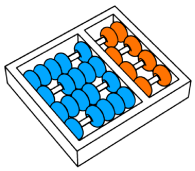
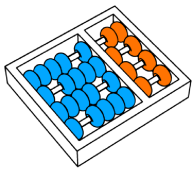


Flávia de Oliveira Santos

"MediaBox: Uma Plataforma Baseada em NoCs para Aplicações Multimídia"

CAMPINAS
2013



Universidade Estadual de Campinas
Instituto de Computação

Flávia de Oliveira Santos

"MediaBox: Uma Plataforma Baseada em NoCs para Aplicações Multimídia"

Orientador(a): **Prof. Dr. Guido Araújo**

Co-Orientador(a): **Prof. Dr. Sandro Rigo**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

Este exemplar corresponde à versão final da Dissertação defendida por Flávia de Oliveira Santos, sob orientação de Prof. Dr. Guido Araújo.

Assinatura do Orientador(a)

CAMPINAS
2013

FICHA CATALOGRÁFICA ELABORADA POR
MARIA FABIANA BEZERRA MULLER - CRB8/6162
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Santos, Flávia de Oliveira, 1986-
Sa59m MediaBox : uma plataforma baseada em NoCs para aplicações multimídia / Flávia de Oliveira Santos. – Campinas, SP : [s.n.], 2013.

Orientador: Guido Costa Souza de Araújo.
Coorientador: Sandro Rigo.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Arquitetura de computador. 2. Sistemas de computação. I. Araújo, Guido Costa Souza de,1962-. II. Rigo, Sandro,1975-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em inglês: MediaBox : a NoCs based platform for multimedia applications
Palavras-chave em inglês:
Computer architecture
Computing systems
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Guido Costa Souza de Araújo [Orientador]
Rodolfo Jardim de Azevedo
Ney Laert Vilar Calazans
Data de defesa: 05-03-2013
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 05 de Março de 2013, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Ney Laert Vilar Calazans
FACIN / PUCRS

Prof. Dr. Rodolfo Jardim de Azevedo
IC / UNICAMP

Prof. Dr. Guido Costa Souza de Araújo
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

MediaBox: Uma Plataforma Baseada em NoCs para Aplicações Multimídia

Flávia de Oliveira Santos

05 de março de 2013

Banca Examinadora:

- Prof. Dr. Guido Araújo (Orientador)
- Prof. Dr. Ney Laert Vilar Calazans
  Faculdade de Informática - PUCRS
- Prof. Dr. Rodolfo Jardim de Azevedo
  Instituto de Computação - UNICAMP

Resumo

Arquiteturas tradicionais para sistemas modernos consistem em SoCs com múltiplos processadores integrados em um único *chip* conhecidos como MPSoCs. A maioria dos IPs de um MPSoC são altamente configuráveis, cada um com uma complicada relação custo-benefício entre métricas como desempenho, área e consumo de energia, tornando o espaço de projeto de um MPSoC bastante amplo.

Aliado a essa complexidade de projeto está o fato de que não é possível realizar a verificação de um MPSoC sem a aplicação em software e muito menos desenvolver o software sem modelos de hardware. Por isso é importante que os projetistas possam começar com modelos do sistema completo nos quais subsistemas possam ser independentemente substituídos por modelos refinados, de forma a haver uma validação contínua do sistema.

Neste contexto, o conceito de plataforma virtual tem sido utilizado para desenvolvimento paralelo de hardware e software. Através de plataformas virtuais, projetistas podem analisar antecipadamente muitos problemas de projeto em um MPSoC, obtendo assim estimativas para consumo de energia, tráfego de barramento, uso de memória, eficiência dos periféricos e, principalmente, desempenho do sistema como um todo.

Este projeto visa prover uma plataforma virtual em nível ESL chamada MediaBox. A MediaBox tem como mecanismo de interconexão uma NoC (*Network-on-Chip*) que introduz o conceito de rede na plataforma e possibilita a comunicação simultânea entre seus IPs. A plataforma desenvolvida possibilita a avaliação de desempenho de sistemas multimídia e sua execução facilita a produção de grandes quantidades de informação poupando tempo e esforço ao desenvolvedor.

O estudo de caso realizado demonstra que a MediaBox é uma boa solução para simular aplicações multimídia e para análise de desempenho. Devido ao grande tráfego existente entre os IPs, o uso de uma NoC como meio de interconexão mostrou-se eficaz. A MediaBox possibilita o uso de diferentes configurações através de um arquivo de configuração e de um mapa de endereçamento que permitem explorar essas opções. Essa flexibilidade permite aos usuários conceber e testar diferentes arquiteturas através das quais pode ser estudado o comportamento e o desempenho de sistemas multiprocessados em um *chip*.

Abstract

Traditional architectures for modern systems consist on SoCs with multiple processors integrated in a single chip known as MPSoCs. Most of the IPs in a MPSoC are highly configurable, each with a complicated trade-off between metrics such as performance, area and energy consumption making the design space of a MPSoC incredibly wide.

Allied to this project complexity is the fact that it is not possible to perform verification of a MPSoC without the application in software and much less develop the software without hardware models. For this reason, it is important that designers start with complete system models in which subsystems may be independently replaced by refined models, so that there is a continuous system validation.

In this context, the concept of a virtual platform has been used for parallel development of hardware and software. Through virtual platforms, designers are able to analyze in advance many design problems in a MPSoC, thus obtaining estimates for energy consumption, bus traffic, memory usage, peripherals efficiency and mainly performance of the system as a whole.

This project aims to provide a virtual platform in ESL called MediaBox. The MediaBox interconnection mechanism is a NoC (Network-on-chip) that introduces the concept of a network inside a platform and enables simultaneous communication between its IPs. The developed platform enables multimedia systems performance evaluation and its execution facilitates the production of a large amount of information saving the developer's time and effort.

The case study developed demonstrates that MediaBox is a good solution for simulating multimedia applications and for performance analysis. Due to the amount of traffic between the IPs, the use of a NoC as the interconnection mechanism proved to be effective. MediaBox enables the usage of different configurations through a configuration file and an address map that allows to explore these options. This flexibility allows the users to conceive and test different architectures through which the behavior and the performance of multiprocessor systems in a single chip can be studied.

Agradecimentos

Eu gostaria de agradecer aos professores Guido Araújo e Sandro Rigo pela orientação no mestrado. Obrigada por toda dedicação, ideias e conselhos que com certeza foram fundamentais para o desenvolvimento do trabalho.

Agradeço também ao professor André Luís Meneses Silva por ter me apresentado a área de pesquisa bem como pelo apoio e incentivo para que eu me inscrevesse no mestrado.

Agradeço aos meus amigos por todo apoio durante o mestrado. Obrigada pelas conversas e pelos momentos de divertimento que ajudaram a manter o nível de stress baixo (na medida do possível)! Teria sido muito mais difícil sem vocês.

Por fim, agradeço aos meus pais e a minha família por todo apoio durante essa jornada. Mãe, obrigada por sua incrível paciência, constante incentivo e inúmeras visitas à Campinas, não teria chegado até aqui sem você. Pai, obrigada por nunca ter duvidado de mim e ter feito o possível para me ajudar a terminar o mestrado. Minha irmã Cíntia, obrigada por sempre atender o telefone e estar sempre disposta a me animar. Essa conquista não teria sido possível sem a ajuda de vocês.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O rápido avanço tecnológico na fabricação de circuitos integrados proporcionou o surgimento de uma nova metodologia de projeto de circuitos integrados chamada SoC (em inglês, *System-on-a-chip*). Um SoC é um circuito integrado complexo que integra em um único *chip* os principais elementos funcionais de um produto final [29]. Ele pode ser encontrado, cada vez mais, no cotidiano das pessoas como por exemplo em telefones celulares, computadores portáteis, *smartphones* e eletrodomésticos.

No contexto de SoCs, uma plataforma é uma infraestrutura de componentes de hardware e software. Ela permite a execução de computação e comunicação entre os componentes que a formam.

Os componentes de hardware de uma plataforma são formados pelos elementos que possuem uma funcionalidade fixa. Exemplos de componentes de hardware são barramentos, DMA, memória, etc. Em geral, tais componentes podem ser parametrizados. Esses parâmetros são utilizados para adequar os componentes à plataforma em desenvolvimento.

Os componentes de software são elementos computacionais que podem ser programados. Sua funcionalidade é adequada de acordo com a aplicação desejada. Módulos de software são modelados através de uma linguagem de alto nível, como C, ou utilizando o próprio conjunto de instruções dos componentes de hardware programáveis. Exemplos de componentes de hardware programáveis são processadores, DSPs (*Digital Signal Processor* — Processador de Sinais Digitais) [57], etc.

A Figura 1.1 apresenta uma arquitetura SoC encontrada em várias aplicações, desde um telefone celular até um sistema de controle automotivo. Os componentes do SoC descrito na Figura 1.1 são chamados de *IP-Cores* (*Intellectual Property Cores* ou simplesmente IPs). Um IP é um componente de hardware ou software que pode ser integrado a uma plataforma. Pode-se observar na Figura 1.1 a presença de dois processadores que executam a computação do sistema, uma memória que armazena código e dados para os processadores, um barramento como estrutura de comunicação entre os IPs e os seguintes

periféricos: controlador de DMA, controlador de memória flash e interface externa.

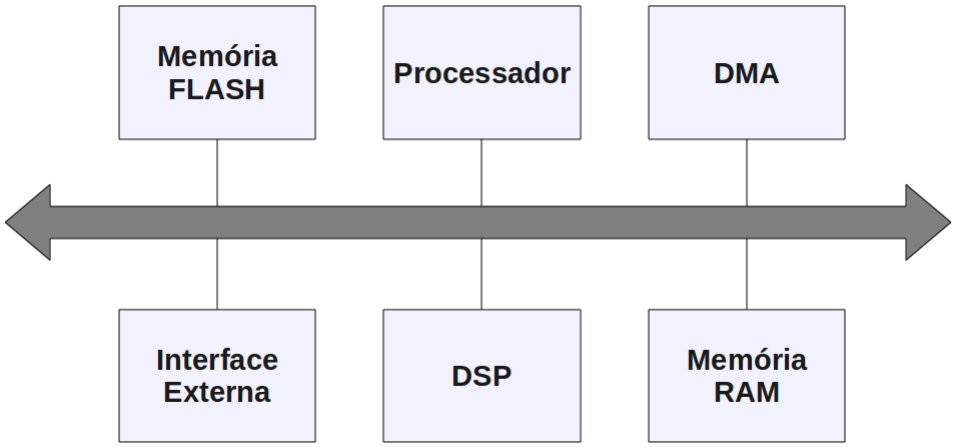


Figura 1.1: Arquitetura de SoC comum.

A integração de um sistema em um único *chip* traz vantagens tais como diminuição de área de silício, melhora de desempenho, consumo de energia reduzido, menor custo e diminuição no tempo de projeto [73].

Arquiteturas tradicionais para sistemas modernos consistem em SoCs com múltiplos processadores integrados em um único *chip* conhecidos como MPSoCs (*Multi-Processor System-on-Chip*) [31]. Um MPSoC é o resultado da combinação de microprocessadores, memórias, barramentos, arquiteturas, padrões de comunicação, protocolos, interfaces e outros IPs. Devido à sua complexidade, descrições de um SoC completo em nível de transferência entre registradores, ou RTL (em inglês, *Register Transfer Level*), são difíceis de se desenvolver e manter. Além disso, simulações realizadas com modelos descritos em linguagens RTL, apesar de fornecerem informações precisas, levam muito tempo.

A complexidade inerente a um MPSoC torna seu projeto um grande desafio. Considerações como otimização de energia, sincronização, testabilidade e verificação são cruciais. Aliado a isso, é necessário levar em consideração o *time-to-market* [72]. Essa métrica relaciona o tempo para introdução de um produto no mercado e o retorno deste investimento (em inglês, *Return On Investment* - *ROI*). É fundamental que o tempo entre o início do desenvolvimento de um produto e o momento de lançá-lo no mercado seja cada vez menor para garantir uma maior lucratividade, forçando a diminuição no tempo de projeto. Visando enfrentar essas restrições e a complexidade de projeto, a adoção de metodologias e técnicas como o projeto baseado em plataformas (PBD - *Platform Based Design*) [63] e a modelagem em nível de sistema [46, 51] favorecem o reuso e o desenvolvimento gradual de um SoC.

O projeto baseado em plataformas surgiu como uma solução para as pressões de tempo

e custo da produção de SoCs. Esta metodologia requer a existência de componentes de hardware e software re-usáveis e compatíveis com padrões estabelecidos pela plataforma. Uma *instância* de uma plataforma é o projeto de um novo sistema a partir da configuração ou parametrização da plataforma básica [25]. Essa instanciação é feita através do ajuste de parâmetros dos componentes reconfiguráveis garantindo com isso a flexibilidade nas instanciações. Dessa forma, uma plataforma é capaz de executar diferentes aplicações. A ideia básica desta metodologia é permitir a concepção de projetos que possam ser reaproveitados.

A adoção da modelagem em nível de sistema, também chamada de ESL (*Electronic System Level*) [64], provê recursos que permitem incorporar aspectos do hardware em descrições com alto nível de abstração. A modelagem em nível de sistema permite que detalhes desnecessários ao modelo sejam ocultados, tornando mais fácil descrever a funcionalidade de um módulo e suas interconexões. Desta forma, é possível obter um melhor entendimento do comportamento do módulo dentro do sistema como um todo.

O fluxo do projeto em ESL provê ferramentas e metodologia para prototipação virtual e torna a verificação e otimização do sistema mais eficiente em estágios iniciais de desenvolvimento do projeto [31]. Por ser descrito em software, o modelo em nível de sistema permite simulações mais rápidas, capazes de fornecer estimativas que podem auxiliar nas decisões de projeto e facilitar a descoberta de erros. Além disso, visto que modelos em alto nível minimizam o esforço de modelagem e podem ser otimizados quanto à velocidade de execução, eles podem também ser aplicados durante os estágios iniciais de desenvolvimento, visando a exploração de alternativas no espaço de projeto [71]. Essas características tornam a técnica bastante atraente, pois plataformas complexas demandam a execução de muitas simulações, visto que cada pequena otimização ou modificação, como a inserção de atrasos, pode introduzir erros. A utilização de plataformas arquiteturais para facilitar o reuso de módulos IPs, combinada com a modelagem e simulação em nível de sistema, tem permitido lidar com a complexidade do projeto de SoCs, além de diminuir significativamente o tempo de desenvolvimento [46].

A maioria dos IPs de um MPSoC são altamente configuráveis, cada um com uma complicada relação custo-benefício entre métricas como desempenho, área e energia, tornando o espaço de projeto de um MPSoC bastante amplo. Aliado a essa complexidade está o fato de que não é possível realizar a verificação de um SoC sem a aplicação em software e muito menos desenvolver o software sem modelos de hardware.

Com o aumento da complexidade dos SoCs, é cada vez mais difícil validar e verificar a sua corretude e desempenho usando apenas metodologias tradicionais baseadas na verificação dos módulos IPs em isolamento seguida da integração do sistema [56]. O fluxo atual de desenvolvimento muitas vezes requer a execução de *traces* de execução em modelos completos do sistema através de *testbenches* realistas. Nessas situações, é importante

que os projetistas possam começar com modelos do sistema completo nos quais subsistemas possam ser independentemente substituídos por modelos refinados, de forma a haver uma validação contínua do sistema [56].

Neste contexto, o conceito de plataforma virtual tem sido utilizado para desenvolvimento paralelo de hardware e software [23] como mostra a Figura 1.2. Uma plataforma virtual corresponde a um conjunto de modelos de hardware que são organizados para representar um sistema completo. Os modelos de hardware aceitam parâmetros para se adaptarem às características do hardware, tal como tamanho de memória, profundidade de FIFO, etc. que podem ser utilizados durante o estágio de especificação para realizar exploração em nível de arquitetura. Assim que a especificação é determinada, o projeto de hardware pode começar com os mesmos parâmetros dos modelos de hardware da plataforma virtual. Quando a plataforma virtual está pronta com os parâmetros determinados pelos projetistas de hardware, os engenheiros de software possuem o ambiente para desenvolver o software muito antes da plataforma real de hardware estar pronta. Como resultado, o tempo total de desenvolvimento é reduzido significativamente.

Figura 1.2: Metodologias utilizadas no desenvolvimento de plataformas. Adaptado de [23].

Através de plataformas virtuais, projetistas podem analisar antecipadamente muitos problemas de projeto em um SoC obtendo assim estimativas para consumo de energia, tráfego de barramento, uso de memória, eficiência dos periféricos e, principalmente, desempenho do sistema como um todo. Como resultado, a habilidade de identificar a melhor implementação do sistema dentro de um tempo aceitável é melhorada [49]. Adicionalmente, um modelo de sistema completo oferece ao projetista a possibilidade de realizar a depuração antecipada da plataforma. Para isto, é preciso ter um ambiente de simulação que forneça recursos suficientes para permitir a exploração e verificação em nível de

sistema.

Esta dissertação visa prover uma plataforma virtual em nível ESL chamada MediaBox. A plataforma servirá para a avaliação de desempenho em sistemas multimídia como também auxiliará a validação de técnicas de depuração de plataformas e o estudo de problemas relacionados a fluxo de dados em plataformas ESL. Adicionalmente, o modelo de sistema gerado pode ser utilizado para exploração do espaço de projeto e validação rápida de sistemas SoCs.

A execução da plataforma MediaBox facilita a produção de grandes quantidades de informação poupando tempo e esforço ao desenvolvedor. De posse de uma plataforma suficientemente complexa que forneça recursos para variados tipos de simulações, os desenvolvedores poderão exercitar diferentes aspectos e executar testes em um modelo abstrato de um SoC complexo.

1.1 Objetivos

Os objetivos desse trabalho são os seguintes:

- Desenvolvimento e adaptação de IPs para compor a plataforma MediaBox.
- Integração de uma infraestrutura de interconexão que dê suporte à sincronização e comunicação entre os IPs.
- Integração do software responsável pela execução da MediaBox bem como pela sincronização e comunicação entre os IPs.
- Fornecimento de informações acerca do funcionamento e do desempenho do sistema de comunicação da plataforma.

1.2 Organização

Esse trabalho possui a seguinte organização.

- O Capítulo 2 apresenta metodologias e técnicas que têm sido utilizadas no desenvolvimento de plataformas virtuais além de fornecer alguns conceitos sobre os componentes que as formam.
- O Capítulo 3 descreve a arquitetura e o funcionamento da MediaBox. Nesse capitulo também são descritos detalhadamente cada um de seus componentes e suas respectivas funções na plataforma.

- O Capítulo 4 relata os resultados obtidos com a implementação da plataforma em um estudo de caso. Adicionalmente, são apresentadas as informações obtidas acerca do funcionamento e do desempenho do sistema de comunicação da plataforma.
- O Capítulo 5 fornece uma visão geral sobre outros trabalhos realizados no contexto de construção de plataformas ESL.
- No Capítulo 6 são feitas as conclusões desse trabalho bem como os trabalhos futuros.

Capítulo 2

Projeto de Plataformas Virtuais

Neste capítulo é apresentada uma visão geral da: (a) metodologia de projeto baseado em plataformas; (b) modelagem ESL e (c) algumas ferramentas utilizadas em seu projeto. Adicionalmente, apresentamos alguns conceitos e características do projeto de plataformas virtuais.

2.1 Projeto Baseado em Plataformas

O projeto baseado em plataformas surgiu como uma solução para as pressões de tempo, complexidade e custo do projeto de SoCs [63]. Em geral, plataformas são compostas por componentes programáveis. Esta característica faz com que plataformas dependam da existência de componentes de hardware e software previamente descritos e configuráveis. Uma instância de uma plataforma é o projeto de um novo sistema a partir da configuração ou parametrização de uma plataforma básica [25]. Os componentes das plataformas podem ser reutilizados em várias outras instâncias de plataformas sendo necessário apenas que sejam configurados. Essa configuração torna os componentes compatíveis com os padrões estabelecidos pela plataforma em que estão sendo inseridos. Dessa forma, a flexibilidade e a capacidade de executar diferentes aplicações de uma mesma instância de uma plataforma são garantidas pelos componentes programáveis.

O fluxo da metodologia de projeto baseada em plataformas é apresentado na Figura 2.1. Primeiro, são selecionadas a aplicação e a plataforma básica onde ela será executada. Em seguida, é feito o mapeamento da aplicação na plataforma, através da configuração de parâmetros presentes na plataforma. Após o mapeamento, é feita uma análise visando a realização de ajustes. Essa análise é feita com a obtenção de estimativas do impacto da instância da plataforma desenvolvida para uma dada aplicação. Este ciclo é realizado até o momento em que obtemos uma plataforma que atenda aos requisitos do projeto. Em seguida, parte-se para a implementação do projeto.

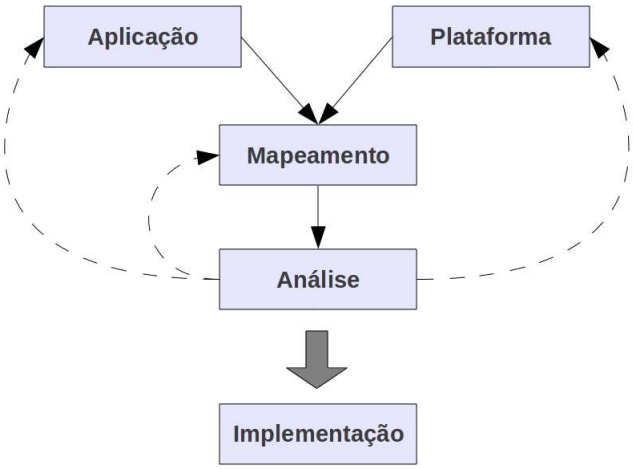


Figura 2.1: Fluxo típico do projeto baseado em plataformas.

Durante a análise, diferentes métricas incluindo tempo de execução, consumo de energia, área de silício utilizada, dentre outras, podem ser avaliadas. Tais métricas dependem dos parâmetros de configuração do sistema, tais como voltagem necessária, tamanho do componente, tecnologia utilizada, etc. Através da modificação dos parâmetros de configuração da plataforma, o projetista é confrontado com um grande número de opções de configuração. Cada uma destas opções possui várias alternativas de implementação e suas respectivas características de desempenho. Desta forma, para obter o desempenho desejado, o projetista precisa avaliar cuidadosamente o número, a localização e o nível de abstração dos componentes que serão inseridos na plataforma.

Apesar do avanço atual da indústria de semicondutores, a criação de plataformas complexas para o projeto de SoCs tem sido um desafio para os projetistas dada a sua complexidade. A solução para esse problema passa necessariamente pelo reuso de componentes pré-projetados. O reuso, além de possibilitar um menor tempo de projeto, visto que os IPs estariam prontos e previamente testados, possibilitou o surgimento de um mercado de IPs para serem utilizados por diferentes clientes [7].

Nos últimos tempos houve, portanto, um deslocamento do problema de projeto. Se antes a validação de um bloco IP e a automação do seu projeto passavam por uma descrição em RTL e posterior síntese, hoje os grandes projetos são feitos através da escolha adequada dos componentes que deverão fazer parte do SoC [26].

