Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

Estimativa de Desempenho de Software e Consumo de Energia em MPSoCs

Sérgio Johann Filho

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Fabiano Passuelo Hessel

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

J65e Johann Filho, Sérgio
Estimativa de desempenho de software e consumo de energia em MPSoCs / Sérgio Johan Filho. – Porto Alegre, 2008.
81 f.

Diss. (Mestrado em Ciência da Computação) – Fac. de Informática, PUCRS.
Orientação: Prof. Dr. Fabiano Passuelo Hessel.

1. Informática. 2. Multiprocessadores. 3. Arquitetura de Computador. 4. Energia Elétrica – Consumo. I. Hessel, Fabiano Passuelo.

CDD 004.35

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada ***"Estimativa de Desempenho de Software e Consumo de Energia em MPSoCs"***, apresentada por Sergio Johann Filho, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 04/03/08 pela Comissão Examinadora:

Prof. Dr. Fabiano Passuelo Hessel – Orientador — PPGCC/PUCRS

Prof. Dr. Eduardo Augusto Bezerra – PPGCC/PUCRS

Prof. Dr. César Augusto Missio Marcon – FACIN/PUCRS

Prof. Dr. Antonio Augusto Medeiros Fröhlich - UFSC

Homologada em 04/12/2009, conforme Ata No. 21/09 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

Este trabalho é dedicado a minha família e amigos.

Agradecimentos

Foram dois anos de muito estudo, esforço, noites mal dormidas e é chegada a hora da conclusão deste projeto. Nenhum esforço foi em vão, entretanto. O trabalho de pesquisa é incessante e envolve inspiração, motivação, idéias e aplicação de conhecimentos adquiridos no decorrer do caminho. Esta tarefa - a pesquisa - seria infinitamente penosa se não houvesse ajuda da família, amigos e professores.

Em primeiro lugar, gostaria de agradecer a família (meu pai Sérgio, minha mãe Madalena, meus irmãos Marcelo e Rodrigo) pela amizade, incentivo e carinho, durante todo o desenvolvimento dos trabalhos realizados. Vocês ouviram falar de parte das minhas atividades, e me aguentaram, sem reclamar. =)

Aos colegas, professores e amigos, meu agradecimento, pelo espírito de equipe, companheirismo e trabalho duro. As cervejas Polar e Antártica, nos momentos em que precisei de um empurrãozinho.

E por último, porém não menos importante, agradeço as minhas guitarras e equipamento, por me manterem focado e consciente ao longo dos anos.

Resumo

Para atender a uma cresente demanda por desempenho de processamento, o projeto de sistemas embarcados inclui a utilização de diversos processadores além de infra-estruturas de comunicação complexas (por exemplo, barramentos hierárquicos e redes intra-chip). Há uma crescente demanda por um número cada vez maior de funcionalidades contidas em um único sistema. Neste cenário, questões relacionadas a estimativas de consumo de energia ganham importância no projeto de sistemas eletrônicos embarcados.

Dessa forma, o fluxo de projeto de sistemas embarcados multi-processados necessita de ferramentas para a geração de estimativas de desempenho e consumo de energia durante todo o ciclo de desenvolvimento, de forma a verificar se o caminho de construção do projeto condiz com a especificação do mesmo. O desempenho, assim como o consumo de energia de um determinado sistema precisam ser avaliadados o mais cedo possível no fluxo de projeto.

Métodos analíticos são propostos para que estimativas de desempenho e de consumo de energia possam ser realizadas de maneira rápida, evitando tempos proibitivos de simulação. Nos métodos analíticos o sistema é modelado como uma série de propriedades e modelos abstratos são utilizados para o cálculo do desempenho do sistema. Apesar de métodos analíticos serem mais rápidos que métodos baseados em simulação a modelagem do sistema é mais complexa. Além disso, devido ao alto nível de abstração em que o sistema é representado, seu uso em sistemas grandes e complexos se torna inviável devido a explosão de estados necessários para a representação sistêmica destes, que é o caso de recentes projetos de sistemas embarcados. Dessa forma, melhorias nos métodos baseados em simulação tornam-se bastante pertinentes, e um estudo dessa área é apresentado nesse trabalho.

Palavras-chave: MPSoC, Desempenho, Consumo de energia, Estimativa

Abstract

To supply the ever-increasing need for processing power, the embedded software project includes the utilization of several processors along with complex communication infrastructures (as hierarchycal buses and networks-on-a-chip). There is an increasing need for a greater number of functionalities inside a single system. In this scenario, issues related to energy consumption estimations become important in the embedded electronic systems project.

This way, the multi-processor embedded systems workflow needs tools to generate performance and energy consumption estimations during all development cycle, in order to verify if the project building process conforms to its specification. The performance, as the energy consumption of a system have to be evaluated as soon as possible in the workflow.

Analytical methods are proposed to allow performance and energy estimations in a fast way, avoiding prohibitive simulation times. In analytical methods the system is modeled as a series of properties and abstract models are used to calculate the system performance. Although analytical methods are faster than simulation ones, their modelling is more complex. Along with this fact, the high abstraction level in which the system is represented becomes unfeasible due to the high increase in states necessary to represent such systems, which is the case of more recent embedded systems. This way, better approaches in simulation based methods become very interesting, and a study in this field is presented in this work.

Keywords: MPSoC, Performance, Energy consumption, Estimation

Lista de Figuras

Lista de Tabelas

Lista de Siglas

CI	Circuitos Integrados	23
SoC	System-on-a-Chip	23
IP	Itellectual Property	23
MPSoC	Multi-Processor System-on-a-Chip	23
DSP	Digital Signal Processor	23
OCP	Open Core Protocol	24
AMBA	Advanced Microcontroller Bus Architecture	24
FPGA	Field Programmable Gate Array	26
RTL	Register Transfer Level	26
ISS	Instruction Set Simulator	27
ADL	Architecture Description Language	30
SMP	Symetric Multi-Processing	31
TLM	Transaction Level Model	32
BCA	Bus Cycle Accurate	33
RISC	Reduced Instruction Set Computer	33
MIPS	Millions of Instructions Per Second	34
RMS	Rate Monotonic Scheduling	35
EDF	Earliest Deadline First	35
TDMA	Time-Division Multiple Accesses	35
CFG	Control Flow Graph	36
WCET	Worst Case Execution Time	36
NoC	Network on a Chip	37
GNU	GNU's Not UNIX	38
GCC	GNU Compiler Collection	38
MOSFET	Metal-Oxide Semiconductor Field-Effect Transistor	39

ET	Execution Time	41
VHDL	Very High-Speed Hardware Description Language	43
UART	Universal Asynchronous Receiver-Transmitter	47
LED	Light Emitter Diode	57
VGA	Video Graphics Array	57
DDR	Double Data Rate	58
SDRAM	Synchronous Dynamic Random Access Memory	58
SRAM	Static Random Access Memory	60
ANSI	American National Standards Institute	65
IPC	Instructions Per Cycle	66
JPEG	Joint Photographic Experts Group	70
CRC	Cyclic Redundancy Check	71
ADPCM	Adaptive Differential Pulse Code Modulation	71

Sumário

1 Introdução

O projeto de sistemas digitais complexos tem avançado muito atualmente, juntamente com um aumento da freqüência de operação de um circuito e do número de transistores em uma mesma pastilha de silício. Em 1980, a maioria dos circuitos integrados (CIs) complexos eram compostos por dezenas de milhares de transistores, e nesta última década já é possível se encontrar CIs com mais de dezenas de milhões de transistores [41]. Esse avanço da tecnologia tem permitido que múltiplos componentes, como processadores, controladores e memória, sejam integrados em uma única pastilha, formando um sistema completo [9]. Esses sistemas são conhecidos como SoCs (do inglês, System-on-a-Chip), e possuem diversos núcleos de propriedade intelectual (do inglês, *IP cores*) que são blocos de circuito, pré-projetado e pré-verificado, utilizados no projeto de um sistema complexo [35].

Certos sistemas embarcados devem considerar requisitos temporais, além das restrições de potência, custo de desenvolvimento, e da pressão exercida pelo mercado - o *time-to-market* - que são características inerentes a este tipo de sistema. Por exemplo funcionalidades multimídia, sistemas de navegação, entre outros, impõem restrições de tempo real para que o comportamento seja executado corretamente no tempo esperado, pois a resposta correta após o tempo esperado (resposta atrasada) é, para este caso, uma resposta errada. Dispositivos móveis alimentados por baterias requerem, ainda, capacidades de baixo consumo de energia. A pressão exercida pelo mercado demanda um projeto rápido, mas, por outro lado, esses produtos precisam ter custo baixo e requerem, dessa forma, um projeto otimizado.

Flexibilidade é outro requisito importante no projeto dos sistemas embarcados atuais. O projeto precisa ser flexível para aceitar novas funcionalidades sem a necessidade de reprojeto. Microprocessadores têm importante papel na flexibilidade de um novo sistema embarcado. Assim os atuais sistemas possuem um ou mais microprocessadores, que podem ser da mesma família, ou de diferentes famílias. Essas soluções chamadas de MPSoCs (do inglês, Multi-Processor System-on-a-Chip), precisam de novas ferramentas e modelos de programação para lidar com a complexidade inerente destes sistemas.

A Figura 1 representa uma arquitetura MPSoC típica, composta por um ou mais processadores, DSPs e componentes de *hardware*. O meio de interconexão interliga os componentes do MPSoC. Interfaces de comunicação (adaptadores de *hardware*) são utilizados para interconectar

microprocessadores e DSPs ao meio de interconexão. Além disso, estas interfaces são responsáveis por adaptar o protocolo de comunicação dos microprocessadores e DSPs ao protocolo de comunicação do meio de interconexão. Com o objetivo de facilitar a interconexão dos diversos componentes (microprocessadores, DSPs, módulos IP) ao meio de interconexão, é possível utilizar padrões como OCP [2] ou AMBA, [1] por exemplo. Em cada microprocessador, uma instância de sistema operacional realiza a gerência das tarefas que executam nesse. Os processadores podem acessar o DSP para acelerar determinada função, que pode alternativamente ser executada em *software*.

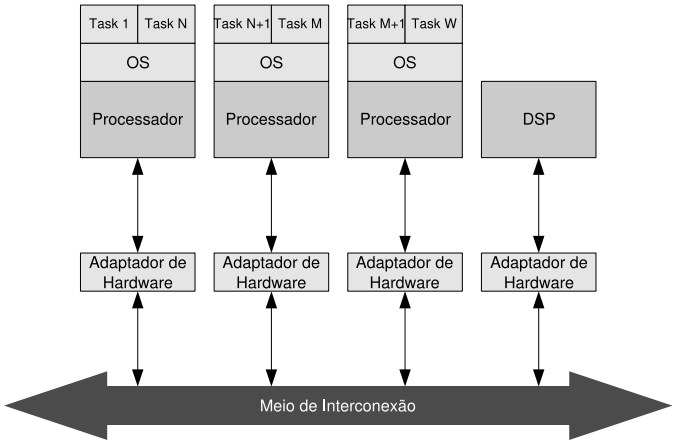

Figura 1 – Exemplo de solução MPSoC.

Em uma arquitetura MPSoC, componentes de *hardware* e *software* precisam ser considerados como um conjunto único, ou seja, devem estar intimamente integrados. Atualmente há uma dificuldade nessa integração, uma vez que esta é feita apenas quando um protótipo dos componentes de *hardware* está disponível para testes. Dessa forma, problemas de integração são normalmente encontrados em um estágio bastante avançado do projeto, o que pode inviabilizá-lo. Desta forma, ferramentas que permitam a validação do sistema como um todo, e não apenas a sua funcionalidade mas também suas restrições de projeto, se fazem necessárias em níveis de abstração superiores.

1.1 Estimativa de desempenho e energia no projeto de MPSoCs

O fluxo de projeto de MPSoCs necessita de ferramentas de estimativa e energia durante seu desenvolvimento, para estimar os requisitos relevantes da especificação tais como desempenho

e consumo de energia do sistema, de forma a verificar se a construção do projeto atende sua especificação inicial.

Em níveis de abstração superiores, a exploração do espaço de soluções se tornou um fator de extrema importância na concepção dos sistemas embarcados atuais. Neste contexto, estimativas passam a ter um papel fundamental nas decisões de projeto [29]. No entanto, ferramentas que realizam estimativas em níveis de abstração superiores, como requer o projeto de sistemas embarcados atuais, são raras [29]. Além disso, estas ferramentas precisam ser integradas em um fluxo de concepção para que a equipe de projetistas possa avaliar diversas possibilidades de implementação de um determinado projeto o mais cedo possível.

Estimativas de desempenho e consumo de energia são tratadas como um processo contínuo e podem ser aplicadas em diferentes níveis de abstração durante todo o fluxo de projeto. No nível de especificação (funcionalidade da arquitetura, ou seja, não são levadas em conta diversas implicações) são realizadas as tarefas de seleção de um determinado processador, assim como o particionamento de *hardware* e *software* e distribuição de tarefas (ou aplicações) entre processadores. A interconexão e o particionamento de *hardware* e *software* podem ser exploradas em um nível de arquitetura virtual, ou seja, as interfaces entre os componentes do sistema são claras, contudo não são modelados detalhes como latências e contagem de ciclos [4]. O nível funcional de interconexão inclui interfaces que descrevem a comunicação de forma precisa, e o *software* é executado em um simulador em nível de instrução ou de ciclo [6]. Este último nível, permite uma avaliação da comunicação mais detalhada, como por exemplo o uso de pacotes que transitam no meio de interconexão, o tempo que os pacotes levam da origem ao destino, entre outros.

1.2 Motivação

O número de processadores em MPSoCs têm crescido de forma acentuada, e dessa forma, funcionalidades que antes eram executadas pelo *hardware* agora são executadas em *software*, aumentando a complexidade do mesmo de forma considerável. Para lidar com esse fato, ferramentas para estimativas em níveis de abstração superiores tornam-se necessárias, pois o tempo de projeto é essencialmente curto [5] [16] [47]. Ferramentas para a estimativa de desempenho de *software* e consumo de energia requerem alta generalidade (possibilidade de adaptação para diferentes arquiteturas) e corretude. Contudo, ferramentas de estimativa devem requerer um baixo esforço para modelagem e avaliação, de tal forma que o tempo necessário para a avaliação seja consideravelmente pequeno. Na maioria dos casos, a corretude conflita com o

esforço para modelagem e avaliação. Estimativas precisas podem ser obtidas com um modelo detalhado, o qual requer um alto esforço de modelagem e tempo de computação. Por outro lado, modelos mais abstratos requerem menos esforço de modelagem ao preço de uma perda de precisão.

As ferramentas de estimativa podem ser classificadas como ferramentas de simulação e ferramentas analíticas. Ferramentas analíticas se baseiam em diferentes modelos matemáticos, como por exemplo cadeias de Markov, redes neurais e funções matemáticas para calcular o tempo de execução do *software* e seu consumo de energia em uma determinada arquitetura. Nos métodos analíticos o sistema é modelado de forma abstrata, como uma série de propriedades, e modelos são utilizados para o cálculo do desempenho do sistema. Ferramentas de simulação utilizam algoritmos que representam a execução do *hardware* para estimar o tempo de execução do *software* e consumo de energia. Existem métodos híbridos que processam anotações em nível de instrução ou bloco básico com o custo de execução do mesmo, sendo mais rápidos que simuladores em nível de ciclo.

Estimativas utilizam diferentes abstrações para modelar os componentes de um sistema e sua comunicação. O uso de diferentes níveis de abstração permite que se faça um balanço entre a velocidade da estimação e a precisão dos resultados [36].

Métodos analíticos são propostos para que uma estimativa possa ser realizada de maneira rápida, evitando-se dessa forma tempos proibitivos de simulação. Apesar de métodos analíticos serem mais rápidos que métodos baseados em simulação a modelagem é mais complexa. Além disso, se torna inviável a modelagem de sistemas grandes e complexos, como os recentes projetos MPSoC, devido a explosão dos estados necessários para sua correta representação. Dessa forma, o estudo aprofundado de melhorias nos métodos baseados em simulação tornam-se pertinentes, e são motivação para este trabalho.

1.3 Objetivos

Os objetivos do presente trabalho são a proposição e a criação de uma plataforma para estimativas de desempenho e consumo de energia de uma arquitetura MPSoC. Uma ferramenta foi construída com base em uma plataforma de *hardware* já especificada, implementada e prototipada em FPGA.

A ferramenta foi implementada com base em modelos de nível RTL. A partir desses modelos, foram extraídas informações relacionadas a contagem de ciclos de relógio para a execução de trechos de código nativo da arquitetura e consumo de energia de cada instrução executada.

Essas informações são então utilizadas pelo ISS (do inglês, Instruction-Set Simulator) e pelo modelo do meio de interconexão.

Desta forma é possível estimar de forma mais rápida o desempenho e o consumo de energia da plataforma MPSoC.

1.4 Contribuições

A principal contribuição deste trabalho é a ferramenta de estimativas de desempenho e consumo de energia implementada, além da apresentação de uma metodologia que torna possível a extensão da mesma, uma vez que os conceitos apresentados podem ser aplicados para a caracterização de outras arquiteturas.

