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

MODELAGEM ABSTRATA PARA HARDWARE DE MPSOC

CARLOS ALBERTO PETRY

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador Prof. Dr. Ney Laert Vilar Calazans

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

P498m Petry, Carlos Alberto
Modelagem abstrata para hardware de MPSoC / Carlos Alberto Petry. – Porto Alegre, 2009.
113 p.

Diss. (Mestrado) – Fac. de Informática, PUCRS
Orientador: Prof. Dr. Ney Laert Vilar Calazans

1. Informática. 2. Multiprocessamento. 3. Modelagem de Sistemas. I. Calazans, Ney Laert Vilar. II. Título.

CDD 004.35

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

Reservado para o “Termo de apresentação”

AGRADECIMENTOS

Meu primeiro agradecimento é, com toda a certeza, ao divino Pai Eterno, Aquele que tudo provê e minha vida conduz.

Em segundo lugar quero agradecer à minha família. À minha esposa pela paciência, pelo estímulo e pelas palavras carinhosas que sempre recebi. Aos meus filhos, por acreditarem e apoiarem meus sonhos, mesmo que isto tenha implicado, em muitas ocasiões, minha ausência em momentos de alegria, de tristeza e de necessidade.

Agradeço também à minha família ascendente. À minha Mãe pela pessoa santa que é por dedicar toda a sua vida em prol de sua família, em especial a mim. Ao meu Pai, em memória, pelos exemplos de vida que me deixou. Aos meus irmãos José Alfredo, Ademar e Gelson, pelos grandes companheiros que tenho. E por fim à minha tia Lita por tantas orações que a mim dedicou.

Agradeço às pessoas que em mim acreditaram quando nesta jornada decidi trilhar. Aos professores Luiz Alberto Steglich e Jair Cordeiro, pelos grandes ensinamentos e confiança que em mim sempre tiveram.

Aos professores do curso de pós-graduação desta instituição, agradeço pela dedicação e empenho em ensinar, sem a ninguém preterir destaco os professores Avelino, Moraes, Hessel e ao meu orientador Ney. Aos grandes colegas de curso, amigos que aqui fiz, desejo grande sucesso. Em especial gostaria de destacar quatro: Edson, pela constância, disponibilidade e presteza em me auxiliar em tantos momentos; Ost, pelo apoio e muitos momentos de alegria; Rafael, embora há tão pouco nos conheçamos pelo sincero amigo que és; Flaviano, pelos muitos momentos de alegria e ansiedade que compartilhamos no convívio deste curso. E não poderia deixar de citar, aos funcionários desta entidade, pela presteza e dedicação que sempre tiveram em suas atividades. A todos vocês um forte abraço e muito sucesso.

Por fim, gostaria de agradecer a qualquer pessoa que tenha esquecido de fazer a justa menção.

MODELAGEM ABSTRATA PARA HARDWARE DE MPSOC

RESUMO

A grande quantidade de funcionalidades integradas aos equipamentos digitais atuais, como telefones celulares, handhelds, consoles de jogos e smart phones, vem criando diversos desafios a serem superados pelos projetistas destes sistemas. Entre estes desafios pode-se citar o aumento do desempenho e a flexibilidade, a diminuição da potência consumida e a redução de custos. As atuais tendências para desenvolvimento de sistemas complexos apontam para o uso de sistemas multiprocessados integrados em um único chip (do inglês, Multiprocessor Systems-on-Chip - MPSoCs). MPSoCs são considerados uma solução apropriada para a realização de sistemas eletrônicos digitais de alta complexidade. A alta capacidade de computação paralela sozinha justifica tal afirmativa. Para utilizar eficientemente o grande número de recursos existentes em MPSoCs se faz necessária a exploração do espaço de projeto em alto nível de abstração, de forma a avaliar diferentes alternativas de implementação em tempo adequado de desenvolvimento. Diversos são os esforços realizados tanto pela academia quanto pela indústria para superar os desafios inerentes ao desenvolvimento de tais sistemas. Entre as propostas consideradas para superar os desafios a maioria capitaliza no uso de duas técnicas: o aumento do reuso de módulos IP e o aumento do nível de abstração em que se faz a captura inicial do projeto. O uso de MPSoCs é uma forma natural de aumentar o reuso de hardware e software. O presente trabalho aborda a modelagem de MPSoCs endereçando a segunda destas técnicas: aumento de abstração na captura do projeto do sistema. Disponibiliza-se um modelo funcional em alto nível de abstração do hardware do sistema multiprocessado HeMPS, desenvolvido no ambiente comercial System Studio da empresa Synopsys. A modelagem abstrata proposta propicia acelerar o tempo de simulação do sistema e permite flexibilidade aumentada na exploração do espaço de projeto de aplicações sobre o sistema HeMPS. O modelo gerado inclui múltiplas instâncias de um processador RISC, o Plasma, e uma rede de comunicação intrachip, HERMES, e módulos de hardware acessórios. O processador é modelado a partir de um simulador do conjunto de instruções, e a rede é descrita no nível de abstração de transação. A modelagem inclui também parte de um núcleo de sistema operacional multitarefa executando sobre os processadores do sistema HeMPS. Resultados iniciais mostram um ganho de até três ordens de magnitude em termos de tempo de simulação, para o processador do sistema, quando comparado à simulação RTL deste.

Palavras Chave: Modelagem abstrata, MPSoC, multiprocessamento, multitarefa, NoC.

ABSTRACT MODELING FOR MPSOC HARDWARE

ABSTRACT

The large amount of functionality integrated in current digital devices such as cell phones, handheld computers, game consoles and smart phones is bringing up several design challenges. Among these challenges, it is possible to cite increase performance and flexibility, reduce power consumption and reduce cost. The current trends in the development of such complex systems point to the use of Multiprocessor Systems-on-Chip (MPSoCs). MPSoCs area considered today as an appropriate solution for the realization of highly complex digital electronic systems. Their high capacity for parallel processing alone justifies this statement. To employ the large amount of resources provided by MPSoCs efficiently, it is necessary to explore the application design space at high levels of abstraction. This is important to assess many different implementation alternatives in a timely fashion. Several efforts are under way both in industry and in the academy to overcome the mentioned challenges to develop such systems. Among the propositions available, several, if not all, plead the use of two techniques: the increase of design reuse and the increase of the abstraction level in which designs are captured. The use of MPSoCs is a natural way to provide hardware and software reuse. The present work addresses the use of MPSoCs and focus on using the second technique. It provides a highly abstract functional model of the hardware for an MPSoC called HeMPS. The abstract modeling employed the commercial environment System Studio of Synopsys. The proposed abstract modeling process enables accelerating the system simulation time and increases the system description flexibility to support design space exploration for applications running on the HeMPS system. HeMPS includes multiple instances of an open source RISC processor called Plasma, an intrachip communication network called HERMES, and some accessory hardware modules. The processor is modeled from its instruction set simulator and the network is described at the transaction abstraction level. The modeling also includes part of a multitask operating system microkernel that executes on HeMPS processors. Initial results for the processor system only display simulation time gains that are up to three orders of magnitude faster than the Plasma RTL model simulation.

Keywords: MPSoC, multiprocessing, multitask, NoC, high abstraction level.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ASIC	Application Specific Integrated Circuit
ASE	Application Specific Extension
CCSS	CoCentric System Studio
CF	Control Flow
CFG	Control Flow Graph
CPU	Central Processing Unit
DF	Data Flow
DFG	Data Flow Graph
DMA	Direct Memory Access
DS	Diagrama de Seqüência
DSL	Digital Subscriber Line
DSP	Digital Signal Processor
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FSM	Finite State Machine
HeMPS	HERMES Multi Processor System
IP	Intellectual Property
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
MIPS	Microprocessor without Interlocked Pipeline Stages
MMU	Management Memory Unit
MoC	Model of Computation
MPSoC	Multi Processor System on Chip
MP	Master Processor
NAT	Network Address Translation
NI	Network Interface
NoC	Network on Chip
NORMA	No Remote Memory Access
OSI	Open System Interconnect
PAT	Payload Abstraction Technique
PC	Program Counter
PDA	Personal Digital Assistant
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SCE	System-on-Chip Environment
SL	Slave Processor
SLD	System Level Design
SoC	System-on-Chip
TA	Transaction Accurate
TCB	Task Control Block
TL	Transaction Level
TLB	Translation Lookaside Buffer
TLM	Transaction Level Model
UART	Universal Asynchronous Receiver-Transmitter
UML	Unified Modeling Language
VA	Virtual Architecture

VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

SUMÁRIO

1 INTRODUÇÃO

Com o desenvolvimento da tecnologia, sobretudo a digital, uma vasta quantidade e variedade de sistemas tecnológicos têm invadido nosso cotidiano. Muitos equipamentos fazem uso de alta tecnologia como meio de viabilizar o seu funcionamento. As funcionalidades oferecidas pelos equipamentos digitais vêm evoluindo rapidamente. Um claro exemplo é o telefone celular, cuja principal característica, permitir a realização de chamadas telefônicas de forma móvel, alia-se a outros recursos incorporados ao aparelho, como agenda e calculadora, acesso à Internet, etc., tornando-se cada vez mais partes essenciais deste tipo de telefone.

A grande quantidade de funcionalidades integradas a equipamentos digitais atuais gera constantes desafios a serem superados pelos projetistas. Entre os vários desafios que se apresentam como críticos ao projetar novos sistemas, cinco são freqüentemente destacados [ANG06] [HEN03] [KOG02] [MAA08] [POP07] [TIB07]:

- Gerenciar o aumento do desempenho e flexibilidade;
- Manter a confiabilidade do sistema;
- Aperfeiçoar o uso da área de silício;
- Gerenciar a crescente complexidade de projeto;
- Evitar o aumento ou, se possível, reduzir o tempo necessário para projeto e verificação do sistema visando:
    - reduzir os custos de projeto e implementação;
    - atender à crescente exigência pela redução do tempo de chegada do produto ao mercado (do inglês time-to-market).

O acréscimo de complexidade tipicamente aumenta a demanda de tempo para o desenvolvimento de novos produtos. Por outro lado, as pressões de mercado estão forçando a diminuição deste tempo para dispositivos mais complexos, atingindo-se o conhecido problema da lacuna de produtividade em projeto (do inglês design productivity gap). A lacuna de produtividade ocorre devido à exigência de aumento da capacidade de produção não acompanhada pela capacidade de projetistas em aproveitar os recursos das tecnologias frente ao avanço destas, a qual imprime circuitos integrados cada vez mais densos.

Duas maneiras fundamentais para atender os desafios citados são o aumento da reutilização maciça de hardware e software de componentes, durante o projeto de sistemas embarcados complexos, e uso de sistemas multiprocessados em um único chip (MPSoC) [JER05]. MPSoCs têm sido crescentemente considerados como a solução mais apropriada para a realização de sistemas eletrônicos digitais de alta complexidade. Porém, para que estes sistemas sejam viáveis precisam ser projetados, simulados e

verificados, devendo operar conforme a sua especificação de requisitos [JER05].

Atualmente, o paradigma de projeto mais empregado baseia-se na captura de descrições de sistemas no nível de abstração de transferência entre registradores (do inglês, Register Transfer Level ou RTL). Entretanto, o projeto partindo de descrições RTL não oferece suporte adequado para o desenvolvimento de sistemas muito complexos [DON04].

Outra maneira de enfrentar os desafios citados anteriormente vem sendo proposta tanto na indústria quanto na academia, como forma de dar suporte à superação dos desafios existentes. Basicamente esta maneira pode ser resumida conforme afirmam Cai e Gajski [CAI03]: o nível de abstração deve ser elevado acima do RTL como forma de manter a produtividade no desenvolvimento de projetos e tratar a crescente complexidade de SoCs e MPSoCs, levando a ganhos de produtividade de várias ordens de grandeza, em termos, por exemplo, de tempo de simulação.

O presente trabalho aborda a modelagem abstrata de MPSoCs como forma de aumentar o desempenho do processo de validação por simulação de sistemas multiprocessados. Isto é obtido utilizando-se de um nível de abstração localizado acima do RTL, aplicado a cada componente modelado do MPSoC. O processo de modelagem abstrata é restrito aqui sobre tudo ao hardware de um MPSoC específico, denominado HeMPS [WOS07]. Outra restrição imposta ao trabalho é o uso de uma ferramenta de modelagem comercial específica, o ambiente System Studio da Synopsys [SYN08]. A justificativa para uso deste ambiente de desenvolvimento se dá pela sua disponibilidade e relativa facilidade para realizar a captura e modelagem abstrata dos componentes considerados fazendo uso da linguagem SystemC.

1.1 Motivação

Entre as motivações do presente trabalho destaca-se:

- a importância de MPSoCs para sistemas eletrônicos digitais atuais, incluindo arquiteturas com centenas de componentes [HEN03] interconectados através de uma infra-estrutura de comunicação, como por exemplo, NoCs;
- o uso de modelos desenvolvidos em níveis de abstração superiores ao RTL largamente propostos na atualidade, a fim de permitir vencer os desafios impostos pelo projeto de equipamentos digitais atualmente em desenvolvimento ou que serão desenvolvidos no futuro próximo.

O presente trabalho tem por principal objetivo disponibilizar uma descrição executável contendo módulos compatíveis com o hardware do sistema HeMPS, descritos em níveis de abstração mais elevados que o RTL. A descrição executável resultante deste trabalho tem como fim ser um modelo que permite o desenvolvimento e teste de software escrito para a arquitetura HeMPS de forma mais eficiente.

1.2 Modelo conceitual

A Figura 1 apresenta o modelo conceitual utilizado como base para este trabalho. Ela mostra o diagrama de blocos do sistema HeMPS, um MPSoC composto de elementos homogêneos de processamento, infra-estrutura de interconexão baseada em uma NoC e um repositório de tarefas contendo aplicações que são distribuídas pelo processador mestre (MP) aos processadores escravos (SL). O elemento de processamento do sistema HeMPS, Plasma-IP SL, por sua vez, inclui um processador, uma memória local, um controlador de acesso direto a memória e uma interface com a NoC.

O software aqui considerado pode ser definido em duas classes: (i) o sistema operacional executado em cada processador, microkernel, e (ii) o conjunto de tarefas a serem executas de forma distribuída nos processadores escravos, contidas inicialmente no Repositório de Tarefas. Conforme já mencionado, o objetivo deste trabalho prioriza a modelagem do hardware do sistema HeMPS. Entretanto, com o intuito de gerar um modelo capaz de executar ao menos parte do microkernel, os módulos produzidos foram validados através do uso do próprio software do sistema.

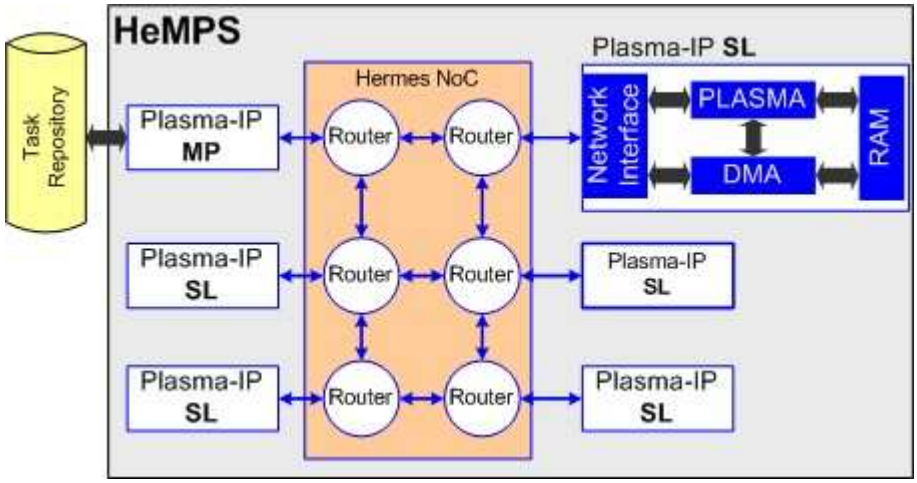

Figura 1 – Modelo conceitual usado como base para este trabalho composto de: elementos de processamento (Plasma-IP), a infra-estrutura de comunicação (NoC HERMES) e como componentes de software: sistema operacional executado em cada Plasma-IP e tarefas contidas inicialmente no Repositório de Tarefas.

