanto de área é necessário quando se adiciona uma FIFO a arquitetura e o quanto de área é utilizada somente para a implementação das NIs, contendo as unidades Empacotadoras e Desempacotadoras e os circuitos de controle de fluxo de dados.

No início desse capítulo confirmou-se que, dependendo da aplicação, mais de uma FIFO pode ser requerida em cada interface de rede. Dessa forma, obtendo-se resultados de síntese para a NI com e sem FIFO, é possível também obter-se um valor aproximado do número de *gates* necessários a cada FIFO adicionada. A partir dos vários resultados de síntese obtidos para diferentes profundidades de FIFO utilizada na NI, verificou-se que aproximadamente 3700 *gates* são acrescidos quando se faz o uso de uma FIFO com profundidade igual a 4. Se a necessidade for por uma FIFO de profundidade igual a 8, são necessários aproximadamente 6800 *gates* para a adição de uma FIFO à arquitetura e 13800 *gates* são adicionados aos resultados de área da NI se a profundidade da FIFO for definida igual a 16. Esses valores de *gates* por profundidade das FIFOs foram obtidos para largura da palavra de dados da FIFO igual a 32.

Obtiveram-se ainda resultados de síntese para FPGA utilizando-se o dispositivo XC5VLX110 Virtex-5. Estes resultados estão apresentados na tabela 6.3 e foram obtidos para a mesma configuração de NI anteriormente considerada. Embora os resultados das tabelas 6.2 e 6.3 apresentem valores de frequência máxima próximos, deve-se salientar que tal proximidade dos valores se deve ao fato de que o dispositivo da família Virtex-5 é definido para a tecnologia de 65nm, enquanto que os resultados de tabela 6.2 são para a tecnologia de 180nm.

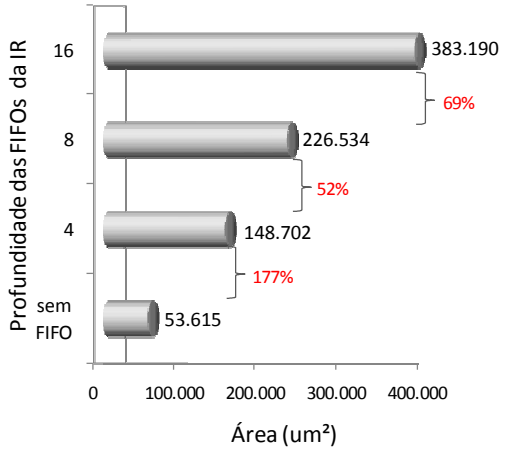
Tabela 6.3: Resultados de síntese para FPGA de uma interface de rede genérica.

| | **Virtex-5 XC5VLX110** | | | | |
|---|---|---|---|---|---|
| **Profundidade da FIFO da NI** | **Slice Registers** | **Slice LUTs** | **LUT/FF Pairs** | **BRAM/FIFO** | **Frequência Máxima (MHz)** |
| **sem FIFO** | 143 | 175 | 110 | 0 | 658 |
| **4** | 138 | 184 | 89 | 1 | 490 |
| **8** | 146 | 190 | 94 | 1 | 442 |
| **16** | 154 | 193 | 98 | 1 | 428 |

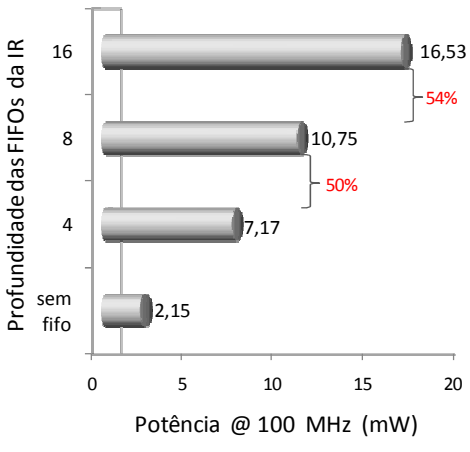
Além das análises acima realizadas, onde se alterou a profundidade das FIFOs utilizadas nas NIs, verificou-se que dois outros parâmetros também impactam nos resultados de síntese das mesmas, que são a largura do canal da NoC e a largura da palavra de dados do IP. Como a largura da palavra de dados do IP pode apresentar qualquer tamanho, principalmente quando se trata de um hardware dedicado, como é o caso dos módulos que compõem o decodificador de vídeo tratados nesse trabalho, decidiu-se verificar os resultados para uma largura da palavra de dados do IP que não fosse igual a um valor múltiplo da largura de dados do canal da NoC. Dessa forma, somente alguns bits do último flit são válidos, e com isso, foi possível validar a arquitetura para qualquer largura de palavra de dados do IP através do esquema de rotação dos dados apresentado nas subseções anteriores. Sendo assim, definiu-se uma largura de palavra de dados aleatória igual a 100, que é um tamanho que não conterá flits completos para nenhum dos valores de largura do canal da NoC analisadas nesse trabalho (foram analisadas larguras de canal igual a 8, 16 e 32 bits). Além disso, verificou-se o impacto dos resultados quando se varia a largura da palavra de dados do canal da NoC para estes valores.

A largura da palavra de dados do IP impacta nos resultados de síntese, pois este tamanho é o mesmo utilizado para a largura da palavra de dados da FIFO presente na NI. Já, a largura da palavra de dados da NoC também influencia nos resultados de síntese da arquitetura, pois é esse parâmetro que define a largura dos flits recebidos e enviados pela NI e por isso, alguns *flip-flops* foram utilizados na arquitetura para armazenamento dos mesmos. Os gráficos da figura 6.20 apresentam os resultados para largura da palavra de dados do IP igual a 100 e largura do canal da NoC igual a 16 para diferentes profundidades de FIFO. Os resultados apresentados na figura 6.20 (a) e 6.20 (b) foram obtidos com a ferramenta da Synopsys e os resultados da figura 6.20 (c) e 6.20 (d) foram obtidos com a ferramenta da Mentor.

Os porcentuais apresentados na figura 6.20 representam o quanto se obteve de aumento nos resultados de síntese em relação à profundidade de FIFO anteriormente considerada em cada caso. Como se pode perceber, os resultados de área obtidos para as duas ferramentas na tecnologia 0,18um variam em um proporção semelhante, apenas observa-se um aumento maior em área obtidos com a ferramenta da Mentor do que com a ferramenta da Synopsys quando se aumenta a profundidade da FIFO de 8 para 16.




(a)




(b)

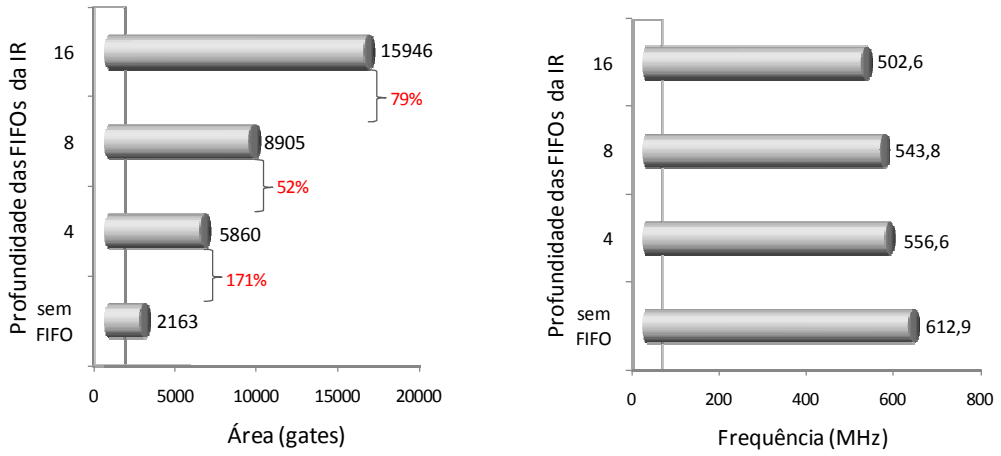
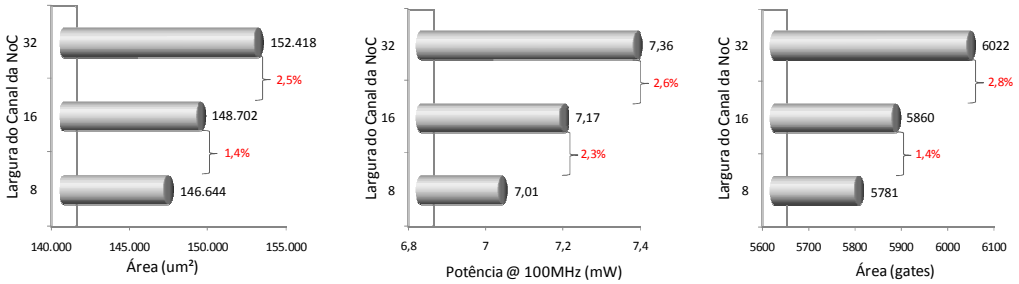



Figura 6.20: Resultados de síntese *standard cells* para a tecnologia 0,18um de uma NI variando a profundidade da FIFO da NI para largura do canal da NoC igual a 16 e largura da palavra de dados do IP igual a 100.

Com relação aos resultados de frequência apresentados na figura 6.20(c), observa-se novamente que ocorre uma redução nos resultados de síntese obtidos com a ferramenta da Mentor para o valor de máxima frequência do circuito quando a profundidade das FIFOs da NI aumenta.

Como já era imaginado, as FIFOs são os circuitos que mais impactam no consumo de potência das NIs, e como se pode perceber, aumentando-se a profundidade da FIFO obtém-se um aumento no consumo de potência de, no mínimo, 50%. No entanto, essas FIFOs precisam ser adicionadas às NIs para atenderem algumas necessidades dos EPs, como, por exemplo, para garantirem o desempenho da aplicação, para a sincronização de pacotes, dentre outros. Sendo assim, se considerarmos o consumo de energia (E=P.t, onde E=energia, P=potência e t =tempo), é possível obter uma profundidade de FIFO ideal para o desempenho da aplicação, sem consumir energia e potência desnecessariamente. Sendo assim, cada NI deve conter FIFOs com a menor profundidade possível que atenda à taxa de processamento exigida pelo módulo a qual ela está conectada.

A última análise realizada nessa etapa foi com relação à largura do canal da NoC. Os gráficos da figura 6.21 apresentam os resultados de síntese obtidos para largura da palavra de dados igual a 100, profundidade da FIFO igual a 4 e largura do canal da NoC igual a 8, 16 e 32. Os resultados da figura 6.21(a) e 6.21(b) foram obtidos com a ferramenta Design Compiler e os resultados de área em *gates* foram obtidos com a ferramenta Leonardo Spectrum. Como se pode perceber nos resultados apresentados na figura 6.21, tanto os valores de área como os de potência apresentam um pequeno aumento nos resultados de síntese da NI quando se varia a largura do canal da NoC. Já, com relação aos resultados de frequência obtidos com a ferramenta da Mentor não se obteve diferença nos resultados para diferentes largura de canal da NoC. Para os resultados obtidos na figura 6.21 (c) a frequência máxima de operação foi igual a 557MHz.




(a) (b) (c)

Figura 6.21: Resultados de síntese *standard cells* para tecnologia 0,18um de uma NI variando-se a largura do canal da NoC para profundidade da FIFO igual a 4 largura da palavra de dados do pacote igual a 100.

Quando se considera FIFO maiores, o impacto no acréscimo nos resultados de área e potência com relação à largura do canal da NoC é ainda menor. Isso ocorre, pois, conforme já foi comentado, a profundidade da FIFO apresenta uma influencia muito maior nos resultados de síntese do que a largura do canal da NoC. A figura 6.22 apresenta os porcentuais de aumento quando se passa a utilizar largura do canal da NoC igual 32 bits ao invés de 16 bits para diferentes profundidades de FIFO na NI. É importante observar neste gráfico que os porcentuais de aumento não se referem ao aumento em função da profundidade da FIFO. Os resultados apresentados na figura 6.22 ilustram os porcentuais de aumento quando a largura do canal da NoC é aumentada de 16 bits para 32 bits para cada uma das profundidades de FIFO utilizadas. Com isso, a informação obtida com este gráfico é de que quanto maior a profundidade da FIFO presente na NI, menor é o impacto para o aumento da largura do canal da NoC nos resultados de síntese da NI.

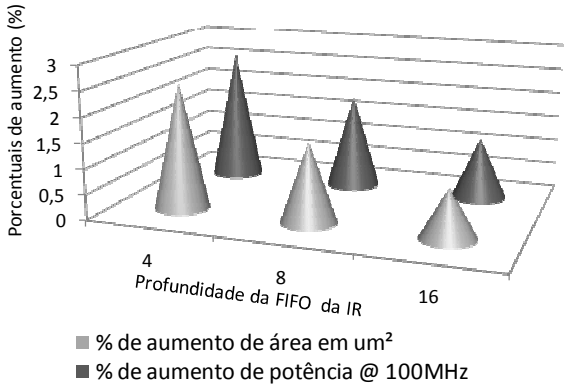



Figura 6.22: Porcentuais de aumento nos resultados de área e potência obtidos com a ferramenta Design Compiler para uma NI quando a largura do canal da NoC é aumentada de 16 para 32 bits, considerando-se diferentes profundidades de FIFO da NI.

### 6.1.7 Comparações da NI desenvolvida com os trabalhos relacionados

A partir dos trabalhos relacionados, procurou-se relacionar as principais características nos projetos de interfaces de rede consideradas nesse trabalho. A tabela

6.4 apresenta um resumo das propostas encontradas, comparando-as com a implementação desenvolvida nesse trabalho. Existem algumas propostas que são específicas para o uso de algum protocolo de barramento, já que muitos EPs existentes apresentam esse tipo de interface. No entanto, conforme já comentado, ao se fazer uso de protocolos de barramentos específicos e complexos, fica a dúvida de quanto de proveito está se obtendo no uso da NoC, já que circuitos complexos são implementados nessas conexões e o tipo de comunicação de um barramento é reproduzido na NoC.

Todas as interfaces de rede apresentam uma arquitetura mais complexa que a desenvolvida nesse trabalho. Como só foram utilizados IPs na implementação do decodificador, não foi preciso adicionar alguns circuitos de controle necessários quando se faz o uso de um processador. No entanto, uma primeira abordagem de interface de rede foi apresentada, permitindo a conexão com outros IPs de forma simples e de fácil compreensão. Quando for necessário o uso de um processador que apresente um protocolo específico, facilmente pode ser adaptado um *wrapper* a esta implementação.

Tabela 6.4: Resultados comparativos das diferentes propostas de interfaces de rede encontradas na literatura.

| | **Roteamento permitido** | **Uso de protocolo específico** | **Largura do pacote de dados** | **Largura do canal da NoC** | **Desvantagens** |
|---|---|---|---|---|---|
| **(RADULESCU, 2004)** | Determinístico | Sim (transaction-based) | Parametrizável | Parametrizável | • Penalidade em desempenho. NI apresenta uma latência de até 10 ciclos. |
| **(SINGH, 2006)** | Determinístico | Não | Parametrização Limitada | Parametrizável | • Uso de muitos recursos de armazenamento;<br>• Necessita de muitos bits utilizados no cabeçalho. |
| **(FERRANTE, 2008)** | Determinístico | Sim (OCP) | Não informado | Não informado | • Necessita utilizar árbitros para definir qual IP pode utilizar os recursos da NI. |
| **(LEE, 2008)** | Adaptativo | Prevê o uso de protocolos do tipo mestre - escravo | Parametrizável | Parametrizável | • Necessita de um grande número de registradores e tabelas para a utilização das NIs. |
| **(ATTIA, 2009)** | Determinístico | Sim (OCP) | Parametrização Limitada | Fixa | • Uso de *wrappers* específicos;<br>• Apenas pacotes múltiplos de 32 bits. |
| **(EBRAHIMI, 2009)** | Adaptativo | Sim (AXI) | Parametrizável | Não informado | • Uso de *wrappers* específicos.<br>• Validada por tráfego sintético. |
| **Interface de rede desenvolvida** | **Determinístico** | **Não** | **Parametrizável** | **Parametrizável** | |

Além disso, percebeu-se que todas as interfaces de rede foram validadas utilizando aplicações com pouca comunicação, ou seja, nodos que recebem dados que foram enviados por apenas um único EP ou, EP que enviam dados para somente um nodo destinatário. Ainda, tiveram outras aplicações que foram validadas utilizando tráfego sintético. Sendo assim, algumas situações não previstas podem ser encontradas para ambos os casos, como ocorreu na implementação do decodificador de vídeo H.264.

A tabela 6.5 apresenta alguns resultados de síntese obtidos para as arquiteturas dos trabalhos relacionados. Nem todos os trabalhos apresentaram esses dados, sendo que na tabela 6.5 informaram-se apenas os dados que foram obtidos na literatura, sendo que diferentes tecnologias foram utilizadas pelos várias propostas, impedindo uma comparação mais direta com a arquitetura proposta nesse trabalho. No entanto, algumas conclusões podem ser obtidas a partir dessa tabela. Como se pode perceber, a solução apresentada, por ser a mais simples, é também a que utiliza menos recursos de área, se considerarmos a tecnologia utiliza na síntese da NI desenvolvida.

Quanto aos resultados de frequência máxima, a arquitetura apresenta um valor elevado se comparado com outras propostas que utilizam tecnologias menores de implementação, e que dessa forma, obtém resultados maiores de frequência.

Tabela 6.5: Resultados de síntese para as diferentes propostas de interfaces de redes encontradas na literatura.

| | **Tecnologia** | **Área Total (*Gates* ou LUTs)** | **Frequência Máxima (MHz)** |
|---|---|---|---|
| **(BHOJWANI, 2003)[1]** | 180 nm | 13.000 gates | 185 MHz |
| **(RADULESCU, 2004)[2]** | 130 nm | 110.000 um² | 500 MHz |
| **(FERRANTE, 2008)[3]** | ST 65 nm | ~15.000 um² | 599 MHz |
| **(LEE, 2008)** | TSMC 90 nm | 53.258 um² | 719 MHz |
| **(ATTIA, 2009)** | FPGA Virtex-5 | 241 LUTs/FF – NI mestre<br>552 LUTs/FF – NI escravo | 463 MHz – NI mestre<br>354 MHz – NI escravo |
| **Interface de rede desenvolvida** | TSMC 180 nm | 54.930 um²<br>2198 gates | 584 MHZ |
| | FPGA Virtex-5 | 89 LUTs/FF | 490 MHZ |

[1]Apenas a etapa de empacotamento, utilizando o processador configurável Xtensa.
[2]Dados consideram apenas o módulo *NI Kernel*
[3]Dados aproximados obtidos a partir de gráficos.

A comparação com relação aos resultados de síntese entre as arquiteturas de NIs encontradas na literatura se torna complicado por que cada implementação apresenta configurações diferentes de profundidade de *buffers* e memórias, além de não apresentarem soluções para os mesmos requisitos. Sendo assim, a tabela 6.5 serve apenas para se ter conhecimento do quanto cada proposta utiliza de recursos e a frequência máxima obtida para a arquitetura, em função dos serviços que cada uma está disposta a atender.

## 6.2 Soluções de Sincronização

Conforme comentado ao longo deste trabalho, embora existam algumas propostas de interfaces de rede para NoC encontradas na literatura, alguns problemas em interconexões com NoC ainda não estão totalmente solucionados. Em aplicações de multimídia, por exemplo, precisa-se garantir que a correta sequência de dados seja entregue ordenadamente a cada núcleo. Dessa forma, em determinados casos, pode ser requerida uma solução de sincronização dos dados. O problema de sincronismo de dados em aplicações de fluxo de dados síncronos foi primeiramente abordado em (LEE, 1987) e a solução utilizada por ele foi o uso de bufferização estática, onde cada nodo da aplicação pode iniciar o seu processamento assim que um número suficiente de dados estiver disponível nas suas entradas (chamado também de *tokens*). Sendo assim, é possível determinar estaticamente os requisitos de buffer (profundidade das FIFOs) necessários para a aplicação. Partindo desse mesmo conceito, o uso de FIFOs para sincronizar os dados em NoCs foi proposto na subseção 6.1.2. Dessa forma, conforme já comentado na subseção 6.1.2, sempre que um módulo depender de dados de diferentes transmissores para iniciar a computação, o uso de FIFOs para cada um dos nodos transmissores na NI pode ser requerido para prover sincronismo entre os dados recebidos pela rede e que devem ser repassados para o módulo.