Com a realização deste trabalho obteve-se um fluxo de estimativas para plataformas MPSoC, visando suprir as deficiências na obtenção de resultados imediatos, evitando-se inicialmente longos tempos de simulação para uma avaliação de projeto preliminar. Além disso, uma plataforma pré-definida e a implementação de uma ferramenta para estimativas auxiliariam na generalidade e reuso de componentes de *hardware* para medições de desempenho de *software* e estimativa de consumo de energia.

1.5 Organização do texto

No Capítulo 2 serão apresentados os trabalhos relacionados, considerando estimativas baseadas em simulação e estimativas baseadas em métodos analíticos. No Capítulo 3 será apresentada uma proposta de plataforma para estimativas. No Capítulo 4, uma ferramenta de estimativas desenvolvida com base na plataforma proposta será apresentada e por fim, nos Capítulos 5 e 6 serão mostrados estudos de caso e apresentadas as conclusões e trabalhos futuros.

2 Trabalhos relacionados

O desenvolvimento de métodos para estimativas e ferramentas de análise de *hardware* e *software* é uma área ativa de pesquisas. No projeto de MPSoCs, estimativas de desempenho e consumo de energia são complexas e requerem métodos que levam em conta especificações de todo o sistema, possibilitando a análise integrada de diferentes processadores, componentes de *hardware* e o meio de interconexão. Dessa forma, torna-se necessário o desenvolvimento de métodos rápidos e precisos de estimativa, em altos níveis de abstração.

Nas próximas seções serão apresentados métodos para a estimativa de desempenho e consumo de energia de *software*. O objetivo destas ferramentas é prover estimativas de desempenho e/ou consumo de energia de forma rápida, através exploração em alto nível do espaço de soluções de projeto. Esses trabalhos [5,6,8,13,17,36,47,49] têm a intenção de prover uma estimativa global (por exemplo, a plataforma do sistema, formada por seus diversos processadores e meio de interconexão) considerando sistemas multiprocessados e problemas relacionados à comunicação.

Dessa forma, são apresentadas diferentes técnicas de estimativa para aplicações executando em uma dada arquitetura. As técnicas de simulação oferecem uma estimativa de desempenho de *software* e consumo de energia bastante precisa, com um alto custo e esforço de modelagem. Modelos analíticos normalmente, são utilizados para se estimar o desempenho do *software* em maior nível de abstração. Mesmo rápidos, o maior desafio com métodos analíticos está na criação de um modelo para arquiteturas complexas de processadores.

2.1 Estimativa baseada em simulação

A seguir serão apresentados trabalhos relacionados ao tema proposto. Assim, são comentados trabalhos que utilizam a técnica de simulação para a obtenção de estimativas.

O cálculo de estimativa de desempenho e consumo de energia baseado em simulação tem como base a utilização de modelos com diferentes níveis de detalhe da funcionalidade da aplicação e da arquitetura. Quanto maior o detalhe, mais custosa torna-se a simulação, uma vez que tenta-se aproximar a complexidade da arquitetura em questão em um nível bastante baixo.

Simplescalar [6] é uma ferramenta flexível para análise de desempenho de processadores. Esta ferramenta pode ser utilizada como um simulador em nível de instrução (ISS) ou simulador em nível de ciclo. A ferramenta possibilita a otimização da arquitetura provendo meios para configurar uma dada arquitetura, como as unidades funcionais do *pipeline*, registradores e *cache*. Simplescalar inclui mecanismos para visualização de desempenho, recursos para análise estatística e uma infraestrutura de depuração.

Algumas ferramentas utilizam linguagens de descrição de arquitetura (ADLs), por exemplo LISA [27], Expression [38] e MIMOLA [30] para descrever a arquitetura do processador. Ferramentas que suportam essas linguagens produzem o simulador, compilador e as vezes o *hardware* sintetizável com base na descrição da arquitetura. ADLs permitem uma exploração rápida da arquitetura devido a geração automática da cadeia de ferramentas necessária para um novo processador. Algumas ferramentas comerciais como Lisatek [16] e MaxCore [5] utilizam a linguagem LISA.

Atualmente, simuladores baseados na linguagem SystemC têm sido desenvolvidos, facilitando a integração de modelos de simulação em nível de sistema. A biblioteca Microlib [37] provê um modelo do processador PowerPC 750 descrito em SystemC. ArchC [4] é uma ADL que gera modelos de simulação baseados em bibliotecas SystemC.

Tensilica [49] é um ambiente que provê o desenvolvimento de processadores baseados em uma dada aplicação alvo através de um conjunto de instruções configurável. Inserido neste ambiente, o compilador XPRES [49] explora automaticamente o espaço de aplicação para um dado código descrito em linguagem C. Após a definição de um modelo de referência da arquitetura, o ambiente gera o simulador, o compilador e o processador sintetizável.

Protótipos virtuais são modelos de simulação que permitem a validação integrada de componentes de *hardware* e *software*. Esses componentes são armazenados em uma biblioteca de componentes. Os modelos de simulação integram um simulador do conjunto de instruções com modelos de simulação do *hardware* como memórias, barramentos e periféricos. Ambientes para modelagem e simulação de protótipos virtuais baseados em SystemC, como o MaxSim [5], Coware ConvergenSC [16] e Synopsys System Studio [47] provêm um amplo conjunto de componentes que podem ser estendidos através de módulos do usuário, descritos em SystemC. Algumas ferramentas suportam a síntese RTL a partir dos componentes de *hardware* e *software* da biblioteca de componentes, provendo um caminho automático para a implementação em silício.

Outros simuladores, como o SIMICS, utilizam modelos funcionais para o processador, barramentos e componentes de *hardware*. Modelos funcionais provêm uma velocidade razoável para executar cargas de trabalho reais, ou seja, é possível a simulação de aplicações com milhões de instruções em tempo viável. Alguns trabalhos propõem a integração entre simuladores

funcionais do sistema e simuladores do processador com precisão de ciclo, como o Simplescalar [36]. Em [13], são apresentados também estimadores de potência, provendo um cálculo integrado de desempenho e consumo de energia.

Uma exploração de projeto com estimativas de consumo de energia e ferramentas de análise para SoCs baseados na arquitetura ARM é proposta em [17]. A ferramenta integra os modelos comportamentais e de energia de diversas unidades de processamento customizadas, como uma extensão ao simulador em nível de instrução para a família de processadores ARM *low-power*. Apesar de diversos estudos mostrarem que técnicas envolvendo tecnologia, *layout* e portas lógicas oferecem reduções pela metade no consumo de energia, a otimização da arquitetura pode frequentemente resultar em redução de energia em ordens de magnitude, conforme [29].

Em [21] são apresentados dois métodos para a integração de simuladores do conjunto de instruções e a linguagem SystemC. O primeiro método faz uso de uma chamada de sistema (ou um *breakpoint*) em *software* para suspender a execução e sincronizar com o *kernel* de simulação SystemC, de forma a sincronizar a simulação *hardware/software*. O segundo utiliza *drivers* do sistema operacional adaptados para que ocorra a sincronização da execução e comunicação com o *kernel* SystemC quando uma operação de entrada/saída ocorre. Em ambos os casos, mudanças no *kernel* SystemC tornam-se necessárias para suportar a sincronização.

MPARM [8] é um ambiente para a exploração do espaço de soluções para o desenvolvimento MPSoC através da utilização da linguagem SystemC. Essa é uma plataforma completa para a simulação MPSoC composta por modelos de processadores (ARM), barramentos (AMBA), modelos de memória, suporte para SMP (*hardware semaphores*), e uma cadeia de desenvolvimento de *software* incluindo um compilador de linguagem C e um sistema operacional (UCLinux). Os componentes de *hardware* são todos descritos em SystemC. O simulador em nível de ciclo do processador ARM foi desenvolvido em C++, encapsulado em um *wrapper* SystemC e integrado na plataforma. A Figura 2 apresenta um exemplo de arquitetura composta por dois processadores ARM, o barramento AMBA, dois módulos de memória e semáforos em *hardware*.

A plataforma MPARM provê suporte para análise de desempenho. As estatísticas de desempenho incluem taxas de *miss/hit* da memória *cache*, contenção do barramento e tempo de espera médio para tranferências. Essas estatísticas são utilizadas para a exploração da política de arbitragem no barramento AMBA.

Uma proposta de integração entre modelos de simulação gerados através da linguagem LISA e simulação em nível de sistema, descrita em SystemC, é realizada por [50]. Objetivo é explorar o processador e comunicação conjuntamente, utilizando uma abordagem a nível de sistema. O ambiente de verificação integrado provê uma forma de analisar o desempenho de *software*

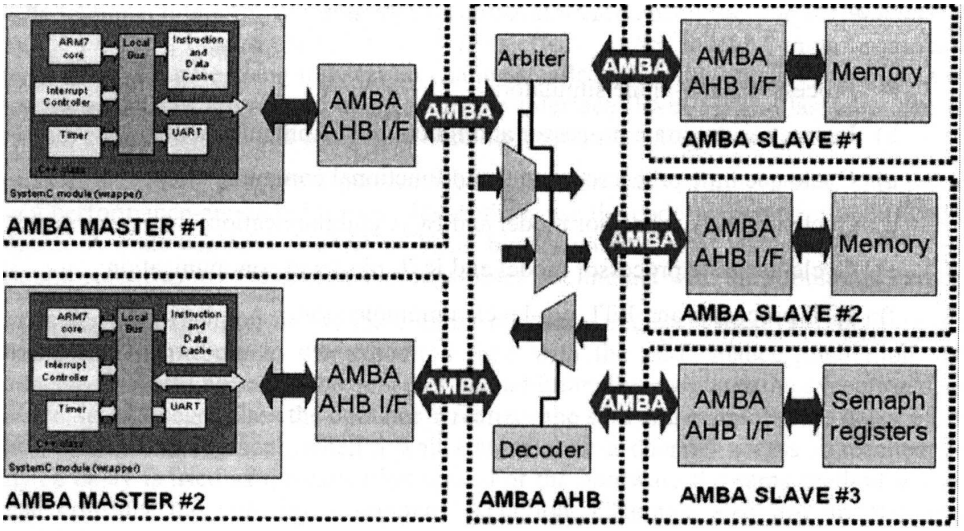

Figura 2 – Exemplo de arquitetura MPARM [8].

baseado nos modelos descritos.

O simulador do processador é modelado em nível de instrução ou de ciclo. O modelo em nível de instrução executa o conjunto de instruções mas ignora os efeitos do *pipeline*. Por outro lado, o modelo em nível de ciclo simula completamente os estágios do *pipeline* e trancamentos devidos a acessos à memória. O processador gerado em LISA é encapsulado em um *wrapper* SystemC e conectado com o resto do sistema por interfaces TLM ou RTL.

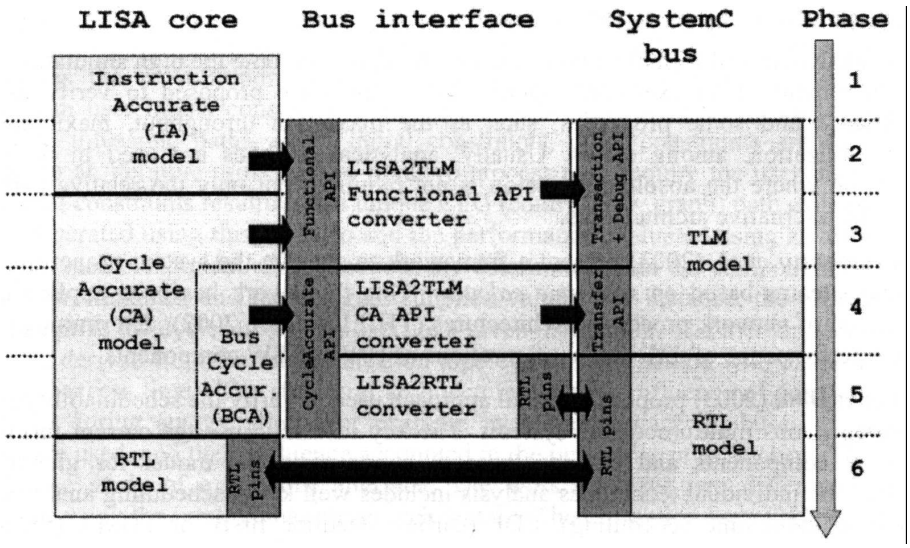

Figura 3 – Níveis de integração entre a simulação LISA e SystemC [50].

Os autores propõem diferentes combinações entre níveis de abstração de processador e mo-

delo de barramento como mostrado na Figura 3. O simulador isolado (fase 1) desconsidera a comunicação e conflitos em recursos compartilhados, levando em conta apenas a execução isolada do *software*. Na fase 2, é considerada a simulação em nível de sistema, e o *software* utiliza o simulador a nível de instrução com interfaces funcionais com o resto do sistema. Essas interfaces modelam as operações sem levar em conta problemas de temporização. Na próxima fase (fase 3), o simulador em nível de instruções é substído por um em nível de ciclo. Na fase 4, a comunicação é modelada com precisão de ciclo utilizando canais de transação BCA. No próximo nível as interfaces são redefinidas pela interface com pinagem completa. O próximo e último passo utiliza modelos sintetizáveis com simuladores RTL.

2.2 Estimativa baseada em métodos analíticos

Métodos analíticos de estimativa de desempenho de *software* são propostos para se prover uma estimativa rápida e com um baixo esforço de modelagem e execução. Esses métodos são úteis para uma exploração do espaço de soluções em níveis de abstração superiores (por exemplo, nível de transações). Usualmente, uma análise da aplicação é feita para a extração do número de instruções de diferentes tipos [22, 28]. Após, é realizado um mapeamento dessas instruções para um modelo de desempenho que calcula o tempo de execução.

Os métodos analíticos e formais são propostos para se evitar longos tempos de simulação e o requerimento de um modelo executável. Essas ferramentas são propostas para a verificação de desempenho do sistema e de algumas propriedades, como *throughput* máximo, atrasos e utilização de *buffers*. Normalmente, métodos analíticos são utilizados no espaço de exploração do projeto para sistemas sem um grau de complexidade grande.

Em [22] a aplicação é compilada em um conjunto virtual de instruções (um conjunto RISC simplificado com 25 instruções). A estimativa é feita através da avaliação do custo de execução de instruções virtuais na arquitetura alvo. Os autores analisam um conjunto de *benchmarks* com 35 aplicações automativas baseadas em controle, considerando o conjunto virtual de instruções, e utilizando um simulador em nível de ciclo para obtenção do número de ciclos consumidos por uma aplicação. Após, uma análise estatística baseada em regreção linear é aplicada nesses dados para se encontrar uma constante K (dependente da aplicação) e os índices P_i da equação a seguir, onde P_i e N_i são os pesos e o número de execuções de cada instrução, respectivamente.

$$ciclos = K + \sum P_i N_i \text{ (Equação 2.1)}$$

Como essa abordagem utiliza um método linear de classificação, o mesmo é apenas ade-

quado quando o conjunto de treinamento do modelo é similar às aplicações para as quais a estimativa é requerida, como demonstrado pelos autores. Não há uma discussão sobre detalhes da arquitetura alvo para as quais as estimativas são obtidas.

Os autores em [10] utilizam um método não linear para a estimativa do tempo de execução. Para um dado conjunto de *benchmarks*, um classificador extrai um *vetor de assinatura funcional* para um processador virtual (com um conjunto de 42 instruções), contendo os tipos de instruções que aparecem no código e o número de execuções de cada uma. Essa assinatura funcional é, segundo os autores, independente da arquitetura alvo, dessa forma pode ser reusada para a estimativa em diferentes processadores. Os autores entretanto não discutem o impacto do uso dessa técnica para estimar o desempenho para um processador de uma arquitetura diferente da qual o processador virtual serve de base para o classificador. Seu método de estimativa é também baseado em uma *assinatura arquitetural* do processador alvo. Os autores propõem dois parâmetros que definem essa assinatura: o número de ciclos de espera para acesso à memória e a taxa entre a freqüência do processador e do barramento. São apresentados resultados de estimativa para um processador MIPS R3000. Nesse estudo, é utilizada uma abordagem de treinamento e após testes. Na fase de testes, é aplicada uma técnica de modelagem chamada *lazy learning* para a escolha de uma função de estimativa que é baseada em um critério de vizinhança entre a aplicação e o conjunto de treinamento. Essa função, que pode ser localmente linear, usa apenas pontos do conjunto de treinamento que são próximos à aplicação. As entradas para essa fase são as assinaturas funcionais e arquiteturais e o número de ciclos para a execução da aplicação do conjunto de *benchmarks*, obtidas através da simulação em nível de ciclo no processador alvo. Os autores propõem um método de treinamento baseado na divisão dos *benchmarks* em dois conjuntos disjuntos de treinamento e teste. No estudo é reportado um erro médio de 8.8% nas estimativas, para um conjunto de 6 *benchmarks*, cada um executado com 15 conjuntos diferentes de entrada de dados. Não são reportados, entretanto, o tamanho dos conjuntos de treinamento e de teste.