1.3 Objetivos

Esta Seção apresenta os objetivos do presente trabalho. Inicialmente são apresentados os objetivos estratégicos:

1. trazer para o grupo de pesquisa do Autor a capacidade de uso de processos eficazes para validação de sistemas complexos em alto nível de abstração;
2. dominar um método de modelagem em alto nível de abstração baseado em ferramentas comerciais;

3. disponibilizar um modelo mais abstrato do hardware da plataforma HeMPS.

Para alcançar os objetivos estratégicos, propõem-se os seguintes objetivos específicos:

4. modelar uma instância do processador Plasma a partir de um simulador do conjunto de instruções (ISS) do MIPS-I;
5. adaptar uma implementação abstrata no nível de transação (em inglês, transaction level, TL) da rede intrachip HERMES [MOR04a] para operar no ambiente comercial selecionado;
6. criar um modelo, descrito em alto nível de abstração, do hardware da plataforma HeMPS a partir dos modelos disponibilizados nos objetivos específicos 1 e 2;
7. simular o modelo obtido no objetivo específico 3 conjuntamente com aplicações simples de software, a fim de validar o uso do modelo abstrato;
8. comparar de forma quantitativa a modelagem realizada com o modelo RTL disponível, do ponto de vista de eficiência de simulação.

1.4 Organização do documento

O restante desta dissertação está organizado conforme descrito a seguir.

O Capítulo 2 apresenta o estado da arte em modelagem de sistemas abstratos. São abordados trabalhos que realizam a modelagem tanto de elementos de comunicação como elementos de processamento.

O Capítulo 3 apresenta os trabalhos relacionados a esta dissertação. Nele são abordados dois trabalhos desenvolvidos no grupo de pesquisa do Autor e um trabalho realizado fora deste grupo, salientando as alterações e adições necessárias para compor o sistema de referência no aqui empregado. A parte final desta seção apresenta o ambiente de desenvolvimento System Studio da empresa Synopsys. Esta ferramenta serviu de ambiente base para desenvolver os módulos do modelo abstrato considerado. São apresentados os dois principais formatos usados para modelagem de sistemas e uma introdução às principais características do ambiente gráfico da ferramenta.

A partir do capítulo 4 inicia-se a apresentação do processo de modelagem dos três principais módulos do sistema HeMPS. Neste Capítulo apresenta-se a modelagem do subsistema de processamento (Plasma). O Capítulo introduz as características do ISS, utilizado como base para a modelagem e apresenta as fases de integração e modelagem executadas para gerar o elemento de processamento.

O Capítulo 5 apresenta a modelagem realizada para integrar e disponibilizar uma instância da infra-estrutura de comunicação HERMES. A geração deste módulo teve por

base o modelo no nível de transação (em inglês, *transaction level model*, TLM) já desenvolvido no grupo do Autor, que necessitou passar por alterações e recebeu acréscimos, a fim de que pudesse ser integrado ao ambiente.

O Capítulo 6 apresenta a modelagem realizada para integrar o modelo abstrato do sistema HeMPS no ambiente System Studio. Foram considerados para esta tarefa os modelos introduzidos nos dois capítulos anteriores.

O Capítulo 7 apresenta o processo de validação dos diversos módulos e os resultados obtidos a partir das simulações executadas. São apresentadas as validações dos módulos desenvolvidos nos capítulos 4 a 6 e os resultados obtidos nas simulações do subsistema Plasma e do sistema HeMPS.

Por fim, o Capítulo 9 apresenta as conclusões do presente trabalho e sugestões de trabalhos futuros.

2 ESTADO DA ARTE

Embora muito seja comentado no meio industrial e na academia sobre a modelagem abstrata de sistemas e apesar da disponibilidade de ferramentas comerciais poderosas dando suporte a este tipo de modelagem, não é tarefa simples obter informações sobre fluxos baseados na captura de projeto em alto nível de abstração. Esta Seção apresenta alguns trabalhos encontrados na literatura sobre o tema.

2.1 Revisão de trabalhos prévios

2.1.1 Modelagem baseada em C/C++/SystemC

Kogel e Bussaglia [KOG02] propõem uma metodologia para realizar a exploração arquitetural de sistemas computacionais descritos em SystemC fazendo uso do ambiente System Studio [SYN08]. O principal objetivo perseguido pelos Autores é definir uma arquitetura de sistema que permita oferecer desempenho, flexibilidade e baixo consumo de potência, e que atenda às especificações de áreas de aplicação tais como comunicação wireless, redes de computadores e processamento de dados multimídia.

Os Autores consideram dois domínios durante o fluxo de projeto: especificação e implementação. Eles propõem também a consideração de um domínio intermediário, denominado nível de sistema (do inglês System Level).

No domínio de especificação, os requisitos são estabelecidos e as funcionalidades necessárias definidas, gerando um documento de especificação do sistema. Inicia-se pelos procedimentos de captura e particionamento das funcionalidades, com o objetivo de gerar o modelo funcional do sistema. O próximo passo é gerar o modelo abstrato da arquitetura a partir do modelo funcional. A fim de permitir a captura das funcionalidades e definir o mapeamento da arquitetura abstrata, estas funcionalidades são particionadas em módulos SystemC. Para imprimir maior rigor metodológico, alguns critérios são usados na definição do mapeamento, incluindo a avaliação da troca de informações entre módulos e a análise de localidade de algoritmos. Como resultado é gerado um modelo mapeado para a arquitetura em um nível de abstração superior ao RTL, porém mais detalhado que a especificação original. O domínio de implementação objetiva elaborar a arquitetura de hardware, conciliando restrições tais como consumo, área do circuito e velocidade. A partir daí segue-se fluxo convencional de desenvolvimento.

Os Autores demonstraram a possibilidade do uso de metodologias de projeto em nível de sistema (em inglês System Level Design ou SLD) no desenvolvimento de SoCs, aplicando-as via uso de ferramentas comerciais, no caso o System Studio. Os resultados finais demonstram a redução de linhas de código escritas e aumento do desempenho da

simulação em cerca de duas ordens de magnitude quando comparado a uma captura a partir de descrição RTL.

Jang et al. [JAN04] apresentam uma metodologia de modelagem de SoCs em alto nível de abstração, que permite projetar, verificar e validar sistemas complexos, além de possibilitar a exploração da arquitetura. O modelo abstrato criado baseou-se em um modelo RTL já implementado e sintetizado, no qual foi detectado baixo desempenho na comunicação. A implementação abstrata foi desenvolvida com precisão próxima ao modelo RTL existente e comparada a última. O desenvolvimento, verificação, validação e posterior exploração da arquitetura foram realizadas através do uso do ambiente System Studio [SYN08]. Módulos de processamento baseados no processador ARM e no barramento AMBA foram providos pela biblioteca DesignWare [SYN08a].

Os projetistas que desenvolveram a versão RTL também implementaram a versão abstrata do sistema. A modelagem se baseou no SoC S3C2510, que corresponde a um microcontrolador de rede para sistemas baseados no padrão Ethernet, desenvolvido pela empresa Samsung, eliminando os módulos não diretamente relacionados à funções de tradução de endereços de rede (em inglês Network Address Translation ou NAT). Os módulos foram projetados usando modelos arquiteturais (SystemC) com metodologia de modelagem TL. Também foram testados quanto à funcionalidade, a partir da execução de aplicações simples. O mesmo modelo abstrato, além de permitir verificar a fonte relacionada ao baixo desempenho detectado na versão RTL, permitiu aos projetistas realizarem testes considerando diferentes variações da arquitetura via simulação.

O modelo de alto nível disponibilizou aos projetistas uma plataforma básica para ser usada no desenvolvimento e avaliação de futuros produtos. Permitiu avaliar problemas ocorridos na versão RTL previamente sintetizada e atingiu velocidades de simulação superior a uma centenas de vezes se comparado à simulação da versão RTL.

Benini et al. [BEN05] propõem um ambiente para exploração e simulação de MPSoCs baseado em SystemC 1.0.2, denominado MPARM. O ambiente contém seis componentes: (i) modelos de processador ARM, (ii) modelos de barramentos AMBA, (iii) modelos de memória, (iv) suporte à programação paralela, (v) sistema operacional uCLinux e (vi) ferramentas para desenvolvimento em linguagem C (GNU toolchain). Outra característica do ambiente é o encapsulamento de um ISS ARM desenvolvido em linguagem C++, fazendo uso de um wrapper SystemC. Segundo os autores o ambiente permite a exploração de diferentes arquiteturas de hardware e a análise de padrões de interação entre processadores, meios de armazenamento compartilhados e meios de comunicação. Segundo os Autores, isto permite definir métricas de desempenho, como média de ciclos de espera pelo barramento de acordo com o tamanho da cache, como esclarecem os resultados apresentados no artigo.

Beltrame et al. [BEL08] propõem uma plataforma de alto nível para modelagem e simulação de MPSoCs baseada em SystemC-TL e Python, denominada ReSP. Conforme

os Autores, a integração da linguagem Python junto à biblioteca SystemC aumenta a capacidade de modelagem e simulação. Os mesmos Autores apresentam em [BEL08b] uma evolução desse trabalho, uma metodologia para exploração de espaço de projeto em alto nível de abstração de plataformas reconfiguráveis. Esta metodologia utiliza a plataforma ReSP estendida para trabalhar com arquiteturas dinamicamente reconfiguráveis de modo a gerar um perfil da aplicação. A partir deste perfil as rotinas de maior custo computacional, como laços aninhados, são identificadas e migradas para hardware de forma dinâmica. A arquitetura sobre a qual a metodologia é aplicada é composta por seis componentes: (i) quatro processadores ARM, (ii) um FPGA, (iii) dois bancos de memórias RAM, (iv) um mecanismo de configuração (do inglês Configuration Engine, CE), (v) um gerenciador de configuração (do inglês Reconfiguration Manager, RM) responsável por interceptar as requisições das funcionalidades implementadas em hardware e desviá-las para o CE, e (vi) um barramento de comunicação.

Nikolov et al. [NIK08] apresentam uma plataforma composta pela ferramenta ESPAM (do inglês, Embedded System-level Platform Synthesis and Application Mapping) conjuntamente ao fluxo de projeto ilustrado na Figura 2 para automatizar a programação, implementação e o projeto de MPSoCs. Essa plataforma recebe como entrada especificações em alto nível de abstração e gera uma plataforma MPSoC especificada em VHDL sintetizável, além de realizar o mapeamento da aplicação C/C++ sobre os processadores da plataforma. Segundo os autores, esta metodologia permite ao projetista desenvolver um sistema multiprocessado em menos tempo, reduzindo portanto a lacuna de produtividade de projeto.

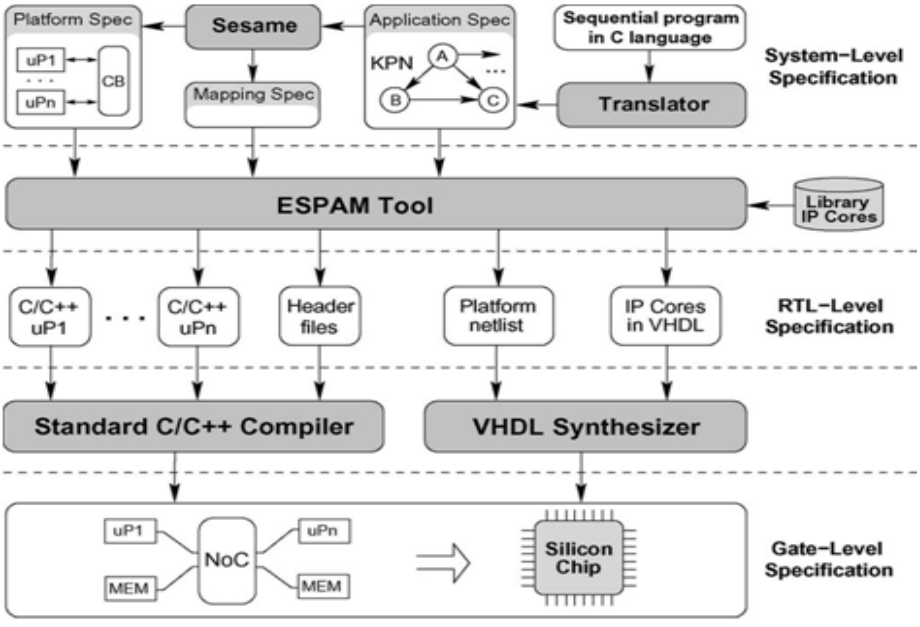

Figura 2 – Fluxo de projeto proposto por Nikolov et al. [NIK08].

A ferramenta ESPAM recebe três tipos de especificações como entrada:

- especificação de plataforma: descreve a topologia de uma plataforma

multiprocessada na qual a comunicação ocorre através de canais FIFO;

- especificação de aplicação: descreve uma aplicação como uma rede de processos Kahn (do inglês, Kahn Process Network, KPN), ou seja, uma rede de processos concorrentes comunicando-se via canais FIFO;
- especificação de mapeamento: descreve a relação entre todos os processos na especificação da aplicação e de todos os componentes na especificação da plataforma.

A plataforma foi validada através do desenvolvimento e programação de sistemas multiprocessados que executam as aplicações Sobel, DWT e M-JPEG considerando uma imagem de 128x128 pixels. Para as aplicações Sobel e o DWT foram utilizados três processadores Microblaze e para a M-JPEG quatro processadores, conectados por uma rede ponto a ponto. Os sistemas multiprocessados foram comparados com um sistema monoprocessado usando Microblaze ou PowerPC, indicando ganhos de desempenho da ordem de 2,2 vezes para Sobel, 2,1 para DWT e 3,75 vezes para M-JPEG.

2.1.2 Modelagem de MPSoCs com Simulink

Moreno et al. [MOR08] adotam três níveis de abstração para projetos de MPSoCs, quais sejam: arquitetura do sistema (do inglês System Architecture, SA), arquitetura virtual (do inglês Virtual Architecture, VA) e preciso em nível de transação (do inglês Transaction Accurate, TA). Dos três níveis, nos níveis VA e TA propõem-se a exploração da arquitetura do MPSoC, tendo sido ali que os autores descreveram modelos de NoC, os quais foram inseridos no fluxo de projeto MPSoC baseado no ambiente Simulink [THE08]. Os modelos de NoC propostos permitem avaliar o desempenho da arquitetura de comunicação através da variação de parâmetros durante a simulação do sistema e inserir definições abstratas de uma NoC no fluxo de projeto proposto originalmente em [POP07].

A tarefa básica para integrar os modelos NoC ao fluxo de projeto MPSoC é a modelagem do sistema. Esta fase é composta de três passos:

- modelagem da aplicação, que descreve a funcionalidade da aplicação alvo, gerando um modelo em alto nível de abstração a partir de Simulink;
- modelagem abstrata da infra-estrutura de comunicação, correspondendo à criação do modelo VA;
- refinamento da infra-estrutura de comunicação, gerando o modelo TA, permitindo detalhar a arquitetura local dos subsistemas.

A inserção dos modelos de NoCs no fluxo de projeto MPSoC permitiu a exploração da arquitetura da NoC e também estimar características relacionadas à implementação, além de permitir obter informações importantes para a tomada de decisão sobre o desenvolvimento da arquitetura de comunicação do sistema a partir dos modelos

propostos.