O problema de sincronismo de dados tem sido amplamente discutido em sistemas que utilizam GALS (BEIGNÉ, 2006), (SHEIBANYRAD, 2007), (KUNDU, 2007) (THONNART, 2009), (NING, 2009). As arquiteturas GALS têm sido propostas para solucionar problemas relacionados à distribuição de relógio síncrono para todo o chip sem a ocorrência de variações no relógio (efeito este também conhecido como escorregamento do sinal do relógio ou *clock skew,* do inglês). No entanto, nestes casos, o principal problema é a possibilidade de falha de sincronização entre dois domínios de relógios diferentes, problema este tratado como metaestabilidade. Problemas de sincronização também podem ocorrer mesmo quando a mesma frequência de relógio é utilizada, porém em fases diferentes, situação esta conhecida como uso de domínio de relógios mesócronos.

Uma solução simples de sincronização para prover tolerância à variação do relógio quando diferentes domínios de relógio são definidos em um sistema é o uso de FIFOs ou *latches* com dois sinais de relógio, um para a recepção e outro para a transmissão de dados (CHAKRABORTY, 2002). É muito comum o uso de FIFOs para essas soluções, pois, dessa forma, é possível desacoplar as decisões entre o transmissor e o receptor e maximizar a vazão da aplicação (*throughput*) (ONO, 2009). Baseado nessas ideias, em (LUDOVICI, 2009) foi proposto um sincronizador para solucionar tanto problemas de relógios mesócronos como o uso de relógios independentes entre os nodos. Este sincronizador apresenta um conjunto de *latches* controlados por um contador e chaveados por um relógio de recepção. Os dados de saída são enviados para um *flip-flop* por um multiplexador que é chaveado por um relógio de transmissão. Soluções semelhantes também foram propostas em (KUNDU, 2007), (NING, 2009) através do uso de FIFOs assíncronas.

Outra solução de interface de rede para NoCs usando GALS para sincronizar domínios de relógio assíncronos foi proposta em (BEIGNÉ, 2006). Neste caso, eles propuseram interfaces baseadas no uso de FIFOs com o uso do código de Gray para os ponteiros de leitura e escrita da FIFO. Segundo os autores, essa solução foi adotada

porque no código Gray apenas um único bit é modificado entre dois valores sucessivos, evitando problemas de metaestabilidade e valores transientes errôneos que podem ocorrer em um código binário. No entanto, mais tarde, os mesmos autores propuseram uma outra solução com FIFOs para este mesmo problema, argumentando que o código Gray apresenta limitações com relação à complexidade da implementação. Como nova solução eles usaram codificação Johnson para os ponteiros de escrita e leitura das FIFOs (THONNART, 2009).

Uma proposta chamada NIUGAP usa o código de Gray para reordenar pacotes em NoCs (KIM, 2006). Nesse caso, o código de Gray é gerado e adicionado aos cabeçalhos dos pacotes como um rótulo (*tag*). Cada pacote tem uma *tag* de tempo e uma *tag* de sequência. Nesse caso, a reordenação é feita no destino, conforme os pacotes chegam, considerando as *tags* utilizadas na geração dos pacotes. Essa necessidade de reordenamento ocorre quando se tem um roteamento adaptativo em NoCs, em que não se sabe qual a ordem que os pacotes chegam ao destino, pois cada pacote pode ter seguido um caminho diferente. Similarmente a uma das propostas de sincronização apresentadas nesse trabalho, NIUGAP utiliza sequência de *tags* nos pacotes. No entanto, essa solução não ataca o mesmo problema de sincronização de pacotes relatada anteriormente.

A necessidade de sincronização para associar pacotes de diferentes nodos quando se utiliza NoCs não foi encontrada na literatura. Nesse trabalho, foi verificada essa necessidade na implementação do decodificador de vídeo segundo o modelo H.264, no entanto, esta necessidade pode ser encontrada em diversas aplicações com comportamento em fluxo de dados. A necessidade de sincronização apresentada aqui não foi relatada nem mesmo em alguns trabalhos encontrados na literatura que utilizam uma NoC para interconectar os módulos do decodificador de vídeo H.264 (XU, 2006), (AGARWAL, 2008), (CHANG, 2008).

Os módulos que compõem o decodificador de vídeo são heterogêneos entre si, ou seja, apresentam larguras de dados e larguras de banda diferentes entre si. Podem-se perceber essas diferenças analisando as taxas de comunicação dos módulos do decodificador de vídeo, conforme apresentado na figura 6.23. A figura 6.23 apresenta respectivamente, as taxas de comunicação mínima, média e máxima de cada módulo. Como se pode observar, as taxas de comunicação dos módulos do decodificador não são constantes. Além disso, verificou-se que as amostras preditas pelo bloco INTRA ocorrem em rajadas, já que a cada pacote enviado pelo *parser* é possível realizar a predição de um macrobloco (16 x 16 amostras). O módulo de predição INTRA depende dos dados enviados pelo *parser* e do retorno das amostras finais pelo bloco de soma. Sendo assim, ao concluir a predição de um macrobloco, um novo pacote enviado pelo *parser* para o INTRA deve ser recebido para que novas predições possam ser realizadas. No entanto, não se pode garantir que um novo pacote esteja sempre disponível após o término da predição de um macrobloco, já que para isso depende-se do tempo de processamento do *parser* e da disponibilidade da rede para envio do pacote do *parser* para o INTRA. Todas estas dependências interferem no comportamento do decodificador em NoC e, dessa forma, necessita-se prover um meio de sincronizar as amostras preditas com os resíduos, já que os módulos não apresentam a mesma taxa de comunicação e ainda dependem da disponibilidade da NoC para envio dos pacotes e da latência no envio dos mesmos pela rede.

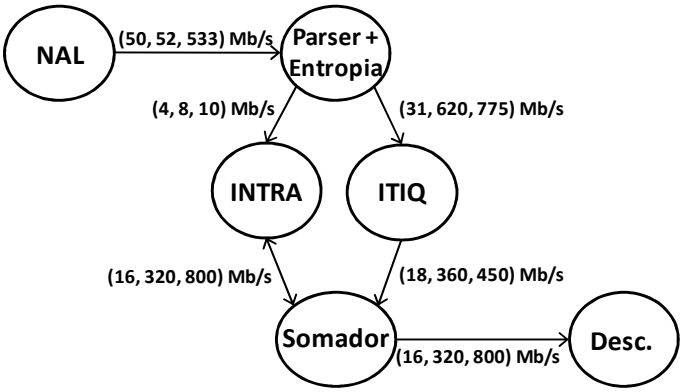



Figura 6.23: Taxas de comunicação dos módulos do decodificador de vídeo H.264.

Conforme já comentado, no processo de decodificação de vídeo, as amostras preditas são somadas aos resíduos que foram calculados na etapa de codificação referente à predição obtida. Cada amostra predita tem uma amostra de resíduos equivalente obtida no processo de codificação. Na etapa de decodificação, o módulo ITIQ decodifica os resíduos e os envia ao somador para que os mesmos sejam adicionados às amostras preditas. Sendo assim, no processo de decodificação, cada amostra predita deve ser somada a amostra de resíduo respectiva.

O processo de predição INTRA apresenta 9 modos de predição para blocos Luma 4x4 e 4 modos de predição para blocos Luma 16x16 e ainda mais 4 modos para predição Croma 8x8. Os modos de predição são definidos para cada macrobloco na etapa de codificação de vídeo. Devido as diferentes possibilidades de modos de predição para cada tamanho de bloco, o tempo de processamento da predição INTRA no decodificador de vídeo não é constante. Além disso, quando se utiliza uma NoC não se garante que os canais da rede estarão disponíveis para envio dos pacotes na mesma taxa em que os dados são gerados pelo módulo, e dessa forma, a taxa de recebimento dos dados no módulo receptor é variável, já que os canais da rede podem ser compartilhados por diferentes módulos conectados.

Considerando-se todos os fatores acima relacionados quando se considera uma NoC, a necessidade de sincronização de pacotes pode ser requerida em diferentes implementações e a necessidade de associação de dados para início do processamento de um módulo é muito comum em aplicação DSP (Processamento de Sinal Digital, do inglês, *Digital Signal Processing*).

Quando um módulo depende simultaneamente do recebimento de dados de diferentes módulos para iniciar o processamento, uma solução de sincronização será requerida sempre que os módulos apresentarem taxas de processamento distintas, domínios de relógio diferentes ou ainda se os módulos apresentarem comportamento de tráfego irregular (MATOS, 2010). Quando se utiliza uma NoC para a conexão dos módulos, a necessidade de um circuito de sincronização se faz necessário simplesmente pelo fato de que não se tem a garantia de envio dos pacotes na mesma taxa em que eles são gerados pelos módulos, necessitando-se de um circuito que garanta esse correto funcionamento.

Além da solução de sincronização de pacotes apresentadas na subseção 6.1.2, onde se utiliza FIFOs na NI para armazenamento dos dados referente a cada nodo que o enviou, uma outra solução para este problema foi analisada. Essa segunda proposta é baseada em rótulos e será apresentada na subseção a seguir.

### 6.2.1 Solução de Sincronização utilizando Rótulos

O modelo computacional utilizando rótulos (ou *tags*) foi primeiramente proposto por Lee e Sangiovanni-Vincentelli (LEE, 1996). Os autores representaram um evento por um par (valor, rótulo). Sendo assim, dado um conjunto de valores *V* e um conjunto de rótulos *T,* um evento *e* é definido como um produto cartesiano *T* x *V*. Dessa forma, cada evento terá sempre um valor $v_i$ associado a um rótulo $t_j$.

O rótulo é uma identificação dada a uma mensagem de forma que permita a ordenação dos elementos computacionais (LEE, 1996). Um conjunto *T* de rótulos é atribuído seguindo uma ordenação, permitindo que seus valores possam ser utilizados para definir a ordem dos eventos. Sendo assim, definem-se eventos concorrentes como aqueles que apresentam a mesma indicação de rótulo (uma revisão desta técnica também foi abordada em MARCON, 2005).

Utilizou-se a definição de rótulos como solução de sincronização de pacotes em NoCs. Sendo assim, as amostras preditas e os resíduos, como se tratam de eventos concorrentes, apresentam o mesmo rótulo, e estes rótulos indicam a ordem de cada evento no tempo. Dessa forma, as amostras de predição apresentam o mesmo rótulo que a sua respectiva amostra de resíduo, sendo possível, através dessa indicação, associar dados correspondentes.

A solução proposta para a sincronização de pacotes pela interface de rede é utilizar um circuito sincronizador que envia os dados referentes a cada módulo para uma pequena memória composta de um conjunto de *flip-flops*. O módulo sincronizador verifica se os dados referentes a cada transmissor apresentam o mesmo rótulo. Sempre que os dados que precisam ser sincronizados apresentarem o mesmo rótulo, os dados podem ser repassados para o IP. A figura 6.24 apresenta como o módulo sincronizador se comunica com os demais módulos da interface de rede.

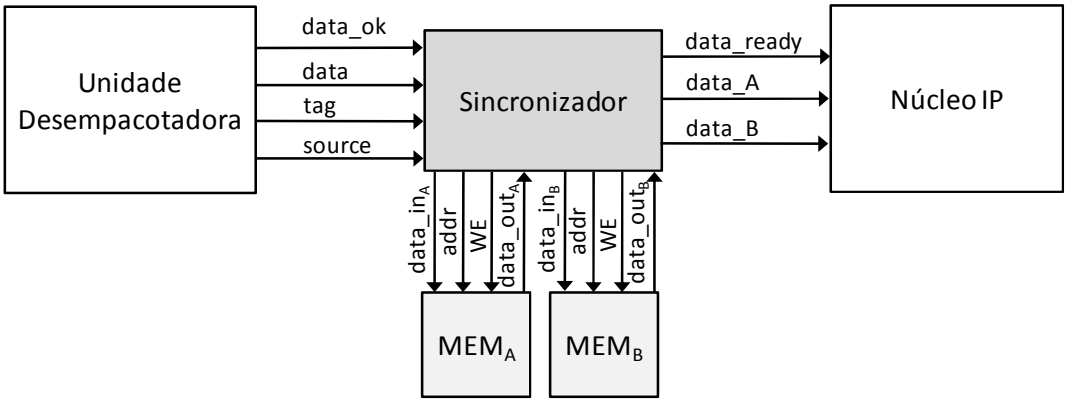



Figura 6.24: Conexão do módulo sincronizador às memórias e aos demais módulos que compõem a NI.

Conforme apresentado na figura 6.24, primeiramente a Unidade Desempacotadora repassa os dados recebidos pela rede para o módulo sincronizador. Juntamente com os dados, são informados ao sincronizador, o código do transmissor (*source*) e o rótulo do dado (*tag*). Tanto o código do módulo transmissor do pacote (*source*) como o rótulo (*tag*) são informados no segundo flit do pacote, conforme apresentado na figura 6.9 e nas máquinas de estados das figuras 6.8 e 6.15.

A figura 6.24 apresenta um exemplo genérico de sincronização. Sempre que um novo pacote é recebido pela Unidade Desempacotadora, antes mesmo que todos os flits tenham sido recebidos pela NI, já é informado para o módulo sincronizador o rótulo do pacote e o código do transmissor no segundo flit recebido. Sendo assim, após o recebimento do último flit, automaticamente o dado é enviado para o módulo sincronizador e conforme o módulo que o enviou (*source*), os dados são repassados para os *buffers* de memória correspondentes ($data\_in_A$ ou $data\_in_B$, conforme ilustrado na figura 6.24). No entanto, o dado só é escrito na memória se o módulo concorrente não tiver enviado nenhum pacote com o mesmo rótulo. Se o módulo concorrente apresenta dados com o mesmo rótulo, este dado é lido e os dados referentes aos dois módulos transmissores são repassados para o IP presente na NI.

O endereço utilizado para endereçamento dos dados nas memórias (*addr*) é o próprio rótulo (*tag*). Nesse caso, este endereço é informado no recebimento do segundo flit e é o mesmo endereço para acessar as duas memórias, ou seja, as memórias de cada um dos módulos transmissores. O fluxograma da figura 6.25 apresenta o funcionamento do sincronizador para a implementação do decodificador de vídeo. Imagine que um dado referente ao módulo INTRA tenha sido enviado para o módulo sincronizador. Nesse caso, é verificado na posição de memória do ITIQ para o endereço igual ao rótulo informado pelo pacote do INTRA, se existe um dado válido para ITIQ. A validade de cada dado armazenado na memória é informada através do bit mais significativo da palavra de dados. Sendo assim, sempre que o dado armazenado na memória for um dado válido, este bit é igual a 1. Se o bit mais significativo do endereço da memória do ITIQ igual ao rótulo recebido for igual a 1, esse dado é repassado para o módulo somador juntamente com os dados recebidos pelo módulo INTRA e um sinal identificando que os dados estão válidos (*data_ready*) é enviado ao somador. Nesse caso, sempre que um dado válido for lido da memória, o bit de validade precisa ser zerado, e para isso, escreve-se uma palavra com todos os bits iguais a zero na mesma posição de memória. Esse processo de escrita é feito paralelamente ao envio de dados para o somador.

A memória utilizada não apresenta sinal de leitura, pois o dado é repassado diretamente para a saída conforme o endereço informado. Sendo assim, apenas um sinal de escrita na memória precisa ser controlado. Para controle do sinal de escrita nas memórias, uma máquina de estados simples foi utilizada. A máquina de estados implementada para esse controle está ilustrada na figura 6.26. Como a verificação do bit de validade da memória do módulo concorrente só depende da informação do endereço (*tag*), este já é repassado ao sincronizador assim que o mesmo é identificado pela Unidade Desempacotadora. Quando todos os flits do pacote tiverem sido recebidos pela NI, o sinal de *data_ok* passa a ser igual a 1 e a máquina de estados da figura 6.26 é ativada. Assim que o sinal *data_ok* estiver ativo, duas possibilidades de escrita na memória são possíveis. Se o bit de validade for igual a um, indicando que o dado armazenado na memória refere-se à mesma sequência do dado recebido pelo pacote concorrente, após a leitura desse dado, é preciso zerar esta posição de memória. Se o bit de validade for igual a zero, significa que o dado do módulo concorrente para o mesmo rótulo do pacote recebido ainda não foi armazenado na memória. Nesse caso, o dado recebido pela NI é armazenado na sua memória correspondente, com o bit de validade igual a 1 e esse dado fica armazenado até que o módulo concorrente envie um dado com o mesmo rótulo.

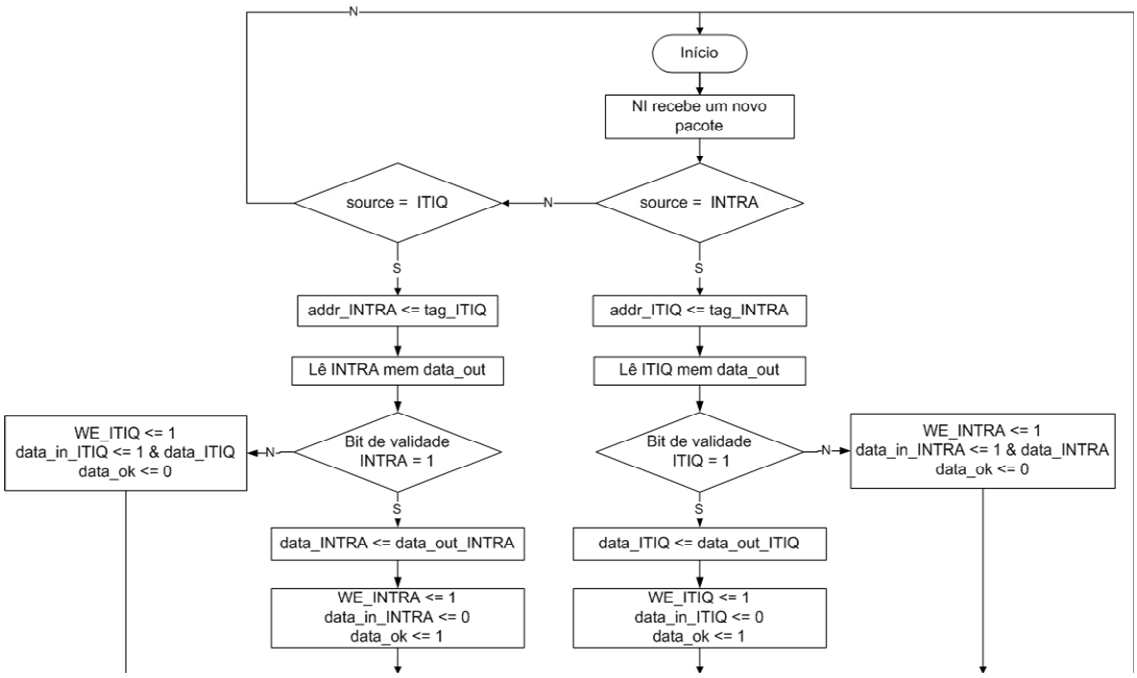



Figura 6.25: Fluxograma do algoritmo implementado pelo módulo sincronizador.