2.2 Modelagem ESL

A abordagem tradicional de desenvolvimento prega o uso de linguagens de descrição de hardware (tais como VHDL [2] e Verilog [1]) para obtenção do modelo inicial do sistema. O uso destas linguagens traz algumas vantagens como a possibilidade de realizar a validação do hardware desenvolvido através de dispositivos programáveis como FPGAs (*Field Programmable Gate Arrays*) [47]. No entanto, para os SoCs atuais, desenvolver em RTL está se tornando impraticável [22]. Devido ao aumento das funcionalidades e da complexidade da tecnologia de interconexão de seus módulos, descrições de um sistema completo em RTL são difíceis de se desenvolver e alterar. Além disso, simulações realizadas com modelos descritos em linguagens RTL, apesar de fornecerem informações precisas, levam muito tempo.

A diminuição do nível de detalhamento de projeto é vista como uma contribuição para o aumento da velocidade de concepção dos projetos de hardware. ESL é uma abordagem que prega o desenvolvimento de sistemas em níveis de abstração acima da descrição tradicional em RTL. A elevação dos níveis de abstração de projeto para o ESL permite que detalhes sejam ocultados, tornando mais fácil descrever a funcionalidade de um módulo e suas interconexões. Ao fim deste processo, as funcionalidades do sistema são particionadas para um conjunto de recursos computacionais de hardware e software [26].

Existem algumas técnicas de modelagem ESL [16, 41, 65]. Em geral, estas técnicas definem um conjunto de níveis de abstração e etapas. Os níveis de abstração se preocupam com funcionalidades específicas do sistema inerentes a etapa no qual ele está sendo adotado. Em geral, cada nível de abstração representa um refinamento do nível de abstração mais alto.

A OCP-IP define um padrão para desenvolvimento de modelos em nível de sistema [40]. Esse padrão define três níveis de abstração:

- TL3 (Transaction Level 3 — Nível de Transação 3): esse é o nível mais abstrato do padrão OCP. A descrição dos modelos é funcional e não há noção de tempo. Modelos desenvolvidos neste nível de abstração possuem execução orientada a eventos. Cada evento é instantâneo, ou seja, o tempo total para realização de uma requisição é nulo. Dessa maneira, podemos observar seu comportamento e realizar a validação do modelo funcional.

- TL2 (Transaction Level 2 — Nível de Transação 2): neste nível são feitas estimativas de tempo. Nos modelos desenvolvidos neste nível, são inseridas estimativas de tempo em cada uma das funcionalidades. Apesar de ser mais lento que o TL3 e da ausência de muitos detalhes, uma descrição neste nível de abstração já permite a obtenção de estimativas de desempenho, potência e área) com um grau de precisão suficiente para que sejam tomadas decisões de projeto.

- TL1 (Transaction Level 1 — Nível de Transação 1): esse é o nível mais baixo de abstração do padrão OCP. Os modelos desenvolvidos neste nível se caracterizam por possuir precisão de ciclos. Todas as funcionalidades do sistema estão associadas a ciclos de *clock*. Neste nível, o desempenho pode ser medido com maior precisão.

Dentre as principais vantagens que se busca com a elevação do nível de abstração estão a facilidade de modelagem do sistema, seu gerenciamento e principalmente a possibilidade de validação do sistema descrito logo nos primeiros estágios do projeto. Com esta abordagem, os domínios de hardware e software podem ser explorados paralelamente e validados de forma homogênea [23].

Atualmente, a modelagem ESL vem sendo utilizada como um subproduto do projeto baseado em plataformas.

2.3 SystemC

Criada pela OSCI (*Open SystemC Initiative*) [3], a linguagem SystemC foi desenvolvida com a finalidade de preencher a lacuna entre a especificação de sistemas computacionais e sua concepção.

SystemC é uma linguagem de descrição de hardware que permite o desenvolvimento de modelos em ESL. Construída como um conjunto de classes sobre a linguagem C++ [35], ela provê um núcleo de simulação que é embutido ao código C++ compilado. É através deste núcleo de simulação que podemos observar o comportamento de um componente modelado em SystemC. O fato de C++ ser uma linguagem bem aceita e comumente usada pela indústria e academia torna o uso de SystemC bastante atraente para os projetistas.

SystemC provê uma linguagem comum tanto para a descrição de modelos em hardware quanto para modelos em software. Isso facilita o processo de compreensão, descrição e validação de um projeto. Com isto, a detecção de erros funcionais e a validação do projeto podem ser realizadas nas fases iniciais de desenvolvimento.

Um dos aspectos mais importantes de SystemC é a possibilidade de modelar, em diferentes níveis de abstração, e até combinar esses diferentes níveis em um só modelo. Há também a possibilidade de modelar variáveis de tempo, comunicação e concorrência junto com o mecanismo de simulação. Modelos podem ser implementados em diferentes níveis de abstração e refinados usando a mesma linguagem para cobrir necessidades complementares como a inclusão da noção de tempo. SystemC possibilita a verificação da comunicação dos componentes do projeto antes da geração do hardware. Essa verificação é feita através da simulação dos componentes de hardware e software que fazem parte do projeto.

2.3.1 SystemC TLM

Transaction Level Modeling (TLM) [37] é um padrão de comunicação também desenvolvido pela OSCI para a linguagem SystemC. TLM foi desenvolvido com o intuito de prover uma plataforma para desenvolvimento de software em estágios iniciais de projeto, bem como permitir exploração e verificação em nível de sistema.

Um dos aspectos mais importantes do TLM é a separação da comunicação da computação em um sistema. Em TLM, componentes são modelados como módulos com um conjunto de processos concorrentes que representam seu comportamento. A comunicação entre esses módulos ocorre via transações que podem envolver os módulos diretamente ou através de um canal abstrato. Interfaces TLM são implementadas para encapsular os protocolos de comunicação. Para estabelecer comunicação, um processo simplesmente precisa acessar essas interfaces através das portas dos módulos. Essencialmente, a interface é responsável por separar a comunicação da computação em um sistema TLM [37].

Diz-se no padrão TLM que uma transação é composta de uma requisição e de uma resposta. A requisição é a informação emitida pelo módulo iniciador, enquanto que a resposta corresponde à informação fornecida de volta pelo módulo alvo. A informação trocada via uma transação depende do protocolo do meio de interconexão utilizado. Entretanto, algumas delas são geralmente comuns a todos os protocolos [16]:

- Tipo de transação: determina a direção da troca de dados. Geralmente uma leitura ou escrita.
- Endereço: é um inteiro que determina o módulo alvo assim como o endereço de memória do registrador ou componente interno.
- Dado: informação de fato enviada ao módulo alvo.
- Informações de controle: *status* de retorno da transação (erro, sucesso, etc.), duração da transação, atributos do meio de interconexão (prioridade, etc.).

A API que padroniza TLM se apoia no conceito de interfaces. As interfaces que modelam comunicação direta podem ser:

- Bloqueantes: o módulo responsável pelo início da transação permanece bloqueado até a conclusão da transação.
- Não Bloqueantes: o módulo responsável pelo início da transação fica livre para realizar outras operações.

Descrições TLM são mais fáceis de desenvolver e usar se comparadas a descrições RTL. Ao contrário do RTL, onde tudo é sincronizado em um ou mais ciclos de *clock*, o TLM não requer o uso de *clocks*. Os componentes que utilizam as interfaces são assíncronos por natureza, com a sincronização ocorrendo durante a comunicação entre os componentes. Essas abstrações permitem a criação de modelos que simulam várias ordens de magnitude mais rápido que modelos em RTL [16].

2.4 Ambientes de Projeto de Plataformas

2.4.1 ArchC

ArchC [18] é uma linguagem para descrição de arquiteturas de processadores baseada na linguagem SystemC. O principal objetivo de ArchC é prover informação suficiente aos usuários para que possam explorar e verificar uma nova arquitetura através de geração automática de ferramentas de software como por exemplo simuladores.

Apesar de ser uma linguagem simples, ArchC é capaz de descrever a arquitetura de um processador e também sua hierarquia de memória. Uma descrição de um processador em ArchC é dividida em duas partes: a descrição da arquitetura do conjunto de instruções (AC_ISA) e a descrição dos elementos da arquitetura (AC_ARCH).

Em AC_ISA o usuário fornece detalhes sobre o conjunto de instruções, tais como os nomes das instruções, formatos, tamanhos, assim como o comportamento de cada instrução e as informações necessárias para decodificá-las.

Em AC_ARCH o usuário fornece os recursos da arquitetura, tais como módulos de armazenamento, estrutura do *pipeline*, etc. Adicionalmente, a descrição do AC_ARCH contém a declaração de portas para que o processador se comunique com componentes externos, tais como barramentos ou memória. Essa interface externa do processador faz uso do protocolo TLM, apresentado na seção 2.3.1, para estabelecer a comunicação.

Baseado nestas duas descrições, ArchC pode gerar um simulador comportamental do processador em SystemC. O modelo em SystemC gerado por ArchC possui um alto nível de abstração e pode ser compilado para geração de uma especificação executável. Modelos dos processadores SPARCV8 [59], PowerPC [13], MIPS [12] e ARM [6] já foram modelados e estão disponíveis em [5].

2.4.2 ARP

ARP (em inglês, *ArchC Reference Platform*) é uma ferramenta fornecida no pacote do ArchC e disponível no mesmo site que [5]. Ela é utilizada para construir modelos de plataformas utilizando os simuladores do ArchC. Para isso, a ARP provê uma estrutura

básica composta de diretórios para os componentes mais comuns de plataformas heterogêneas e *scripts* que auxiliam na compilação e simulação das plataformas. A estrutura de diretórios da ARP pode ser vista na Figura 2.2.

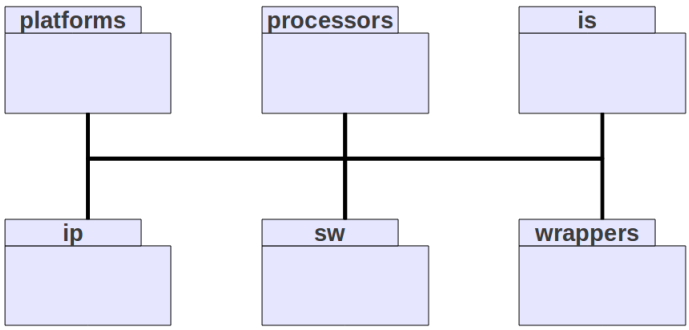


Figura 2.2: Estrutura da ARP.

Na Figura 2.2 temos:

- *platforms*: onde são instanciadas plataformas;
- *processor*: onde ficam os processadores da plataforma;
- *is*: onde ficam as estruturas de interconexão da plataforma;
- *ip*: onde ficam os IPs adicionados a plataforma;
- *sw*: onde ficam as aplicações a serem executadas na plataforma;
- *wrappers*: onde ficam tradutores entre diferentes protocolos utilizados pelos processadores, barramentos e IPs.

Essa estrutura de diretórios permite uma melhor organização dos componentes do sistema, isolando-os de forma a facilitar seu reuso. A listagem dos componentes que a plataforma requer é feita em um arquivo chamado `defs.arp` que fica dentro do diretório das plataformas. O código abaixo demonstra a estrutura do arquivo. Os itens que compõem cada parte da arquitetura da plataforma são indicados. Listas em branco, como por exemplo a `IP`, podem ser utilizadas também para especificar a ausência de um componente.

```
IP :=
IS := simple_bus_ua
PROCESSOR := mips1
```

```
SW := fft_shared
WRAPPER := ac.simple_bus_ua
```

Cada diretório de cada componente contém ainda um arquivo *Makefile* que contém instruções de compilação específicas com a finalidade de criar uma biblioteca do componente. O diretório principal da ARP contém também um *Makefile* que executa os comandos de compilação apenas nos componentes necessários da plataforma de acordo com a definição em `defs.arp`. Ou seja, após a definição da plataforma, a ARP provê o arcabouço para a compilação e simulação da plataforma através de *scripts* que automatizam essas tarefas.

2.5 Network on Chip (NoC)

Apesar do uso de MPSoCs melhorar significativamente o processamento e a versatilidade dos sistemas atuais, seu projeto apresenta uma grande dificuldade: a eficácia do mecanismo de interconexão entre seus componentes internos, à medida que o número de componentes aumenta devido aos avanços tecnológicos [17]. O grande número de IPs em um SoC e o uso ineficiente dos padrões de interconexão tornam a reutilização de módulos uma tarefa complexa. Além de permitir altas taxas de comunicação, deseja-se que os mecanismos de comunicação sejam também reusáveis [76]. Portanto, os mecanismos de comunicação precisam ser escaláveis, oferecer paralelismo de comunicação entre os vários IPs e, ao mesmo tempo, serem reusáveis diminuindo com isto o custo de projeto do SoC.

A comunicação entre IPs de um SoC costuma ser realizada seguindo duas abordagens distintas: ponto-a-ponto ou barramento. Comunicação ponto-a-ponto é ótima em termos de disponibilidade de banda, latência e uso de energia visto que os canais dedicados são projetados especialmente para esse propósito. Além disso, eles são simples de projetar e verificar. Entretanto, o comprimento da interconexão cresce à medida que o número de IPs aumenta, surgindo assim problemas de área e de roteamento. Adicionalmente, essa abordagem não é vista como eficiente já que dificulta o reuso de IPs devido ao dimensionamento de cada conexão necessário a cada nova versão do sistema.

Meios de comunicação compartilhados como barramentos são reusáveis porém tornam-se ineficientes à medida que cresce o número de IPs em um SoC [76]. Apesar da escalabilidade obtida com o uso de barramentos, eles ainda apresentam alguns problemas como [26]:

- Permitem apenas uma transação por ciclo, serializando completamente a comunicação.
- Todos os IPs do sistema são conectados ao mesmo barramento que deve passar por todo o circuito integrado. Consequentemente, a capacitância de carga é elevada.

- Apresentam um maior consumo de energia devido a elevada capacitância.
- Diminuição da frequência disponível para comunicação devido ao uso compartilhado do canal de comunicação por um número crescente de IPs.

Alguns destes problemas podem ser resolvidos pelo uso de barramentos hierárquicos como o Core Connect [42]. Porém, esta abordagem também apresenta problemas devido à possibilidade de comunicação bloqueante atingindo todo o barramento e à diferença entre as frequências de operação e largura de banda entre cada barramento [54].

O uso de barramentos em SoCs se tornou o gargalo da comunicação em um *chip* e, para lidar com esse problema, uma nova micro-arquitetura chamada NoC (*Network-on-Chip*) foi proposta [20, 45] de forma a atender às necessidades dos projetos atuais. NoC introduz o conceito de rede para o projeto de arquitetura de comunicação em um *chip* e possibilita a comunicação simultânea entre diversos pares de IPs. Atualmente NoCs são amplamente pesquisadas em aspectos como topologia de rede, estrutura de nós e algoritmo de roteamento. Além disso, atraso, taxa de injeção e vazão da rede são frequentemente usados para descrever o desempenho de uma NoC [28].

2.5.1 Fundamentos de NoCs

NoCs são estruturas de interconexão de IPs em SoCs, assim como barramentos, porém mais complexos que estes, visto que fazem uso de conceitos mais elaborados, vistos a seguir.

Uma NoC é composta por um conjunto de canais e de roteadores, estes últimos podendo serem vistos como núcleos dedicados à comunicação. Cada roteador está conectado a um ou mais IPs e possui um conjunto de canais para se comunicar com outros roteadores. Uma mensagem transita entre IPs ao fazer vários saltos (em inglês, *hops*) pelos canais e roteadores compartilhados entre o IP de origem e o IP de destino. Como cada roteador possui um tempo diferente de zero para processar a mensagem e decidir para onde encaminhá-la, quanto mais longe um IP estiver de outro, maior o tempo de comunicação envolvido. Este aparente prejuízo é amplamente compensado por outros fatores [26]:

- A NoC pode transmitir uma mensagem numa frequência maior, pois a capacitância resultante das conexões é menor que a de um barramento. A conexão entre os roteadores é ponto-a-ponto e por isso, a inclusão de mais IPs não degrada a frequência de transmissão da informação no fio.
- Diferentemente de um barramento, em uma NoC mesmo que um roteador esteja ocupado, outros roteadores podem transmitir várias mensagens em paralelo.

- Uma NoC é facilmente escalável bastando apenas adicionar mais roteadores.

Todas as NoCs possuem três componentes fundamentais que são: os adaptadores de rede, os roteadores e os canais [17]. O adaptador de rede implementa a interface através da qual IPs conectam-se à NoC. Sua função é separar a computação (nos IPs) da comunicação (na rede). Ele é responsável pelo encapsulamento das mensagens ou transações geradas pelos IPs e encaminhadas à rede. A mensagem enviada pelo roteador consiste em um conjunto coerente de informações a ser transmitido entre uma origem e um destino. Um pacote é a unidade de informação empregada pelo meio de comunicação para transmitir mensagens.

Roteadores direcionam os dados de acordo com protocolos específicos e implementam uma estratégia de roteamento. Já os canais conectam os nós, provendo largura de banda. Nesse nível, os pacotes podem ser subdivididos em um conjunto de pequenas unidades de informação chamadas *flits* (em inglês, *flow control units*). Um *flit* é a menor quantidade de informação reconhecida pelo método de controle de fluxo. Alguns pesquisadores trabalham ainda com outra subdivisão chamada de phit (em inglês, *physical unit*), que corresponde ao tamanho mínimo do dado que pode ser transmitido em um canal por transação [21].

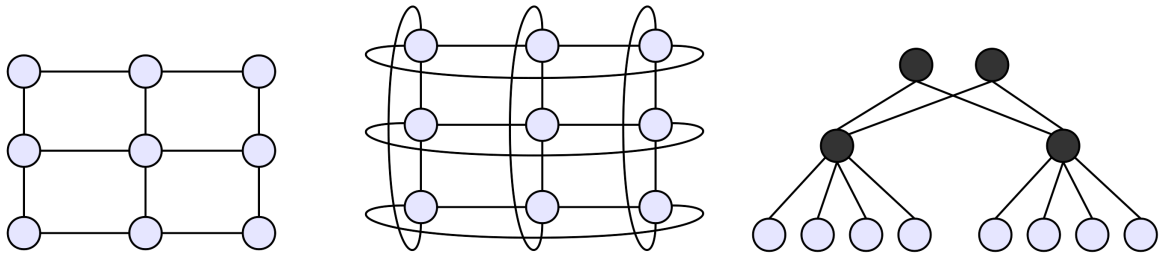
Dependendo da estrutura de pacote utilizada, um *flit* pode ser de cabeçalho (em inglês *header*), de corpo (em inglês *payload*) ou de cauda (em inglês *trailer*). Em geral, o cabeçalho contém informação sobre o IP de destino, o corpo da mensagem contém dados úteis ao IP de destino e a cauda pode conter dados de correção de erros e bits especiais de fim de pacote. Um flit de cabeçalho é o primeiro flit de um pacote e é seguido de zero ou mais flits de corpo e um flit de cauda. Flits do corpo e da cauda não possuem informação de roteamento ou ordem e por isso devem seguir o flit de cabeçalho por sua rota e permanecer em ordem [33].

Uma NoC é uma rede de interconexão e adapta conceitos originalmente surgidos nas áreas de redes de computadores. Como tal, ela pode ser descrita por sua topologia, estratégia de roteamento e controle de fluxo. Esses conceitos serão explicados nas próximas sessões.

2.5.2 Topologia

A eficiência de redes de interconexão vem do compartilhamento dos recursos de comunicação. Ao invés de criar canais dedicados entre pares de IPs, uma rede de interconexão é implementada como uma coleção de nós de roteadores conectados por canais compartilhados. O padrão de conexão desses nós define a topologia da rede. A topologia de uma NoC pode ser representada através de grafos. Nós podem ser considerados os vértices

de um grafo e os canais que os ligam as arestas. Exemplos de topologias de NoCs são: malha, torus e árvore gorda, representadas na Figura 2.3.



(a) Malha (b) Torus (c) Árvore Gorda

Figura 2.3: Diferentes topologias de uma NoC.

2.5.3 Roteamento

Quando uma topologia é escolhida, podem existir muitos caminhos possíveis (sequências de nós e canais) que uma mensagem pode tomar através da rede para alcançar seu destino. O roteamento determina por qual desses possíveis caminhos uma mensagem deve seguir. Uma boa escolha de caminho minimiza seu tamanho (normalmente medido como o número de nós visitados) e consequentemente a latência da mensagem, enquanto balanceia a demanda por recursos compartilhados da rede [33].

Algoritmos de roteamento podem ser classificados de acordo com diversos critérios. Uma dessas classificações está relacionada à adaptatividade do algoritmo e estabelece como esses selecionam entre as possíveis rotas entre um nó de origem `X` e um nó de destino `Y` [33]:

Determinístico

Algoritmos de roteamento determinísticos sempre escolhem o mesmo caminho entre `X` e `Y`, mesmo quando existem múltiplos caminhos. Esses algoritmos ignoram a diversidade de caminhos provida pela topologia utilizada e por isso deixam a desejar quanto ao balanceamento de cargas. Apesar disso, eles são muito comuns na prática porque são fáceis de implementar e de inserir mecanismos de prevenção de *deadlock*.

Adaptativo

Algoritmos adaptativos adaptam-se ao estado da rede. Eles baseiam suas decisões de roteamentos entre `X` e `Y` em fatores da rede. Esses fatores podem incluir o *status* do nó ou da conexão, o tamanho das filas dos recursos da rede, o histórico de carga do canal e congestionamento.

Adicionalmente, algoritmos de roteamento têm influência direta em algumas propriedades da rede de interconexão e por isso devem ser projetados de forma a evitar a ocorrência de *deadlock* e *livelock*.

Deadlock

Deadlock é uma situação que ocorre quando pacotes em um ciclo ficam esperando a liberação de recursos (por exemplo, um canal físico ou virtual ou um buffer de pacotes compartilhado) entre si e por isso ficam bloqueados eternamente [33].

Deadlocks em NoCs podem ser categorizados em duas classes: de roteamento (do inglês, *routing-dependent*) e de mensagem (*message-dependent*) [17]. *Deadlocks* de roteamento ocorrem quando há uma dependência cíclica de recursos criada por pacotes nos vários caminhos da rede. *Deadlocks* de mensagem ocorrem quando, em algum ponto da rede, interações e dependências são criadas entre diferentes tipos de mensagem (por exemplo, requisições e respostas) ao compartilhar recursos na rede. Mesmo quando a rede em si é projetada para ser livre de *deadlocks* de roteamento, *deadlocks* de mensagens podem bloquear a rede definitivamente, afetando portanto o operação correta do sistema.

Para prevenir essa situação, NoCs devem ou usar mecanismos de prevenção de *deadlock* (métodos que garantam que a rede não entrará em *deadlock*) ou de recuperação de *deadlock* (na qual o *deadlock* é detectado e corrigido). Quase todas as redes modernas utilizam a prevenção de *deadlock*, normalmente ao impor uma ordem aos recursos em questão e insistindo que os pacotes adquiram esses recursos em ordem [33].

Livelock

Livelock é definido como uma situação em que uma informação transmitida jamais atinge seu destino, devido ao fato desta percorrer caminhos cíclicos e o ciclo não incluir o destino da informação. Este problema está normalmente associado à utilização de algoritmos de roteamento não mínimos (permitem que pacotes percorram caminhos não mínimos pela rede) [54].

2.5.4 Controle de Fluxo

O controle de fluxo trata da alocação de canais e buffers para que uma mensagem percorra o caminho necessário da origem até o destino. Essa influência do controle de fluxo se torna crítica à medida que a utilização de recursos aumenta. Um bom controle de fluxo encaminha pacotes com baixa latência e evita a inatividade de recursos sob altas cargas [33].