2.1.3 Modelagem de MPSoCs baseada em Java e UML

Ost et al. [OST08] apresentam uma técnica para avaliação de características de MPSoCs com comunicação baseada em NoCs, reduzindo o tempo de simulação a partir do aumento do nível de abstração em que o sistema é descrito, mantendo precisão a nível de ciclo de relógio. Os Autores propõem uma técnica de abstração dos dados do pacote, denominada PAT (do inglês Payload Abstraction Technique), que permite avaliar o desempenho sob o ponto de vista da latência e vazão, a partir do uso de simulação e métodos analíticos.

PAT permite obter resultados relativamente precisos mantendo alto nível de abstração, enquanto reduz o tempo total de simulação, devido ao uso de um número menor de eventos de comunicação obtida a partir da abstração dos dados do pacote. Neste trabalho, a NoC HERMES é modelada usando dois componentes: controle e buffer: controle é responsável pela arbitragem da porta de entrada e o roteamento pelo repasse de dados; buffer é uma estrutura de memória do tipo FIFO, modelada com o uso de máquinas de estado finitas, que controlam o fluxo de dados. Também permite avaliar latência e vazão aproximadamente 2.3 vezes mais rápido quando comparado ao modelo de referência, RTL. Questões relativas a congestionamento de tráfego não foram observadas, podendo gerar algum impacto sobre os resultados.

Indrusiak et al. [IND08] apresentam uma abordagem para dar suporte à exploração do espaço de projeto de arquiteturas de interconexão usando NoCs. Os Autores visam criar modelos abstratos de NoCs e definir níveis de abstração adequados acima do nível RTL. A idéia básica é partir de um modelo NoC RTL e gerar um modelo orientado a atores pela análise de interações, cujo o espaço de projeto será validado e explorado de forma a chegar a um modelo RTL otimizado em relação ao primeiro.

Dois aspectos foram considerados importantes na criação de NoCs abstratas: abstração estrutural e comportamental. A abstração estrutural compreende como os subsistemas são divididos e a abstração comportamental envolve como e quando os subsistemas atualizam seu estado interno e interagem concorrentemente com outros subsistemas.

Duas etapas básicas são desenvolvidas: modelagem dos atores e análise baseada em interconexão. A modelagem dos atores separa a funcionalidade, representada pelos atores, da comunicação entre componentes, formalizada como modelos de computação (do inglês, Models of Computation, MoC) concorrentes, que se comunicam através de tokens de dados, e são gerenciados por um diretor. A análise baseada em interações é usada para abstrair interconexões da NoC RTL, formalizadas através de diagramas de seqüência (DS) UML. Estes descrevem todas as transações ocorridas e definem a ordem parcial entre mensagens, desconsiderando tempo e

concorrência. Esta abordagem facilita a exploração do espaço de projeto, permitindo ao projetista analisar diferentes alternativas para interconexão do sistema de forma interativa, visando atender adequadamente requisitos como desempenho, área utilizada e consumo de potência.

2.1.4 Modelagem de MPSoCs baseada em SpecC

Dömer et al. [DÖM08] propõem um fluxo de projeto denominada SCE (do inglês, System-on-chip Environment) baseado no paradigma “especificação-exploração-refinamento” com suporte para plataformas MPSoC heterogêneas e que emprega a linguagem SpecC. O fluxo parte de uma especificação abstrata do sistema alvo, onde modelos em vários níveis de abstração são automaticamente gerados através de sucessivos passos de refinamento. A Figura 3apresenta o fluxo de projeto adotado pelo SCE.

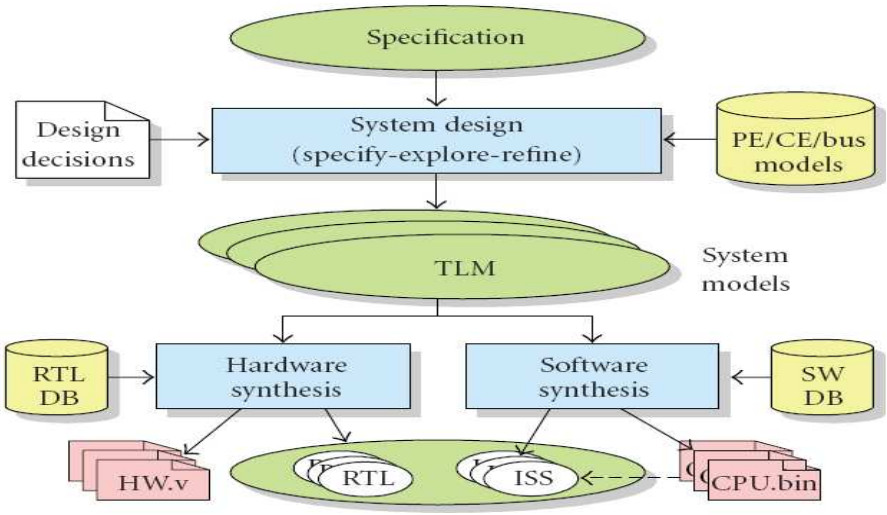

Figura 3 – Fluxo de projeto SCE [DÖM08].

A partir de uma especificação abstrata que descreve as funcionalidades do sistema, o projetista pode gerar modelos TL do projeto sucessivos em direção a níveis de abstração mais baixos, para tanto o fluxo usa modelos de componentes do repositório da ferramenta. Conforme o projetista gera novos modelos menos abstratos é possível tomar novas decisões, que serão integradas ao modelo recém gerado, a partir do modelo anterior. Após gerar os modelos, implementa-se componentes de hardware e software e cria-se descrições em Verilog no nível RTL.

3 TRABALHOS RELACIONADOS

Este Capítulo introduz os trabalhos relacionados que foram considerados e/ou utilizados para o desenvolvimento abordado nos Capítulos seguintes. O Capítulo desenvolve-se em quatro Seções. A primeira aborda o sistema empregado como ponto de partida do trabalho, a plataforma multiprocessada HeMPS [WOS07]. Em seguida, as Seções 3.2 e 3.3 discutem os dois principais componentes da plataforma HeMPS, a rede intrachip HERMES [MOR04] e o processador Plasma, uma implementação da arquitetura RISC MIPS-I [RHO07]. Finalmente, a Seção 3.4 discute a ferramenta de modelagem escolhida para uso neste trabalho, o ambiente CoCentric System Studio da Synopsys [SYN08a].

3.1 Sistema multiprocessado HeMPS

O sistema HeMPS [WOS07] é uma infra-estrutura de hardware e software que permite a implementação de MPSoCs. HeMPS usa o processador Plasma como elemento de processamento do sistema multiprocessado e baseia sua comunicação na infra-estrutura de comunicação intrachip HERMES. O sistema HeMPS emprega multiprocessamento homogêneo e possui uma arquitetura de sistema baseada no paradigma mestre-escravo.

3.1.1 Componentes de hardware

A infra-estrutura de hardware presente no sistema HeMPS é composta basicamente por três módulos:

- elementos de processamento: correspondem aos núcleos IPs dedicados ao processamento, que correspondem aos processadores Plasma. Existem duas classes de processadores: mestre (master) e escravo (slave). O mestre basicamente é responsável por distribuir as tarefas a serem executadas nos processadores escravo e gerenciar informações a respeito destas tarefas, existindo apenas um processador desta classe no sistema. Os processadores escravo são responsáveis por executar tarefas, podendo existir mais de uma tarefa em cada processador, pois estes executam um núcleo de sistema operacional que implementa a operação multitarefa. O mesmo núcleo implementa o suporte à comunicação entre tarefas, locais e/ou remotas;
- elementos de comunicação e roteamento: corresponde aos roteadores da rede HERMES, que habilitam a comunicação entre os vários elementos de processamento;
- canais de comunicação: representam aos meios que interligam os diversos roteadores presentes na rede HERMES.

Uma instância de sistema que mostra a estrutura básica de hardware da HeMPS aparece na Figura 4.

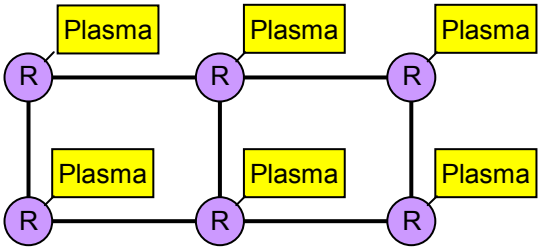

Figura 4 – Componentes de hardware básicos do sistema HeMPS: elementos de processamento, roteadores e canais de comunicação.

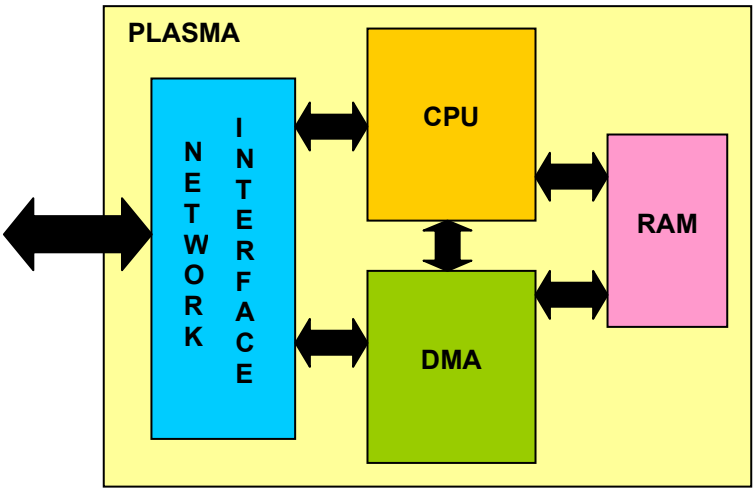

Figura 5 – Diagrama de blocos do Plasma, mostrando seus quatro módulos principais e sua interconexão [WOS07].

Na Figura 5 podem ser observados os quatro componentes básicos do módulo:

- CPU: denominado MLite, é um processador baseado na arquitetura RISC MIPS-I de 32 bits, discutido na Seção 3.3;
- RAM: corresponde à memória principal local do processador. Nesta memória são armazenados tanto o código das tarefas locais como os seus dados;
- Network Interface: trata-se da interface de rede, responsável por permitir a comunicação bidirecional entre processador e uma rede externa a este. Foi desenvolvido para dar suporte à comunicação do Plasma com a rede HERMES e vice-versa;
- DMA: é o módulo de acesso direto à memória (do inglês, direct memory access). O DMA permite aumentar o desempenho no sistema, uma vez que gerencia a transferência de grandes blocos de dados entre memória e rede de forma independente do processador.

Com relação ao projeto Plasma original o módulo Plasma usado no sistema HeMPS sofreu alterações a fim de adequar este projeto às necessidades do sistema. Entre as mudanças destaca-se a inserção do módulo DMA, a inserção da interface de

rede NI e do repositório de tarefas Mem Ext.

Mais detalhes da implementação do hardware do sistema HeMPS podem ser encontrados em [WOS07], sobretudo no Capítulo 4 desta dissertação.

A Figura 6 apresenta uma versão do módulo Plasma com maior nível de detalhes.

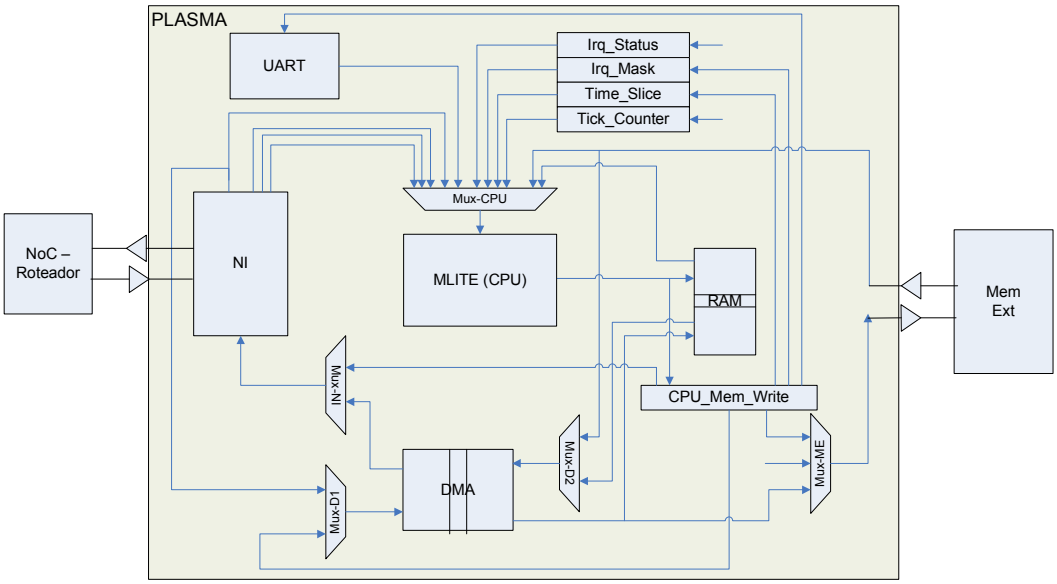

Figura 6 – Detalhamento do caminho de dados do módulo Plasma no sistema HeMPS.

3.1.2 Componentes de software

A infra-estrutura de software existente no sistema HeMPS compreende dois grupos básicos de programas: microkernel e tarefas. O microkernel por sua vez possui duas versões, uma destinada à execução no processador mestre e outra que executa nos processadores escravos. As tarefas correspondem às aplicações a serem executadas em cada processador escravo, visto que na versão da HeMPS usada aqui o mestre não executa tarefas, apenas o microkernel.

Sinteticamente, a versão mestre do microkernel realiza as seguintes tarefas: (i) executa o processo de inicialização (boot) onde todas as estruturas de dados necessárias são criadas e atribuídas; (ii) realiza a distribuição das tarefas a serem executadas de forma estática ou dinâmica; (iii) controla o escalonamento das tarefas em execução; (iv) implementa a comunicação entre os processadores, tanto no nível do microkernel (sistema) quanto no das tarefas; (v) operacionaliza o tratamento de interrupções tanto de software como de hardware.

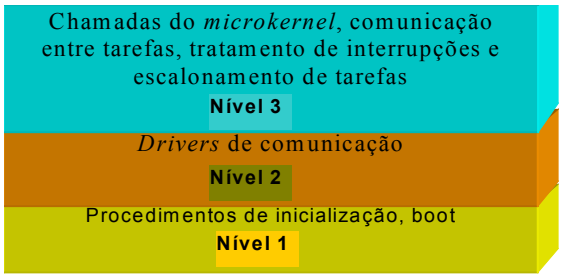

Figura 7 – Estrutura em níveis no microkernel da plataforma HeMPS.

A Figura 7 introduz a estrutura em camadas composta de três níveis adotada na arquitetura do microkernel juntamente à função executada por cada nível.

3.2 Infra-estrutura de comunicação intrachip HERMES

3.2.1 Conceitos básicos

A infra-estrutura de comunicação intrachip HERMES [MOR04] foi desenvolvida no grupo de pesquisa do autor, o Grupo de Apoio ao Projeto de Hardware (GAPH) [GAP08]. O objetivo principal foi atingir largura de banda superior, quando comparada a arquiteturas de comunicação intrachip tradicionais, baseadas em barramento.

Um conjunto de modelos VHDL parametrizáveis foi desenvolvido para dar suporte à infra-estrutura HERMES. Através do emprego da ferramenta ATLAS [GAP08] é possível implementar diversas arquiteturas de comunicação a partir destes modelos. A infra-estrutura de comunicação gerada a partir dos modelos utiliza chaveamento por pacotes onde as mensagens a serem transmitidas são encapsuladas em pacotes roteados de forma individual e continua entre os nodos da rede, sem o estabelecimento prévio de um caminho. Este mecanismo de comunicação requer o uso de um modo de roteamento para definir como os pacotes devem se mover entre os roteadores. O modo de roteamento adotado para a comunicação é o wormhole. O roteamento de pacotes é implementado a partir do algoritmo XY distribuído, mas existem outras estratégias disponíveis na ferramenta ATLAS. O pacote é dividido em flits, unidades de controle de fluxo utilizadas para sincronizar a transferência de dados. Apenas os dois flits iniciais possuem informações de roteamento, os flits restantes compõem o conteúdo da mensagem e seguem o caminho determinado pelo algoritmo de roteamento escolhido.