Exemplificando-se a explicação anterior, quando o módulo INTRA enviar um pacote para a NI, se para o rótulo do pacote recebido, o bit de validade na memória do módulo ITIQ for igual a zero, o dado referente ao módulo INTRA é armazenado na sua memória correspondente. Caso o bit de validade na memória do módulo ITIQ para o endereço recebido seja igual a 1, o dado armazenado na memória referente ao módulo ITIQ é enviado para a saída do sincronizador e esta posição de memória é zerada (MATOS, 2010).

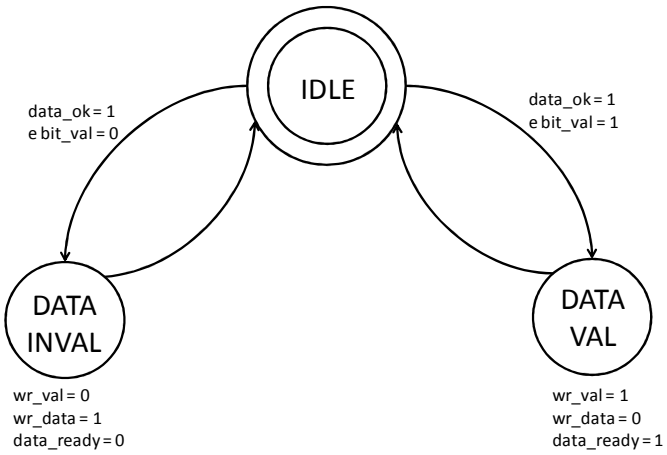



Figura 6.26: Máquina de estados que controla a escrita nas memórias utilizadas pelo módulo sincronizador.

Considerando-se a máquina de estados da figura 6.26, dois sinais foram utilizados para escrita nas memórias (*wr_val* e *wr_data*), um para cada uma das situações anteriormente comentadas. Estes sinais podem ser utilizados para controlar a escrita em

qualquer uma das duas memórias. A definição de qual sinal de escrita é enviado para cada memória é definido conforme o código do transmissor que enviou o pacote (*source*). O sinal *wr_val* é ativo quando o dado na memória correspondente é válido. Dessa forma, esse sinal habilita a escrita de zeros na posição de memória do módulo correspondente indicado no rótulo do pacote recebido, informando que a posição está vazia.

Seguindo o mesmo exemplo anteriormente apresentado, se um pacote referente ao módulo INTRA tiver sido recebido pela NI, então, se o dado lido na memória do módulo ITIQ for válido, o sinal *wr_val* será repassado para o *WE* da memória ITIQ, ativando a escrita de uma palavra com todos os bits iguais a zero. Em uma situação inversa, se o pacote recebido fosse do módulo ITIQ, então o sinal *wr_val* controlaria a escrita na memória do INTRA. Já o sinal *wr_data* controla a escrita do próprio dado recebido. Dessa forma, se foi recebido um pacote referente ao módulo INTRA e a palavra de dados armazenada na memória ITIQ para a posição referente ao rótulo recebido não for válida, então a palavra de dados recebida é armazenada na memória utilizada para o módulo INTRA e quem define essa escrita é o sinal *wr_data*. Se o pacote recebido fosse referente ao módulo ITIQ, então o sinal *wr_data* controlaria a escrita na memória utilizada para o módulo ITIQ. Sendo assim, um processo paralelo à implementação da FSM indica para qual memória cada um dos sinais que habilitam a escrita devem ser repassados, conforme a indicação do módulo transmissor (*source*). O pseudocódigo ilustrado na figura 6.27 apresenta como são repassados os sinais de escrita da memória definidos na máquina de estados da figura 6.26, conforme o pacote recebido.

```
1  SE source == INTRA ENTÃO
2      WE_INTRA = wr_data;
3      WE_ITIQ = wr_val;
4  SE source == ITIQ ENTÃO
5      WE_ITIQ = wr_data;
6      WE_INTRA = wr_val;
```

Figura 6.27: Pseudocódigo para controle de escrita nas memórias utilizadas pelo módulo sincronizador a partir dos sinais gerados na máquina de estados da figura 6.26.

### 6.2.2 Resultados

Os resultados de síntese da implementação em VHDL da NI com o módulo sincronizador está apresentada na tabela 6.6. Os resultados estão apresentados com e sem as memórias. Dessa forma, consegue-se separar a área utilizada somente para a NI com o sincronizador, já que o número de posições da memória deve ser calculada conforme as necessidades da aplicação em questão. Os resultados de área em *gates* foram obtidos com a ferramenta Leonardo Spectrum, já os resultados de potência, frequência e área em um² foram obtidos com a ferramenta Design Compiler. A frequência apresentada é o inverso do caminho crítico obtido pela ferramenta Design Compiler, no entanto, na ferramenta Leonardo Spectrum, o valor de frequência máxima obtida foi de 563MHz, essa diferença nos resultados de frequência se deve ao fato da ferramentas de síntese acrescentar *flip-flops* em algumas etapas do circuito. No entanto, seria possível aumentar a frequência de operação no próprio código simplesmente acrescentando um *flip-flop* na saída da Unidade Desempacotadora, de forma a permitir que novos flits sejam recebidos enquanto o processo de sincronização está sendo executado.

Para os resultados da tabela 6.6, a largura do canal da NoC foi definida para 16 bits. A largura da palavra de dados definidas para as memórias é de 33 bits (32 bits de dados mais 1 bit de validade). Como são obtidas 4 amostras na saída do módulo INTRA, tem-se ao total 32 bits de dados. Nesta implementação, definiu-se 4 bits para a identificação do rótulo, já que, analisando-se a implementação do decodificador de vídeo, verificou-se que nenhum dos módulos (INTRA e ITIQ) enviará mais do que 16 pacotes sem que nenhum pacote tenha sido enviado pelo módulo concorrente. No entanto, a definição dessa profundidade dependerá das necessidades de cada aplicação e esta informação também define o tamanho do rótulo utilizado.

Tabela 6.6: Resultados de síntese para *standard cells* de uma NI com o módulo sincronizador.

| | **Área (*gates*)** | **Frequência** | **Área (um²)** | **Potência @ 100MHz (mW)** |
|---|---|---|---|---|
| **NI + sincronizador** | 2122 | 127 MHZ | 50.019 | 1,95 |
| **NI + sincronizador + memórias** | 10748 | 126 MHZ | 169.817 | 6,54 |

Os resultados da tabela 6.6 foram obtidos a partir de uma primeira proposta em que, no processo de indexação dos sinais presentes na interface de rede, foi considerado o uso de multiplexadores para o envio e o recebimento de cada flit. Dessa forma, se comparado os resultados de área com os obtidos com a NI genérica anteriormente apresentada, percebe-se que estes resultados apresentam uma área menor, no entanto, a solução anterior é mais genérica.

Se as redes-em-chip forem comparadas ao protocolo ISO/OSI utilizados para as redes convencionais (PASRICHA, 2008), algumas das soluções apresentadas nesse trabalho poderiam estar em camadas mais elevadas. No entanto, para isso, a aplicação em NoC necessitaria ter a mesma estrutura hierárquica para permitir os mesmos níveis de abstração. Porém, dependendo da aplicação pode não ser justificado esse tipo de solução já que seria necessário se fazer o uso de processador para realizar os controles definidos em hardware pelas NIs propostas. Além disso, dependendo das exigências da aplicação, utilizar soluções em software não garantiria o mesmo desempenho obtido em hardware para o sistema (RADULESCU, 2004), (BHOJWANI, 2003). Uma implementação em hardware fornece uma latência muito menor do que comparada a uma implementação em software. Conforme demonstrado em (BHOJWANI, 2003), para uma solução de empacotamento em software são necessários 47 ciclos (considerando 1 instrução por ciclo).

## 6.3 Considerações

Neste capítulo foram apresentadas algumas soluções de interfaces de rede considerando-se diferentes situações que ocorrem quando se faz a interconexão dos módulos de uma aplicação aos roteadores de uma NoC. Vários cenários e necessidades de NIs foram comentados e por fim, foram apresentados alguns resultados de síntese para uma interface de rede genérica. A interface de rede proposta pode ser utilizada para a conexão de qualquer elemento de processamento, permitindo o recebimento e envio de dados de/para diferentes nodos da rede. Ainda nesse capítulo foram apresentadas duas soluções de sincronismo de pacotes em NoCs. Soluções estas que devem ser

tomadas quando um EP depende de dados vindos de diferentes nodos da rede para desenvolver a computação.

Os modelos de NIs apresentados neste capítulo foram obtidos observando-se o comportamento e necessidade das aplicações. Dentro destas análises, verificaram-se como os módulos se comunicam e constataram-se ainda a heterogeneidade entre eles com relação à taxa de processamento, largura da palavra de dados, comportamento de tráfego, dentre outras características.

Verificou-se, neste capítulo, o impacto das FIFOs adicionadas às NIs e constatou-se que a definição das profundidades das mesmas deve levar em consideração a necessidade de largura de banda do EP, de forma que a NI garanta a taxa de tráfego exigida pela aplicação. No entanto, a FIFO deve ser configurada para a profundidade mínima que atenda aos requisitos acima mencionados, porém sem que haja desperdício de energia.

Como validação das interfaces de rede projetadas, as NIs foram utilizadas na conexão dos módulos do decodificador de vídeo H.264 à NoC SoCIN e as etapas dessa implementação serão apresentadas no próximo capítulo. A partir dessa implementação, serão analisados e comentados diversos resultados para diferentes configurações da aplicação em NoC.

# 7 MAPEAMENTO DO DECODIFICADOR DE VÍDEO EM NOC

## 7.1 Interfaces de rede para os módulos do decodificador de vídeo

Neste capítulo será comentado como cada um dos módulos que compõem o decodificador de vídeo foi conectado à rede-em-chip SoCIN através das interfaces de rede projetadas. O decodificador de vídeo foi particionado em cinco módulos: *NAL*, *PARSER*, *INTRA*, *ITIQ* (transformadas inversas e quantização inversa) e *DESCRAMBLER*, conforme apresentado na figura 7.1 que ilustra as taxas de comunicação (mínima, média e máxima) dos módulos.

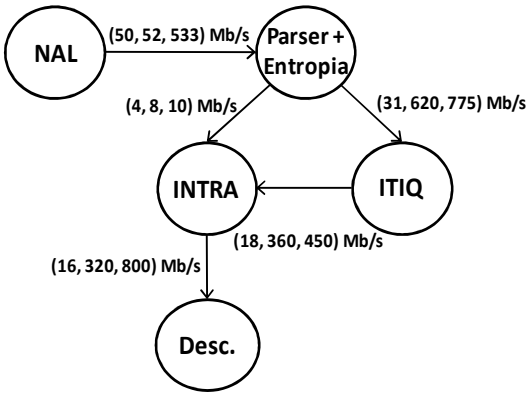



Figura 7.1: Taxas de comunicação dos módulos do decodificador de vídeo H.264 composto por cinco módulos.

A divisão dos módulos foi definida conforme a dependência de dados existente entre as tarefas. Dessa forma, alguns módulos foram agrupados em apenas um núcleo, como o módulo *ITIQ* que é composto das transformadas inversas e quantização inversa e o módulo composto por *parser* e decodificação de entropia. Estes módulos foram mapeados em cinco roteadores, conforme ilustrado na figura 7.2. A posição dos módulos na rede foi definida conforme os resultados de desempenho obtidos para três possibilidades de mapeamentos, que serão comentados na seção 7.2. Conforme se pode observar na figura 7.2, somente os canais necessários para esse mapeamento foram utilizados nessa implementação.

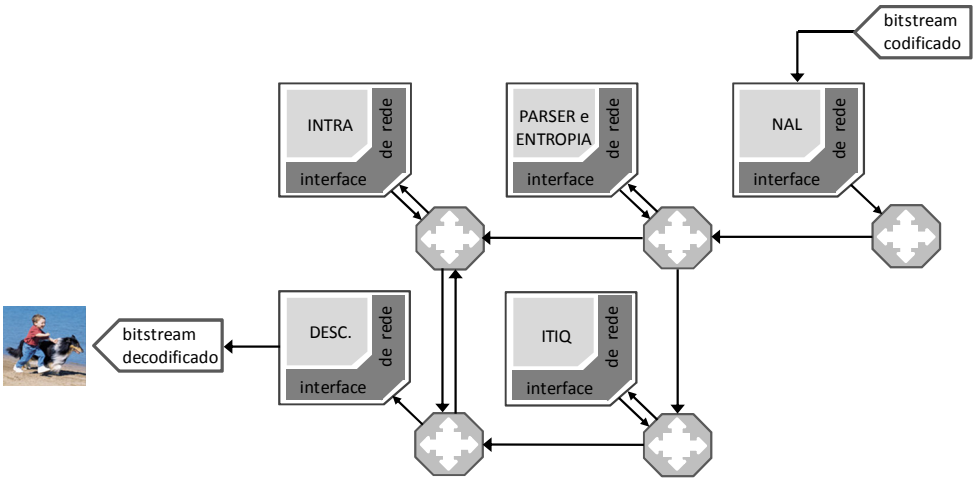



Figura 7.2: Mapeamento do decodificador de vídeo H.264 à rede SoCIN.

Para permitir uma conexão de forma mais direta com os módulos do decodificador de vídeo, algumas pequenas alterações foram incorporadas ao projeto das NIs com o objetivo de utilizar a implementação dos IPs sem realizar-se muitas alterações na sua descrição. Nas próximas subseções serão apresentados os detalhes da conexão destes módulos à rede SoCIN, utilizando as interfaces de rede desenvolvidas e apresentando as pequenas adaptações no modelo genérico de NI proposto para permitir o correto funcionamento desta aplicação.

### 7.1.1 Interface de rede para o módulo NAL

O módulo NAL, conforme já relatado na seção 3, é o primeiro módulo que compõe o decodificador de vídeo utilizado como estudo de caso nesse trabalho. Esse módulo tem por função receber o bitstream codificado e repassar para o *parser* as informações utilizadas no processo de codificação. Como este é o primeiro módulo da aplicação, ele apenas envia os pacotes para a rede, e não precisa receber nenhuma informação da mesma. Sendo assim, conforme apresentado na figura 7.3, verifica-se que este módulo necessita apenas de uma Unidade Empacotadora (*NI_PACK_NAL*, na figura 7.3) que será responsável por enviar os flits que constituem cada pacote para o *parser*. Nesta interface de rede, primeiramente a unidade NAL recebe o bitstream em pacotes, remove os bits que foram utilizados para formar as unidades NAL e armazena os dados em uma FIFO para posteriormente serem enviados pela NoC. A interface de rede da figura 7.3 apresenta os mesmos sinais de controle da interface genérica, ou seja, os dados do IP são enviados para uma FIFO e são lidos pela Unidade Empacotadora (*NI_PACK_NAL*) para a formação dos flits. Um sinal indicando que os dados estão prontos e podem ser escritos na FIFO é enviado pelo módulo *NAL_DELIMITER*. Como esse módulo apenas organiza os dados recebidos para serem armazenados na FIFO, o sinal indicando que a FIFO está cheia (*prog_full*) é repassado para um módulo externo que envia os bitstream para o decodificador sempre que há espaço na FIFO. Quando os dados estão prontos para serem escritos na FIFO, um sinal do módulo *NAL_DELIMITER* é ativado. O controle de leitura da FIFO e ativação do processo de empacotamento é realizado conforme mostrado na interface genérica anteriormente apresentada.

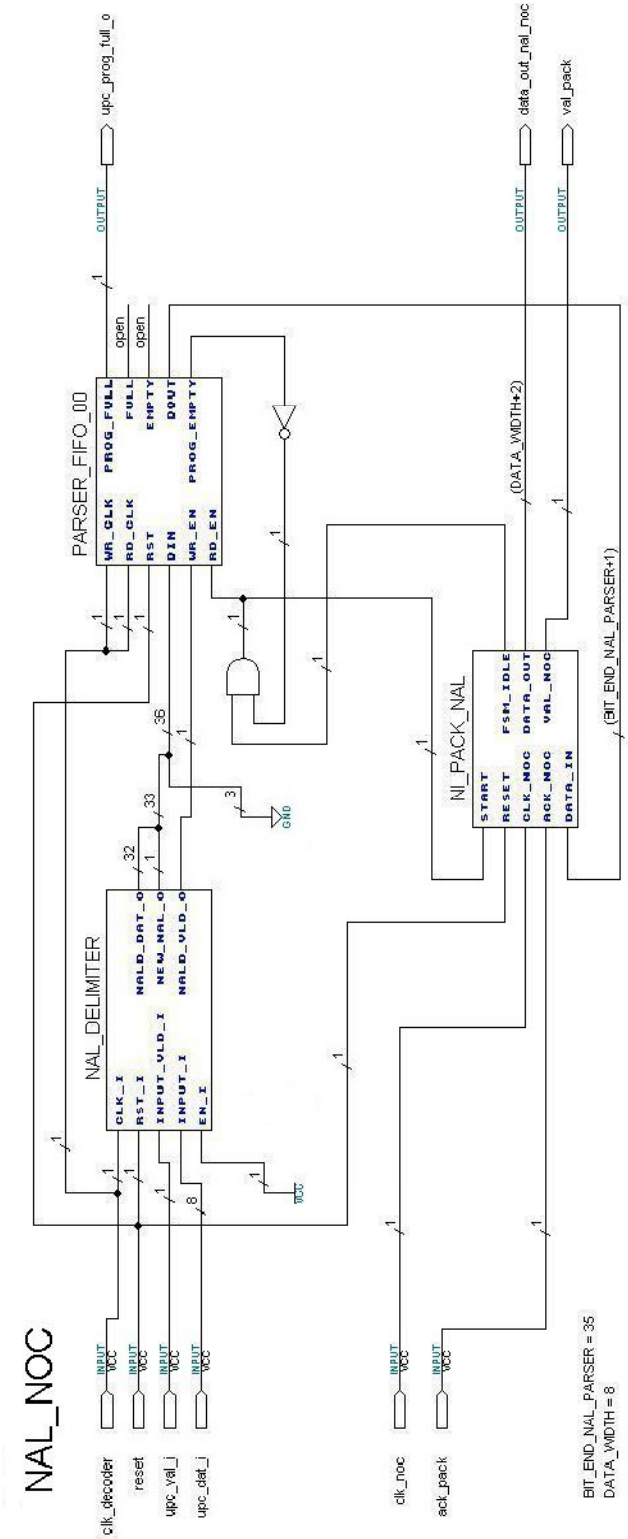



Figura 7.3: Interface de rede utilizada para a conexão do módulo NAL à rede.

### 7.1.2 Interface de rede para o módulo do Parser

Na sequência do processo de decodificação, os dados são enviados para o módulo *Parser* e Decodificação de Entropia (chamado apenas de *parser* nessa subseção). O *parser* tem a função de identificar e decodificar os elementos sintáticos e dividir os dados que devem ser enviados para o INTRA e para o ITIQ. Esta interface de rede é um exemplo de interface apresentada na seção 6.1.3, quando uma NI necessita enviar dados para mais de um destinatário. A figura 7.4 apresenta a NI utilizada para o módulo *parser*. Conforme os modelos de NI apresentados na seção 6, primeiramente os dados são desempacotados (modulo *unpack*) e repassados para o módulo (*parser_top).*

Como o *parser* envia pacotes para dois módulos, ITIQ e INTRA, e como estes dados são enviados considerando-se uma política de arbitragem, foi necessário utilizarmos uma FIFO para o armazenamento dos dados referentes a cada destinatário. A decisão por utilizarmos um dispositivo de armazenamento independente para os dados referentes a cada um destes módulos foi feita devido a vários fatores que serão comentados a seguir. Primeiramente, esta opção facilitou a conexão do *parser* à rede, já que as interfaces do *parser* já apresentavam um controle separadamente para dados a serem enviados para os módulos ITIQ e INTRA, sendo assim, não foi necessário alterar implementação deste módulo com relação a este controle. Outro motivo para esta decisão é devido ao fato de os dados do *parser* para o ITIQ e o INTRA não serem gerados a uma taxa constante, além disso, é possível que os dados para ambos os módulos sejam gerados simultaneamente, já que o *parser* apresenta saída de dados para os dois módulos independentemente. Sendo assim, caso a decisão fosse por utilizar-se uma única FIFO, seria necessário utilizar um esquema de arbitragem para definir quem pode armazenar os dados primeiramente na FIFO. Por estes motivos, decidiu-se utilizar duas FIFOs, cada uma responsável pelo armazenamento de dados referentes a cada um dos núcleos destinatários.