As diferentes abordagens de controle de fluxo se diferenciam quanto à granularidade na qual os recursos da rede são reservados (canais ou buffers) e às condições que devem ser satisfeitas para que um pacote avance para o próximo nó. Quanto à granularidade de recursos, os principais tipos de transporte são [70]:

- *Circuit switching* (chaveamento de circuito): é aquele onde previamente ao envio de mensagens, deve ser estabelecido um caminho do nó de origem até o nó de destino, denominado circuito, e logo após são enviadas todas as mensagens. Esse mecanismo pode ser ineficiente no sentido de desperdício de banda do canal para evitar utilizar espaço de armazenamento que é relativamente mais barato [33].
- *Packet switching* (chaveamento de pacotes): é aquele onde a transferência dos dados entre a origem e o destino é realizada a partir de uma estrutura padronizada, denominada pacote. Neste mecanismo, o caminho a ser percorrido pelo pacote é definido em tempo de roteamento, podendo ser armazenado parcialmente em memória durante o caminho.

Pode-se fazer melhor uso dos recursos de uma rede ao se armazenar pacotes (parte deles ou por completo) em cada nó. Esse armazenamento é feito em buffers localizados nos roteadores da rede. Na maioria das arquiteturas de NoC, buffers são responsáveis pela maior parte da área do roteador. Por isso, é uma grande preocupação minimizar a quantidade de armazenamento necessária diante dos requisitos de desempenho [21]. Quanto à satisfação de condições para o avanço de um pacote para o próximo nó, destacam-se três tipos de armazenamento ilustrados na Figura 2.4:

Store-and-Forward

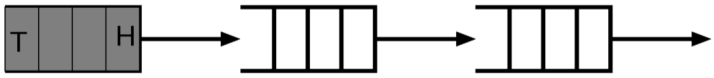
Cada nó armazena o pacote completo antes de encaminhá-lo. Assim, o envio do pacote pode parar se um roteador no caminho até o destino não tiver espaço de armazenamento suficiente. A maior desvantagem no armazenamento *store-and-forward* é o potencial aumento da latência. Visto que o pacote é recebido completamente em um nó antes de avançar para o próximo, a cada nó tem-se uma serialização da latência [33].

Virtual Cut-Through

Antes de encaminhar o pacote, o nó garante que o próximo nó do caminho pode aceitá-lo por completo. Assim, se o pacote parar, ele pode permanecer no nó atual sem bloquear outras conexões. Esse método possui a desvantagem de, ao alocar buffers na unidade de pacotes, poder fazer uso ineficiente do espaço de armazenamento. [33].

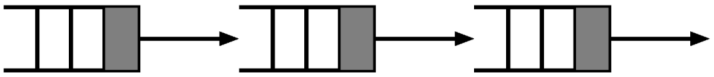
Wormhole

O roteador verifica o flit de cabeçalho do pacote para determinar seu próximo nó e o encaminha imediatamente. Os flits subsequentes são encaminhados à medida que chegam. Isso faz com que o pacote percorra seu caminho pela rede, possivelmente distribuindo-se por vários nós. Assim, a latência no roteador não é a do pacote inteiro. Entretanto, um pacote parado pode ter o efeito colateral de ocupar as conexões pelas quais o pacote está espalhado. Comparado ao *virtual cut-through*, esse método torna mais eficiente o uso do espaço de armazenamento visto que aloca canais e buffers para flits ao invés de pacotes [33].



(a) Store-and-Forward: Avança somente quando todo o pacote está armazenado no nó local e há espaço no buffer do próximo nó para armazenar todo o pacote.

(b) Virtual Cut-Through: Avança somente quando há espaço no buffer do próximo nó para armazenar todo o pacote.



(c) Wormhole: Avança se houver espaço para armazenar o próximo flit do pacote no buffer do próximo nó.

Figura 2.4: Diferentes estratégias de armazenamento de pacotes no transporte via pacotes.

O esquema que geralmente prevalece nos projetos de NoCs é o Wormhole. Suas vantagens são a baixa latência e a economia em área obtida com as custosas filas de armazenamento [21].

Capítulo 3

MediaBox

Neste capítulo apresentamos a arquitetura e o funcionamento da MediaBox. Descrevemos detalhadamente cada um de seus componentes e suas respectivas funções na plataforma, assim como as aplicações em execução. Adicionalmente, as ferramentas e o ambiente utilizados no desenvolvimento da MediaBox são apresentados.

3.1 Arquitetura da Plataforma

A plataforma MediaBox é composta pelos seguintes IPs interconectados através de uma NoC:

- Um processador PowerPC: executa o software na plataforma.
- Três memórias: armazenam dados da plataforma.
- Decodificador MP3: decodifica áudio em dados no formato `PCM`.
- Decodificador MPEG-2: decodifica vídeo.
- *Player* de Áudio AUDIO-OUT: toca os dados de áudio fornecidos pelo decodificador MP3.
- *Player* de Vídeo VIDEO-OUT: exibe o vídeo a partir dos *frames* obtidos do decodificador MPEG-2.
- USB: armazena os dados originais de áudio e vídeo.

Com exceção dos modelos de processadores e do USB, todos os outros IPs foram desenvolvidos especificamente para uso na MediaBox. Adicionalmente, tanto o USB quanto

os *Players* de Áudio e Vídeo possuem em sua interface um DMA responsável pela transferência dos dados entre eles e outros IPs da plataforma. Já a NoC utilizada consiste em um interconector 3x3 que conecta cada um dos IPs descritos. Além disso, os nós do Processador PowerPC, do Decodificador MP3 e do Decodificador MPEG-2 possuem uma memória interna utilizada na execução de suas respectivas tarefas. A arquitetura da plataforma MediaBox pode ser vista na Figura 3.1.

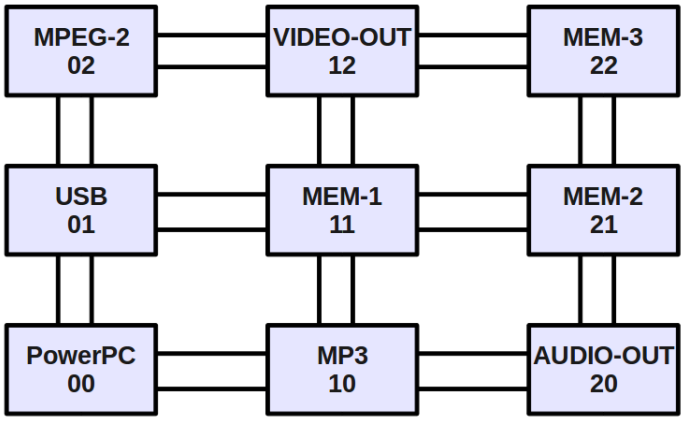


Figura 3.1: Arquitetura da plataforma MediaBox.

É importante ressaltar que os IPs utilizados na plataforma estão em diferentes níveis de abstração, de acordo com o padrão OCP-IP. O processador PowerPC, os Decodificadores MPEG-2 e MP3, os *Players* AUDIO-OUT e VIDEO-OUT, as Memórias e as interfaces DMA estão descritos no nível TL3. Já a NoC está descrita no nível TL2 e o USB está descrito no nível TL1.

3.2 Aplicações

Atualmente, os SoCs mais utilizados estão em TVs, telefones celulares e MP3 *players* e executam aplicações multimídia ou de telecomunicações. Pensando nisso, a plataforma MediaBox foi construída com o intuito de executar aplicações multimídia. Essas aplicações exibem um comportamento dinâmico e seus custos de execução (tal como o número de ciclos e consumo de energia) dependem dos dados de entrada. Além disso, essas aplicações são normalmente implementadas através de um laço principal, chamado de laço de interesse, que é executado várias vezes, lendo, processando e escrevendo objetos do fluxo de execução da aplicação. Um objeto de fluxo pode ser desde um bit pertencente ao fluxo de bits comprimido do clipe de um vídeo codificado, até um *frame* de vídeo ou uma amostra de áudio [38].

A parte de leitura do laço de interesse recebe um objeto do fluxo da entrada e o separa em duas partes: cabeçalho e dados. A parte de processamento consiste em diversas operações e transformações sobre esses dados de entrada. A parte de escrita do laço envia os dados processados para os dispositivos de saída, tal como a tela ou alto-falantes, e salva o estado interno da aplicação para um próximo uso. Em um decodificador de vídeo por exemplo, o *frame* decodificado no laço anterior pode ser necessário para decodificar o *frame* atual. As ações executadas nesse laço de interesse formam o modo de operação interno da aplicação [38].

Nesse contexto, foram escolhidas duas aplicações para executar na MediaBox: um decodificador de áudio MP3 e um decodificador de vídeo MPEG-2. O decodificador de áudio escolhido corresponde a um decodificador ISO MP3 [43] disponível online como software livre. Já o decodificador de vídeo corresponde a uma versão do decodificador MPEG-2 [55] também disponível na Internet. Ambas as aplicações utilizam a linguagem C e foram modificadas de forma a adaptar suas execuções aos modelos de processadores e aos dispositivos de entrada (USB) e de saída (*Players*) correspondentes.

A execução da decodificação de áudio corresponde a um típico *pipeline* MP3 que tem como entrada dados no formato `mp3` e como saída dados no formato `PCM`. A decodificação de vídeo por sua vez tem como entrada dados de vídeo no formato `m2v`, e como saída *frames* no formato `ppm`. O funcionamento da plataforma consiste na execução das seguintes tarefas:

1. O PowerPC e o Decodificador MPEG-2 solicitam a transferência de dados de áudio/vídeo do USB para sua memória interna.

2. O PowerPC realiza o pré-processamento dos dados de áudio em memória e os encaminha para o respectivo decodificador. O Decodificador MPEG-2 por sua vez realiza o pré-processamento dos dados e inicia a decodificação do vídeo.

3. A cada *frame* decodificado, os decodificadores MP3 e MPEG-2 solicitam a transferência dos dados de saída de áudio e vídeo para os respectivos dispositivos de saída representados pelos *Players* AUDIO-OUT e VIDEO-OUT.

4. O processo se repete até que todos os dados tenham sido decodificados.

5. Quando, e se solicitado pelo usuário, os *Players* executam o áudio/vídeo.

Como se pode observar pela descrição das tarefas executadas, o fluxo de execução da plataforma pode ser dividido em um fluxo de áudio e outro de vídeo que funcionam de maneira similar. O fluxo de áudio pode ser observado no diagrama de sequência da Figura 3.2, enquanto que o de vídeo está descrito no diagrama da Figura 3.3. Vale ressaltar que os

dados são lidos a partir do USB de forma a preencher o buffer de entrada das respectivas aplicações. Cada aplicação possui seu respectivo tamanho de buffer de entrada bem como quantidade de dados processados por *frame*. Assim, a leitura de novos dados no USB acontece à medida que o processamento dos dados contidos no buffer chega ao fim e não necessariamente a cada *frame*.

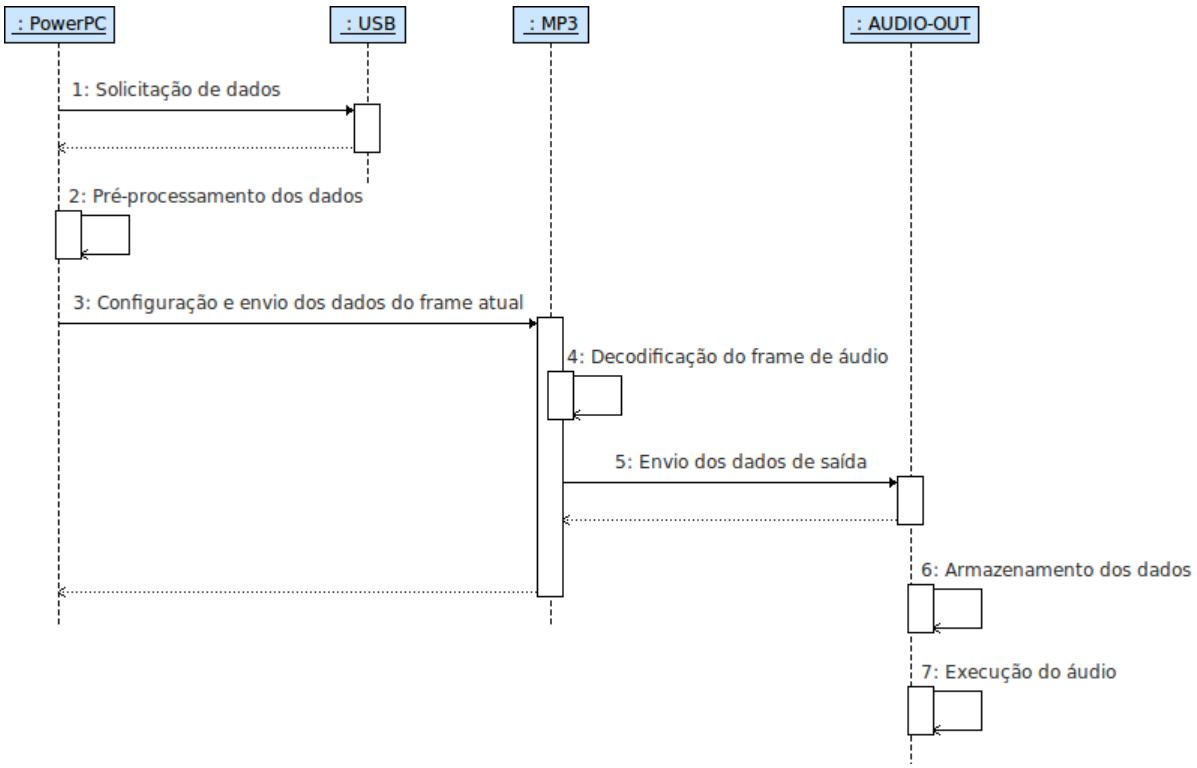


Figura 3.2: Diagrama de sequência da aplicação de áudio na plataforma MediaBox.

É importante ressaltar também que a fase de pré-processamento presente em ambas as aplicações se caracteriza pela interpretação do formato do arquivo de entrada (no caso `mp3` e `m2v`). Durante essa fase, não só o cabeçalho do arquivo como também o cabeçalho do *frame* são interpretados e as informações necessárias para a decodificação são extraídas. Adicionalmente, o armazenamento dos dados de saída nos *Players* variam para cada aplicação. No AUDIO-OUT os dados `PCM` são armazenados e, somente no fim da decodificação de todo o arquivo de entrada, é feita a escrita dos dados no arquivo de saída. Já no VIDEO-OUT cada novo *frame* gerado é salvo assim que sua decodificação chega ao fim.

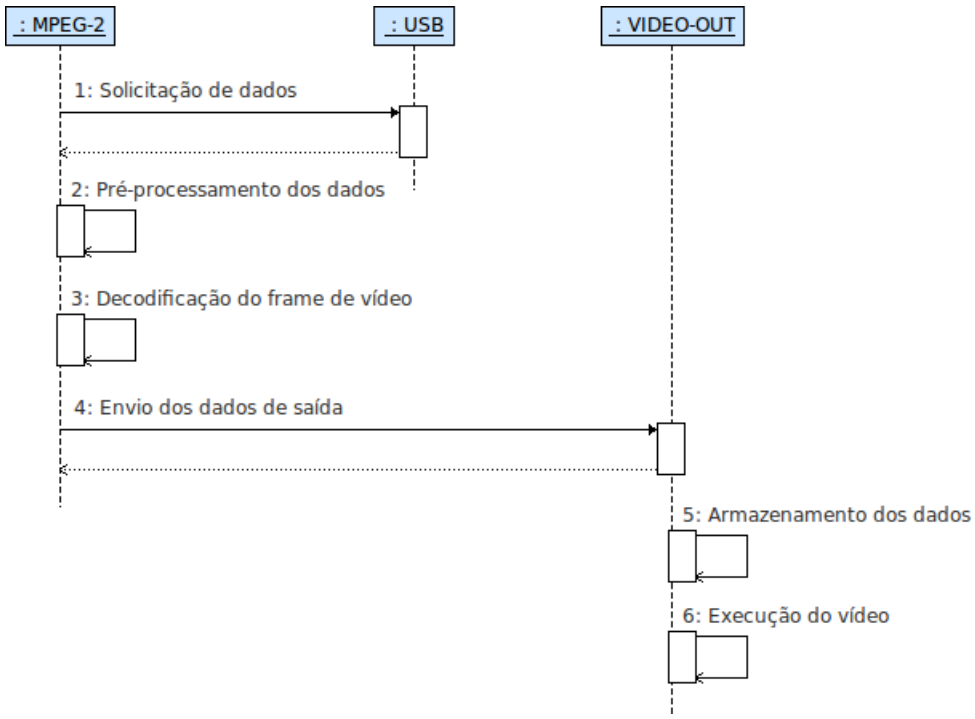


Figura 3.3: Diagrama de sequência da aplicação de vídeo na plataforma MediaBox.

3.3 Parametrização e Endereçamento

Pensando em prover maior facilidade no uso das possíveis configurações da MediaBox, é fornecido um arquivo de configuração que permite explorar essas opções. Nesse arquivo, o usuário pode parametrizar a plataforma quanto ao uso do USB, dos *Players* de Áudio e Vídeo e quanto à execução do áudio/vídeo no fim da decodificação. Essa parametrização é feita configurando os valores dos registradores na Tabela 3.1 como "yes"ou "no".

Adicionalmente, no arquivo de configuração estão as faixas de endereço utilizadas pelos IPs e os endereços de seus registradores. Utilizou-se como tamanho padrão 5MB para os processadores e 1MB para os demais IPs. Essas informações podem ser modificadas de acordo com a utilização da plataforma e a necessidade do usuário. O arquivo de configuração da MediaBox pode ser visto no Apêndice na seção A.1.

Já o mapa de endereçamento da MediaBox pode ser observado na Tabela 3.2. Esse mapa de memória corresponde a um modelo híbrido global + local, ou seja, nele estão representados tanto as faixas de endereço internas aos IPs (PowerPC e MPEG-2) como também as faixas de endereçamento globais para comunicação entre os IPs (demais componentes). Como o PowerPC e o MPEG-2 nunca se comunicam na execução das aplicações, ambos possuem a mesma faixa de endereçamento. O mesmo acontece com

Tabela 3.1: Parâmetros de configuração do uso do USB e dos *Players* de Áudio e Vídeo.

Nome do registrador	**Função**
MB_USB_MP3	Uso ou não do USB como dispositivo de entrada na decodificação do MP3.
MB_USB_MPEG2	Uso ou não do USB como dispositivo de entrada na decodificação do MPEG-2.
MB_PLAYER_MP3	Uso do *Player* como dispositivo de saída na decodificação do MP3.
MB_PLAYER_MPEG2	Uso do *Player* como dispositivo de saída na decodificação do MPEG-2.
MB_AUDIO_ENABLED	Execução do áudio no fim da decodificação.
MB_VIDEO_ENABLED	Execução do vídeo no fim da decodificação.

os *Players* visto que eles só são acessados pelos respectivos decodificadores de áudio e vídeo. O USB possui uma faixa de 1MB de endereçamento que é dividida em duas de 0,5MB que correspondem aos canais de áudio e vídeo do DMA. Por fim, a faixa final de endereçamento é reservada para utilização das memórias.

Tabela 3.2: Mapa de endereçamento da plataforma MediaBox.

IP/Endereço	PowerPC	MPEG-2	MP3	Players	USB	MEM-1	MEM-2	MEM-3
0x000000								
0x500000								
0x600000								
0x700000								
0x800000								
0x830000								
0x860000								

A configuração dos arquivos de áudio e vídeo a serem decodificados é feita também em arquivo. Entretanto, como é necessário fazer a cópia desse arquivo de origem para a pasta tanto da plataforma quanto do USB durante a compilação, essa configuração é feita no arquivo Makefile principal. Ou seja, nesse arquivo principal, é definida a plataforma que será executada, o local onde estão os arquivos de áudio e vídeo e qual arquivo será utilizado na execução da plataforma.

3.4 Características e Funcionamento dos IPs

3.4.1 MP3

O decodificador MP3 consiste em um software-IP composto por um processador MIPS e uma memória. Sua estrutura pode ser vista na Figura 3.4. A memória desse nó não só armazena o código-fonte do processador, como também funciona como veículo de comunicação entre o MP3 e o PowerPC. Para isso, foi criada uma área de memória compartilhada fixa onde o processador PowerPC escreve dados necessários para a comunicação e a decodificação do MP3.

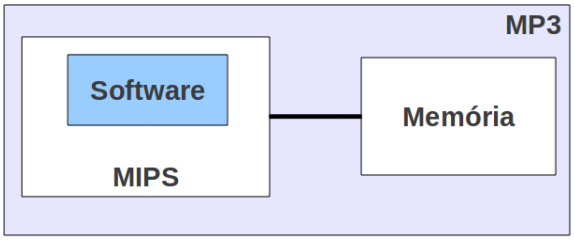


Figura 3.4: Estrutura interna do software-IP MP3.

O MP3 funciona tanto como mestre quanto como escravo em relação aos outros IPs. No início da execução da plataforma, ele fica em modo de espera verificando a mudança do registrador de controle, que indica a existência de novos dados de áudio para serem decodificados. Quando o PowerPC modifica o valor desse registrador na área de memória compartilhada pelos dois IPs, o MP3 faz a leitura dos dados do áudio e de configuração e executa a decodificação. Após a transferência dos dados de saída para o AUDIO-OUT, o MP3 copia para a área de memória compartilhada os dados necessários ao pré-processamento do próximo *frame* e sinaliza ao PowerPC o término da decodificação do *frame* modificando o valor desse registrador de controle. Para o PowerPC, o término do processamento do *frame* atual implica a leitura dos dados disponibilizados na área de memória compartilhada e a repetição de todo o processo explicado na Seção 3.3.

O MP3 possui uma memória interna de 5MB. Tanto a faixa de 0MB a 2MB quanto a de 3MB a 5MB da memória são reservadas para uso interno do processador. A faixa de 2MB a 3MB corresponde à área de memória compartilhada explicada anteriormente. Os registradores de configuração e de dados acessados pelo PowerPC são explicados detalhadamente na Seção A.2 do Apêndice.

3.4.2 MPEG-2

O MPEG-2 consiste em um software-IP composto por um processador PowerPC e uma memória. Sua estrutura pode ser vista na Figura 3.5.

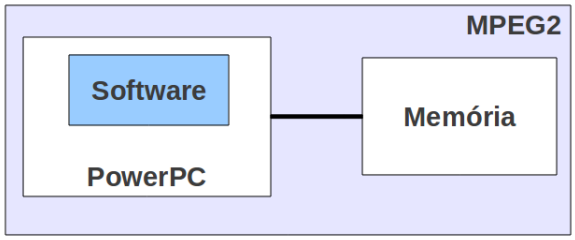


Figura 3.5: Estrutura interna do software-IP MPEG-2.

Como explicado anteriormente, a função do MPEG-2 é decodificar vídeo no formato `m2v` para *frames* isolados no formato `ppm`. Similarmente ao MP3, o MPEG-2 funciona tanto como mestre quanto como escravo. Modificamos o software original de forma a adaptá-lo ao uso de IPs externos para leitura e escrita dos dados. Consequentemente, a decodificação do vídeo é feita a partir de diversas leituras ao USB e escritas no VIDEO-OUT.

3.4.3 Players de Áudio e Vídeo

Os dispositivos de saída da MediaBox correspondem a *players* de áudio e vídeo. No arquivo de configuração no Apêndice na Seção A.1 eles são representados pelos IPs AUDIO-OUT e VIDEO-OUT respectivamente. Como mencionado anteriormente, os dois IPs possuem em sua interface um DMA responsável pela transferência dos dados a partir do IP original (MP3 ou MPEG-2). Sua estrutura pode ser vista na Figura 3.6.