Os autores de [7] comparam o uso da técnica de anotação utilizando um conjunto de instruções utilizando dois métodos. O primeiro método traduz a aplicação descrita em linguagem C para um conjunto virtual de instruções. Cada instrução do conjunto virtual possui um custo associado à arquitetura alvo, que é obtido por simulação ou por um método estatístico, como apresentado em [22]. O segundo método utiliza simulação compilada, onde o código *assembly* é traduzido para um código de simulação que será executado na máquina hospedeira usando anotações de atraso. Os autores reportam resultados mais precisos na abordagem baseada em código objeto porque dessa forma podem capturar as otimizações do compilador. Os autores reportam os resultados utilizando um processador MIPS R3000 para uma aplicação de pro-

dutor/consumidor. O método de instruções virtuais resulta em um erro entre 0.29% e 80% comparado à simulação em nível de ciclo, enquanto que o método de código objeto reporta um erro entre 0.29% e 10.5%.

Em [42] uma abordagem formal usada para verificar as propriedades de escalonamento de sistemas multiprocessados heterogêneos é apresentada. A idéia principal é utilizar a caracterização de componentes indiviuais e estendê-los em um modelo composicional (ou seja, um sistema) para análise global de MPSoCs. As técnicas individuais de análise incluem algoritmos bem conhecidos de escalonamento, como o RMS, o EDF e TDMA. Essas técnicas de análise modelam a tarefa com ativação de comunicação como fluxos de eventos. O problema principal no modelo composicional é que, normalmente, os modelos de saída não são aceitos como modelos de entrada. Para resolver esse problema, um conjunto de interfaces de modelos e funções de adaptação de eventos são utilizadas para adaptar automaticamente o fluxo de eventos de saída no modelo de eventos de entrada já estabelecido.

Uma técnica de macromodelagem baseada em caracterização de trechos de código é apresentada em [39]. A técnica possibilita a extração de modelos funcionais em alto nível de componentes reutilizáveis de *software*, utilizando modelos mais detalhados (em nível de ciclo ou instrução). O esforço que é dirigido para a construção de macromodelos para um módulo de *software* é amortizado pelo grande número de aplicações que o utilizam.

Os autores em [43] apresentam um método baseado em modelos abstratos de desempenho e cenários de aplicação. Um cenário de aplicação é um caminho definido no grafo de fluxo de controle que exprime as características mais importantes da aplicação em questão. O cenário é extraído estaticamente através de restrições de projeto informadas pelo usuário. As restrições são propagadas para guiar os nodos em um caminho praticável de execução. A partir das restrições iniciais, outras restrições são derivadas e propagadas em um processo iterativo. O processo iterativo pode requerer interação do usuário para definição manual de restrições, resultando em um CFG de caminho único chamado de cenário. Um *trace* é gerado com esse cenário e o desempenho é avaliado com o uso de funções de custo abstratas para cada componente. Funções de custo são determinadas pelas propriedades de cada componente, características da arquitetura e valores informados pelo projetista. Por exemplo, a função de custo do processador é composta por ciclos necessários para a execução de uma dada instrução. Acessos à memória utilizam uma função de custo derivada da topologia de interconexão em combinação com valores previamente determinados. A estrutura da arquitetura leva em conta a influência de componentes que são utilizados durante uma operação. Por exemplo, em um acesso à memória, barramentos e controladoras de memória são acessados, então sua influência é contabilizada na estimativa de desempenho. Os autores apresentam um estudo de caso para uma interface de

rede. O trabalho analisa duas organizações de memória diferentes utilizando como arquitetura alvo o processador Intel i960. Os erros de estimativa reportados estão em torno de 20%.

Em [33] é demonstrada uma proposta de método estático de análise utilizando uma técnica chamada de enumeração implícita de caminhos, determinando o número de execuções de cada bloco básico em um pior caso de execução. Os valores são calculados por equações lineares obtidas da análise estrutural e funcional da aplicação. Restrições estruturais são geradas a partir da análise do grafo de fluxo de controle (CFG). Restrições funcionais são informadas pelo usuário e descrevem a informação que não pode ser obtida pelo CFG como limites de laços (limite de vezes que um determinado laço executa) e caminhos falsos (destinos de saltos não satisfeitos por uma condição). Um método de programação linear pode maximizar essas equações e ter-se dessa forma o WCET (do inglês, *Worst Case Execution Time*).

Em [51] é apresentado um método para reduzir as equações lineares apresentadas em [33] e conseqüentemente, a complexidade na tentativa de extração de um único caminho possível. Um único caminho possível pode ser extraído quando a execução do programa é independente da entrada de dados. Mesmo não sendo esse o caso para todos os programas, algumas subpartes podem ser classificadas como um caminho único, como núcleos de algoritmos de processamento de sinais. O trabalho não utiliza o pior caso mas intervalos que são calculados considerando-se que o custo de um bloco básico de execução é variável. Intervalos retornam resultados mais precisos, pois eles utilizam um custo de execução de bloco básico preciso.

A análise semântica pode dar outras informações que são também relevantes para estimativas de desempenho na presença de características complexas de arquitetura. Em [32], o cálculo de estimativa é realizado a partir do número de *misses* na *cache* que é obtido pela aplicação de equações lineares. Em [31], um método que modela o impacto da execução especulativa é descrito. A estimativa de desempenho pode ser calculada em função do número de *misses* do preditor de desvios, que pode ser estaticamente obtido, como apresentado em [14]. Essas predições podem aumentar a precisão do cálculo do tempo de execução de cada bloco básico, uma vez que esse cálculo utiliza apenas informação local. Nessa fase, simuladores em nível de ciclo podem ser alternativamente utilizados [33, 51], mas com um custo mais elevado. Uma alternativa é a utilização de modelos mais abstratos de processador que reduzem a complexidade e facilitam o processo de estimativas para diferentes processadores [19, 44].

3 Plataforma proposta

3.1 Características da plataforma N-MIPS

A plataforma é composta por microprocessadores MIPS, por um meio de interconexão e por módulos de memória. O número de processadores utilizados na plataforma está relacionado ao desempenho necessário para a execução da aplicação-alvo e ao consumo de energia esperado. Não existe nenhuma restrição ao número de processadores a serem utilizados, o que torna a plataforma escalável e flexível.

Esta plataforma tanto pode ser simulada como implementada em um dispositivo FPGA. No caso da implementação em FPGA existe uma limitação natural quanto ao número de processadores devido a área (tamanho) do dispositivo FPGA.

A escolha do microprocessador MIPS para compor a plataforma levou em consideração uma série de vantagens, como implementações *open source* disponíveis, facilidade de integração de múltiplos núcleos através de diferentes meios de interconexão, existência de compiladores e alto desempenho para boa parte das aplicações embarcadas. Além disso, a arquitetura do microprocessador MIPS é genérica o suficiente para representar grande parte das características de processadores embarcados [6].

Atualmente interconexões entre núcleos em um *SoC*, são realizadas através de canais ponto-a-ponto ou de canais multi-ponto [52]. A utilização de canais ponto-a-ponto conecta núcleos por canais dedicados, sendo que cada canal é constituído por um conjunto de fios interligando dois núcleos. Os canais multi-ponto utilizam uma estrutura de interconexão com a forma de um barramento compartilhado e multiplexado no tempo, no qual os núcleos do sistema são conectados [12].

Uma nova tendência de interconexão denominada redes intra-chip (NoC) está surgindo. Seu conceito de interconexão é oriundo de redes de computadores e de sistemas distribuídos, onde os núcleos do sistema são interligados por meio de uma rede de roteadores através de canais ponto-a-ponto, e os dados são transferidos através de roteadores e canais até seus destinos [52].

As redes intra-chip estão surgindo como uma solução para alguns problemas de comunicação existentes em sistemas digitais modernos. Segundo Dally [18], a substituição de fios de

propósito específico, utilizados atualmente para comunicação por redes de interconexão, tem tornado o processo de roteamento de pacotes mais rápido e econômico.

Observou-se, entretanto, que para alguns poucos núcleos interconectados e onde não ocorra em demasia comunicação paralela, a interconexão através de canais multi-ponto torna-se mais econômica, com relação a área (em silício) e consumo de energia. Além disso, a maioria dos MPSoCs (em torno de 80%) industriais utilizam entre 2 e 4 microprocessadores, e, segundo a indústria eletro-eletrônica, este número não deve aumentar nos próximos 10 anos.

Dessa forma, foi utilizado um meio de comunicação por canal multi-ponto (barramento) para construção da arquitetura utilizada no contexto desse trabalho. Essa escolha é justificada devido ao fato desse meio ter sido prototipado em *hardware* com sucesso, além de ter pouca ocupação em área e *throughput* satisfatório.

3.1.1 Processador Plasma

A arquitetura adotada neste trabalho utiliza o processador Plasma. Este processador é um *soft-core*[1] descrito em VHDL, e implementa uma parte do conjunto de instruções MIPS [26]. O processador Plasma é um projeto simples, dessa forma apenas o primeiro co-processador (*System Control Processor*) descrito na arquitetura MIPS é implementado, e instruções de acesso desalinhado a memória estão ausentes, por questões de patente.

3.1.2 Toolchain

No processador Plasma não são implementadas instruções de acesso desalinhado, como citado anteriormente. Dessa forma, foi necessário a utilização de um compilador que não gerasse as instruções LWL, SWL, LWR e SWR do conjunto de instruções MIPS. O compilador utilizado foi o GNU GCC 4.0.2, modificado para não gerar tais instruções.

Além do compilador, foram criados *makefiles* para cada projeto (aplicação). Nos *makefiles* foram passados ao compilador parâmetros específicos, ordem de montagem, compilação, processo de *linking*, desmontagem e criação de *hex dumps* das aplicações. Dessa forma, tem-se como saída da compilação diferentes formatos do código objeto, para simulação e prototipação.

[1] Descrição RTL em alto nível de um módulo de *hardware*, que pode ser visualizada, modificada ou adaptada para determinado fim.

3.2 Modelagem da plataforma

3.2.1 Modelo de energia

As medições elétricas são baseadas em cálculos de potência e energia descritos na literatura, e apresentados a seguir.

A dissipação de potência (Equação 3.1) em um sistema digital é calculada através da soma da potência estática de um dado circuito com a potência dinâmica [11, 23]:

$$P = P_{static} + P_{dynamic}$$

(Equação 3.1)

A potência estática, P_{static}, independe da atividade do circuito e é determinada pela tecnologia alvo (que para este trabalho está sendo considerada a tecnologia MOSFET), curto circuitos não desejados e corrente de fuga dos transistores. A potência dinâmica $P_{dynamic}$ (Equação 3.2) pode ser calculada através da soma da potência dissipada em curto circuitos (Equação 3.3) e a potência de chaveamento (Equação 3.4):

$$P_{dynamic} = P_{switching} + P_{short}$$

(Equação 3.2)

A potência dissipada em curto circuitos P_{short} é originada pela pequena corrente existente entre a fonte e o terra, e que aparece na saída de uma porta lógica MOSFET durante o chaveamento de um nível lógico para outro, no exato momento em que ambos transistores conduzem corrente.

$$P_{short} \approx V_{DD} \times I$$

(Equação 3.3)

O termo mais importante da equação é a potência dissipada pela carga e descarga de capacitâncias parasitas presentes em todas as portas lógicas do circuito. Essa dissipação de potência é referenciada como potência de chaveamento, e assim como a potência dissipada em curto circuitos, depende da atividade do circuito (número de transições entre os níveis lógicos para todas as portas).

Dessa forma, a potência de chaveamento em um circuito digital CMOS pode ser determinada usando-se a seguinte fórmula:

$$P_{switching} = \alpha \times C \times f \times V_{dd}^2$$
(Equação 3.4)

Onde α é definido como a atividade do chaveamento, C como a capacitância de cada nodo chaveado, f como a freqüência do circuito e V_{dd} é definido como a voltagem.

Sabe-se que a potência de um dado circuito é um fator instantâneo. O consumo de energia consiste do acúmulo da dissipação de potência em um determinado intervalo de tempo[2]. Desta forma a dissipação de potência do circuito como um todo é calculada através da média das dissipações de potências obtitas em diferentes instantes de tempo. Assim, basta multiplicar a potência dissipada pelo intervalo de tempo desejado, para se obter o consumo de energia. Assim, pode ser representado o consumo de energia de um circuito conforme apresentado na Equação 3.5:

$$E = P \times \Delta t$$
(Equação 3.5)

Onde E é a energia consumida, P é a potência dissipada e Δt é um intervalo de tempo.

A metodologia aplicada neste trabalho analisa a atividade de chaveamento de portas lógicas de uma descrição VHDL, sendo este um item que projetistas podem modificar em uma arquitetura para a redução ou a otimização do consumo de energia. Deve ser observado que a simulação de uma descrição VHDL é muito mais rápida que, por exemplo, a simulação de uma descrição SPICE do mesmo circuito. O nível de abstração da linguagem VHDL ajuda a acelerar a estimativa de dissipação de potência de um circuito, e o uso de uma plataforma facilita o processo de simulação.

O consumo de energia de uma arquitetura MPSoC é calculado pelo somatório do consumo de energia de cada processador, acrescido do consumo de energia da estrutura que forma o meio de interconexão. Esse consumo é representado pela Equação 3.6:

$$E_{mpsoc} = \sum_{i=0}^{n} E_{p_i} + E_{ic}$$
(Equação 3.6)

Onde E_{p_i} é definido como o consumo de energia do processador i e E_{ic} o consumo de energia do meio de interconexão.

A ferramenta proposta utiliza um modelo de energia em alto nível e baseia-se em um método de classificação de instruções apresentado no próximo capítulo. Assim, tem-se um cálculo

[2]Integra-se a dissipação de potência em um intervalo de tempo, obtendo-se o consumo de energia.

do consumo de energia por ciclo de processamento diferente para cada instrução de diferente classe.

Dessa forma, o ISS utiliza as medições elétricas anteriormente caracterizadas em uma biblioteca VHDL e simuladas para estimar o consumo de energia de uma determinada aplicação. A biblioteca VHDL de estimativas, assim como a metodologia empregada para estimar o consumo de energia de um circuito são apresentadas nas próximas seções.

Além disso, na ferramenta de estimativas foi utilizado um modelo em alto nível para estimativas de energia do meio de interconexão, baseado na atividade de comunicação do meio e detalhado no próximo capítulo.

3.2.2 Modelo de desempenho

O tempo de execução de uma aplicação no processador Plasma é determinado pelo número de instruções executadas, ciclos de pausa do *pipeline* devido a leituras e escritas (operações de *load* e *store*), ciclos de pausa do controle do processador devido a operações multi-ciclo[3], tempo de acesso a memória e frequência do processador. Dessa forma, o tempo de execução (ET) em milissegundos é dado pela Equação 3.7:

$$ET_{p_n} = \frac{(((Instr+C_{ls}+C_{pause})*MemLatency)*RefClock)}{1000}$$

(Equação 3.7)

Onde $Instr$ representa o número de instruções executadas (considerando-se uma instrução por ciclo), C_{ls} representa o número de ciclos perdidos por pausa do pipeline em leituras e escritas (operações de *load* e *store*), C_{pause} representa o número de ciclos onde o controle do processador permanece em pausa, $MemLatency$ a latência (em ciclos) de acesso a memória e $RefClock$ representa a frequência de operação do processador. Para a implementação do processador em questão, parâmetros como tempo de acesso a memória e frequência de operação são fatores fixos. Neste trabalho, o tempo de acesso a memória foi definido como 1 ciclo, e a frequência de operação 25MHz. Dessa forma, o tempo de execução pode ser calculado através da Equação 3.8:

$$ET_{p_n} = \frac{((Instr+C_{ls}+C_{pause})*25000000)}{1000} = (Instr + C_{ls} + C_{pause}) * 25000$$

(Equação 3.8)

[3]Como exemplo, podem ser citadas instruções de multiplicação e divisão.

A estrutura que forma o meio de interconexão entre os processadores também é responsável pela variabilidade do tempo de execução de uma aplicação quando ocorrer comunicação entre os mesmos. Além disso, como o meio de interconexão utilizado no contexto desse trabalho é uma estrutura em forma de barramento, é sabido que a comunicação entre diversos núcleos ocorre multiplexada no tempo. Assim, tem-se que:

$$MF = \frac{1}{CPU_{Send}}$$

(Equação 3.9)

Onde MF é o fator de multiplexação no tempo, ou utilização máxima da banda por processador e CPU_{Send} é o número de processadores que estão requisitando acesso concomitantemente, ao barramento para o envio de dados.

