Um *framework* de desenvolvimento de plataformas e um mecanismo de depuração baseado em reflexão computacional

Bruno de C. Albertini

Dissertação de Mestrado

Um *framework* de desenvolvimento de plataformas e um mecanismo de depuração baseado em reflexão computacional

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Bruno de C. Albertini e aprovada pela Banca Examinadora.

Campinas, 16 de março de 2007.

Sandro Rigo (Orientador)

Guido Araújo (Co-orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Miriam Cristina Alves – CRB8a / 5094

Albertini, Bruno de Carvalho

AL14u Um *framework* de desenvolvimento de plataformas e um mecanismo de depuração baseado em reflexão computacional /Bruno de Carvalho Albertini -- Campinas, [S.P. :s.n.], 2007.

Orientadores : Sandro Rigo; Guido Araújo

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Hardware - Arquitetura. 2. Sistemas embutidos de computador. 3. Simulação (Computadores). 4. Sistemas e Computação. I. Rigo, Sandro. II. Araújo, Guido Costa Souza de. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Título em inglês: A platform development framework and a debugging mechanism based on computational reflection

Palavras-chave em inglês (Keywords): 1. Hardware (Architecture). 2. Embedded computer systems. 3. Simulation (Computers). 4. Computer systems.

Área de concentração: Arquitetura de computadores

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Sandro Rigo (IC-Unicamp)
Prof. Dr. Cristiano Coelho Araújo (CIn-UFPE)
Prof. Dr. Ricardo Pannain (IC-Unicamp)

Data da defesa: 23/03/2007

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 23 de março de 2007, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Cristiano Coêlho de Araújo
CIN - UFPE.

Prof. Dr. Ricardo Pannain
IC – UNICAMP.

Prof. Dr. Sandro Rigo
IC – UNICAMP.

Instituto de Computação
Universidade Estadual de Campinas

Um *framework* de desenvolvimento de plataformas e um mecanismo de depuração baseado em reflexão computacional

Bruno de C. Albertini[1]

Fevereiro de 2007

Banca Examinadora:

- Sandro Rigo (Orientador)
- Guido Araújo (Co-orientador)
- Cristiano Coelho Araújo
  Centro de Informática – CIn – UFPE
- Ricardo Pannain
  Instituto de Computação – IC – Unicamp
- Rodolfo Azevedo
  Instituto de Computação – IC – Unicamp (Suplente)

[1]Suporte financeiro de: Bolsa CNPq (132916/2005-3) 2005–2007, através do PNM – Programa Nacional de Microeletrônica.

Resumo

Com o passar do anos, os sistemas digitais estão se tornando cada vez mais complexos, aglutinando processadores de propósito geral com *hardware* e barramentos especializados em uma única pastilha de silício, devido às restrições de consumo, espaço e desempenho. Para contornar esta complexidade e o curto *time-to-market*, os projetistas estão adotando novas metodologias de descrição de *hardware* em alto nível baseadas em linguagens de descrição de sistemas como o SystemC. Estas descrições permitem o desenvolvimento e o teste do *software* cedo, sobre um ambiente simulado, e são mais rápidas de escrever e simular que as descrições em baixo nível. A desvantagem é a perda da precisão da simulação no que diz respeito aos ciclos de *clock*, que pode ser ignorada nas fases iniciais de projeto.

O ArchC é um projeto do LSC que tem como alvo a geração de simuladores de conjuntos de instruções e outras ferramentas a partir de modelos descritos em uma linguagem similar a SystemC. Os simuladores gerados são compatíveis com SystemC e podem ser compilados com ferramentas gratuitas como GCC. Seguindo os passos da indústria, ele suporta descrições de alto nível com comunicação por chamada de funções (TLM – Modelagem em nível de transações[2]) desde a versão 2.0.

Um problema comum quando se está desenvolvendo *hardware* especializado usando linguagens de descrição de alto nível é a depuração. A utilização das ferramentas existentes como o GDB (GNU *Debugger*) não é trivial dado que a biblioteca SystemC passa a fazer parte do simulador quando este é compilado. Propomos uma metodologia de depuração baseada em reflexão computacional de módulos SystemC para gerar dicionários que alimentam um módulo capaz de inspecionar e alterar outros módulos em tempo de execução.

No presente trabalho, apresentaremos a ARP, a plataforma de referência do ArchC. Seu público alvo são os arquitetos de projetos baseados em plataformas, fornecendo um ambiente para o desenvolvimento de plataformas utilizando simuladores ArchC e os novos usuários, introduzindo o protocolo de comunicação do ArchC, o SystemC e as metodologias relacionadas ao projeto de plataformas.

[2]Do inglês: *Transaction Level Modeling.*

Abstract

Digital systems are becoming more and more complex through the years, putting general purpose processors together with specialized hardware and buses into the same silicon die, due to power, area and performance constraints. In order to deal with this complexity and a short time-to-market, designers are adopting high level hardware descriptions, based on languages such as SystemC. Those descriptions permit early software development and test under a simulated environment, and are also faster to be coded and simulated than low level descriptions. The tradeoff is the loss of simulation precision regarding clock cycles, that can be ignored in early project phases.

ArchC is an architecture description language aiming retargetable instruction set simulator generation described in a SystemC like language. The generated simulator is full SystemC compatible and can be compiled with free available tools, as GNU GCC. Following industry path, it supports high level descriptions with Transaction Level Modeling (TLM) communication capabilities since version 2.0.

A common problem when developing specialized hardware using high level description languages is debugging. The use of existing tools like GDB (GNU Debugger) is not straightforward since SystemC library becames part of the executable simulator. We propose a new platform debugging methodology based on computational reflection of SystemC modules to generate a dictionary. This dictionary feeds a special SystemC module capable of inspecting and changing attributes of platform modules at run time.

In the present work, the ArchC Reference Platform is introduced. It aims the platform based architects, supplying a framework for platform design using ArchC simulators, introducing ArchC communication protocol, SystemC and platform design methodologies.

Agradecimentos

Ao meu orientador Sandro Rigo pela paciência dispensada em horas de discussão sobre os rumos dos projetos e ao meu co-orientador Guido Araújo pelas sugestões norteantes.

Ao professor Rodolfo Azevedo por aplicar em sala de aula os programas desenvolvidos e pelo retorno obtido de suas experiências na utilização do ambiente como plataforma de ensino.

A todas as pessoas que utilizaram as ferramentas desenvolvidas, direta ou indiretamente, pelas críticas e sugestões cruciais para a melhora das mesmas, em especial aos desenvolvedores do *Plataform Designer* da UFPE, coordenados pelo professor Cristiano Araújo.

Aos colegas do LSC, pela inestimável troca de conhecimentos proporcionada pelo convívio diário, em especial Klein, Baldas, Bartho, Sigrist e Borin, e ao professor Paulo Ducatte por sempre solucionar meus problemas burocráticos.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) por fomentar o projeto e a bolsa de mestrado a mim concedida através do Programa Nacional de Micro-eletrônica (PNM).

Sumário

Lista de Figuras

Capítulo 1

Introdução

O *roadmap* [25] do ITRS (*International Technology Roadmap for Semiconductors*) prevê que em breve 70% dos projetos ASIC (*Application Specific Integrated Circuit*) incluirão ao menos um processador programável embutido, o que significa que a maioria dos ASICs serão SoCs (*System-on-Chip*). A crescente complexidade dos SoCs aliada à demanda de processamento das aplicações apontam para um número cada vez maior de SoCs *multicores* ou multiprocessados.

O paradigma atual de desenvolvimento de SoCs traz consigo diversas complicações devido à crescente complexidade dos mesmos, aumentando o tempo de projeto e conseqüentemente piorando o *time-to-market*. O projeto em RTL (*Register Transfer Level*), cujo fluxo de modelagem é complexo e suscetível a erros, tornou-se demorado e custoso a ponto de inviabilizar uma descrição não automatizada. Para ajudar os projetistas a lidar com o crescente número de processadores nas plataformas, surgiram nos últimos anos diversas ADLs (*Architecture Description Language*), como nML[24], LISA [48], EXPRESSION [31] e ArchC [4]. Estas linguagens movem os desenvolvedores um nível de abstração acima das linguagens de descrição de *hardware* convencionais, como VHDL (*Very High speed Hardware Description Language*), AHDL (*Altera Hardware Description Language*) ou Verilog, permitindo maior flexibilidade na administração dos projetos e facilitando as fases de verificação e teste.

O desafio de implementar, testar e validar os projetos força os projetistas de *hardware* a utilizar plataformas heterogêneas, nem sempre compatíveis ou estáveis. A comunicação *onchip* não foge à regra: vários tipos de barramentos e interconexões estão disponíveis para lidar com o crescente tráfego interno de dados [46]. Em um futuro próximo, poderemos ter NoCs (*Network on Chip*) inteiras dentro de uma pastilha, inclusive com QoS (*Quality of Service*) [6].

O método tradicional de projeto parte de uma descrição textual da arquitetura, seguido de uma implementação das partes *hardware* e *software*, quase sempre desconexas.

Na etapa final, os processadores, barramentos, memórias e blocos especializados são descritos em uma implementação HDL (*Hardware Description Language*), geralmente sintetizável. A alta complexidade devida à descrição detalhada do *hardware* e a lentidão da simulação neste nível decorrente do alto detalhamento da descrição não permitem uma troca intensiva das partes de *hardware* para avaliação, e as partes de *software* geralmente só são avaliadas quando o silício já está pronto. Devido a esta brecha no desenvolvimento, os SoCs são muitas vezes superestimados ou não alcançam os requisitos de projeto, principalmente no que diz respeito ao desempenho. O desempenho do *software*, como o tempo de resposta ou a carga da CPU, são muito difíceis de serem analisados sem um simulador CA (*Cycle Accurate*).

O desempenho do *software* é diretamente afetado pela arquitetura de comunicação, portanto a simulação de um processador isolado que não leve em conta a utilização do barramento e da memória é praticamente inútil no desenvolvimento de SoCs *multicores* ou multiprocessados. Esta avaliação deve ser feita o quanto antes para evitar os altos custos decorrentes da troca da especificação da arquitetura.

Com o SystemC [47] se firmando como padrão entre os desenvolvedores de *hardware*, está claro que qualquer ferramenta que possa ser desenvolvida deve ser uma extensão ou ser compatível com descrições em SystemC.

O trabalho que apresentarei foi desenvolvido no LSC [20] (Laboratório de Sistemas Computacionais), sendo baseado em uma ADL já estabelecida e firmada, o ArchC [19], desenvolvido e mantido pelo próprio LSC. Este trabalho consiste no desenvolvimento de um *framework* voltado para a criação de modelos de simulação de plataformas que utilizem simuladores ArchC, multiprocessadas ou não, com os seguintes objetivos:

- Organizar em forma de repositório os elementos utilizados para a construção de plataformas, sejam *software* ou *hardware*;
- Facilitar e agilizar a composição de modelos de plataformas a partir dos componentes contidos no repositório;
- Oferecer um suporte adequado e um modelo de referência em construção de plataformas aos usuários do ArchC, experientes ou novatos.

Além disso, com base neste *framework* foi desenvolvida uma nova metodologia de depuração para os modelos de plataformas descritos em alto nível de abstração usando o conceito de reflexão computacional.

Esta dissertação está assim organizada:

Capítulo 2 Há diversas ferramentas comerciais disponíveis para EDA [1]. Este capítulo faz um apanhado das principais e uma relação dos trabalhos acadêmicos relacionados.

[1] Automação de projeto eletrônico, do inglês: *Electronic Design Automation*.

Capítulo 3 Uma introdução ao desenvolvimento de plataformas usando ARP [2] e ArchC, com definições das ferramentas e metodologias adotadas.

Capítulo 4 O conceito de reflexão computacional e como ele foi utilizado para depuração.

Capítulo 5 Os principais resultados obtidos e uma descrição detalhada das plataformas disponíveis publicamente.

Capítulo 6 Sugestões para a continuidade do projeto e as conclusões que puderam ser feitas até o momento.

Os apêndices contêm uma breve descrição da simbologia TLM utilizada nesta dissertação e em toda a documentação que acompanha os pacotes da ARP e a caracterização das máquinas utilizadas nos testes de desempenho, nesta ordem.

[2]Plataforma de referência do ArchC, do original em inglês: *ArchC Reference Platform*.

Capítulo 2

Trabalhos Relacionados

Esta seção reúne os trabalhos relacionados ao desenvolvimento de plataformas que mais se destacaram. Os principais trabalhos nesta área constituem produtos de grandes empresas de EDA, motivo pelo qual várias delas serão citadas. Procurei concentrar trabalhos relacionados com descrições funcionais, mas incluí algumas abordagens de mais baixo nível por conterem características muito próximas ou relevantes ao trabalho desenvolvido durante o mestrado. Ressalto que as características das ferramentas comerciais foram avaliadas com base em relatos de outros usuários, incluindo manuais do próprio fabricante. Algumas das ferramentas foram avaliadas pelo autor sem critério científico, motivo pelo qual nenhum estudo comparativo foi feito.

Na área de depuração de IPs (Propriedade Intelectual[1]), não há referências a tentativas anteriores de utilização de reflexão computacional, apesar de existirem outras abordagens.

Aos leitores não familiarizados com os termos relativos a área, sugerimos uma leitura prévia sobre TLM [11] e SystemC [47].

2.1 Ferramentas comerciais

CoWare

A CoWare [17] é uma empresa de ESL localizada em San Jose, Califórnia especializada em ferramentas de suporte ao projetista.

Possui um *software* chamado de *Plataform Architect* para captura gŕafica de um projeto estilo arraste e solte[2]. A ferramenta suporta SystemC, inclusive TLM. Suporta cosimulação de diversos níveis de abstração na própria ferramenta, permitindo que o desenvolvimento dos diversos módulos seja feito em separado. A geração dos adaptadores entre

[1] Do inglês: *Intelectual Property.*
[2] Do inglês: *drag-and-drop.*

os níveis de abstração (e.g. SystemC funcional e VHDL) é feita automaticamente.

A biblioteca de IPs é chamada *CoWare Model Library*. Possui IPs variados, mas concentrados nas plataformas ARM/AMBA e MIPS/OCP. Os IPs possuem vários níveis de abstração, permitindo que um determinado IP seja avaliado com precisão usando uma descrição detalhada mas mantendo o restante da plataforma em descrições de mais alto nível, permitindo maior velocidade de simulação. A maioria dos IPs possui versões sintetizáveis e passíveis de simulação com precisão de ciclos.

O *Model Designer* é um ambiente baseado em Eclipse [23] que oferece suporte para a modelagem de IPs em SystemC. Possui suporte para depuração de comunicação, um sistema de visualização de execução dos processos SystemC, dos eventos gerados e recebidos pelo módulo e do desempenho da simulação. É compatível com o *Plataform Architect*, de modo que os módulos desenvolvidos neste IDE são diretamente importados por ele.

Outro *software* interessante da *CoWare* é o *Processor Designer*. Consiste em um ambiente de desenvolvimento de ASIPs (processador com conjunto de instruções para aplicação específica – *Application Specific Instruction Set Processor*) provido de primitivas para o desenvolvimento de processadores RISC, DSPs, SIMD e VLIW. É baseado na ADL LISA, presente já há alguns anos no mercado e comprovadamente bem aceita. Possui ferramentas paralelas capazes de gerar as ferramentas de suporte ao *software*, como montador, *linker* e compilador de C. Uma vez descrito o processador na linguagem LISA, é possível gerar simuladores com precisão de ciclos e instrução ou até mesmo uma descrição sintetizável.

Com a descrição de uma plataforma pronta com IPs, processadores e barramentos já definidos, pode-se usar uma ferramenta chamada *Virtual Plataform* para gerar uma máquina virtual correspondente à plataforma. Os projetistas de *software* podem então utilizar esta máquina virtual para desenvolver e testar as aplicações enquanto a plataforma é refinada para um modelo sintetizável. Esta possibilidade adianta o desenvolvimento do *software*, diminuindo o *time-to-market*.

Um ponto negativo é a utilização de bibliotecas proprietárias. O SystemC, assim como a biblioteca TLM, é compatível com as versões atuais da OSCI [47], mas possui anotações e modificações específicas da empresa. Isto pode gerar atrasos no lançamento de novas versões. De modo geral, a *CoWare* se destaca com o pacote de ferramentas mais completo na área de prototipação funcional de hardware, pecando apenas por não fornecer um sintetizador de código de alto nível e um simulador RTL, apesar de prover meios para a integração de seus programas com ferramentas de outros fabricantes.

Cadence

A *Cadence* [10] é uma das maiores empresas de EDA do mercado, contribuindo ativamente para a padronização e ditando muitos das convenções adotadas. Destacou-se pelas ferramentas OrCad, para desenho eletro-eletrônico em geral, Allegro, para prototipação de barramentos, Virtuoso, para projeto de circuitos analógicos e Incisive para prototipação de *hardware*. Apesar de a tradição da empresa ser verificação, para o propósito desta dissertação descreverei apenas as ferramentas de simulação ESL da família Incisive.

O *Incisive Simulator* é o núcleo da família, consistindo em três partes capazes de simular a maioria das linguagens utilizadas para desenvolvimento de *hardware*. Possui suporte a simulação multi-linguagem, utilização de asserções dinâmicas[3], suporta TLM sobre SystemC e possui ferramentas de instrumentação e depuração do código, inclusive automáticas. Oferece ao projetista a possibilidade de programar plataformas usando orientação a aspectos, diminuindo o tempo gasto com geração de testes.

O suporte a TLM é bastante avançado, contando com suporte a geração de estímulos por *software*, inclusive com simulação híbrida envolvendo módulos já em descrições sintetizáveis, como Verilog ou VHDL. Assim como a maioria das empresas de EDA, possui uma biblioteca com implementações próprias dos barramentos mais comuns como AMBA e AXI em diversos níveis de abstração, incluindo TLM. A biblioteca é bem mais completa quando comparada com os concorrentes, abrangendo IPs de uso comum, como USB e SATA.

O ambiente *Incisive Scenario Builder* é uma ferramenta gráfica totalmente voltada para a criação e gerenciamento de testes. O principal objetivo é ser utilizável por desenvolvedores não especializados em testes. Justamente com este ambiente, o programa *Incisive Verification Manager* ajuda o projetista com a execução dos testes durante o desenvolvimento do projeto, incluindo a mudança de nível de abstração. Suporta uma linguagem proprietária chamada *Specman Elite*, capaz de gerar testes funcionais com o objetivo de reaproveitamento em fases posteriores.

Celoxica

A *Celoxica* [12] é uma empresa pequena se comparada com as apresentadas até o momento, mas possui dois produtos interessantes para o contexto desta dissertação. O primeiro deles é o *Agility Compiler*, capaz de gerar código simulável e sintetizável a partir de descrições SystemC RTL. Trabalha em conjunto com as ferramentas da *Synopsys*, alegando inclusive ser capaz de sintetizar descrições TLM.

O segundo ambiente, chamado *DK Design Suite*, permite a descrição de plataformas completas utilizando uma variação do ANSI-C chamada Handel-C, desenvolvida e mantida

[3]Primitivas capazes de verificar pré-condições e abortar a simulação se elas não forem atingidas.

pela própria empresa. Apesar de este ser um ponto discutível devido à adoção em massa do SystemC pela indústria EDA, o ambiente permite síntese direta de descrições nesta linguagem para FPGA. Os benefícios podem ser grandes se esta abordagem se tornar madura o suficiente, dado que o particionamento entre *hardware* e *software* é muito mais simples de ser realizado devido à inexistência de elementos processadores simulados, sendo o *software* executado na própria máquina. As simulações também tendem a ser mais rápidas, pois pode-se concentrar em uma parte específica do código, trabalhar em um *hardware* que execute a mesma tarefa e manter o restante da plataforma em código nativo.