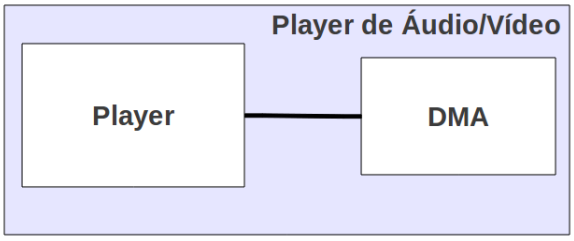


Figura 3.6: Estrutura interna dos *players* de áudio e vídeo.

No arquivo de configuração podemos observar que os dois IPs possuem alguns registradores necessários para execução do áudio/vídeo. Esses registradores são descritos nas

Tabelas 3.3 e 3.4 para o áudio e para o vídeo respectivamente.

Tabela 3.3: Registradores do *Player* de Áudio.

Nome do registrador	**Função**	**Endereço**
MB_DMA_CH00_xx	Configuração do DMA.	0x0 a 0x30
MB_AUDIO_OUT_CHANNELS_INFO	Quantidade de canais dos dados de áudio.	0x38
MB_AUDIO_OUT_DATA_RATE_INFO	Frequência dos dados de áudio.	0x3C
MB_AUDIO_OUT_CONTROL_INFO	Controle da execução do áudio.	0x40

Tabela 3.4: Registradores do *Player* de Vídeo.

Nome do registrador	**Função**	**Endereço**
MB_DMA_CH01_xx	Configuração do DMA.	0x0 a 0x30
MB_VIDEO_OUT_DATA_RATE_INFO	Quantidade de *frames* por segundo.	0x38
MB_VIDEO_OUT_CONTROL_INFO	Controle de execução do vídeo.	0x3C

Com o intuito de facilitar a utilização dos *Players*, ambos os IPs foram construídos com base em bibliotecas independentes e disponíveis na Internet. A execução do vídeo no *player* de vídeo utiliza as bibliotecas de vídeo do Fast Light ToolKit (FLTK) [8] versão 1.3.0 enquanto que a execução do áudio no *player* de áudio utiliza as biblioteca de áudio do Advanced Linux Sound Architecture (ALSA) [4] versão 1.0.23. Para executar o áudio/vídeo no final da decodificação os parâmetros de configuração correspondentes discutidos na Seção 3.3 precisam estar devidamente configurados. Além disso, as bibliotecas FLTK e ALSA utilizadas nos *Players* precisam estar devidamente instaladas.

3.4.4 USB

O USB consiste em um IP composto por uma interface DMA, um conector USB-Connector e um USB-Function que permite o acesso às funcionalidades do IP. No caso da MediaBox, o IP USB funciona como um dispositivo de armazenamento. Sua estrutura pode ser vista na Figura 3.7.

O USB-Connector corresponde à versão 1.1 do padrão *Universal Serial Bus* [32] e foi obtido na Internet a partir do site *OpenCores* [15] e modificado para uso na plataforma. Visto que o USB funciona como armazenamento tanto para o fluxo de áudio quanto para

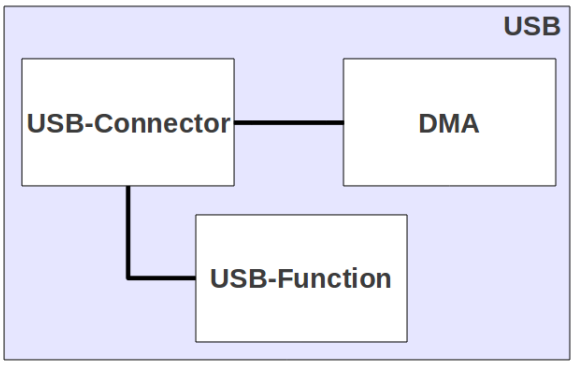


Figura 3.7: Estrutura interna do IP USB.

o de vídeo foi necessário duplicar os canais de DMA e USB. Logo, temos dois canais dedicados internos ao IP, um do DMA e um do USB, para cada um dos fluxos. No arquivo de configuração podemos observar que o USB possui uma faixa de memória de 1MB. Desses 1MB, os primeiros 500K correspondem à faixa de endereços do canal dedicado ao áudio e os 500K restantes à faixa de endereços do canal dedicado ao vídeo. Assim como os *Players*, o USB possui uma interface DMA que corresponde, em cada canal, à faixa de endereço 0x0 a 0x30.

O componente USB-Function foi criado com o intuito de fazer a interface entre o conector e o arquivo de entrada. Esse componente é também dedicado à execução da plataforma e sua função está ligada à leitura dos dados de áudio e vídeo de acordo com as especificações da aplicação correspondente. Isto é, o USB-Function foi implementado de acordo com as particularidades de leitura de cada aplicação. A utilização do USB por outras aplicações requer a instanciação de um novo USB-Function com as particularidades da aplicação utilizada.

3.4.5 DMA

Um DMA (*Direct Memory Access* — Acesso Direto à Memória) é um dispositivo que permite a transferência de dados sem a necessidade de utilização do processador durante a transferência. Como explicado anteriormente, alguns IPs possuem uma interface DMA dedicada que realiza a transferência de dados do/para o IP.

A exemplo de outros componentes, o DMA possui uma série de parâmetros de configuração tais como: tamanho das transferências individuais, suporte a interrupção, tamanho do buffer e tamanho total da transferência. A interface DMA utilizada foi desenvolvida no contexto de um trabalho anterior, em [34]. Ela possui 13 registradores que contêm as informações necessárias para que o DMA realize a transação. Esses registradores e suas

respectivas funções podem ser observados na Tabela 3.5.

Tabela 3.5: Registradores do DMA.

Nome do registrador	**Função**	**Endereço**
MB_DMA_START	Sinaliza a existência de um processo de configuração do DMA em andamento.	0x0
MB_DMA_READADD	Endereço de leitura dos dados.	0x4
MB_DMA_WRITEADD	Endereço de escrita dos dados.	0x8
MB_DMA_LENGTH	Tamanho total dos dados a serem transferidos (em bytes).	0xC
MB_DMA_INDSIZE	Tamanho das transferências de leitura/escrita individuais (em bits).	0x10
MB_DMA_INTERRUPT	Suporte à interrupção	0x14
MB_DMA_END_MODE	Sinaliza se o fim da transação é de acordo com o tamanho total especificado ou por controle de fluxo.	0x18
MB_DMA_READPERIF	Tipo de dispositivo de leitura: endereços variáveis ou periférico.	0x1C
MB_DMA_WRITEPERIF	Tipo de dispositivo de escrita: endereços variáveis ou periférico.	0x20
MB_DMA_RESET	Habilita o reinício do DMA.	0x24
MB_DMA_TRANSF_OK	Sinaliza o fim da programação e habilita o DMA a iniciar a transação.	0x28
MB_DMA_END_TRANS	Sinaliza o *status* do DMA (ocupado/disponível).	0x2C
MB_DMA_DATA_COUNT	Sinaliza a quantidade de dados transferidos até o momento pelo DMA.	0x30

Quando há necessidade de realizar uma transferência utilizando o DMA, basta que um nó mestre da plataforma realize a configuração dos 11 primeiros registradores. Após habilitar a transação nesse último registrador, o nó mestre fica livre para executar outras tarefas. Os dois últimos registradores são configurados pelo DMA e sinalizam as informações que o nó mestre solicitante necessita para acompanhar a transação.

Como explicado anteriormente, o tipo de aplicação do decodificador MP3 limita o uso dos dados já que para processar um *frame*, primeiro é necessário preencher totalmente o buffer de entrada e segundo os dados de *frames* anteriores são necessários para a decodificação do *frame* atual. Consequentemente, o PowerPC só pode realizar a extração dos *frames* MP3 após o fim da transferência dos dados do USB para sua memória. Logo, o PowerPC fica em estado de espera verificando se houve alguma mudança de valor no registrador `MB_DMA_END_TRANS`, o que sinaliza que a transação do DMA chegou ao fim.

Quando ele verifica o fim da transação, o PowerPC continua então sua execução.

Os *Players* de Áudio e Vídeo funcionam de maneira semelhante. O buffer de saída contendo os dados `PCM` e `ppm` dos decodificadores não pode sofrer modificações até que os dados tenham sido transferidos para os *Players*. Logo, os decodificadores também ficam em espera até verificarem o fim da transação do DMA.

3.5 NoC

A estrutura de interconexão da plataforma é uma NoC (Network-on-Chip) baseada na NoC Hermes [53]. A NoC Hermes foi desenvolvida pelo GAPH da Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS). Nossa implementação consiste em uma NoC com 9 roteadores conectados numa topologia malha regular 3x3. Apesar desta topologia ter sido definida, a NoC Hermes admite outras topologias, bastando apenas interconectar os roteadores de acordo com a topologia escolhida e selecionar um algoritmo de roteamento adequado.

Cada roteador da NoC possui cinco portas bidirecionais denominadas *WEST*, *EAST*, *NORTH*, *SOUTH* e *LOCAL*. As quatro primeiras portas são utilizadas para conexão entre roteadores vizinhos de acordo com a topologia de malha utilizada. Já a porta LOCAL é utilizada para conectar-se ao IP associado ao roteador. A Figura 3.8 representa a arquitetura da NoC.

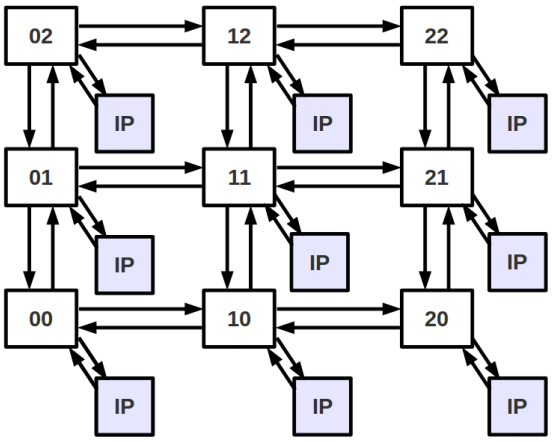


Figura 3.8: Estrutura da NoC Hermes. O endereço de cada nó é dado pelo par de coordenadas XY. Figura adaptada de [53].

Para cada nó da NoC são feitas as interconexões das portas e atribuída uma lógica de controle. Quando uma porta recebe um pacote, ela faz uma solicitação de roteamento e

espera ganhar direito para tentar encaminhar seu pacote. Ao receber direito de acesso ao roteamento, a porta que está armazenando o pacote é atendida pela lógica de controle. Caso não seja possível encaminhar o pacote, o mesmo fica armazenado na porta e, novamente, espera ganhar direito de acesso. Caso seja possível encaminhar o pacote, seus flits são repassados para a porta de destino que lhe foi atribuída pelo algoritmo de roteamento, criando assim uma ligação entre uma porta de entrada e uma porta de saída. Essa ligação é mantida até que o pacote seja enviado completamente para o próximo roteador ou para seu IP local. Quando isso acontece, as portas de E/S conectadas são liberadas para um próximo roteamento.

3.5.1 Roteamento

Na NoC Hermes, quando um novo pacote chega ao nó, o cabeçalho do pacote é armazenado e uma requisição de roteamento é realizada. A arbitragem utilizada para o atendimento das requisições de roteamento é definida com prioridade rotativa (utilizando o algoritmo *round robin*) [69]. Assim, de acordo com a ordem em que as portas são servidas, define-se a prioridade das portas [54].

O algoritmo de roteamento utilizado é o XYPuro e o endereço de cada nó na rede é dado pelas suas coordenadas XY. De posse do cabeçalho do pacote com o endereço de destino (XYalvo) e com o endereço local do nó onde se encontra o cabeçalho (XYlocal), o roteamento com o algoritmo XY puro é realizado da seguinte forma:

```
se (Yalvo == Ylocal){
       se (Xalvo == Xlocal)
              porta = LOCAL;
       senão se (Xalvo > Xlocal)
              porta = LESTE;
       senão se (Xalvo < Xlocal)
              porta = OESTE;
}
senão {
       se (Yalvo > Ylocal)
              porta = NORTE;
       senão se (Yalvo < Ylocal)
              porta = SUL;
}
```

Vale lembrar que, por serem portas bidirecionais, essas portas podem receber e enviar flits ao mesmo tempo. A reserva de portas só é possível se a porta destino estiver livre,

ou seja, se a porta pela qual o pacote será enviado não estiver ocupada com alguma outra transmissão.

3.5.2 Controle de Fluxo

A Hermes é uma NoC que utiliza a estratégia de *packet switching* em sua transmissão de mensagens. Adicionalmente, o tipo de armazenamento utilizado é o *wormhole* com filas de tamanho parametrizável para armazenamento temporário em cada porta de entrada. Como já explicado anteriormente, nesse tipo de armazenamento os pacotes são transmitidos sob a forma de flits. O tamanho do flit na NoC Hermes é parametrizável e atualmente aceita os tamanhos 8, 16, 32, e 64 bits.

De forma a adaptar nossa implementação da NoC à demanda da plataforma, foi definida a estrutura de flits no pacote da Figura 3.9. O primeiro flit corresponde ao cabeçalho e contém tanto o nó de destino (8 bits menos significativos) quanto o de origem (8 bits mais significativos) do pacote. O segundo flit corresponde ao tamanho total da carga útil do pacote expressa em flits, isto é, à soma do flit de tipo com a quantidade de flits de dados. O terceiro flit corresponde ao tipo do pacote. Os demais flits representam o corpo do pacote.

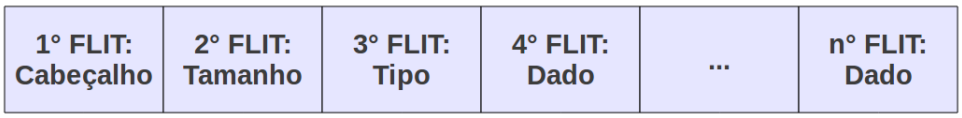


Figura 3.9: Estrutura de flits no pacote de transmissão da MediaBox.

Percebemos no início da implementação da MediaBox que, visto que nossa plataforma possui diversos mestres e escravos comunicando-se entre si, seria necessário criar uma distinção entre os pacotes que chegavam e saíam do nó. Essa distinção se fez necessária ao percebermos a ocorrência do seguinte cenário: um nó mestre envia uma requisição de leitura a um segundo IP e fica esperando pela resposta, enquanto que esse segundo IP faz uma nova requisição a esse nó mestre. Verificamos que não havia garantia que o próximo pacote seria a resposta e, sem um tipo específico de pacote, seria impossível distinguir entre pacotes de requisições e de respostas ocasionando uma execução incorreta da aplicação. Tendo isso em mente, definimos os seguintes possíveis tipos de pacote na MediaBox: requisição de leitura, requisição de escrita e resposta à leitura. Os possíveis valores e pacotes com base nessa classificação podem ser observados nas Figuras 3.10, 3.11 e 3.12.

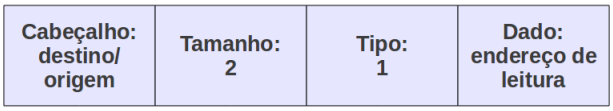


Figura 3.10: Estrutura do pacote de requisição de leitura da MediaBox.

Figura 3.11: Estrutura do pacote de requisição de escrita da MediaBox.

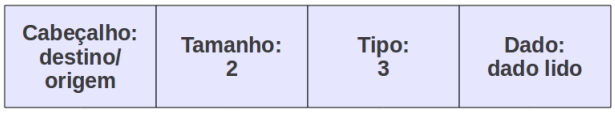


Figura 3.12: Estrutura do pacote de resposta à leitura da MediaBox.

3.5.3 Interface com a NoC

A nossa implementação da NoC inclui uma interface de comunicação entre a NoC Hermes e os nós da MediaBox. Ela foi implementada em SystemC como parte integrante da NoC. Todos os pacotes que entram e saem do roteador passam antes por essa interface, quer seja para realizar a montagem do pacote ou para interpretação dos flits recebidos.

Com base nessa interface é possível criar nós com comportamento mestre e escravo ou somente escravo. Os nós escravos correspondem às memórias enquanto que todos os outros (nós que envolvem processadores ou utilizam DMA) se comportam ora como mestres ora como escravos. A interface foi implementada com base em herança de classes. Temos uma classe base chamada `baseNocModule` que contém os métodos básicos que todos os nós executam tais como o envio e o recebimento de cada flit do/para o roteador. Derivando dessa classe temos uma classe que implementa as operações dos nós escravos chamada `slaveNocModule`. Por fim, temos a classe `masterNocModule` que deriva da classe `slaveNocModule` e, além de possuir as operações referentes ao nó escravo, implementa novas operações que realizam o envio de solicitações de leitura e escrita. Na Figura 3.13 podemos observar a estrutura da interface criada.

Os métodos da classe `baseNocModule` interagem diretamente com as portas do roteador `outPort` e `inPort`, solicitando o envio e o recebimento de cada um dos flits do pacote através dos métodos `sendFlit(FLITTYPE payload)` e `receiveFlit()` respectivamente.

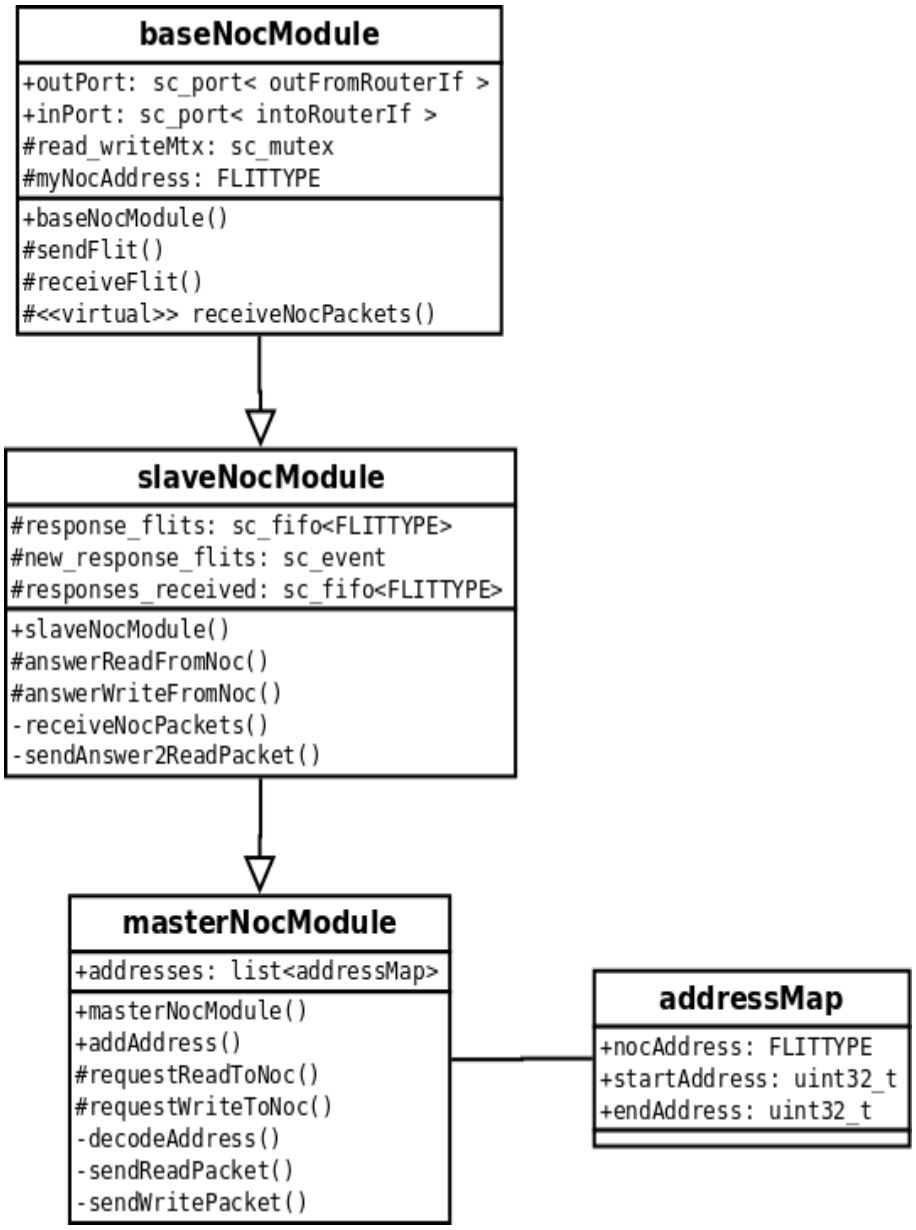


Figura 3.13: Diagrama de classes da interface de comunicação entre os IPs e a NoC.

Já a classe `slaveNocModule` implementa as operações necessárias para o recebimento de solicitações e o envio de respostas. Os métodos presentes na classe e suas respectivas funções podem ser vistos na Tabela 3.6.

Tabela 3.6: Métodos presentes na classe `slaveNocModule`.

Nome do método	**Função**
`answerReadFromNoc(uint32_t address)`	Funciona como meio de interação entre o IP e a NoC na obtenção de dados de leitura do IP. Esse método precisa ser implementado por todos os nós escravos.
`answerWriteFromNoc(uint32_t address, uint32_t data)`	Funciona como meio de interação entre o IP e a NoC na escrita de dados no IP. Esse método precisa ser implementado por todos os nós escravos.
`receiveNocPackets()`	Método responsável por receber os flits, extrair as informações originais do pacote e disparar a execução da ação correspondente à solicitação.
`sendAnswer2ReadPacket()`	Método responsável por enviar eventuais respostas à leituras solicitadas por outros IPs.

A classe `masterNocModule` contém os métodos necessários para o envio tanto de requisições quanto de respostas a outros nós. Os métodos presentes na classe e suas respectivas funções podem ser vistos na Tabela 3.7.

Podemos observar ainda na Figura 3.13 uma classe chamada `addressMap`. Essa classe contém as informações necessárias para comunicação entre os nós. Para que um nó mestre se comunique com outros nós é necessário que ele saiba o endereço deste nó na rede (`nocAddress`) e a faixa de endereçamento do IP vinculado a ele (representado por `startAddress` e `endAddress`).

Na interface de comunicação as informações do mapeamento dos endereços dos IPs na NoC são passadas no momento da instanciação da plataforma. Isso é feito através do método `addAddress(uint8_t nocAddress, uint32_t startAddress, uint32_t endAddress)`. Esse método tem como parâmetros o endereço do roteador da NoC e os endereços inicial e final do IP conectado ao roteador. Dessa forma, ao receber uma requisição com um endereço de memória de destino, a interface de comunicação consegue determinar o roteador ao qual o IP de destino está conectado e encaminhar a requisição. Abaixo está um exemplo do mapeamento feito para a instância de um nó mestre.

```
ppc_audio00.addAddress(0x01,MB_USB_BASE_ADDRESS,MB_USB_END_ADDRESS);
ppc_audio00.addAddress(0x10,MB_MP3_SHARED_MEMORY_START,MB_MP3_SHARED_MEMORY_END);
```

Tabela 3.7: Métodos presentes na classe `masterNocModule`.