O tempo de comunicação entre núcleos é definido pela latência entre o envio de um pacote de dados de um processador e recebimento dos dados em outro. Esse tempo é influenciado pela latência dos *drivers* de comunicação, latência do controle entre o meio de interconexão e os processadores (tratamento de sinais e geração de *bursts*), tempo de arbitragem do barramento e fator de multiplexação (MF). Simulações possibilitaram a observação de que o tempo de execução dos *drivers*, a latência do controle de acesso ao meio e do protocolo de comunicação utilizado são bastante superiores ao tempo de transmissão em si, mantendo o barramento ocioso na maioria do tempo. Assim, o fator de multiplexação pode ser desconsiderado, pois não existe concorrência entre os núcleos no uso do barramento, apenas pequenos atrasos podem ocorrer no tempo de arbitragem. Para se obter o tempo de comunicação, foi simulado o envio de milhares de pacotes de dados (em torno de 8000) e calculado o tempo médio para o envio de um pacote. Dessa forma, define-se a latência para o envio de um pacote como apresentado na Equação 3.10:

$$WCET_{packet} = WCET_{driver} + WCET_{control} + WCET_{bus}$$

(Equação 3.10)

Cada pacote tem latência em torno de 50 ciclos para seu envio (apenas 8 ciclos são necessários para o envio efetivo de dados pelo barramento, após arbitragem que tem em torno de 4 ciclos, ou 1 ciclo por porta), então a latência total do meio de interconexão é influenciada pelo número de pacotes enviados. A latência total da comunicação pode ser definida através da Equação 3.11:

$$WCET_{comm} = WCET_{packet} * n\_packets$$

(Equação 3.11)

Tendo-se o tempo de execução de cada processador (Equação 3.8) e também a latência de comunicação (Equação 3.11), pode-se calcular o tempo de execução da plataforma MPSoC, que consiste do maior tempo de execução entre o conjunto de processadores e da latência de comunicação entre estes processadores. A Equação 3.12 representa o tempo de execução da plataforma MPSoC:

$$ET_{mpsoc} = Max(ET_{p_0}, ET_{p_1}, ET_{p_2}, ET_{p_3}) + WCET_{comm}$$

(Equação 3.12)

O modelo de desempenho apresentado é utilizado pela ferramenta de estimativas na execução de todas as instruções de uma aplicação em um ou mais ISSs (fator esse que depende do número de processadores da arquitetura), considerando atrasos decorridos de diversos fatores, conforme apresentado anteriormente. Assim, cada instrução é buscada em memória no simulador do processador e executada, em um processo iterativo. Atrasos (em ciclos) que representam a implementação do *hardware* são incluídos na contagem e modelam dessa forma o funcionamento da plataforma.

3.2.3 Metodologia aplicada

A principal idéia por trás da metodologia aplicada nesse trabalho, é utilizar eventos VHDL para detectar a atividade de dispositivos lógicos. Durante a simulação, um dispositivo lógico pode ser estimulado pela mudança do nível lógico de suas portas de entrada. O dispositivo reage com a execução de um processo RTL comportamental, e escalona, então, uma transação nos sinais conectados em suas portas de saída. Esse fato é conhecido como escalonamento de uma transação naquele sinal. Se o novo valor for diferente do valor anterior, um evento ocorre, e outros dispositivos lógicos com portas de entrada conectadas àquele sinal podem ser ativados. Com o uso de processos em VHDL, é possivel a coleta de todas as transições em um circuito e, consequentemente, é possível estimar o consumo de energia.

O primeiro passo do método é sintetizar a descrição VHDL comportamental da arquitetura utilizando uma biblioteca alvo, no caso, CMOS TSMC $0.35\mu m$ na ferramenta Leonardo Spectrum. Após a síntese, é gerado um circuio em nível de portas lógicas, ou seja, um *netlist*.

O segundo passo é a conversão do circuito usando a ferramenta CAFES [34]. Esse processo converte o *netlist* gerado pela ferramenta Leonardo Spectrum para um formato compatível com a biblioteca de estimativas de energia [34].

O terceiro passo consiste na simulação do circuito utilizando a descrição VHDL gerada e convertida com a biblioteca de estimativas, em um simulador VHDL, como por exemplo o ActiveHDL [3] ou ModelSim [24].

A Figura 4 apresenta o fluxo de projeto utilizado para estimativas de energia de circuitos digitais CMOS. Para a biblioteca CMOS utilizada, foram implementadas 10 portas lógicas. Com essas portas lógicas básicas, é possível descrever qualquer circuito. Um *script* utilizado com a ferramenta Leonardo Spectrum restringe as funções geradas àquelas implementadas pela biblioteca, para que o circuito possa posteriormente ser simulado com a mesma.

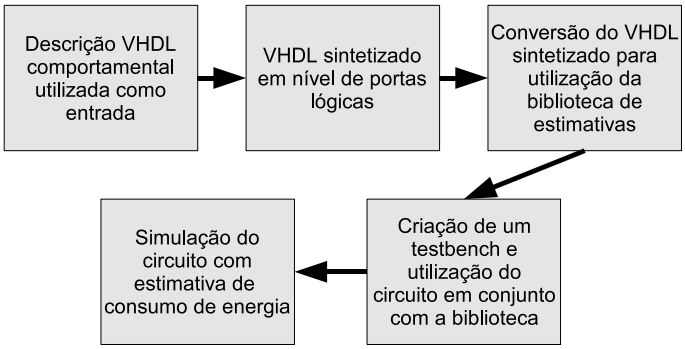

Figura 4 – Fluxo da metodologia.

3.2.4 Biblioteca VHDL para estimativas de consumo de energia

O componente chave para estimativas de consumo de energia é a biblioteca de estimativas, implementada por [34]. A idéia principal dessa biblioteca é usar eventos em VHDL para detectar a atividade de circuitos, e calcular (ou acumular) a potência dissipada pelo circuito em simulação. Durante a simulação, um circuito pode ser estimulado pela mudança do estado lógico de suas entradas, e essas mudanças podem ser avaliadas com relação à estimativa de consumo de energia.

De fato, o estimador de consumo de energia não reporta o consumo da real implementação, mas um valor aproximado. Essa estimativa é consistente de acordo com o consumo de energia por dispositivos lógicos em uma implementação em *hardware*. A metodologia aplicada também leva em consideração a potência dissipada por componentes parasitas, introduzidos por interconexões (metais e vias), bem como o *fanout* de cada porta lógica.

Para a implementação da biblioteca, simulações de nível elétrico foram feitas e medidas

com uso da ferramenta SPICE. Diversos *scripts* foram organizados para automatizar o processo de captura dos valores reportados pela ferramenta SPICE e converter esses valores para VHDL, para diversas funções lógicas. As funções escolhidas podem descrever qualquer tipo de circuito, e dessa forma a ferramenta Leonardo Spectrum teve que ser instruida para utilizar apenas esssas funções para a geração do circuito em nível de portas lógicas.

As funções escolhidas foram inicialmente descritas e sintetizadas na ferramenta SPICE, usando a biblioteca de tecnologia CMOS TSMC $0.35\mu m$. A Figura 5 apresenta um *tri-state* descrito em SPICE. Essa porta lógica é composta pelos seguintes subcircuitos: Inversor e TransmissionGate, e é modelada com uma capacitância de saída de 50fF e uma resistência de 50 ohms, representando um *fanout* típico de 3 portas lógicas. Esse *tri-state* foi incluído na biblioteca de estimativas, descrita em VHDL.

```
.subckt Inversor out in Vcc 0
MP1  out in Vcc Vcc MODP L=0.35U W=5.0U AD=10.0P AS=10.0P PD=9.0U PS=9.0U
MN2  out in   0   0 MODN L=0.35U W=2.0U AD= 4.0P AS= 4.0P PD=6.0U PS=6.0U
.ends Inversor

.subckt TransmissionGate out in control notControl Vcc 0
M1 in  control    out   0 MODN L=0.35U W=2.0U AD= 4.0P AS= 4.0P PD=6.0U
PS=6.0U
M2 in  notControl out Vcc MODP L=0.35U W=5.0U AD=10.0P AS=10.0P PD=9.0U
PS=9.0U
.ends TransmissionGate

.subckt TriState out in control Vcc   0
X1 notControl control                    Vcc 0 Inversor
X2 out_1           in control notControl Vcc 0 TransmissionGate
R0 out          out_1                    50
C0 out            0                      50fF
.ends TriState
```

Figura 5 – *Tri-state* descrito em SPICE.

Medidas foram extraídas para *fanouts* variando entre 1 e 10, e os valores gerados foram contabilizados na biblioteca de estimativa. As portas lógicas foram então descritas em VHDL de acordo com o comportamento de suas entradas. Nessa descrição, a dissipação de potência foi anotada para cada transição, juntamente com a dissipação de potência estática durante o período em que as entradas não tiveram mudanças no nível lógico.

Para exemplificar a implementação, valores de consumo dinâmico de energia de um inversor são ilustrados na Tabela 1. Os parâmetros elétricos foram extraídos para a biblioteca de tecnologia CMOS TSMC $0.35\mu m$, com um *fanout* típico de 3 portas lógicas. As duas primeiras colunas ilustram as transições de entrada e saída de um inversor, respectivamente. A coluna "Potência (W)"apresenta a potência média dissipada em um intervalo de 10ns (borda de subida e descida em um *clock* de 50MHz). A última coluna mostra a energia consumida pela porta lógica na transição, e esses valores são descritos na biblioteca de estimativas em VHDL. Os valores da última coluna são calculados pela integral da potência consumida no período medido (10ns).

A Tabela 2 apresenta a potência estática dissipada pelo inversor, quando as entradas não são

Entrada	Saída	Potência (W)	Energia (J)
0->1	1->0	4.5964E-05	4.5964E-13
1->0	0->1	4.9066E-05	4.9066E-13

Tabela 1 – Cálculo do consumo de energia dinâmica para um inversor.

alteradas. Os valores nessa tabela são multiplicados pelo tempo de operação do circuito e adicionados ao consumo de energia pelo chaveamento, resultando dessa forma em uma estimativa completa de consumo de energia do inversor.

Entrada	Saída	Potência (W)
0->0	1->1	3.6825E-10
1->1	0->0	7.1430E-14

Tabela 2 – Potência estática dissipada por um inversor.

Para utilizar essa informação na estimativa de consumo de energia de um circuito, valores obtidos, semelhantes aos das tabelas anteriores, foram organizados na descrição VHDL do inversor. A descrição tem informação sobre o comportamento da porta lógica e a lógica que permite ao simulador contabilizar o consumo de energia em todas as transições do circuito. Na Figura 6 é apresentada a descrição VHDL do inversor. Essa descrição faz parte da biblioteca de portas lógicas para estimativas de energia.

```
entity Inversor is
        port(   saida: out STD_LOGIC;
                entrada: in STD_LOGIC);
end Inversor;

architecture Inversor of Inversor is
begin
        process(entrada)
                variable entradaAnterior: STD_LOGIC := '0';
                variable energiaDaPortaComTransicao: REAL := 0.0;
                variable energiaTransicao: REAL;
                variable potenciaSemTransicao: REAL:= 0.0;
                variable transition: STD_LOGIC_VECTOR(1 downto 0);
        begin
                transition := entradaAnterior & entrada;
                energiaTransicao := 0.0;
                case transition is
                        when "01" => energiaTransicao := 4.5964E-13;
                        when "10" => energiaTransicao := 4.9066E-13;
                        when others =>
                end case;
                energiaDaPortaComTransicao := energiaDaPortaComTransicao + energiaTransicao;
                energiaTotalComTransicao := energiaTotalComTransicao + energiaTransicao;
                potenciaTotalSemTransicao := potenciaTotalSemTransicao - potenciaSemTransicao;
                transition := (entrada & entrada);
                case transition is
                        when "00" =>  potenciaSemTransicao:= 3.6825E-10;
                        when "11" =>  potenciaSemTransicao:= 7.1430E-14;
                        when others =>
                end case;
                potenciaTotalSemTransicao := potenciaTotalSemTransicao + potenciaSemTransicao;
                saida <= not entrada;
                entradaAnterior := entrada;
        end process;
end Inversor;
```

Figura 6 – Inversor descrito em VHDL.

3.2.5 Estimativa de energia utilizando a simulação VHDL

Para que o ISS do processador Plasma pudesse ser alimentado com dados sobre estimativas de desempenho e consumo de energia, simulações do nível lógico do processador foram necessárias.

Uma plataforma para contagem de ciclos e estimativa de consumo de energia foi implementada para realizar as simulações. Essa plataforma é composta por diversos componentes, mostrados na Figura 7.

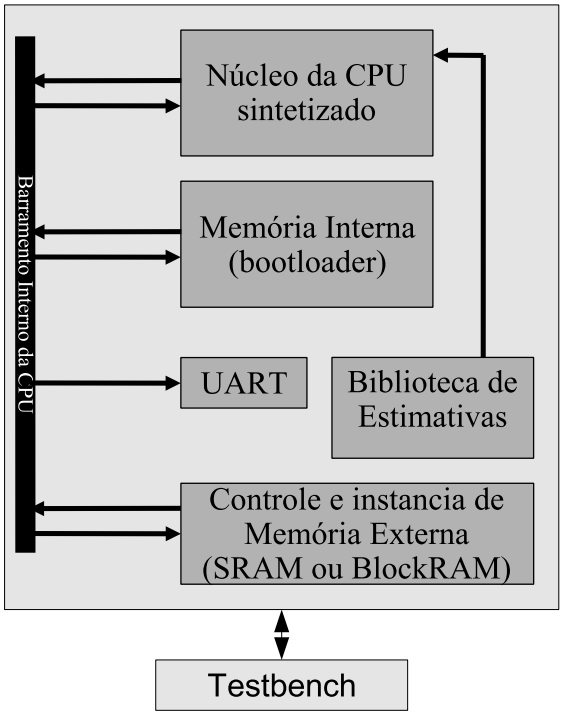

Figura 7 – Plataforma para estimativas de energia do processador Plasma.

O primeiro passo no processo foi a geração de uma nova descrição VHDL do núcleo do processador Plasma. Os níveis superiores da descrição original do processador foram alterados, para que o núcleo do processador fosse isolado da memória interna e da UART. Isso foi feito para que as medidas fossem fiéis ao que será emulado no ISS (em um primeiro momento, apenas o núcleo e banco de registradores).

O núcleo do processador foi sintetizado através da ferramenta Leonardo Spectrum utilizando a tecnologia TSMC $0.35\mu m$, e as funções lógicas foram restritas àquelas implementadas na

biblioteca de estimativa de consumo de energia. Esse passo gerou uma descrição VHDL em nível de portas lógicas (*netlist*).

Nessa etapa, o *netlist* obtido foi então processado pela ferramenta CAFES, que traduziu as referências das células lógicas padrão para referências as células da biblioteca de estimativa, considerando o *fanout* das mesmas. Com o processador sintetizado para estimativa de consumo de energia, as partes foram interligadas para a formação de uma plataforma completa.

A plataforma inicial (monoprocessada) é composta pelo núcleo do processador Plasma em forma de *netlist*, um modelo de memória[4], e uma UART. Um controle de memória externa foi também implementado. Esse segundo modelo é bastante genérico, parametrizável, sintetizável, e não possui limitação de tamanho, sendo esse apenas restrito aos recursos disponíveis no dispositivo FPGA. Nessa memória externa é carregada a aplicação a ser avaliada. A latência da memória externa é de 1 ciclo apenas. Todas as partes se comunicam através de um barramento interno. Um *testbench* gera os sinais de *clock* e *reset* para o processador e periféricos, e dessa forma o funcionamento do processador pode ser simulado, e uma estimativa de consumo de energia do núcleo pode ser realizada.

3.2.6 Estimativa de desempenho utilizando a simulação VHDL

Observou-se que os modelos VHDL tanto comportamental quando em nível de portas lógicas correspondem funcionalmente. Dessa forma, a estimativa de desempenho de aplicações é consistente, comparando-se os dois modelos.

Para estimativas de desempenho não são necessários mecanismos especiais como em estimativas de energia. Assim, uma avaliação de desempenho pode ser feita diretamente com o uso de uma descrição VHDL comportamental, sendo essa executada mais rapidamente em simuladores que uma descrição em nível de portas lógicas.

A mesma plataforma descrita na seção anterior foi utilizada para estimativas de desempenho, entretanto, um núcleo não sintetizado foi utilizado (em virtude da aceleração nas estimativas) e não foi necessária a utilização de bibliotecas adicionais.

[4] A descrição de memória implementada instancia BlockRAMs de dispositivos FPGA, com o código objeto para *boot* já carregado.

3.3 Meio de interconexão

Como citado anteriormente, no contexto desse trabalho está sendo utilizado um barramento[5] como meio de interconexão. O mesmo foi descrito em VHDL, simulado e prototipado em *hardware* com sucesso, e mostrou-se bastante econômico com relação a ocupação em área. Além disso, o barramento apresentou um *throughput* satisfatório.

As características do barramento podem ser observadas na Tabela 3:

Largura da palavra	8 *bits* (parametrizável)
Número de portas	4 (parametrizável)
Tamanho das filas	16 *bytes* (nas portas de saída, parametrizável)
Arbitragem	algoritmo rotativo
Latência	2 ciclos por palavra, após arbitragem
Freqüência de operação	25MHz (protótipo), até 222MHz (CMOS TSMC $0.35\mu m$)
Ocupação em área	8372 portas lógicas
Throughput	100Mb/s (8 *bits* de dados @ 25MHz)
Protocolo de E/S	*handshake*

Tabela 3 – Características do barramento.