A abordagem da *Celoxica* usando elementos processadores com código nativo executando na máquina é um tanto quanto alternativa quando comparada com as escolhas feitas pelas outras empresas, mas promissora por se tratar de um novo paradigma capaz de acelerar o início do fluxo de projeto.

Mentor Graphics

A *Mentor Graphics* [44] possui uma família inteira de produtos para projeto de *hardware* baseado em plataformas. Os principais são os da família *Plataform Express*. Um deles (*Platform Express Professional*) oferece um ambiente gráfico de captura de projeto modular com módulos baseados em XML. Apesar de todos os prospectos não citarem nada sobre a compatibilidade com descrições SPIRIT, há fortes indícios da adoção do padrão (a principal é a presença da empresa no consórcio SPIRIT). A adoção do SPIRIT visa a padronizar a indústria ESL, possibilitando a troca de código entre as ferramentas e aumentando a reusabilidade dos módulos. Uma característica interessante desta ferramenta não encontrada nos concorrentes é a possibilidade de descrições parciais. Pode-se definir um IP com os pinos de entrada e saída e integrá-lo à plataforma mesmo sem a funcionalidade implementada. Esta "casca", além de servir como base para o desenvolvimento do IP, pode ser substituída por um *software* que imita sua funcionalidade em qualquer linguagem suportada pelos simuladores da empresa, incluindo C/C++.

Há programas para desenvolvimento de IPs em diversos níveis de abstração, todos com suporte a descrição XML. Entre eles, destaco o *System Vision*, que possui integração com as ferramentas de simulação da empresa (família *ModelSim*) e auxilia a geração de casos de testes.

Outro destaque importante da empresa é o *Catapult*, um ambiente que permite o projeto de plataformas a partir de um código ANSI-C, inclusive com síntese das partes que forem escolhidas para se tornarem *hardware* na fase de particionamento. Há diversas críticas a esta ferramenta, principalmente quanto aos códigos passíveis de sintetização. A empresa, porém, alega que qualquer código ANSI-C é sintetizável sem extensões proprietárias.

Synopsys

A *Synopsys* [60] é uma das empresas de grande porte do mercado de EDA. Possui um pacote chamado *Galaxy Design Platform* para o desenvolvimento de plataformas, mas a empresa atualmente concentra os seus esforços em descrições RTL.

O produto mais próximo do trabalho desenvolvido neste projeto de mestrado é o *System Studio*, especializado em captura de plataformas algoritmicamente. O projetista inicia com uma descrição gráfica do comportamento da plataforma representando o algoritmo ou tarefa que ela executa, passando em seguida para uma descrição em SystemC e fazendo refinamentos sucessivos até o nível de síntese (SystemC RTL). Este produto destina-se mais ao particionamento *hardware/software* e está longe de ser uma ferramenta de captura de projeto.

O *Innovator* foi criado pensando no desenvolvimento de plataformas virtuais. Possui um ambiente gráfico que permite a instanciação e a conexão de IPs. As plataformas geralmente utilizam uma descrição de alto nível dos IPs, gerando um simulador capaz de emular a plataforma real, visando ao desenvolvimento adiantado das partes de *software* de um projeto.

Magillem

A *Magillem Design Services* é uma empresa francesa iniciante no mercado de EDA. O objetivo principal da empresa é fortalecer a utilização de SPIRIT (seção 3.1.4) pela indústria.

O produto principal da empresa é um ambiente de captura de projeto chamado *Magillem IDE*, totalmente compatível com o SPIRIT. Consiste em um módulo para Eclipse [23] que possibilita a montagem de uma plataforma a partir de componentes modulares com descrições SPIRIT. A conexão entre os componentes também pode ser feita através do ambiente, mas a empresa não oferece suporte a simulação diretamente, apenas por ferramentas externas.

Para fomentar o uso da ferramenta, um segundo produto foi desenvolvido, chamado *SPIRIT Packager*. Também é um módulo para Eclipse consistindo em um guia para descrição SPIRIT. Já possui compatibilidade com as bibliotecas dos principais fabricantes e possibilita que uma descrição seja feita manualmente, permitindo a inclusão de praticamente qualquer módulo na sua biblioteca. Com a descrição SPIRIT de um IP é possível utilizá-lo no *Magillem IDE*.

O SPIRIT possui diretivas para informar como o módulo sendo descrito deve ser simulado e como ele se conecta com o mundo exterior. Um IP corretamente descrito pode ser facilmente simulado se a ferramenta estiver disponível. A descrição em diferentes níveis de abstração também é prevista no SPIRIT. As ferramentas se aproveitam destas características permitindo a invocação de simuladores de terceiros a partir do ambiente,

inclusive com simulações híbridas (diferentes níveis de abstração).

Ferramentas gráficas de captura como o *Magillem* tendem a se tornar centralizadoras de projetos ESL, dado que esta é a primeira fase prática de qualquer projeto de *hardware*.

Outros

A ACE [1] (*Associated Compiler Experts*) mantém o projeto *CoSy Express* com o objetivo de gerar compiladores otimizados para um determinado ISS. O *software* é proprietário e não possui licença acadêmica ou de avaliação.

Beach Solutions [5] é uma companhia relativamente pequena que entrou no mercado há pouco tempo. Possui um pacote de ferramentas chamado EASI-Studio, composto de três programas principais. *System Capture ESL Tool* é um ambiente de captura gráfica de projetos de plataformas completas. Analogamente, o *IP Block Interface Capture ESL Tool* permite que um IP composto por módulos possa ser "montado" graficamente. Ambas as ferramentas suportam SPIRIT e vários níveis de abstração SystemC. Não há menção alguma de simuladores internos na documentação, mas há uma lista de compatibilidade que inclui os simuladores das principais empresas. Há diversos programas extras, capazes de checar a validade de uma descrição ou configurar um sintetizador para ser chamado, por exemplo.

A empresa americana *Bluspec* [7] foi fundada em 2003 por pesquisadores do MIT a partir de pesquisas da utilização de TRS (*Term Rewriting Systems*) para modelagem de *hardware*. TRS é baseado em execução predicativa de regras predefinidas. Basicamente existem termos que descrevem estados do *hardware* e regras que descrevem as mudanças de estado e o momento em que elas ocorrem. A empresa extendeu a biblioteca SystemC para suportar descrições usando esta metodologia e lançou um pré-processador alegando ser capaz de gerar código sintetizável a partir das mesmas.

2.2 Trabalhos Acadêmicos

Riccobene *et al.* apresentam em [50] um ambiente para o desenvolvimento de plataformas baseado em modelagem UML. A idéia central é que o projetista parta de uma descrição UML da plataforma e refine os módulos até a síntese RTL. O ambiente possui ferramentas de apoio, como um programa gráfico para capturar a plataforma em módulos UML, derivar código SystemC (com entradas e saídas) a partir de um módulo UML e fazer a engenharia reversa de um módulo SystemC para sua representação UML. O ponto forte do artigo é a utilização de descrições em metalinguagens, que está se tornando freqüente pelas ferramentas de EDA.

O ambiente descrito por Schaumont *et al.* em [55] possui várias características desejáveis em um simulador de plataformas, como precisão e possibilidade de testes integrados. O problema é que o ambiente utiliza uma linguagem específica de modelagem chamada GEZEL, desenvolvida pelos autores especialmente para a modelagem de plataformas. A linguagem utiliza um modelo de computação determinístico baseado em máquinas de estado, o que torna a simulação mais rápida em casos específicos, como descrições de processadores em RTL, e permite simulação interpretada (o SystemC utiliza um modelo baseado em eventos discretos). Os ganhos de velocidade reportados no estudo de caso chegam a duas vezes em relação ao mesmo modelo em SystemC para descrições RTL, mas a linguagem possui uma desvantagem de três vezes quando a simulação é completamente funcional, ganhando apenas no tempo de compilação.

Lapalme *et al.* desenvolveram em [40] um ambiente para simulação de plataformas que utiliza como base a linguagem Easy.NET, desenvolvida pelos autores. A linguagem é derivada de .NET e conseqüentemente de C#, a linguagem utilizada neste ambiente. Analogamente ao SystemC, os autores estenderam a linguagem para que ela suportasse as primitivas intrínsecas da modelagem de *hardware*, como tipos ternários[4]. Os autores deixam claro que não propõem a substituição do SystemC, mas descrevem todo um arcabouço para o desenvolvimento de ferramentas EDA baseado em .NET. Duas vantagens de suma importância foram identificadas também pelos autores: a necessidade de instrospecção de código em tempo de execução e a lentidão do sistema de troca de contexto entre processos SystemC. A instrospecção é nativa no C#, assim como a reflexão, e os autores cogitam o uso de ambas. O problema da troca de contexto pode ser solucionado com as *fibers* [5] de .NET, que são processos leves gerenciados pela aplicação.

Kempf *et al.* apresentaram em [36] uma técnica de modelagem de plataformas voltada para plataformas multiprocessadas. A metodologia é baseada em VPUs (Unidades Virtuais de Processamento [6]). As VPUs possuem a funcionalidade representada por código executado na máquina hospedeira da simulação, tornando-as mais rápidas que as abordagens onde realmente simula-se a máquina alvo. As anotações de temporização são descritas em uma metalinguagem derivada de XML. Este modelo possibilita uma mobilidade muito alta nas fases iniciais de projeto, quando o projetista ainda está decidindo as partes da aplicação que serão transformadas em *hardware*.

Na contramão das pesquisas na área, Genz e Drechsler apresentam em [29] uma ferramenta chamada *ViSyC* que tem como objetivo principal facilitar a exploração de plataformas descritas em SystemC. O programa lê as descrições e as transforma em representações gráficas capazes de serem filtradas em três níveis de abstração diferentes. Uma

[4]Valores verdadeiro, falso e alta impedância.

[5]`http://msdn2.microsoft.com/en-us/library/ms682661.aspx`

[6]Do inglês: *Virtual Processing Unit.*

plataforma totalmente RTL pode ser visualizada como um todo ou ter seus módulos analisados individualmente, por exemplo. Os autores alegam que tal visualização possibilita ao desenvolvedor uma visão geral não proporcionada pela descrição SystemC e, apesar de esta afirmação ser perfeitamente plausível, há poucas possibilidades de tal ferramenta obter sucesso se não permitir alterações que se reflitam na descrição SystemC.

Susan Xu e Hugh Pollitt-Smith [63] mostram a possibilidade de simular um sistema híbrido que mistura descrições VHDL e SystemC, utilizando a plataforma Incisive da Cadence. O estudo de caso mostrado não é reaproveitável, sendo necessário reescrever toda a parte em SystemC especificamente para cada projeto. O exemplo que foi exposto inclui um AMBA escrito em SystemC interagindo com componentes em SystemC e VHDL através de uma interface TLM.

Diversos ambientes para experimentação de um nicho específico de *hardware* podem ser encontrados na bibliografia. Plataformas de simulação de processadores específicos são comuns, como o PEK [34] (*PowerPC Evaluation Kit*) e o Cell (não disponível publicamente), ambos fornecidos pela IBM. Na área acadêmica são mais notáveis os trabalhos envolvendo plataformas de simulação de barramento, sendo as mais comuns plataformas de experimentação de NoCs [16] [62].

Um ambiente específico que merece destaque é o GRAAL, especializado em plataformas de aceleração gráfica, publicado por Crisu *et al.* em [18]. A ferramenta é comparável em recursos com as ferramentas comerciais, apesar de se concentrar em um tipo de *hardware* muito específico. Possui um controle gráfico da simulação SystemC, visualizador de dados e estimador de desempenho. Acompanha ainda uma biblioteca com as construções mais comuns da área, com suporte a *pipeline* gráfico e descrições RTL sintetizáveis.

Na área de depuração de IPs não existe referência bibliográfica sobre a utilização de reflexão computacional. Serão apresenteados os dois trabalhos que serviram como ponto de partida para a idéia original.

Déharbe e Medeiros apresentam em [22] uma abordagem usando programação orientada a aspectos. O foco principal do artigo não é a depuração e sim a instrumentação para a retirada de métricas como taxa de utilização e tempo gasto na simulação, mas o modelo pode ser facilmente adaptável para depuração. A dupla utilizou em seus testes a biblioteca AspectC++ [3] e não foi observada queda de desempenho significativa. Uma grande desvantagem deste método é que ele não permite alterações nos valores internos dos módulos (ex. atributos) enquanto o código do módulo está executando, dado que a instrumentação é feita nas bordas de chamada das funções. A adoção de técnicas deste tipo também deve ser acompanhada de uma mudança de paradigma por parte dos desenvolvedores, o que nem sempre é viável.

O PINAPA, apresentado em [45] por Moy e outros, é um pré-processador de C++ capaz de entender as estruturas de SystemC. As estruturas geradas por este pré-processador

podem ser utilizadas externamente por outros módulos SystemC simplesmente para analisar a estrutura e o comportamento da arquitetura ou para validá-la. Não há menção sobre a utilização destas estruturas para depuração, mas o autor considera viável tal uso.

O PDesigner é um ambiente de captura gráfico baseado no Eclipse. Este foi um trabalho desenvolvido em conjunto com o trabalho deste mestrado e por possuir uma grande integração com o mesmo será apresentado em seção separada (seção 4.4).

O presente trabalho insere-se na categoria de ferramentas de captura de projetos de plataformas mas não constitui ferramenta gráfica, sendo a modelagem realizada completamente em SystemC. Em contrapartida, o repositório de IPs é integrado, não necessitando de nenhuma biblioteca adicional para a geração do simulador, que pode ser derivado diretamente da descrição da plataforma. A depuração de IPs é um trabalho a parte realizado sobre a infra-estrutura desenvolvida, sem adições ou modificações. Há excelentes ferramentas de depuração, inclusive comerciais, mas todas tendo como base bibliotecas SystemC, compiladores ou simuladores modificados especialmente para tal tarefa.

Capítulo 3

ARP: Uma infra-estrutura para construção de plataformas

3.1 Introdução

Nesta seção apresentaremos os conceitos básicos necessários para a construção de modelos de simulação de plataformas heterogêneas, para na seção 3.2 discutirmos como todas as ferramentas foram integradas na infraestrutura que chamamos de ARP (*ArchC Reference Platform*).

3.1.1 SystemC

> "SystemC é uma nova linguagem de modelagem baseada em C++ que possibilita o desenvolvimento em nível de sistema e a troca de IPs."

A citação acima de Stuart Swan foi retirada de um de seus artigos [59]. No meio ESL existem duas visões do SystemC, sendo a primeira ilustrada na citação e a segunda compartilhada pelo autor: SystemC é uma extensão da linguagem C++ na forma de bibliotecas que provêm tipos, métodos e objetos necessários para a prototipação de *hardware* e um núcleo de simulação.

A linguagem C++ não possui nenhum tipo capaz de representar todos os níveis lógicos. O tipo *bool*, por exemplo, é capaz de armazenar a informação binária mas não as informações relativas ao estado elétrico da ligação (*tri-state* ou indefinido, por exemplo). Analogamente, a execução de uma simulação de *hardware* obrigatoriamente deve manipular a característica de paralelismo inerente aos processos de *hardware*, inexistente na execução normal de um programa seqüencial. Mesmo com o uso de bibliotecas de execução paralela como a *pthread* [28], o esforço necessário para simular uma execução

real de *hardware* torna tal tarefa inviável, pois seria necessário um arcabouço personalizado para cada projeto. O *kernel* do SystemC preenche este espaço fornecendo um ambiente de simulação baseado em eventos totalmente controlável e com uma simulação paralela próxima à execução real.

Uma característica importante do SystemC é a possibilidade de utilização de ferramentas usuais de programação C++ para o desenvolvimento de projetos como o compilador GCC. Tais ferramentas estão publicamente disponíveis e possuem alto grau de confiabilidade e estabilidade. A única exceção é o processo de depuração. A utilização de ferramentas usuais como o GDB (GNU *debugger*) mostrou-se inviável devido às inúmeras heranças, inclusive múltiplas, entre as classes de SystemC. O código da biblioteca não é simples e quase sempre o usuário acaba com os programas de depuração apontando para trechos delas.

A simulação também difere de outros paradigmas por gerar um executável da descrição. Normalmente esta abordagem produz simulações mais rápidas que as interpretadas e a conexão com programas externos pode ser feita utilizando os recursos convencionais do sistema operacional, como *pipes* e FIFOs.

A biblioteca está disponível para *download* no *site* da OSCI [47], bem como o manual do usuário e o guia de referência da linguagem. O código é aberto e disponível publicamente mas possui licença específica não compatível com GPL (*General Public License*). Atualmente é mantido por um consórcio de 13 das maiores empresas de EDA do mundo, como Cadence, Mentor, Forte, ARM, CoWare, Intel etc.

3.1.2 TLM

Os projetos de *hardware* modernos atingiram um nível de complexidade tal que a metodologia utilizada até então para o desenvolvimento e verificação de *hardware* tornou-se obsoleta. A maioria dos SoCs possuem múltiplos processadores heterogêneos entre si, geralmente mais de um barramento e várias caches ou dispositivos de armazenamento, sem contar os dispositivos de entrada e saída, periféricos de controle e aceleradores projetados para uma tarefa específica.

A modelagem RTL, que foi um dos pilares do desenvolvimento e teste de *hardware* na última década, tornou-se lenta, tanto para o tempo de modelagem quanto para o tempo de simulação, devido ao aumento de complexidade dos projetos [58]. A necessidade de um modelo funcional o mais cedo possível no fluxo de projeto também contribui para colocar as metodologias tradicionais em risco. A exploração da arquitetura e a integração entre *hardware* e *software* tem influenciado muito o fluxo de projeto e conseqüentemente o *time-to-market*: a primeira por permitir mudanças no projeto antes mesmo da descrição RTL começar a ser feita, e a segunda por permitir uma integração entre os projetistas de

software e *hardware* cedo o suficiente para que adequações e *feedbacks* sejam feitos ainda nas fases de simulação funcional.

Enquanto os paradigmas RTL e *gate level* utilizam sinais para modelar a comunicação entre os blocos, o TLM utiliza chamadas de funções. As chamadas de funções são feitas em *software* e podem passar vários parâmetros (como um vetor de bits) de uma só vez, o que acelera a simulação. Apesar de ser possível a modelagem com precisão de ciclos[1] utilizando TLM, a maioria dos projetistas posterga esta simulação para a fase RTL com o intuito de ganhar velocidade. Em [64], Kim e outros mostram o desenvolvimento de um barramento AMBA usando TLM com precisão de ciclos, obtendo 97% de precisão em relação ao modelo RTL e com uma velocidade de simulação média 353 vezes maior. Usando TLM puramente funcional para descrever um modelo similar, Schirner [56] publicou ganhos de velocidade da ordem de 10^4, sacrificando para isso a precisão de ciclos.