Nome do método	Função
`requestReadToNoc(uint32_t address)`	Funciona como meio de interação entre o IP e a NoC nas requisições de leitura para outros nós.
`requestWriteToNoc(uint32_t address, uint32_t data)`	Funciona como meio de interação entre o IP e a NoC nas requisições de escrita para outros nós.
`decodeAddress(uint32_t memaddr)`	Método que verifica para qual nó a requisição será enviada baseado no endereço de memória passado pelo IP.
`sendReadPacket(FLITTYPE _nocAddress, int size, FLITTYPE address)`	Responsável por compor o pacote de requisição de leitura e solicitar seu envio flit a flit.
`sendWritePacket(FLITTYPE _nocAddress, int size, FLITTYPE address, FLITTYPE data)`	Responsável por compor o pacote de requisição de escrita e solicitar seu envio flit a flit.

Nesse exemplo, o nó do PowerPC utilizado na decodificação do áudio representa o endereço 0x00 na rede e se chama `ppc_audio00`. Durante a execução da plataforma, sabemos que esse nó se comunica tanto com o USB na leitura dos dados quanto com o MP3 na decodificação. O nó USB possui o endereço 0x01 na rede e o nó MP3 o endereço 0x10. Os argumentos passados no método representam respectivamente as faixas de endereçamento dos IPs USB e do MP3.

Tanto o modelo utilizado do PowerPC, proveniente do ArchC, quanto o modelo de DMA utilizam a interface TLM. Essa interface funciona enviando requisições no formato `ac_tlm_req` e recebendo respostas no formato `ac_tlm_rsp`. Para interagir com a NoC, adaptamos esses nós da seguinte forma:

- Quando necessitam fazer uma solicitação à NoC, os nós extraem as informações necessárias dessa requisição para compor os flits. Em seguida, os dados resultantes são encaminhados para a interface com a NoC para serem transferidos.

- Quando recebem flits da NoC, os nós constroem um pacote no formato TLM (como o exemplo descrito na Seção 2.3.1) com as informações dos flits e os encaminham para o IP.

3.5.4 Adaptação da Hermes à MediaBox

Nós encontramos alguns problemas com o uso da NoC na MediaBox. Apesar do algoritmo utilizado ser livre de *deadlocks* de roteamento, os *deadlocks* de mensagens (explicados na Seção 2.5.3) estavam causando o bloqueio da NoC. Percebemos que isso acontecia quando dois mestres enviavam solicitações de leitura um ao outro. Ambos recebiam a requisição porém não a atendiam pois ficavam à espera da resposta de sua própria requisição, bloqueando assim a rede indefinidamente. Resolvemos esse problema fazendo o desligamento da dependência do envio e recebimento de respostas no nó. Para isso, criamos duas filas separadas de envio `response_flits` e recebimento `responses_received` de respostas à leituras ambas na classe `slaveNocModule`. O evento `new_response_flits` da mesma classe é utilizado para disparar o envio de flits de resposta assim que estiverem disponíveis.

Mesmo após a resolução desse problema percebemos que ainda havia um conflito no uso dos recursos afetando a execução da plataforma e causando novamente *deadlocks* de mensagem. A NoC Hermes utiliza um buffer de entrada para armazenamento prévio dos flits antes do envio do pacote. Visto que a separação entre o envio e o recebimento de respostas foi feita, havia agora uma disputa pelas posições do buffer entre novas solicitações e envio de respostas. Como não havia nenhum controle de acesso ao buffer, flits de diferentes requisições acabavam por se intercalar no buffer causando uma execução errônea e muitas vezes um *deadlock*.

De forma a resolver esse problema, criamos na classe `baseNocModule` um semáforo chamado `read_writeMtx`. A função desse semáforo é limitar o acesso ao buffer, somente uma requisição coloca flits por vez. Ao ganhar acesso ao buffer, o solicitante envia todo o pacote antes de desbloquear o acesso.

3.6 Inclusão de novos IPs à MediaBox

A inclusão de novos IPs à MediaBox exige pouca implementação. As etapas a seguir descrevem como fazer essa inclusão.

1. Incluir o IP na pasta "ip"da estrutura da ARP utilizada;
2. Incluir o nome do IP no arquivo "defs.arp"presente na pasta da plataforma MediaBox.
3. No arquivo de instanciação da plataforma, realizar a instanciação do IP e atribuir a ele um nó da NoC.
4. No código do novo IP, incluir o arquivo de interface da NoC e codificar a utilização dos métodos existentes.

5. Definir uma nova aplicação ou realizar mudanças na aplicação já existente que incluam o IP.

No item 4 faz-se necessária a implementação de alguns dos métodos apresentados na Seção 3.5.3 dependendo do comportamento do IP (mestre ou escravo). Caso seja apenas mestre, basta utilizar os métodos de envio de pacotes `requestReadToNoc` e `requestWriteToNoc` já implementados. Caso o IP seja escravo, faz-se necessária a implementação dos métodos virtuais `answerReadFromNoc` e `answerWriteFromNoc`. Esses métodos possuem como argumentos endereços e/ou dados e cabe ao programador definir o comportamento do novo IP ao receber essas solicitações.

3.7 Ferramentas

Para o desenvolvimento da plataforma MediaBox, alguns critérios foram estabelecidos de forma a manter um certo controle sobre a implementação.

Escolhemos o Linux como sistema operacional, mais especificamente a distribuição Ubuntu. Além de ser uma distribuição gratuita, Ubuntu é uma distribuição bastante popular e apresenta uma grande disponibilidade de software. Outro diferencial é sua estabilidade. No desenvolvimento dos IPs da plataforma utilizamos o GCC 4.4.5 em conjunto com a biblioteca GLIBC versão 2.12.1 ambos gratuitos e disponíveis na Internet.

A linguagem de programação escolhida foi SystemC, versão 2.2. SystemC oferece uma série de recursos que facilita a criação de modelos de simulação de componentes de hardware. Adicionalmente, SystemC possui uma biblioteca que facilita o desenvolvimento dos componentes de comunicação entre os módulos. Esta biblioteca, TLM 2.0, também é utilizada.

Capítulo 4

Resultados Experimentais

Neste capítulo apresentamos os resultados experimentais obtidos com a plataforma MediaBox. Inicialmente, descrevemos os cenários desenvolvidos para realização dos experimentos bem como as configurações utilizadas. Em seguida, apresentamos o estudo de caso realizado, o mecanismo implementado para a extração de informações e as informações obtidas durante a execução da plataforma. Por fim, apresentamos os resultados obtidos.

4.1 Experimentos

Os experimentos realizados na MediaBox têm o intuito não só de verificar a corretude de seu funcionamento como também a capacidade da plataforma com relação à quantidade de dados processados. Essa capacidade de processamento é avaliada exercitando a plataforma com arquivos de entrada de diferentes tamanhos. Nos experimentos nós utilizamos arquivos de entrada (áudio e vídeo) de um e cinco segundos de duração.

Já a corretude do funcionamento da plataforma é verificada com base nos arquivos de saída. Como explicado no Capítulo 3, nós realizamos a adaptação de duas aplicações de decodificação de áudio e vídeo para uso na MediaBox. De posse dessas duas aplicações e antes de fazer qualquer modificação nas mesmas, nós executamos a decodificação de todos os arquivos de entrada fora do contexto da plataforma. Como resultado dessas execuções em uma máquina nativa, obtivemos arquivos de saída considerados corretos e que foram utilizados como base para comparação com os arquivos gerados pela MediaBox.

Os arquivos de entrada utilizados correspondem a dez clipes diferentes de filmes extraídos de [10]. Os clipes foram selecionados aleatoriamente e encontravam-se no formato `mp4`. Como esse formato não é compatível com a nossa aplicação, foi preciso fazer a codificação do filme no formato `mp2`. Em seguida, foi feita a redução de cada um dos clipes para amostras de um e de cinco segundos para uso nos experimentos. Finalmente, foi feita a separação dos fluxos de áudio e vídeo de cada um dos dez clipes. Todas as

tarefas de codificação, redução e separação foram realizadas nas amostras utilizando o programa FFMPEG [9] disponível gratuitamente online. Informações referentes a cada um dos arquivos de entrada utilizados podem ser vistas na Tabela 4.1 e na Tabela 4.2.

Tabela 4.1: Informações acerca dos arquivos de entrada de vídeo.

Frames (1s)	24
Frames (5s)	120
Resolução de vídeo	852x362
Frames por segundo	24

Tabela 4.2: Informações acerca dos arquivos de entrada de áudio.

Amostra	**Frames (1s)**	**Frames (5s)**	**Frequência**	**Taxa de bits**	**Canais**
01	42	208	48 KHz	112 kbps	2
02	39	192	44,1 KHz	80 kbps	2
03	39	192	44,1 KHz	96 kbps	2
04	39	192	44,1 KHz	160 kbps	2
05	42	208	48 KHz	128 kbps	2
06	42	208	48 KHz	80 kbps	2
07	39	192	44,1 KHz	112 kbps	2
08	39	192	44,1 KHz	128 kbps	2
09	42	208	48 KHz	64 kbps	2
10	42	208	48 KHz	96 kbps	2

4.1.1 Cenários e Configurações

Para a realização dos experimentos definimos que seriam utilizados dois tipos de cenários possíveis de posicionamento dos nós da NoC: o melhor e o pior caso. O intuito de utilizar dois cenários diferentes é verificar o funcionamento da interface da NoC desenvolvida bem como avaliar o desempenho da NoC e da plataforma.

Inicialmente, o critério escolhido para a definição desses cenários foi a distância entre dois nós que se comunicam na rede. Com base nesse critério, nós escolhemos uma NoC de nove nós, três a mais do que o utilizado pelas aplicações da MediaBox. Esse três nós a mais foram preenchidos com três memórias. O fato da MediaBox ter três nós sobressalentes, não prejudicava a simulação e facilitava o reuso da plataforma, bastando apenas substituir as memórias por outros IPs.

O tamanho da NoC escolhido não afetou a definição do cenário de melhor caso, visto que todos os nós que se comunicam foram posicionados um ao lado do outro. Já para o

cenário de pior caso, esse tamanho de NoC possibilitou a definição de um cenário com maior distância entre os nós que se comunicam, de forma a provocar uma maior utilização da NoC.

Entretanto, simulações iniciais desses cenários da plataforma mostraram que o critério escolhido não produziu o resultado esperado. Os canais e roteadores da NoC não apresentavam maior estresse no cenário de pior caso em comparação ao cenário de melhor caso. O desempenho da plataforma como um todo era praticamente o mesmo, só havia variação no tempo pois os pacotes demoravam mais para percorrer o caminho entre a origem e o destino.

Dada a necessidade de uma mudança de critério de definição dos cenários, nós nos baseamos nos resultados obtidos nas simulações iniciais e verificamos que uma maneira de provocar o estresse desejado na NoC seria escolher os cenários com base na quantidade de colisões nas rotas dos nós que mais se comunicam. Ou seja, nós procuramos obter o máximo e o mínimo de colisões em cada um dos cenários. No melhor caso, a distância entre dois nós comunicantes permaneceu a mesma, igual a um. Visto que esses nós são adjacentes, o número de colisões é minimizado. Esse cenário pode ser visto na Figura 4.1.

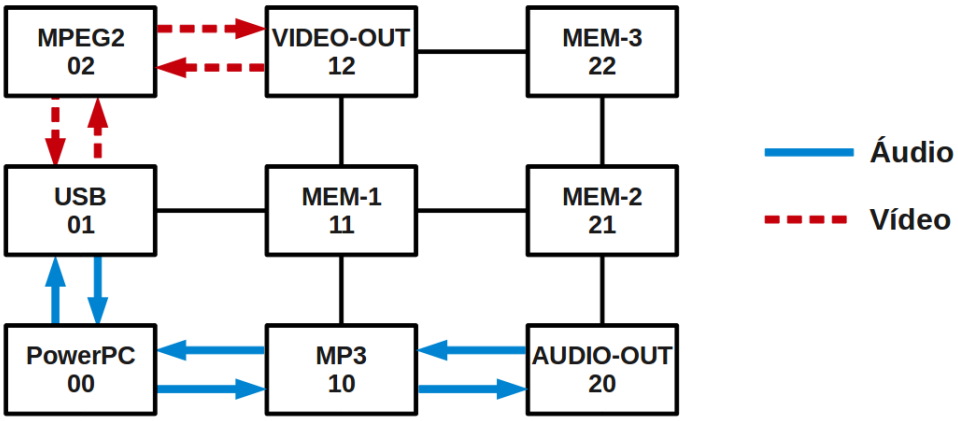


Figura 4.1: Cenário de melhor caso de posicionamento dos nós da NoC com rotas de comunicação dos fluxos de áudio e vídeo.

Ainda na Figura 4.1, podemos observar os caminhos feitos pelos pacotes para cada um dos fluxos de áudio e de vídeo. No fluxo de áudio, a execução tem início com o PowerPC solicitando ao USB a transferência de dados de áudio via DMA. Após a transferência dos dados pelo USB, o PowerPC inicia o processamento dos dados e os encaminha para o MP3 para decodificação. Ao término da decodificação, dados de controle são enviados ao PowerPC que prossegue para o próximo *frame*. Durante a decodificação, dados de saída, assim que disponíveis, são transferidos via DMA do MP3 para o dispositivo de

saída AUDIO-OUT.

Similarmente, no fluxo de vídeo a execução tem início com o MPEG-2 solicitando ao USB a transferência de dados de vídeo via DMA. Após a transferência dos dados pelo USB a decodificação começa no MPEG-2. Dados de saída, assim que disponíveis, são também transferidos via DMA do MPEG-2 para o dispositivo de saída VIDEO-OUT.

Para a definição do pior caso, procuramos posicionar os nós de forma a obter um maior número de colisões entre as rotas de dois nós comunicantes. O cenário de pior caso pode ser visto na Figura 4.2.

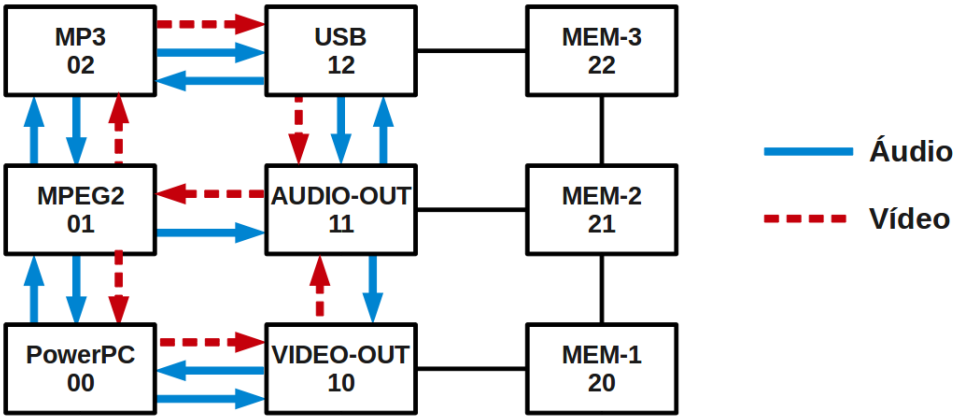


Figura 4.2: Cenário de pior caso de posicionamento dos nós da NoC com rotas de comunicação dos fluxos de áudio e vídeo.

No cenário de pior caso, os fluxos de execução de áudio e vídeo são idênticos aos explicados anteriormente. Porém, como podemos observar na Figura 4.2, os caminhos percorridos pelos pacotes são mais longos. Para criar esse cenário, utilizamos como base as saídas do algoritmo de roteamento `XYPuro` apresentado em 3.5.1 e utilizado pela NoC. Os caminhos percorridos correspondentes às ações de cada um dos fluxos são:

- **PowerPC ao USB:** PowerPC -> MPEG-2 -> MP3 -> USB;
- **USB ao PowerPC:** USB -> AUDIO-OUT -> VIDEO-OUT -> PowerPC;
- **PowerPC ao MP3 e vice-versa:** PowerPC <-> MPEG-2 <-> MP3;
- **MP3 ao AUDIO-OUT:** MP3 -> MPEG-2 -> AUDIO-OUT;
- **AUDIO-OUT ao MP3:** AUDIO-OUT -> USB -> MP3;
- **MPEG-2 ao USB:** MPEG-2 -> MP3 -> USB;

- **USB ao MPEG-2:** USB -> AUDIO-OUT -> MPEG-2;
- **MPEG-2 ao VIDEO-OUT:** MPEG-2 -> PowerPC -> VIDEO-OUT;
- **VIDEO-OUT ao MPEG-2:** VIDEO-OUT -> AUDIO-OUT -> MPEG-2;

O mapa de endereçamento da plataforma durante os experimentos corresponde ao que está no Apêndice na Seção A.1. Nós configuramos os registradores de forma a utilizar os *Players* apenas como dispositivos de armazenamento, sem a execução do áudio/vídeo no final da decodificação. Essa escolha foi feita pois as simulações foram executadas remotamente em um Cluster Condor [11] que não possui interface gráfica. Adicionalmente, foi criada mais uma variável no arquivo de configuração chamada `MB_DISPLAY_INFO`. Assim como as outras variáveis de configuração apresentadas na Seção 3.1, ela pode ser configurada para "yes"ou "no"de forma a exibir ou não as informações obtidas durante a simulação.

Os experimentos correspondem à realização de dez simulações referentes a cada uma das dez amostras de um segundo no melhor caso, dez simulações referentes a cada uma das amostras de um segundo no pior caso e dez simulações referentes a cada uma das amostras de cinco segundos no melhor caso. Foi utilizada a mesma máquina para todas as simulações. A máquina possui um processador `Intel Xeon CPU X5650` com 2 CPUs, cada um com 6 núcleos, e frequência de 2.66GHz.

4.1.2 Geração de Informação

Com o objetivo de extrair informações acerca do funcionamento e do desempenho da MediaBox, um mecanismo de obtenção de informações foi implementado na NoC. O modelo da NoC Hermes utilizado já provia as seguintes informações cumulativas em cada um dos nós:

- Latência associada ao número de vezes que houve uma competição pelo acesso ao recurso de roteamento antes de enviar o pacote (cada tentativa equivale a um nanossegundo).
- Número de tentativas de enviar um pacote.

Visto que essas informações não eram associadas a nenhum par (origem, destino) específico, de forma a fazer uma melhor avaliação do desempenho da MediaBox, nós definimos outras informações a serem extraídas. São elas:

- Quantidade de pacotes que passam pelo nó.

- Quantidade de pacotes que são do/para o IP local do nó.
- Latência, número de tentativas e quantidade de pacotes associados aos nós.

As informações contidas no último item são extraídas com base no nó de origem e no nó de destino do pacote de forma a verificar as informações associadas entre cada par de nós que se comunicam. Dessa forma, quando o pacote passa pelo nó, essas informações são acumuladas para cada par (origem, destino).

As informações são obtidas através da interceptação e interpretação dos pacotes assim que seu caminho é definido e antes da sua saída do roteador. Caso assim esteja configurado, é feita a impressão dessas informações no fim da simulação. Para realizar essa impressão direto da NoC, foi necessário especificar um novo tipo de pacote: requisição de informação. A estrutura desse novo pacote pode ser observada na Figura 4.3.

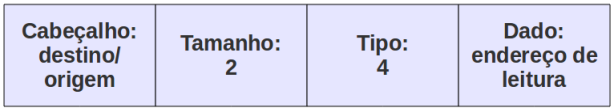


Figura 4.3: Estrutura do pacote de requisição de informação da MediaBox.

Visto que os processadores utilizados fazem apenas leituras e escritas em endereços de memória, a montagem do pacote como sendo de requisição de informação é feita pela interface da NoC. O que nós fizemos foi selecionar uma faixa de memória não utilizada pelos nós da MediaBox e atribuir qualquer requisição para essa faixa como sendo uma solicitação de impressão de informações. Quando uma leitura à essa faixa de endereço especial chega à interface com a NoC, o pacote é interceptado, convertido para o novo formato e enviado. Na MediaBox, essa faixa é especificada pelo registrador `MB_REQ_INFO` e pode ser vista no arquivo de configurações da Seção A.1 no Apêndice.

A interpretação do pacote como uma solicitação de informações e a impressão dos dados só é feita pelos nós de origem e de destino. Quando um pacote de informação passa por esses roteadores, significa que a simulação chegou ao fim e falta apenas imprimir os dados salvos. De forma a centralizar e controlar melhor a impressão das informações foi restrito ao PowerPC e ao MP3 a capacidade de solicitar a impressão dos dados. Logo, o PowerPC faz a solicitação ao USB, ao MPEG-2, ao *Player* de Vídeo e às Memórias, enquanto que o MP3 a faz ao *Player* de Áudio. Um exemplo das informações geradas para um dos nós durante a simulação pode ser vista na Seção A.3 no Apêndice.

4.2 Resultados

Os resultados obtidos nas simulações realizadas na MediaBox são apresentados nesta Seção. Com exceção do tempo de execução, todos os resultados apresentados são relativos apenas às simulações com as amostras de um segundo. Além disso, os resultados referentes à latência, quantidade de pacotes e contenção dos nós foram calculados como a média dos resultados obtidos nas dez simulações realizadas. Por fim, para cada simulação, os arquivos de saída foram devidamente verificados e encontravam-se de acordo com os arquivos base.

4.2.1 Tempo de Execução

O tempo total de execução da simulação da plataforma para cada uma das dez amostras utilizadas no estudo de caso pode ser observado na Tabela 4.3. O impacto causado pelos diferentes cenários foi pequeno. Entre o melhor e o pior cenário, a variação do tempo foi de no mínimo 14 e no máximo 27 minutos para as amostras de um segundo. As amostras de cinco segundos tiveram um tempo de execução variando entre 21 e 33 horas de execução.

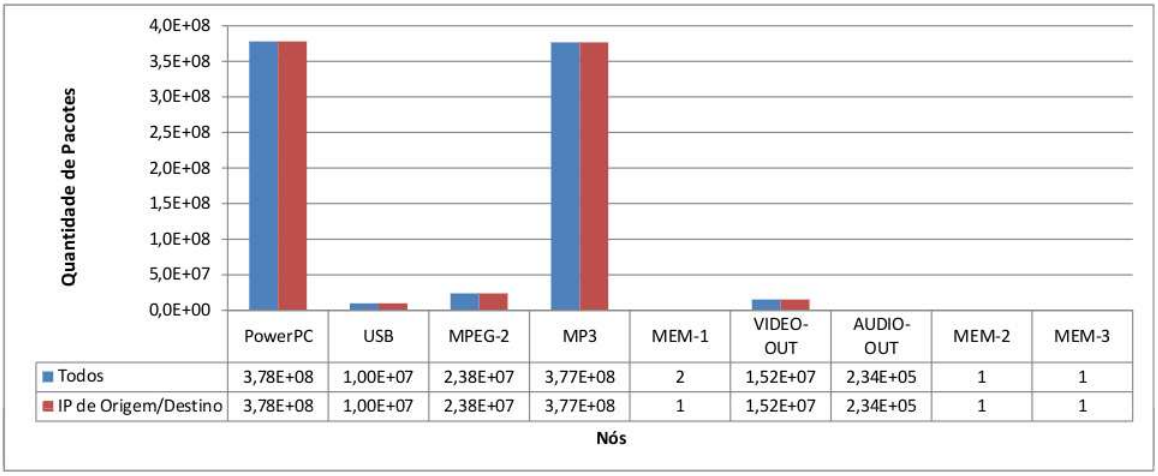
Tabela 4.3: Tempo de execução das amostras.