A Figura 8 apresenta uma das portas de dados do barramento. Essa porta permite acesso transparente ao barramento para dispositivos (IPs) conectados ao mesmo. A porta de dados é formada por uma porta de controle de acesso ao meio (1), uma fila com tamanho parametrizável (2), um módulo que implementa a fila[6] na porta de controle (3) e um meio de acesso de baixo nível ao barramento (4). Esse meio de acesso é responsável por enviar dados para o barramento, receber dados do mesmo e colocar o barramento em alta impedância quando não estiver sendo utilizado (através da utilização de *tri-states*).

O número de portas, o tamanho das filas, assim como a largura do barramento são parametrizáveis. No protótipo da plataforma, foram instanciadas 4 portas de dados de 8 *bits* cada, com filas de saída de 16 *bytes*, e o barramento completo é representado na Figura 9, onde é apresentada a ligação entre as partes.

[5] A implementação foi criada pelo autor, sendo chamada *HotWire Bus*.
[6] Implementa *buffers* para a saída de dados.

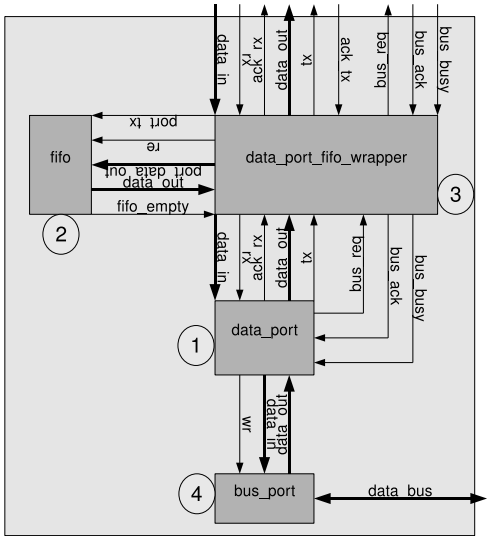

Figura 8 – Detalhe da porta de dados do barramento *HotWire*.

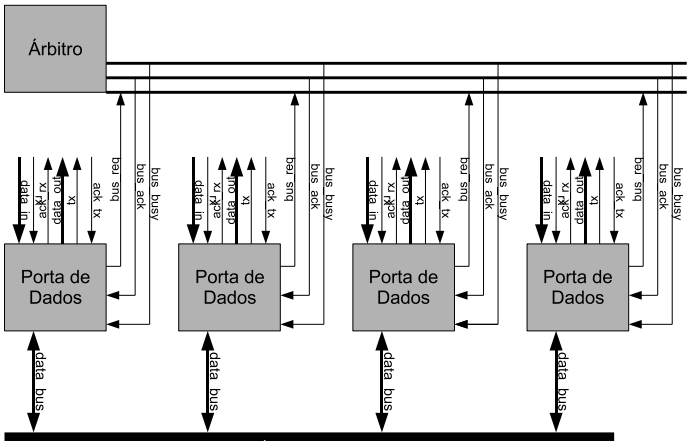

Figura 9 – Barramento utilizado na plataforma.

3.3.1 O protocolo *handshake*

Cada porta de acesso ao barramento implementa o protocolo *handshake*. Assim, os dados são colocados nas portas de entrada (1) ou saída (2) e um sinal é enviado (3) (4), sinalizando que o dado está pronto. Quando a porta de entrada do barramento ou o dispositivo controlado pela porta de saída receberem efetivamente o dado (envio ou recebimento), um sinal de *ackno-*

wledgement é enviado (5) (6), e um novo ciclo de transferência pode ser iniciado. Um exemplo de transferência de dados é demonstrado na Figura 10.

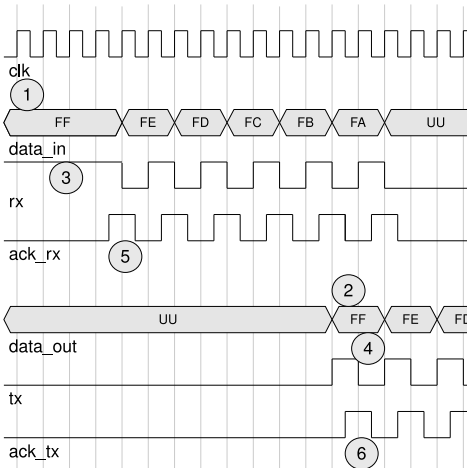

Figura 10 – Representação dos sinais para a entrada e saída de dados conforme o protocolo *handshake*.

É importante observar que a latência entre os sinais *rx* e *tx* e seus respectivos *acknowledgements* é de apenas meio ciclo. A baixa latência acontece devido a utilização de ambas as bordas de relógio na implementação, com o intuito de manter a latência por palavra transferida em apenas 2 ciclos (após arbitragem).

3.3.2 Controle entre os processadores e o barramento

Para que os processadores pudessem se comunicar utilizando o barramento implementado, foi necessária a definição de um controle e protocolo de comunicação. O processador Plasma possui um barramento interno com largura de 32 *bits* e portas de dados síncronas, já o barramento externo tem largura de 8 *bits* e portas de comunicação assíncronas.

Para conectar os processadores ao barramento foram utilizadas as portas de comunicação externas de entrada e saída, chamadas de GPIOA_IN e GPIO0_OUT, respectivamente. A porta GPIOA_IN possui largura de 32 *bits*, e os *bits* 31 e 30 estão conectados à linhas de interrupção, o que facilita a implementação de *drivers* de acesso ao meio de interconexão de alto desempenho. Além disso, o uso de portas já implementadas na descrição padrão do processador facilita a compatibilidade com sistemas operacionais e aplicações já existentes para essa arquitetura.

A Figura 11 representa a implementação do controle entre as portas de E/S de um processador e uma porta de acesso ao barramento.

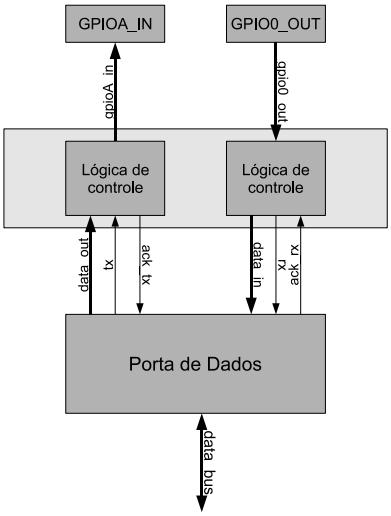

Figura 11 – Controle entre o processador e o barramento.

A função principal do controle entre os processadores da plataforma e o barramento é serializar palavras de 32 *bits* provenientes do processador (porta GPIO0_OUT) para palavras de 8 *bits* que são enviadas pelo barramento, e, também, paralelizar palavras de 8 *bits* provenientes do barramento para palavras de 32 *bits* que são enviadas para o processador (porta GPIOA_IN). Além disso, o controle dos sinais do protocolo de comunicação do barramento é recebido ou enviado e sinais de interrupção nos *bits* 31 e 30 da porta GPIOA_IN de cada processador são gerados.

O controle gera *bursts* de acesso de 4 palavras para leituras e escritas no barramento, e o desempenho do mesmo é limitado apenas pelo tempo de arbitragem. As perdas pelo tempo de arbitragem e latência dos *drivers* são amenizadas pelo tamanho das filas nas portas de saída do barramento. Além da geração de *bursts*, o controle realiza o endereçamento de cada processador na plataforma, apenas sinalizando o recebimento de dados na porta de entrada de um processador quando o destino confirmar, ou no envio de uma mensagem de *broadcast* (um processador pode enviar dados a múltiplos processadores).

A Figura 12 apresenta o formato do pacote de dados. Esse pacote é comum tanto para o envio quanto para o recebimento de dados no processador, entretando os *bits* 31 e 30 possuem significados distintos. No envio, os dados são colocados nos *bits* 0 a 23 da porta GPIO0_OUT, o endereço nos *bits* 24 a 29 e o sinal de recebimento é ativado quando o *bit* 31 se torna alto,

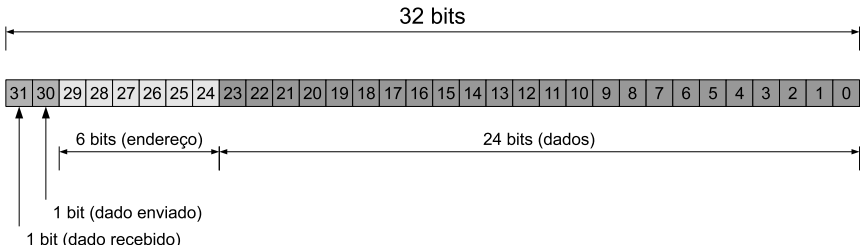

Figura 12 – Formato do pacote de dados.

gerando uma requisição de acesso ao barramento. Apenas quando o *bit* 30 da porta GPIOA_IN se tornar alto que o dado foi efetivamente enviado (sinal gerado pelo controle, interrompe o processador). No recebimento, os dados são colocados nos *bits* 0 a 23 da porta GPIO0_IN, o endereço é colocado nos *bits* 24 a 29 e o *bit* 31 se torna alto quando os dados estiverem prontos para recebimento (é gerada uma interrupção no processador). Esse *bit* recebe o valor "1"quando o endereço do dado corresponder ao processador em questão (destino real ou *broadcast*).

3.4 Arquitetura da plataforma multiprocessada

A plataforma completa de estimativas é formada não apenas por um único processador, mas sim por diversos processadores comunicando-se por um meio de interconexão. O objetivo da plataforma é contabilizar não apenas o consumo de energia em cada um dos núcleos, mas também o consumo de energia do barramento e controle do mesmo, principalmente em aplicações onde ocorra comunicação entre os núcleos. Além disso, a estimativa de desempenho será afetada, uma vez que existirão processadores trabalhando em paralelo.

Para a implementação foi definida uma plataforma constituída por quatro processadores comunicando-se por um barramento *HotWire*. Na Figura 13 é apresentado um exemplo onde quatro processadores executam código em paralelo e comunicam-se pelo meio de interconexão. Existem quatro portas de acesso ao meio de interconexão, e a cada uma está conectado um processador Plasma. As portas de entrada e saída de cada processador estão interligadas em uma controladora de acesso as portas do barramento.

Essa plataforma foi implementada em VHDL e um protótipo encontra-se em funcionamento em um dispositivo FPGA. Foram implementados *drivers* simples para comunicação entre os processadores, entretando não foi criado um mecanismo que faz uso de interrupções, o que possibilitaria maior flexibilidade.

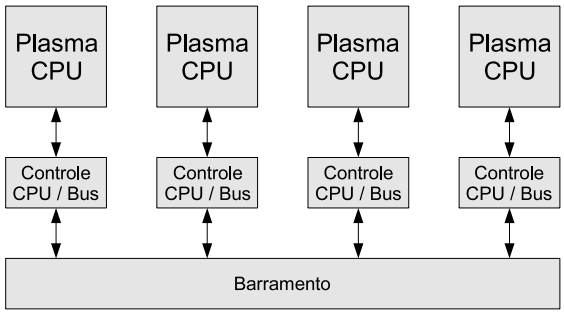

Figura 13 – Plataforma para estimativas de desempenho e consumo de energia.

Os *drivers* utilizam primitivas de comunicação *Send* e *Receive* bloqueantes e não bloqueantes. A primitiva *Send()* recebe como parâmetro uma palavra de 32 *bits*, contendo o endereço de destino do pacote e os seus dados. Apenas os 30 *bits* menos significativos são considerados para endereçamento e dados, sendo os 2 primeiros *bits* utilizados para sinalização da controladora de acesso ao meio de interconexão (dado pronto e dado enviado). A primitiva fica bloqueada até a controladora de acesso notificar que foi concedido acesso ao meio de interconexão e o dado tenha sido enviado ou ocorra um *timeout* do *driver* na versão não bloqueante. A primitiva *Receive()* recebe como retorno uma palavra de 32 *bits*, tendo apenas os 24 *bits* menos significativos os dados de recebimento. Apenas será recebido um dado quando o endereço de destino corresponder, e esse controle é feito pela controladora de acesso ao meio. A primitiva fica bloqueada até o recebimento de um dado, ou *timeout* na versão não bloqueante.

Essa plataforma já implementada e prototipada serviu de base para a implementação da ferramenta de estimativas. O barramento, o controle de acesso ao meio e os processadores foram simulados para uma estimativa de desempenho de *software* e o consumo de energia com sucesso. Os dados coletados nas simulações foram utilizados para a implementação a nível de ciclo na ferramenta, para que a estimativa da plataforma completa pudesse ser feita em mais alto nível, com ganhos consideráveis nos tempos de simulação.

3.4.1 A plataforma multiprocessada em simulação

A simulação da plataforma de estimativas completa permite uma avaliação detalhada da execução de código em cada um dos processadores, além da comunicação entre os mesmos.

Através da simulação, foi possível observar o funcionamento dos componentes de *hardware* e *software*.

Como abordado anteriormente, a simulação de um circuito em conjunto com a utilização de uma biblioteca de estimativas, permite que se obtenha o consumo de energia do mesmo. É importante ressaltar, que a simulação da arquitetura serviu de base para a implementação da ferramenta de estimativas, e dessa forma, os dados coletados durante diversas simulações foram essenciais para que fossem feitas avaliações da corretude de funcionamento da plataforma, estimativa de tempo de execução de aplicações e consumo de energia.

Diferentes componentes formam a arquitetura multiprocessada utilizada no contexto desse trabalho. Cada componente é responsável por uma parte do consumo de energia da arquitetura, e influencia no tempo de execução de uma dada aplicação devido a sua latência. Os componentes são:

- Processadores
- Barramento
- Controle entre os processadores e o barramento

Componente	**Área (portas lógicas)**
CPUs	82348 (20587 cada CPU)
Barramento	8372
Controle barramento / CPUs	2893
Total	**93613**

Tabela 4 – Área dos componentes em portas lógicas.

A Tabela 4 apresenta a área utilizada em portas lógicas, após o processo de síntese, para os componentes que compõem a plataforma descrita em VHDL. A arquitetura da plataforma isola os processadores da estrutura de interconexão. Tal característica possibilita a medição de cada componente separadamente após a síntese lógica. Os processadores Plasma representam 87.97% da lógica utilizada e os restantes 12.03% são utilizados pela estrutura de interconexão. A maior parte da lógica utilizada pelo meio de interconexão diz respeito ao tamanho das filas nas portas de saída do barramento.

A Figura 14 representa a conexão entre os componentes, de forma hierárquica, conforme as descrições VHDL utilizadas para a implementação da plataforma. Cada um dos quatro processadores, assim como o meio de interconexão, é instanciado em um nível superior, que é por sua vez estimulado com o auxílio de um *testbench*. O meio de interconexão é formado por um

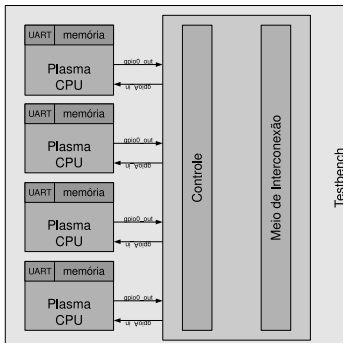

Figura 14 – Hierarquia dos componentes que compõem a plataforma de estimativas.

controle de acesso ao meio e um barramento *HotWire*. Tanto a memória interna (memória de *boot*) quanto a UART, representadas na Figura, não são avaliadas para consumo de energia.

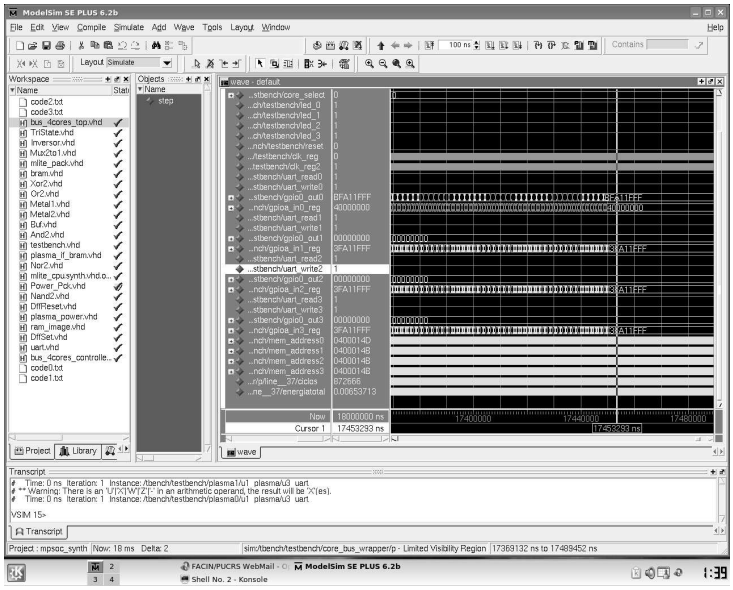

Figura 15 – Simulação de uma aplicação multiprocessada na plataforma de estimativas.