A única desvantagem real da modelagem TLM identificada até o presente é a baixa reusabilidade dos modelos, pois necessitam de canais específicos para cada comunicação [11]. Cada canal possui um tipo amarrado a ele e ambos os extremos, o emissor e o receptor, devem ser construídos de forma a entender o protocolo utilizado. A experiência no laboratório onde foi realizado o mestrado[2] mostrou que este problema pode ser facilmente contornado quando se utiliza um modelo de comunicação fornecido pelo SystemC chamado transporte (*transport*). Neste modelo, toda a parte de chamada de funções passa a ser administrada pelo próprio SystemC, bastando que se defina o tipo do pacote a ser transmitido. Os modelos fazem o acesso com primitivas do tipo *read* e *write* em uma porta conectada a um canal que implementa a interface de transporte. Apesar de ainda não ser totalmente genérica, a reutilização de modelos é alta o suficiente para justificar a adoção deste modelo. Lembro que nesta metodologia a transmissão é considerada instantânea. Qualquer tentativa de contagem de tempo ou geração de estatística deve levar este fato em conta.

3.1.3 ArchC

O surgimento da tecnologia VLSI possibilitou a inclusão de milhares de transistores em uma única pastilha, tornando viável a inclusão de vários elementos processadores em um único projeto. A área e a complexidade deixaram de ser o fator determinante na escolha entre um processador de uso geral e um *hardware* desenvolvido sob medida, prevalecendo opções por potência dissipada ou flexibilidade de uso. Para reduzir o tempo de projeto, a grande maioria das empresas de desenvolvimento passou nos últimos anos a utilizar módulos *off-the-shelf*, que são baratos em comparação a um projeto a partir do zero e

[1] Do inglês: *cycle accurate*.

[2] LSC – Laboratório de Sistemas Computacionais, Instituto de Computação, Unicamp.

dispensam testes.

Como os problemas mais comuns em sistemas embutidos (com exceção de áreas críticas, como equipamentos médicos) não são restritivos quanto a velocidade ou podem ser paralelizados para atingir esta meta, a maioria dos projetistas prefere transferir o problema do domínio do *hardware* para o do *software*, onde a alteração, adaptação e correção são muito mais rápidas, além de permitir certa mobilidade de projeto mesmo depois de fabricado o componente. Essa transferência de domínio se dá pela substituição dos componentes que antes seriam projetados especificamente (*full custom hardware*) por elementos processadores genéricos executando um *software* capaz de realizar a mesma tarefa. O relatório do ITRS [25] de 2003 já apontava uma migração dos ASICS para SoCs, e o relatório de 2005 vai mais além: aponta um crescimento dos MPSoCs (*Multiprocessor System-on-Chip*) em detrimento dos SoCs.

Com esta crescente utilização de processadores de uso geral, faz-se necessário que as ferramentas de projeto de alto nível se adaptem ao novo estilo de modelagem. Este cenário levou ao surgimento de diversas ADLs como LISA [48], nML [24] e EXPRESSION [31]. Com esta idéia em mente, o LSC [20] desenvolveu e mantém o ArchC [51, 4], uma ferramenta de modelagem de processadores baseada em SystemC. O ArchC possui uma linguagem própria, complementar e baseada em SystemC. Esta ADL foi desenvolvida com o objetivo de fornecer informações suficientes para gerar, além do simulador do processador, outras ferramentas como montadores e ligadores.

Uma descrição ArchC é dividida em duas partes: a descrição da arquitetura (`AC_ARCH`) e a descrição do conjunto de instruções (`AC_ISA`). Na primeira são descritas as características inerentes da arquitetura, como a hierarquia de memória, estrutura do *pipeline* (aplicável em modelos com precisão de ciclos), dispositivos de armazenamento (e.g. *cache* interna) e outras informações relevantes à arquitetura. Na segunda parte da descrição, cada instrução é descrita formalmente com dados sobre:

- formato, tamanho e sintaxe (para o montador);
- código para decodificação (*opcode*);
- comportamento da instrução.

O ArchC possui diversas palavras chave para a definição da arquitetura, como `ac_mem` (memória interna), `ac_regbank` (banco de registradores), `ac_wordsize` (tamanho da palavra de memória) entre outros. Na listagem 3.1 podemos ver a definição de uma arquitetura MIPS, composta por um banco de 32 registradores (linha 3) de 32 bits e uma memória interna de 5MB (linha 2). Apesar de a arquitetura MIPS ser uma arquitetura *Harvard*, a arquitetura descrita não será utilizada para explorar esta característica, omitindo assim

a separação entre as memórias de dados e instruções e utilizando uma só memória para armazenar todas as informações.

Listagem 3.1: Descrição de uma arquitetura MIPS

```
AC_ARCH(mips1){
  ac_mem DM:5M;
  ac_regbank RB:32;
      ac_wordsize 32;
      ARCH_CTOR(mips1) {
        ac_isa("mips1_isa.ac");
        set_endian("big");
      };
};
```

Esta descrição, apesar de completa funcionalmente, exclui detalhes de temporização por se tratar de uma descrição puramente funcional. Há palavras reservadas para a descrição com precisão de ciclos, como `ac_pipe` (estágios de *pipeline*). Em [4] pode ser encontrada uma descrição mais detalhada do ArchC e em [19] está disponível um guia de referência da linguagem.

Listagem 3.2: Trecho de descrição ISA de uma arquitetura MIPS.

```
AC_ISA(mips1){
  ac_format Type_R  = "%op:6 %rs:5 %rt:5 %rd:5 %shamt:5 %func:6";
  ac_format Type_I  = "%op:6 %rs:5 %rt:5 %imm:16:s";
  ac_instr<Type_I> lb, lbu, lh, lhu, lw, lwl, lwr;
  ac_instr<Type_R> add, addu, sub, subu, slt, sltu;
  ISA_CTOR(mips1){
    lwl.set_asm("lwl %reg, \%lo(%exp)(%reg)", rt, imm, rs);
    lwl.set_asm("lwl %reg, (%reg)", rt, rs, imm=0);
    lwl.set_asm("lwl %reg, %imm(%reg)", rt, imm, rs);
    lwl.set_decoder(op=0x22);
    addu.set_asm("addu %reg, %reg, %reg", rd, rs, rt);
    addu.set_asm("move %reg, %reg", rd, rs, rt="$zero");
    addu.set_decoder(op=0x00, func=0x21);
  };
};
```

Na listagem 3.2 pode-se visualizar um pequeno trecho da descrição ISA da arquitetura MIPS. Nas linhas 2 e 3 foram declarados dois formatos de instrução com os bits do *opcode* divididos da forma descrita (ex. Formato R: MSB 31–26 = *opcode*, 25–21 = RS (*source register*) etc). Logo abaixo, nas linhas 4 e 5, podemos ver a declaração das instruções em si, cada uma atribuída a um formato de instrução. A instrução `addu`, por exemplo, faz parte do formato R (no MIPS este é o formato para todas as instruções aritméticas).

Na seção `ISA_CTOR` (a partir da linha 6), as informações sobre as instruções são detalhadas para permitir a decodificação e geração de ferramentas. A instrução `lwl` foi

declarada com três formatos diferentes de mnemônicos, todos previstos pela arquitetura. Essa sobrecarga permitirá que o montador interprete corretamente a entrada utilizando casamento de padrões. A instrução `addu` foi declarada com dois mnemônicos diferentes, como a instrução anterior, mas com um mnemônico de instrução completamente diferente da instrução em si (`move`, linha 12). Isto foi feito pois a instrução `move` não existe diretamente no MIPS, mas deve ser interpretada pelo montador e transformada em uma adição simples com um dos operandos sendo o registrador zero (que contém o cardinal zero). Outra sutileza foi a declaração do campo `func`. Todas as operações do MIPS que envolvem a ULA possuem o mesmo *opcode*, sendo diferenciadas pelo valor contido neste campo, previsto no formato da instrução (formato R). O ArchC é capaz de diferenciar instruções com esta característica e mapear corretamente o comportamento.

A descrição do comportamento funcional das instruções é feita diretamente em SystemC (ou C++ se aplicável) em um arquivo separado. Na listagem 3.3 podemos ver a descrição das duas instruções citadas anteriormente. Como não foi utilizado nenhum recurso de SystemC na arquitetura, como tipos tipos especializados da sua biblioteca, a descrição pode ser inteiramente feita em C++. O tipo `ac_regbank` que utilizamos para declarar o banco de registradores possui entre suas funcionalidades uma sobrecarga que permite o acesso direto como vetor C, facilitando ainda mais a tarefa. O tipo `ac_Uword` é um tipo interno do ArchC que será sempre mapeado para um tipo com metade do tamanho de palavra da arquitetura descrita, independentemente da combinação plataforma/arquitetura em que se está executando a simulação (Linux/i386, Solaris/SPARC etc.). A memória (DM) que declaramos possui métodos de acesso de leitura e escrita universais independentes da implementação de memória utilizada (adiante ilustrarei a utilização do TLM no ArchC).

Listagem 3.3: Trecho de descrição funcional ISA de uma arquitetura MIPS.

```
//!Instruction lwl behavior method.
void ac_behavior( lwl )
{
  unsigned int addr, offset;
  ac_Uword data;
  addr = RB[rs] + imm;
  offset = (addr & 0x3) * 8;
  data = DM.read(addr & 0xFFFFFFFC);
  data <<= offset;
  data |= RB[rt] & ((1<<offset)-1);
  RB[rt] = data;
};

//!Instruction addu behavior method.
void ac_behavior( addu )
```

```
{
  RB[rd] = RB[rs] + RB[rt];
};
```

O comportamento esperado para a instrução `lwl` (*Load Word Left*) é uma carga alinhada indireta da metade menos significativa a partir do endereço contido em `rs` e do imediato especificado na instrução para o registrador `rt`. As outras funções de controle do MIPS (como incremento do contador de programa) são executadas em um comportamento com descrição especial que é executado para todas as instruções.

É importante lembrar que todo o comportamento da instrução será executado instantaneamente pelo núcleo do SystemC pois a descrição ilustrada é um modelo funcional. Modelos com precisão de ciclos possuem uma descrição peculiar que foge do propósito desta dissertação. Novamente, em [4, 19] podem-se encontrar referências a este tipo de descrição, bem como um detalhamento das demais funções presentes no ArchC.

Comunicação TLM no ArchC 2.0

A versão 2.0 do ArchC, já disponível em domínio público para testes, possui uma característica fundamental para os trabalhos apresentados nesta dissertação: a comunicação TLM. Até então, os simuladores gerados ficavam restritos à memória interna declarada em `AC_ARCH` (vide listagem 3.1). A comunicação TLM possibilita que os acessos à memória sejam redirecionados para módulos externos através de uma porta específica para tal, permitindo que dispositivos mapeados em memória sejam conectados à mesma.

Listagem 3.4: Descrição de uma arquitetura MIPS com porta TLM.

```
AC_ARCH(mips1){
  ac_tlm_port DM:5M;
  ac_regbank RB:32;
      ac_wordsize 32;
      ARCH_CTOR(mips1) {
        ac_isa("mips1_isa.ac");
        set_endian("big");
      };
};
```

Na listagem 3.4 podemos ver que no lugar da memória (listagem 3.1, linha 2) há uma porta TLM (linha 2). Esta porta receberá todas as requisições do processador, agindo como uma memória para ele. As requisições não são mais tratadas pelo simulador e são deixadas a cargo do usuário, que deve implementar os dispositivos externos de suporte necessários (incluindo a memória, requisito obrigatório mínimo). É importante salientar que nenhuma mudança se faz necessária na descrição ISA ou no comportamento das instruções dado que a `ac_tlm_port` possui os mesmos métodos dos dispositivos de

armazenamento. A metodologia TLM adotada é a *transport*, citada na seção 3.1.2. Na seção 3.2 há detalhes de como conectar e utilizar a porta.

Além do gerador de simulador e do montador, o pacote do ArchC oferece outras ferramentas. A simulação compilada permite a geração de um simulador específico para determinado programa, o que torna a simulação mais eficiente em detrimento da capacidade de executar qualquer código. Este tipo de simulação está caindo em desuso devido à alta velocidade obtida pelo simulador interpretado, mas ainda está disponível no ArchC. A integração com o GDB também é muito apreciada, pois permite a utilização de sua interface para a depuração da aplicação executada no modelo em desenvolvimento. Pode-se por exemplo executar um programa passo a passo e analisar o conteúdo dos registradores. Outra característica interessante é a emulação de sistema operacional. A emulação permite o uso de primitivas de entrada e saída, como `printf` ou `open`, mesmo sem um *hardware* ou *driver* para tal. As requisições (*syscalls*) são capturadas pelo ArchC e executadas na máquina hospedeira, tornando possível a abertura de arquivos, impressão na tela e captura de teclado, por exemplo.

3.1.4 SPIRIT

Com o objetivo de acelerar o projeto de grandes plataformas SoCs, a indústria de semicondutores criou o SPIRIT [15], um padrão para descrever e manipular IPs que possibilita configuração automática por ferramentas EDA.

A utilização de descrições formais de IPs aumenta a automação por parte das ferramentas, uma vez que a especificação deve cobrir o mínimo possível de informações sobre o IP. A portabilidade aumenta também, dado que qualquer desenvolvedor de ferramenta pode implementar sua própria versão de *software* para ler, interpretar e manusear IPs compatíveis com o padrão. Este fator ajuda a manter a coerência entre as bibliotecas de IPs internas e externas de um desenvolvedor de *hardware*, além de possibilitar a troca de IPs entre fabricantes.

O SPIRIT foi pensado para cobrir todo o fluxo de projeto, da descrição funcional até a pastilha de silício. Atualmente a especificação IP-XACT (disponível em [15]) está na versão 1.2, que não padroniza descrições funcionais. As extensões da versão 1.4 já incluem tais descrições mas ainda não foram aprovadas pelo consórcio.

Listagem 3.5: Fragmento da descrição SPIRIT do processador MIPS do ArchC.

```
<spirit:component
   xmlns:spirit="../processor/mips1"
   xmlns:xsi="../processor/mips1"
   xsi:schemaLocation="../processor/mips1/mips1.xsd">
  <spirit:vendor>ArchC</spirit:vendor>
  <spirit:library>mips1</spirit:library>
```

```
  <spirit:name>mips1</spirit:name>
  <spirit:type>processor</spirit:type>
  <spirit:version>1.0</spirit:version>
  <spirit:busInterfaces>
    <spirit:busInterface>
      <spirit:name>DM_port</spirit:name>
      <spirit:busType spirit:vendor="ArchC"
                      spirit:library="mips1"
                      spirit:name="DM_port"
                      spirit:version="1.0"/>
      <spirit:master>
        <spirit:addressSpaceRef
         spirit:addressSpaceRef="port_DM"/>
      </spirit:master>
      <spirit:maxMasters
        spirit:multiplicity="single">1
      </spirit:maxMasters>
      <spirit:connection>required
      </spirit:connection>
    </spirit:busInterface>
  </spirit:busInterfaces>
</spirit:component>
```

Um exemplo de descrição contendo um fragmento da descrição SPIRIT do processador MIPS do ArchC pode ser visto na listagem 3.5. A porta TLM não é explicitamente declarada, passando apenas como uma conexão de barramento genérica.

3.2 ARP: *ArchC Reference Platform*

A ARP (ArchC *Reference Platform*) é um espaço de trabalho dedicado à modelagem de plataformas utilizando modelos do ArchC. É composta por uma estrutura de diretórios fixa com locais específicos para os componentes mais comuns em modelos heterogêneos de plataformas e um conjunto de *scripts* para manipular e simular plataformas.

O objetivo principal é oferecer um ponto de partida para os usuários que desejam desenvolver plataformas utilizando os simuladores ArchC, mas a infra-estrutura pode ser útil para introduzir novos usuários ao ArchC e ao SystemC.

A estrutura de diretório da ARP é fixa para que o usuário possa separar os diversos módulos da plataforma e trabalhar separadamente em cada um.

- `bin`: ARP scripts;
- `doc`: Documentação;
- `platforms`: Plataformas exemplos e do usuário;

- `processors`: Simuladores gerados pelo ArchC e utilizados como *cores* nas plataformas;
- `is`: Barramentos e estruturas de interconexão (*interconnection structures*);
- `ip`: IPs, também utilizados como *cores*;
- `sw`: *Software* que é executado nos simuladores de processadores;
- `wrappers`: Os tradutores entre os *cores*, IPs e níveis de abstração.

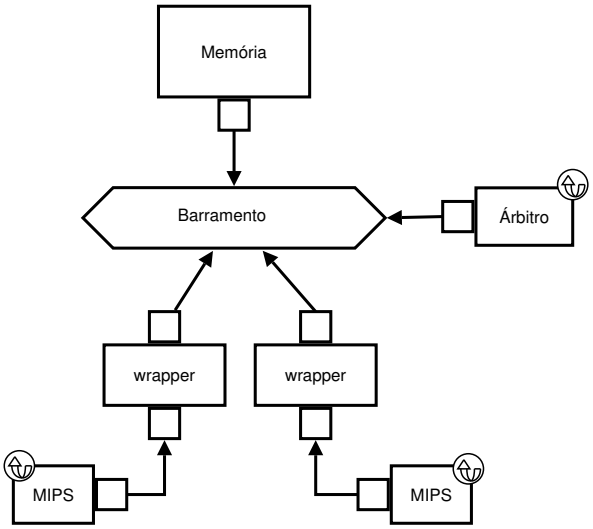

Figura 3.1: Arquitetura da plataforma *dual_mips*.

Cada diretório dentro do diretório `platforms` representa uma plataforma e deve conter um arquivo de definições chamado `defs.arp` contendo a estrutura da plataforma. O exemplo abaixo é o arquivo de definições da plataforma exemplo `dual_mips` que será utilizada como exemplo e cuja estrutura pode ser vista na figura 3.1. Este exemplo está disponível publicamente no *site* [19] do projeto ArchC e uma descrição mais detalhada da sua estrutura e funcionamento pode ser encontrada na seção 5.1.

```
IP :=
IS := simple_bus_ua
PROCESSOR := mips1
SW := fft_shared
WRAPPER := ac.simple_bus_ua
```

A estrutura deste arquivo é muito similar à estrutura de um arquivo `Makefile`, mas as opções são listas. Se a plataforma utilizar mais de um *wrapper*, a variável correspondente será uma lista de *wrappers* separados por espaço. Listas em branco também são permitidas. Exemplo de lista:

```
WRAPPER := ac.simple_bus_ua  graph_terminal.simple_bus_ua
```

Uma questão que surge sempre que as metodologias TLM são utilizadas é como proceder para interconectar dispositivos que possuem assinaturas diferentes. No caso do *transport* essa equivalência se dá na estrutura dos pacotes, que podem diferir entre dois componentes que precisam se comunicar. A solução proposta e utilizada na ARP é a inserção de um tradutor, citado na bibliografia como *wrapper*. O *wrapper* é um módulo de SystemC que conhece o protocolo das duas partes envolvidas e sabe como traduzir os pacotes de um em outro. Seu trabalho consiste em atender as requisições de ambos os lados, transformar os pacotes para serem compatíveis com a estrutura do protocolo de destino (literalmente uma tradução) e agir como se fosse o próprio transmissor.

Na maioria das vezes o *wrapper* não introduz nenhum efeito colateral ao sistema, incluindo a temporização. Em alguns casos, como a modelagem BFM (Modelo Funcional de Barramento [3]) [11], o *wrapper* pode introduzir um tempo de espera para simular as latências envolvidas na simulação, mas na grande maioria dos casos estudados o propósito é que ele não seja notado no fluxo de dados.