A NoC HERMES possui uma topologia regular malha 2D direta, que habilita a implementação do algoritmo de roteamento escolhido, a distribuição dos nodos de comunicação e a inserção de núcleos IP. Uma instância de rede HERMES ilustrando a topologia pode ser vista na Figura 8.

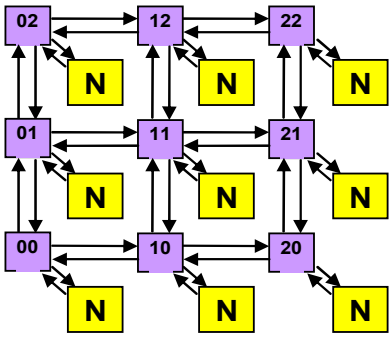

Figura 8 – Uma instância de uma NoC HERMES 3x3. Retângulos numerados representam os roteadores existentes e os valores identificam sua posição na rede, segundo a distribuição cartesiana XY. Assim, 21 indica o roteador localizado na terceira coluna (abscissas) e segunda linha (ordenadas). Os retângulos N representam os núcleos IP conectados a cada roteador.

O componente básico da infra-estrutura HERMES é o roteador, com estrutura apresentada na Figura 9. O roteador HERMES possui cinco portas bidirecionais que se conectam com até quatro outros roteadores vizinhos e com um núcleo IP. O roteador usa armazenamento de dados na entrada e possui lógica de arbitragem interna. O roteamento é determinístico, distribuído e pode usar caminho mínimo e não mínimo entre origem e destino. Uma lógica de controle centralizada (CL na Figura) implementa arbitragem de acesso à lógica de roteamento e o próprio roteamento.

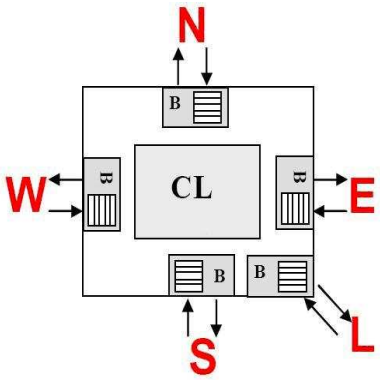

Figura 9 – Estrutura básica do roteador [MOR04].

O roteador HERMES contém três estruturas fundamentais:

- cinco portas de conexão (N, S, E, W e L): cada porta é composta por dois canais, um de entrada e outro de saída, e um elemento de armazenamento na entrada. São responsáveis pelo controle do recebimento e envio de dados que chegam ao roteador, tanto através de roteadores vizinhos como do núcleo IP local;
- lógica de controle (CL): realiza a arbitragem entre pedidos de roteamento concorrentes e controla o roteamento de dados entre as portas, podendo ocorrer transferências em paralelo entre mais de um par de portas de entrada/saída distintas. Cada transferência de dados passa pela lógica de controle;
- armazenamento (B): memória local para armazenamento temporário de

informações de entrada (buffer), o que permite um melhor desempenho da comunicação devido a possíveis bloqueios no fluxo de comunicação.

O formato do pacotes da rede HERMES é apresentado na Tabela 1.

Tabela 1 – Seqüência de envio de flits na rede HERMES. O primeiro flit contém o endereço de destino (target); o segundo carrega o tamanho da área de dados (size); a partir do terceiro flit são enviados os dados (payload).

Seqüência de envio	0	1	2	...	n-1	n
Conteúdo	Destino	Tamanho	Dado 1	...	Dado tam-1	Dado tam

Estes pacotes são compostos de duas partes: cabeçalho (header) e carga útil (payload). Estes por sua vez são transmitidos pela rede em unidades denominadas flits. Existem três diferentes tipos de flits: destino (target), tamanho (size) e carga útil (payload), os quais são enviados nesta mesma seqüência conforme a Tabela.

3.2.2 Modelo RTL sintetizável

A NoC HERMES foi desenvolvida originalmente via descrições RTL [MOR04]. Ela foi descrita em VHDL e validada por simulação funcional, sendo posteriormente sintetizada com sucesso em FPGAs. O protocolo de comunicação foi baseado no modelo de referência OSI, usando como de hábito apenas um subconjunto das camadas deste modelo: físico, enlace, rede e transporte. Basicamente cada camada provê, respectivamente, as seguintes funcionalidades:

- definir o mecanismo de comunicação entre os roteadores;
- estabelecer o protocolo handshake sobre a camada física, de forma a garantir o fluxo de controle dos dados enviados e recebidos;
- realizar a segmentação e remontagem dos dados em flits, ou seja, implementar a técnica de comunicação de chaveamento por pacotes;
- estabelecer a comunicação fim a fim entre origem e destino na rede.

A NoC HERMES implementa roteadores compostos por cinco portas de comunicação bidirecionais que podem estar conectadas a até cinco dispositivos periféricos, dependendo de qual posição ocupa na topologia. O caso em que todas as portas são usadas caracteriza-se quando o roteador se encontra inserido no interior da rede, ou seja, não ocupa nenhuma posição de borda quando algumas portas estarão desconectadas. Por exemplo, na Figura 8 o roteador 11 possui todas as portas conectadas enquanto o roteador 22 possui duas portas não utilizadas (N e E).

A Figura 10 apresenta a janela inicial da ferramenta gráfica para geração automática da infra-estrutura de rede HERMES denominada ATLAS [GAP08], que comporta quatro setores: (i) geração da NoC a partir da ferramenta NoCGen, (ii) geração do tráfego, (iii) simulação e (iv) avaliação. Estes quatro setores são alcançados a partir da janela inicial da ferramenta, conforme pode ser observado na Figura 10. A ferramenta

aceita em cada setor certa quantidade de parametrização, permitindo definir características como: largura de banda, profundidade de armazenamento de dados na entrada e número de canais virtuais. O setor de geração de tráfego produz dados para serem injetados na rede, o que permite avaliar diferentes cenários de tráfego. Na simulação, os dados gerados são injetados na NoC e a funcionalidade de comunicação é simulada entre os roteadores, conforme o cenário previamente definido. No último setor, avaliação, é possível analisar diversos aspectos de desempenho a partir de gráficos, tabelas, mapas e relatórios gerados automaticamente pela ferramenta, auxiliando a compreensão dos resultados gerados.

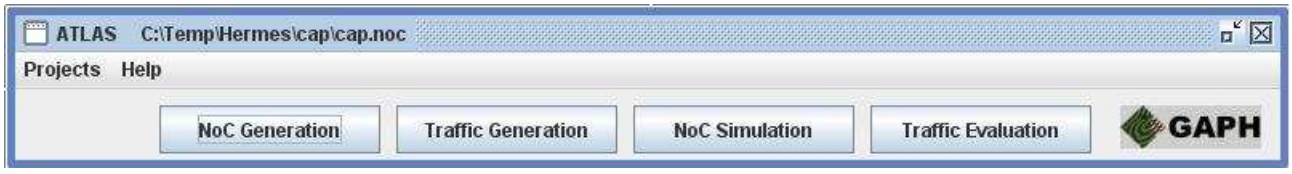

Figura 10 – Janela inicial do ambiente ATLAS.

3.2.3 Modelo abstrato simulável

Outro trabalho, desenvolvido no grupo do Autor foi a implementação de um modelo em nível de abstração superior em relação à HERMES RTL, o chamado nível de transação (TL). Esta versão foi proposta por Moreno em [MOR04a] e utilizou a linguagem SystemC, tendo como objetivo principal disponibilizar uma representação comportamental mais abstrata da rede HERMES.

Neste trabalho, consideraram-se modelos abstratos aqueles desenvolvidos, observando-se menor nível de detalhamento relativo ao existente no nível de transferência entre registradores (RTL).

O modelo abstrato desenvolvido permite a detecção de possíveis erros de projetos em estágios iniciais de desenvolvimento além de prover modelos executáveis completos para a validação da comunicação em SoCs que fazem uso de NoCs como infra-estrutura básica para comunicação.

As características básicas apresentadas na Seção 3.2.1 foram mantidas. Assim, a compatibilidade entre versões pode ser mantida em vários aspectos. Os quatro níveis da pilha de protocolos foram mantidos, bem como o tipo de chaveamento (por pacotes) o modo de chaveamento (wormhole) e o algoritmo de roteamento (distribuído, XY).

O roteador contém os mesmos elementos da definição original: portas bidirecionais de entrada e saída, área de armazenamento e lógica de controle que constitui um canal de comunicação entre as portas. Embora esta versão mantenha os componentes básicos da especificação RTL, seguindo a especificação geral definida para este nível, algumas alterações foram realizadas, incluindo que:

- as interfaces foram descritas de forma abstrata; portanto detalhes como sinais

específicos nas interfaces e o sinal de relógio foram abstraídos;

- o comportamento interno, originalmente implementado através de máquinas de estado finitas (em inglês, finite state machines, FSMs), foi definido a partir de uma descrição algorítmica por meio de chamadas de métodos de classe em SystemC, que realizam toda a funcionalidade necessária.

A Figura 11 detalha a estrutura interna do roteador implementada neste modelo abstrato.

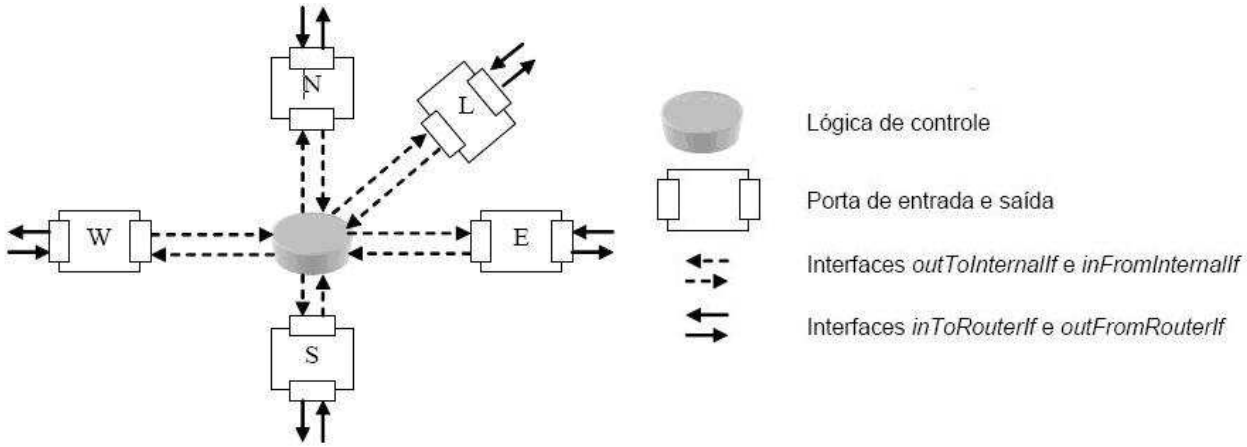

Figura 11 – Estrutura interna do roteador para a rede HERMES TL [MOR04a].

Mais detalhes podem ser obtidos em [MOR04a] e no Capítulo 4 desta dissertação.

3.3 Arquitetura MIPS-I de processadores de propósito geral

No início dos anos 80, as Universidades de Berkeley e Stanford eram alguns dos centos acadêmicos que pesquisavam formas de produzir processadores que atingissem alto desempenho a partir da execução de instruções simples, porém a freqüências de relógio elevadas. O paradigma resultante tornou-se conhecido pelo acrônimo RISC (do inglês, Reduced Instruction Set Computer). Um dos resultados deste trabalho foi a arquitetura denominada MIPS (abreviatura do inglês Microprocessor without Interlocked Pipeline Stages).

3.3.1 A arquitetura MIPS-I

Com o intuito de permitir um melhor entendimento da arquitetura MIPS, apresenta-se a seguir um breve relato da evolução e das características básicas dos principais processadores produzidos com base nesta arquitetura. As características apresentadas não exaurem todo o desenvolvimento desta família de arquiteturas. Foram intencionalmente omitidos muitos dos modelos mais recentes, bem como características mais avançadas, uma vez que não são necessários aos objetivos pretendidos e raramente são incluídas em versões embarcadas deste processador.

O conjunto de instruções MIPS foi desenvolvido inicialmente como um projeto acadêmico na Universidade de Stanford através de um grupo de pesquisadores liderado por professor John Hennessy a partir de 1981. Este projeto se tornou um dos principais precursores da tecnologia RISC. A arquitetura do conjunto de instruções (em inglês instruction set architecture, ISA) MIPS-I é regular, todas as instruções possuem o mesmo tamanho. Conforme [SWE06], esta característica permitiu que seus conceitos fossem assimilados mais facilmente se comparado a arquiteturas irregulares. Em 1984, foi fundada a empresa MIPS Computer Systems, Inc. cujo objetivo foi comercializar a tecnologia desenvolvida no meio acadêmico.

O primeiro processador MIPS comercial lançado foi o modelo R2000 lançado em 1985. Esta versão contou com a adição de uma unidade de gerenciamento de memória (em inglês, Memory Management Unit, MMU). Em 1987 tornou-se disponível o co-processador aritmético R2010, disponibilizando capacidade superior de cálculos em relação à versão inicial. Em 1988 foi lançada uma nova versão de processador, o R3000. O modelo incorporava duas memórias cache de 32KB, uma para dados e outra para instruções. Logo se tornou disponível o co-processador aritmético para esta versão, o R3010. Do ponto de vista dos programadores este processador era quase idêntico ao anterior, permitindo que o novo modelo pudesse ser rapidamente utilizado. Entre as versões derivadas do R3000, seguiram os modelos R3400 e R3500 produzidos pelas empresas PACEMIPS e IDT, respectivamente. Estas versões incluíam o co-processador R3010 no mesmo chip. Outro modelo, o R3900 introduzido pela Toshiba, tornou-se um dos primeiros processadores destinados ao uso em handhelds baseados no Windows CE. O Mongoose, desenvolvido pela empresa Synova, Inc. para o Centro de Vôos Espaciais Goddard da NASA, disponibilizou uma versão expandida do R3000, com o R3010 já incorporado. Esta versão foi usada em aplicações espaciais empregando métodos de fabricação resistentes à radiação como forma de tornar seus componentes eletrônicos utilizáveis fora da atmosfera terrestre.

Em 1990 a empresa Bipolar Integrated Technology (BIT, Inc) produziu o modelo R6000, que incluía suporte para uso do ECL (do inglês Extensible Computer Language), linguagem de programação extensível desenvolvida na Universidade de Harvard. Possuía uma arquitetura de cache com tecnologia Translation Lookaside Buffer (TLB) diferente dos outros modelos da arquitetura MIPS, tendo sido pouco usado.

Em 1991 foi lançada a série R4000 que ampliou o conjunto de instruções para uma arquitetura de 64 bits e a inserção de uma unidade de tratamento de ponto flutuante (do inglês Floating Point Unit, FPU) dentro do mesmo chip, porém reduzindo as memórias cache de dados e instruções para 8KB cada uma, a fim de atingir maior velocidade de relógio. Neste mesmo ano a Silicon Graphics, Inc. (SGI) adquiriu a MIPS Computer Systems, criando a MIPS Technology, Inc. (MTI), uma divisão da SGI existente até hoje.

No início dos anos 90 alguns desenvolvedores de sistemas embarcados, como a

Sony, a SGI e a Cisco começam a utilizar processadores MIPS em seus sistemas, tais como dispositivos de rede, vídeo games, PDAs, modems a cabo e DSL, TV digital e roteadores. A Quantum Effect Devices (QED) desenvolveu o modelo R4650 usado em seu set-top box WebTV. Os modelos R4600 e R4700 foram usados em roteadores Cisco e nas estações de trabalho Indy de baixo custo da SGI, que posteriormente passaram a usar o modelo R5000, uma vez que sua FPU era mais eficiente que a existente no R4000.