O envio dos dados gerados pelo *parser* para o INTRA e ITIQ são definidos conforme o esquema de prioridade apresentado na figura 7.5. Na lógica apresentada na figura 7.5, o envio de dados para um dos destinatários ocorre sempre que a Unidade Empacotadora não estiver enviando flits para a rede e que, pelo menos uma das FIFOs não esteja cheia. Primeiramente é verificada se a FIFO com dados para o INTRA (INTRA_FIFO) não está vazia, somente se essa FIFO não contiver dados, é que a FIFO que contém dados para o ITIQ é verificada. Essa definição foi necessária porque para um determinado mapeamento dos módulos à rede, em que um canal da NoC era compartilhado entre dois nodos, a aplicação travou quando não se priorizou o envio da dos dados ao INTRA. Detalhes sobre estes experimentos serão comentados na subseção 7.2.1. Essa necessidade de priorização para envio dos dados pode ocorrer quando a escrita dos dados em cada uma das FIFOs ocorre em taxas diferentes e/ou existe dependência no recebimento destes dados no destinatário.

O *parser* já foi implementado para a conexão com FIFOs, sendo assim, o controle de escrita nas FIFOs, considerando-se a situação de FIFOs cheias, é realizado internamente pelo IP da mesma forma como apresentado no modelo genérico de NI. Uma única unidade de Empacotamento é utilizada para envio dos dados para o módulo INTRA e para o módulo ITIQ. Conforme a definição do destinatário para qual os dados serão enviados, o sinal de leitura da FIFO respectiva é acionado. O sinal que habilita o envio de um pacote para a rede será igual a ‘1’ sempre que pelo menos uma das FIFOs não

estiver vazia e a Unidade Empacotadora estiver em estado ocioso, definições estas conforme comentadas quando as interfaces de redes genéricas foram apresentadas.

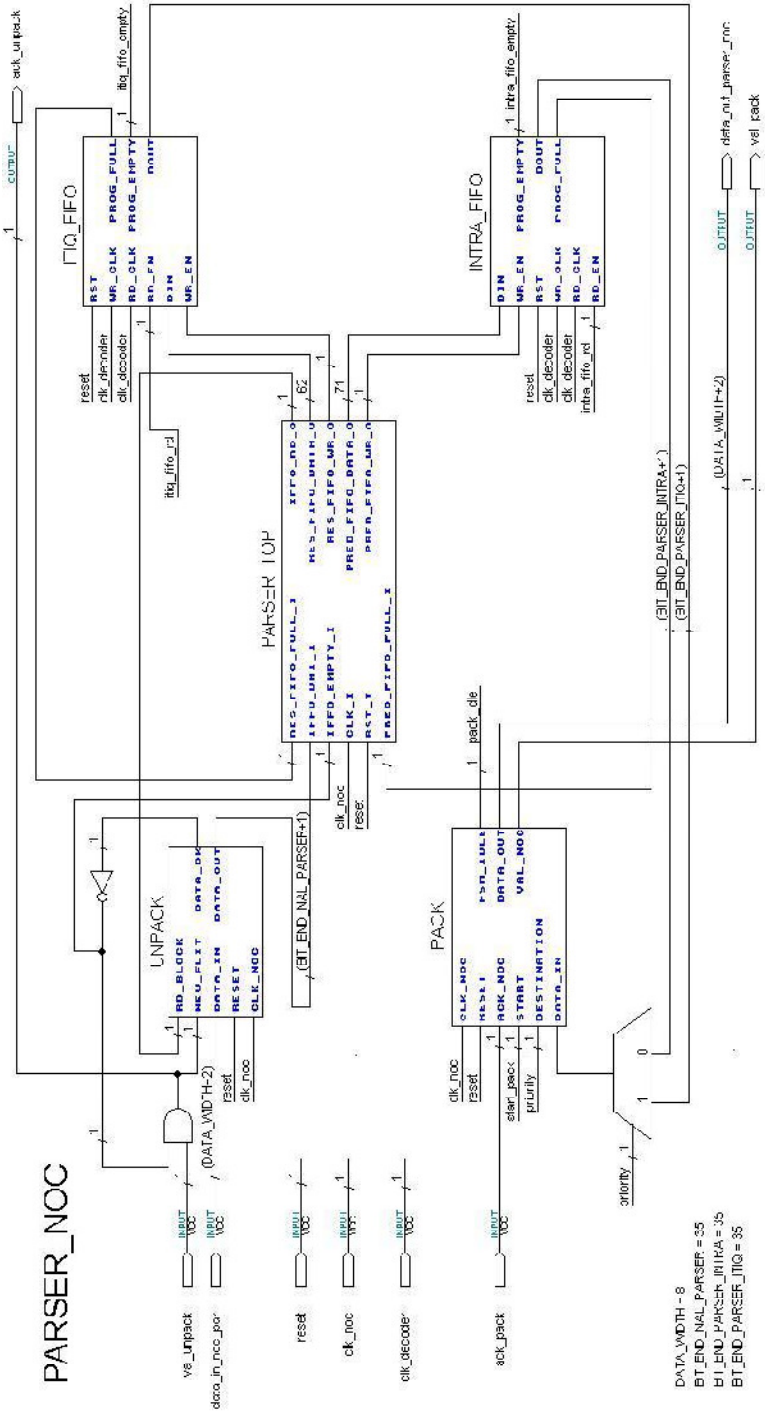



Figura 7.4: Interface de rede utilizada para a conexão do módulo *parser* à rede.

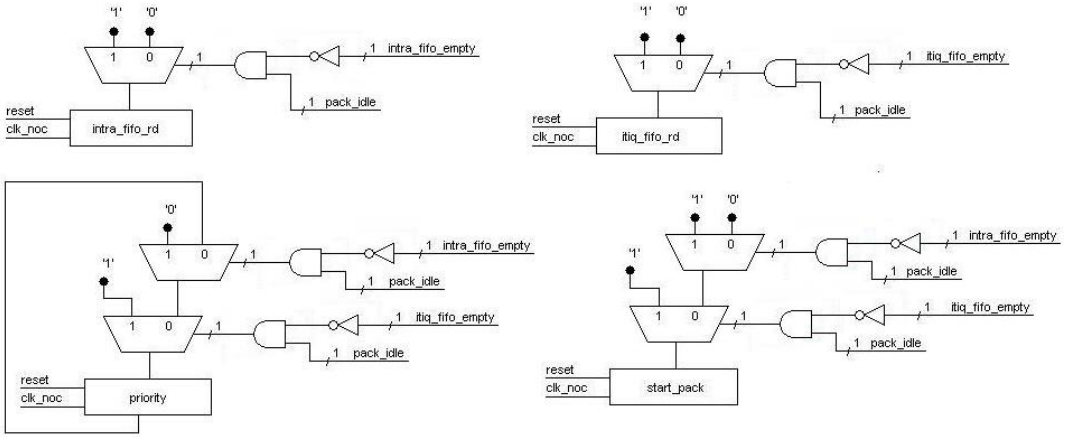



Figura 7.5: Lógica de arbitragem utilizada para a definição do módulo para qual a NI enviará o pacote.

### 7.1.3 Interface de rede para o módulo das Transformadas e Quantização Inversas

O *parser* repassa os resíduos codificados para o módulo das transformadas e quantização inversas (ITIQ). Conforme comentado no capítulo 4, primeiramente é realizada a operação de quantização inversa que se trata de operações do tipo multiplicação, soma e deslocamento das amostras por constantes. Logo após são realizadas um conjunto de transformadas inversas, como Hadamard e IDCT-2D. Na NI utilizada para a conexão do módulo ITIQ, os dados são recebidos pela Unidade Desempacotadora, repassados para o módulo ITIQ e posteriormente são enviados para uma FIFO. A Unidade Empacotadora lê os dados da FIFO e os prepara para envio do pacote em flits pela rede. Todo esse processo foi implementado nesta NI sem nenhuma alteração do modelo genérico de NI apresentado anteriormente. A figura 7.6 apresenta a conexão dos sinais dos módulos utilizados nesta interface de rede. A única diferença apresentada nessa figura com relação ao modelo genérico de NI é que o módulo ITIQ já possuía uma FIFO interna na sua saída. No entanto, como este módulo já apresentava uma interface de saída a partir de uma FIFO, considerou-se esta mesma lógica para a conexão com os demais módulos que compõem a NI. O mesmo esquema de leitura dos dados da FIFO em função da disponibilidade da Unidade Empacotadora e da existência de dados na FIFO é realizado nesta NI.

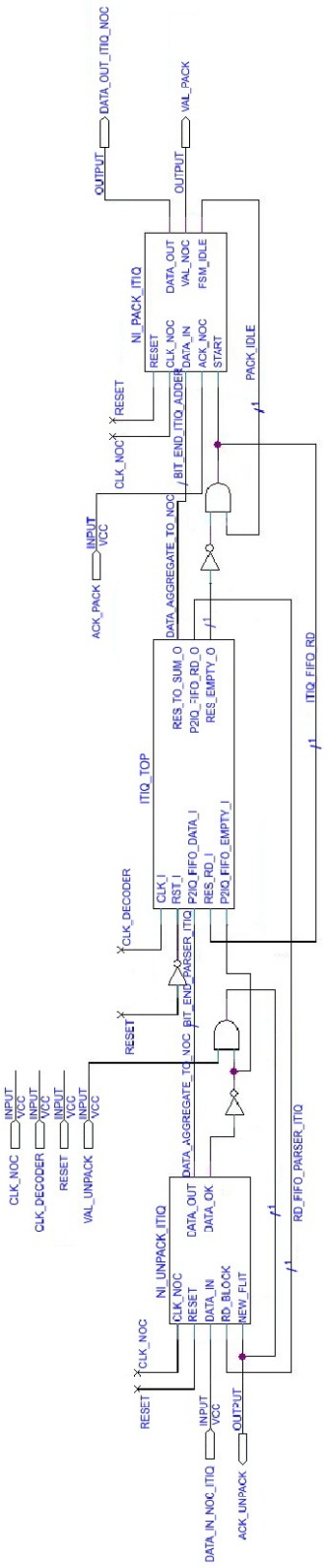



Figura 7.6: Interface de rede utilizada para a conexão do módulo ITIQ à rede.

### 7.1.4 Interface de rede para o módulo de Predição INTRA

A NI do módulo INTRA recebe os dados do *parser* com informações quanto à predição. Após o desempacotamento dos dados enviados pelo *parser*, o módulo INTRA gera as amostras preditas baseado nas amostras da vizinhança. A vizinhança é repassada para o módulo INTRA através da soma das amostras anteriormente preditas aos resíduos recebidos pelo bloco ITIQ. Como este módulo já apresentava um esquema de sincronização dos dados a partir de FIFOs, não se utilizou a proposta de sincronização por rótulos, pois seria necessário fazer diversas alterações na implementação atual do INTRA, já que o mesmo apresenta uma interface muito dependente da conexão originalmente implementada. Por estes motivos, optou-se pela segunda alternativa de sincronização que é baseado por FIFOs. Neste caso, a geração das amostras preditas pelo INTRA é dependente de dados disponíveis nas duas FIFOs presentes após a Unidade Desempacotadora.

A figura 7.7 apresenta a NI utilizada para a conexão do módulo INTRA à rede. Esta interface de rede foi implementada utilizando-se o mesmo esquema apresentado na subseção 6.1.2, em que a NI recebe pacotes de múltiplos núcleos. Logo após a obtenção das amostras preditas elas devem ser somadas aos resíduos para servirem de vizinhança no processo de predição e enviadas para o módulo *descrambler*. Para o funcionamento do módulo INTRA, necessita-se sincronizar os resíduos recebidos às amostras preditas, para isso, foram utilizadas duas FIFOs logo após o recebimento do pacote.

Cada pacote recebido pela NI apresenta a informação do transmissor. Sabendo quem enviou o pacote, após o recebimento de todos os flits e reconstituição do dado original, o mesmo é enviado para a sua FIFO respectiva. Quando a Unidade Desempacotadora tiver um dado pronto para ser armazenado na FIFO, é verificado se a FIFO respectiva ao IP transmissor não está cheia. Quando houver espaço disponível na FIFO, o dado é lido da Unidade Desempacotadora, liberando-a para o recebimento de outros flits. As XORs apresentadas na figura 7.7 realizam a comparação do código recebido (*source*) com os códigos definidos para cada um dos módulos transmissores de pacotes para esta NI.

Depois de realizada a predição pelo módulo INTRA, as amostras, juntamente com os resíduos, vão para o somador (*sum_sel*) e o resultado da soma é enviado para o módulo INTRA e para uma FIFO. Posteriormente estes dados são lidos da FIFO pela Unidade Empacotadora para serem transmitidos ao *descrambler* via rede. Realizou-se alguns testes fazendo com que esta soma fosse realizada em outro núcleo, porém, esta solução prejudicaria muito o desempenho da aplicação, sendo assim, decidiu-se realizar a soma logo após o módulo INTRA em um mesmo núcleo, já que este módulo é totalmente dependente destes resultados.

O processo de empacotamento dos dados a serem enviados pela rede ocorre da mesma forma que nas demais NIs. A FIFO que armazena os resultados após a etapa de soma das amostras é indispensável nessa NI já que o somador é totalmente combinacional. Dessa forma, mesmo que a largura do canal fosse igual à largura dos dados após a soma, como ainda é enviado o cabeçalho e como o INTRA envia dados em rajadas e dessa forma, apresenta uma taxa de processamento que não é constante, os dados após a soma precisam ser armazenados para que se garanta a taxa de processamento da aplicação.

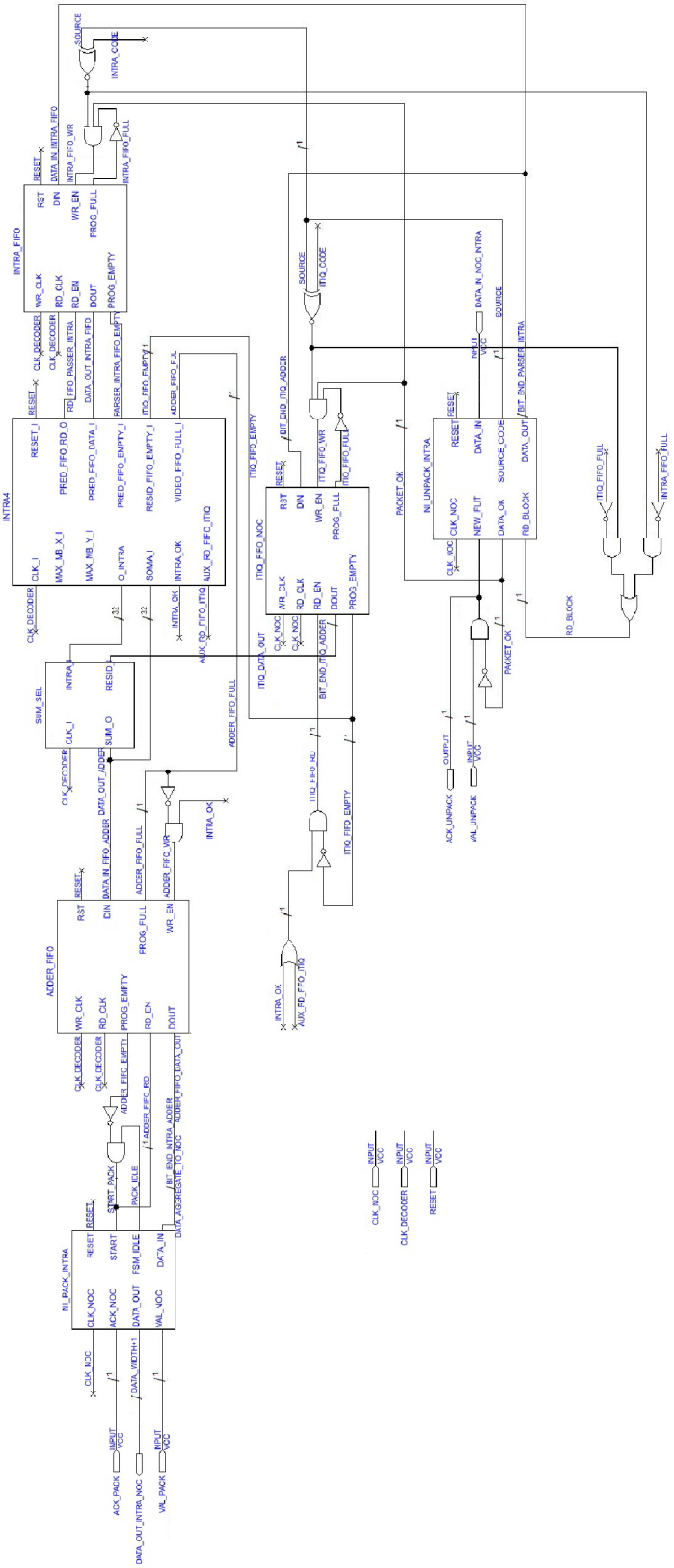


Figura 7.7: Interface de rede utilizada para a conexão do módulo INTRA à rede.

### 7.1.5 Interface de rede para o módulo do Descrambler

Como este é o último módulo da aplicação, a única função da NI é receber os pacotes após o processo de decodificação das amostras e repassá-las para o *descrambler*. Conforme já comentado, o *descrambler* trata- de uma memória com possibilidade de armazenamento de dois linhas de macroblocos. A cada linha de macroblocos armazenada na memória é possível reordenar as amostras e reenviá-las para a saída permitindo que sejam exibidas como linhas de pixel da imagem. Esta reordenação é necessária, pois no processo de codificação, os macroblocos são ordenados em duplo Z e toda a decodificação é feita nesta mesma ordenação. Sendo assim, após a soma dos resíduos às amostras preditas, é necessário organizar os macroblocos em linhas de pixels. Por exemplo, tratando-se de uma resolução QCIF, o número de amostras é 176x144, dessa forma, o vídeo é exibido linha a linha, e cada linha é composta de 176 amostras. Após o processo de *descrambler*, os dados são enviados para um conversor de vídeo que sincroniza o sinal para a exibição em um monitor ou sistema de TV no formato VGA.

A figura 7.8 apresenta a NI utilizada para a conexão do *descrambler* à rede. O *inverter* apresentado nesta figura faz parte do processo de reordenamento, pois quatro amostras são processadas por vez e a ordem das amostras são organizadas antes de repassadas para o *descrambler*. No entanto, esta é uma função do IP e não da NI. Como o *descrambler* apresenta uma memória na entrada, o sinal de escrita na memória é ativo quando a mensagem original estiver reconstituída e quando a memória do *descrambler* não estiver cheia.

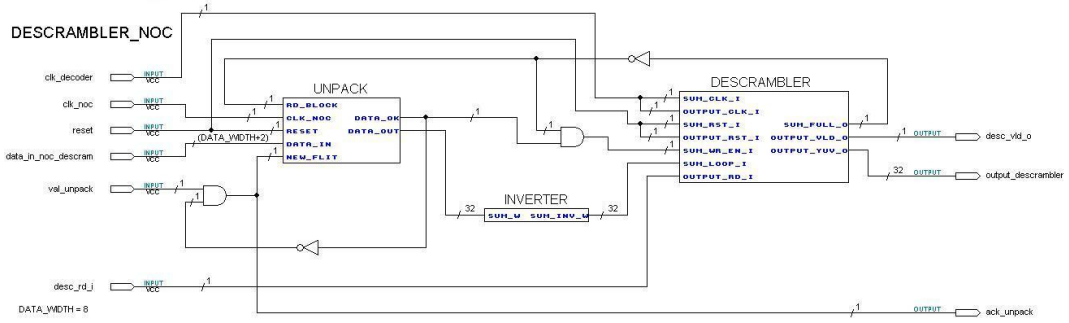



Figura 7.8: Interface de rede utilizada para a conexão do módulo *descrambler* à rede.