Listagem 3.6: Exemplo de wrapper com *transport*.

```
ac_tlm_rsp w_sb::transport( const ac_tlm_req &request ) {
  ac_tlm_rsp response;
  unsigned int toff;
  toff=m_address;
  // Common memory area
  if ((request.addr>=0x200000)&&(request.addr<=0x300000))
    toff=0x0;
  switch( request.type ) {
    case READ :     // Packet is a READ one
      response.status = readm( request.addr+toff, response.data );
          break;
    case WRITE:     // Packet is a WRITE
      response.status = writem( request.addr+toff , request.data );
          break;
    default :
      response.status = ERROR;
      break;
  }
```

[3] Do inglês: *Bus Functional Model*.

```
  return response;
}

ac_tlm_rsp_status w_sb::writem( const uint32_t &a , const uint32_t &d )
{
  simple_bus_status status;
  uint32_t td;
  td = d;

  status = bus_port->burst_write(m_unique_priority, (int*)&td, a, 1, m_lock);

  wait(m_timeout,SC_NS);
  if (status == SIMPLE_BUS_ERROR)
    return ERROR;
  else
    return SUCCESS;
}
```

Na listagem 3.6 há um trecho do *wrapper* utilizado pelo exemplo `dual_mips`. Este *wrapper* é inserido entre o processador e o barramento e é responsável por fazer a tradução entre os dois protocolos. O tráfego de dados é bidirecional, mas o processador é sempre considerado mestre. O protocolo sempre é o mesmo: o processador envia um pacote de requisição e recebe um pacote de resposta em troca. O SystemC invoca a função `transport` (linha 1) sempre que um pacote de requisição é colocado no canal entre o processador e o *wrapper*. O `switch` (linha 8) desta função é responsável por decidir qual o tipo da requisição (o protocolo `ac_tlm` possui vários tipos, sendo os tipos `READ` e `WRITE` os utilizados aqui) e chamar a função correspondente. Na mesma listagem há na linha 22 um exemplo da função *write* onde podemos ver a chamada TLM para o barramento (linha 28).

O barramento utilizado nesta plataforma foi o `simple_bus`, disponível com o pacote TLM e escrito originalmente por Ric Hilderink, da Synopsys. Consiste em um barramento abstrato totalmente modelado usando TLM. O modelo de barramento é baseado em ciclos, o que implica que o mesmo simula todos os eventos em cada ciclo de relógio como um barramento real, sem se importar com os sub-eventos internos que acontecem dentro de cada ciclo.

Neste exemplo, o *wrapper* foi utilizado para mais duas tarefas. A primeira é o mapeamento do espaço de endereçamento. A plataforma possui uma memória de 10MB, compartilhada pelos dois processadores. Qualquer requisição feita pelo primeiro processador será mapeada para os primeiros 5MB de memória, enquanto as requisições feitas pelo segundo processador são mapeadas para os últimos 5MB. A exceção é a área comum entre 2 e 3MB, que o *wrapper* mapeia igualmente para o mesmo endereço físico na memória (ver linha 6 da listagem 3.6, onde um *offset* é ou não gerado de acordo com

a área). Esta tarefa é considerada uma especialização do *wrapper* por se tratar de uma característica da plataforma. A figura 3.2 representa o espaço de endereçamento deste exemplo, com o mapeamento.

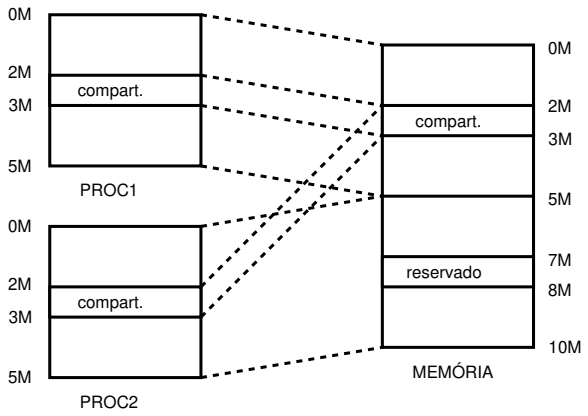

Figura 3.2: Espaço de endereçamento da plataforma *dual_mips*.

A segunda tarefa é a emulação de temporização. Na função `write` da listagem 3.6 há uma chamada à função `wait` do SystemC na linha 30, o que faz com que o processo seja suspenso por um determinado tempo, simulando o atraso referente à transação de escrita em memória e tornando a simulação mais próxima do *hardware* real. Esta característica é genérica o suficiente para ser utilizada em outros projetos mas, neste em especial, foi utilizada com o propósito de permitir uma disputa mais justa pelo barramento entre os processadores.

Os *scripts* que acompanham a ARP foram inseridos para automatizar algumas tarefas rotineiras. Atualmente, as opções disponíveis são `pack` e `unpack`, `list` e `delete`. O `pack` recebe como parâmetro uma plataforma e gera um pacote intercambiável contendo todos os arquivos necessários para que a plataforma seja executada em outro ambiente, facilitando a troca de modelos. É possível empacotar somente uma parte (e.g. um IP) também. O `unpack` é utilizado por quem recebe o pacote para extrair o conteúdo do arquivo. O conteúdo será colocado exatamente nos locais originais, imitando a estrutura de quem montou o pacote. A opção `list` é útil para visualizar o que será empacotado e o que está sendo incluído na compilação e simulação da plataforma. A opção `delete` permite que se apague uma plataforma do ambiente sem prejudicar as demais plataformas presentes (e.g. um IP compartilhado por duas plataformas não será apagado se uma delas for apagada através deste *script*). O exemplo a seguir mostra as dependências da plataforma `dual_mips`, obtida através do *script* de listagem.

```
$ ./arp --list dual_mips.01
```

```
PLATAFORM dual_mips.01 is:
 -- is/simple_bus_ua
 -- processors/mips1-archc2x-branch
 -- sw/fft_shared
 -- wrappers/ac.simple_bus_ua
---
```

Recentemente, um esforço foi realizado para aumentar a integração entre a ARP e o *PDesigner* [14]. O *PDesigner* é um projeto do CIn (Centro de Informática) da UFPE (Universidade Federal de Pernambuco) que tem como objetivo principal permitir a captura de um projeto de plataforma graficamente. Por exemplo: o usuário pode montar uma plataforma arrastando componentes gráficos representando processadores, IPs e barramentos. A descrição da plataforma gráfica é então transformada em uma descrição SPIRIT (seção 3.1.4). No presente momento a ARP possui *scripts* capazes de importar uma estrutura de plataforma capturada com o *PDesigner* e transformá-la em uma plataforma ARP executável e simulável. Outra funcionalidade interessante é o uso dos geradores previstos no padrão SPIRIT, implementados tanto na ARP quanto no PDesigner, que permitem que um IP possa ser desenvolvido de maneira genérica e especializado de acordo com a utilização. Estas funcionalidades não estão disponíveis publicamente e somente serão liberadas para uso público quando a especificação atingir um nível estável de maturidade.

A estrutura apresentada até o momento, incluindo a ARP que foi desenvolvida no decorrer do mestrado, serviu como base para os trabalhos realizados. No próximo capítulo (4.1) apresentarei uma metodologia para depuração de IPs utilizando reflexão computacional, metodologia esta aplicada a uma plataforma descrita sobre a ARP. No capítulo 5.1 há mais exemplos de plataformas descritas utilizando o *framework*.

Capítulo 4

Reflexão Computacional

4.1 Conceito

Reflexão computacional [43], doravante denominada apenas por reflexão, é a habilidade de inspecionar e modificar tipos de dados para um determinado sistema. Em um objeto, por exemplo, a reflexão deve retornar ao usuário todos os seus métodos e atributos, com os respectivos tipos e o mínimo de informação necessária para acessá-los fora do escopo do objeto.

Há dois tipos de reflexão: em tempo de compilação e em tempo de execução (*compile-time* e *run-time* respectivamente). A reflexão em tempo de compilação visa facilitar a programação genérica obtendo propriedades de determinado tipo ou a relação entre tipos no momento em que o programa é compilado. Entre outros objetivos, esse tipo de informação pode ser utilizada pelo compilador para otimizações de memória, por exemplo.

A reflexão em tempo de execução utiliza as informações sobre as estruturas de dados no momento em que o programa está sendo executado sem que o código que está utilizando essas informações as conheça em tempo de compilação. Repare que as informações podem ser coletadas em tempo de compilação; a classificação só diz respeito ao código que fará uso das informações. Ambos os tipos podem ser utilizados como mesmo propósito, respeitando a restrição imposta pela reflexão em tempo de compilação, que requer o conhecimento prévios dos tipos a serem utilizados. Todas as bibliotecas analisadas durante este trabalho utilizam o procedimento de captura em tempo de compilação. A captura em tempo de execução é inviável pois mesmo que o compilador fosse alterado para manter no código binário as informações sobre os objetos, a análise dessas informações é custosa computacionalmente. Uma característica indesejável do método de captura em fase de compilação é a utilização de memória, dado que o usuário deve indicar quais estruturas irá utilizar ou carregar todas as informações coletadas para que elas estejam disponíveis em tempo de execução. Entre outras finalidades, este tipo de reflexão é utilizado para

troca de estruturas entre linguagens, persistência de objetos e por interpretadores.

Linguagens recentes, como Java, C# e Python, possuem algum tipo de reflexão embutido na especificação da linguagem, mas a linguagem C++ possui recursos muito limitados para a reflexão. Na realidade, a única informação já disponível por padrão faz parte da especificação RTTI (*Runtime Type Information*). A RTTI somente armazena informações necessárias para as operações de *cast* dinâmico, que se resumem ao tipo e ao nome da estrutura. Obviamente isto é insuficiente para qualquer propósito real de reflexão, sendo útil apenas ao sistema operacional (no momento de alocação, *cast* e *recast* dinâmicos).

Os módulos de *hardware* descritos usando SystemC são classes e objetos de C++. Neste trabalho a reflexão será utilizada como meio de inspeção e alteração do estado dos módulos SystemC, possibilitando ao usuário interagir com a simulação por meio de intervenções no estado dos módulos, representado pelas informações contidas em seus atributos.

4.2 Bibliotecas

A biblioteca idealizada por Knizhnik [38] estende a estrutura RTTI do C++ para armazenar as informações necessárias para a reflexão. Como por padrão o GCC só disponibiliza as informações RTTI para classes com pelo menos um método virtual, esta biblioteca já não seria o ideal pois este tipo de método é raro em modelagem de *hardware* com SystemC. Outro fator é que as informações extras devem ser incluídas pelo projetista no código da classe na forma de entradas de dicionário. Como mostrarei na seção 4.3, um dos requisitos deste trabalho é ser transparente ao usuário.

Similarmente, a biblioteca AReflection [2] apóia-se em informações inseridas no código da classe para obter as informações refletidas, inviabilizando seu uso por necessitar de intensiva instrumentação por parte do usuário. Outras abordagens como a utilização de metaclasses também foram descartadas pelo mesmo motivo. O modelo Mirrors [9] é uma proposta excelente e promissora por implementar reflexão dinâmica automática e sem intervenção do usuário, mas até o momento não havia uma implementação para C++ disponível publicamente.

O pacote OpenC++ [13] é um conjunto completo de ferramentas para reflexão. Parece ser interessante, mas não foi testado por necessitar de um compilador especialmente desenvolvido para permitir o uso de metaobjetos.

Há uma biblioteca de reflexão disponível no repositório virtual *SourceForge* chamada CPPReflect [41], mas o projeto parece estagnado e não há documentação disponível, tornando impossível os testes por parte do autor. Além disso as descrições encontradas apontam para uma abordagem também baseada em metaclasses, portanto é provável que necessite de anotações no código da classe.

O conjunto de bibliotecas *Boost* [8] fornece um pacote chamado *type_traits* capaz de fornecer informações dinâmicas sobre tipagem, mas utiliza uma abordagem em tempo de compilação. O código que fará a introspecção dos dados deve conhecer os tipos em tempo de compilação, o que não inviabiliza sua utilização para o nosso propósito, mas aumenta o esforço dado que o código deve ser modificado sempre que os objetos a serem analisados sofrerem alguma modificação.

O SystemC provê uma biblioteca oficial para verificação chamada SCV (*SystemC Verification Library*) [35]. O propósito principal não é oferecer reflexão mas sim meios para verificação, porém há uma seção dedicada à introspecção de dados. Esta biblioteca parece ser a mais promissora de todas e possivelmente será estudada em futuras versões do mecanismo de depuração de IPs desenvolvido neste mestrado, mas na atual versão utiliza um mecanismo similar aos *type_traits* da biblioteca *Boost*, sofrendo as mesmas restrições e conseqüentemente inviabilizando sua adoção para os propósitos deste trabalho.

A utilização de programação orientada a aspectos (*AspectC* [3]) foi analisada por necessitar intrinsecamente de reflexão, mas foi descartada pois gera uma instrumentação do código objeto similar aos *softwares* de *profiling*, tornando o código muito lento. Além disso, uma mudança no paradigma de programação poderia se tornar um inconveniente, dado que a maioria dos projetistas de *hardware* estão acostumados com construções orientadas a objetos e não a aspectos.

A biblioteca Reflex-SEAL [52] foi desenvolvida pelo CERN como parte do projeto ROOT. Entre todas as bibliotecas testadas, esta foi a que teve o melhor resultado em termos de facilidade de uso e quantidade de informações disponibilizadas. Algumas características que levaram a essa escolha:

- implementa reflexão total para C++;
- coleta de informações não intrusiva;
- coleta semi-automática;
- não possui dependências externas;
- é compilável com o GCC padrão.

A biblioteca somente foi testada para a iteração sobre a lista de atributos de uma classe C++ e para o acesso de leitura e escrita aos atributos. Permite ainda definir o escopo de uma variável interna e a chamada de métodos da classe (mesmo por outro objeto), mas estas funcionalidades não fazem parte deste trabalho e portanto não foram testadas na totalidade, apesar de funcionarem perfeitamente para os exemplos executados.

A coleta de informações é feita em duas fases, ambas em tempo de compilação. Na primeira fase, utiliza-se o programa `GCC_XML` [37]. Este programa gera um arquivo XML

contendo a estrutura das classes de um determinado arquivo cabeçalho (os arquivos com extensão `H`). Esta representação é o equivalente da descrição interna do analisador sintático de C++ do pacote GCC em um formato XML.

Na segunda fase, as informações sobre a estrutura da classe passam por um gerador de dicionário fornecido com a biblioteca de reflexão. Este programa fornece um arquivo chamado dicionário compilável e passível de ligação (arquivo objeto) com o restante do sistema. As informações contidas neste arquivo dicionário são basicamente a estrutura de cada classe analisada acompanhada de informações de tipagem, tamanho e *offset* a partir do endereço base do objeto. Esta coleta não necessita de interferência direta do usuário, é feita somente quando a classe sendo refletida é alterada (modificações no código que irá utilizar as informações não afetam o dicionário) e não depende de instrumentação no código (não intrusiva).

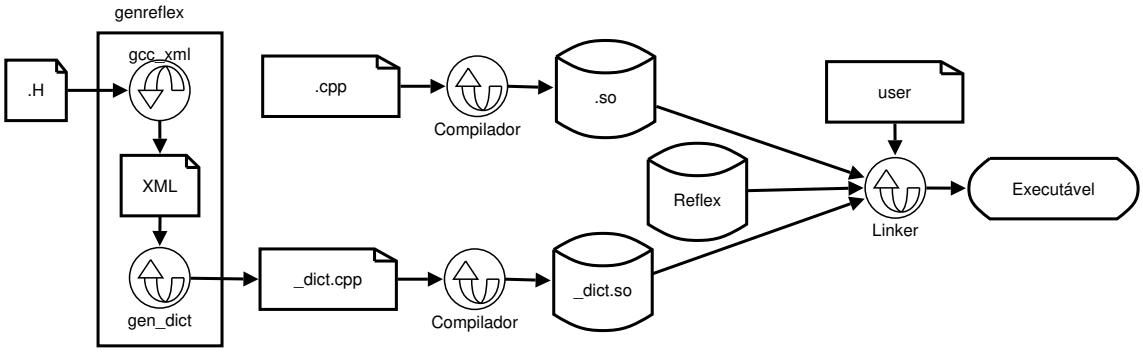

Figura 4.1: Diagrama da coleta de dados e compilação usando reflexão.

Na figura 4.1 podemos visualizar os passos necessários para coletar os dados, gerar o dicionário e compilar o sistema utilizando reflexão. Os arquivos representados por `.H` e `.cpp` contêm a descrição da classe (ex. `teste.H` e `teste.cpp`). Para efeito de documentação, executar o gerador de dicionário (*genreflex*) é suficiente se o arquivo XML não existir, pois o mesmo encarrega-se de invocar o `gcc_xml`. A saída do gerador é um arquivo compilável com a estrutura do dicionário (no exemplo, `teste_dict.cpp`). Após compilação de ambos, os objetos gerados (`teste.so` e `test_dict.so`) são ligados com a biblioteca (Reflex) e com o restante do código do sistema, já compilado (representado no diagrama pelo arquivo "user"), gerando o executável. O código representado por "user" terá então acesso à biblioteca e aos recursos fornecidos.

A biblioteca em si é fornecida na forma de uma biblioteca estática sem dependências externas mas o código fonte está disponível publicamente para aqueles que utilizam outra plataforma. Não há dependências externas, bastando fazer a ligação da biblioteca, do dicionário e dos objetos envolvidos no momento da compilação. A biblioteca possui primitivas de acesso ao dicionário, de modo que o usuário não precise utilizar aritmética

de ponteiros para acessar os atributos ou métodos de uma classe. O compilador utilizado é o GCC 4.1.1 padrão sem nenhum tipo de adição ou modificação em seu código.

4.3 Implementação

Uma das carências do SystemC é a verificação e depuração do código. Atualmente somente a biblioteca SCV é fornecida e ela não cobre todos os aspectos de depuração. A utilização de ferramentas de depuração como o GDB também foi descartada pois ele não é capaz de distinguir entre o código do objeto sendo depurado e o código da própria biblioteca SystemC. Como todos os módulos em SystemC possuem ao menos uma herança múltipla (`sc_module`), hora ou outra o depurador acaba caindo em código da biblioteca. Além de não ser interesse do projetista, o código possui otimizações de velocidade que o torna de difícil compreensão até mesmo para usuários experientes.

Para tentar suprir esta necessidade, foi desenvolvida neste trabalho uma implementação de um *WhiteBox*. O *WhiteBox* é um dispositivo descrito na especificação SPIRIT (seção 3.1.4) e capaz de monitorar as atividades de um módulo SystemC observando os dispositivos de armazenamento de dados internos representados pelos atributos da classe que contém o módulo. Quando uma mudança acontece em algum deles, o *WhiteBox* pode tomar uma decisão como parar a simulação, alterar o valor ou simplesmente gravar o valor e o evento da mudança em um arquivo para posterior análise. Até o momento, não foi possível identificar nenhuma tentativa de implementação deste tipo de dispositivo na literatura. Os dispositivos mais próximos são apenas observadores passivos, incapazes de manipular valores, usados principalmente como geradores de *log*.

A nossa implementação de *WhiteBox* foi definida com alguns requisitos, todos respeitados na atual implementação:

- Não intrusivo: o projetista do IP a ser analisado não precisa saber de antemão que o IP será observado, dispensando portanto instrumentação interna;
- Reflexão automática: o processo de geração de dicionário deve ter o mínimo de intervenção possível. Na verdade, atualmente não há intervenção alguma;
- O *WhiteBox* deve se comunicar externamente usando *sockets*;
- Deve ser possível filtrar a condição de parada através do *WhiteBox*.

A estrutura que compõe o *WhiteBox* consiste de um módulo SystemC contendo um processo de inspeção. O módulo foi construído utilizando *templates* de C++, o que permite uma especialização de acordo com o IP que está sendo depurado. O módulo pode ser sensível ao relógio para o caso de ser utilizado com IPs que possuam tal característica.