Em 1994 o R4400 ampliou a memória cache para 16KB e suporte para maior quantidade memória cache nível 2. Neste mesmo ano foi lançado o R8000, primeira versão superescalar do processador MIPS, executando duas instruções inteiras ou de ponto flutuante e duas instruções para operações em memória por ciclo de relógio. Em 1995 foi lançado o modelo R10000, mais rápido que o R8000, dispondo de memória cache de 32KB, tecnologia superescalar e principalmente dispondo da capacidade de execução de instruções fora de ordem. O R12000 e R14000, lançados respectivamente em 1998 e 2001, basearam-se no R10000, caracterizando-se por menor uso de área de silício e freqüências de relógio mais elevadas. O último modelo tem suporte para tecnologia de memórias DDR-SDRAM.

Por fim, em 1999 a arquitetura MIPS foi licenciada sob dois grupos básicos: (i) MIPS32, baseado na ISA MIPS-II com características do MIPS-III, IV e V e (ii) MIPS64 baseado no MIPS V. A partir daí estas duas denominações passaram a caracterizar os futuros de processadores MIPS, desenvolvidos pela MTI ou sob licença desta.

3.3.1.1 Arquitetura do conjunto de instruções MIPS

As versões iniciais de processadores MIPS usavam arquitetura de 32 bits, como o R2000 e o R3000. Versões posteriores passaram a empregar arquitetura de 64 bits com a introdução do R4000. Durante a evolução da família de processadores diversas revisões do conjunto de instruções (ISA) MIPS foram realizadas, incluindo MIPS-I, MIPS-II, MIPS-III, MIPS-IV, MIPS-V, MIPS32 e MIPS64. As duas últimas revisões são as especificações atualmente disponíveis para 32 e 64 bits respectivamente. Todas as revisões são retroativamente compatíveis [KAN91] [SWE06].

Os primeiros quatro conjuntos de instruções foram introduzidos nos seguintes processadores:

- MIPS-I: modelos R2000 e R3000;
- MIPS-II: modelo R6000;
- MIPS-III: modelo R4000;
- MIPS-IV: modelos R8000 e R10000.

A primeira arquitetura de conjunto de instruções, denominada MIPS-I, foi implementada nos processadores R2000 e R3000. O conjunto de instruções MIPS-II foi

introduzido a partir do modelo R6000, adicionando instruções como load linked e store conditional, usadas para realizar operações do tipo read-modify-write de forma atômica e suporte a operações load/store de 64 bits. A revisão MIPS-III foi introduzida em 1991 com a arquitetura R4000. Disponibilizava instruções de 64 bits com inteiros (e registradores de mesma largura de palavra) e uma instrução de raiz quadrada em ponto flutuante. A ISA MIPS-IV, implementada inicialmente no modelo R8000, adicionou instruções de transferência condicionais e uma instrução de raiz quadrada inversa em ponto flutuante. A arquitetura MIPS-V foi introduzido em 1994 pela SGI, porém nunca foi realmente utilizada por nenhum processador MIPS.

MIPS32 e MIPS64 foram definições criadas pela SGI em 1999 com o objetivo de melhorar a coerências das definições da arquitetura, além de inserir novas características, principalmente a de tornar o desenvolvimento mais fácil e fornecer melhor suporte à criação de ferramentas de desenvolvimento em ambas as plataformas de 32 e 64 bits. MIPS32 é um subconjunto de 32 bits da ISA MIPS64, sendo esta última uma extensão da MIPS-V. MIPS32 e MIPS64 versão 2.0 são extensões opcionais DSP-ASE (do inglês Application Specific Extensions) usadas para aceleração de processamento multimídia como de áudio e de vídeo.

A opção pelo uso da ISA baseada em MIPS-I se deu pela decisão de projeto adotada para o sistema HeMPS, a qual selecionou o projeto Plasma como seu elemento de processamento.

3.3.1.2 Características da arquitetura de processadores baseados na ISA MIPS-I.

Os conceitos básicos relacionados à arquitetura MIPS-I podem ser resumidos pelas seguintes características:

- execução implementada sobre um pipeline com cinco estágios;
- conjunto de instruções regular, onde todas as instruções possuem comprimento fixo (32 bits);
- instruções lógicas e aritméticas compostas de três operandos: fonte, alvo e destino;
- 32 registradores de propósito geral, cada um com 32 bits;
- nenhuma instrução complexa, como gerenciamento de pilha ou operações sobre cadeias de caracteres;
- possibilidade de dar suporte nativo a até quatro co-processadores opcionais, destinados ao gerenciamento do sistema, operações com ponto flutuante e funcionalidades específicas;
- instruções de acesso à memória exclusivas: load e store, tanto para memória principal como para dispositivos periféricos, o que força o uso do conceito de entrada e saída mapeada em memória;

- espaço de endereçamento contínuo de 4GB para acesso à memória, alinhado à palavra de 32 bits, com possibilidade de acesso à palavra (32bits), meia palavra (16 bits) ou byte (8 bits);
- mapeamento de endereços virtuais para endereços reais realizado por uma MMU;
- código em linguagem de montagem otimizado pelo compilador, em substituição à intervenção manual ou otimização via hardware;
- a estrutura de hardware não verifica dependências de instruções. Porém um conjunto de ferramentas de software considera as características do hardware, o que permite uma gerência eficaz na geração de código. Por exemplo, a ISA MIPS-I dá suporte ao uso de implementações do tipo delayed branch, onde a próxima instrução sempre é executada antes de um desvio condicional. Assim, a instrução que segue pode ser uma instrução sem efeito (NOP), inserida automaticamente ou a instrução anterior deslocada para após a instrução de desvio.

O conjunto de registradores da arquitetura MIPS-I é constituído por três grupos de registradores de 32 bits distintos: (i) 32 para propósito geral, numerados de 0 a 31, (ii) dois para operações de multiplicação e divisão (HI e LO) e (iii) um contador de programa (PC). Todos os registradores de propósito geral podem ser usados como fonte ou destino de dados para instruções lógico-aritméticas, acesso à memória e controle de fluxo. A única exceção é o registrador 0 (zero), não alterável, que possui seu valor fixado em zero pelo hardware. Portanto, operações de leitura deste registrador sempre retornam valor zero e as de escrita são ignoradas, sendo o valor escrito descartado.

O conjunto de instruções MIPS-I divide-se em três formatos básicos, conforme descrito na Tabela 2:

Tabela 2 – Divisão das instruções MIPS quanto ao seu formato.

Posição		***Campos da palavra de 32 bits***					
		31 26	**25 21**	**20 16**	**15 11**	**10 6**	**5 0**
Formato	**R**	**opc**	**rs**	**rt**	**rd**	**shamt**	**funct**
	I	**opc**	**rs**	**rt**	**imm-16**		
	J	**opc**	**addr-26**				

Nesta Tabela as siglas dos campos correspondem a:

- opc: código de operação;
- rs: endereço do registrador origem;
- rt: endereço do registrador alvo (origem ou destino, conforme instrução específica);
- rd: endereço do registrador destino;
- shamt: quantidade de deslocamento;

- funct: função;
- imm-16: constante imediata de 16 bits;
- addr-26: endereço parcial de 26 bits.

As características de cada formato podem ser resumidas como segue.

- instruções com formato R (register): inclui instruções lógicas e aritméticas com três registradores, inclusive multiplicação e divisão, instruções de deslocamento de bits, desvio baseados em registrador e instruções de exceções syscall e break. O campo funct atua como um campo adicional de 6 bits para identificação da instrução. O campo shamt é usado para indicar a quantidade de bits a ser deslocada em instruções de deslocamento de bits. Os registradores rs e rt são usados como origem de dados e rd como destino do resultado da operação;
- instruções com formato I (immediate): incluem instruções de acesso à memória (alinhado ou não), instruções lógicas e aritméticas que fazem uso de constantes imediatas e instruções de desvio condicional (branches). O registrador rs é usado como origem de dados e o rt como destino do resultado da operação. O campo imm-16 contém um valor imediato de 16 bits usado de uma de duas formas: como valor literal (positivo ou negativo) em instruções lógicas e aritméticas ou como um inteiro com sinal (entre -32678 a +32467) indicando o deslocamento relativo a um registrador (modo de endereçamento base-deslocamento);
- instruções com formato J (jump): inclui instruções de salto que usam modo de endereçamento pseudo-direto. O campo addr-26 contém uma constante (imediata) de 26 bits usada no cálculo do endereço absoluto alvo do desvio. Este valor é acrescentado de 2 bits à esquerda, equivalente a uma multiplicação por 4 devido ao uso de alinhamento de todo programa em endereço múltiplo de quatro. Isto gera um endereço de 28 bits, que é então concatenando com os 4 bits superiores do PC, ou seja, addr-26 indica o deslocamento em quantidade de palavras não de bytes dentro da página de memória (de 256 Mbytes) onde se encontra a instrução seguinte à instrução de desvio.

Além da classificação das instruções MIPS-I quanto ao formato, também se pode dividir o conjunto de instruções quanto à funcionalidade das instruções. Existem três classes de instruções, conforme este critério:

- classe Padrão (Standard): caracterizada pelas instruções que possuem o campo opc diferente de 0 ou 1 (zero ou um). Nesta classe enquadram-se as instruções de acesso à memória (alinhado ou não), instruções lógicas e aritméticas que fazem uso de constantes (imediatas), instruções de salto que

usam modo de endereçamento pseudo-direto e instruções de desvio condicional que usam dois registradores, além das instruções blez e bgtz;

- classe Especial (Special): caracterizada pelas instruções que possuem o valor 0 (zero) no campo opc. Inclui instruções lógicas e aritméticas com três registradores inclusive multiplicação e divisão, instruções de deslocamento de bits, instruções de salto baseadas em registrador e as instruções para tratamento de exceções syscall e break. O campo funct atua como campo complementar ao opc, permitindo selecionar a operação desejada;
- classe Registrador-imediato (RegImm): caracterizada pelas instruções que possuem o valor 1 (um) no campo opc. Inclui instruções de desvios condicionais (branches) que comparam valores em relação a 0 (zero), exceto as instruções blez e bgtz. O campo rt define qual o tipo de comparação a ser realizada, portanto é usado em conjunto com o campo opc para decodificar a instrução.

Embora os 32 registradores de propósito geral possam ser usados indistintamente para qualquer objetivo (com exceção eventual do 0), existe uma convenção para uso destes registradores em programação, com o objetivo de padronizar o desenvolvimento de software básico, como montadores, compiladores e carregadores. Esta convenção é mostrada na Tabela 3. Estas convenções obviamente não implicam a existência de qualquer estrutura no hardware que as dê suporte a elas.

Tabela 3 – Descrição das convenções de uso dos registradores de propósito geral do MIPS.

Número	Nome	Descrição do uso
0	zero	Valor zero, sempre retorna ou armazena o valor 0x00000000
1	at	Reservado para uso do montador (*assembler temporary*)
2-3	v0-v1	Valores retornados por sub-rotinas (*returned value*)
4-7	a0-a3	Primeiros 4 argumentos passados a sub-rotinas (*arguments*)
8-15	t0-t7	Uso livre para sub-rotinas, não necessário salvá-los (*temporaries*)
24-25	t8-t9	Uso livre para sub-rotinas não necessário salvá-los (*temporaries*)
16-23	s0-s7	Valores preservados entre chamadas de sub-rotinas, devem ser restaurados após o retorno (*saved*)
26-27	k0-k1	Reservado para rotinas de interrupção/captura, usados pelo sistema operacional (*kernel*)
28	gp	Usado para acesso à variáveis estáticas ou "extern" (*global pointer*)
28	sp	Ponteiro para área de pilha (*stack pointer*)
30	fp|s8	Ponteiro para área de *frame* (*frame pointer*) ou registrador adicional para valor preservado (*saved*)
31	ra	Usado como endereço de retorno de sub-rotina (*return address*)

A Tabela 4 reapresenta a convenção para os registradores de propósito geral sob outro formato.

Tabela 4 – Distribuição dos registradores de propósito geral.

Posição	Distribuição dos registradores								Posição
00	zero	at	v0	v1	a0	a1	a2	a3	07
08	t0	t1	t2	t3	t4	t5	t6	t7	15
16	s0	s1	s2	s3	s4	s5	s6	s7	23
24	t8	t9	k0	k1	gp	sp	fp|s8	ra	31

A arquitetura pipeline de cinco estágios pressuposta como natural para a implementação do hardware de um processador MIPS inclui:

- IF – estágio de busca (instruction fetch): busca a próxima instrução a ser executada;
- ID – estágio de decodificação (instruction decode): realiza a decodificação da instrução e a leitura dos operandos fonte do banco de registradores;
- EX – estágio de execução (execution): realiza a execução da instrução ou o cálculo de endereço,quando aplicável;
- MEM – estágio de acesso à memória (memory access): realiza o acesso à memória externa;
- WB – estágio de escrita do resultado (write back): escrita do resultado no banco de registradores.

Para efeito ilustrativo, a Figura 12 apresenta um diagrama esquemático simplificado de um pipeline típico do MIPS, conforme empregado para dispositivos como os MIPS R2000 e R3000.

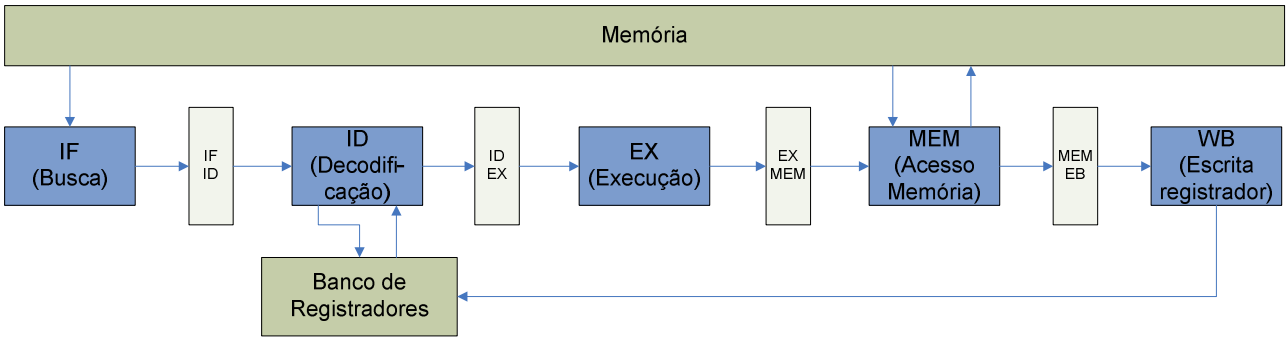

Figura 12 – Diagrama de blocos típico de um pipeline para processador MIPS usando ISA MIPS-I.

3.3.2 Implementação do processador Plasma

O projeto Plasma está disponível no Opencores [RHO07], tendo sido desenvolvido originalmente por Steve Rhoads, baseado nas informações disponíveis no livro de Kane e Heinrich [KAN91]. Este projeto consiste de uma implementação compatível, embora não completamente, com o modelo R3000 do MIPS, juntamente com uma infra-estrutura de processamento contendo periféricos de apoio, sendo o processador denominado MLite.

O processador MLite executa todas as instruções da arquitetura do conjunto de instruções (ISA) MIPS-I, com exceção das instruções para cálculo de ponto flutuante, uma vez que o módulo co-processador aritmético não está presente no projeto. As instruções para acesso não alinhado à memória não estão disponíveis no MLite, devido a restrições de uso, pois foram patenteadas pela MIPS Computer Systems em 1986, patente cuja

validade é de 20 anos.

O Plasma foi descrito em linguagem de descrição de hardware VHDL contendo os seguintes periféricos, considerando a versão 3.5, a mais recente até o momento da escrita deste documento:

- porta serial bidirecional;
- controlador de interrupções;
- temporizador de hardware;
- controlador DDR SDRAM;
- MAC Ethernet;
- interface Flash.

Este projeto foi sintetizado em dispositivos FPGA Xilinx Spartan-3 XC3S200, Spartan-3E XC3S500 e Altera EP20K200EFC484-2X.