## 7.2 Resultados

A implementação deste trabalho foi realizada em linguagem VHDL e as simulações foram feitas utilizando-se a ferramenta ModelSim. Todos os códigos da descrição do decodificador de vídeo, bem como a descrição da rede-em-chip SoCIN também foram utilizadas versões implementadas em VHDL. Para a integração do decodificador de vídeo à rede SoCIN, a implementação de diversos outros trabalhos foram utilizadas, dentre estes trabalhos estão: (PEREIRA, 2009) que foi responsável pela implementação do *parser* e pela integração do decodificador utilizando conexão direta entre os módulos, (SILVA, 2009) que implementou o CAVLD, (STAEHLER, 2006) que desenvolveu o módulo de predição INTRA, (AGOSTINI, 2007) que implementou o módulo das transformadas e quantização inversa, (ZEFERINO, 2003) que implementou a rede SoCIN, dentre várias outras pessoas que colaboraram no desenvolvimento dos

módulos utilizados neste trabalho. A seguir serão apresentados os resultados obtidos para implementação do decodificador de vídeo em NoC, como latência, vazão, área e frequência para diversas possibilidades de configuração do sistema.

Para os experimentos reportados a seguir, utilizou-se resolução QCIF e Qp igual a 22. Este valor de Qp trata-se de um valor médio, já que os valores de Qp variam de 0 a 51 e é responsável por definir a qualidade do vídeo.

O arquivo de entrada contendo *bitstreams* codificados de uma sequência de vídeo conhecida como *Foreman* foi utilizado na verificação desse trabalho. Esse arquivo foi obtido a partir de uma implementação do decodificador de vídeo em C++. Para a verificação do correto funcionamento da aplicação em NoC, realizaram-se comparações com a implementação original a partir de escrita em arquivos. O arquivo de saída contendo os *bitstreams* corretamente decodificados para a comparação destes resultados também foi gerado a partir da implementação em C++ do decodificador de vídeo. Todas as possibilidades de configuração utilizadas nesse trabalho foram verificadas a partir da estratégia de verificação de escrita em arquivos acima comentada. Na verificação do funcionamento da aplicação para diferentes configurações da implementação, simplesmente redefiniu-se alguns parâmetros do sistema a partir de uma tabela (*package*). Estes parâmetros podem facilmente serem configurados para análise do comportamento da aplicação a fim de obter as características desejadas para a mesma. A verificação do funcionamento da aplicação para todas as configurações que serão comentadas nos resultados apresentados nas próximas subseções comprovam a flexibilidade desta implementação e a facilidade de configuração da mesma graças às NIs projetadas e a NoC utilizada neste trabalho.

Conforme já comentado, o *descrambler* armazena duas linhas de macroblocos em uma memória, e reordena as amostras para serem exibidas em linhas de pixels. Cada linha de macroblocos possui 16 linhas de pixels da imagem. Um controlador de vídeo manda um sinal de leitura das amostras para o *descrambler* para essa exibição. A memória utilizada no *descrambler* trata-se, na verdade, de um buffer com endereçamento, já que a mesma apresenta controle de cheia e vazia. Este controle é realizado conforme as amostras são organizadas e entregues ao controlador de vídeo. Sendo assim, se todas as amostras presentes na memória já tiverem sido entregues, é informado que a mesma está vazia. Quando todas as posições da memória apresentarem dados que ainda não foram lidos pelo controlador de vídeo, a memória tem a indicação de cheia.

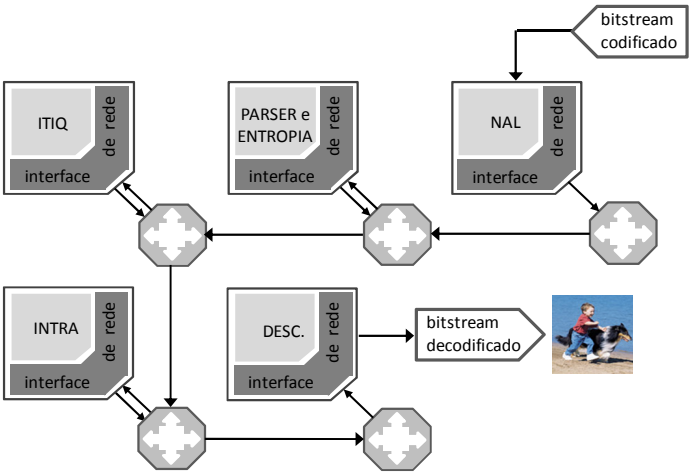
Observando-se o comportamento da implementação, o decodificador implementado sem NoC apresenta um intervalo de tempo ocioso quando as amostras estão sendo exibidas na tela, já que a memória do *descrambler* estará cheia nessas situações, e dessa forma, os demais blocos do decodificador necessitam aguardar até que a mesma tenha espaço disponível. Sendo assim, em muitas situações, a implementação com NoC não apresentou perdas com relação aos resultados de latência e vazão da aplicação considerando-se a taxa com que as amostras são lidas. Isto se deve ao fato de que o tempo de leitura das amostras de saída para exibição do vídeo é maior do que o tempo de decodificação. Essa situação será detalhada para cada caso de configuração analisada para esse sistema. Por esse motivo, resolveu-se fazer três análises com relação à latência destes resultados:

- analisou-se a latência na saída do *descrambler,* considerando-se o tempo de leitura das amostras pelo controlador de vídeo;

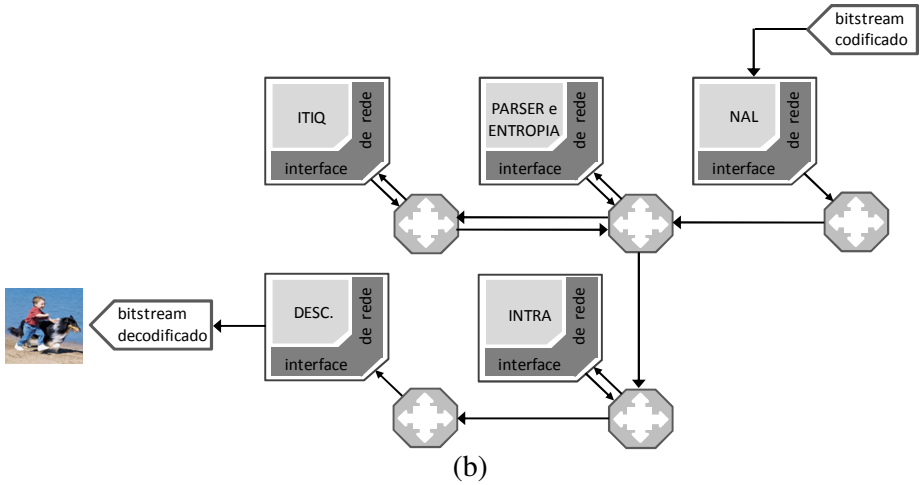
- analisou-se a latência na entrada do *descrambler*, considerando-se as disponibilidade do *descrambler* de recebimento das amostras conforme o tempo de leitura das amostras pelo controlador de vídeo;
- analisou-se a latência para a taxa máxima de envio das amostras para a saída, ou seja, não considerando o tempo de leitura das amostras pelo controlador de vídeo.

### 7.2.1 Mapeamento

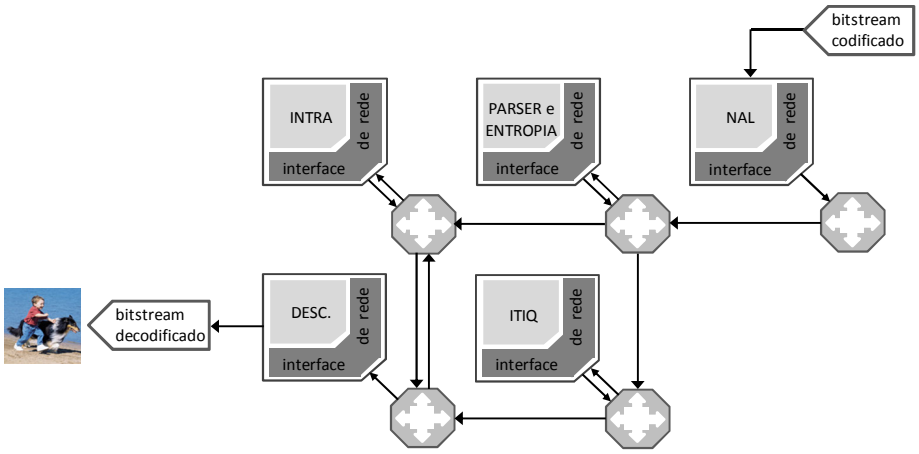
A primeira análise realizada foi feita considerando-se três possibilidades de mapeamento do decodificador em NoC. Todas as três possibilidades de mapeamento utilizam somente cinco roteadores e apresentam necessidade mínima de roteamento dos pacotes. Dessa forma, somente através dos resultados de latência é que foi possível verificar a melhor solução de mapeamento para a aplicação. A figura 7.9 apresenta as três possibilidades analisadas.




(a)




(b)




(c)

Figura 7.9: Possibilidades de mapeamento do decodificador de vídeo em NoC: mapeamento A (a), mapeamento B (b) e mapeamento C (c).

Como se pode observar no mapeamento A, o roteamento utilizado para envio dos pacotes do *parser* para o INTRA compartilha um mesmo *link* com o roteamento utilizado para envio de pacotes do módulo ITIQ para o INTRA. Neste tipo de comportamento, deve-se observar alguns detalhes imprescindíveis para o correto funcionamento da aplicação. A leitura dos dados enviados pelo *parser* para o INTRA na NI do módulo INTRA depende da existência de dados armazenados tanto na FIFO do ITIQ quanto na FIFO que armazena os dados recebidos do *parser*. No entanto, a taxa de geração das amostras de resíduos é muito maior do que o envio de pacotes do *parser* para o INTRA. Sendo assim, é necessário certificar-se de que não ocorrerá travamento da aplicação, e isso é possível quando se garante uma sequência apropriada do envio de pacotes pelas NIs. Analisando-se a implementação deste mapeamento, a aplicação trava quando não se considera uma prioridade no envio dos dados da NI do *parser* para os demais módulos. Nessa situação, como parte do caminho da NoC é utilizada para envio de pacotes distintos (*parser* - ITIQ e *parser* - INTRA), é possível que, por exemplo, a FIFO para os dados do ITIQ dentro da NI do INTRA esteja cheia e que um novo pacote do ITIQ para o INTRA esteja ocupando os buffers da NoC. Nesse caso, se a FIFO que armazena dados do *parser* para o INTRA estiver vazia, o INTRA não processa novas amostras, e sendo assim, os dados de resíduos armazenados na FIFO do ITIQ na NI do INTRA não são lidos. Consequentemente estes dados não sendo lidos, a FIFO do ITIQ não poderá armazenar novos pacotes, não liberando os buffers dos canais da NoC para o armazenamento de novos flits referentes a pacotes do *parser* para o INTRA. A figura 7.10 apresenta uma ilustração deste problema para ajudar na compreensão do mesmo.

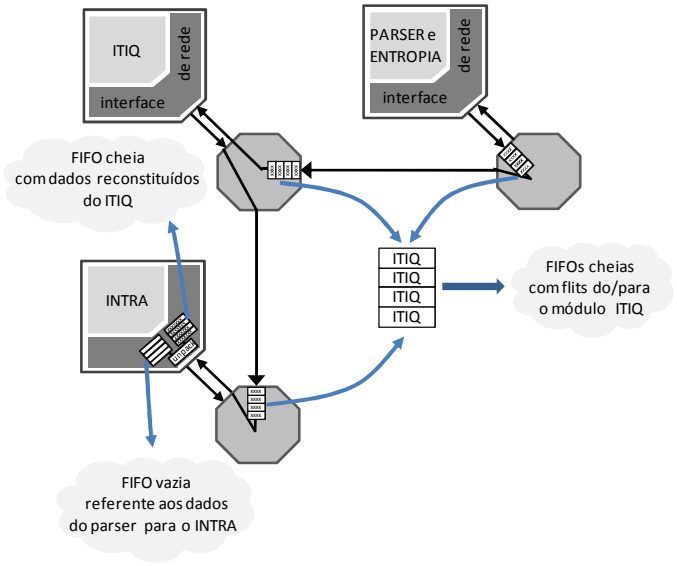



Figura 7.10: Exemplo de situação de travamento da aplicação devido à dependência de dados para o funcionamento do módulo INTRA.

Para a solução deste problema é possível aumentar a profundidade das FIFOs de modo que se garanta que essa situação não ocorra, ou ainda, utilizar um esquema de prioridade no envio dos pacotes na NI do *parser*. Como as FIFOs são os principais consumidores de área e potência das NIs, decidiu-se adotar uma solução de arbitragem no envio dos pacotes e, com isso, não se restringe o uso de FIFOs maiores na aplicação. A solução de prioridade, conforme comentado na subseção 6.1.3, foi priorizar o envio dos pacotes do *parser* para o INTRA, já que esses dados são produzidos a uma taxa muito menor que os dados para o ITIQ. Nas demais possibilidades de mapeamento apresentadas nesse trabalho, esse problema não ocorreu, pois os pacotes do *parser* para o módulo INTRA e do *parser* para o módulo ITIQ não utilizam o mesmo roteamento.

As soluções de mapeamento A e B apresentam *links* que são utilizados por mais de um IP. Já na solução C, considerando-se que a NoC utiliza o roteamento X-Y, todos os *links* só são utilizados para envio de pacotes de um mesmo transmissor para um mesmo receptor. A figura 7.11 apresenta os resultados de latência obtidos para as possibilidades de mapeamento A, B e C. Como a solução C não divide seus links, este foi o mapeamento que apresentou resultados de latência menor. Esses resultados foram obtidos para largura do canal igual a 8, profundidade das FIFOs da NoC igual a 8 e profundidades das FIFOs das NIs igual a 8. Os resultados de latência referem-se ao número de ciclos necessários para a obtenção de 200.000 amostras no *descrambler*, considerando-se o comportamento normal da aplicação, ou seja, respeitando-se o tempo de envio das amostras para a saída do *descrambler,* conforme o sinal de leitura enviado pelo controlador de vídeo. Para essa situação, as diferenças nos resultados de latência são insignificantes, no entanto se tornam muito mais relevantes se considerarmos a taxa máxima de processamento da aplicação. De acordo com os resultados obtidos na figura 7.11, decidiu-se utilizar o mapeamento C para a análise dos demais parâmetros do projeto, já que este foi o mapeamento que apresentou melhores resultados de desempenho, embora seja também o que apresente maior número de canais totais utilizados nos roteadores. Como para a obtenção dos demais resultados, muitos valores

de síntese e desempenho precisaram ser medidos, levando um tempo elevado para a geração dos resultados de simulação, foi necessário optar-se por apenas uma das alternativas de mapeamento da figura 7.9.

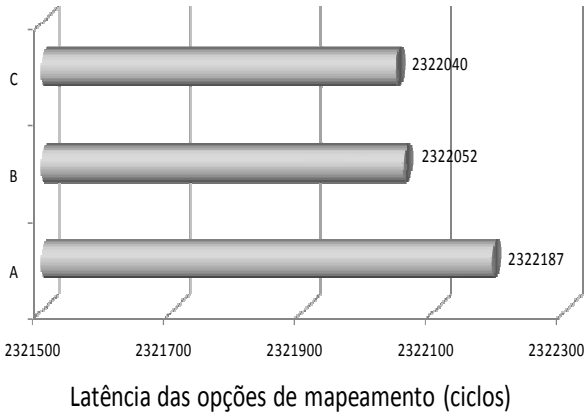



Figura 7.11: Resultados de latência, em ciclos, para 3 possibilidades de mapeamento do decodificador de vídeo em NoC.

### 7.2.2 Profundidade das FIFOs das NIs

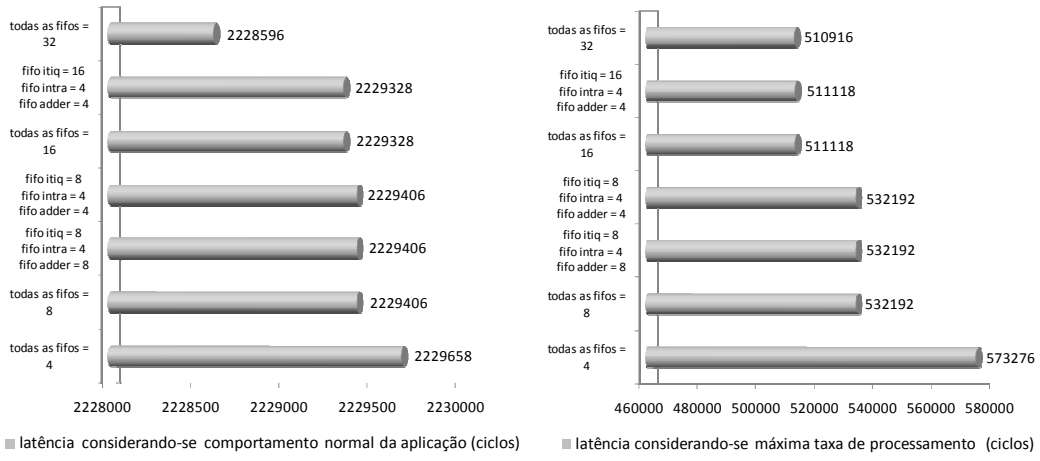
A próxima análise realizada foi com relação aos resultados de latência para diferentes profundidades das FIFOs utilizadas nas interfaces de redes. As FIFOs presentes nas NIs que mais impactam nos resultados são as utilizadas nas interfaces dos módulos *parser* e INTRA, pois são estas FIFOs que armazenam dados que apresentam dependência para o processamento do módulo INTRA. As FIFOs que foram analisadas são as duas FIFOs presentes no módulo *parser*, que enviam dados para o módulo INTRA e para o módulo ITIQ e que estão indicadas nos gráficos como *fifo_intra* e *fifo_itiq*, respectivamente. Também foi analisado o impacto dos resultados das FIFOs presentes no módulo INTRA que recebem dados do módulo *parser* e do módulo ITIQ e indicou-se da mesma forma, como *fifo_intra* (armazena dados do *parser* para o INTRA) e *fifo_itiq* (armazena dados recebidos do módulo ITIQ). Decidiu-se indicar com o mesmo nome as FIFOs que apresentam o mesmo impacto no desempenho da aplicação e que apresentam dados referentes á predição ou aos resíduos para facilitar a compreensão dos resultados. Um outra FIFO presente na NI do INTRA também foi avaliada que é a FIFO que armazena os dados após a soma dos resultados de resíduos e predição e que posteriormente são enviados para o *descrambler*. Esta FIFO está indicada nos gráficos como *fifo_adder.*

A figura 7.12 apresenta os resultados de latência obtidos para diferentes profundidades das FIFOS anteriormente comentadas. Os resultados apresentados na figura 7.12 foram obtidos para largura do canal da NoC igual a 16 e profundidades da FIFOs da NoC igual a 4. Como na solução de mapeamento definida para os demais experimentos nenhum *link* é compartilhado entre os IPs, os *links* da NoC são utilizados para envio direto de dados de um determinado transmissor para um determinado receptor. Por esse motivo, definiu-se uma profundidade pequena para as FIFOs da NoC. Além disso, antes de avaliar os demais resultados de latência, constatou-se que como a largura da palavra de dados dos módulos desta aplicação vão de 32 bits a 71 bits, a largura de bits dos canais da NoC apropriada para a aplicação poderia ser definida como

16 ou 32 bits. Sendo assim, para estas possibilidades de largura de canal, tem-se mais um fator que não justifica definir FIFOs muito profundas, pois os módulos necessitarão enviar apenas alguns poucos flits. No entanto, uma análise aprofundada destes parâmetros será realizada nas próximas subseções.