A cada fatia de tempo de simulação no *kernel* do SystemC, o módulo é invocado e congela a simulação para fazer a análise. Como o módulo ganha o controle da simulação SystemC, basta ele não o devolver através dos mecanismos de `wait` ou `next_trigger` para manter o *kernel* de simulação do SystemC congelado. Para sistemas sem relógio, a porta deve ser ligada a uma instância de relógio sem valor para que o módulo seja escalonado em todos os ciclos de simulação.

Listagem 4.1: Trecho do *WhiteBox* utilizando a biblioteca de reflexão.

```
// Assinatura do construtor
template <class T>
whitebox<T>::whitebox(sc_module_name name, string objname, T *originalObj,
                      unsigned int port);
// Funcao principal com exemplos de uso da reflexao
template <class T>
void whitebox<T>::do_inspect(){
        // Obtendo uma referencia ao objeto sendo inspecionado
        Type t1 = Type::ByName(objname);
        // Cria uma referencia para o objeto a partir do dicionario
        Object objtest(t1,objpointer);
        // Adiciona todos os atributos refletidos a uma lista
        for (Member_Iterator mi = t1.DataMember_Begin();
         mi != t1.DataMember_End(); ++mi) {
                tmpattr.name = (*mi).Name(SCOPED);
                tmpattr.type = (*mi).TypeOf().Name(QUALIFIED);
                tmpattr.oldvalue = convert(tmpattr.type, (void*) (
                           objtest.Get(((*mi).Name(FINAL))).Address()));
                tmpattr.address = (objtest.Get(((*mi).Name(FINAL))).Address());
                attrs.push_back(tmpattr);
        }
}
```

Na listagem 4.1 podemos visualizar um trecho de código do *WhiteBox* que utiliza a biblioteca de reflexão. Este trecho pertence a função `do_inspect` (linha 7) e corresponde ao início da execução do módulo. Este código é responsável por varrer os atributos refletidos e armazenar suas informações em uma lista interna. O objeto é passado para o *WhiteBox* de duas maneiras: a primeira é uma *string* contendo o seu nome e a segunda é um ponteiro apontando para a instância a ser analisada. Na linha 9 o nome do objeto é utilizado para uma busca no dicionário. Com o resultado dessa busca, declaramos na linha 11 um objeto da biblioteca de reflexão que contém uma representação da estrutura do objeto proveniente do dicionário e a referência para a instância.

Com a estrutura interna da biblioteca corretamente inicializada (temos um objeto com a estrutura interna e com o ponteiro corretos), podemos realizar várias operações de reflexão, como determinar o escopo de um atributo e chamar métodos, entre outras.

A mais importante para os nossos propósitos é a varredura dos atributos. O laço `for` da linha 14 utiliza a infra-estrutura de iteradores da biblioteca para realizar esta tarefa. Cada atributo é percorrido para ser colocado em uma lista contendo informações como seu nome, seu valor atual, seu tipo e seu endereço físico nesta instância. Esta lista é utilizada em outras partes do *WhiteBox* para ler e alterar os valores de cada atributo de acordo com as instruções passadas pelo usuário.

Listagem 4.2: Exemplo de uso do *WhiteBox*.

```
stubip stubinstance("stubip",3,0x300500,0x3FFFFF);
whitebox<stubip> wbstub("wbstub","stubip",&stubinstance,6000);
stub.clock(bus_clock);
wbstub.clock(bus_clock);
```

Na listagem 4.2 podemos ver um exemplo de uso do *WhiteBox*. O IP a ser refletido é o `stubip` cujos parâmetros são irrelevantes para o entendimento. Para o *WhiteBox*, passamos (segundo parâmetro) o nome da classe no IP (para a procura no dicionário) e um ponteiro para a a instância que queremos refletir (terceiro parâmetro). O quarto parâmetro é opcional e determina qual porta o *WhiteBox* irá escutar por conexões remotas. Caso seja aplicável, a amarração (*bind*) com o relógio também deve ser feita.

O protocolo utilizado para a comunicação via *socket* pode ser visto abaixo na forma BNF (*Backus–Naur Form*). É um protocolo bastante simples, baseado em *strings* para permitir um certo grau de entendimento por um ser humano mesmo sem o uso de alguma interface.

Listagem 4.3: Descrição BNF do protocolo de comunicação.

```
<ListaDeAtributos> ::= <IDAtributo> | <IDAtributo>; <ListaDeAtributos>
<IDAtributo> ::= "NomeClasse"::"NomeAtributo"|<NULL>
<PacotePD2WB> ::= I<IDAtributo>;"ValorAtributo";<TipoBreakPoint>;
                  "ValorBreakPoint";<Boleano>;<EOL>
<PacoteWB2PD> ::= I"TempoDeSimulacao";<IDAtributo>;"ValorAtributo";
                  <Boleano>;<Boleano>;<EOL>
<Boleano> ::= TRUE | FALSE
<TipoBreakPoint> ::= Always|OnChange|ConditionGreater|ConditionLess|
                     ConditionEqual|None
<PacotePDGet> ::= G<EOL>
<PacoteWBGet> ::= G<ListaDeAtributos>;<EOL>
<PacotePDHandshake> ::= Handshake_Finished<EOL>
<PacoteWBHandshake> ::= <Response>
<Response> ::= Ok<EOL>|Fault<EOL>
```

Na representação desta listagem, `PD` representa o usuário ou o programa de interface e `WB` representa o *WhiteBox*. O termo PD foi adotado pois o protocolo foi desenvolvido com o programa PDesigner em mente. O PDesigner faz o papel de interface gráfica neste

caso e será apresentado na seção 4.4. Uma descrição mais detalhada dos dois pacotes principais do protocolo pode ser vista abaixo:

- **PacotePD2WB:** O booleano indica se o `PD` espera resposta do tipo `<Response>`. A estrutura do pacote do tipo `<Response>` pode ser vista na listagem anterior.
- **PacoteWB2PD:** O primeiro booleano com valor verdadeiro indica que o `WB` espera por resposta do tipo `PD2WB` com `ValorAtributo` não nulo.
- **PacoteWB2PD:** O segundo booleano indica que `WB` espera por resposta do tipo `PD2WB` (caso acima) ou `<Response>` (se o primeiro caso acima for nulo).

O *WhiteBox* inicia passivo enquanto que o programa de interface inicia ativo (responsável pela primeira comunicação). Na listagem 4.4 podemos ver um exemplo de comunicação.

Listagem 4.4: Exemplo de comunicação.

```
1  PD: <PacotePDGet> : G
2  WB: <PacoteWBGet> : Gjpeg::at1;jpeg::at2;jpeg::at3;
3
4  PD:<PacotePD2WB> ::= Ijpeg::at1;<NULL>;Always;<NULL>;F;
5  PD:<PacotePD2WB> ::= Ijpeg::at2;<NULL>;<ConditionEqual>;"5";F;
6  PD:<PacotePD2WB> ::= Ijpeg::at3;<NULL>;<NULL>;<NULL>;F;
7
8  <PacotePDHandshake> ::= Handshake_Finished
9  <PacoteWBHandshake> ::= Ok
10
11 WB: <PacoteWB2PD> : I0.2;jpeg::at1;"h";F;F;
12 WB: <PacoteWB2PD> : I0.3;jpeg::at1;"he";F;F;
13 WB: <PacoteWB2PD> : I0.4;jpeg::at1;"hel";F;F;
14 WB: <PacoteWB2PD> : I0.5;jpeg::at1;"hell";F;F;
15 WB: <PacoteWB2PD> : I0.6;jpeg::at1;"hello";F;F;
16 WB: <PacoteWB2PD> : I0.6;jpeg::at2;"5";F;F;
17 PD: <PacotePD2WB> : Ijpeg::at2;"6";<NULL>;<NULL>;F;
```

A primeira fase é de *handshake*, onde o programa interface obtém a lista de atributos refletidos. Nas duas primeiras linhas podemos ver a requisição e a resposta. Como podemos ver na segunda linha, o *WhiteBox* provê para este IP exemplo três atributos. Em seguida, a partir da linha 4 a interface envia os pacotes com instruções sobre o que o *WhiteBox* deve fazer com os atributos. O `F` é uma abreviação do valor booleano `False` previsto na especificação BNF. A primeira mensagem diz que o primeiro atributo deve ser inspecionado em todos os ciclos de simulação. A segunda mensagem gera uma condição de parada para o segundo atributo, que será analisado somente se o seu valor for igual a

5. O terceiro atributo está sendo ignorado pois o usuário decidiu que não tem interesse nele.

A linha 8 determina o final da primeira fase e o começo da segunda fase. Até o momento, não houve simulação e todos os processos SystemC se encontram na fase de elaboração. O início da segunda fase libera o *kernel* para a simulação normal e inverte os papéis na comunicação. O *WhiteBox* passa agora a ser ativo e o programa de interface passa a ser passivo até ser solicitado. A partir da linha 11 podemos ver que a simulação está rodando e o *WhiteBox* está enviando os dados. A primeira linha indica que no tempo de simulação 0.2 o valor do atributo `at1` mudou para `h`. A simulação prossegue com a impressão do primeiro atributo até a linha 16, onde um *breakpoint* foi encontrado. Como o valor do atributo atingiu o valor determinado, a simulação é parada e espera-se uma ação por parte do usuário. Neste caso, o usuário decidiu alterar o valor do atributo para "6". O IP exemplo foi modelado com a funcionalidade de copiar uma *string* interna para o primeiro atributo caractere por caractere até que o valor de caracteres copiado seja igual a 6, o que finaliza a simulação.

4.4 Integração

O *PDesigner* (ou *Platform Designer*) é um projeto em desenvolvimento pelo GRECO (Grupo de Engenharia de Computação) do CIn (Centro de Informática) da Universidade Federal de Pernambuco (UFPE). O propósito é fornecer um ambiente de modelagem de plataformas de *hardware* com um suporte gráfico para a fase de captura do projeto. O usuário pode utilizar o *mouse* para clicar e arrastar componentes como processadores ArchC, barramentos ou memórias e interligá-los visualmente em uma área de edição.

Por trás da simulação, há o SystemC como linguagem de modelagem, o ArchC como gerador de simuladores de processadores e alguns conceitos definidos na ARP, como o *wrapper* TLM e a comunicação via transporte. A estrutura de *Makefiles* foi aproveitada de uma das primeiras versões da ARP, mas a representação da modelagem utiliza somente SPIRIT. A área de edição e todas as ferramentas de suporte foram escritas como *plugins* para o Eclipse [23], o que torna a ferramenta portável e de fácil manutenção, além de aproveitar diversos itens de interface que já estão disponíveis.

O projeto em conjunto com o LSC deste instituto, consiste em adaptar o *PDesigner* para que o mesmo suporte a reflexão em IPs e sirva como interface para o usuário no momento da simulação. A adaptação foi feita através de um terminal capaz de se conectar ao *socket* aberto pelo *WhiteBox* e interagir com o usuário.

Na figura 4.2 podemos ver uma tela do PDesigner. A organização é feita com paletas, sendo que as principais podem ser vistas na figura. A aba (A) mostra o navegador de projeto, contendo todos os arquivos a ele pertencentes. É possível editar os arquivos

fontes diretamente no Eclipse utilizando o editor interno. Logo abaixo desta aba vemos uma aba contendo a visão geral da perspectiva que está em edição; no nosso caso estamos na perspectiva de plataforma. À direita podemos ver a área de edição. Por estarmos na perspectiva de plataforma, esta área é composta pela paleta de componentes (B) e pela área de desenho (C). Para construir a plataforma, basta arrastar componentes da paleta para a área de desenho e interligá-los utilizando a ferramenta de conexão, também disponível na paleta. Na parte direita inferior (D) há uma área de saída. Esta área é usada para alterar as propriedades do objeto selecionado e para visualização da saída da simulação.

O exemplo utilizado é uma plataforma de decodificação JPEG, descrita na seção 5.1, mas com o processador MIPS. O acesso ao IP é feito diretamente pelo processador (C2) através do barramento (C4), dado que os registradores programáveis do IP são mapeados em memória nesta arquitetura. Ainda na mesma figura, podemos ver a presença de duas memórias (C3), pois o espaço de endereçamento é bipartido. O espaço entre o final da primeira memória e o início da segunda é preenchido pelos registradores do IP (C1). Após programado corretamente com os endereços das matrizes, o IP é disparado por meio de um bit sinalizador do registrador de controle.

O acesso aos dados refletidos é feito simplesmente clicando-se com o botão direito sobre o IP e escolhendo a opção de ativar a reflexão. Um diálogo como o da figura 4.3 é aberto para se escolher as opções.

As opções escolhidas serão passadas para o *WhiteBox* no momento da elaboração, quando o núcleo do SystemC escalona pela primeira vez todos os processos. Neste exemplo, vamos observar o registrador de controle. Como esse registrador é alterado para 1 pelo processador no início do processamento e alterado para 0 pelo próprio IP quando este termina, poderemos tomar medidas de tempo sobre o IP. O resultado pode ser visto no item D da figura 4.2. Utilizando os tempos mostrados, podemos calcular subtraindo os *time-stamps* que as duas últimas execuções do IP demoraram $451000ps$ e $402000ps$ respectivamente, sendo que o processador gastou $819000ps$ executando código e programando o IP entre as duas execuções. Isto indica que o IP está passando muito tempo ocioso, sendo necessária a adoção de medidas alternativas como o uso de técnicas de *double-buffering*, *pipeline* e processamento paralelo para amenizar o problema.

No próximo capítulo mostraremos os resultados relativos ao desempenho. Também descrevemos as plataformas usadas nos testes, todas descritas usando a ARP, o PDesigner ou ambos, aproveitando a infra-estrutura desenvolvida.

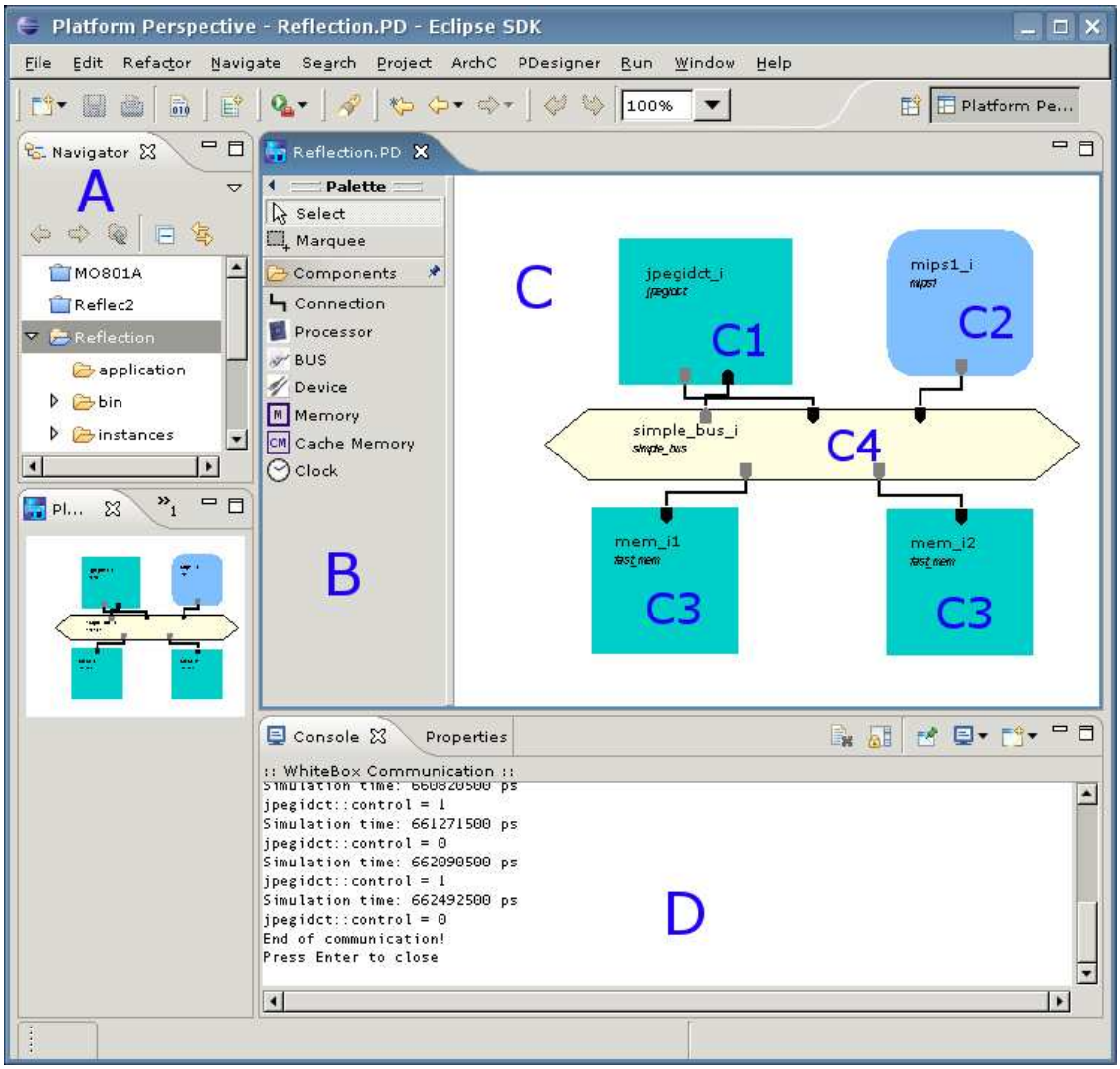

Figura 4.2: Tela do *Platform Designer* com o exemplo JPEG.

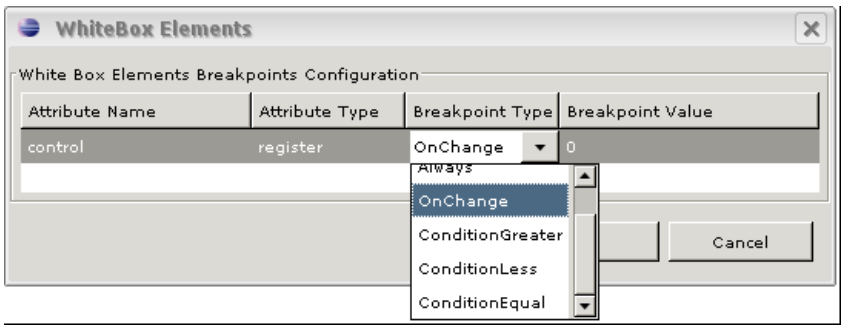

Figura 4.3: Escolhendo as opções de reflexão.

Capítulo 5

Resultados

5.1 Plataformas

No decorrer do mestrado, foram desenvolvidas diversas plataformas com o objetivo de exemplificar os conhecimentos adquiridos. A construção de plataformas exemplo também é uma maneira de arquivar estes conhecimentos e uma tentativa de passá-los às pessoas que estão iniciando na área.

A maior parte do conhecimento imbuído nas plataformas exemplo dizem respeito a conexão de simuladores de processadores ArchC com o mundo externo utilizando TLM e soluções para os problemas decorrentes de tal conexão, como *wrappers* e métodos de sincronização de processadores em sistemas MPSoC. O particionamento entre *hardware* e *software*, parte fundamental na definição de um modelo de plataforma, também está ilustrado nos exemplos.