Amostras / Plataforma	1s		5s
	Melhor Caso	Pior Caso	Melhor Caso
01	5:15:20	5:30:57	25:48:22
02	4:25:14	4:40:33	21:05:58
03	4:56:23	5:13:46	24:22:24
04	4:56:18	5:12:09	25:10:05
05	5:40:14	6:01:31	27:38:33
06	4:52:52	5:07:16	23:49:30
07	5:01:03	5:21:06	24:44:24
08	6:42:35	7:00:43	33:07:56
09	4:49:51	5:03:58	24:31:09
10	6:51:08	7:18:46	28:07:13

4.2.2 Número de Pacotes e Latência

De forma a verificar o impacto do posicionamento dos nós na NoC, extraímos as informações referentes ao número de pacotes que passam pelo roteador. Identificamos a quantidade total e a quantidade de pacotes que têm como origem ou destino o IP local. Essas informações podem ser vistas na Figura 4.4 para o cenário de melhor caso. Como

esperado, não há variação entre os dois valores visto que os nós que se comunicam são adjacentes.



	PowerPC	USB	MPEG-2	MP3	MEM-1	VIDEO-OUT	AUDIO-OUT	MEM-2	MEM-3
Todos	3,78E+08	1,00E+07	2,38E+07	3,77E+08	2	1,52E+07	2,34E+05	1	1
IP de Origem/Destino	3,78E+08	1,00E+07	2,38E+07	3,77E+08	1	1,52E+07	2,34E+05	1	1

Figura 4.4: Histórico dos pacotes associado aos nós da MediaBox no cenário de melhor caso.

Nós verificamos também a latência total associada aos nós em cada simulação. Esse valor corresponde à soma das latências associadas ao envio e recebimento de cada um dos pacotes que passaram pelo nó. Com base nos valores de cada uma das dez simulações, calculamos a média da latência total dos nós no cenário de melhor caso. O resultado desse cálculo pode ser observado na Figura 4.5.

Como explicado anteriormente, essa latência está associada ao número de vezes que houve uma competição pelo acesso ao recurso de roteamento antes de enviar o pacote. Por padrão, cada tentativa equivale a 1 nanossegundo. Podemos observar pela quantidade de pacotes enviados no cenário de melhor caso da Figura 4.4, que a latência equivale em geral ao dobro dessa quantidade, ou seja, a latência associada ao envio de um pacote é, em média, de 2 nanossegundos.

O número de pacotes dos nós e as latências associadas ao cenário de pior caso podem ser observadas nas Figuras 4.6 e 4.7 respectivamente.

No cenário de pior caso, houve uma variação entre a quantidade de pacotes total e a que tem como origem ou destino o IP local em praticamente todos os nós. Isso mostra que o posicionamento escolhido como pior caso provê o cenário desejado. Os pacotes circulam pela NoC até chegarem ao seu destino impactando na disputa por recursos de roteamento, na latência da rede e no tempo de simulação.

Além disso, o resultado obtido em ambos cenários mostra claramente uma maior quantidade de pacotes e de latência nos nós do PowerPC e do MP3. Isso está associado ao

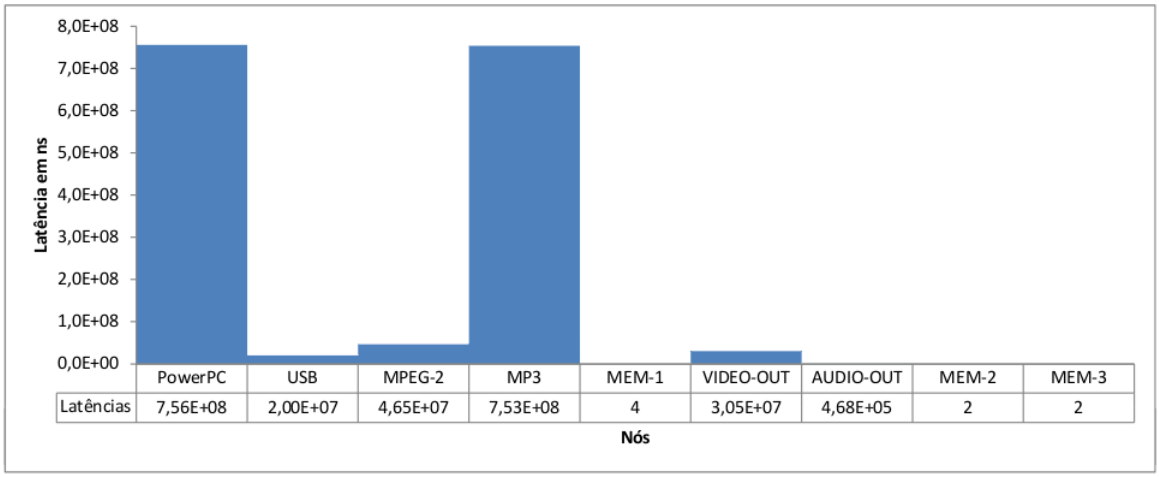


Figura 4.5: Média das latências totais associadas aos nós da MediaBox no cenário de melhor caso.

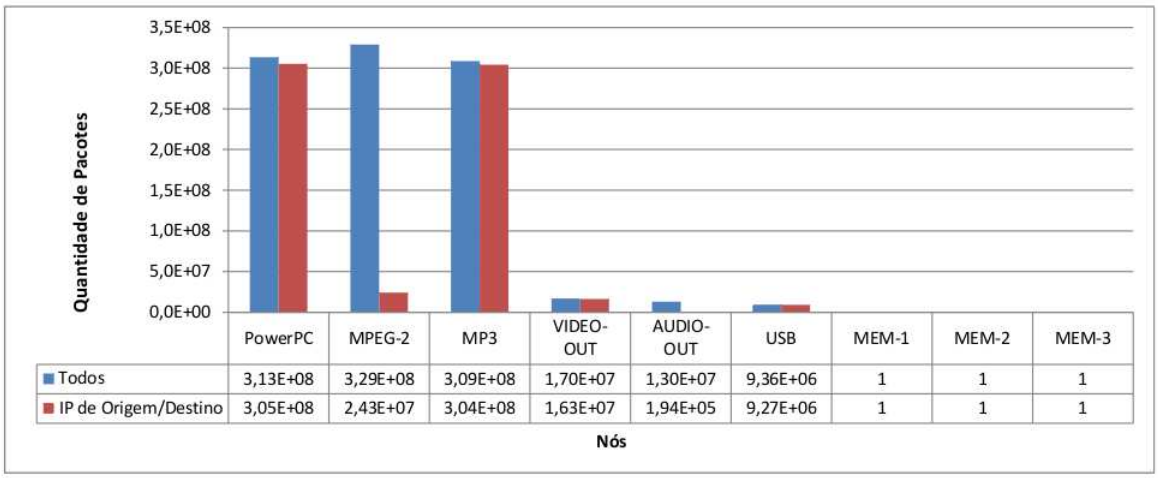


Figura 4.6: Histórico dos pacotes associado aos nós da MediaBox no cenário de pior caso.

fato de que o algoritmo de decodificação de vídeo é mais simples que o de áudio, ou seja, o MP3 realiza mais cálculos. Por esse motivo, o processamento de cada *frame* de áudio é significativamente mais demorado que o de um *frame* de vídeo. Durante a decodificação realizada pelo MP3, o PowerPC envia constantemente pacotes de leitura endereçados ao registrador de controle do MP3 para verificar o andamento da decodificação. O MP3 responde à tais solicitações aumentando assim a quantidade de pacotes que circula entre os IPs e, consequentemente, a latência associada ao envio de cada um. O mesmo não acontece com os outros nós logo, em comparação ao PowerPC e ao MP3, eles apresentam

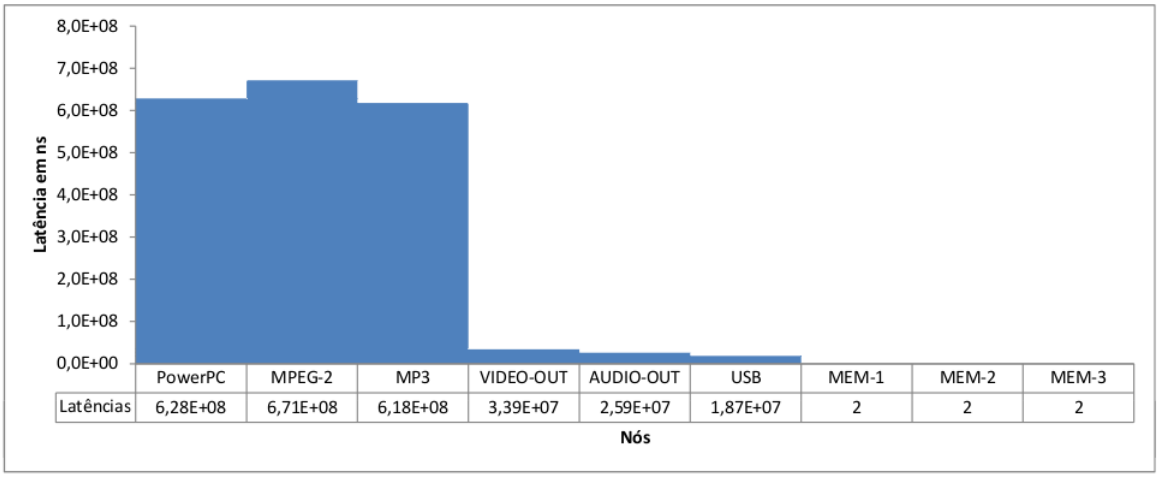


Figura 4.7: Média das latências totais associadas aos nós da MediaBox no cenário de pior caso.

uma baixa latência e quantidade de pacotes.

Podemos ver ainda na Figura 4.6 que a quantidade de pacotes do nó MPEG-2 cresceu vertiginosamente, chegando a ultrapassar o PowerPC e o MP3. De acordo com a Figura 4.2 esse resultado era esperado visto que a quantidade de caminhos e pacotes que passam pelo MPEG-2 é muito maior que no cenário de melhor caso. Além disso, os nós VIDEO-OUT e AUDIO-OUT apresentaram também uma variação maior em sua quantidade de pacotes e latência pelos mesmos motivos.

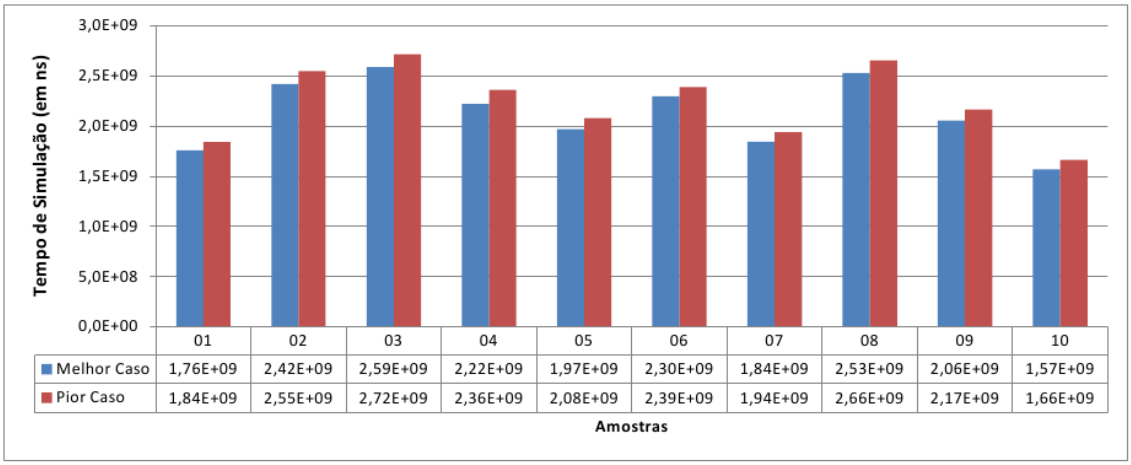
Na Figura 4.7 podemos observar que o posicionamento dos nós teve uma influência tanto positiva quanto negativa sobre a latência associada ao nó. Os valores de latência do PowerPC e do MP3, apesar de altos, foram menores que os valores apresentados no cenário de melhor caso. Isso pode ser atribuído ao fato de que os pacotes tanto de leitura quanto de resposta que circulam entre esses IPs demoram mais tempo para chegar ao seu destino, tempo no qual o MP3 continua a decodificação. Consequentemente, tanto a quantidade de pacotes trocada entre os dois IPs (como podemos observar pelos resultados em 4.4 e 4.6) quanto a latência associada acabam sendo menores.

Os nós MEM-1, MEM-2 e MEM-3, não foram utilizados pelas aplicações desse estudo de caso. Por esse motivo, os números de pacotes e latências associadas a esses nós correspondem ao recebimento do pacote de informação. As variações nos valores do nó MEM-1 no cenário de melhor caso acontecem porque ele está no caminho do pacote de informação do nó MEM-2.

4.2.3 Tempo de Simulação

O tempo de simulação também foi medido para cada uma das amostras. Essa métrica corresponde ao tempo corrente da simulação que o simulador do SystemC mantém durante sua execução. A obtenção desse valor foi feita através da chamada da função `sc_time_stamp` após o término da execução das aplicações de áudio e vídeo.

O tempo de simulação em nanossegundos de cada uma das amostras nos cenários de melhor e pior caso para o fluxo de vídeo pode ser observado na Figura 4.8. Como esperado, todas as amostras tiveram um tempo de simulação maior no cenário de pior caso.



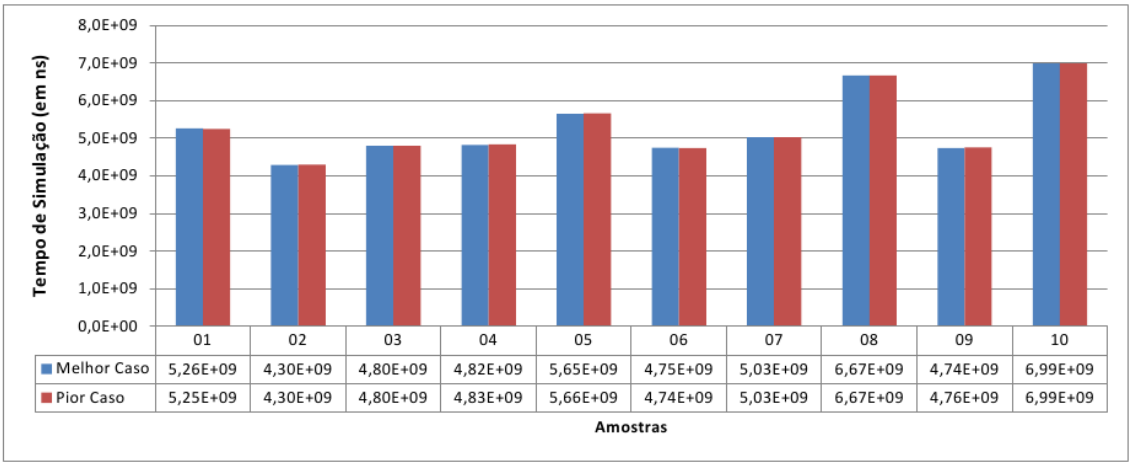
	01	02	03	04	05	06	07	08	09	10
Melhor Caso	1,76E+09	2,42E+09	2,59E+09	2,22E+09	1,97E+09	2,30E+09	1,84E+09	2,53E+09	2,06E+09	1,57E+09
Pior Caso	1,84E+09	2,55E+09	2,72E+09	2,36E+09	2,08E+09	2,39E+09	1,94E+09	2,66E+09	2,17E+09	1,66E+09

Figura 4.8: Tempo de simulação do fluxo de vídeo para os cenários de melhor e pior caso.

Já o tempo de simulação em nanossegundos de cada uma das amostras nos cenários de melhor e pior caso para o fluxo de áudio pode ser observado na Figura 4.9. A diferença entre os tempos dos diferentes cenários é pequena se comparada ao resultado obtido no fluxo de vídeo. Isso pode ser atribuído ao fato de que a maior parte do tempo da decodificação do áudio é gasta no MP3. Como essa parcela de tempo não é afetada pelo posicionamento diferente do nós, temos como resultado apenas uma pequena variação entre o melhor e o pior caso.

Adicionalmente e ao contrário do esperado, as amostras 01 e 06 tiveram um tempo de simulação maior no cenário de melhor caso. Nós acreditamos que essa diferença possa estar ligada à diferença na quantidade de pacotes e na latência associada ao seu envio entre o melhor e pior caso. Como apresentado na Seção 4.2.2, o número de pacotes no pior caso acabou sendo menor para alguns IPs e isso pode ter resultado no menor tempo de simulação para essas amostras no mesmo cenário.

Como pode ser observado em ambas as Figuras, em geral, a variação entre o melhor e o pior cenário em ambos os fluxos é considerável e equivale a cerca de 10^8. Além disso,



	01	02	03	04	05	06	07	08	09	10
Melhor Caso	5,26E+09	4,30E+09	4,80E+09	4,82E+09	5,65E+09	4,75E+09	5,03E+09	6,67E+09	4,74E+09	6,99E+09
Pior Caso	5,25E+09	4,30E+09	4,80E+09	4,83E+09	5,66E+09	4,74E+09	5,03E+09	6,67E+09	4,76E+09	6,99E+09

Figura 4.9: Tempo de simulação do fluxo de áudio para os cenários de melhor e pior caso.

podemos notar que o tempo de simulação do fluxo de áudio é muito maior que o de vídeo. Novamente, isso acontece porque o processamento de cada *frame* de áudio é significativamente mais demorado que o de um *frame* de vídeo, impactando assim no tempo de simulação. Uma outra justificativa está no fato de que a aplicação de decodificação de áudio utilizada foi desenvolvida com um intuito didático e por isso é mais lenta. Já a aplicação de decodificação de vídeo foi projetada para ter um maior desempenho impactando assim na diferença entre os tempos de simulação.

Adicionalmente, visto que o fluxo de áudio termina sua decodificação por último, o tempo de simulação apresentado na Figura 4.9 corresponde também ao tempo total da simulação das amostras.

4.2.4 Contenção entre os Nós da NoC

Com o intuito de analisar a contenção da rede, foram capturadas as informações de tentativas de enviar o pacote e quantidade de pacotes associadas entre um par de nós (origem, destino). A taxa de contenção foi calculada através da divisão do número de tentativas pelo número de pacotes. Os valores referentes às simulações no cenário de melhor caso podem ser vistos na Tabela 4.4.

Na Tabela, os nós da primeira coluna representam os nós de origem enquanto os nós da primeira linha representam os nós de destino. Os valores representados por "0"correspondem a nós que não se comunicam. Já os valores "−−−"representam a impossibilidade de haver comunicação entre um mesmo nó de origem e de destino.

No cenário de melhor caso há variação somente na contenção associada ao USB. Isso ocorre porque, inicialmente, tanto o MPEG-2 quanto o PowerPC estão com os buffers

Tabela 4.4: Contenção associada aos nós da MediaBox no cenário de melhor caso.

Origem/Destino	00 - PowerPC	01 - USB	02 - MPEG-2	10 - MP3	11 - MEM-1	12 - VIDEO-OUT	20 - AUDIO-OUT	21 - MEM-2	22 - MEM-3
00 - PowerPC	----	1,000007362	1	1	1	1	0	1	1
01 - USB	1	----	1	0	0	0	0	0	0
02 - MPEG-2	0	1,000000543	----	0	1	1	0	0	0
10 - MP3	1	0	0	----	1	0	1	0	0
11 - MEM-1	1	0	0	0	----	0	0	0	0
12 - VIDEO-OUT	1	0	1	0	0	----	0	0	0
20 - AUDIO-OUT	0	0	0	1	0	0	----	0	0
21 - MEM-2	1	0	0	0	0	0	0	----	0
22 - MEM-3	1	0	0	0	0	0	0	0	----

vazios e fazem solicitações de novos dados ao USB. A variação é pequena pois, por possuírem tamanhos de buffers diferentes e terem diferentes durações de decodificação de *frames*, novas solicitações de dados não se intercalam.

Os valores de contenção referentes às simulações de pior caso podem ser observados na Tabela 4.5. A variação nesse cenário é maior e mais frequente. Isso ocorre porque há comunicação constante entre nós não adjacentes. Boa parte da comunicação com origem/destino no PowerPC precisa passar pelo MPEG-2. O MPEG-2, por trabalhar com vídeo, utiliza muito mais dados que o decodificador de áudio. Por esse motivo, leituras constantes são feitas ao USB. O caminho percorrido por essas requisições de leitura colide com o caminho de dados entre o MP3 e PowerPC que por si só trocam bastante dados. Esses fatores afetam a contenção entre esses nós.

Tabela 4.5: Contenção associada aos nós da MediaBox no cenário de pior caso.

Origem/Destino	00 - PowerPC	01 - MPEG-2	02 - MP3	10 - VIDEO-OUT	11 - AUDIO-OUT	12 - USB	20 - MEM-1	21 - MEM-2	22 - MEM-3
00 - PowerPC	----	1	1,003667358	1	0	1,00000712	1	1	1
01 - MPEG-2	0	----	0	1,062949897	0	1,099105184	0	0	0
02 - MP3	1,014206972	0	----	0	1	0	1	0	0
10 - VIDEO-OUT	0	1	0	----	0	0	0	0	0
11 - AUDIO-OUT	0	0	1,005670583	0	----	0	0	0	0
12 - USB	1	1	0	0	0	----	0	0	0
20 - MEM-1	0	0	0	0	0	0	----	0	0
21 - MEM-2	0	0	0	0	0	0	0	----	0
22 - MEM-3	0	0	0	0	0	0	0	0	----

A contenção associada aos pacotes enviados do MPEG-2 ao VIDEO-OUT também teve variação. Isso aconteceu porque o número, o tamanho e, consequentemente, a quantidade de dados de cada *frame*, é grande. Além disso, o caminho entre esses nós passa pelo PowerPC que, como já explicado, possui um grande tráfego de pacotes.

O gerenciamento da grande quantidade de pacotes que passam pelo roteador acaba aumentando a disputa pelo recurso de roteamento e interferindo na contenção associada aos nós envolvidos.

Capítulo 5

Trabalhos Relacionados

Nesse capítulo apresentamos alguns trabalhos relacionados cujo foco é o desenvolvimento de plataformas que permitem a verificação de SoCs bem como técnicas de otimização e análise aplicadas no contexto de plataformas virtuais.

5.1 Revisão Bibliográfica

A literatura atual inclui diversas direções de pesquisa envolvendo plataformas virtuais. Alguns critérios gerais que definem os trabalhos atuais são: velocidade de simulação, análise de desempenho e verificação.

Ao se abstrair a comunicação de baixo nível e os detalhes da computação, pode-se obter uma melhora significativa na velocidade de simulação. Entretanto, visto que uma aplicação pode executar centenas de milhões ou bilhões de instruções, essa melhoria pode não ser suficiente e o tempo de execução pode ser longo demais para produzir resultados práticos. Nesse sentido, trabalhos como [50, 62] propõem mecanismos para aumentar a velocidade da simulação.

O trabalho desenvolvido por Lin et al. em [50] apresenta métodos de sincronização baseados em dependência de dados e técnicas de escalonamento para simulações de plataformas virtuais. A combinação de mecanismos de dissociação de *clocks* e de acesso direto à memória permite realizar uma sincronização virtual nas simulações. Como resultado, tem-se o aumento da velocidade em relação a simulações baseadas em ciclos de *clock*.

Já em [62], Qin et al. misturam simulação interpretada com simulação compilada de um ISS para permitir uma abordagem de simulação multiprocessada. O simulador seleciona os blocos de código executados frequentemente e os traduz em bibliotecas de carregamento dinâmico (DLLs) que são acopladas ao simulador em tempo de execução. Enquanto o simulador executa a simulação em um processador, as tarefas de tradução são distribuídas entre vários processadores auxiliares contribuindo assim com o aumento

da velocidade.