A figura 7.12(a) apresenta os resultados de latência na entrada do *descrambler* considerando-se o funcionamento normal da aplicação, ou seja, considerando-se o tempo em que o vídeo é exibido e que o decodificador necessita aguardar o envio de novas amostras para o *descrambler*. Conforme se pode observar no gráfico da figura 7.12(a), a *fifo_itiq* é a FIFO que mais influencia nos resultados de latência da aplicação, isto se deve ao fato da taxa de comunicação para envio destes dados ser maior que a taxa de envio de dados pelos demais módulos. Sendo assim, utilizando-se uma FIFO de profundidade pequena para a *fifo_intra* e *fifo_adder* não se obtém alteração nos resultados de latência obtidos para a aplicação. Pode-se observar essa situação, quando se utiliza profundidade da *fifo_itiq* igual a 16 para todas as FIFOs e quando se utiliza profundidade da FIFO igual a 16 apenas para a *fifo_itiq* e FIFOs com profundidades menores para os demais casos. Nessa situação, bem como quando se utiliza profundidade da *fifo_itiq* igual a 8, os resultados de latência ocorrem apenas em função da *fifo_itiq*. Sendo assim, conclui-se que as demais FIFOs podem ser definidas com uma profundidade pequena que a aplicação não terá seu desempenho prejudicado.

A figura 7.12(b) apresenta os resultados de latência para a máxima taxa de processamento da aplicação. Percebe-se, pela figura 7.12(b), que os resultados de latência para essa situação também ocorrem conforme a profundidade da *fifo_itiq*.




(a) (b)

Figura 7.12: Resultados de latência, em ciclos, variando-se a profundidade das FIFOs presentes nas NIs que conectam os módulos do decodificador de vídeo à rede. Em (a) os resultados de latência considerando-se a taxa normal de envio dos dados da aplicação; em (b), os resultados de latência para a máxima taxa de processamento obtida para cada uma das configurações de FIFO.

Como se pode observar, os resultados de latência ilustrados na figura 7.12 não apresentam muita diferença para as diferentes profundidades de FIFOs analisadas. Verifica-se apenas que quando a *fifo_itiq* é definida com profundidade igual a 32 se obtém uma pequena redução nos resultados de latência da aplicação e que a latência

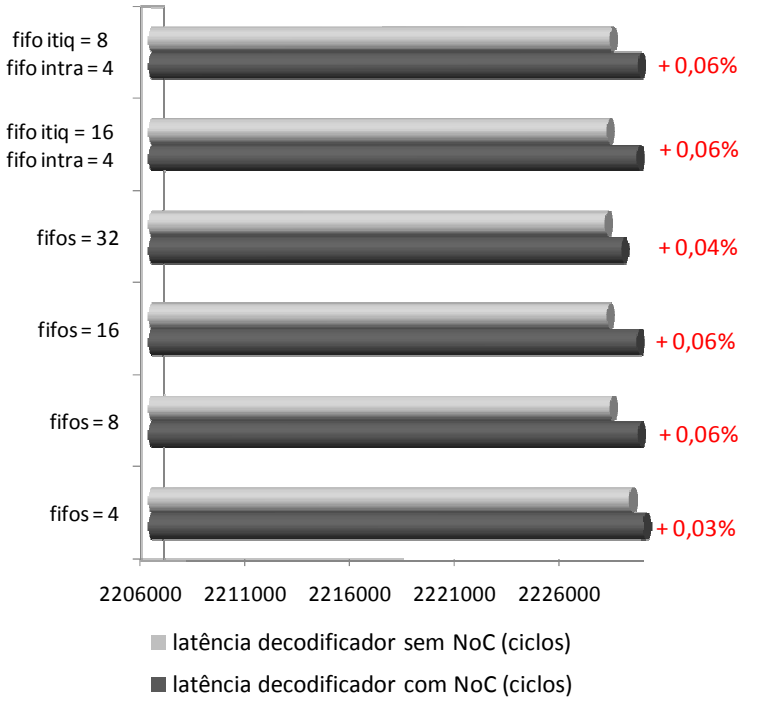
apresenta um aumento maior quando a *fifo_itiq* é igual a 4 (aproximadamente 10% de redução considerando-se profundidade da FIFO igual a 32 ao invés de 4).

Para os resultados da figura 7.12 considerou-se o número total de ciclos necessários para a obtenção de 200.000 amostras. Preferiu-se mostrar os resultados para um número considerável de amostras, pois considerando a latência média em ciclos para obtenção de uma amostra, as diferenças nos resultados são muito pequenas para as alterações de configurações anteriormente realizadas. Com isso, conclui-se que, especificamente para essa aplicação, é possível definir profundidades menores para as FIFOs presente nas NIs, já que ainda assim, garante-se o funcionamento da aplicação no tempo necessário.

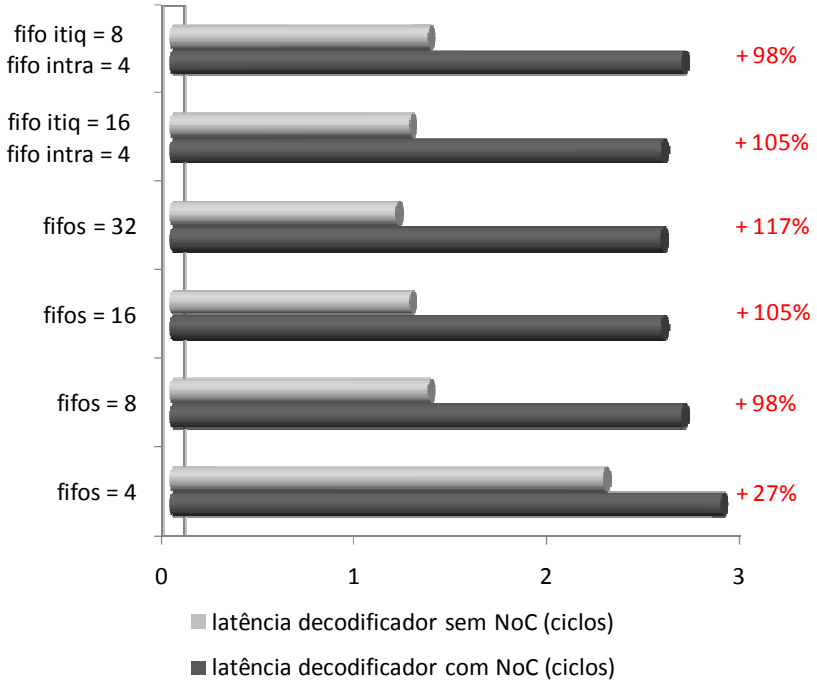
A figura 7.13 apresenta os resultados comparativos da implementação do decodificador de vídeo com NoC e sem NoC. Como a implementação original do decodificador de vídeo apresenta algumas FIFOs na implementação, pode-se fazer uma comparação destes resultados, definindo-se as mesmas profundidades de FIFOs para as duas aplicações. Os valores acrescentados ao lado de cada linha no gráfico representam a porcentagem de aumento de latência para cada configuração. Como se pode observar na figura 7.13 (a), a perda de desempenho da aplicação com NoC é praticamente nula, comparada à implementação original que utiliza apenas conexão direta entre os módulos. No entanto, isso ocorre porque, nesse caso, considerou-se o tempo de leitura das amostras pelo controlador de VGA, sendo que estes resultados de latência foram obtidos para as amostras recebidas na entrada do *descrambler* para as duas implementações. Nessa situação, para a escrita dos dados na memória do *descrambler*, é necessário aguardar que as amostras armazenadas na mesma já tenham sido enviadas para a saída.

Deve-se lembrar, porém, que este desempenho está sendo medido na entrada do *descrambler*, já que na saída do *descrambler* os resultados de desempenho são os mesmos para a implementação com ou sem NoC, pois as amostras são enviadas para a saída considerando-se o tempo de solicitação das mesmas pelo controlador de vídeo externo. Dessa forma, o que se precisa garantir é que quando as amostras forem solicitadas pelo controlador de vídeo, elas estejam disponíveis para serem exibidas. Caso as amostras ainda não estejam armazenadas na memória quando forem solicitadas, tem-se um funcionamento incorreto da aplicação, ou seja, ocorrerá falha na exibição das imagens do vídeo. Dessa forma, quando para uma determinada configuração da aplicação com NoC não se conseguir garantir esse comportamento na implementação, essa configuração não poderá ser utilizada. Ao final desta seção, estas situações serão comentadas e analisadas.

Para os resultados da figura 7.13(a) considerou-se o número de ciclos necessários para a obtenção de 200.000 amostras, já na figura 7.13(b) considerou-se a latência em número de ciclos para a obtenção de uma amostra. Como na figura 7.13(b) as diferenças nos resultados de latência são maiores, preferiu-se apresentar o número de ciclos para a obtenção de uma amostra. Para as configurações apresentadas na figura 7.13(b), obtém-se, na média, para a melhor configuração apresentada, uma amostra a cada **2,6** ciclos para o decodificador de vídeo com NoC e uma amostra a cada **1,2** ciclos para a implementação sem NoC.




(a)




(b)

Figura 7.13: Comparação dos resultados de latência para a aplicação com e sem NoC, variando-se a profundidade das FIFOs presentes nas NIs para o processamento da aplicação em tempo normal (a) e para o processamento máximo da aplicação (b).

A figura 7.13(b) apresenta a comparação dos resultados de latência considerando-se a máxima taxa de envio das amostras pelos módulos do decodificador de vídeo com e sem NoC. Como se pode constatar, nessa situação, a diferença nos resultados de latência é muito grande. Porém, estes resultados só foram apresentados para ilustrar a diferença de latência de uma aplicação operando na sua máxima taxa de processamento quando se faz o uso de uma NoC ao invés de utilizar-se conexão ponto-a-ponto entre os módulos. No entanto, considerando-se o funcionamento normal da aplicação em questão, não se obtém nenhuma perda de desempenho na conexão por NoC.

Como já foi relatado, é impossível obter-se os mesmos resultados de desempenho de uma aplicação executando na sua taxa máxima de operação quando se faz o uso de uma NoC comparado a uma conexão direta entre os módulos. No entanto, ao mesmo tempo em que se tem essa limitação, sabe-se que raramente é necessário que os módulos da aplicação operem na sua máxima taxa de processamento, como ocorre no estudo de caso em questão. No entanto, quando se opta por utilizar uma NoC, obtém-se várias outras vantagens, como por exemplo, escalabilidade, que permite que outros módulos sejam adicionados à aplicação sem comprometer o desempenho do sistema.

Como o decodificador de vídeo implementado em NoC não necessita executar na sua máxima taxa de processamento, é possível ter os módulos trabalhando a uma frequência de operação mais baixa e com isso, obter-se um consumo de potência menor. Além disso, nesse caso, pode-se optar por uma configuração da aplicação mínima o suficiente que garanta o correto funcionamento da mesma.

Na figura 7.14 são apresentados os resultados de vazão considerando-se o número de macroblocos obtidos por segundo para a máxima taxa de operação da aplicação. Os porcentuais apresentados na figura 7.14 indicam a redução nos resultados de vazão da aplicação com NoC comparada a implementação sem NoC.

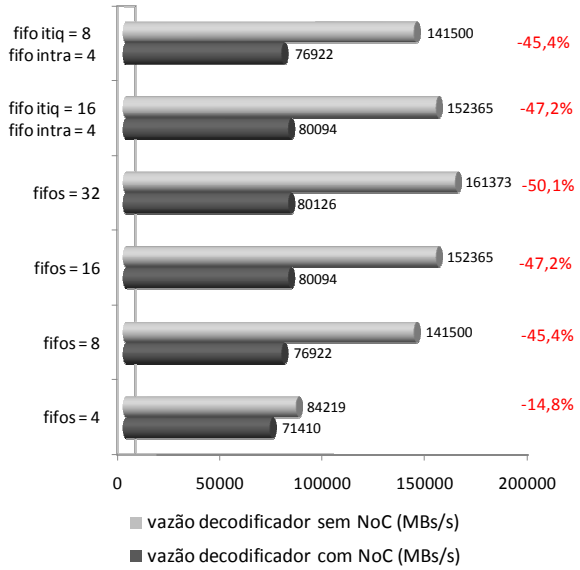



Figura 7.14: Comparação dos resultados de vazão em macroblocos/s para a aplicação com e sem NoC para o processamento máximo da aplicação.

Para os cálculos de vazão apresentados na figura 7.14, considerou-se os resultados de frequência obtidos na tabela 7.2, no entanto, considerando-se a mesma frequência

para o decodificador de vídeo e para o controlador VGA, tanto para a implementação com e sem NoC.

Após a análise de latência e vazão da aplicação, verificaram-se os resultados de síntese desta implementação e compararam-se estes resultados aos obtidos com a arquitetura original. A figura 7.15 apresenta os resultados em LUTs para síntese em FPGA utilizando-se o dispositivo XC2VP30 da família Virtex-II Pro da Xilinx. A versão do ISE utilizada para esta síntese foi a 8.1. Para estes resultados, considerou-se uma configuração suficiente para o correto funcionamento da aplicação. Neste caso, considerou-se profundidade das FIFOs da NoC igual a 4 e largura do canal igual a 16.

A figura 7.15 apresenta, ao lado de cada coluna, o aumento nos resultados de área para cada uma das implementações. Com os resultados de síntese obtidos para este dispositivo, a implementação com NoC e NIs apresentou um aumento de 11% no número de LUTs e em torno de 15% no número *flip-flops* quando comparada a implementação que utiliza conexão direta entre os módulos da aplicação. Este porcentual comprova que para aplicações complexas e com núcleos que apresentam muita lógica e circuitos de armazenamento, como é o caso da implementação do decodificador de vídeo H.264 disponível, a NoC juntamente com as NIs não comprometem muito os resultados de área para síntese em FPGA. No entanto, podem agregar diversas vantagens ao sistema, como escalabilidade e independência entre os módulos. As pequenas variações presentes em cada resultado se devem ao fato de que a implementação original não apresenta todas as FIFOs que se fazem necessárias na implementação com NoC. Sendo assim, nas duas colunas de cima para baixo da figura 7.15, considerou-se a *fifo_adder* igual a 4 e, para os demais casos, considerou-se a *fifo_adder* igual ao valor considerado para as demais FIFOs.

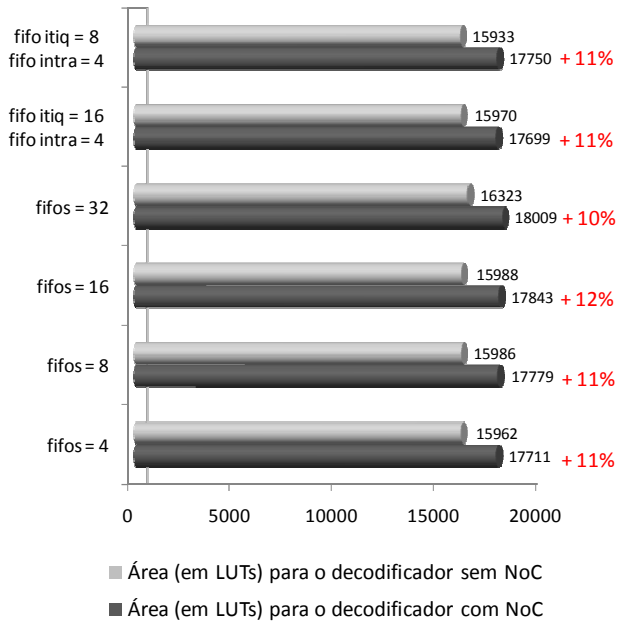



Figura 7.15: Resultados de área para FPGA do decodificador de vídeo com e sem NoC.

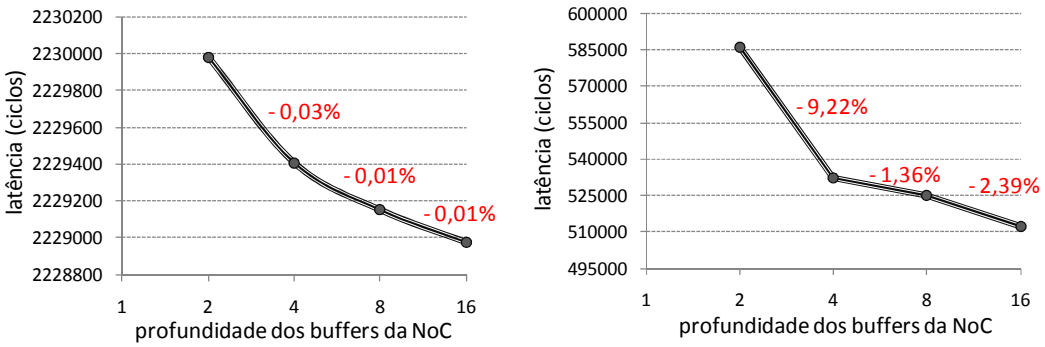
As implementrações com e sem NoC apresentaram os mesmos resultados de frequência para síntese em FPGA para todas as configurações analisadas e esse valor foi

igual a 42MHz. No entanto, conforme já comentado, pode-se utilizar um valor menor de frequência que ainda assim garante-se o correto funcionamento da aplicação.

### 7.2.3 Profundidade das FIFOs da NoC

Outra análise realizada foi com relação à profundidade das FIFOs da NoC. Nesta análise, os resultados de latência e vazão para diferentes tamanhos de FIFOs presentes nos canais da NoC foram verificados e a sua profundidade foi variada de 2 a 16 unidades de *buffers*. Tanto para análise da profundidade das FIFOs presentes nos canais como para largura do canal da NoC, a profundidade das FIFOs presentes na NI, conforme os experimentos realizados foram *fifo_itiq* igual a 8, *fifo_intra* igual a 4 e *fifo_adder* igual a 4. Esta definição foi tomada devido à confirmação de que as FIFOs utilizadas para os dados referentes aos resíduos influenciam mais nos resultados de latência do que as demais FIFOs. Tanto para os resultados de latência variando-se a profundidade das FIFOs da NoC quanto para os resultados variando-se a largura do canal da NoC, foram verificados quantos ciclos são necessários para a obtenção de 200.000 amostras.

A figura 7.16 apresenta os resultados de latência para as diferentes profundidades de FIFO presente nos canais de entrada da NoC. Os resultados apresentados na figura 7.16(a) foram obtidos avaliando-se a latência das amostras na entrada do *descrambler* considerando-se o funcionamento normal do decodificador, já os resultados apresentados na figura 7.16(b) considera a máxima taxa de processamento do decodificador de vídeo. Como se pode observar, os resultados de latência apresentam uma redução maior quando se aumenta a profundidade do buffer de 2 para 4, no entanto, para os demais valores de profundidade dos buffers considerados, a redução de latência é pequena. Os porcentuais apresentados nos gráficos indicam em quanto foi à redução de latência quando se aumenta a profundidade das FIFOs da NoC com relação a profundidade anterior apresentada no gráfico. Considerando-se o funcionamento normal da aplicação, a redução de latência é desprezível.




(a) (b)

Figura 7.16: Resultados de latência variando-se a profundidade dos buffers dos canais da NoC, considerando-se o comportamento normal da aplicação (a) e considerando-se a máxima taxa de processamento da aplicação (b).