Plataforma *Hello World*

A primeira plataforma apresentada foi feita com os usuários iniciantes em mente. O ArchC permite que os simuladores sejam gerados com uma memória interna, sem porta TLM, tornando-se uma plataforma limitada, mas completa o suficiente para que uma simulação seja feita (também chamada de simulação *standalone*). A plataforma *helloworld* foi projetada para imitar o comportamento da simulação sem porta TLM, sendo composta apenas pelo processador e por uma memória externa. A memória atende os requisitos do protocolo *ac_tlm_protocol*, de modo que pode ser ligada diretamente ao processador, sem necessidade de barramento ou canal. Na figura 5.1 podemos visualizar a estrutura da plataforma [1].

[1] Todas as ilustrações deste capítulo seguem o padrão TLM definido pela OSCI, também disponível no apêndice A.

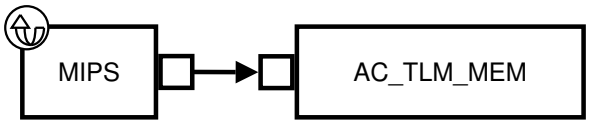

Figura 5.1: Estrutura da plataforma *Hello World*.

A `ac_tlm_mem` simula uma memória utilizando um vetor de `uint8_t` do tamanho escolhido na instanciação. Não há alinhamento nem dependências para acesso além do pacote. O módulo é compatível com o `ac_tlm_protocol` e portanto implementa a função `transport` necessária. Esta função será chamada toda vez que o processador enviar um pacote `ac_tlm_request` e deve sempre responder com um pacote `ac_tlm_rsp`. Como os pacotes só permitem o acesso de 32 bits, o acesso à memória é restrito a esse tamanho (tanto para escrita como para leitura). Por se tratar de uma descrição totalmente funcional, não há temporização na memória, sendo o acesso considerado instantâneo.

O *software* executado no processador MIPS consiste de apenas um laço com uma chamada a uma função de impressão (`printf`). A análise do código binário gerado é simples – devido a simplicidade do código a função *main* possui 24 instruções – e foi utilizada diversas vezes para explicar como funciona a emulação de chamadas de sistema operacional (*syscalls*) no ArchC, dado que a plataforma não possui nenhum periférico de saída mas permite o uso de funções como `printf`.

Plataforma *SPARC Game*

A plataforma *SPARC Game* foi criada a partir de outra plataforma construída como trabalho final de uma disciplina de graduação sobre um sistema de simulação pronto, com processador, barramento e memória fixos. Originalmente este sistema é de difícil expansão ou adaptação devido ao barramento, uma NoC proprietária com protocolo OCP/IP. A estrutura da plataforma pode ser vista na figura 5.2 e foi mantida inalterada com exceção do barramento, que na versão ARP tornou-se o *Simple Bus* (disponível *online* em [47]).

Todo o código foi reescrito na nova versão. O *software* foi adaptado para utilizar o IP gráfico especialmente desenvolvido para essa plataforma, mas a lógica interna permanece a mesma. O IP Gráfico é mapeado em memória, com seus registradores diretamente acessíveis pelo processador por meio de um acesso normal a memória (*load* e *store*). A biblioteca utilizada para emular a saída gráfica é a FLTK [57]. Qualquer sistema operacional suportado pela biblioteca (e obviamente SystemC e ArchC) poderá compilar e simular a plataforma.

O mapeamento em memória pode ser observado na arquitetura da plataforma pela presença de duas memórias. Como o acesso é feito com instruções normais de carga, o espaço reservado para o mapeamento deve ser único. Por convenção, não se utiliza o início

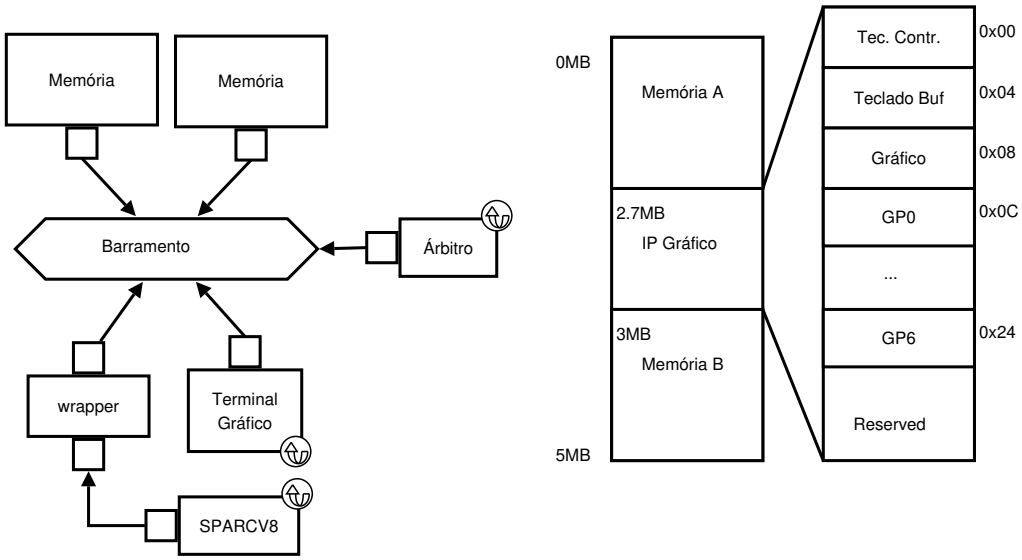

Figura 5.2: Estrutura e espaço de endereçamento da plataforma *SPARC Game*.

e o final do espaço de endereçamento para mapeamento de *hardware* por esses lugares serem geralmente reservados para o código e para a pilha, respectivamente. O espaço deixado em aberto pelas duas memórias pode ser ocupado então por qualquer *hardware* que entenda o protocolo do barramento. No nosso caso, o IP é sempre um escravo com um funcionamento idêntico ao das memórias, porém com cada posição representando um registrador interno.

O acesso ao IP se dá através de primitivas como `insert_line`, `remove_line`, `insert_rect` etc. Cada função necessita de parâmetros específicos que são previamente colocados nos registradores `GP0--GP6`. Com os parâmetros corretamente inicializados, basta que o registrador gráfico seja escrito com o valor correspondente à função desejada para que o processo de desenho e atualização da tela seja disparado. O mapeamento de funções disponíveis e os respectivos parâmetros podem ser vistos na documentação que acompanha o IP na ARP.

O IP Gráfico também captura os eventos relacionados ao mouse e ao teclado quando o foco for a janela aberta. Nesta plataforma, somente os eventos relacionados ao teclado são capturados, mas a funcionalidade para o mouse está implementada. O registrador de controle do teclado funciona como um indicador de evento recebido. Qualquer valor diferente de zero indica que há novos dados no registrador *buffer* do teclado. Escrever um valor nulo no registrador de controle é responsabilidade do usuário e sinaliza para o IP que o *buffer* já foi lido e pode ser descartado. Uma tentativa de escrita no *buffer* resulta em erro de barramento dado que ele é especificado como somente de leitura.

O *software* consiste de dois jogos. O primeiro é o clássico jogo da cobra, onde o

jogador controla uma cobra que cresce uma unidade quando come maçãs. As maçãs são espalhadas aleatoriamente pelo tabuleiro e aparecem uma por vez (desaparecendo quando capturadas pela cobra ou por estouro de um tempo limite). Chocar-se contra a parede ou contra si mesmo configura o fim de jogo. Há um adicional de dificuldade que aumenta a velocidade da cobra proporcionalmente ao número de maçãs ingeridas.

O segundo jogo foi criado como parte do trabalho final da disciplina em que esta plataforma foi desenvolvida e é chamado faroeste. Consiste em um atirador e um alvo, ambos limitados a movimentação vertical. O alvo aparece aleatoriamente na tela e nela permanece por um período de tempo. O atirador só atira horizontalmente, portanto o jogador deve movimentá-lo para acertar o alvo. Se o jogador não conseguir movimentar o atirador e atirar enquanto o alvo permanece na tela, ele perde uma vida. Após dez vidas perdidas o jogo é encerrado. Há um sistema de pontuação baseado no tempo que o jogador demorou para acertar o alvo.

Além do atrativo gráfico, esta plataforma introduz dois conceitos importantes no desenvolvimento de plataformas: *wrapper* e *wrapper* transparente. O primeiro é utilizado entre o processador e o barramento, dado que não utilizam o mesmo protocolo de comunicação. O *wrapper* faz a tradução entre o `ac_tlm_protocol` do processador agindo como um escravo para o mesmo. A tradução é enviada para o `simple_bus` utilizando a interface `simple_bus_master_if`.

O segundo conceito é introduzido pelo IP gráfico. O IP gráfico não é capaz de gerar ou receber comunicação diretamente pois não possui nenhum tipo de interface, consistindo apenas de um módulo SystemC com primitivas de desenho e captura de teclado implementadas na forma de métodos. O *wrapper* transparente deste exemplo conhece o protocolo `simple_bus_slave_if` e sabe manipular a classe gráfica. Quando instanciado, o mesmo instancia também o módulo SystemC gráfico. Quando recebe uma requisição de escrita pelo barramento, ele escreve no atributo correspondente da classe. É chamado de transparente pois o usuário não sabe que existe um *wrapper* neste ponto, pois não há uma comunicação explícita entre o *wrapper* e a classe atendida. Para o usuário, instanciar o *wrapper* transparente é o mesmo que instanciar o módulo SystemC se ele fosse construído para implementar o protocolo do barramento. A única vantagem de utilizar esta metodologia é a modularidade. A classe fica independente de protocolo de comunicação, bastando reescrever o *wrapper* para utilizá-la em outro barramento, sem a necessidade de inserir mais um *wrapper* tradutor entre o novo barramento e a classe.

Plataforma *dual_mips*

A plataforma `dual_mips` foi construída para exemplificar a utilização de mais de um processador compartilhando o mesmo barramento. Os problemas decorrentes da presença

de dois mestres no mesmo barramento são muitos, mas neste exemplo apenas a solução da concorrência por recursos compartilhados foi exemplificada.

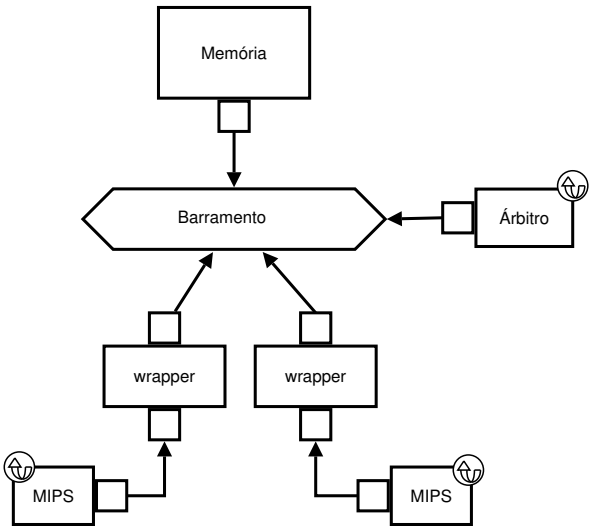

Figura 5.3: Estrutura da plataforma *dual_mips*.

Como podemos ver na figura 5.3, a memória é única para ambos os processadores. Ambos os processadores concorrem pelo barramento. Tal concorrência é resolvida pelo árbitro ligado ao barramento utilizando prioridade fixa para requisições conflitantes. Para as requisições não conflitantes (e.g. feitas em ciclos de relógio distintos), o árbitro utiliza uma política de quem chegou primeiro, ou FCFS (*First Come, First Served*). Os processadores ArchC não oferecem primitivas específicas de barramento como a escolha da prioridade. Para emular este comportamento, o *wrapper* dos processadores foi construído com esta característica. A prioridade é passada na instanciação do *wrapper* e ele se encarrega de propagá-la quando for registrado no barramento.

Outra função relevante do *wrapper* neste exemplo é o mapeamento de espaço de endereçamento. Os processadores dividem a memória de 10MB entre si mas não possuem espaços de endereçamento conflitantes. Ambos os processadores acessam a memória como se esta possuísse apenas 5MB. Para o primeiro processador, este acesso é mapeado no endereço real na memória, ou seja, quando ele acessar o endereço 0x0 estará acessando o mesmo endereço na memória. Para o segundo processador há um mapeamento feito pelo *wrapper*. Os acessos são propagados ao barramento com um *offset* de 5MB, ou seja, uma requisição ao *wrapper* para o endereço 0x0 será propagada para o barramento como sendo uma requisição ao endereço 0x500000. Desta maneira, isola-se o espaço de endereçamento de cada processador, virtualizando uma única memória de 10MB em duas de 5MB. Sem este mapeamento, os processadores buscariam o mesmo código na memória para executar. É possível a solução sem identificação explícita de cada processador, mas envolveria um

sistema de gerenciamento de memória, portanto optamos pela solução mais simples.

Resolvido o problema da superposição de espaços de endereçamento, devemos permitir a troca de dados entre os processadores. Para tal, reservou-se uma área de memória não tocada pelos *wrappers*. Esta área corresponde aos endereços de 2MB a 3MB, como podemos ver na figura 5.4. Todas as requisições de ambos os processadores para qualquer endereço nesta faixa são propagadas diretamente para o barramento sem alguma modificação. Além do resultado esperado, uma área de memória compartilhada, teremos um efeito colateral indesejado, mas inevitável. Quando o segundo processador fizer um acesso na faixa compartilhada, o pacote propagado será uma requisição para a faixa real de memória (2 a 3MB) e não na faixa mapeada, de 7MB a 8MB. Isto faz com que esta área da memória seja sombreada (*shaded*), ou seja, inacessível.

A quantidade de memória desperdiçada é diretamente proporcional ao número de processadores e ao tamanho da área compartilhada. Neste caso, a área de memória compartilhada é pequena e há somente dois processadores, mas em um *hardware* real esta solução certamente não se aplicaria, pois memórias ocupam grande parte da área de uma pastilha (*die*). Novamente, optamos pela solução com região sombreada por ser a técnica mais simples e por se tratar de uma simulação.

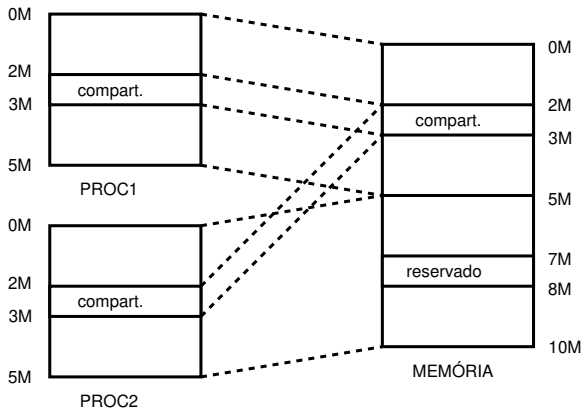

Figura 5.4: Mapeamento de memória da plataforma *dual_mips*.

O *software* escolhido foi o FFT[2] do pacote SPLASH-2 [61]. Este pacote foi desenvolvido especialmente para sistemas paralelos multiprocessados, especificamente com espaço de endereçamento compartilhado. O algoritmo FFT é uma versão em seis passos distintos que utiliza o algoritmo radix-$\sqrt{n}$, onde n é o número de pontos do vetor. Os pontos são organizados em matrizes $\sqrt{n} \times \sqrt{n}$ que são divididas entre os processadores de modo que cada um receba um conjunto contíguo de pontos. A correção foi verificada de duas maneiras: confrontando o resultado obtido pela simulação com o resultado obtido na execução

[2]Transformada Rápida de Fourier, do inglês *Fast Fourier Transform*.

nativa não simulada e com a própria função de teste do algoritmo, que faz a transformada inversa e compara os resultados com o conjunto inicial de pontos.

Em vários pontos é necessário trocar dados entre os processadores ou fazer sincronização entre eles. Para isso, foi desenvolvido um semáforo de Dekker simples [32]. A partir do semáforo, pode-se simular áreas de memória compartilhada com acesso de exclusão mútua, barreiras de sincronização e protocolos de troca de mensagens. No exemplo somente a troca de mensagens não foi utilizada dado que a área de memória compartilhada cumpre a função de troca de dados. O semáforo foi expandido para suportar n processadores, mas atualmente a versão disponível publicamente é restrita a dois processadores por semáforo.

O problema da identificação dos processadores não foi resolvido. Como os processadores possuem área de programa sem superposição, não necessariamente precisam executar o mesmo código. A identificação do processador torna-se portanto desnecessária dado que pode ser embutida no programa. Em nosso caso, as diferenças entre programa mestre e escravo se resumem a uma variável que contém a identificação do processador e as rotinas de preenchimento inicial dos vetores de entrada e impressão de resultados na tela, inexistentes no escravo. Todo o restante do programa, incluindo o algoritmo original do pacote SPLASH, é idêntico para ambos processadores.

Em certas ocasiões, há a necessidade de se executar o mesmo código em todos os processadores. O exemplo clássico é a carga de um sistema operacional em uma plataforma multiprocessada que acaba de ser colocada em execução, pois os processadores iniciarão a busca por instruções no mesmo local da memória (assumindo que todos são idênticos). Para a solução deste problema, deixo como sugestão utilização de interrupção para definir o primeiro processador. Todos os processadores começam em um laço infinito e o primeiro processador interrompido é retirado do laço e se responsabiliza por ser o mestre, coordenando a carga dos demais. É possível provar que não há solução para este problema sem suporte do *hardware*, como instruções do tipo TSL (*Test and Set Lock*) ou uma trava de barramento, mas isto foge ao propósito deste trabalho.

Plataforma DJPEG

Esta plataforma não está disponível publicamente, mas foi utilizada em diversos testes, incluindo o teste da reflexão sobre IP mostrado na seção 5.2. Consiste em uma plataforma de decodificação JPEG, utilizando como base o programa `djpeg` do MediaBench [27]. Esta seção descreve como a plataforma foi construída utilizando as técnicas de modelagem de *hardware* definidas durante o mestrado.

O primeiro passo a se tomar quando se está transformando um *software* em *hardware* é o *profiling*. Para tanto, utilizei o programa GPROF [30], da GNU [26]. Estes programas

instrumentam o código em todas as entradas e saídas de funções marcando o tempo em cada uma delas e gerando um grafo de chamadas de funções, com os respectivos tempos gastos em cada uma. É possível determinar, por exemplo, as funções mais custosas computacionalmente.

No nosso exemplo, a função `jpeg_idct_slow` responde por cerca de 47% do tempo de processamento total do programa, sendo uma candidata natural a se tornar um *hardware* especializado, acelerando a execução final. A função é responsável pelo algoritmo IDCT (*Integer Discrete Cosine Transform*), comum em *softwares* de compressão. Lembro que para ter um *profiling* mais próximo da execução real, deve-se ter um conjunto de entrada representativo, caso contrário funções secundárias como a leitura de arquivos sobressairão. No exemplo, basta que a imagem de entrada utilizada seja grande o suficiente (utilizei um JPEG de 49MB).

O segundo passo é a montagem da plataforma. Esta configuração possui somente um IP representando a função escolhida. Os outros componentes são o barramento, a memória e o processador PowerPC que rodará o restante do *software*. Na figura 5.5 podemos ver o diagrama arquitetural da plataforma.

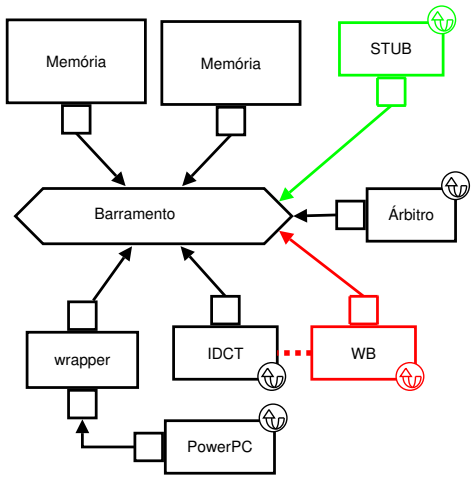

Figura 5.5: Arquitetura da plataforma DJPEG.