A Figura 15 apresenta a simulação de uma aplicação multiprocessada, utilizando a plataforma de estimativas multiprocessada descrita em VHDL. Apenas alguns elementos são ilustrados. As interfaces de comunicação dos processadores são representadas pelos números de 1 a 4, e no número 5 representa o consumo de energia estimado para a aplicação. O número 6 representa a lista de descrições VHDL (processadores, meio de interconexão, biblioteca de estimativas, testbench e inicialização das memórias). Nessa simulação estão presentes quatro

processadores Plasma sintetizados além do meio de interconexão também sintetizado, executando 18 ms do *hardware*. Essa simulação levou em torno de 2 dias, executando em uma máquina com processador *dual core* de 2.8 GHz e 2 GB de memória.

A simulação desses componentes, de acordo com a metodologia adotada nesse trabalho, torna-se demasiadamente custosa com relação ao tempo de processamento. Entretanto, a simulação isolada de cada um dos componentes de acordo com a metodologia é viável, e foi utilizada para as medições. Dessa forma, foram criados modelos de diferentes níveis de abstração, divididos em modelos funcionais e modelos detalhados. Os modelos funcionais utilizam descrições VHDL em nível comportamental ou estrutural (não sintetizado) e podem ser rapidamente simulados. Modelos detalhados utilizam descrições em VHDL em nível de portas lógicas (*netlists*) em conjunto com a biblioteca de estimativas e requerem alto tempo de simulação. A criação de quatro diferentes derivações dos componentes permitiu que todos os dados necessários para a implementação da ferramenta fossem obtidos. As derivações utilizadas são:

- VHDL não sintetizado dos processadores, meio de interconexão e controle
- VHDL não sintetizado dos processadores, VHDL sintetizado do meio de interconexão e controle
- VHDL sintetizado dos processadores, VHDL e não sintetizado do meio de interconexão e controle
- VHDL sintetizado dos processadores, meio de interconexão e controle

As derivações estão listadas em ordem crescente de complexidade computacional. Observou-se, por exemplo, que é praticamente inviável a simulação de quatro processadores Plasma em nível de portas lógicas. Dessa forma, nas simulações que foram utilizadas para a geração de dados para a ferramenta de estimativas, um único núcleo foi simulado. A simulação de todos os núcleos, em conjunto com o meio de interconexão e controle (em nível de portas lógicas) foi realizada, entretanto não pode ser extendida devido a sua complexidade.

3.4.2 Prototipação

A plataforma completa pode ser verificada em sua situação real de funcionamento através de um protótipo, implementado em um dispositivo FPGA. A placa utilizada contém diversos componentes, como LEDs, chaves, botões, conector padrão RS-232, saída VGA, *slot* para a

inserção de memória padrão DDR-SDRAM, cartão de memória Flash, entre outros além de um dispositivo FPGA XCV2VP30.

Um nível superior da plataforma, descrito em VHDL realiza as amarrações entre a plataforma formada por quatro processadores, controle, meio de interconexão, e o controle de *reset*. Esse nível também implementa um controle de seleção de *interface* serial de cada processador. O controle de seleção é implementado por meio de duas chaves (onde pode-se endereçar 2 *bits*, ou quatro processadores) e, também, conecta LEDs para cada pino de escrita dos processadores na interface serial. Dessa forma, pode-se enviar diferentes códigos objeto para cada processador, e visualizar, independentemente, a saída de cada *interface* serial.

4 Ferramenta de estimativas

A simulação de eventos em nível de portas lógicas da arquitetura MPSoC desenvolvida no contexto deste trabalho, apresentada no Capítulo 3, permite a realização de estimativas de tempo de execução do *software* e estimativas de consumo de energia da arquitetura bastante precisas. Entretanto, os tempos de simulação tornam-se impraticáveis, mesmo para aplicações relativamente simples, que levam algumas centenas de milhares de ciclos para executarem.

Neste sentido, a implementação de uma ferramenta de estimativas que baseia-se na execução detalhada, em nível de ciclo da arquitetura proposta, torna-se uma idéia interessante. O intuito de uma ferramenta é minimizar o tempo de simulação e, também, obter resultados que permitam ao projetista a exploração do espaço de soluções de forma adequada. Muitos detalhes, como chaveamento de portas lógicas e a complexidade da implementação do *hardware* podem ser abstraídos, diminuindo-se dessa forma o tempo necessário para o término de uma simulação.

4.1 Implementação do ISS e sua adaptação para estimativas

Um simulador do conjunto de instruções (ISS) da arquitetura MIPS foi implementado. O simulador executa código objeto nativo, de acordo com a implementação do processador Plasma. Além da execução de instruções, o simulador inclui mecanismos para estimativas de desempenho e consumo de energia. O mecanismo de contagem de ciclos é responsável por estimativas de desempenho. Além disso, mecanismos para abortar a simulação foram implementados, sendo um deles o número de ciclos a serem executados ou o tempo de execução real do *hardware*, parâmetros esses informados pelo usuário.

4.1.1 Funcionamento e estrutura do ISS

O ISS possui estruturas e definições que caracterizam a implementação do processador em *hardware*. Dessa forma, tamanho da memória, faixas de endereços específicos, estruturas como registradores e lógica de controle são bem definidos nessa implementação. O ISS tem como

intuito emular o funcionamento do *hardware*.

Algumas constantes foram definidas na implementação, com o objetivo de generalizar a estrutura do ISS. Essas constantes dizem respeito ao consumo de energia por ciclo, freqüência de operação do processador, latência da UART, faixas de endereço de memória e máscaras de interrupção.

O funcionamento do simulador é relativamente simples. Inicialmente, a aplicação (código objeto) é carregada na memória e o ISS é inicializado (o ponteiro de memória é modificado para o endereço da aplicação). A seguir, as instruções são executadas uma a uma, em um laço. Nesse laço, a rotina que executa um ciclo do processador é chamada, e o contexto do processador é atualizado. Quando um *breakpoint*[1] é encontrado, a execução é interrompida, e relatórios da execução de instruções e saída da UART são gravados em disco. No mesmo laço de execução são atualizadas *flags* referentes ao registrador de interrupções. Esse registrador é alterado conforme o estado da UART, do *timer* e modificações em pinos das portas de entrada e saída. Além disso, a execução de instruções fica interrompida no caso de pausas no processador, ocasionadas por acessos a dispositivos de entrada e saída ou operações multi-ciclo.

A principal estrutura do ISS armazena o estado (ou contexto) do processador. Esta estrutura representa os 32 registradores de propósito geral implementados pela arquitetura MIPS, os registradores PC[2] e PC_NEXT, registradores de resultado das operações de divisão e multiplicação (HI e LO), registrador com o endereço de destino do salto (SKIP), um ponteiro para o endereço de memória, estado de funcionamento do processador e definição de *endianness*[3].

Existem vetores que armazenam os nomes de cada instrução. Esses nomes são utilizados para tornar mais legível a implementação de algumas funções, e também para listar nos relatórios de execução do código o número de vezes que cada instrução foi invocada. Algumas variáveis (ou vetores) são definidos, como o vetor SRAM que representa a memória externa do processador, *cpu_cycles* armazena a contagem de ciclos e *ins_counter_op, ins_counter_func* e *ins_counter_rt* armazenam a contagem de cada instrução executada. Além dessas variáveis são definidas *max_cycles* e *est_energy* que armazenam o número máximo de ciclos a serem executados e a energia consumida estimada.

Na atual implementação funções específicas realizam a leitura e escrita em memória. Como periféricos são mapeados em memória nessa arquitetura, nessas funções é verificado o endereço de leitura ou escrita, e uma decisão quanto ao destino da mesma é tomada (por exemplo,

[1] Ponto onde a execução da simulação é abortada. Esse ponto pode ser um tempo limite de execução pré-definido, ou escrita em um endereço de memória específico utilizado como *trap*.

[2] *Program Counter*.

[3] Forma de representação de dados, ou seja, ordem de *bits* em uma dada arquitetura. Tipicamente, sistemas são entitulados como *Big Endian* ou *Little Endian*.

Nome do endereço	Endereço
RAM_INTERNAL_BASE	0x00000000
RAM_EXTERNAL_BASE	0x10000000
MISC_BASE	0x20000000
UART_WRITE	0x20000000
UART_READ	0x20000000
IRQ_MASK	0x20000010
IRQ_STATUS	0x20000020
GPIO0_OUT	0x20000030
GPIO1_OUT	0x20000040
GPIOA_IN	0x20000050
COUNTER_REG	0x20000060
EXIT_TRAP	0x200000F0

Tabela 5 – Mapa de memória do ISS.

escrever no vetor SRAM ou colocar um caracter no terminal, representando uma escrita em memória ou na UART, respectivamente). Na Tabela 5 é apresentado o mapa de memória do ISS. Os endereços de memória são idênticos aos endereços da implementação do processador em VHDL, contudo, um endereço de *trap* foi reservado com o intuito de abortar a simulação de um algoritmo por *software* (uma escrita neste endereço causa a parada do simulador, entretanto nada acontece quando um dado for escrito em tal endereço na implementação do processador em *hardware*).

A função mais importante do ISS simula a execução de uma instrução do processsador. Nessa função, a estrutura que representa o estado do mesmo é verificada e decisões são tomadas, para a execução da instrução corrente. Para isso, a operação atual é lida da memória (no ISS, do vetor SRAM) e decodificada (no ISS, desmembrada em diversas variáveis). Assim, é possível tomar decisões conforme o tipo de operação a ser realizada. Após a operação, registradores são atualizados e um novo ciclo de funcionamento do processador é executado. No final de cada iteração dessa função, são incrementados os contadores de ciclo (utilizado para estimativas de desempenho) e de consumo estimado de energia, conforme a instrução executada.

4.2 Classificação de instruções

Cada instrução gera estímulos diferentes dentro do processador. Assim, o consumo de energia por ciclo varia de acordo com esses estímulos. Para uma estimativa dinâmica do consumo de energia, as instruções que fazem parte da arquitetura do processador Plasma foram distribuídas

de acordo com diferentes classes, sendo essas: *Arithmetic*, *Branches*, *Loads / Stores*, *Logical*, *Moves* e *Shifts*. Além disso, instruções específicas foram reavaliadas de acordo com a quantidade de dados que manipulam[4]. Dessa forma, a estimativa de energia consumida por ciclo de execução do processador é definida não apenas pelo tipo de instrução, mas também pelos dados manipulados por ela. A modelagem de consumo de energia por instruções, ao invés de classes, aumentaria em muito a complexidade do simulador, além de tornar impraticável a criação de vetores para o treinamento da ferramenta para todos os casos.

A Tabela 6 apresenta o conjunto de instruções implementado no ISS, sendo essas divididas de acordo com as classes apresentadas anteriormente. Este conjunto representa apenas as instruções implementadas no processador Plasma, e não inclui todas as instruções especificadas na arquitetura MIPS.

Arithmetic	**Branch/Jump**	**Load/Store**	**Logical**	**Move**	**Shift**
ADD	BLTZL	LB	AND	MFHI	SLL
ADDI	BLTZALL	LBU	ANDI	MFLO	SLLV
ADDIU	BEQ	LH	LUI	MTHI	SRA
ADDU	BGEZ	LHU	NOR	MTLO	SRAV
DIV	BGEZAL	LW	OR	COP0	SRL
DIVU	BGTZ	SB	ORI	MOVZ	SRLV
MULT	BLEZ	SH	XOR	MOVN	
MULTU	BLTZ	SW	XORI		
SLT	BLTZAL	LL			
SLTI	BNE	SC			
SLTIU	J				
SLTU	JAL				
SUB	JALR				
SUBU	JR				
DADDU	BEQL				
	BNEL				
	BLEZL				
	BGTZL				
	BGEZL				
	BGEZALL				

Tabela 6 – Instruções do processador Plasma divididas em classes.

Para cada classe de instruções foram criadas aplicações de teste, para que fossem utilizadas como vetores de treinamento da ferramenta. Cada aplicação (vetor de treinamento) utiliza na

[4]Algumas instruções, ao serem executadas, ocasionam a inversão de muitos *bits* na lógica interna do processador, enquanto outras não.

maior parte do tempo instruções de uma determinada classe, estimulando o processador de uma determinada forma, de acordo com as instruções. Torna-se evidente que é praticamente impossível se utilizar apenas uma classe de instruções em uma dada aplicação[5], entretanto foi realizado um grande esforço para estressar cada classe de instruções em cada um dos vetores de treinamento.

Classe	Consumo VHDL (J)
Arithmetic	4.0216E-05
Branch/Jump	5.9974E-05
Load/Store	4.2295E-05
Logical	6.2987E-05
Move	4.8211E-05
Shift	7.3199E-05

Tabela 7 – Consumo de energia reportado pela plataforma VHDL monoprocessada.

Cada vetor de treinamento foi executado por 1ms (o que corresponde a 25000 ciclos, a 25MHz) na plataforma inicial[6]. No final da simulação obteve-se um consumo de energia para cada aplicação teste, conforme apresentado na Tabela 7. O consumo de energia reportado pelo simulador foi dividido por 25000, obtendo-se o consumo de energia aproximado por ciclo, para cada classe de instruções.

A Tabela 8 apresenta os resultados obtidos pela simulação de consumo de energia por ciclo para cada classe de instruções.

Classe	Energia por ciclo (J)
Arithmetic	1.60864E-09
Branch/Jump	2.39897E-09
Load/Store	1.69180E-09
Logical	2.51948E-09
Move	1.92844E-09
Shift	2.92796E-09

Tabela 8 – Consumo de energia por classe.

Esses valores foram utilizados como base da ferramenta de estimativas. Entretanto, observou-se que algumas instruções específicas de cada classe (por exemplo, instruções de deslocamento) manipulam dados de forma diferente durante a execução da instrução, excitando mais o circuito

[5]Por exemplo, para se realizar uma multiplicação, é necessário que previamente se carregue os operandos, o que é realizado com instruções de uma classe diferente da instrução de multiplicação.

[6]A plataforma de estimativas inicial é formada pelo núcleo do processador Plasma sintetizado e convertido para uso com uma biblioteca de estimativas de energia, além de outros periféricos.

do processador. Para esses casos, o valor de consumo de energia por ciclo foi multiplicado por dois[7], para levar em conta o consumo de energia devido a variabilidade dos dados no processamento de um determinado algoritmo. O consumo de energia reportado para aplicações reais apresentou resultados satisfatórios, como apresentado no Capítulo 5.

4.3 Replicação do ISS e comunicação entre núcleos

Para a emulação da plataforma MPSoC implementada em VHDL, foi necessária a replicação de diversas estruturas do ISS, além da criação de portas de entrada e saída para comunicação entre processadores. As estruturas de controle, contexto, memória, contagem de instruções e contagem de energia são independentes para cada ISS. Dessa forma, cada processador executa um ciclo por vez em um laço iterativo. O contexto de cada processador é passado para a função que executa uma instrução da arquitetura.

As portas de entrada e saída, entituladas GPIOA_IN e GPIO0_OUT respectivamente, foram implementadas e replicadas na ferramenta, e apresentam funcionalidade semelhante a implementação em VHDL. Além disso, o mesmo protocolo de comunicação entre estas portas e o controle de acesso ao meio de interconexão foi utilizado para que fosse mantida a compatibilidade com os *drivers*[8] já implementados, ou seja, as características de funcionamento do *hardware* que implementa o controle de acesso ao meio de interconexão foram representadas na ferramenta. O uso de códigos objeto torna-se transparente, e não são necessárias adaptações do código ou recompilações para uso no protótipo, simulação VHDL ou na ferramenta.

O consumo de energia do meio de interconexão foi modelado de forma bastante simples. Quando não ocorre transmissão de dados, apenas o consumo estático de energia é anotado no contador de energia do meio de interconexão. Entretando, quando um pacote é enviado, o consumo de energia é anotado para a transmissão do pacote inteiro, que leva diversos ciclos (em torno de 50). A Tabela 9 apresenta o consumo de energia do meio de interconexão, quando ocioso e durante a transmissão de dados. O consumo de energia apresentado foi retirado de simulações dos modelos RTL, conforme a metodologia apresentada no Capítulo 3.

Para anotações de energia, o meio de interconexão foi simulado por 1 ms quando não foram enviados pacotes de dados, e obteve-se dessa forma o consumo estático de energia por ciclo (o valor reportado foi dividido por 25000 ciclos). Após, foram enviados 256 pacotes[9], e o valor de

[7] O valor foi escolhido com base em testes realizados em aplicações diversas

[8] Primitivas de comunicação *Send()* e *Receive()*, utilizadas para trocas de mensagens entre os processadores.

[9] O valor escolhido apresentou resultados satisfatórios e a simulação de mais pacotes não afetaria na precisão dos cálculos realizados.

Meio de interconexão	Energia por ciclo (J)	Energia por pacote (J)
Ocioso	4.68900E-11	2.34450E-09
Dados	1.62562E-10	8.12808E-09

Tabela 9 – Consumo de energia do meio de interconexão.

energia consumida foi divido pelo número de pacotes, obtendo-se dessa forma o consumo de energia médio por pacote de dados. Além disso, pode-se observar a latência para o envio de um pacote de dados, que tem aproximadamente 50 ciclos[10].

4.4 Implementação da ferramenta de estimativas

A ferramenta de estimativas foi descrita em linguagem C, e pode ser compilada com qualquer compilador ANSI. Durante o desenvolvimento, foi utilizado o compilador GCC 4.1.2, em um ambiente Slackware Linux com *kernel* 2.6.21.5. Não foi implementada uma interface gráfica, portanto, a ferramenta é controlada por linha de comando.