Os resultados de vazão para as diferentes possibilidades de profundidade dos buffers da NoC estão apresentados na figura 7.17. Os porcentuais indicados no gráfico referem-se aos ganhos obtidos para os resultados de vazão quando se aumenta a profundidade da FIFO com relação a profundidade anterior apresentada no gráfico. Os resultados de

vazão apresentam as mesmas situações reportadas na análise de latência para os diferentes valores de profundidades das FIFOs da NoC.

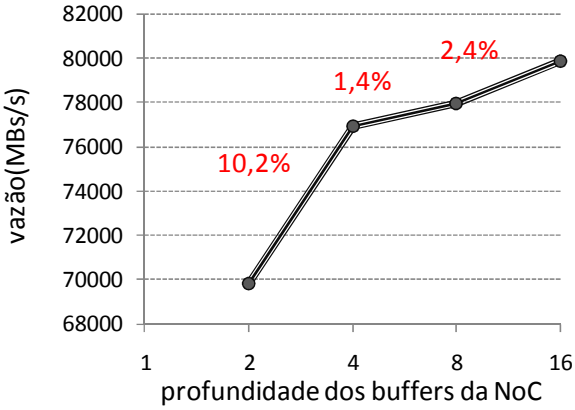



Figura 7.17: Resultados de vazão variando-se a profundidade dos buffers dos canais da NoC para a máxima taxa de processamento da aplicação.

Os resultados de área em LUTs do decodificador de vídeo para diferentes valores de buffers definidos para os canais da NoC estão apresentados na figura 7.18. Estes resultados também foram obtidos para o dispositivo XC2VP30 da família Virtex-II Pro da Xilinx para a versão 8.1 da ferramenta ISE. Os porcentuais indicados nesta figura representam o quanto se obteve de aumento de área a cada aumento da profundidade do buffer comparado à profundidade anterior. Como se pode observar, pelos resultados de área da aplicação, considerando-se as diferentes profundidades de FIFOs para os canais da NoC, o aumento de área foi menor para a maior redução em latência que é quando se opta por profundidade das FIFOs igual a 4 ao invés de 2. Sendo assim, essa a melhor relação desempenho *versus* área e a partir dos resultados obtidos, verifica-se que a profundidade ideal dos buffers da NoC para essa aplicação é 4.

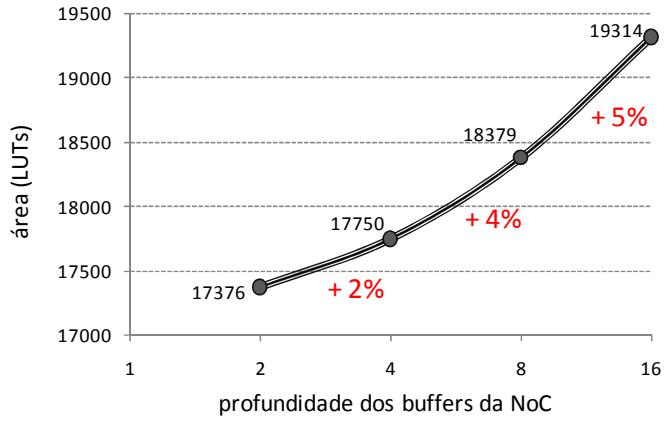
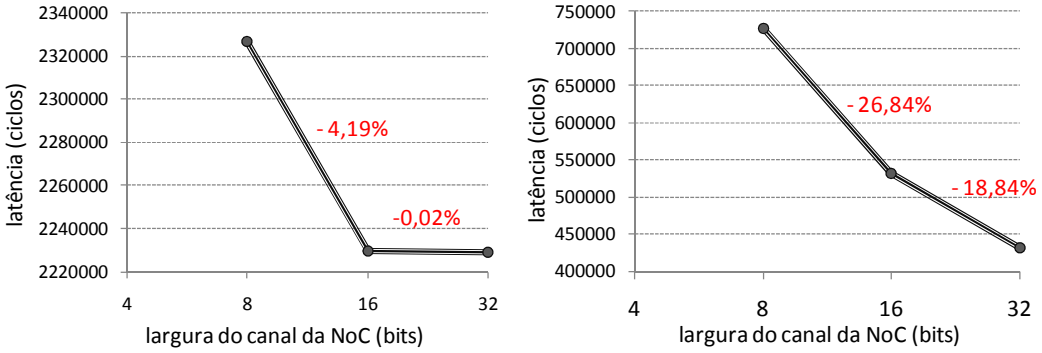



Figura 7.18: Resultados de área em LUTs para o decodificador de vídeo variando-se a profundidade das FIFOs nos canais da NoC.

### 7.2.4 Largura dos Canais da NoC

As mesmas simulações foram realizadas variando-se a largura dos canais da NoC. A figura 7.19 apresenta os resultados de latência do decodificador de vídeo para a largura de dados dos canais igual a 8, 16 e 32 bits (além dos bits de dados, ainda são utilizados mais 2 bits que indicam o tipo de flit). A figura 7.19(a) apresenta os resultados de latência considerando-se a chegada das amostras na entrada do *descrambler*. Os resultados da figura 7.19 ilustram que a latência da aplicação diminui consideravelmente quando a largura do canal passa de 8 para 16 bits, principalmente quando se considera a taxa de processamento máxima da aplicação (figura 7.19(b)). O mesmo se pode observar com relação aos resultados de vazão apresentados na figura 7.20. Já, na taxa de processamento real da aplicação, considerando-se o tempo em ciclos para o recebimento das amostras na entrada do *descrambler*, obtém-se alguma redução quando se aumenta a profundidade dos canais de 8 para 16 bits, porém, a redução de latência é desprezível quando se aumenta a largura do canal de 16 para 32 bits.




(a) (b)

Figura 7.19: Resultados de latência variando-se a largura dos canais da NoC, considerando-se o comportamento normal da aplicação (a) e considerando-se a máxima taxa de processamento da aplicação (b).

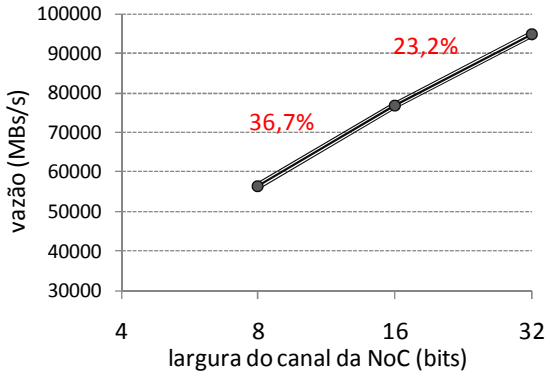



Figura 7.20: Resultados de vazão (MBs/s) considerando-se diferentes valores de largura para o canal da NoC para a máxima taxa de processamento da aplicação.

Conforme já comentado anteriormente, a largura do canal da NoC ideal para cada aplicação irá depender da largura da palavra de dados dos módulos que compõem a aplicação e da taxa com que os pacotes precisam ser enviados para o núcleo destinatário. Sendo assim, considerando-se a largura do canal da NoC igual a 16 para o

decodificador de vídeo, os pacotes são compostos com poucos flits e a aplicação atende aos requisitos de tempo exigidos pelo controlador de vídeo.

Os porcentuais indicados na figura 7.19, da mesma forma que para os demais resultados apresentados, indicam em quanto foi a redução na latência para uma largura do canal maior, considerando-se a largura anteriormente apresentada no gráfico. Na figura 7.20, este porcentual indica em quanto foi o aumento na vazão em função do aumento da largura do canal comparado à largura definida no ponto anterior do gráfico.

A figura 7.21 apresenta os resultados de área em LUTs, considerando-se o mesmo dispositivo de FPGA utilizado na síntese para os demais experimentos. Como se pode observar, os aumentos de área em função da largura do canal da NoC são menores do que quando se aumenta a profundidade das FIFOs dos canais da rede.

Uma outra análise que pode ser feita é com relação aos resultados de desempenho obtidos para o decodificador de vídeo. A partir destes resultados, verificou-se que é obtida uma maior redução de latência e aumento na vazão quando se aumenta a largura dos canais da NoC do que quando se aumenta a profundidade dos buffers dos canais. Isso se deve ao fato de que, para a aplicação analisada, como não existe contingência na rede, pode-se definir canais mais largos e assim, profundidade mínima para os buffers dos canais. Utilizando-se uma largura de canal maior para a rede, os pacotes podem ser repassados rapidamente entre os roteadores e, como os pacotes são constituídos de poucos flits, buffers pequenos podem ser utilizados para essa aplicação.

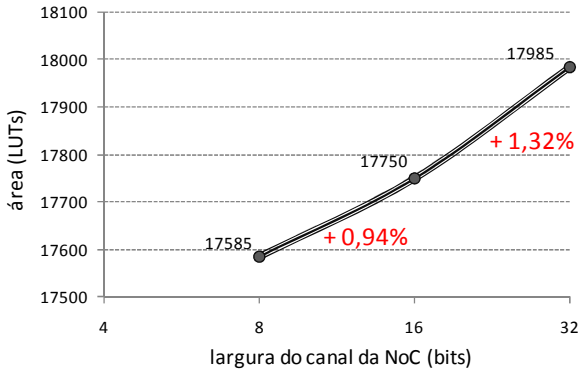



Figura 7.21: Resultados de área em LUTs para o decodificador de vídeo variando-se a largura dos canais da NoC.

### 7.2.5 QoS

Verificou-se que para algumas configurações do decodificador de vídeo, as amostras na saída do *descrambler* não estão disponíveis no tempo necessário para exibição. A tabela 7.1 apresenta as configurações que atendem à taxa de processamento requerida para esta aplicação. Para essa análise, considerou-se resolução QCIF, valor de Qp igual a 22 e frequência de relógio igual para o decodificador de vídeo, para NoC, para as NIs e para o controlador de vídeo (VGA). Nesse caso, como o controlador de VGA opera a 25MHz, todo o sistema estaria operando com essa mesma frequência, que é bem inferior aos resultados de síntese obtidos para a aplicação. No entanto, operando-se a 25MHz, é possível reduzir o consumo de potência da aplicação, caso a mesma fosse projetada para operar na sua máxima frequência. Para algumas configurações apresentadas na tabela 7.1, é possível até mesmo utilizar-se uma frequência de operação menor para o sistema que ainda assim se obtém um correto funcionamento da aplicação.

Uma solução para que algumas das configurações que não atendem a taxa de processamento requerida pelo controlador de vídeo possam ser utilizadas é definir uma frequência de operação maior para o sistema do que para o controlador de vídeo.

O aumento da frequência de operação também pode ser necessário para atender a outros formatos e níveis de vídeo. No entanto, para que as taxas de processamento fiquem ainda mais próximas às obtidas com a implementação ponto-a-ponto da aplicação, é possível utilizar uma frequência de operação maior para a NoC e NIs do que para os módulos do decodificador de vídeo.

Tabela 7.1: Configurações do decodificador de vídeo em NoC que atendem a aplicação.

<table>
<tr><td></td><td colspan="12">Largura do Canal da NoC</td></tr>
<tr><td></td><td colspan="4">8</td><td colspan="4">16</td><td colspan="4">32</td></tr>
<tr><td></td><td colspan="12">Profundidade das FIFOs da NoC</td></tr>
<tr><td>Profundidade das FIFOs nas IRs</td><td>2</td><td>4</td><td>8</td><td>16</td><td>2</td><td>4</td><td>8</td><td>16</td><td>2</td><td>4</td><td>8</td><td>16</td></tr>
<tr><td>todas as fifos = 4</td><td colspan="5" rowspan="9">Não atende</td><td colspan="7" rowspan="9">Atende</td></tr>
<tr><td>todas as fifos = 8</td></tr>
<tr><td>fifo itiq = 8<br>fifo intra = 8<br>fifo adder = 4</td></tr>
<tr><td>fifo itiq = 8<br>fifo intra = 4<br>fifo adder = 4</td></tr>
<tr><td>todas as fifos = 16</td></tr>
<tr><td>fifo itiq = 16<br>fifo intra = 4<br>fifo adder = 4</td></tr>
<tr><td>todas as fifos = 32</td></tr>
</table>

### 7.2.6 Análises Finais dos Resultados do Decodificador de Vídeo

Considerando-se os experimentos realizados, procurou-se definir uma configuração que obtenha o correto funcionamento da aplicação e que utilize o mínimo de recursos possíveis. Utilizando-se a mesma frequência de VGA, a largura do canal mínima para o correto funcionamento da aplicação é de 16 bits e a profundidade das FIFOs das NIs deve ser maior do que 4, considerando-se a profundidade das FIFOs da NoC igual a 2. É possível utilizar profundidade das FIFOs de todas as NIs analisadas igual a 4, no entanto, para isso, as FIFOs da NoC devem ter profundidade mínima igual a 4.

Por último, verificou-se os resultados de síntese da aplicação com e sem NoC para o dispositivo Virtex-5 XC5VLX110 utilizando-se a ferramenta ISE 10. Para a obtenção destes resultados, definiu-se profundidade das FIFOs da NoC igual a 4 e largura do canal de dados igual a 16 bits. Quanto a profundidade das FIFOs das NIs, definiu-se profundidade igual a 8 para a *FIFO_itiq*, por ser a FIFO que apresenta um impacto maior nos resultados de desempenho e profundidade igual a 4 para as demais FIFOs analisadas. Nos resultados de síntese FPGA realizados anteriormente para o dispositivo Virtex-II Pro, a frequência obtida tanto para a implementação com NoC quanto para a implementação original foi a mesma e esse valor foi igual a 42MHz. Essa frequência não apresentou variação considerando-se as diferentes configurações anteriormente analisadas. A frequência obtida para os resultados de sínteses anteriores reflete o caminho crítico do módulo de predição INTRA, conforme verificado pelos resultados obtidos individualmente para cada módulo da aplicação original. Dessa forma, mesmo alterando-se as configurações da aplicação, como a frequência do módulo de predição

INTRA é muito inferior aos resultados de frequência obtidos para os demais módulos, as parametrizações do sistema não influenciam nestes resultados.

Conforme os resultados apresentados na tabela 7.2, verificou-se que, na síntese para FPGA utilizando o dispositivo Virtex-5, diferentemente dos resultados obtidos anteriormente para o Virtex-II Pro, obtém-se um ganho de 7,8% em frequência na implementação do decodificador de vídeo com NoC quando comparada à implementação original. A ocorrência desse ganho se deve ao fato de que o módulo INTRA é conectado ao somador e posteriormente ao *descrambler* sem o uso de FIFOs. Na implementação com NoC, dividiu-se estes módulos em dois roteadores que possuem FIFOs nas interfaces de rede, bem como nos canais de entrada dos roteadores da NoC, adicionando barreiras temporais à aplicação. Como o caminho crítico da aplicação está no processamento desses módulos, certamente a obtenção desse aumento em frequência quando se utiliza uma NoC se deve à redução do caminho crítico nessa etapa da aplicação. No entanto, para os valores de síntese apresentados na tabela 7.2 obtiveram-se porcentuais de aumento nos resultados de área maior do que os obtidos na síntese para o dispositivo da família Virtex-II Pro.

Tabela 7.2: Resultados de síntese do decodificador de vídeo para FPGA utilizando o dispositivo Virtex-5 da Xilinx.

| | **Slice Registers** | **Slice LUTs** | **LUT/FF Pairs** | **BRAM/ FIFO** | **Frequência Máxima (MHz)** |
|---|---|---|---|---|---|
| **Decodificador sem NoC** | 9146 | 10453 | 3329 | 27 | 72,9 |
| **Decodificador com NoC** | 10704 | 11966 | 4258 | 30 | 78,6 |
| **Diferença (%)** | 17% | 14,5% | 27,9% | 11,1% | 7,8% |

Verificou-se ainda, para o mesmo dispositivo de FPGA (Virtex-5, XCVLX110), os resultados de área obtidos para cada uma das NIs + IP presentes na implementação do decodificador de vídeo. A tabela 7.3 apresenta estes resultados. Cada interface apresenta necessidades distintas com relação ao número de FIFOs e com relação à profundidade de cada uma delas. Além disso, algumas NIs foram implementadas com algumas necessidades específicas, e ainda existem casos em que as FIFOs das NIs estão presentes no próprio IP. Por estes motivos, o aumento no número de LUTs não é constante para cada NI + IP quando comparada a área obtida apenas para o IP.

Tabela 7.3: Resultados de área dos módulos que compõem o decodificador de vídeo, com e sem as interfaces de rede, em FPGA utilizando o dispositivo Virtex-5 da Xilinx.

| Módulo | LUTs |
|---|---|
| NAL | 38 |
| NAL + NI | 185 |
| PARSER | 3012 |
| PARSER + NI | 3287 |
| ITIQ | 2616 |
| ITIQ + NI | 2755 |
| INTRA | 4232 |
| INTRA + NI | 4866 |
| DESCRAMBLER | 119 |
| DESCRAMBLER + NI | 157 |

## 7.3 Considerações

Neste capítulo foram apresentadas as NIs utilizadas para cada módulo que compõe o decodificador de vídeo. Após o detalhamento da implementação das interfaces de rede, foram verificados os resultados de latência, vazão, área e frequência obtidos para o decodificador de vídeo considerando-se diferentes configurações do sistema. Estes resultados foram verificados variando-se a profundidade dos buffers presente nas NIs, a profundidade dos buffers presentes na NoC e a largura do canal da NoC. Especificamente para a implementação do decodificador de vídeo utilizada neste trabalho, verificou-se que é possível utilizar uma frequência de operação inferior à frequência de operação máxima da aplicação. A partir dos resultados de desempenho obtidos para o decodificador de vídeo conectado em NoC, pôde-se definir a arquitetura mínima que satisfaz as taxas de processamento requeridas para esta aplicação. No próximo capítulo serão apresentadas as conclusões obtidas com o desenvolvimento deste trabalho e também serão levantadas algumas possibilidades de continuação do mesmo.

# 8 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foram apresentadas propostas de interfaces de rede que podem ser utilizadas para a conexão dos módulos de uma aplicação a uma rede-em-chip. Estas propostas são consideradas genéricas, pois apresentam configurações para diferentes parâmetros. Através desse trabalho, puderam-se analisar diversas necessidades que devem ser atendidas por uma interface de rede quando uma aplicação é conectada por uma NoC. Além de garantir o correto funcionamento da aplicação, adaptando o protocolo da rede ao protocolo do EP, empacotando e desempacotando os dados corretamente, as interfaces de redes também têm a responsabilidade de garantir o desempenho de comunicação da aplicação.

Através da implementação das interfaces de rede, verificou-se que as FIFOs presentes nas NIs devem ser configuradas conforme a necessidade da aplicação, além de elas apresentarem outras vantagens, como possibilidade de sincronização dos dados e maior controle no envio e recebimento dos pacotes. Contudo, deve-se considerar a profundidade mínima de FIFO para as NIs que atenda a largura de banda exigida pelo EP, evitando-se com isso, o uso de FIFOs maiores que, no entanto, não contribuem para o desempenho da aplicação, porém comprometem o consumo de potência do sistema. Neste trabalho, foram levantados os resultados de síntese em *standard cells* para a tecnologia 0,18um alterando-se diversos parâmetros, tais como a profundidade da FIFO presente na NI, a largura do canal da NoC e a largura do pacote de dados enviados para a rede.

As interfaces de rede projetadas foram validadas utilizando-as para a conexão do decodificador de vídeo H.264 em NoC. Considerando-se essa implementação, uma série de resultados foram extraídos e comparados para diferentes configurações do sistema. Também foram comparados os resultados obtidos com a implementação original do decodificador de vídeo que não utiliza uma rede-em-chip para interconexão dos módulos. A partir dessa análise, verificou-se que para o formato de vídeo QCIF, a implementação que utiliza NoCs e NIs pode operar a 25MHz para diferentes configurações do sistema. No entanto, se for preciso, é possível ainda que a implementação do decodificador de vídeo em NoC opere a um valor de frequência maior para possibilitar a decodificação de vídeo para outros formatos.