A escrita do IP é simples. Como a descrição é funcional, o código original da função `jpeg_idct_slow` foi aproveitado na totalidade, alterando-se apenas os acessos a memória. Como este módulo representa um IP, todos os acessos (tanto leitura quanto escrita) devem ser transformados em requisições ao barramento. Este IP age como escravo de barramento quando está esperando a programação do processador e como mestre de barramento quando está realizando processamento.

```
z2 = DEQUANTIZE(inptr[DCTSIZE*2], quantptr[DCTSIZE*2]);
```

```
z2 = DEQUANTIZE(rmem16s((uint32_t)(&inptr[DCTSIZE*2])),
                rmem32u((uint32_t)(&quantptr[DCTSIZE*2])));
```

No exemplo acima podemos ver dois acessos a memória, ambos uma atribuição à variável `z2`. O primeiro acesso corresponde ao programa original, onde se pode acessar a memória diretamente, e o segundo ao código do IP, onde os acessos são feitos por meio de rotinas de acesso ao barramento. Estas rotinas são semelhantes à rotina `writem` da linha 22 da listagem 3.6 na seção 3.2.

O IP possui vários registradores, cada um representando o que originalmente era um parâmetro da chamada da função original. Um registrador extra foi adicionado para que o processador controle quando os dados são válidos. A parte do *software* correspondente a esta função foi substituída por uma chamada ao IP. Cada chamada coloca os valores dos parâmetros da função original nos respectivos registradores e levanta um bit do registrador de controle sinalizando que os dados são válidos e que o IP pode trabalhar. O IP processa os dados simulando a função original e abaixa o bit do registrador de controle indicando que terminou.

Na próxima seção mostrarei os dados de execução da plataforma.

Outras plataformas

De todas as outras plataformas desenvolvidas, duas merecem destaque por sua relevância técnica: a `arp-os` e a `arp-acmix`.

A `arp-os` é uma plataforma base para as freqüentes tentativas de carga de sistema operacional sobre processadores ArchC. Uma plataforma com um processador SPARC-V8, um barramento, uma memória e um IP de comunicação serial bidirecional foi fornecida para os projetistas, com um exemplo de um micro sistema operacional chamado Contiki [21]. Diversos grupos de trabalho foram formados com o objetivo de executar um sistema operacional qualquer, desde que seja utilizado na atualidade em sistemas embutidos. Desenvolvimentos posteriores resultaram em diversos IPs de suporte como MMU (unidade de gerenciamento de memória[3]) e controladores de interrupção. O grupo mais avançado até o momento já consegue executar uma versão do uCLinux sem suporte a *syscalls*.

A `arp-acmix` foi um projeto de colaboração com o ACMIX [42], resultado do mestrado do aluno Richard Maciel deste instituto, visando co-simulação entre linguagens de diferentes abstrações. A plataforma consiste em um PowerPC ligado a um barramento com uma memória e uma FPU (unidade de ponto flutuante[4]). As instruções de ponto flutuante do processador são enviadas para a FPU como se esta fosse um co-processador

[3]Do inglês: *Memory Management Unit*.
[4]Do inglês: *Floating Point Unit*.

matemático. A FPU foi programada em SystemC e em VHDL, e o papel do ACMIX foi o de ligar o simulador VHDL à plataforma em SystemC.

5.2 Desempenho

O desempenho de uma simulação de *hardware* está ligado a dois fatores principais: o nível de abstração e a ferramenta de simulação. No nosso caso, padronizamos a ferramenta de simulação como sendo o SystemC versão 2.2.05, compilado com o GCC versão 4.1.1. As opções de compilação estão listadas no apêndice B. O compilador utilizado no código que executa nos simuladores de processadores foi compilado com o *cross-compiler* disponível no *site* do ArchC [19], que é baseado no GCC versão 3.3.1.

Quanto ao nível de abstração, limitamos-nos a utilizar em nossos testes a descrição funcional usando TLM e o processador *standalone* do ArchC, com memória interna. Na tabela 5.1 podemos ver uma medida do desempenho das plataformas, em quiloinstruções por segundo (Kips).

Tabela 5.1: Tabela de desempenho, dados em Kips.

Programa	**Máquina A**	**Máquina B**	**Máquina C**	**Máquina D**
Dual MIPS FFT (P1)	324.90	541.16	362.34	344.31
Dual MIPS FFT (P2)	236.57	394.18	263.93	250.80
Hello World TLM	5,778.12	9,933.96	8,578.11	8,067.95
Hello World ArchC	6,666.11	10,967.30	8,491.37	8,267.51
CJPEG MIPS TLM	6,684.07	11,281.80	9,537.32	9,467.79
CJPEG MIPS ArchC	7,136.82	12,086.50	10,290.40	9,356.36
DJPEG MIPS TLM	6,818.18	11,396.30	9,789.12	9,487.11
DJPEG MIPS ArchC	7,579.59	12,405.20	10,744.20	9,849.80
CJPEG SPARC ArchC	12,262.30	20,925.80	12,814.00	12,713.30
DJPEG SPARC ArchC	11,724.20	19,407.50	11,732.50	12,056.80

As aplicações e as plataformas estão descritas abaixo. As plataformas `helloworld` e `dual_mips` foram descritas na seção 5.1. A marcação *TLM* no final de uma plataforma indica que foi utilizada comunicação modelada com técnicas de TLM. Similarmente, a marcação *ArchC* significa que foi utilizada a memória interna do ArchC, sem comunicação externa (somente o simulador de processador).

Dual MIPS FFT Dois processadores MIPS (P1 e P2) conectados a um barramento com árbitro e uma memória como escravo.

Hello World Um processador MIPS e uma memória, sem barramento.

JPEG O codificador (CJPEG) e o decodificador (DJPEG) do pacote MediaBench [27]. A plataforma é idêntica à plataforma `helloworld`.

A caracterização das máquinas utilizadas nos testes pode ser observada abaixo. No apêndice B há uma descrição com mais detalhes.

Máquina A AMD Sempron Mobile 2800+ (1,6GHz) com 256KB de cache, 3201,32 bogomips, kernel 2.6.18-gentoo-r2, Gentoo.

Máquina B AMD Athlon 64 4000+ (2,4GHz) com 1024KB de cache, 4812,63 bogomips, kernel 2.6.15-27-386, Ubuntu.

Máquina C Intel Pentium 4 2,80GHz com 512KB de cache, 5591,51 bogomips, kernel 2.6.15-26-386, Ubuntu.

Máquina D Dual Intel Pentium 4 3,00GHz com 2048KB de cache cada, 5988,36/5984,14 bogomips, kernel 2.6.18-1.2257 SMP, Fedora 5.

Como podemos observar na tabela 5.1, a utilização de TLM reduz o desempenho da plataforma. Por exemplo na plataforma `helloworld`, foi experimentada uma redução de $9,4\%$ de velocidade de simulação. Lembro que as plataformas são similares, diferindo apenas na técnica de modelagem. A mesma plataforma apresenta na máquina C um desempenho ligeiramente superior no caso TLM, mas este é um caso isolado provavelmente ligado a otimizações do compilador, pois o binário da simulação é o mesmo para todas as máquinas e foi compilado com chaves genéricas para arquitetura i386 (estes casos só foram identificados em máquinas Intel).

A redução de desempenho é explícita na plataforma `dual_mips`. Nas outras plataformas apresentadas, há apenas um processo SystemC executando, enquanto que nesta há três: os dois processadores e o árbitro do barramento. Como podemos ver, a velocidade de simulação caiu consideravelmente e de modo não linear. A disputa pelo barramento também contribui para a lentidão da simulação. O fato de o segundo processador ser mais lento que o primeiro já era esperado dado que ele possui prioridade menor e perde todas as disputas de barramento, ficando ocioso até o próximo ciclo.

De modo geral, a utilização de técnicas de modelagem em nível de transações leva a uma redução de desempenho em comparação com a simulação *standalone*, mas permite uma flexibilidade maior do projetista, possibilitando conectar o simulador em dispositivos externos. Em comparação com modelagens de mais baixo nível, o TLM intrinsecamente é mais rápido, como podemos ver nas simulações com AMBA [64, 56].

Na figura 5.6 podemos ver um gráfico da simulação de uma plataforma com vários processadores. A plataforma é composta por um barramento, processadores ArchC e uma memória. O barramento é uma versão modificada do `simple_bus` para suportar transações atômicas de barramento. Os processadores ArchC são dotados de um *wrapper* que emula uma memória cache L1 e a memória em si é uma memória transacional [39]. O *software* executado é um conjunto de produtores e consumidores igualmente distribuídos. As máquinas são as mesmas utilizadas nos experimentos anteriores. O desempenho é a média harmônica do desempenho de todos os processadores da plataforma, interpolado com *splines* (a amostragem foi feita com o número de processadores em potências de 2).

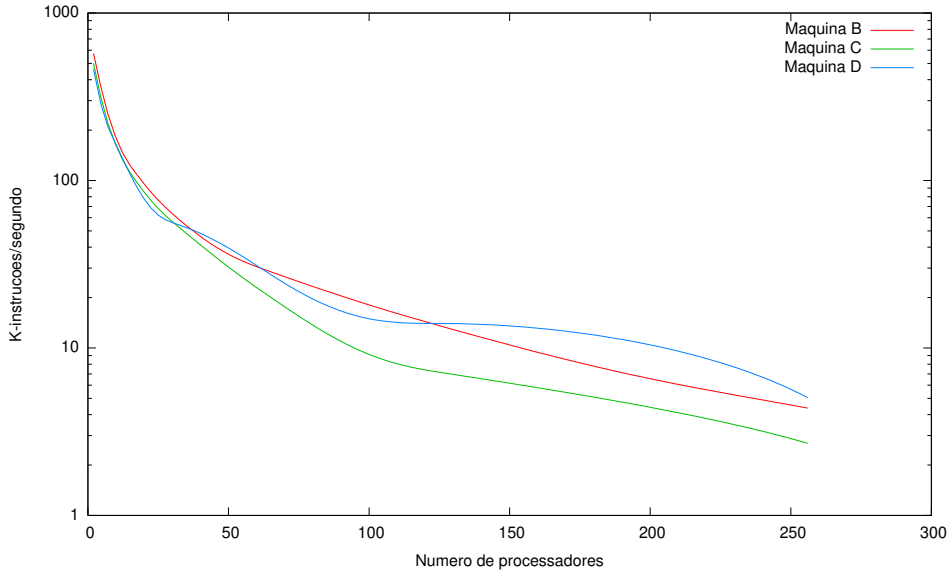

Figura 5.6: Desempenho da plataforma para experimentação de memórias transacionais.

Como pode ser observado, o desempenho cai exponencialmente quando o número de processadores aumenta. Esse comportamento reflete o resultado da plataforma `dual_mips`, confirmando a perda de desempenho mas indicando que o número de processos passa a contribuir para esta queda de desempenho na mesma proporção que o uso do TLM.

Paralelismo no SystemC

Um dos grandes problemas da simulação de *hardware* é a quantidade de processos executados por ciclo de relógio. Em condições normais, há um processo por módulo não combinacional.

O SystemC adota por padrão o modelo de concorrência chamado *quickthreads* onde cada processo é responsável pela própria preempção, direta ou indiretamente. Como o

núcleo de simulação do SystemC é baseado em eventos, o fluxo de execução é único, sem tarefas executando em paralelo, com o objetivo de manter o determinismo.

Como opção ao usuário, o SystemC permite ser compilado utilizando a biblioteca *pthread*, ou POSIX *threads*, mas ela não é utilizada para o paralelismo, apenas para resolver a questão da concorrência ao acesso a estruturas compartilhadas. No código da biblioteca há um comentário sobre esta questão explicitando o uso apenas como alternativa em sistemas que não possuam suporte a *quickthreads*. Em medidas obtidas no laboratório, constatamos que a diferença de desempenho é pequena para poucos processos mas aumenta significativamente para vários processos. O pior caso obtido foi em máquinas com mais de um processador pois o sistema operacional escalona um processo para cada processador. A impressão é que este escalonamento seria interessante mas, lembrando que o fluxo de execução é único, os processos não executarão ao mesmo tempo. Além disso, devido à metodologia baseada em eventos discretos, possuirão dependências de dados entre si, forçando a troca completa de contexto entre duas execuções. Em máquinas deste tipo, a degradação de desempenho chegou a 5,64 vezes. Por não aumentar o grau de paralelismo, a execução com *pthreads* não é recomendada e pode inclusive degradar o desempenho.

Savoiu et al. [54] sugerem que o modelo atual de simulação é deficiente. Além de um excelente resumo das metodologias de simulação utilizadas os autores sugerem um modelo de simulação não baseado em eventos capaz de execução paralela. Para tentar amenizar o impacto da mudança de paradigma nos desenvolvedores de *hardware*, os autores propõem um pré-processador capaz de alterar o fluxo de dados de um modelo SystemC e adaptá-lo ao novo modelo de simulação. O núcleo de simulação do SystemC obviamente deve estar preparado para este novo modelo. Os autores reportam ganhos de desempenho médios de duas vezes mas, considerando a data do artigo (2002), uma reexecução dos testes é mandatória para conclusões mais precisas devido aos avanços nas arquiteturas *multicores*. Pérez et al. [49] propuseram uma modificação do escalonador do núcleo do SystemC capaz de aumentar a velocidade de simulação em até 3.56 vezes utilizando um escalonador acíclico.

Ferramenta de depuração

A ferramenta de depuração desenvolvida foi testada sobre a plataforma descrita na seção 5.1, com e sem IP de reflexão. O IP foi configurado para monitorar o registrador de controle e escrever em um arquivo o seu valor e o tempo de simulação a cada vez que o valor fosse alterado. Quando se utiliza *breakpoint*, a simulação é literalmente parada pelo IP de reflexão. O tempo marcado pelo SystemC não é influenciado pelo tempo que o usuário manteve o sistema parado, mas o ArchC utiliza o tempo total de sistema para

suas estatísticas.

Tabela 5.2: Tabela comparativa da plataforma DJPEG.

Programa	**KIPS**	**Instruções**
Plataforma 1: sem IP	815,55	56379125
Plataforma 2: com IP	774,37	35923071
Plataforma 3: com dois IPs	725,87	35923071
Plataforma 4: com IP e reflexão	726,42	35923071

Na tabela 5.2 podemos ver os dados colhidos pela execução da plataforma em quatro versões, todas na Máquina B caracterizada anteriormente. As colunas mostram a quantidade de instruções executadas por segundo (em milhares de instruções) e a quantidade total de instruções, respectivamente.

A primeira plataforma, sem o IP, aparentemente parece mais rápida que a segunda plataforma, com o IP, entretanto se repararmos na quantidade de instruções executadas pelo MIPS, veremos que é muito menor na plataforma com IP. O ArchC utiliza o tempo "de relógio" da simulação, contando como seu o tempo gasto na simulação do IP. Em um sistema real, a plataforma com IP seria mais rápida que a sem IP pois o processamento que era feito pelo processador agora é feito pelo IP, mais veloz na prática.

A terceira plataforma possui a mesma configuração da anterior (com o IP JPEG) mas contém um IP extra sem funcionalidade alguma (*stub*). Este IP adiciona um processo SystemC à lista de processos ativos, é escalonado em todos os ciclos de simulação e liga-se ao barramento, mas não faz nenhum tipo de computação ou comunicação. Foi inserido propositalmente para efeito de comparação.

A quarta plataforma constitui a plataforma completa com o IP JPEG e o IP de reflexão. Como podemos observar, não foi gerada nenhuma sobrecarga na simulação pela adição do IP de reflexão. A adição do IP na plataforma é equivalente à adição de qualquer outro IP com execução rápida, como o IP *stub* da plataforma 3. De fato, podemos ver que o IP é um pouco mais rápido que o IP nulo da plataforma 3 em sua execução, mas a diferença entre os valores pode ser considerada desprezível.

5.3 Utilização como ferramenta de ensino

A utilização do pacote ArchC como ferramenta de ensino já é consagrada [4] em matérias relacionadas a Arquitetura de Computadores. Há notícias de pelo menos quatro instituições de ensino superior que adotam a metodologia de simulação de plataformas

apresentada neste trabalho. O pacote é suficiente para exploração de uma arquitetura específica isolada, e tem se mostrado excelente para o entendimento de arquiteturas alternativas ou que seriam de difícil acesso para os alunos de outra forma, permitindo inclusive modificações com intuito didático.

A capacidade de extensão conectando-se diversos dispositivos externos e a utilização de mais de um processador abriram novas possibilidades de utilização didática. Matérias envolvendo sistemas paralelos, sistemas operacionais, comunicação de baixo nível e prototipação de *hardware* agora podem fazer uso da infra-estrutura presente. Foram obtidos dois retornos sobre a utilização do conjunto ARP+ArchC, sendo o primeiro relativo a uma matéria relacionada com arquiteturas de computadores e o segundo com prototipação de *hardware*.

A matéria de prototipação de *hardware* foi ministrada na Unicamp pelo professor Sandro Rigo no segundo semestre de 2006. O retorno obtido informalmente dos alunos sem experiência em SystemC e modelagem de plataformas indicou que a maior dificuldade é o projeto da comunicação usando a metodologia TLM. A documentação específica da plataforma utilizada também demonstrou um papel essencial, demonstrando um ponto claro a ser explorado.

A matéria de arquitetura de computadores foi ministrada pelo professor Rodolfo Azevedo na Unicamp. O objetivo foi a exploração da arquitetura de uma plataforma como um todo, com os alunos inclusive escrevendo IPs de alto nível. Houve dificuldades quanto a compatibilidade das bibliotecas com o compilador, mas o sucesso dos alunos em implementar plataformas completas indica que os exemplos da ARP são suficientes para um entendimento introdutório do funcionamento da técnica TLM. Uma quantidade grande de reaproveitamento de estruturas, principalmente *wrappers*, também foi notada.

Além destes cursos na Unicamp, a ARP foi utilizada com fins didáticos em outras duas instituições de ensino superior, em matérias relacionadas com arquiteturas de computadores.

Como menciona Harcourt em [33], a utilização de SystemC torna-se complicada em cursos oferecidos no início da graduação quando a maioria dos alunos mal conhece C++ ou conceitos de orientação a objetos. Neste ponto a ARP destaca-se por apresentar uma curva de aprendizado suave em relação ao SystemC. Os alunos podem explorar uma arquitetura já definida, inclusive com processadores de uso geral, o que considero um ganho em relação às tentativas que vêm sendo feitas de introduzir SystemC diretamente nos cursos de arquiteturas de computadores. Todos os três casos estudados ressaltaram esta característica e em alguns a aplicação de ferramentas utilizando SystemC seria impossível sem tal suporte.

Capítulo 6

Conclusões e Trabalhos Futuros

A ARP até agora serviu ao seu propósito de oferecer um ponto de partida para os usuários que queiram desenvolver plataformas usando os simuladores ArchC. O conjunto de exemplos foi escolhido para reunir em um só lugar a maior parte do conhecimento adquirido durante o mestrado, relativo à utilização e conexão de IPs usando TLM, a modularização de plataformas, o desenvolvimento de *wrappers* e a simulação funcional de plataformas.