O processador MLite permite executar código nativo da arquitetura MIPS-I gerado a partir de compiladores como GNU-GCC, tanto em ambiente Linux como MS-Windows.

O diagrama de blocos do processador é apresentado na Figura 13, onde podem ser vistos seus principais blocos funcionais:

- unidades lógicas e aritméticas (Alu, Mult e Shifter);
- unidade de controle/decodificação (Control);
- controlador de memória (Mem_ctrl);
- banco de registradores (Reg_bank);
- multiplexador (Bus_mux);
- contador de programa (PC_next).

O processador MLite foi implementado com um pipeline diferente do padrão adotado na arquitetura MIPS, e possui duas formas de execução: uma com dois e outra com três estágios de pipeline, configurável no código VHDL. Além dos estágios citados, existe outro adicional, usado para operações de leitura e escrita em memória. Esta implementação difere da definição original do MIPS, uma vez que a organização de memória segue o modelo Von Neumann e não Harvard, como a maioria das implementações do MIPS.

O projeto disponibiliza também um simulador do conjunto de instruções (em inglês Instruction Set Simulator, ISS) escrito em linguagem C. Este simulador permite executar o conjunto de instruções da arquitetura MIPS-I, exceto as já citadas instruções de acesso não alinhado à memória e instruções para cálculo com ponto flutuante. O simulador é invocado via linha de comando e permite executar programas em formato core dump, ou seja, uma imagem binária do código objeto.

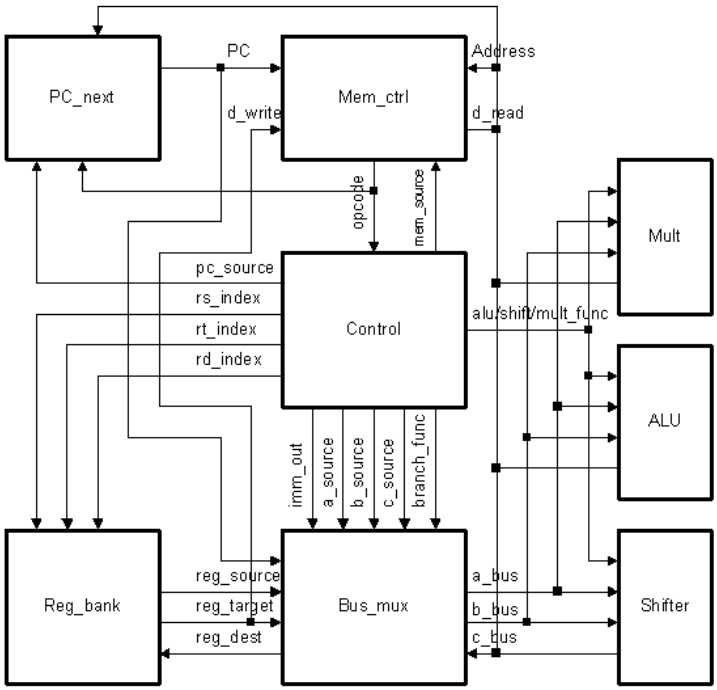

Figura 13 – Diagrama de blocos do processador MLite do projeto PLASMA [RHO07].

3.4 Synopsys System Studio – ferramenta de modelagem de sistemas

Monografia anterior do Autor [PET07], tem como tema um estudo da ferramenta comercial System Studio da empresa Synopsys Inc., conhecida anteriormente como CoCentric System Studio (CCSS). Conforme o fabricante introduz em seu sítio na internet [SYN08a], esta ferramenta se destina ao desenvolvimento e análise de projetos eletrônicos descritos a nível de sistema (do inglês Electronic System Level, ESL), baseado em modelos e usados na elaboraçào de produtos como telefones sem fio e móveis, modem a cabo e DSL e codecs multimídia. Segundo a companhia, mais de 50% dos telefones móveis produzidos utilizam algoritmos desenvolvidos no ambiente System Studio. A ferramenta permite a captura, modelagem e simulação de algoritmos, tanto para software como para hardware em múltiplos níveis de abstração, que podem coexistir e interagir em um único espaço de projeto, o que habilita implementar o conceito de co-simulação de hardware e software na prática.

3.4.1 Introdução

A estrutura do ambiente System Studio pode ser utilizada para o desenvolvimento de sistemas em um único chip (do inglês System on a Chip, SoC). Este ambiente permite a captura e a simulação através de ferramentas, metodologias e bibliotecas específicas, que facilitam a implementação de modelos em diversos níveis de abstração e detalhamento. System Studio permite o reuso de componentes previamente desenvolvidos, além de possuir uma grande quantidade de módulos disponíveis em sua biblioteca de componentes.

A modelagem e simulação permitem descrever software e hardware através da combinação de diversas técnicas e ferramentas que compõem o ambiente de projeto, incluindo: um núcleo de simulação, suporte à biblioteca SystemC, controle de fluxo de projeto, construção de protótipos virtuais e exploração arquitetural.

System Studio está dividido em dois domínios principais, cada um refletindo o objetivo principal de projeto a que cada domínio dá suporte:

- domínio algorítmico;
- domínio arquitetural.

O domínio algorítmico descreve a funcionalidade de um sistema em nível não temporizado ou implicitamente temporizado. Os projetos são capturados usando uma combinação de fluxo de dados (do inglês Data Flow, DF) e FSMs hierárquicas estendidas. Detalhes como precisão em nível de ciclo de relógio e sinais de inicialização (em inglês reset) não são capturados, permitindo que um processo de modelagem mais simples, aumentando a velocidade de simulação e evitando que o projeto sofra restrições demasiadas nas suas fases iniciais.

O domínio arquitetural, baseado em descrições SystemC, captura a arquitetura de um sistema em vários níveis de abstração e granularidade. Isto habilita prover uma visão global da arquitetura em termos de processadores, memórias, barramentos e componentes específicos para uma dada aplicação (em inglês, Application Specific Integrated Circuits, ASICs). Além disto, é possível descrever a arquitetura individual dos componentes existentes sob o aspecto do fluxo de controle (em inglês Control Flow, CF) e DF. Os modelos permitem expressar variações no nível de abstração desde níveis muito altos, apropriados para análise em estágios iniciais do projeto, até modelos de hardware precisos em nível de ciclo de relógio e de pinos. O ambiente System Studio habilita o desenvolvimento integrado de abordagens arquiteturais e algorítmicas em um mesmo espaço de projeto.

3.4.2 Projeto algorítmico

O projeto algorítmico, cujo fluxo é ilustrado na Figura 14, consiste de esboços de algoritmos não temporizados ou implicitamente temporizados em vários níveis de precisão, representações de ponto flutuante ou ponto fixo e outros formatos abstratos. System Studio provê suporte a uma semântica gráfica de captura para grafos de fluxo de dados (em inglês Data Flow Graphs, DFGs) e FSMs.

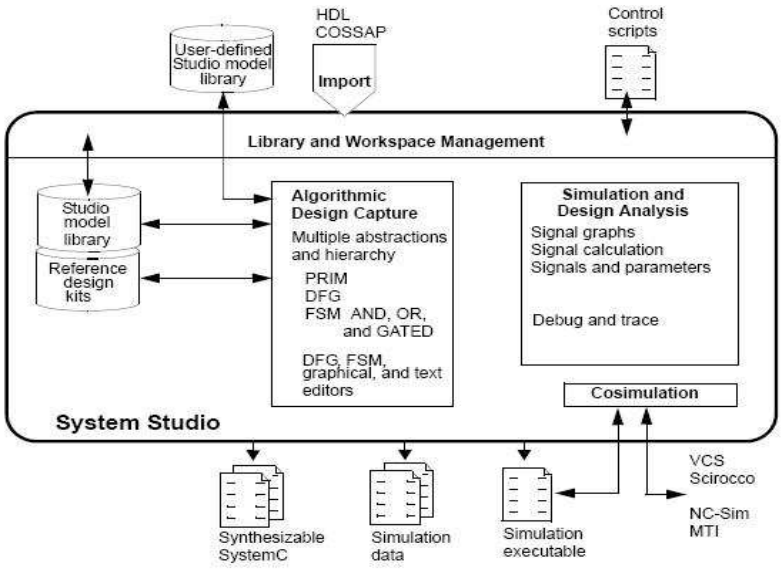

Figura 14 – Fluxo de projeto algorítmico no ambiente System Studio: entradas, saídas e estrutura interna [SYN06].

Nos modelos algorítmicos, a comunicação é modelada através de conexões ponto a ponto entre blocos. Algoritmos são modelados pela combinação de código e processamento de sinais expressos através do fluxo de dados e FSMs. Tais implementações ocorrem pelo uso de DFGs, implementados sobre modelos DF, para representar fluxo de dados e pelo uso de grafos de fluxo de controle (do inglês Control Flow Graphs, CFGs), implementados em modelos OR, para representar fluxo de controle. Outros dois modelos existentes, AND e GATED, são derivações hierárquicas dos modelos anteriores. Eles podem, portanto, conter tanto DFGs como CFGs. OR é um modelo usado para fluxo de controle (CFG) que permite especificar um diagrama de transição de estados no qual as instâncias existentes representam os estados contendo transições entre si; AND é um modelo hierárquico também usado para controle de fluxo, porém possui a capacidade de modelar concorrência, através múltiplas instâncias de CFGs denominadas páginas; GATED é um modelo hierárquico usado para controle de fluxo. Ele consiste, porém de uma ou duas páginas e uma condição de gating, que controla qual página deve ser executada e qual deve ser suspensa [SYN06].

O uso de hierarquia admite a modelagem de projetos complexos. DFGs de nível superior podem conter, em níveis de hierarquia inferiores, outros DFGs e/ou CFGs. Implementações CFG podem conter estados atômicos ou hierárquicos. Os hierárquicos por sua vez podem conter DFGs ou outros CFGs.

3.4.3 Projeto arquitetural

A Figura 15 apresenta o fluxo de projeto arquitetural do ambiente System Studio. Este tipo de projeto permite verificar software e hardware de forma concomitante, projetar plataformas e explorar arquiteturas a partir de seus componentes básicos tais como elementos de processamento, de interconexão, de armazenamento e periféricos.

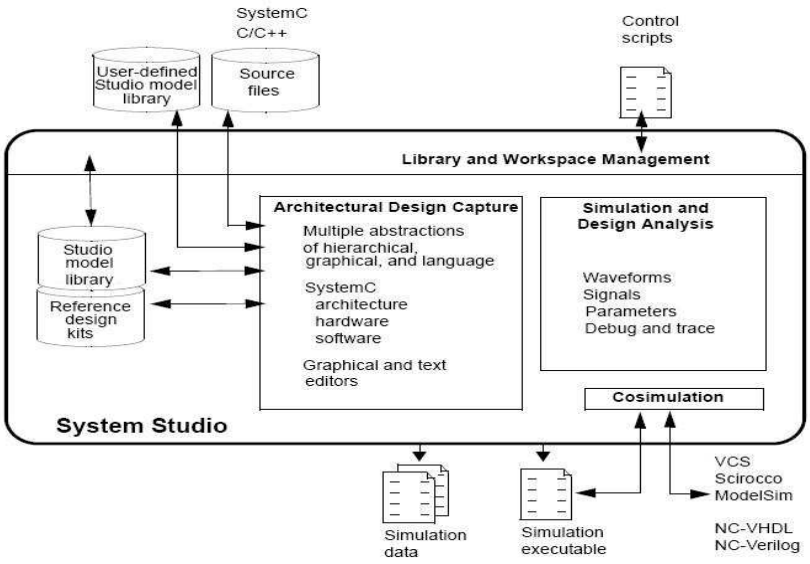

Figura 15 – Projeto arquitetural no ambiente System Studio: entradas, saídas e fluxo interno [SYN06].

System Studio dá suporte à modelagem em TL, ou seja, à exploração de arquiteturas onde um conjunto de elementos se comunicarem através de canais abstratos. O uso de modelagem TL permite aumentar a velocidade das simulações quando comparado a simulações RTL tradicionais, o que reduz o tempo de validação e, portanto, o tempo de desenvolvimento do projeto. O desenvolvimento de software pode se beneficiar do uso de modelos TL, uma vez que modelos funcionais podem estar disponíveis em estágios iniciais do desenvolvimento, permitindo que o software da plataforma possa ser desenvolvido e testado mais cedo.

A arquitetura é criada habitualmente fazendo-se uso da linguagem SystemC, embora existam outras possibilidades como C e C++. Isto leva a uma descrição completa da modelagem em nível de código fonte dentro do System Studio. Também podem ser importados códigos previamente desenvolvidos. Em ambos os casos, cada módulo possuirá uma representação gráfica, que poderá ser utilizada na criação de modelos hierárquicos, compondo novos módulos mais complexos de maneira relativamente simples e rápida.

Existem basicamente dois tipos de modelos usados em projetos arquiteturais: modelos primitivos e modelos hierárquicos.

Modelos primitivos são implementados em linguagem SystemC e se caracterizam por ser o modelo de mais baixo nível hierárquico, ou seja, não permite que outros modelos sejam instanciados dentro de um modelo primitivo.

Modelos hierárquicos também são implementados em SystemC, mas permitem que outros modelos, tanto primitivos como hierárquicos, possam ser instanciados dentro deste modelo. É possível criar estruturas com níveis arbitrários de hierarquia, onde cada modelo é percebido pelos modelos que o contêm como uma caixa preta funcional acessível através de sua interface.

Devido à disponibilidade de descrições de partida em SystemC como o modelo da rede HERMES descrito na Seção 3.2.3, a escolha para a modelagem do hardware proposto neste trabalho recai naturalmente sobre o domínio arquitetural da ferramenta System Studio.

3.4.4 Interface gráfica do ambiente System Studio

A janela principal da interface gráfica do ambiente System Studio é dividida em cinco áreas principais, conforme pode ser visualizado na Figura 16.

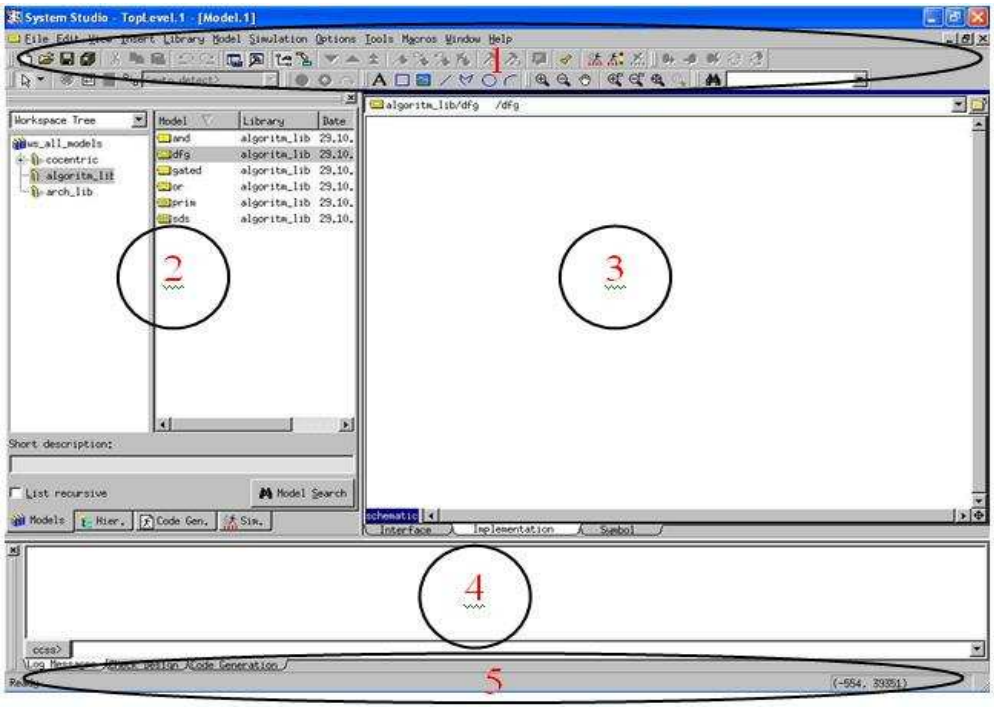

Figura 16 – Interface gráfica do ambiente System Studio [SYN06], mostrando as áreas de trabalho principais da ferramenta.