Desempenho é normalmente um dos principais critérios adotados para guiar o projeto arquitetural de um SoC. Devido à complexidade e à heterogeneidade dos SoCs atuais, ferramentas de análise de desempenho para estágios iniciais de projeto têm atraído a atenção da comunidade de pesquisa. Nesse contexto, os trabalhos [74, 58, 30] apresentam mecanismos para analisar o desempenho de plataformas virtuais.

Em [74], Yeh et al. apresentam mecanismos de análise de desempenho e consumo de energia usando plataformas virtuais multi-core para aplicações multimídia. Enquanto que o desempenho é avaliado com a ajuda do *chip* equivalente, a análise do consumo de energia é feita considerando as técnicas de DVFS (*Dynamic Voltage and Frequency Scaling*) implementados especificamente para a plataforma ESL.

Já Oyamada et al. [58] propõem uma metodologia integrada para o projeto de sistemas e análise de desempenho. Essa metodologia envolve o uso de uma técnica de estimativa de desempenho de software baseada em redes neurais para seleção do processador adequado, seguido de um refinamento do ambiente de HW/SW baseado em *profiling* para criação de um protótipo virtual. Esse protótipo é então utilizado para realizar a análise de desempenho.

Por fim, no trabalho desenvolvido em [30] por Chen et al. é apresentado um framework de avaliação de desempenho e consumo de energia em nível de sistema. O framework não só possibilita a configuração de diversos parâmetros da arquitetura SoC em análise como também disponibiliza uma interface gráfica para visualização dos resultados.

Uma outra tendência dos trabalhos sobre análise de desempenho em plataformas virtuais está ligada à paralelização de software como apresentado pelos autores Chuang et al. em [31]. Este trabalho propõe uma plataforma virtual escalável multi-core para auxiliar projetistas no co-projeto de hardware e software. Uma aplicação paralela de um decodificador H.264 é utilizada nos experimentos, mostrando que técnicas de paralelismo de software são viáveis e conseguem melhorar o desempenho do sistema.

No fluxo atual de desenvolvimento de SoCs, a verificação representa o gargalo do projeto. Abordagens tradicionais não conseguem lidar com a complexidade de projetos atuais. Surge daí a necessidade de novas metodologias de verificação, melhorando a observação do comportamento do sistema e a controlabilidade do processo de verificação.

O trabalho desenvolvido por Chai et al. [28] apresenta um plataforma de simulação de NoC baseada em SystemC para exploração de topologias e algoritmos de roteamento. A plataforma desenvolvida possui uma rede do tipo malha ou torus de tamanho arbitrário e adota como padrão os algoritmos de roteamento `XY` para malha e `TXY` para torus. A verificação da arquitetura escolhida é feita através da simulação de diferentes padrões de tráfego de rede.

Já os autores Beltrame et al. em [19] apresentam uma plataforma virtual reflexiva

baseada na integração de SystemC e Python. A plataforma explora o conceito de reflexão proveniente de Python, permitindo a integração de componentes SystemC sem modificações no código fonte e provê observabilidade do seu estado interno. A plataforma oferece também controle da simulação e permite a avaliação de diferentes configurações de hardware/software para uma dada aplicação, facilitando assim a exploração do espaço de projeto.

Os trabalhos que têm como foco o desenvolvimento de plataformas virtuais são apresentados a seguir.

Os autores Han et al. de [39] propuseram uma plataforma virtual ARM que oferece aos desenvolvedores um ambiente de testes que é bastante similar à plataforma real em hardware. A plataforma foi projetada de forma a imitar o comportamento da plataforma de hardware `SYS-Lab5000 ARM`.

A plataforma virtual desenvolvida em [39] consiste em um simulador virtual, uma interface gráfica, um controlador de eventos de entrada, um temporizador e alguns dispositivos de E/S. O simulador virtual decodifica e executa as instruções ARM fornecidas enquanto interage com a interface gráfica na exibição dos resultados obtidos da simulação. A interface gráfica representa uma imagem da placa em hardware. Ela é utilizada para manipular eventos de entrada como interações do teclado e interruptores. O temporizador desenvolvido na plataforma virtual é utilizado em testes que envolvem interrupção, troca de contextos e simulações simples de sistemas operacionais. Os experimentos realizados restringem-se à verificação da conformidade de execução da plataforma virtual com a plataforma em hardware.

O trabalho desenvolvido em [39] por Han et al. é bastante útil quando se pretende obter informações precisas de desempenho ou consumo de energia visto que a simulação imita o comportamento da placa em hardware. Entretanto, a interface gráfica projetada dificulta a obtenção de informações além das que são visíveis no hardware. Além disso, o software utilizado nas simulações da plataforma está em baixo nível de abstração, restringindo a sua utilização a estágios mais avançados de desenvolvimento do projeto.

Já em [66] Silva et al. descrevem uma plataforma virtual para acelerar o desenvolvimento e teste de aplicações e de componentes de hardware para um MPSoC. O ambiente visa o desenvolvimento de aplicações embarcadas *multithread* de tempo real na linguagem Java. Além disso, permite avaliar o desempenho e os custos de energia da aplicação em desenvolvimento, tanto no nível de instruções para o processador quanto no nível arquitetural para o sistema de interconexão dos componentes da plataforma.

A infraestrutura de comunicação da plataforma SIMPLE (em inglês, *Simple Multiprocessor Platform Environment*) apresentada em [66] é baseada em uma NoC que conecta elementos de processamento (EPs). Esse trabalho utiliza processadores Java [36] de tempo real como EPs. Os EPs são todos iguais e implementam as APIs RTSJ e COM, dando

assim suporte a *multithread*, aplicações de tempo real e comunicação entre os componentes da rede. Isso permite aos projetistas de software programar aplicações diretamente na linguagem Java, utilizando APIs de alto nível de abstração. Uma aplicação de tempo real é utilizada para demonstrar as capacidades e resultados de latência e consumo de energia.

A desvantagem da estratégia descrita em [66] é que o projetista precisa aprender novas linguagens de descrição e APIs com as quais não está acostumado. Além disso, visto que boa parte das aplicações e dos IPs utilizados nas plataformas já foram projetados, existe a necessidade de se reescrever o código dos IPs para a incorporação no framework proposto.

Em [27] Ceng et al. apresentam uma plataforma virtual de alto nível chamada HVP que visa o desenvolvimento de software para MPSoC em estágios iniciais de projeto. O framework desenvolvido provê um simulador genérico construído sobre SystemC que faz uso de modelos abstratos de processadores e um SO abstrato incorporado a eles. A comunicação entre os modelos é feita através de memórias compartilhadas que podem ser controladas e acessadas pelos softwares através de uma API desenvolvida especificamente para a HVP.

Em [27], os softwares desenvolvidos pelos programadores são compilados em bibliotecas compartilhadas e carregados dinamicamente antes do início da simulação por ferramentas para geração de código para a HVP embutidas no framework. Adicionalmente, uma interface gráfica é fornecida para configurar a plataforma, controlar a execução da simulação, exibir estatísticas da execução e visualizar as informações gráficas e textuais produzidas. Um estudo de caso foi realizado com algumas plataformas MPSoC e mostra que o código desenvolvido na HVP pode ser reutilizado em diferentes plataformas. Além disso, a alta velocidade de simulação alcançada pela HVP agiliza o projeto do software da plataforma.

O trabalho desenvolvido por Ceng et al. mostra-se útil no desenvolvimento de software independente de plataforma e no início do projeto. Entretanto, características do hardware utilizado influenciam diretamente não só o software como também o desempenho da plataforma. Por ser uma plataforma muito abstrata, ela dificulta a obtenção de estimativas precisas de desempenho que podem afetar tanto o projeto do hardware quanto do software em desenvolvimento. Adicionalmente, a independência do software da plataforma possui a desvantagem de não focar nos serviços de comunicação e sincronização entre os recursos de hardware utilizados. Visto que esses tipos de problemas são comuns e difíceis de detectar, a HVP não traz grandes contribuições como uma ferramenta para verificação eficiente de plataformas.

No seu trabalho Bu et al. [23] apresentam uma plataforma virtual para desenvolvimento paralelo de hardware e software em estágios iniciais de projeto. A construção da plataforma é feita através de uma ferramenta genérica que lê arquivos de configuração (tanto da plataforma quanto de cada um dos IPs), compila os códigos fontes dos IPs em uma biblioteca compartilhada, carrega essa biblioteca na instanciação da plataforma e,

por fim, constrói a arquitetura da plataforma automaticamente. Todas essas tarefas são realizadas em tempo de execução.

Os autores de [23] também apresentam um sub-sistema para acelerar a verificação do hardware. Para cada processador na plataforma há um servidor de depuração associado que suporta GDB RSP (em inglês, *Remote Serial Protocol*). De acordo com o modo de configuração do processador, o mesmo pode executar no modo depuração GDB ou no modo normal. Quando em modo GDB, GDB ou outros depuradores que suportem RSP, podem conectar-se ao processador, através de um *socket* TCP, para fazer depuração em nível de código fonte.

A desvantagem do mecanismo de construção de plataforma automático apresentado por Bu et al. em [23] é que a leitura dos arquivos de configurações de cada IP e da plataforma, bem como a instanciação da plataforma e o carregamento da biblioteca compartilhada, ocorrem em tempo de execução, afetando assim o desempenho da plataforma e consequentemente a produtividade dos projetistas.

Em [60] Peng et al. apresentam uma plataforma virtual *dual-core* de co-simulação para validar paralelamente a funcionalidade de hardware e software. A plataforma é composta por componentes de hardware implementados em SystemC e dois processadores ARM que são emulados pelo QEMU [14] executando o software. O formato de requisição criado nesse trabalho transforma a transação original SystemC em uma transação TCP-IP que é enviada para o barramento do sistema. A comunicação entre os componentes é feita via *wrappers* que, além de transmitirem os dados de requisição ou resposta, também são responsáveis por sincronizar o tempo de simulação com o barramento do sistema.

Para lidar com o sistema de *threads* entre os diferentes cores o artigo em [60], desenvolve um controlador para o barramento do sistema utilizando *sockets* BSD para integrar diferentes transações de dados entre os simuladores (SystemC) e os emuladores (QEMU). A plataforma virtual *dual-core* foi verificada utilizando um modelo em SystemC de um codificador H.264/AVC e um decodificador H.264/AVC em software que é executado por um dos processadores ARM.

O trabalho desenvolvido por Peng et al. assim como em [66] exige do projetista aprender novos protocolos de comunicação aos quais não está acostumado a trabalhar. Adicionalmente, apesar da plataforma desenvolvida ter bastante potencial para a extração de diversas informações da simulação, o mesmo não é feito pelos autores, que se restringem à validação da funcionalidade em um estudo de caso limitado.

Já o trabalho desenvolvido em [75] por Yeh et al. apresenta uma plataforma virtual QEMU baseada em SystemC que dá suporte à modelagem de hardware utilizando a interface TLM 2.0. A plataforma proposta é capaz de executar um sistema operacional e possibilita a conexão de modelos de hardware, tal como um ISS, a um modelo de barramento. O desenvolvimento da plataforma virtual com suporte ao TLM 2.0 teve

como base um ISS conectado a um modelo de um controlador de acesso direto à memória (DMAC) utilizando o *Simple Bus* modelado na interface TLM 2.0. Foram feitas diversas adaptações nas interfaces do *Simple Bus* visto que algumas propriedades da modelagem em transação são mantidas internamente no QEMU.

A funcionalidade da plataforma apresentada em [75] foi verificada utilizando-a para inicializar um *kernel* Linux em uma plataforma virtual formada pelo *Simple Bus* modificado como o meio de interconexão entre um IA-ISS e um `ARM PrimeCell PL080 DMAC` modelado em SystemC conectados à uma sub-plataforma do QEMU. Estimativas de desempenho no nível de instruções tais como número de instruções executadas, acessos de memória, e operações de leitura e escrita do DMAC também foram extraídas durante a execução do sistema operacional.

A plataforma desenvolvida por Yeh et al. em [75] demonstra a importância e viabilidade da extração de informações a partir de simulações de plataformas virtuais utilizando interfaces. A análise das informações auxilia o projetista e permite identificar possíveis otimizações de desempenho.

Em [61], Pulka et al. apresentam uma plataforma virtual implementada em SystemC baseada no padrão de comunicação do barramento AMBA. A plataforma foi implementada como uma estrutura multi-core de diversos componentes comuns a sistemas embarcados. A estrutura permite uma modelagem flexível e simulação de várias aplicações compostas de vários componentes em cooperação.

No trabalho desenvolvido em [61] foram implementados quatro tipos diferentes de módulos: árbitro, decodificador, mestre e escravo. Cada um dos módulos possui uma funcionalidade e papel diferente no sistema. Além disso, foram desenvolvidos cinco canais de comunicação entre os vários componentes e cada canal possui um par de interfaces que cobrem todos os métodos necessários para lidar com as operações associadas àquele componente. Alguns experimentos foram feitos de forma a validar a plataforma e mostrar possíveis aplicações.

Tanto Pulka et al. em [61] quanto os outros trabalhos apresentados demostram a flexibilidade e a eficiência da simulação de modelos de um sistema completo em alto nível. De posse de uma plataforma virtual, é possível realizar não só exploração do espaço de projeto mas também analisar diversas métricas de desempenho, energia, etc. Dessa forma, o projetista ainda em estágios iniciais do projeto pode fazer escolhas que atendam às restrições de projeto.

Dentre os trabalhos, listados acima, que mais se assimilam à MediaBox existem diferentes pontos positivos e negativos. Essas diferentes características podem ser observadas na Tabela 5.1.

Na Tabela 5.1 as características e seus valores possuem os seguintes significados:

- Nível de abstração: corresponde à linguagem utilizada.

Tabela 5.1: Diferenças entre os trabalhos relacionados apresentados e a MediaBox.

Característica / Trabalho	Han et al. [39]	Silva et al. [66]	Ceng et al. [27]	Bu et al. [23]	Peng et al. [60]	Yeh et al. [75]	Pulka et al. [61]	MediaBox
Nível de abstração	RTL	Java e SystemC	C e SystemC	SystemC, RTL	SystemC	SystemC	SystemC RTL	C e SystemC
Novas linguagens / protocolos	não	sim	sim	não	sim	não	não	não
Obtenção de informações	não	sim	poucas	poucas	não	poucas	poucas	sim
Qualidade das informações	----	estimativas	imprecisas	precisas	----	precisas	precisas	estimativas
Execução em NoC	não	sim	não	não	não	não	não	sim
Integração de novos IPs	não	não	não	sim	sim	sim	sim	sim

- Novas linguagens/protocolos: corresponde à necessidade de aprendizado de novas linguagens, APIs ou protocolos aos quais projetistas não estão acostumados.
- Obtenção de informações: corresponde à disponibilidade de informação da plataforma desenvolvida.
- Qualidade das informações: corresponde à confiabilidade das informações. As informações, quando disponíveis, podem ser precisas, estimativas, ou imprecisas.
- Execução em NoC: corresponde à utilização de NoC.
- Integração de novos IPs: corresponde à facilidade de integração de novos IPs à plataforma.

5.2 Ferramentas Comerciais

O mercado de ferramentas para plataformas virtuais tem se desenvolvido bastante. Várias empresas, como a Synopsys e a Mentor, têm lançado no mercado ferramentas que facilitam a criação dessas plataformas compostas por modelos de hardware em nível de sistema.

A Synopsys provê algumas ferramentas para plataformas virtuais e desenvolvimento de software como o *System Studio* [67] e o *Platform Architect* [68]. Ambas as ferramentas dão suporte à construção de plataformas virtuais a partir de modelos de hardware em SystemC e ao desenvolvimento de software para a plataforma. Além disso, as ferramentas possibilitam a obtenção de estimativas de desempenho e a realização de exploração arquitetural.

OVP (*Open Virtual Platform*) [44], desenvolvida pela Imperas, provê modelos de plataformas virtuais rápidos e com precisão de instruções. OVP é composto por três componentes principais: APIs que permitem a modelagem em C de componentes de hardware, uma coleção de modelos de periféricos e processadores de código aberto, e o simulador OVPsim que executa esses modelos.

O *Vista Architect* da Mentor [52] provê um ambiente de projeto arquitetural e criação de plataformas de simulação que permite realizar a modelagem, a análise e a otimização de desempenho e consumo de energia em nível de transação. O *Vista Architect* possui uma interface gráfica para montagem e visualização da plataforma e cria automaticamente as interconexões TLM entre os IPs e o modelo de simulação da plataforma virtual.

Já o *Virtual Platform Studio* da Cadence [24] possibilita o desenvolvimento de software antes do hardware estar pronto, verificação funcional, análise de desempenho e otimizações utilizando plataformas virtuais. A ferramenta dispõe de uma biblioteca de IPs em SystemC assim como modelos de processadores de alto desempenho para o desenvolvimento de soluções de software reais.

Capítulo 6

Conclusões

Nesse capítulo fazemos algumas considerações finais a respeito da MediaBox assim como o impacto da utilização de uma NoC na plataforma. Em seguida, abordamos as dificuldades enfrentadas durante o desenvolvimento da plataforma e os trabalhos futuros decorrentes da experiência adquirida ao fim desse trabalho.

6.1 Considerações Finais

Arquiteturas padrões para os sistemas modernos consistem em SoCs com múltiplos processadores integrados em um único *chip* conhecidos como MPSoCs. Um MPSoC é o resultado de uma combinação de microprocessadores, memórias, barramentos, arquiteturas, padrões de comunicação, protocolos, interfaces e outros IPs. Essa complexidade inerente ao MPSoC torna seu projeto um grande desafio. Considerações como otimização de energia, sincronização, testabilidade e verificação são cruciais.

Aliado a essa complexidade está o fato de que não é possível realizar a verificação de um SoC sem a aplicação em software e muito menos desenvolver o software sem modelos de hardware. Nestas situações, é importante que os projetistas possam, assim que possível, começar a utilizar modelos do sistema completo nos quais subsistemas possam ser independentemente substituídos por modelos refinados e módulos IPs, de forma a haver uma validação contínua do sistema.

Neste contexto, o conceito de plataforma virtual tem sido utilizado para desenvolvimento paralelo de hardware e software. Uma plataforma virtual corresponde a um conjunto de modelos de hardware que são organizados para representar um sistema completo. Quando a plataforma virtual está pronta com os parâmetros determinados pelos projetistas de hardware, os engenheiros de software possuem o ambiente para desenvolver o software muito antes da plataforma real de hardware estar pronta.

Através de plataformas virtuais, projetistas podem analisar antecipadamente muitos

problemas de projeto em um sistema completo usando estimativas de consumo de energia, tráfego de barramento, uso de memória, eficiência dos periféricos e principalmente desempenho do sistema como um todo. Ao analisar essas métricas, eles podem tomar decisões sobre o tipo de unidades de processamento, escolher tamanho de memórias e de caches e tipos de interconexão e periféricos a serem incluídos no SoC. Essa análise os ajuda a determinar o custo-benefício no particionamento do hardware/software antes da integração final.

O objetivo deste projeto foi prover uma plataforma virtual de simulação completa chamada MediaBox. Essa plataforma fornece recursos para variados tipos de simulações nas quais os desenvolvedores podem exercitar diferentes aspectos e executar testes em um modelo abstrato de um SoC complexo.

Por ser descrita em ESL, a MediaBox permite simulações mais rápidas e facilita a produção de grandes quantidades de informação acerca do funcionamento e do desempenho da plataforma poupando assim tempo e esforço ao desenvolvedor. O relatório gerado pela MediaBox possui diversas informações úteis a um projetista e que realmente podem auxiliá-lo em decisões de projeto. Adicionalmente, o uso das possíveis configurações da MediaBox é facilitado pelo arquivo de configuração e pelo mapa de endereçamento que permitem explorar essas opções.

Os IPs desenvolvidos no contexto da plataforma, foram descritos de forma a facilitar seu reuso e por isso podem ser reutilizados em diversos projetos. Além disso, a heterogeneidade tanto dos processadores quanto dos IPs utilizados torna a MediaBox atraente para o desenvolvimento de outros projetos ligados não só a aplicações multimídia.

O uso de uma NoC como meio de interconexão mostrou-se eficaz. Com o grande tráfego existente entre os IPs, um barramento não apresentaria o desempenho desejado. A flexibilidade da NoC utilizada permite a utilização de diferentes topologias, tamanho e algoritmos, facilitando assim sua reutilização. Adicionalmente, tanto a estrutura de pacotes quanto as interfaces criadas mostraram-se válidas pois adaptaram a NoC para tráfegos de dados reais no contexto de um SoC e viabilizaram sua utilização em outros projetos.

O estudo de caso realizado demonstra que a MediaBox é uma boa solução para simular aplicações multimídias e para análise de desempenho. A flexibilidade provida pela MediaBox fornece aos usuários a oportunidade de conceber e testar múltiplas arquiteturas. A plataforma provê um conjunto rico de opções e de informações através das quais pode ser estudado o comportamento e os desempenhos de sistemas multiprocessados em um *chip*, tanto do ponto de vista do hardware quanto do software.

6.2 Trabalhos Futuros

Dentre as tendências de pesquisa apresentadas no capítulo de Trabalhos Relacionados, a MediaBox é uma boa candidata para pesquisas no contexto de aumento da velocidade da simulação, análise de desempenho e teste de novas metodologias de verificação. Adicionalmente, técnicas relacionadas à paralelização de software poderiam ser implementadas e testadas no contexto da plataforma desenvolvida. A aplicações de decodificação de áudio e vídeo são viáveis para tal fim pois trabalham em forma de *pipeline*, *frame* a *frame*.

Além disso, em plataformas complexas como a MediaBox, realizar a depuração é uma tarefa que exige tempo e esforço. Um possível trabalho futuro seria construir uma ferramenta de depuração de plataforma. Além disso, visto que a MediaBox corresponde a um sistema completo, ela pode servir como base para auxiliar na adaptação de ferramentas de depuração já existentes e na validação de novas.

Outra vantagem de ser um sistema completo é que a MediaBox pode ser utilizada para estudar problemas relacionados a fluxo de dados em plataformas ESL. A teoria já existente pode ser adaptada para o contexto de uma plataforma. Dessa forma, ela poderia ser usada para resolver problemas de concorrência e sincronização entre os IPs.

Adicionalmente, a MediaBox pode ser utilizada para exploração do espaço de projeto. Partindo da configuração inicial disponível da MediaBox, sua arquitetura poderia ser otimizada especificamente para aplicações multimídia. Parâmetros como tamanho do buffer da NoC e do DMA, configuração do algoritmo, dos nós e da topologia da NoC, tipo de processador utilizado, etc. poderiam ser configurados de forma a aumentar a eficiência do sistema e reduzir seu tempo de execução e consumo de energia.

De forma a tornar simulação mais rápida, a MediaBox poderia por fim ser adaptada para execução em hardware. Uma placa com vários núcleos poderia funcionar como a NoC. A lógica de envio de pacotes da NoC seria implementada junto a lógica de passagem de informações entre os núcleos como aqueles disponíveis em sistemas como o SCC (*Single-Chip Cloud Computer*) da Intel [48]. Cada IP seria alocado a um núcleo diferente e teria uma instância de interface similar à criada para a NoC para interpretação e conversão dos pacotes. O cenário de melhor caso apresentado neste trabalho seria um bom candidato para esse tipo de execução.

Referências Bibliográficas

[1] IEEE standard for verilog hardware description language. *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pages 1–560, 2006.

[2] IEEE standard VHDL language reference manual. *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pages 1–626, 2009.