Invocando-se a ferramenta sem parâmetros, a seguinte mensagem é retornada (Figura 16), apresentando as instruções de uso.

```
MPSoC Emulator
Sergio Johann Filho - 2007

Usage: mpsoc_emu [n_cycles]
         or
       mpsoc_emu [time unit] e.g. 1000 ns 10 us, 50 ms, 1 s
         or
       mpsoc_emu -1 for infinite running.

 - Object codes must be in /objects directory and named
   code0.bin, code1.bin, code2.bin and code3.bin.
```

Figura 16 – Instruções de uso da ferramenta de estimativas.

A ferramenta encontra-se em estado funcional e estável. Estruturas de controle, memória, registradores internos da arquitetura do processador Plasma, contagem de instruções e estimativas de consumo de energia foram replicados para uma emulação da plataforma completa. A ferramenta atualmente simula a execução de quatro processadores trabalhando em paralelo, interconectados por um controle de acesso e um barramento, sendo as diversas unidades de execução ativadas de acordo com a quantidade de códigos objeto (aplicações) disponíveis para a

[10]Considera-se o desempenho dos *drivers* de comunicação, latência do controle de acesso ao meio de interconexão, *handshake* do barramento, arbitragem e transmissão de dados em si.

ferramenta (entre 1 e 4). Quando apenas um único processador for simulado (um único código objeto disponível) a simulação do meio de interconexão é desabilitada.

Existem diversos diretórios responsáveis pela organização e funcionamento da ferramenta. No diretório raiz, encontram-se o código binário compilado da ferramenta, um *makefile* e os diretórios *objects/*, *reports/* e *source/*. No diretório *objects/* ficam armazenados os códigos objeto (ou aplicações) a serem carregados pela ferramenta. No diretório *reports/* ficam armazenados os relatórios de simulação e no diretório *source/* ficam armazenados os códigos fonte da ferramenta.

```
Code Execution Report - Core 0

CPU cycles: 444370
WCET: 17.7748ms
Estimated energy consumption: 0.002048J (20587 gates Plasma CPU core, CMOS TSMC 0.35um)

Instructions (MIPS I instruction set):
       J:          0;      JAL:       8193;      BEQ:      42321;      BNE:      17385;
    BLEZ:          0;      BGTZ:         0;     ADDI:          0;    ADDIU:      17388;
    SLTI:          0;    SLTIU:          0;     ANDI:      42321;      ORI:      24582;
    XORI:          0;      LUI:      24582;     COP0:          0;     BEQL:          0;
    BNEL:          0;    BLEZL:          0;   BGTZL:          0;       LB:          0;
      LH:          0;       LW:      42321;      LBU:          0;      LHU:          0;
      SB:          0;       SH:          0;       SW:      24580;       LL:          0;
      SC:          0;      SLL:      92835;      SRL:          0;      SRA:          0;
    SLLV:          0;     SRLV:          0;     SRAV:          0;       JR:       8192;
    JALR:          0;     MOVZ:          0;     MOVN:          0;     MFHI:          0;
    MTHI:          0;     MFLO:          0;     MTLO:          0;     MULT:          0;
   MULTU:          0;      DIV:          0;     DIVU:          0;      ADD:          0;
    ADDU:       8193;      SUB:          0;     SUBU:          0;      AND:       8192;
      OR:       8192;      XOR:          0;      NOR:          0;      SLT:       8193;
    SLTU:          0;    DADDU:          0;     BLTZ:          0;     BGEZ:          0;
   BLTZL:          0;    BGEZL:          0;   BLTZAL:          0;   BGEZAL:          0;
 BLTZALL:          0;  BGEZALL:          0;

Instructions executed: 377470
Effective instructions per cycle (IPC): 0.849450
I/O wait cycles: 0

Instructions executed from each class:
Arithmetic:         33774 (8.947466%)
Branch/Jump:        76091 (20.158158%)
Load/Store:         66901 (17.723528%)
Logical:           107869 (28.576841%)
Move:                   0 (0.000000%)
Shift:              92835 (24.594007%)

Packets sent: 8192
```

Figura 17 – Relatório de execução.

Relatórios individuais são reportados para cada processador simulado, como representado pela Figura 17. Nesse relatório são apresentados o número de ciclos executados no processador, o tempo de execução e uma estimativa do consumo de energia. Além disso, detalhes da simulação, como instruções executadas e iterações das mesmas, o IPC, ciclos gastos em entrada e saída, instruções executadas por classe e número de pacotes são apresentados. Além dos relatórios individuais, é reportado um relatório referente a plataforma completa. Esse relatório, representado pela Figura 18 contabiliza o tempo de execução total da plataforma MPSoC, o consumo de energia de cada processador, do meio de interconexão e consumo de energia global, além do número de pacotes trocados entre os processadores (enviados, recebidos e *broadcasts*[11]).

A comunicação entre os processadores encontra-se funcional, e os diversos processadores trocam mensagens entre si utilizando os mesmos *drivers* implementados para uso no protótipo.

[11] Pacotes do tipo *broadcast* são mensagens enviadas de um processador para todos os outros.

```
MPSoC Report

MPSoC cycles: 444379
WCET: 17.7752ms

Estimated energy consumption: 0.007936J
(4 * 20587 gates Plasma CPU core + 11265 gates interconnection structure, CMOS TSMC
0.35um)
    core 0: 0.002048J
    core 1: 0.001933J
    core 2: 0.001933J
    core 3: 0.001933J
       bus: 0.000087J

Packets sent: 8192
    core 0: 8192
    core 1: 0
    core 2: 0
    core 3: 0

Packets received: 24576
    core 0: 0
    core 1: 8192
    core 2: 8192
    core 3: 8192

Broadcasts: 8192
    core 0: 8192
    core 1: 0
    core 2: 0
    core 3: 0
```

Figura 18 – Relatório geral de execução da ferramenta.

Além disso, foram incluídos atrasos da comunicação na ferramenta, com o objetivo de emular de forma coerente o tempo de execução de *software* no ambiente MPSoC. A estrutura mais complexa da ferramenta, além de gerar as maiores variações com relação a estimativa de desempenho e consumo de energia é o ISS, e melhorias na sua precisão aumentariam a precisão da ferramenta.

Arquitetura	**Plataforma VHDL**	**Ferramenta**	**Speedup**
4 CPUs, bus	2400 s	0.0078 s	307692x
2 CPUs, bus	1080 s	0.0052 s	207692x
1 CPU	480 s	0.0038 s	126316x

Tabela 10 – Tempos de simulação.

A Tabela 10 apresenta o tempo necessário para a simulação de 1 ms de execução do *hardware*. São apresentadas três variações de arquitetura (4 processadores interconectados, 2 processadores interconectados ou 1 processador isolado). Cada arquitetura possui diferentes tempos de simulação tanto na ferramenta de estimativas, quanto na simulação VHDL e o ganho em velocidade conseguido com o uso da ferramenta, comparado a simulação VHDL é apresentado na última coluna. Esses valores foram extraídos de simulações executadas em um computador com processador *dual* a 2.8 GHz com 2 GB de memória. A simulação de quatro processadores e meio de interconexão é a mais intensa em termos computacionais. O uso da ferramenta de estimativas permite acelerações no processo de simulação na ordem de centenas de milhares de vezes, comparada a simulação em nível de portas lógicas.

5 Estudos de caso

Neste capítulo serão apresentados estudos de caso, formados por aplicações executando em diferentes arquiteturas. Em um primeiro momento, serão apresentados testes com aplicações monoprocessadas e após, aplicações distribuídas.

Os testes foram escolhidos para simular a execução de aplicações reais em sistemas embarcados, estressando dessa forma as diferentes configurações da arquitetura implementada. Além disso, a maior parte das aplicações aqui simuladas são utilizadas como testes em outros *benchmarks* como MiBench [25].

5.1 Tempos de simulação

As aplicações que foram simuladas tanto em uma plataforma VHDL quanto na ferramenta de estimativas executaram em um computador *dual core* de 2.8GHz e 2GB de memória. A simulação VHDL mostrou-se impraticável para a plataforma completa, composta por quatro processadores e meio de interconexão para aplicações complexas, sendo esse um dos motivos para a não realização de mais testes.

	ET		**ET em**	**ET na**
Benchmark	**Aplicação**	**Arquitetura**	**VHDL**	**Ferramenta**
Sobel	33.7394 ms	1 CPU	16195 s	0.128 s
JPEG	205.3345 ms	1 CPU	98561 s	0.780 s
CRC32	7.8674 ms	1 CPU	3776 s	0.030 s
ISqrt	16.6440 ms	1 CPU	7989 s	0.064 s
ADPCM	267.1405 ms	1 CPU	128228 s	1.015 s
Random	83.1352 ms	1 CPU	39905 s	0.316 s
Broadcast	17.4519 ms	4 CPUs, bus	41885 s	0.136 s
MP ADPCM	206.7305 ms	2 CPUs, bus	223269 s	1.074 s
MP JPEG	85.9968 ms	4 CPUs, bus	206392 s	0.671 s

Tabela 11 – Tempos de simulação dos *benchmarks*.

A Tabela 11 apresenta os tempos de execução das diferentes aplicações utilizadas como

benchmarks, assim como o tempo necessário para a simulação da execução em VHDL (nível de portas lógicas) e na ferramenta de estimativas. O ET da aplicação refere-se ao tempo de execução da mesma no *hardware*, com processadores e meio de interconexão operando na freqüência de 25 MHz.

5.2 Aplicações monoprocessadas

Uma aplicação baseada na implementação de um filtro Sobel [20] foi utilizada como o primeiro exemplo para estimativas de tempo de execução e consumo de energia. Essa aplicação aplica um filtro de detecção de bordas em uma imagem monocromática de dimensões 32x32 *pixels*. O resultado de estimativa de consumo de energia reportado pelo ISS, para a execução da aplicação Sobel foi de $1.617E-03J$, sendo o tempo de execução da aplicação estimado em 34.1119 ms (ou 852797 ciclos a 25 MHz). A simulação em VHDL reportou um consumo de energia de $1.3279E-03J$ e o tempo de execução do software foi de 33.7394 ms (ou 843484 ciclos). O erro na estimativa de desempenho da ferramenta foi de 1.092% enquanto o erro na estimativa de consumo de energia foi de 17.87%. Esse erro de estimativa de consumo de energia, foi um dos maiores encontrados nas aplicações testadas, e trata-se de um *corner case*. Algumas aplicações, devido a suas características, não podem ser eficientemente avaliadas para estimativas de energia em um nível alto como da ferramenta proposta.

Outra aplicação implementada foi um codificador JPEG, que conforma com a norma ISO-10918, sendo implementado inicialmente para a arquitetura x86 e portado para a arquitetura MIPS. Esse codificador foi implementado de acordo com o padrão de referência [48] e o fluxo de *bits* gerado pelo codificador verificou-se conformar com o mesmo. Mais informações são também encontradas em [40, 46]. O codificador utilizado no experimento possui uma imagem de dimensões 32x32 *pixels* estaticamente alocada em memória. O processamento dessa imagem é feito pelo codificador e o resultado (um *bitstream*) é armazenado em memória. O resultado de estimativa de consumo de energia reportado pelo ISS, para a execução da aplicação JPEG foi de $11.122E-03$, sendo o tempo de execução da aplicação estimado em 207.2952 ms (ou 5182381 ciclos). A simulação em VHDL reportou um consumo de energia de $11.094E-03J$ e o tempo de execução estimado foi de 205.3345 ms. Os tempos de simulação são apresentados na Tabela 11. Novamente, o erro na estimativa de desempenho da ferramenta foi baixo (0.954%) e o erro no consumo de energia para esta aplicação foi de 0.25%. Neste caso, a modelagem do processador em alto nível utilizando classificação de instruções foi bem sucedida, estimando de forma coerente o consumo de energia da aplicação e aproximando a simulação VHDL em

termos de precisão.

Outros *benchmarks* foram utilizados para validar os resultados fornecidos pela ferramenta de estimativas. A metodologia foi a mesma utilizada para as aplicações do filtro sobel e do codificador JPEG. Em um primeiro momento a estimativa de desempenho e consumo de energia é realizada através da ferramenta e posteriormente estes resultados são comparados aos resultados obtidos através da simulação VHDL. Os algoritmos baseiam-se nas implementações encontradas em [25], e que são utilizadas para avaliação em processadores embarcados.

A aplicação CRC32 executa o algoritmo de CRC de 32 *bits* (CCITT X.25, seqüência de verificação de nível de enlace), sobre um bloco de dados de tamanho de 16 KBytes, estaticamente alocado na memória. A aplicação ISqrt executa o algoritmo de raíz quadrada com inteiros escalados dos números entre 0 a 1000. A aplicação ADPCM (Intel/DVI ADPCM) codifica e decodifica o equivalente a 1 segundo de áudio. Para tanto, um bloco de dados com 44100 amostras de 16 *bits* gerado aleatoriamente é utilizado na codificação como entrada da aplicação. Como resultado da codificação são geradas pela aplicação 44100 amostras de 4 *bits*. Essas amostras são utilizadas como entrada para o decodificador. O *benchmark* Random é parte da aplicação ADPCM, e realiza a geração de números randômicos.

A Tabela 12 apresenta os resultados dos diversos *benchmarks* utilizados para as medições de desempenho e consumo de energia. É importante observar que estes testes foram realizados considerando uma arquitetura monoprocessada.

Benchmark	**ET (VHDL)**	**Energia (VHDL)**	**ET (ISS)**	**Energia (ISS)**	**Erro ET**	**Erro Energia**
Sobel	33.7394 ms	1.328E-03J	34.1119 ms	1.617E-03J	1.092%	17.87%
JPEG	205.3345 ms	11.094E-03J	207.2952 ms	11.122E-03J	0.954%	0.25%
CRC32	7.8674 ms	0.793E-03J	7.8656 ms	0.772E-03J	0.022%	2.65%
ISqrt	16.6440 ms	1.894E-03J	16.6430 ms	1.960E-03J	0.000%	3.37%
ADPCM	267.1405 ms	31.464E-03J	267.1386 ms	25.751E-03J	0.000%	18.15%
Random	83.1352 ms	7.823E-03J	83.1312 ms	7.740E-03J	0.000%	0.01%

Tabela 12 – *Benchmarks* monoprocessados.

Em dois testes ocorrem erros de maior magnitude (em torno de 18%), e conforme citado anteriormente, tratam-se de *corner cases*. Um erro dessa magnitude apresenta as imprecisões possíveis em uma simulação em mais alto nível, onde o modelo da arquitetura não é demasiadamente refinado, e pode não corresponder ao comportamento da implementação RTL para algumas aplicações. A maioria dos outros casos, entretanto, apresentou imprecisões inferiores a 4%. Os erros relativos a estimativa de desempenho referem-se principalmente à imprevisibilidade de algumas estruturas no controle do processador, por exemplo, predição de desvios

condicionais e também ao estado do *pipeline*. Esses erros, entretanto, apresentaram-se insignificantes. Algumas aplicações possuem instruções que têm um comportamento que pode ser mais precisamente representado pelo modelo em alto nível do processador implementado na ferramenta, e dessa forma apresentam resultados mais coerentes com a implementação do processador em RTL.

5.3 Aplicações multiprocessadas

A arquitetura MPSoC implementada possui até 4 processadores. Foi observado que a maioria dos MPSoC atuais (em torno de 80% destes) possuem entre 2 e 4 processadores. Alguns MPSoCs (em torno de 18%) contém até 16 processadores sendo a maioria microcontroladores como PICs e o 8051.

A Tabela 13 apresenta os resultados de 3 *benchmarks*. Esses testes são aplicações multiprocessadas, onde ocorre comunicação entre os processadores. Primitivas comunicantes *Send()* e *Receive()* são utilizadas para troca de mensagens entre os processadores. O número de mensagens trocadas entre os processadores varia em cada caso.

	ET	**Energia**	**ET**	**Energia**	**Erro**	**Erro**
Benchmark	**(VHDL)**	**(VHDL)**	**(ISS)**	**(ISS)**	**ET**	**Energia**
Broadcast	17.4519 ms	6.537E-03J	17.7752 ms	7.936E-03J	1.818%	17.63%
MP ADPCM	206.7305 ms	47.943E-03J	203.4426 ms	41.106E-03J	1.590%	14.26%
MP JPEG	85.9968 ms	28.806E-03J	86.7661 ms	27.463E-03J	1.153%	4.66%

Tabela 13 – *Benchmarks* multiprocessados.

O *benchmark* Broadcast é uma aplicação do tipo produtor / consumidor, onde um produtor envia 8192 pacotes em *broadcast* para três consumidores. Esta aplicação utiliza o *throughput* máximo da plataforma MPSoC, que é em torno de 510204 pacotes por segundo, ou 1531MB de dados por segundo, considerando todos os atrasos introduzidos pela comunicação e descontando-se 25% da banda utilizada para sinalização e endereçamento.