Cada área possui um propósito específico conforme sintetizado a seguir:

1. Barras de Menu e Ferramentas: permite executar as ações disponíveis no System Studio sobre arquivos, bibliotecas, modelos, simulações, etc. A barra de ferramentas è composta de cinco grupos conforme segue:
   - Padrão: botões para operações básicas sobre o objeto selecionado, como: novo, abrir, copiar, recortar e desfazer;
   - Navegação: botões para movimentação através da hierarquia de modelos do projeto;
   - Projeto: botões para verificar módulos e executar simulações.
   - Esquemáticos: botões e caixa de texto para operações sobre os objetos dos modelos, como selecionar e adicionar instâncias e definir tipos de canais;
   - Gráficos: botões para fazer acesso e instanciar elementos gráficos

simples no projeto tais como: texto, círculos, linhas e retângulos;

- Zoom: botões para controlar o nível de zoom de visualização de objetos de diversas formas;
- Procura: botões e caixas de texto para localizar objetos inseridos nos modelos do projeto.

2. Janela Área de Trabalho: pode conter uma ou mais bibliotecas, dentro das quais podem existir coleções de modelos ou outras bibliotecas aninhadas, representadas por uma estrutura de árvore expansível em níveis hierárquicos. Possui quatro seções, cada uma agrupando facilidades distintas.
   - Modelos: centraliza os componentes do projeto desenvolvido. Apresenta a visualização das bibliotecas e módulos, possuindo duas molduras:
     - Hierárquica: provê uma exibição tipo árvore expansível em níveis, que apresenta a distribuição hierárquica dos objetos disponíveis no projeto;
     - Modelos: apresentam uma lista dos objetos, normalmente modelos, inseridos na estrutura hierárquica selecionada na moldura hierárquica.
   - Hierarquia: provê uma exibição tipo árvore expansível em níveis que permite navegar sobre toda a hierarquia disponível, não apenas de bibliotecas e modelos, mas também portas, parâmetros, redes, canais, etc. Esta página permite exploração dos objetos disponíveis na página modelos, além de apresentar as propriedades do objeto selecionado e permitir a aplicação de filtros de visualização;
   - Geração de Código: permite gerar código objeto e o ajuste de vários parâmetros relativos à simulação tais como: tipo de código gerado (otimizado, depurado), script de simulação e formas de execução da simulação;
   - Simulação: permite executar e controlar diversos aspectos relativos a simulações, incluindo: forma de início, tempo de simulação, formas de execução da depuração, além de dispor de guias para visualização de dados da simulação.
3. Janela Área de Projeto: local onde os modelos são desenvolvidos, com visualização gráfica ou textual dos objetos instanciados. É dividida em três abas interligadas logicamente: visualização de interface, visualização de

implementação e visualização de símbolos;

4. Janela de Mensagens: apresenta mensagens retornadas por ações realizadas pelo System Studio, agrupadas em páginas separadas de acordo com a atividade realizada;
5. Barra de Status: apresenta informações sobre o estado atual das ferramentas. Possui basicamente três abas distintas: registro de mensagens, com informações gerais sobre a execução de ações System Studio; verificação do projeto com informações sobre processo de verificação sintática de código dos objetos; geração de código, com informações sobre a geração e compilação do código fonte dos objetos. Adicionalmente, criam-se aqui abas para saídas geradas por simulações em execução.

4 MODELAGEM DO SUBSISTEMA PLASMA

O processo de modelagem e integração do processador MLite ao ambiente System Studio se deu pelo uso do ISS disponibilizado junto ao projeto Opencores. Para tal, duas opções de ISS foram consideradas como base para nortear desenvolvimento: o ISS Plasma desenvolvido disponível no projeto Plasma do Opencores [RHO07] e o ISS gerado a partir do uso da linguagem para descrição de arquiteturas ArchC [ARC08].

A opção finalmente adotada pra o presente trabalho foi o emprego do ISS do Plasma. Entre os principais motivos que levaram a sua escolha estão: (i) maior facilidade em relação à análise do código fonte, escrito em C e (ii) facilidade de acesso a informações a respeito do projeto, obtidas tanto a partir da documentação como junto ao seu Autor.

4.1 Características do simulador Plasma (projeto Opencores)

O ISS Plasma é um simulador comportamental do processador MLite escrito em linguagem C, cujo código ocupa um único arquivo, formado por diversas funções. Além da função principal (main), outras sete funções realizam as funcionalidades essenciais à execução do código de máquina MIPS-I. Adicionalmente, cinco funções dão suporte ao uso de memória cache. Considerando que o núcleo IP do sistema HeMPS não faz uso deste tipo de memória, esta parte do código não foi considerada na atividade de modelagem e integração do ISS no ambiente System Studio, não sendo portanto abordada neste trabalho.

Os parágrafos seguintes apresentam uma visão geral do ISS Plasma e das características operacionais realizadas pelas diversas funções do simulador.

O estado do processador, incluindo o conteúdo dos registradores de propósito geral e o valor do contador de programa, é mantido em uma estrutura de dados global e algumas variáveis adicionais declaradas no programa. Entre os campos da estrutura de dados destacam-se os seguintes:

- R[32]: vetor composto por 32 elementos do tipo long. Cada elemento do vetor representa um dos 32 registradores de propósito geral;
- PC, PC_NEXT, EPC: variáveis do tipo long relacionadas a registradores para endereçamento de instruções. Os dois primeiros indicam os endereços da instrução que atualmente está em execução e da próxima instrução, respectivamente. O EPC é utilizado quando da ocorrência de exceções, indicando o endereço da instrução seguinte ao ponto onde foi gerada a exceção;
- HI e LO: variáveis do tipo unsigned long que representam registradores

usados os cálculos aritméticos envolvendo multiplicação ou divisão;

- exceptionID: variável do tipo long, utilizada para sinalizar a ocorrência de uma exceção. No caso do ISS é usada para tratar instruções syscall e break;
- irqStatus: variável do tipo long que representa uma palavra de 32 bits utilizada para armazenar a máscara de bits relativa à ocorrência de interrupções;
- *mem: ponteiro para um vetor do tipo char que representa a memória principal. Seu tamanho é definido a partir de uma constante contida no código fonte;
- big_endian: variável do tipo long que indica se o código a ser executado pelo simulador possui o formato de armazenamento em memória do tipo big endian ou little endian.
- a funcionalidade básica do ISS pode ser resumida como segue:
- a função main aloca o espaço necessário para manter os registradores e a memória principal conforme tamanho pré-definido. O valor padrão é de oito MBytes mas pode ser alterado, a partir de uma constante inserida no código. A estrutura de dados é inicializada com zeros tanto nos registradores como na região alocada para conter a memória principal. A partir de um arquivo definido como argumento na linha de comando, carrega o código de máquina a ser executado na região destinada à memória. A mesma função main atribui ao PC o valor inicial 0 (zero). Finalmente, ela invoca a função do_debug para dar continuidade ao processo de execução do código via simulação;
- a função do_debug calcula o valor de PC_NEXT somando o valor 4 ao seu conteúdo atual que corresponde ao endereço da próxima instrução e invoca uma função que apresenta o estado atual do processador, ou seja, o conteúdo da estrutura de dados. Esta função mostra o valor dos 32 registradores de propósito geral, dos registradores PC, PC_NEXT, HI e LO além da instrução atualmente em execução juntamente com algumas instruções anteriores e posteriores, decodificadas para um formato modificado do código de montagem. A partir deste ponto a função executa um laço infinito. Este laço implementa o modo de execução iterativo, apresentando um menu que permite ao usuário controlar a forma como o simulador será executado. Pode-se assim realizar execução passo a passo ou de segmentos do código;
- a função cycle, chamada a partir da função do_debug, realiza as operações de busca, decodificação e execução de cada instrução do código de máquina.

Adicionalmente, outras duas funções são invocadas pela função cycle, mem_read e mem_write. A função mem_read permite realizar operações de leitura de dados da memória principal. A função mem_write realiza operações de escrita de dados nesta memória. Ambas as funções permitem realizar operações utilizando palavra (32 bits), meia palavra (16 bits) ou byte (8 bits), conforme a instrução utilizada, por exemplo lh (load

half word) determina a leitura de uma meia palavra.

4.2 Integração e modelagem do processador Plasma

Na versão atual do sistema HeMPS RTL, além do processador Plasma outros módulos de apoio estão presentes, como o controlador de acesso direto á memória e a interface de rede, compondo uma infra-estrutura de processamento Plasma, semelhante à versão original do projeto Plasma, porém diferente daquela apresentada na seção 3.3.2. Assim, a partir deste ponto o termo Plasma será considerado como a infra-estrutura de processamento na qual está inserido um elemento de processamento denominado MLite.

A infra-estrutura de processamento Plasma do sistema HeMPS RTL é composta por cinco módulos: processador MLite, a memória principal, a unidade de comunicação serial UART (do inglês, universal asynchronous receiver-transmitter), a interface com a rede (NI) e o controlador de acesso direto a memória (DMA). Os dois últimos módulos foram acrescidos para atender a requisitos de comunicação do sistema, bem como habilitar o uso de uma rede intrachip e a recepção de código para carga no processador através desta rede.

O Plasma abstrato desenvolvido neste trabalho é constituído por sete módulos. Embora todos estes façam parte da infra-estrutura Plasma, esta Seção irá tratar apenas do processador MLite e da memória principal, denominada RAM. O critério para tal abordagem se deve ao fato de que os demais módulos estão fortemente relacionados à rede de comunicação, sendo então abordados na Seção 6.2 que trata da modelagem de módulos complementares do sistema HeMPS.

O processo de modelagem ocorreu em três etapas:

- primeira Etapa: integração e modelagem do ISS no ambiente System Studio, com objetivo de gerar um modelo abstrato executável do processador neste ambiente. Este primeiro módulo, MLite, foi desenvolvido com base no ISS Plasma;
- segunda Etapa: objetivou a modelagem da memória principal como um módulo separado, visto que no ISS Plasma a memória é uma estrutura interna. Esta separação foi necessária para permitir que se faça acesso à memória de forma independente, uma vez que tanto o processador MLite como o controlador DMA podem ser mestres da memória;
- terceira Etapa: concentrou-se na modelagem dos demais componentes da infra-estrutura de processamento. Esta última fase será abordada na Seção 6.2, mantendo aqui o foco nos componentes processador e memória apenas.

4.2.1 Integração e modelagem: Etapa 1

A idéia inicial para integração do ISS no ambiente System Studio foi criar um módulo empacotador (do inglês, wrapper module) que funcionaria como interface para integrar diretamente o código C do ISS. Entretanto devido a questões técnicas relativas ao ambiente System Studio e à natureza interativa do simulador esta abordagem não se mostrou viável. Assim, para realizar a integração do código C do ISS e gerar um modelo funcional abstrato do processador MLite foram necessários diversos procedimentos de alteração do código do ISS, conforme descreve-se a seguir.

As definições básicas do código, tais como diretivas, macros, constantes e estrutura de dados foram separadas em um arquivo de cabeçalho, servindo como base para todas as instâncias de módulos Plasma.

O código necessário para executar a funcionalidade do processador foi inserido em um módulo baseado em um modelo arquitetural primitivo SystemC. Este código está contido em dois arquivos, cabeçalho e implementação, contendo uma combinação de código C e SystemC. O arquivo cabeçalho (.h) contém as definições de classe, a declaração e atribuição de portas SystemC, os parâmetros do sistema, o construtor e o protótipo das funções e processos. O arquivo de implementação (.cpp) contém o código C e SystemC que definem o comportamento do simulador. Este modelo primitivo do processador MLite, ilustrado na Figura 17, possui apenas duas portas SystemC em sua interface: data_in e data_out, implementadas como estruturas de comunicação do tipo FIFO. A primeira permite a entrada de dados a partir de módulos externos e a segunda envia dados para os módulos externos. Estas portas compreendem canais de comunicação dedicados, permitindo o fluxo de entrada e saída de dados do processador. O tipo de dado usado nos canais é unsigned long (palavra de 32 bits).

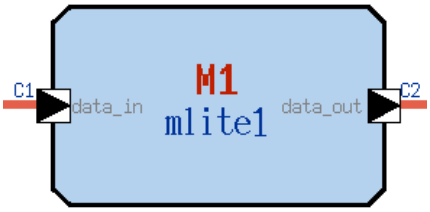

Figura 17 – Módulo do processador MLite criado no ambiente System Studio: representação do processador e de sua interface. O retângulo chanfrado representa a instância do módulo e os quadrados com seta os dois canais FIFO para comunicação de entrada e saída.

Nesta fase, foram definidos seis parâmetros para a chamada do ISS, usados para repassar informações necessárias à geração do código a ser simulado, além de permitir definir características relativas ao processo de simulação, conforme segue:

- big_endian: valor lógico, quando verdadeiro indica ao simulador que o formato do código de máquina a ser executado deve ser tratado como big endian. Quando falso indica formato little endian;

- decode_only: valor lógico, quando verdadeiro indica que o simulador deve apenas realizar a decodificação do código de máquina e mostrar a versão modificada do código de montagem. Quando falso procede à execução normal da simulação;
- cycle_show: valor lógico, quando verdadeiro o simulador mostra a versão modificada do código de montagem, além de executar cada instrução, quando falso inibe a apresentação;
- print_state: valor lógico, quando verdadeiro apresenta o estado atual do processador entre cada instrução executada. Sendo falso inibe esta apresentação;
- step_count: valor lógico, quando verdadeiro apresenta o valor correspondente ao número de “delta time” executados pelo simulador SystemC. Quando falso não apresenta este valor;
- app_name: valor string, define o nome do arquivo que contém o código de máquina MIPS-I a ser executado pela instância do simulador.

As funcionalidades gerais do código C original do ISS foram mantidas. Entretanto, foram necessárias adequações que exigiram diversas mudanças no código original. A maioria das funções foi modificada a fim de adequá-las à metodologia de desenvolvimento do ambiente System Studio As funções referentes ao tratamento de memória cache foram retiradas, uma vez que não são utilizadas na plataforma HeMPS.

Criou-se um processo SystemC que substitui as funções main e do_debug originais, de forma a retirar a característica interativa via linha de comando e criar um perfil de código compatível com a execução do ambiente. Também é incluído uma nova função para dar suporte ao processo inicial da simulação e alterou-se o código de forma a garantir que apenas as instruções da ISA MIPS-I sejam reconhecidas e executadas, além de dar suportar apenas às instruções aritméticas e lógicas destinadas às operações com inteiros.

De forma a permitir o controle do término da simulação a partir do código fonte, foi implementado um comportamento especifico para a instrução break, a qual causa o término da simulação ao ser identificada. Isto permite ao desenvolvedor de software encerrar a simulação pelo simples fato de inserir esta instrução no seu código de montagem.

4.2.2 Integração e validação: Etapa 2

Embora a Etapa 1 tenha produzido um modelo funcional válido do processador MLite, observou-se que este modelo não seria adequado ao uso no sistema HeMPS, uma vez que a memória foi embutida como interna ao modelo, o que impede seu uso pelo módulo de DMA. Caso a memória permanecesse dentro do módulo do processador, além

de dificultar o seu acesso também tornaria o modelo conceitualmente diferente da implementação original do Plasma no sistema HeMPS. Desta forma a Etapa 2 teve como principal objetivo realizar as adequações necessárias no módulo MLite e a criação de um módulo adicional para conter a memória principal, RAM. A Etapa 1 serviu contudo à validação inicial da integração do ISS Plasma ao ambiente System Studio.