Com os resultados obtidos, comprovou-se que, como a implementação do decodificador de vídeo não apresenta rotas de pacotes que dividam o mesmo caminho da rede, é possível aumentar o desempenho do sistema definindo-se canais mais largos para a NoC. Esta solução é preferida ao invés de utilizar FIFOs maiores para os canais

da NoC, pois, aumentando-se a largura do canal da NoC, obtém um aumento maior no desempenho a um custo menor de recursos.

Para a síntese em FPGA realizada para o dispositivo Virtex 5, foi possível ainda obter uma frequência máxima de operação da implementação com NoC e NIs um pouco maior do que a implementação do decodificador de vídeo que utiliza conexão ponto-a-ponto. Como tanto as NIs como a NoC apresentam FIFOs em suas implementações, esse ganho em frequência se deve à inserção destes dispositivos de armazenamento que possibilitaram reduzir o caminho crítico da aplicação.

Além disso, verificando-se os resultados de área obtidos para a implementação original do decodificador de vídeo pôde-se considerar que a implementação que utiliza NoC e NIs não apresentou um grande aumento nestes resultados para a síntese obtida para o dispositivo da família Virtex-II Pro.

Comparando-se a utilização de uma NoC para a conexão dos módulos ao invés de realizar-se conexão direta entre eles, é natural obter-se uma queda de desempenho para a aplicação operando na sua taxa máxima de operação. No entanto, alguns aspectos podem ser considerados com relação a estes resultados. Primeiramente, considerando-se aplicações complexas, nem sempre é possível utilizar conexão ponto-a-ponto entre os módulos e dificilmente é exigido que uma aplicação opere na sua taxa máxima. Um sistema com um alto grau de complexidade, que integra inúmeros EPs, provavelmente a utilização de uma rede-em-chip se fará necessária. Neste trabalho, os resultados foram obtidos a partir de uma conexão com poucos módulos que serviram apenas para a validação das interfaces de rede desenvolvidas. Uma NoC possibilita paralelismo entre as comunicações, que normalmente são exigidos em SoCs para atender aos requisitos de desempenho da aplicação. Contudo, o uso de uma NoC ainda permite reuso dos EPs, bem como, apresenta escalabilidade, possibilitando a inserção de núcleos à rede sem comprometer o desempenho do sistema.

Por fim, tem-se como objetivo na continuidade desse trabalho, agregar a esta implementação outros módulos que ainda não foram integrados ao decodificador de vídeo e até mesmo outros dispositivos, que juntamente ao decodificador de vídeo, constituem um *set-top box*, possibilitando, assim, analisar outras necessidades de interfaces de redes ainda não incorporadas nesse trabalho.

## 8.1 Trabalhos Futuros

Como trabalhos futuros, vários caminhos podem ser seguidos a partir desta primeira implementação do decodificador de vídeo H.264 em NoC. Uma primeira possibilidade é dar continuidade à integração dos módulos que constituem o decodificador de vídeo segundo o padrão H.264. Para a obtenção de um decodificador de vídeo completo, conforme o padrão H.264, ainda precisam ser adicionados a esta implementação os módulos de filtro de deblocagem e predição INTER-quadros. No entanto, para a integração deste último módulo, serão necessárias a inserção de uma memória para armazenar as imagens que são utilizadas como referência por este módulo e ainda alterações no módulo *parser.*

A partir da implementação completa do decodificador de vídeo H.264, outros módulos ainda podem ser estudados para a integração junto ao sistema a fim de se constituir um *set-top box*. Agregando-se tais módulos à rede, outras necessidades de

interfaces de rede e alterações da NoC poderão ser estudadas e analisadas. No entanto, partindo da própria implementação já existente, algumas pequenas alterações podem ser aplicadas a fim de realizarem-se alguns estudos e obterem-se algumas conclusões do sistema sobre outros aspectos.

Podem ser consideradas ainda outras subdivisões dos módulos e com isto, outras possibilidades de mapeamentos para a rede. Por exemplo, pode-se dividir parte do módulo CAVLD em outro nodo da rede, como também se pode estudar a possibilidade de duplicar o módulo ITIQ. Estas variações da implementação original permitirão que se obtenham mais nodos na rede, possibilitando-se encontrar outras conclusões e análises desse projeto. Imagina-se ainda que, a implementação do decodificador de vídeo H.264 em NoC juntamente com os demais módulos necessários para a obtenção de um *set-top box*, poderá ser utilizada como *benchmark* para inúmeras outras análises relacionadas à estudos de SoCs em NoCs.

Uma outra análise que pode ser considerada na continuação deste trabalho é a realização de um estudo com relação as frequências de operação dos núcleos. Devido à heterogeneidade dos módulos, cada EP pode operar em uma frequência diferente. No entanto, também é possível considerar a NoC e as NIs operando em uma frequência maior da frequência definida para os EPs. Sendo assim, nesse caso, estudos com relação ao uso de GALS precisarão ser considerados para a obtenção do correto funcionamento da aplicação. Ainda com relação à frequência de operação dos módulos, pode-se verificar outras situações, configurando-se adequadamente a frequência da NoC, das NIs e dos módulos do decodificador com o objetivo de atender às taxas exigidas para obter decodificação para outros formatos de vídeo.

Por fim, as NIs ainda poderão ser utilizadas para a implementação de outras aplicações onde outras necessidades no projeto para as interfaces poderão ser levantadas. Uma interessante utilização das interfaces de rede é adaptá-las a alguns processadores comumente conhecidos, desenvolvendo *wrappers* específicos para o protocolo destes dispositivos. Ainda podem ser consideradas outras alterações ao projeto das interfaces de rede com o objetivo de adaptá-las em tempo de execução, conforme as mudanças de aplicação do sistema.

# REFERÊNCIAS


AGOSTINI, L. V., **Desenvolvimento de Arquiteturas de Alta Performance Dedicadas à Compressão de Vídeo Segundo o Padrão H.264**. Tese (Doutorado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, RS, 2007.

AL FARUQUE A., HENKEL, J. Minimizing Virtual Channel Buffer for Routers in On-Chip Communication Architectures, Conference on Design, Automation and Test in Europe – DATE, p. 1238-1243, 2008.

AHMAD, B., ARSLAN, T. Dynamically Reconfigurable NoC for Reconfigurable MPSoC. Proceedings of the IEEE Custom Integrated Circuits Conference, p. 277-280, 2005.

AMBA Specification Rev2.0, ARM Ltd, 1999.

AGARWAL A. et al. System-Level Modeling of a Noc-Based H.264 Decoder, Annual IEEE Systems Conference, p. 1-7, 2008.

ARM , ARM7EJ-S processor, Disponível em:

<www.arm.com/products/processors/classic/arm7/arm7ej-s.php>. Acesso em: março de 2010.

ARTERIS, NoC industrial Arteris. Disponível em: <http://www.arteris.com/>. Acesso em: março de 2010.

ATTIA, B. et al. A Modular Network Interface Adapter Design for OCP Compatibles NoC. International Journal of Computer and Network Security – IJCNS, p. 101-109, 2009.

AZEVEDO, A. P. **MoCHA: Arquitetura Dedicada para a Compensação de Movimento em Decodificadores de Vídeo de Alta Definição, Seguindo o Padrão H.264**. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre, RS, 2006.

BEIGNÉ E. e VIVET P. Design of On-chip and Off-chip Interfaces for a GALS NoC Architecture, ASYNC, pp. 172-183, 2006.

BENINI L., DE MICHELI G. Powering Networks on Chips. Proceedings of the 14th International Symposium on Systems Synthesis, Montreal, p. 33-38, 2001.

BENINI L., DE MICHELI G. Networks on Chips: a New SoC Paradigm, IEEE Computer, p. 70-78, 2002.

BERTOZZI, D. et al. NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip, IEEE Transaction on Parallel and Distributed System, p. 113-129, 2005.

BHOJWANI, P., MAHAPATRA, R. Interfacing Cores with On-Chip Packet-Switched Networks. International Conference on VLSI Design, VLSI, p. 382 – 387, 2003.

BJERREGAARD T., MAHADEVAN S., A Survey of Research and Practice of NoC. ACM Computing Surveys, USA, 2006.

BOBDA, C. et al. DyNoC: A Dynamic Infrastructure for Communication in Dynamically Reconfigurable Devices. International Conference on Field Programmable Logic and Applications, p. 153-158, 2005.

BOLOTIN, E. et al. QNoC: QoS Architecture and Design Process for Network on Chip. Journal of Systems Architecture, v.50, n.2, p. 1-24, 2004.

CHAKRABORTY A. e GREENSTREET, M.. A Minimal Source-Synchronous Interface, International ASIC/SOC Conference, p. 443- 44, 2002.

CHAN, J., PARAMESWARAN S. NoCOUT: NoC Topology Generation with Mixed Packet-Switched and Point-to-Point Networks. Asia and South Pacific Design Automation Conference - ASPDAC, p. 265 – 270, 2008.

CHANG, J. et al. Star-Mesh NoC Based Multi-Channel H.264 Decoder Design. International SoC Design Conference – ISOCC, p. 170-173, 2008.

CHANG, K.C, SHEN, J.S., CHEN, F.T. Evaluation and Design Trade-offs Between Circuit-Switched and Packet-Switched NOCs for Application-Specific SOCs, Design Automation Conference – DAC, p. 143-148, 2006.

DALLY, W., TOWLES, B. Route Packets, Not Wires: On-Chip Interconnection Networks. Design Automation Conference - DAC, p. 684-689, 2001.

DE MICHELI, G., BENINI, L. **Networks on Chip: Technology and Tools**, San Francisco: Editora Morgan Kaufmann, p. 203 – 284, 2006.

DEPRÁ, D., ROSA, V., BAMPI, S. A Novel Hardware Architecture Design for Binary Architecture Decoder Engines Based on Bit-Stream Flow Analysis. Symposium on Integrated Circuits and System Design, SBCCI, p. 239-244, 2008.

DINIZ, C., **Arquitetura de Hardware Dedicada para a Predição Intra-Quadro em Codificadores do Padrão H.264/AVC de Compressão de Vídeo**. 2009. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

DUATO, J. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. IEEE Transactions on Parallel and Distributed Systems, p. 1320-1331, 1993.

DUATO, J., A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks, IEEE Transactions on Parallel and Distributed Systems, p. 1055-1067, 1995.

EBRAHIMI, E., et al. Efficient Network Interface Architecture for Network-on-Chips. Proceedings of 27th Norchip, IEEE Press, p. 1 - 4, 2009.

FERRANTE, A., MEDARDONI, S., BERTOZZI, D. Network Interface Sharing Techniques for Area Optimized NoC Architecture. Proceedings of EUROMICRO Conference on Digital System Design Architectures, p. 10-17, 2008.

GUERRIER P.; GREINER, A. A Generic Architecture for On-Chip Packet-Switched Interconnections. Design Automation and Test in Europe - DATE, p. 250–256, 2000.

HAIBO Z., PANDE .P, GRECU C. Performance Evaluation of Adaptive Routing Algorithms for achieving Fault Tolerance in NoC Fabrics. IEEE International Conf. on Application -Specific Systems, Architectures and Processors, ASAP, p. 42-47, 2007.

HANSSON, A. and GOOSSENS K. Trade-Offs in the Configuration of a Network on Chip for Multiple Use-Cases. International Symposium on Networks-on-Chip, NOCS, p. 233-242, 2007.

INOCs – NoC industrial INOCs. Disponível em: <http://www.inocs.com>. Acesso em: março de 2010.

ITU – INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.264/AVC (05/03)**: advanced video coding for generic audiovisual services. [S.l.], 2003.

ITU – INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.264/AVC (11/07)**: advanced video coding for generic audiovisual services. [S.l.], 2007.

ITU – INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Home**. Disponível em: <www.itu.int/ITU-T/>. Acesso em: dezembro 2009.

JERGER, N., PEH, L. **On Chip Networks**, Synthesis Lectures on Computer Architecture. Toronto: Editora Morgan & Claypool, 2009.

KIM, D. et al. NIUGAP: Low Latency Network Interface Architecture with Gray Code for Networks-on-Chip. IEEE International Symposium on Circuits and Systems - ISCAS, p. 3901-3905, 2006.

KUMAR, S. et al. A Network-on-Chip Architecture and Design Methodology. Symposium on Very Large Integration Scale, p. 117–124, 2002.

KUNDU S. E CHATTOPADHYAY S. Interfacing Cores and Routers in Network-on-Chip Using GALS, ISIC, pp. 154-157, 2007.

LAHTINEN V., SALMINEN E., Comparison of synthesized bus and crossbar interconnection architectures. IEEE International Symposium on Circuits and Systems - ISCAS. v. 5, p. 433-436, 2003.

LEE, E., MESSERSCHMITT, D. “Synchronous Data Flow”, Proceedings of the IEEE, v. 75, p. 1235- 1245, 1987.

LEE, E.; SANGIOVANNI_VINCENTELLI, A. **The Tagged Signal Model**: A Preliminary Version of a Denotational Framework for Comparing Models of Computation. Berkely, University of California: 1996.

LEE, S., LEE, C., LEE, H. A New Multi-Channel On-Chip-Bus Architecture for System-on-Chips. **Proceedings…** IEEE International SOC Conference, p. 305–308, 2004.

LEE, E. S. et al. A Generic Network Interface Architecture for a Networked Processor Array (NePA). **Proceedings**... Architecture of Computing Systems – ARCS, vol. 4934, p. 247-260, 2008.

LIU, D. et al. SoCBUS: The solution of high communication bandwidth on chip and short TTM”, **Proceedings** … Real Time and Embedded Computing Conference, 2002.

LUDOVICI D.; STRANO, A.; BERTOZZI D. Architecture Design Principles for the Integration of Synchronization Interfaces into Network-on-Chip Switches - International Workshop on Network-on-Chip Architectures – NoCArC, p. 31-36, 2009.

MARCON, C. **Modelo para o Mapeamento de Aplicações em Infra-estruturas de Comunicação Intrachip**. 2005. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MATOS, D.; CONCATTO, C.; KREUTZ, M.; CARRO, L.; KASTENSMIDT, F.; SUSIN, A. Highly Efficient Reconfigurable Routers in NoCs. IFIP/IEEE International Conference on Very Large Scale Integration - VLSI-SoC., 2009a.

MATOS, D.; CONCATTO, C.; KOLOGESKI, A.; KREUTZ, M.; CARRO, L.; KASTENSMIDT, F.; SUSIN, A. Adaptive Router Architecture Based on Traffic Behavior Observability. Network on Chip Architectures - NoCARC, 2009b.

MATOS, D.; CARRO, L. e SUSIN, A. Associating Packets of Heterogeneous Cores Using a Synchronizer Wrapper for NoCs. IEEE International Symposium on Circuits and Systems – ISCAS, 2010.

MING L., QING-AN Z. e WEN-BEN J. DyXY - A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Network on Chip, Design Automation Conference - DAC, p. 849-852, 2006.

MÖLLER, L. et al. Infrastructure for Dynamic Reconfigurable Systems: Choices and Trade-offs. Symposium on Integrated Circuits and Systems Design - SBCCI, MG, Brasil, p. 44-49, 2006.

NICOPOULOS, C., PARK, D., KIM, J., VIJAYKRISHNAN, N., YOUSIF, S., DAS R. ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers. Int. Symp. Microarchitecture - MICRO, p. 333 – 346, 2006.

NING W.; FEN G. e FEI W. Design of a GALS Wrapper for Network on Chip, World Congress on Computer Science and Information Engineering – CSIE, p. 592-595, 2009.

ONO, T. e GREENSTREET M. A Modular Synchronizing FIFO for NoCs, International Symposium on Networks-on-Chip – NOCS, p. 224-233, 2009.

PASRICHA, S., DUTT, N., **On-Chip Communication Architectures: System on Chip Interconnect**, San Francisco: Editora Morgan Kaufmann, p. 439 - 466, 2008.

PEREIRA, F., **Hardware Implementation of a High Performance Minimalist H.264 Video Decoder.** 2009. Master's Thesis (Multimedia Communications Master), Metropolia, Finland.

PURI, A. et al. **Video Coding Using the H.264/MPEG-4 AVC Compression Standard**. Elsevier Signal Processing: Image Communication, n. 19, p.793–849, 2004.

RADULESCU, A. et al. **An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration.** Design, Automation, and Test in Europe – p. 20878 – 29883, 2004.

REGO, R., **Projeto e Implementação de uma Plataforma MP-SoC usando SystemC**, Dissertação (Mestrado em Sistemas e Computação), UFRN, Natal, 144 pag., 2006.

RICHARDSON, I. **Video Codec Design – Developing Image and Video Compression Systems**. Chichester: John Wiley and Sons, 2002.

RICHARDSON, I. **H.264/AVC and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia**. Chichester: John Wiley and Sons, 2003.

ROSA, V., SUSIN, A. e BAMPI, S. **A High Performance H.264 Deblocking Filter.** Pacific Rim Symposium on Advances in Image and Video Technology, p. 855-964, 2009.

ROSA, V. **Projeto de Arquiteturas de Hardware para a Compressão de Vídeo no Padrão H.264/AVC**. Proposta de Tese (Doutorado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, RS, 2009b.

RYU K., SHIN E., MOONEY V. A Comparison of Five Different Multiprocessor SoC Bus Architectures. **Proceedings…** Euromicro Symposium Digital Systems Design, p. 202-209, 2001.

SHEIBANYRAD, A., GREINER A. Hybrid-Timing FIFOs to use on Networks-on-Chip in GALS Architectures, International Conference on Embedded Systems and Applications - ESA, pp. 27-33, 2007.

SILISTIX – NoC industrial Silistix. Disponível em: http://www.silistix.com. Acesso em: março de 2010.

SILVA, T. PEREIRA, F., SUSIN, A., BAMPI, S., AGOSTINI, L. **High Performance and Low Cost Architecture for H.264/AVC CAVLD Targeting HDTV.** Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design – SBCCI, p. 259-263, 2009.

SINGH, P. S. et al. Generic Network Interface for Plug and Play NoC Based Architecture. Applied Reconfigurable Computing - ARC, pp. 287-298, 2006.

STAEHLER, W. T.; BERRIEL, E. A.; SUSIN, A. A.; BAMPI, S. Architecture of an HDTV Intraframe Predictor for a H.264 Decoder. **Proceedings…** IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), IEEE, 2006. Nice: IEEE, 2006.

STENSGAARD, B.; SPARSO, J. ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology, Networks-on-Chip - NoCS, p. 55 – 64, 2008.

SULLIVAN, G.; et al. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In: Conference on Applications of Digital Image Processing, SPIE, 2004. **Proceedings…** Denver: SPIE, 2004.

TANENBAUM A. S., **Organização Estruturada de Computadores**, Quarta edição, Editora LTC, 1999.

THONNART, Y.; BEIGNÉ, E. e VIVET, P. Design and Implementation of a GALS Adapter for ANoC Based Architectures, ASYNC, p.13-22, 2009.

VELLANKI P.et al. Quality-of-Service and Error Control Techniques for Network-on-Chip Architectures. Great Lakes Symposium on VLSI. Boston, p. 45-50, 2004.

XU, J. et al. A Design Methodology for Application-Specific Networks-on-Chip, ACM Transactions on Embedded Computing Systems -TECS, p. 263-280, 2006.

XUNING, C. e PEH, L. Leakage power modeling and optimization in interconnection networks. International Symposium on Low Power Electronics and Design - ISLPED, p. 90-95, 2003.

ZEFERINO, C. et al. A Study on Communication Issues for Systems-on-Chip, 15th Symposium on Integrated Circuits and Systems Design. p. 121 – 126, 2002.

ZEFERINO, C. **Redes-em-Chip: Arquiteturas e Modelos para Avaliação e Área e Desempenho**. 2003. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.