O *benchmark* MP ADPCM é composto por um codificador e um decodificador ADPCM distribuídos em dois processadores. O primeiro processador preenche um bloco de dados com 44100 amostras de 16 *bits* geradas aleatoriamente e codifica 4410 amostras. Após, o resultado do processamento (4410 amostras de 4 *bits*) é enviado ao segundo processador, através de 735 mensagens (cada mensagem carrega 24 *bits* de dados). O segundo processador recebe as amostras, as coloca em um bloco de dados, e decodifica o bloco, reconstruindo-se dessa forma

4410 amostras PCM de 16 *bits*. O processo é iterado 10 vezes (ou seja, são enviados 100 ms de áudio a cada iteração), e ocorre paralelismo entre os algoritmos de codificação e decodificação, o que diminui o tempo de execução em torno de 34.62%[1], comparado a implementação monoprocessada dos algoritmos de codificação e decodificação.

O último *benchmark* consiste de um codificador JPEG distribuído entre quatro processadores. O primeiro processador trabalha como mestre, e além de coordenar todo o processo de codificação também é responsável pela geração do *bitstream* JPEG e adição de cabeçalhos. Os outros três processadores são responsáveis pelo processamento DCT, quantização e ordenação das amostras das três componentes da imagem (luminância e duas crominâncias). Esse processamento é realizado em paralelo.

Inicialmente, o processador mestre envia o tamanho da imagem aos processadores escravos através de duas mensagens de *broadcast*. Os processadores escravos aguardam em primitivas *Receive* bloqueantes. Em seguida, o processador mestre envia um bloco de 64 amostras das três componentes de imagem por *broadcast* aos processadores escravos, que decodificam cada um sua parte dos pacotes (cada pacote possui 24 *bits* de dados, ou seja, três amostras de 8 *bits*, correspondendo as componentes Y, Cb e Cr). Após o recebimento, os escravos realizam o processamento DCT em paralelo, e aguardam ordens do processador mestre para o recebimento dos resultados. O mestre envia sequencialmente um pacote para cada escravo, desbloqueando-os. Entre cada desbloqueio, o mestre recebe de cada escravo 64 amostras de 16 *bits*, resultantes do processamento, e realiza a codificação Huffman e geração de cabeçalhos. O processo é iterado até o processamento total da imagem. Neste estudo de caso, foi utilizada uma imagem de 32x32 *pixels*, resultando em 16 blocos com dimensões de 8x8 *pixels*.

Foi utilizada a mesma imagem para os algoritmos JPEG monoprocessado e JPEG multiprocessado. O algoritmo distribuído resultou em uma diminuição no tempo de processamento em torno de 58.14%, comparado a implementação monoprocessada. O tempo de processamento aqui citado refere-se a execução da aplicação no *hardware*.

5.4 Consumo de energia do meio de interconexão em aplicações multiprocessadas

O consumo de energia da plataforma é influenciado pelo consumo de energia do meio de interconexão durante a execução de algoritmos distribuídos. Assim, tornam-se necessárias medições e comparações entre a plataforma VHDL e a ferramenta.

[1]Foi desconsiderado o tempo gasto para a geração aleatória de amostras, nos dois casos.

Benchmark	CPUs	Energia (VHDL)	Energia (ISS)	Pacotes Enviados	Erro	Energia Global
Broadcast	4	0.8435E-04J	0.8700E-04J	8192	2.833%	1.29%
MP ADPCM	2	2.8607E-04J	2.9800E-04J	7350	4.003%	0.60%
MP JPEG	4	1.2825E-04J	1.3500E-04J	4130	5.000%	0.45%

Tabela 14 – Estimativas de energia do meio de interconexão.

A Tabela 14 apresenta uma comparação entre as estimativas de consumo de energia do meio de interconexão da ferramenta de estimativas e da simulação VHDL. Medições mostram que os erros nas estimativas de consumo de energia do meio de interconexão da ferramenta são bastante baixos. Além disso, a representatividade do consumo de energia do meio de interconexão no consumo de energia global da plataforma MPSoC é bastante baixo (menor que 1.5%). Isto se deve ao fato de ocorrer baixa atividade na lógica do meio de interconexão, e também por esta lógica representar em torno de 12% da lógica global.

6 Conclusões e trabalhos futuros

Neste trabalho foi apresentada a proposta de uma plataforma MPSoC para estimativas de desempenho de *software* e consumo de energia. Essa platafoma é constituída por quatro processadores MIPS e um meio de interconexão em forma de barramento.

Uma metodologia para estimativas foi apresentada, assim como cada passo para a geração da plataforma de forma a atender diferentes necessidades de simulação. Observou-se, entretanto, que devido ao grande número de portas lógicas necessárias para a implementação da plataforma, sua simulação tornou-se demasiadamente custosa em termos computacionais. Dessa forma, uma ferramenta de simulação em alto nível foi especificada e implementada, com o intuito de reduzir a complexidade computacional para a simulação de aplicações reais.

A ferramenta de estimativas baseia-se na execução do conjunto de instruções MIPS e emulação da comunicação entre diversos processadores. Para o cálculo do tempo de execução de uma aplicação, foi criado um modelo matemático do processador MIPS, de forma a representar o número de ciclos necessários para a execução. O consumo de energia de uma aplicação foi modelado com base em simulações RTL, e um método de classificação de instruções foi utilizado de forma a abstrair detalhes do *hardware* e permitir uma análise em alto nível.

Estudos de caso com algoritmos amplamente utilizados em sistemas embarcados foram criados, com o intuito de validar a ferramenta de estimativas comparada a simulação VHDL. A ferramenta apresentou resultados satisfatórios, e foi comprovado que será de grande importância para a análise e modelagem de aplicações embarcadas distribuídas, em um curto espaço de tempo.

A plataforma MPSoC foi prototipada em um dispositivo FPGA com sucesso, e o funcionamento esperado da plataforma pode ser comprovado. Nessa plataforma podem ser executadas aplicações distribuídas, e os *drivers* de comunicação implementados podem ser utilizados por sistemas operacionais para sincronizar informações entre processadores.

Ferramentas de estimativa são peças fundamentais para análise de cenários (ou restrições) existentes em ambientes MPSoC em níveis de abstração superiores. Com a análise, torna-se possível a verificação da necessidade de otimizações de uma dada arquitetura, para o atendimento de restrições de um determinado projeto. Essas restrições são muitas vezes difíceis de serem atendidas, segundo [45]. As principais restrições existentes atualmente podem ser resu-

midas em:

- *Deadlines*
- Consumo de energia
- Área

Deadlines são restrições de tempo existentes para a execução de um determinado trecho de código, ou aplicação. Dessa forma, para que uma aplicação cumpra com seu *deadline*, esta deve executar dentro de um limite de tempo estipulado conforme a necessidade do projeto. O consumo de energia é, na maioria dos casos, um limite imposto principalmente para o aumento da durabilidade de baterias e diminuição de dissipação de calor. Área é um fator que determina, entre outros fatores, o custo final de uma plataforma MPSoC.

Para atacar essas três restrições, torna-se necessária a utilização de partes de *hardware* que possam ser configuradas, de acordo com a necessidade. O *software* tem grande importância na funcionalidade de um ambiente MPSoC e sua análise, através do uso de ferramentas, permite a obtenção de informações sobre as necessidades em termos de *hardware* para que as restrições de uma aplicação sejam atendidas de maneira mais eficiente. De acordo com [15], eficiência e flexibilidade devem ser cuidadosamente balanceadas para que os requisitos de uma aplicação possam ser atendidos.

Processadores configuráveis são peças chaves nesse aumento de eficiência de sistemas MPSoC. A otimização do *software*, aliado a customização de *hardware* torna-se motivação para um trabalho futuro.

Referências

[1] AMBA Specification (rev 2.0) - http://www.arm.com/products/solutions/AMBAHomePage.html. Acessado, Maio 1999.

[2] Open Core Protocol - http://www.ocp-ip.org. Acessado, Maio 2005.

[3] Aldec. ActiveHDL - http://www.aldec.com/products/. Acessado, Julho 2006.

[4] ArchC. Architecture Description Language - www.archc.org. Acessado, Agosto 2005.

[5] ARM. MaxCore - http://www.arm.com. Acessado, Maio 2005.

[6] T. Austin. SimpleScalar - http://www.simplescalar.com. Acessado, Novembro 2006.

[7] J. R. Bammi, W. Kruijtzer, L. Lavagno, E. Harcourt, and M. T. Lazarescu. Software performance estimation strategies in a system-level design tool. In *CODES '00: Proceedings of the eighth international workshop on Hardware/software codesign*, pages 82–86, New York, NY, USA, 2000. ACM.

[8] L. Benini, A. Bertozzi, and F. Menichelli. *Journal of VLSI Signal Processing*, v.41, n. 2, p. 169-182, 2005.

[9] R. A. Bergamaschi and J. Cohn. The A to Z of SoCs. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 790–798, New York, NY, USA, 2002. ACM.

[10] G. Bontempi and W. Kruijtzer. A Data Analysis Method for Software Performance Prediction. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 971, Washington, DC, USA, 2002. IEEE Computer Society.

[11] T. D. Burd. *Energy-Efficient Processor System Design*. PhD thesis, University of California, Berkeley, 2001.

[12] E. Carara. *Arquiteturas para Roteadores de Redes Intra-chip - Trabalho de Conclusão I - Pontifícia Universidade Católica do Rio Grande do Sul*. Porto Alegre, Brasil, 2004. (Em português).

[13] J. Chen, M. Dubois, and P. Stenstrom. Integrating complete-system and user-level performance/power simulators: the SimWattch approach. In *Performance Analysis of Systems and Software, IEEE International Symmposium on Performance Analysis of Systems and Software*, volume 1, pages 1–10, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[14] A. Colin and I. Puat. Worst-Case Execution Time analysis for a Processor With Branch Prediction. *Journal of Real-Time Systems, v.18, n:2/3, p.249-274*, 2000.

[15] J. Cong, Y. Fan, G. Han, and Z. Zhang. Application Specific Instruction Generation for Configurable Processor Architectures. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 183–189, New York, NY, USA, 2004. ACM.

[16] Coware. LisaTek - http://www.coware.com/products/lisatek.php. Acessado, Agosto 2005.

[17] D. Crisu, S. Cotofana, S. Vassiliadis, and P. Liuha. High-level energy estimation for arm-based socs. In Andy D. Pimentel and Stamatis Vassiliadis, editors, *SAMOS*, volume 31 of *Lecture Notes in Computer Science*, pages 168–177. Springer, 2004.

[18] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Elsevier Inc, Amsterdam, 2004.

[19] J. Engblom, A. Ermedahl, and F. Stappert. A Worst-Case Execution-Time Analysis Tool Prototype for Embedded Real-Time Systems. In *The first Workshop on Real-Time Tools (RT-TOOLS 2001) held in conjunction with CONCUR 2001, Aalborg*. Paul Pettersson, August 2001.

[20] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Sobel Edge Detector - http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm. Acessado, Outubro 2006.

[21] F. Fummi, S. Martini, G. Perbellini, and M. Poncino. Native ISS-SystemC Integration for the Co-Simulation of Multi-Processor SoC. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10564, Washington, DC, USA, 2004. IEEE Computer Society.

[22] P. Giusto, G. Martin, and E. Harcourt. Reliable estimation of execution time of embedded software. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 580–589, Piscataway, NJ, USA, 2001. IEEE Press.

[23] J. R. Goodman. *Energy Scalable Reconfigurable Cryptographic Hardware for Portable Applications*. PhD thesis, Massachusetts Institute of Technology, Agosto 2002.

[24] Mentor Graphics. ModelSim - http://www.mentor.com/products/. Acessado, Agosto 2006.

[25] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. MiBench: A free, commercially representative embedded benchmark suite. volume 3, pages 3–14, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[26] J. Heinrich. MIPS R4000 Microprocessor User's Manual - http://www.mips.com/manuals/R4000_Users_Manual_2Ed.pdf. Acessado, Junho 2006.

[27] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr. A methodology for the design of application specific instruction set processors (ASIP) using the machine description language LISA. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 625–630, Piscataway, NJ, USA, 2001. IEEE Press.

[28] M. Lajolo, M. Lazarescu, and A. Sangiovanni-Vincentelli. A compilation-based software estimation scheme for hardware/software co-simulation. In *CODES '99: Proceedings of the seventh international workshop on Hardware/software codesign*, pages 85–89, New York, NY, USA, 1999. ACM.

[29] P. Landman. High-level power estimation. In *ISLPED '96: Proceedings of the 1996 international symposium on Low power electronics and design*, pages 29–35, Piscataway, NJ, USA, 1996. IEEE Press.

[30] R. Laupers. HDL-based modeling of embedded processor behavior for retargetable compilation. In *ISSS '98: Proceedings of the 11th international symposium on System synthesis*, pages 51–54, Washington, DC, USA, 1998. IEEE Computer Society.

[31] X. Li, T. Mitra, and A. Roychoudhury. Accurate Timing Analysis by Modeling Caches, Speculation and their Interaction. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pages 466–471, New York, NY, USA, 2003. ACM.

[32] Y. S. Li, S. Malik, and A. Wolfe. Performance Estimation of Embedded Software with Instruction Cache Modeling. In *Readings in hardware/software co-design*, pages 167–178, Norwell, MA, USA, 1995. Kluwer Academic Publishers.

[33] Y.-T. S. Li and S. Malik. Performance Analysis of Embedded Software Using Implicit Path Enumeration. In *DAC '95: Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, pages 456–461, New York, NY, USA, 1995. ACM.

[34] C. A. M. Marcon. *Modelos para o Mapeamento de Aplicações em Infra-estruturas de Comunicação Intrachip*. PhD thesis, Universidade Federal do Rio Grande do Sul, UFRGS, Brasil., 2005.

[35] G. Martin and H. Chang. System on Chip Design. In *9th International Symposium on Integrated Circuits, Devices Systems (ISIC) - Tutorial 2*, pages 12–17, 2001.

[36] C. J. Mauer, M. D. Hill, and D. A. Wood. Full-System Timing-First Simulation. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 108–116, New York, NY, USA, 2002. ACM.

[37] Microlib. Microlib - PPC750 - http://microlib.org. Acessado, Novembro 2006.

[38] P. Mishra, M. Mamidipaka, and N. Dutt. Processor-memory Coexploration Using an Architecture Description Language. In *ACM Transactions on Embedded Computer Systems*, volume 3, pages 140–162, New York, NY, USA, 2004. ACM.

[39] A. Muttreja, A. Raghunathan, S. Ravi, and N. K. Jha. Automated Energy/Performance Macromodeling of Embedded Software. In *DAC '04: Proceedings of the 41st annual Design Automation Conference*, pages 99–102, New York, NY, USA, 2004. ACM.

[40] W. B. Pennebacker and J. L. Mitchell. *JPEG: Still Image Data Compression Standard.* New York: Van Nostrand-Reinhold, 1992.

[41] R. Reis. *Concepção de Circuitos Integrados*. Editora Sagra Luzzatto, Instituto de Informática da UFRGS, 2000.

[42] K. Richter, M. Jersak, and R. Ernst. A Formal Approach to MPSoC Performance Verification. *IEEE Computer*, v. 36, n. 4, p. 60-67, 2003.

[43] J. T. Russell and M. F. Jacome. Architecture-Level Performance Evaluation of Component-Based Embedded Systems. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pages 396–401, New York, NY, USA, 2003. ACM.

[44] J. Schneider and C. Ferdinand. Pipeline Behaviour Prediction for Superscalar Processors by Abstract Interpretation. In *LCTES '99: Proceedings of the ACM SIGPLAN 1999 workshop on Languages, compilers, and tools for embedded systems*, pages 35–44, New York, NY, USA, 1999. ACM.

[45] Lesley Shannon and Paul Chow. Standardizing the Performance Assessment of Reconfigurable Processor Architectures. In *FCCM '03: Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, page 282, Washington, DC, USA, 2003. IEEE Computer Society.

[46] D. Solomon. *Data Compression. The Complete Reference*. Springer - Verlag, New York, 2nd edition, 2000.

[47] Synopsys. Synopsys System Studio - http://www.synopsys.com. Acessado, Outubro 2005.

[48] The International Telegraph and Telephone Consultative Committee. ISO/IEC 10918-1 Information Technology Digital Compression and Coding of Continuous-tone Still Images Requirements and Guidelines, Novembro 1992.

[49] Tensilica. Compilador XPRES - http://www.tensilica.com/. Acessado, Novembro 2006.

[50] A. Wieferink, T. Kogel, R. Leupers, G. Ascheid, H. Meyr, G. Braun, and A. Nohl. A System Level Processor/Communication Co-Exploration Methodology for Multi-Processor System-on-Chip Platforms. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 21256, Washington, DC, USA, 2004. IEEE Computer Society.

[51] F. Wolf and R. Ernst. Intervals in Software Execution Cost Analysis. In *ISSS '00: Proceedings of the 13th international symposium on System synthesis*, pages 130–135, Washington, DC, USA, 2000. IEEE Computer Society.

[52] C. Zeferino. *Redes-em-Chip: Arquiteturas e Modelos para Avaliação de Área e Desempenho*. PhD thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil, 2003. (Em português).