[3] Accellera, 2012. Disponível online em: http://www.accellera.org/.

[4] Advanced Linux Sound Architecture (ALSA). Disponível online em: www.alsa-project.org, 2012.

[5] The Archc architecture description language. Disponível online em: www.archc.org, 2012.

[6] ARM processor architecture. Disponível online em: www.arm.com/products/processors/instruction-set-architectures/index.php, 2012.

[7] Design and Reuse, 2012. Disponível online em: http://www.design-reuse.com/.

[8] Fast Light Toolkit (FLTK). Disponível online em: www.fltk.org, 2012.

[9] FFMPEG. Disponível online em: http://ffmpeg.org/, 2012.

[10] HD-trailers. Disponível online em: http://www.hd-trailers.net/, 2012.

[11] HTCondor - High Throughput Computing. Disponível online em: http://research.cs.wisc.edu/htcondor, 2012.

[12] MIPS technologies: Processor cores. Disponível online em: www.mips.com/products/processor-cores/, 2012.

[13] Power architecture offerings. Disponível online em: www-01.ibm.com/chips/techlib/techlib.nsf/products/PowerPC, 2012.

[14] QEMU - open source processor emulator. Disponível online em: www.qemu.org, 2012.

[15] SystemC USB1.1 IP core. Disponível online em: www.opencores.org/project,usb11, 2012.

[16] Maman Abdurohman, Kuspriyanto K., Sarwono Sutikno, and Arif Sasongko. Transaction level modeling for early verification on embedded system design. In *Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science*, ICIS '09, pages 277–282, Washington, DC, USA, 2009. IEEE Computer Society.

[17] David Atienza, Federico Angiolini, Srinivasan Murali, Antonio Pullini, Luca Benini, and Giovanni De Micheli. Invited paper: Network-on-Chip design and synthesis outlook. *Integr. VLSI J.*, 41(3):340–359, May 2008.

[18] Rodolfo Azevedo, Sandro Rigo, Marcus Bartholomeu, Guido Araujo, Cristiano Araujo, and Edna Barros. The ArchC architecture description language and tools. *Int. J. Parallel Program.*, 33(5):453–484, October 2005.

[19] Giovanni Beltrame, Luca Fossati, and Donatella Sciuto. Resp: A nonintrusive transaction-level reflective MPSoC simulation platform for design space exploration. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28(12):1857–1869, December 2009.

[20] L. Benini and G. De Micheli. Networks on Chips: A new SoC paradigm. *Computer*, 35(1):70–78, jan 2002.

[21] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of Network-on-Chip. *ACM Comput. Surv.*, 38(1), June 2006.

[22] David C. Black, Jack Donovan, Bill Bunton, and Anna Keist. *SystemC from the Ground Up*. Springer, 2 edition, 2010.

[23] Yong-Hua Bu, Zhen-Zhong Tao, Ming-Ju Lei, Ching-Tung Wu, and Chi-Feng Wu. A configurable SystemC virtual platform for early software development and its subsystem for hardware verification. In *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*, pages 29–32, april 2010.

[24] Cadence. Cadence Virtual System Platform. Disponível online em: www.cadence.com/rl/Resources/datasheets/virtual_system_platform_ds.pdf, 2011.

[25] Luca P. Carloni, Fernando De Bernardinis, Claudio Pinello, Alberto L. Sangiovanni-Vincentelli, and Marco Sgroi. *Platform-based Design for Embedded Systems*, pages 1–26. CRC Press, 2005.

[26] Luigi Carro and Flávio Rech Wagner. *Sistemas Computacionais Embarcados*, chapter 2, pages 45–94. Jornadas de Atualização em Informática, Sociedade Brasileira de Computação, Campinas, 2003.

[27] Jianjiang Ceng, Weihua Sheng, Jeronimo Castrillon, Anastasia Stulova, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. A high-level virtual platform for early MPSoC software development. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '09, pages 11–20, New York, NY, USA, 2009. ACM.

[28] Song Chai, Chang Wu, Yubai Li, and Zhongming Yang. A NoC simulation and verification platform based on SystemC. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 3, pages 423–426, dec. 2008.

[29] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew J. McNelly, and Lee Todd. *Surviving the SOC Revolution: A Guide to Platform-based Design*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[30] Yi-Jung Chen, Ye-Jyun Lin, Chin-Chie Huang, Tzu-Ching Lin, Jaw-Wei Chi, and Chia-Lin Yang. TunableVP: A tunable virtual platform for easy SoC design space exploration. In *SoC Design Conference, 2008. ISOCC '08. International*, volume 01, pages I–250–I–251, nov. 2008.

[31] I-Yao Chuang, Tso-Yi Fan, Chi-Hung Lin, Chun-Nan Liu, and Jen-Chieh Yeh. HW/SW co-design for multi-core system on ESL virtual platform. In *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*, pages 1–4, april 2011.

[32] Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, and NEC Corporation. Universal Serial Bus specification revision 1.1. Disponível online em: esd.cs.ucr.edu/webres/usb11.pdf, 1998.

[33] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[34] Flávia de Oliveira Santos and André Luís Meneses Silva. Desenvolvimento de modelos de software para simulação de componentes DMA no contexto de sistemas embarcados. In *Anais do 18º Encontro de Iniciação Científica da UFS*. 2008.

[35] P. Deitel and H.M. Deitel. *C++ How to Program*. How to program series. Pearson/Prentice Hall, 5 edition, 2005.

[36] P. Deitel and H.M. Deitel. *Java How to Program*. How to program series. Pearson/Prentice Hall, 5 edition, 2005.

[37] Frank Ghenassia. *Transaction Level Modeling with SystemC*. Springer, 2006.

[38] Stefan Valentin Gheorghita, Twan Basten, and Henk Corporaal. Application scenarios in streaming-oriented embedded-system design. *IEEE Des. Test*, 25(6):581–589, November 2008.

[39] A.H. Han, Young-Si Hwang, Young-Ho An, So-Jin Lee, and Ki-Seok Chung. Virtual ARM platform for embedded system developers. In *Audio, Language and Image Processing, 2008. ICALIP 2008. International Conference on*, pages 586–592, july 2008.

[40] Anssi Haverinen, Maxime Leclercq, Norman Weyrich, and Drew Wingard. SystemC based SoC communication modeling for the OCP protocol. Technical report, OCP-IP, 2002.

[41] Zhe-Mao Hsu, Jen-Chieh Yeh, and I-Yao Chuang. An accurate system architecture refinement methodology with mixed abstraction-level virtual platform. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 568–573, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[42] IBM. The coreconnect bus architecture. White Paper, 2012.

[43] ISO MPEG Audio Subgroup Software Simulation Group IMASSSG. dist10. Disponível online em: ftp.tnt.uni-hannover.de/pub/MPEG/audio/mpeg2/software/technical_report/dist10.tar.gz, 1996.

[44] Imperas. Open Virtual Platforms. Disponível online em: www.ovpworld.org/, 2012.

[45] Axel Jantsch and Hannu Tenhunen. *Networks on Chip*. Kluwer Academic Publishers, 2003.

[46] K. Keutzer, A.R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(12):1523–1543, dec 2000.

[47] Steve Kilts. *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Wiley-IEEE Press, 2007.

[48] Intel Labs. Single-chip Cloud Computing platform overview. Disponível online em: www.intel.com/content/www/us/en/research/intel-labs-single-chip-platform-overview-paper.html, 2010.

[49] Kuen-Jong Lee, Chin-Yao Chang, and I-Jou Chen. EPIDETOX: An ESL platform for integrated circuit design and tool exploration. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*, pages 381–384, oct. 2011.

[50] Kuen-Huei Lin, Siao-Jie Cai, and Chung-Yang Huang. Speeding up SoC virtual platform simulation by data-dependency-aware synchronization and scheduling. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 143–148, jan. 2010.

[51] Grant Martin, Brian Bailey, and Andrew Piziali. *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. Morgan Kaufmann Publishers Inc., 2007.

[52] Mentor. Vista Architect. Disponível online em: www.mentor.com/esl/vista/upload/vista-architect-ds.pdf, 2009.

[53] Fernando Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost. Hermes: An infrastructure for low area overhead packet-switching networks on chip. *Integr. VLSI J.*, 38(1):69–93, October 2004.

[54] Edson Ifarraguirre Moreno. Modelagem, descrição e validação de redes intrachip no nível de transação. Master's thesis, PPGCC - FACIN - PUCRS, Disponível online em: http://www.inf.pucrs.br/~calazans/publications/Edson_MScDiss.pdf, 2004.

[55] MPEG Software Simulations Group MSSG. MPEG-2 decoder. Disponível online em: euler.slu.edu/~fritts/mediabench/mb2/mediabench2_video/mpeg2dec/mpeg2vidcodec_v12.tar.gz, 1996.

[56] R.S. Nikhil. ESL flows are enabled by high-level synthesis with universality. In *High Level Design Validation and Test Workshop (HLDVT), 2010 IEEE International*, page 137, june 2010.

[57] Robert Oshana. *DSP Software Development Techniques for Embedded and Real-Time Systems*. Elsevier, 2006.

[58] M. Oyamada, F. R. Wagner, M. Bonaciu, W. Cesario, and A. Jerraya. Software performance estimation in MPSoC design. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, ASP-DAC '07, pages 38–43, Washington, DC, USA, 2007. IEEE Computer Society.

[59] R.P. Paul. *Sparc Architecture, Assembly Language Programming, and C.* Prentice Hall, 2000.

[60] Cheng-Shiuan Peng, Li-Chuan Chang, Chih-Hung Kuo, and Bin-Da Liu. Dual-core virtual platform with QEMU and SystemC. In *Next-Generation Electronics (ISNE), 2010 International Symposium on*, pages 69–72, nov. 2010.

[61] A. Pulka, L. Golly, and A. Milik. SystemC hardware-software design and simulation platform based on AMBA bus. In *Mixed Design of Integrated Circuits and Systems (MIXDES), 2011 Proceedings of the 18th International Conference*, pages 644–649, june 2011.

[62] Wei Qin, Joseph D'Errico, and Xinping Zhu. A multiprocessing approach to accelerate retargetable and portable dynamic-compiled instruction-set simulation. In *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, CODES+ISSS, pages 193–198, New York, NY, USA, 2006. ACM.

[63] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *Design Test of Computers, IEEE*, 18(6):23–33, nov/dec 2001.

[64] Luiz Santos, Sandro Rigo, Rodolfo Azevedo, and Guido Araujo. Electronic system level design. In Sandro Rigo, Rodolfo Azevedo, and Luiz Santos, editors, *Electronic System Level Design*, pages 3–10. Springer Netherlands, 2011.

[65] Gunar Schirner and Rainer Dömer. Quantitative analysis of the speed/accuracy trade-off in transaction level modeling. *ACM Trans. Embed. Comput. Syst.*, 8(1):4:1–4:29, January 2009.

[66] Jr Elias T. Silva, Daniel Barcelos, Flávio R. Wagner, and Carlos E. Pereira. A virtual platform for multiprocessor real-time embedded systems. In *Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems*, JTRES '08, pages 31–37, New York, NY, USA, 2008. ACM.

[67] Synopsys. System Studio. Disponível online em: www.synopsys.com/apps/docs/pdfs/sld/system_studio_ds.pdf, 2010.

[68] Synopsys. Platform Architect. Disponível online em: www.synopsys.com/apps/docs/pdfs/sld/platform_architect_ds.pdf, 2012.

[69] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall international editions. Prentice Hall, 2 edition, 2001.

[70] Andrew S. Tanenbaum. *Computer Networks*. Elsevier, 4 edition, 2003.

[71] M. Thompson, H. Nikolov, T. Stefanov, A.D. Pimentel, C. Erbas, S. Polstra, and E.F. Deprettere. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, pages 9–14, 30 2007-oct. 3 2007.

[72] Frank Vahid and Tony Givardis. *Embedded System Design: A Unified Hardware/Software Introduction*. Wiley, 2001.

[73] Chih-Chyau Yang, Hui-Ming Lin, Shih-Lun Chen, Tien-Ching Wang, Jun-Jie Zhu, Chien-Ming Wu, Chun-Ming Huang, and Chin-Long Wey. A configurable prototyping platform for Multi-project System-on-a-Chip. In *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*, pages 1–4, april 2011.

[74] Jen-Chieh Yeh, Kung-Ming Ji, Shing-Wu Tung, and Shau-Yin Tseng. Heterogeneous multi-core SoC implementation with system-level design methodology. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 851–856, sept. 2011.

[75] Tse-Chen Yeh, Zin-Yuan Lin, and Ming-Chao Chiang. Enabling TLM-2.0 interface on QEMU and SystemC-based virtual platform. In *IC Design Technology (ICICDT), 2011 IEEE International Conference on*, pages 1–4, may 2011.

[76] Cesar A. Zeferino, Márcio E. Kreutz, Luigi Carro, and Altamiro A. Susin. A study on communication issues for Systems-on-Chip. In *Proceedings of the 15th symposium on Integrated circuits and systems design*, SBCCI '02, pages 121–, Washington, DC, USA, 2002. IEEE Computer Society.

Apêndice A

Parâmetros de Configuração e Resultados da MediaBox

A.1 Arquivo de Configuração

O arquivo de configuração da plataforma com informações das faixas de endereçamento assim como os parâmetros configuráveis pode ser visto abaixo.

```
#define MB_PPC01_BASE_ADDRESS     0x000000
#define MB_PPC01_END_ADDRESS      0x4FFFFF

#define MB_USB_BASE_ADDRESS       0x700000
#define MB_USB_END_ADDRESS        0x7FFFFF

#define MB_MPEG2_BASE_ADDRESS     0x000000
#define MB_MPEG2_END_ADDRESS      0x4FFFFF

#define MB_MP3_BASE_ADDRESS       0x000000
#define MB_MP3_END_ADDRESS        0x4FFFFF

#define MB_MEMORY01_BASE_ADDRESS  0x800000
#define MB_MEMORY01_END_ADDRESS   0x82FFFF

#define MB_MEMORY02_BASE_ADDRESS  0x830000
#define MB_MEMORY02_END_ADDRESS   0x85FFFF
```

```
#define MB_AUDIO_OUT_BASE_ADDRESS  0x600000
#define MB_AUDIO_OUT_END_ADDRESS   0x6FFFFF

#define MB_MEMORY03_BASE_ADDRESS   0x860000
#define MB_MEMORY03_END_ADDRESS    0x8FFFFF

#define MB_VIDEO_OUT_BASE_ADDRESS  0x600000
#define MB_VIDEO_OUT_END_ADDRESS   0x6FFFFF

#define MB_REQ_INFO_BASE_ADDRESS   0x900000
#define MB_REQ_INFO_END_ADDRESS    0x9FFFFF

#define MB_MPEG2_INFO_BASE_ADDRESS 0x900000
#define MB_MPEG2_INFO_END_ADDRESS  0x9000FF

#define MB_VIDEO_OUT_INFO_BASE_ADDRESS  0x900100
#define MB_VIDEO_OUT_INFO_END_ADDRESS   0x9001FF

// Endereço dos registradores internos dos DMAs
#define MB_DMA_CH00        0x700000
#define MB_DMA_CH01        0x780000
#define MB_DMA_START       0x0
#define MB_DMA_READADD     0x4
#define MB_DMA_WRITEADD    0x8
#define MB_DMA_LENGTH      0xC
#define MB_DMA_INDSIZE     0x10
#define MB_DMA_INTERRUPT   0x14
#define MB_DMA_END_MODE    0x18
#define MB_DMA_READPERIF   0x1C
#define MB_DMA_WRITEPERIF  0x20
#define MB_DMA_RESET       0x24
#define MB_DMA_TRANSF_OK   0x28
#define MB_DMA_END_TRANS   0x2C
#define MB_DMA_DATA_COUNT  0x30

// Registradores de controle do mp3
#define MB_MP3_CONTROL              0x200000
#define MB_MP3_SHARED_MEMORY_START  0x300000
```

```
#define MB_MP3_SHARED_MEMORY_END    0x5FFFFF
#define MB_MP3_SHARED_MEMORY_OFFSET 0xC0000

// Registradores de controle do ppc_audio
#define MB_PPC01_CONTROL 0x500000;

//Registradores de controle do mpeg2
#define MB_USB_MPEG2_START  0x500000
#define MB_USB_MPEG2_END    0x9FFFFF

// Registradores de controle do audio_out
#define MB_AUDIO_OUT_CHANNELS_INFO   0x38
#define MB_AUDIO_OUT_DATA_RATE_INFO  0x3C
#define MB_AUDIO_OUT_CONTROL_INFO    0x40

// Registradores de controle do video_out
#define MB_VIDEO_OUT_DATA_RATE_INFO  0x38
#define MB_VIDEO_OUT_CONTROL_INFO    0x3c

//Uso do USB
#define MB_USB_MP3    "yes"
#define MB_USB_MPEG2  "yes"

//Uso do Player
#define MB_PLAYER_MP3   "yes"
#define MB_PLAYER_MPEG2 "yes"

//Exibir Informações
#define MB_DISPLAY_INFO "yes"

#define MB_AUDIO_ENABLED  "no"
#define MB_VIDEO_ENABLED  "no"
```

A.2 Tabela de Configuração do MP3

Os dados de configuração que são enviados pelo PowerPC ao MP3 para decodificação dos dados `mp3` podem ser vistos a seguir.

Tabela A.1: Registradores da área de memória compartilhada do MP3.

Nome	Função	Endereço
mp3_control	Registrador de controle de comunicação entre o PowerPC e o MP3	0x200000
layers	Arquivo fonte MP3 - camada de codificação	0x200004
error_protection	Arquivo fonte MP3 - código de proteção de erros	0x200008
crc_error_count	Contador de erros CRC ocorridos durante a decodificação	0x20000C
total_error_count	Total de erros ocorridos durante a decodificação	0x200010
topSb	Formato do arquivo de saída	0x200014
stereo	Quantidade de canais do áudio do arquivo fonte MP3	0x200018
done	Variável de controle de fim de dados do arquivo fonte MP3	0x20001C
Max_gr	Variável de controle do tipo de codificação do arquivo fonte MP3	0x200020
old_crc	CRC dos dados antes da decodificação	0x200024
frameNum	Número do frame sendo decodificado	0x200028
bitsPerSlot	Quantidade de bits por slot de dados	0x20002C
samplesPerFrame	Quantidade de amostras de áudio por frame	0x200030
sample_frames	Quantidade de amostras de áudio processadas	0x200034
bs.buf_size	Buffer de entrada - tamanho	0x200038
bs.totbit	Buffer de entrada - contador de bits processados do fluxo de áudio	0x20003C
bs.buf_byte_idx	Buffer de entrada - apontador para o primeiro byte	0x200040

bs.buf_bit_idx	Buffer de entrada - apontador para o primeiro bit do primeiro byte	0x200044
bs.mode	Buffer de entrada - configuração de abertura do arquivo fonte MP3	0x200048
bs.eob	Buffer de entrada - Índice de fim do buffer	0x20004C
bs.eobs	Buffer de entrada - Índice de fim dos dados do arquivo fonte MP3	0x200050
bs.format	Buffer de entrada - Formato do arquivo fonte MP3	0x200054
bs.new_data	Buffer de entrada - Flag de notificação de novos dados	0x200055
bs.buf[]	Buffer de entrada - Dados de áudio do arquivo fonte MP3	0x200059
header.version	Cabeçalho do frame - versão do MPEG	0x2040EB
header.lay	Cabeçalho do frame - camada de codificação do arquivo fonte MP3	0x2040EF
header.error_protection	Cabeçalho do frame - código de proteção de erros do arquivo fonte MP3	0x2040F3
header.bitrate_index	Cabeçalho do frame - índice da taxa de bits	0x2040F7
header.sampling_frequency	Cabeçalho do frame - frequência da amostragem de áudio	0x2040FB
header.padding	Cabeçalho do frame - Bit de *padding*	0x2040FF
header.extension	Cabeçalho do frame - Bit privado	0x204103
header.mode	Cabeçalho do frame - Modo do canal	0x204107
header.mode_ext	Cabeçalho do frame - Extensão do modo do canal	0x20410B
header.copyright	Cabeçalho do frame - Copyright do áudio	0x20410F
header.original	Cabeçalho do frame - Originalidade do frame	0x204113
header.emphasis	Cabeçalho do frame - indicação da ênfase a ser usada	0x204117
frame.actual_mode	Frame - modo do canal	0x20411B
frame.alloc[]	Frame - tabela de alocação de bits	0x20411F
frame.tab_num	Frame - número da tabela carregada	0x205E2F

frame.stereo	Frame - quantidade de canais	0x205E33
frame.jsbound	Frame - primeira banda de codificação de estéreo	0x205E37
frame.sblimit	Frame - número total de sub-bandas	0x205E3B

A.3 Coleta de Informações

Um exemplo dos dados gerados pela NoC para o nó 00 pode ser visto abaixo.

[NOC]: Node 0x0 info ---------- start
[NOC]: Pacote de informação recebido. Imprimindo dados...
[NOC]: Latência Total = 747738267
[NOC]: Tentativas totais = 373869001
[NOC]: Quantidade Total de Pacotes = 373869001
[NOC]: Pacotes do/para o IP local = 373869001
[NOC]: Latência, Tentativas e Pacotes entre os nós:

Latência	|00|	|01|	|02|	|10|	|11|	|12|	|20|	|21|	|22|
|00|	0	1677084	0	372294160	2	0	0	0	0
|01|	1685235	0	0	0	0	0	0	0	0
|02|	0	0	0	0	0	0	0	0	0
|10|	372081786	0	0	0	0	0	0	0	0
|11|	0	0	0	0	0	0	0	0	0
|12|	0	0	0	0	0	0	0	0	0
|20|	0	0	0	0	0	0	0	0	0
|21|	0	0	0	0	0	0	0	0	0
|22|	0	0	0	0	0	0	0	0	0

Tentativas	|00|	|01|	|02|	|10|	|11|	|12|	|20|	|21|	|22|
|00|	0	838657	0	186147080	1	0	0	0	0
|01|	842370	0	0	0	0	0	0	0	0
|02|	0	0	0	0	0	0	0	0	0
|10|	186040893	0	0	0	0	0	0	0	0
|11|	0	0	0	0	0	0	0	0	0
|12|	0	0	0	0	0	0	0	0	0
|20|	0	0	0	0	0	0	0	0	0
|21|	0	0	0	0	0	0	0	0	0
|22|	0	0	0	0	0	0	0	0	0

Pacotes	|00|	|01|	|02|	|10|	|11|	|12|	|20|	|21|	|22|
|00|	0	838657	0	186147080	1	0	0	0	0
|01|	842370	0	0	0	0	0	0	0	0
|02|	0	0	0	0	0	0	0	0	0
|10|	186040893	0	0	0	0	0	0	0	0
|11|	0	0	0	0	0	0	0	0	0
|12|	0	0	0	0	0	0	0	0	0
|20|	0	0	0	0	0	0	0	0	0
|21|	0	0	0	0	0	0	0	0	0
|22|	0	0	0	0	0	0	0	0	0

[NOC]: Latência, Tentativas e Pacotes por canais:
--- East (0):
Latency = 372294160
Attempts = 186147080
Packets = 186147080
--- West (1):
Latency = 0
Attempts = 0
Packets = 0
--- North (2):
Latency = 1677086
Attempts = 838658
Packets = 838658
--- South (3):
Latency = 0
Attempts = 0
Packets = 0
--- Local (0):
Latency = 373767021
Attempts = 186883263
Packets = 186883263
[NOC]: Node 0x0 info ---------- end