A modularização dos componentes permite um alto grau de reusabilidade, comprovada de fato pelas plataformas desenvolvidas pelos usuários as quais contêm trechos de código baseados nos exemplos disponíveis.

Diversos projetos surgiram a partir da idéia da ARP, muitos aproveitando os códigos exemplo e seguindo a sua metodologia de desenvolvimento, como o projeto ARP-OS e o *PDesigner*. O *framework* foi utilizado com relativo sucesso como ferramenta de apoio ao ensino em pelo menos duas matérias no Instituto de Computação e duas em outras universidades. O retorno proporcionado foi fundamental para o aprimoramento das funcionalidades dos *scripts* e da estrutura de *Makefiles*.

Para a continuação deste trabalho, enumero as seguintes sugestões:

- Expansão da base de IPs para conter mais exemplos de modo a cobrir os *hardwares* mais utilizados pelos desenvolvedores;
- Aumentar a integração com as ferramentas gráficas como o *PDesigner*, eliminando a fase de captura manual do projeto. Este trabalho já está em andamento;
- Expandir a interpretação SPIRIT para cobrir toda a especificação, com especial atenção para a versão que atualmente está em fase final de aprovação pelo consórcio, que suporta nativamente descrições TLM.

O trabalho sobre a depuração também merece destaque pois é pioneiro na utilização de reflexão para tal tarefa. A possibilidade de utilização do *WhiteBox* não só como ferramenta de de observação de transações mas como ferramenta de inspeção e modificação dos

módulos em tempo de execução abre um novo paradigma a ser explorado, principalmente com a integração com ferramentas gráficas.

Deixo como sugestão a implementação de uma interface dedicada a depuração, plugável ao módulo de reflexão e capaz de filtrar e gerar estatísticas a partir dos dados coletados dos módulos. Um exemplo de tal aplicação é a determinação da cobertura dos casos de teste de um determinado IP usando o módulo para verificar internamente a utilização de todos os caminhos do fluxo de dados.

Como contribuição acadêmica, o presente trabalho gerou até o presente duas publicações: a primeira sobre uma plataforma [39] gerada utilizando a infra-estrutura descrita, voltada para memórias transacionais e a segunda sobre simulação de plataformas heterogêneas [42] usando VHDL sobre a ARP.

Apêndice A

Simbologia TLM

Padrão TLM

A OSCI [47] disponibiliza o pacote TLM na forma de um arquivo compactado contendo alguma documentação. Um dos artigos disponíveis [53] formaliza o padrão TLM, definindo entre outras coisas o significado dos desenhos utilizados.

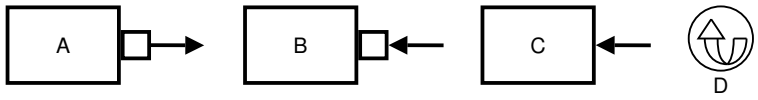

Figura A.1: Simbologia TLM padrão. (a) porta, (b) porta especial, (c) canal e (d) processo.

- **A** – seta saindo de um quadrado ligado ao módulo;
- **B** – seta entrando em um quadrado ligado ao módulo;
- **C** – seta entrando diretamente em um módulo;
- **D** – seta curva (usada anexa a um módulo).

Na figura A.1 podemos ver os quatro símbolos utilizados. O primeiro símbolo (A) representa uma porta do tipo `sc_port`. Normalmente, este é o tipo de porta utilizado em descrições com canais. Esta porta necessariamente deve estar ligada a um canal (símbolo (C)) que implementa algum tipo de interface.

A partir da versão 2.1 do SystemC, está disponível um tipo especial de porta, chamado de `sc_export`, representado pelo símbolo (B). O símbolo (D) indica que o módulo possui ao menos um processo, o que trata a conexão com a porta.

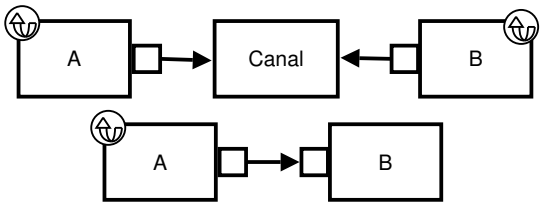

Figura A.2: Exemplos de uso TLM.

Como podemos ver na figura A.2, a vantagem da `sc_export` é justamente eliminar o canal. No primeiro desenho, temos uma conexão através de duas portas `sc_port` que implementam a interface de envio e a interface de recebimento de dados, ambas conectadas no canal. Neste caso, são necessários processos em ambos os módulos, obrigatoriamente, dado que o canal não é capaz de sinalizar a presença de dados ao módulo receptor por outro meio.

O segundo desenho representa a mesma conexão com o uso de `sc_export`. Não há presença de canal e seu papel é representado pela própria porta. O módulo receptor não possui processos, pois o método receptor é invocado diretamente pela porta. Este método possui mais velocidade de simulação em detrimento da modularidade (a interface receptora é implementada no próprio módulo).

Apêndice B

Caracterização de Ferramentas e Máquinas

Os dados a seguir são provenientes dos seguintes comandos:

Processador `cat /proc/cpuinfo`

Kernel `uname -a`

Memória `cat /proc/meminfo` (somente o total)

Máquina A

```
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 15
model           : 28
model name      : Mobile AMD Sempron(tm) Processor 2800+
stepping        : 0
cpu MHz         : 1600.000
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
```

```
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 syscall nx mmxext fxsr_opt
3dnowext 3dnow lahf_lm ts fid vid ttp
bogomips        : 3201.32

Linux katrina 2.6.18-gentoo-r2 #1 Tue Nov 21 09:54:55 BRST 2006 i686
Mobile AMD Sempron(tm) Processor 2800+ AuthenticAMD GNU/Linux

MemTotal:       449600 kB
```

Máquina B

```
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 15
model           : 7
model name      : AMD Athlon(tm) 64 Processor 4000+
stepping        : 10
cpu MHz         : 2403.736
cache size      : 1024 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 syscall nx mmxext lm 3dnowext
3dnow
bogomips        : 4812.63

Linux node12 2.6.15-27-386 #1 PREEMPT Fri Dec 8 17:51:56 UTC 2006 i686
GNU/Linux

MemTotal:      2076208 kB
```

Máquina C

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 2
model name      : Intel(R) Pentium(R) 4 CPU 2.80GHz
stepping        : 5
cpu MHz         : 2793.592
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe cid xtpr
bogomips        : 5591.51

Linux node06 2.6.15-26-386 #1 PREEMPT Thu Aug 3 02:52:00 UTC 2006 i686
GNU/Linux

MemTotal:      1019076 kB
```

Máquina D

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 4
model name      : Intel(R) Pentium(R) 4 CPU 3.00GHz
stepping        : 3
cpu MHz         : 2800.000
cache size      : 2048 KB
physical id     : 0
siblings        : 2
```

```
core id         : 0
cpu cores       : 1
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx lm
constant_tsc pni monitor ds_cpl est cid cx16 xtpr
bogomips        : 5988.36

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 15
model           : 4
model name      : Intel(R) Pentium(R) 4 CPU 3.00GHz
stepping        : 3
cpu MHz         : 2800.000
cache size      : 2048 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 1
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx lm
constant_tsc pni monitor ds_cpl est cid cx16 xtpr
```

```
bogomips        : 5984.14

Linux xaveco.lab.ic.unicamp.br 2.6.18-1.2257.fc5smp #1 SMP Fri Dec 15
16:33:51 EST 2006 i686 i686 i386 GNU/Linux

MemTotal:      1033492 kB
```

Referências Bibliográficas

[1] ACE. Website, janeiro de 2007. `http://www.ace.nl`.

[2] Adams Arne. AReflection. Website, janeiro de 2007. `http://www.arneadams.com/reflection\_doku/`.

[3] AspectC++. C++ Aspect Oriented Programming. Website, janeiro de 2007. `http://www.aspectc.org/`.

[4] Rodolfo Azevedo, Sandro Rigo, Marcus Bartholomeu, Guido Araújo, Cristiano Araújo e Edna Barros. The ArchC Architecture Description Language and Tools. *International Journal of Parallel Programming*, 33(5):453–484, 2005.

[5] Beach solutions. Website, janeiro de 2007. `http://www.beachsolutions.com`.

[6] Luca Benini e Giovanni De Micheli. Networks on Chips: A New SoC Paradigm. *Computer*, 35(1):70–78, 2002.

[7] Bluespec. Website, janeiro de 2007. `http://www.bluespec.com`.

[8] Boost. Boost C++ Libraries. Website, janeiro de 2007. `http://www.boost.org`.

[9] Gilad Bracha e David Ungar. Mirrors: Design Principles for Meta-level Facilities of Object-Oriented Programming Languages. Em *Proceedings of the 19th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04)*, pp. 331–344, New York, NY, USA, 2004. ACM Press, Vancouver, BC, Canada.

[10] Cadence. Website, janeiro de 2007. `http://www.cadence.com`.

[11] L. Cai e D. Gajski. Transaction Level Modeling: An Overview. Em *Proceedings of the Hardware/Software Codesign and System Synthesis conference (CODES+ISSS'03)*, pp. 19–24, outubro de 2003.

[12] Celoxica. Website, janeiro de 2007. `http://www.celoxica.com`.

[13] Shigeru Chiba. OpenC++. Website, janeiro de 2007. `http://opencxx.sourceforge.net/`.

[14] CIn. PDesigner. Website, janeiro de 2007. `http://www.pdesigner.org`.

[15] SPIRIT Consortium. SPIRIT. Website, janeiro de 2005. `http://www.spiritconsortium.org/`.

[16] Marcello Coppola, Stephane Curaba, Miltos D. Grammatikakis, Giuseppe Maruccia e Francesco Papariello. OCCN: A Network-on-Chip Modeling and Simulation Framework. Em *Proceedings of the conference on Design, Automation and Test in Europe (DATE'04)*, p. 30174, Washington, DC, USA, 2004. IEEE Computer Society.

[17] Coware. Website, janeiro de 2007. `http://www.coware.com`.

[18] D. Crisu, S. D. Cotofana, S. Vassiliadis e P. Liuha. GRAAL – A Development Framework for Embedded Graphics Accelerators. Em *Proceedings of the conference on Design, Automation and Test in Europe (DATE'04)*, p. 21366, Washington, DC, USA, 2004. IEEE Computer Society.

[19] LSC (Laboratório de Sistemas Computacionais). ArchC. Website, janeiro de 2007. `http://www.archc.org`.

[20] LSC (Laboratório de Sistemas Computacionais). LSC. Website, janeiro de 2007. `http://www.lsc.ic.unicamp.br`.

[21] Adam Dunkels, Björn Grönvall e Thiemo Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. Em *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (EMNETS'04)*, Tampa, Florida, USA, novembro de 2004.

[22] David Déharbe e Sergio Medeiros. Aspect-oriented design in SystemC: Implementation and applications. Em *Proceedings of the 19th annual symposium on Integrated circuits and systems design (SBCCI'06)*, pp. 119–124, New York, NY, USA, 2006. ACM Press, Ouro Preto, MG, Brazil.

[23] Eclipse. Website, janeiro de 2007. `http://www.eclipse.org`.

[24] A. Fauth, J. Van Praet e M. Freericks. Describing instruction set processors using nML. Em *Proceedings of the 1995 European conference on Design and Test (EDTC'95)*, p. 503, Washington, DC, USA, 1995. IEEE Computer Society.

[25] ITRS (International Technology Roadmap for Semiconductors). ITRS. Website, janeiro de 2007. `http://www.itrs.net`.

[26] Free Software Foundation. GNU. Website, janeiro de 2007. `http://www.gnu.org`.

[27] Jason Fritts et alli. Mediabench. Website, janeiro de 2007. `http://euler.slu.edu/~fritts/mediabench/`.

[28] Felix Garcia e Javier Fernandez. POSIX thread libraries. *Linux J.*, 2000(70es):36, 2000.

[29] Christian Genz e Rolf Drechsler. System exploration of systemC designs. Em *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, p. 335, Washington, DC, USA, 2006. IEEE Computer Society.

[30] GNU. GPPROF: GNU profiler. Website, janeiro de 2007. `http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html`.

[31] Ashok Halambi, Peter Grun, Vijay Ganesh, Asheesh Khare, Nikil Dutt e Alex Nicolau. EXPRESSION: A language for architecture exploration through compiler/simulator retargetability. Em *Proceedings of the conference on Design, Automation and Test in Europe (DATE'99)*, p. 100, New York, NY, USA, 1999. ACM Press, Munich, Germany.

[32] Per Brinch Hansen. *Operating System Principles*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1973.

[33] Ed Harcourt. Teaching computer organization and architecture using SystemC. *J. Comput. Small Coll.*, 21(2):27–39, 2005.

[34] IBM. PowerPC evaluation kit. Website, janeiro de 2007. `http://www-128.ibm.com/developerworks/power/pek/`.

[35] C. Norris Ip e Sturat Swan. A tutorial introduction on the new SystemC verification standard. `http://www.systemc.org`, janeiro de 2003.

[36] Torsten Kempf, Malte Doerper, R. Leupers, G. Ascheid, H. Meyr, Tim Kogel e Bart Vanthournout. A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC platforms. Em *Proceedings of the conference on Design, Automation and Test in Europe (DATE'05)*, pp. 876–881, Washington, DC, USA, 2005. IEEE Computer Society.

[37] Brad King. GCC_XML. Website, janeiro de 2007. `http://www.gccxml.org/`.

[38] Konstantin Knizhnik. CPP reflection package. Website, janeiro de 2007. `http://www.garret.ru/~knizhnik/`.

[39] Fernando Kronbauer, Alexandro Baldassin, Bruno Albertini, Paulo Centoducatte, Sandro Rigo, Guido Araujo e Rodolfo Azevedo. A flexible platform framework for rapid transactional memory systems prototyping and evaluation. Em *18th IEEE/IFIP International Workshop on Rapid System Prototyping*, pp. 123–129, maio de 2007.

[40] James Lapalme, El Mostapha Aboulhamid e Gabriela Nicolescu. A new efficient EDA tool design methodology. *Trans. on Embedded Computing Sys.*, 5(2):408–430, 2006.

[41] Fábio Lombardelli. CPPReflect. Website, janeiro de 2007. `http://sourceforge.net/projects/cppreflect/`.

[42] Richard Maciel, Bruno Albertini, Sandro Rigo, Guido Araujo e Rodolfo Azevedo. A methodology and toolset to enable systemC and VHDL co-simulation. Em *IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07)*, pp. 351–356, março de 2007.

[43] Pattie Maes. Concepts and experiments in computational reflection. Em *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications (OOPSLA'87)*, pp. 147–155, New York, NY, USA, 1987. ACM Press, Orlando, Florida, United States.

[44] Mentor graphics. Website, janeiro de 2007. `http://www.mentor.com`.

[45] Matthieu Moy, Florence Maraninchi e Laurent Maillet-Contoz. Pinapa: An extraction tool for SystemC descriptions of systems-on-a-chip. Em *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT'05)*, pp. 317–324, New York, NY, USA, 2005. ACM Press, Jersey City, NJ, USA.

[46] Osamu Ogawa, Sylvain Bayon de Noyer, Pascal Chauvet, Katsuya Shinohara, Yoshiharu Watanabe, Hiroshi Niizuma, Takayuki Sasaki e Yuji Takai. A practical approach for bus architecture optimization at transaction level. Em *Proceedings of the conference on Design, Automation and Test in Europe (DATE'03)*, p. 20176, Washington, DC, USA, 2003. IEEE Computer Society.

[47] OSCI. SystemC library. Website, janeiro de 2007. `http://www.systemc.org`.

[48] Stefan Pees, Andreas Hoffmann, Vojin Zivojnovic e Heinrich Meyr. LISA - machine description language for cycle-accurate models of programmable DSP architectures. Em *Proceedings of the 36th ACM/IEEE conference on Design automation (DAC'99)*, pp. 933–938, New York, NY, USA, 1999. ACM Press, New Orleans, Louisiana, United States.

[49] Daniel Gracia Pérez, Gilles Mouchard e Olivier Temam. A new optimized implemention of the systemC engine using acyclic scheduling. Em *Proceedings of the conference on Design, Automation and Test in Europe (DATE'04)*, p. 10552, Washington, DC, USA, 2004. IEEE Computer Society.

[50] E. Riccobene, P. Scandurra, A. Rosti e S. Bocchio. A model-driven design environment for embedded systems. Em *Proceedings of the 43rd annual Conference on Design Automation (DAC'06)*, pp. 915–918, New York, NY, USA, 2006. ACM Press, San Francisco, CA, USA.

[51] Sandro Rigo, Guido Araújo, Marcus Bartholomeu e Rodolfo Azevedo. ArchC: A SystemC-based architecture description language. Em *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)*, pp. 66–73, Washington, DC, USA, 2004. IEEE Computer Society.

[52] S. Roiser e P. Mato. The SEAL C++ reflection system. Em *CHEP '04: Presented in the Computing in High Energy and Nuclear Physics congress (CHEP'04)*. CERN, setembro de 2004.

[53] Adam Rose et alli. Transaction modeling with SystemC. disponível em `http://www.systemc.org` com o pacote TLM.

[54] N. Savoiu et alli. Automated concurrency re-assignment in high level system models for efficient system-level simulation. Em *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*, pp. 875–881. IEEE Computer Society, 2002. `http://ieeexplore.ieee.org/iel5/7834/21541/00998404.pdf`.

[55] Patrick Schaumont, Doris Ching e Ingrid Verbauwhede. An interactive codesign environment for domain-specific coprocessors. *ACM Trans. Des. Autom. Electron. Syst.*, 11(1):70–87, 2006.

[56] G. Schirner e R. Domer. Quantitative analysis of transaction level models for the AMBA bus. Em *Proceeding of the Design, Automation and Test in Europe conference (DATE'06)*, volume 1, pp. 1–6, março de 2006.

[57] Bill Spitzak et alli. FLTK fast light toolkit. Website, janeiro de 2007. `http://www.fltk.org/`.

[58] S. Swan. SystemC transaction level models and RTL verification. Em *Proceedings of the Design, Automation and Test in Europe conference (DATE'06)*, pp. 90–92, julho de 2006.

[59] Stuart Swan. An introduction to system level modeling in SystemC 2.0. `http://www.systemc.org`, maio de 2001.

[60] Synopsys. Website, janeiro de 2007. `http://www.synopsys.com`.

[61] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh e Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. Em *Proceedings of the 22nd annual International Symposium on Computer Architecture (ISCA'95)*, pp. 24–36, New York, NY, USA, 1995. ACM Press, S. Margherita Ligure, Italy.

[62] Jinwen Xi e Peixin Zhong. A transaction-level NoC simulation platform with architecture-level dynamic and leakage energy models. Em *Proceedings of the 16th ACM Great Lakes symposium on VLSI (GLSVLSI'06)*, pp. 341–344, New York, NY, USA, 2006. ACM Press, Philadelphia, PA, USA.

[63] H. Xu, S.and Pollitt-Smith. A TLM platform for system-on-chip simulation and verification. Em *VLSI Design, Automation and Test (VLSI-TSA-DAT)*, pp. 220–221. IEEE Computer Society, abril de 2005.

[64] Kim Young-Taek, Kim Taehun, Kim Youngduk, Shin Chulho, Chung Eui-Young, Choi Kyu-Myung, Kong Jeong-Taek e Eo Soo-Kwan. Fast and accurate transaction level modeling of an extended AMBA2.0 bus architecture. Em *Proceedings of the Design, Automation and Test in Europe conference (DATE'05)*, volume 3, pp. 7–11, março de 2005.