A Figura 18 mostra a representação esquemática do módulo Plasma contendo o processador MLite, a memória RAM e o canal hierárquico que provê comunicação entre os dois módulos. Canais hierárquicos são uma estrutura de programa que pode conter outras estruturas SystemC embutidas, não se limitando apenas a um simples mecanismo de comunicação. Este tipo de canal oferece um método mais poderoso para modelar estruturas de comunicação complexas [GRO02], devido à disponibilização de métodos virtuais implementados dentro do canal e invocados pelos módulos que a este se ligam. Os módulos Mlite e RAM são interconectados por um canal hierárquico, que é composto de portas e interfaces SystemC. O canal pode ser visto no centro da Figura 18. Este canal hierárquico possui vários métodos SystemC utilizados para dar suporte ao fluxo de dados entre o processador MLite e a memória RAM.

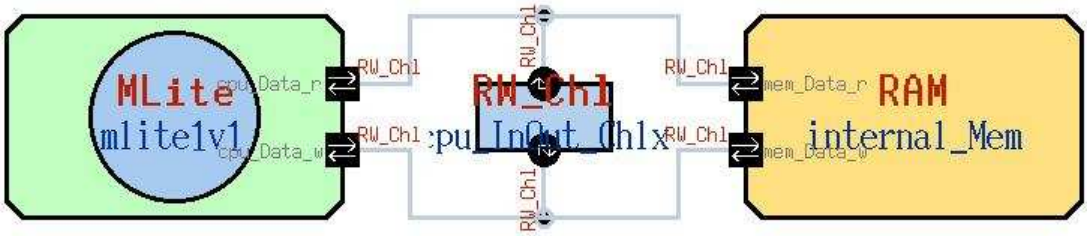

Figura 18 – Módulos MLite e RAM interconectados através de portas e canais de comunicação através de um canal hierárquico.

A separação da memória e do processador exigiu a alteração da estrutura de dados que mantém o estado do processador, a recodificação de partes do código fonte e a criação de um canal hierárquico SystemC que define a interface entre os módulos.

A estrutura de dados foi subdividida, ficando parte contida no módulo MLite, que manteve apenas informações relativas ao processador, como: banco de registradores de propósito geral, registradores PC e PC NEXT e algumas variáveis de controle. A outra parte da estrutura ficou no módulo RAM que basicamente manteve o vetor de bytes correspondente à memória e sua informação de endianismo.

O módulo MLite é implementado basicamente a partir de um processo e três funções de apoio. O processo do_iss controla a execução de cada instrução que é efetivamente realizada pela função cycle. Esta função realiza a busca, decodificação e execução de cada instrução. Ela é apoiada por duas outras funções, mem_write e mem_read, que realizam os acessos à memória, que ocorrem através de chamadas de métodos contidos na interface. O protocolo usado nas operações de leitura e escrita segue o paradigma “requisição/reconhecimento” (em inglês request/acknowledge) de

forma bloqueante, ou seja, a operação de requisição invoca o serviço desejado e aguarda que o reconhecimento ocorra, o que indica o fim da execução daquele serviço.

O canal hierárquico implementa os métodos invocados pelo processador e pela memória, métodos que efetivamente criam o mecanismo “requisição/reconhecimento”. Existem três grupos de métodos a serem usados: o primeiro é empregado no momento da carga do código objeto, enquanto os outros dois são usados pelas operações de leitura e escrita sobre memória, respectivamente.

Cada processador mestre ou escravo do sistema HeMPS possui sua própria memória principal onde é armazenado o código de suas tarefas e o microkernel. Um requisito existente no sistema é o suporte ao mecanismo de paginação de memória. Este mecanismo permite a execução de múltiplas tarefas compartilhando a mesma memória, porém em páginas distintas. No Plasma original um endereço de memória é composto de 13 bits, o que permite um espaço de endereçamento de apenas 8 Kbytes. Na plataforma HeMPS, o endereçamento de memória principal foi estendido em 3 bits possibilitando uma capacidade total de 64KBytes. Um destes bits é usado para ampliar o espaço endereçável para 16KB e os outros dois são usados como um seletor de página de memória permitindo endereçar até 4 páginas. Este mecanismo permite que os 64KBytes de memória endereçáveis sejam seccionados em 4 páginas de memória de 16Kbytes cada uma.. O registrador de página (PAGE), contido no co-processador 0 do MLite, é utilizado para armazenar o endereço da página ativa que concatenado aos demais quatorze bits de endereço compõem o endereço físico de memória.

A memória principal corresponde ao módulo RAM é o qual possui uma estrutura de dados contendo três variáveis: (i) uma variável tipo ponteiro para char usada para endereçar o início da memória, (ii) uma variável tipo long, que contém o tamanho do código de máquina carregado na primeira página, destinada ao microkernel e (iii) uma variável tipo long usada para indicar o formato de armazenamento dos dados, sendo 0 usado para big endian e 1 para little endian. Ainda existem três parâmetros usados para caracterizar a RAM:

- master_ram, variável lógica que indica se a instância do módulo é destinada para uso em um processador mestre (correspondendo ao valor verdadeiro) ou escravo (correspondendo ao valor falso);
- mem_pages, variável inteira que define o número de páginas existentes; considerando que cada página possui capacidade de 16Kbytes a partir deste parâmetro é possível obter a quantidade total de memória;
- big_endian, variável lógica que indica se o conteúdo da memória deve ser considerado no formato big endian (correspondendo ao valor verdadeiro) ou little endian (correspondendo ao valor falso).

O construtor do módulo instanciado cria um vetor que representa a memória,

conforme o número de páginas definido pelo parâmetro mem_pages. Adicionalmente, seleciona-se o microkernel a ser carregado, a partir do arquivo kernel_master.txt caso seja o processador mestre da HeMPS ou a partir do arquivo kernel_slave.txt caso o processador seja do tipo escravo, ambos gerados pela ferramenta HeMPS Editor. HeMPS Editor é uma ferramenta de software destinada a automatizar diversos passos do fluxo de projeto para o sistema HeMPS, permitindo personalizar várias características da configuração da plataforma [CAR09]. Mais detalhes podem ser encontrados em [CAR09].

Existem quatro processos responsáveis pela funcionalidade da memória:

- load_code: processo responsável por selecionar e carregar código de máquina do microkernel a ser executado no processador, de acordo com o seu tipo, mestre ou escravo;
- cpu_write_service e cpu_read_service: processos que implementam serviços relativos às operações de leitura e escrita em memória requisitadas pelo processador. Existe suporte para operações sobre palavra, meia-palavra e byte, conforme a instrução que esteja sendo utilizada;
- dma_write_service: processo que implementa o serviço de escrita em memória originado pelo DMA. Esta operação é requisitada quando o DMA realiza a transferência de código de máquina da tarefa a ser alocada recebida através da rede intrachip. Como não existe nenhuma situação em que o DMA necessite ler da memória, não foi implementado o processo correspondente a esta operação.

De forma a garantir eficácia aos processos que dão suporte às operações em memória, ou seja, assegurar que a carga na memória do código de máquina ocorra antes que qualquer requisição à memória seja feita pelo processador (cpu) ou DMA, é necessário que o processo load_code seja executado antes da ocorrência de qualquer requisição..

5 MODELAGEM DA REDE INTRACHIP HERMES

5.1 Introdução

A modelagem da infra-estrutura de comunicação HERMES [MOR04], implementada neste trabalho, foi realizada a partir da versão abstrata escrita em SystemC por Moreno e descrita em [MOR04a], conforme introduziu a Seção 3.2.3. Esta modelagem foi implementada a partir de uma hierarquia de módulos funcionais estruturada em camadas. A camada mais interna é composta apenas de modelos arquiteturais (ver Seção 3.4.3) primitivos do System Studio, escritos em SystemC. As demais camadas são compostas de modelos arquiteturais hierárquicos System Studio introduzidos na mesma Seção e escritos em SystemC. Os dados enviados pela rede são transmitidos como unidades de comunicação denominadas flits, cujo tamanho foi definido em 16 bits seguindo o padrão adotado na plataforma HeMPS, embora possa assumir outros valores conforme a necessidades. A parametrização dos modelos não é, entretanto endereçada neste trabalho.

5.2 Modelagem hierárquica dos módulos da HERMES

5.2.1 O roteador (router)

O componente fundamental mais importante da rede HERMES é o roteador, aqui denominado router. Este componente é construído como um módulo hierárquico SystemC, no qual são instanciados cinco módulos do tipo portas de comunicação. Estas portas, conforme discutido na Seção 3.2.1, controlam o recebimento e envio de dados que chegam ao roteador, e um módulo para a lógica de controle, também introduzida na mesma Seção. As portas para comunicação são denominadas door e a lógica de controle é denominada intraRouterChl. Toda a comunicação que flui pela rede passa por roteadores. O fluxo de pacotes entra através de uma das cinco portas de entrada, é processado no interior do roteador e sai através de uma das portas de saída. Cada transferência é controlada por um árbitro e pelo algoritmo de roteamento. O roteador não possui nenhuma descrição algorítmica, servindo apenas como uma estrutura hierárquica que permite interconectar portas de comunicação (de entrada e de saída) e a lógica de controle.

A Figura 19 mostra a representação esquemática do roteador conforme aparece na janela gráfica do System Studio. Nela podem ser vistas as cinco instâncias das portas de comunicação (East_0, West_1, North_2, South_3 e Local_4) e uma instância da lógica de controle (interRouter) ao centro. Localizados nas extremidades da Figura, existem dez retângulos menores que estabelecem a interconexão entre as portas systemC do módulo

hierárquico router às portas SystemC dos módulos portas de comunicação instanciados no interior deste. No System Studio estas interconexões são chamadas pseudo-portas systemC, pois estabelecem a interconexão através de ponteiros que relacionam a porta systemC do módulo hierárquico com a porta systemC do módulo embutido.

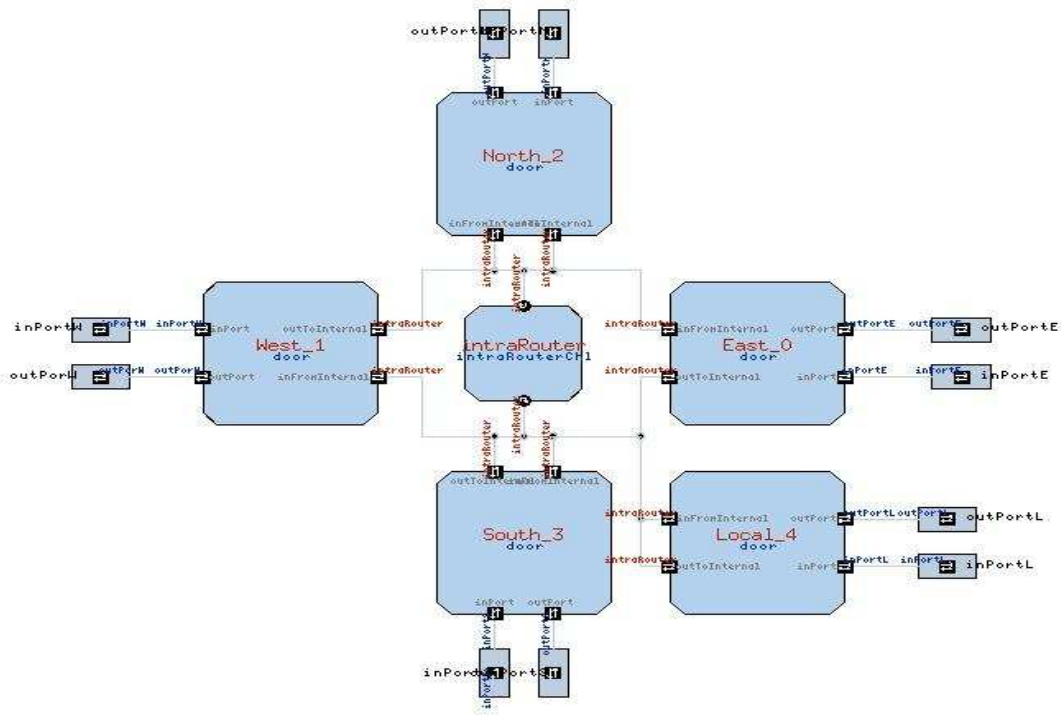

Figura 19 – Representação esquemática do roteador HERMES: componente usado para transferência de dados entre nodos da rede ou núcleos IP.

5.2.2 Módulo intraRouterChl

O módulo intraRouterChl é um canal hierárquico que interconecta portas de comunicação. Toda transferência interna de dados passa por este canal. O módulo é composto de um árbitro, que controla o acesso das portas de entrada ao roteamento, que estabelece a interconexão interna entre pares de portas. O roteamento elege por qual caminho (porta) o dado recebido pelo roteador deve sair, além de realizar o controle de fluxo da transferência entre portas. É possível ocorrer transferências em paralelo, desde que usando pares distintos de portas para comunicação entrada-saída. Devido a esta característica se faz necessário uma identificação das portas envolvidas, de forma a evitar possíveis colisões.

Conforme introduzido na Seção 3.2.1, cada porta do roteador possui uma identificação: EAST, WEST, NORTH, SOUTH E LOCAL, as quais são representadas internamente por valores numéricos, respectivamente de 0 a 4.

A Figura 20 mostra a representação esquemática de uma instância do módulo intraRouterChl. A parte central da figura representa o módulo, com a funcionalidade descrita anteriormente. O símbolo círculo com seta dentro representa uma porta de interface, pela qual as portas SystemC das portas de comunicação se vinculam à lógica

de controle; os componentes periféricos da Figura representam as portas de comunicação. A porta de interface é na prática um ponteiro para função que permite acesso a métodos virtuais implementados no módulo intraRouter.

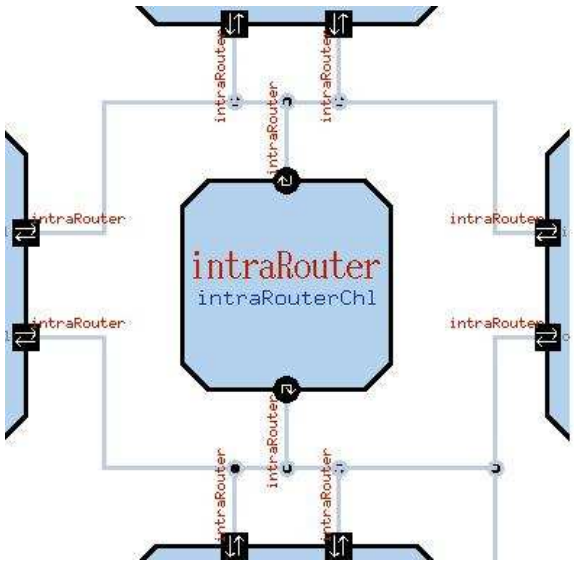

Figura 20 – Instância do módulo IntraRouterChl. Mostra-se o canal de comunicação interno entre portas de comunicação do roteador HERMES.

5.2.3 Módulo door

As portas para comunicação são denominadas door e são implementadas como módulos hierárquicos. Um módulo door é o componente fundamental para apoiar o fluxo de entrada e saída de dados dentro de um roteador. Através do módulo door é possível receber dados que chegam ao roteador e enviá-los em direção ao seu destino. A Figura 21 mostra a representações esquemáticas deste módulo. No exemplo da Figura este aparece instanciado no ambiente System Studio como uma porta oeste do roteador. A parte esquerda da Figura mostra duas pseudo-portas conectadas às duas portas SystemC do módulo exemplo (inPortW-inPort e outPortW-outPort) usadas para realizar a comunicação com portas de roteadores vizinhos. A parte direita mostra duas outras portas, inFromInternal e outToInternal, usadas para realizar a comunicação com a parte interna do roteador, a lógica de controle.

Semelhante ao roteador, o módulo door não possui nenhuma descrição algorítmica, servindo como estrutura hierárquica que interconecta portas SystemC do módulo router à lógica de controle interna.

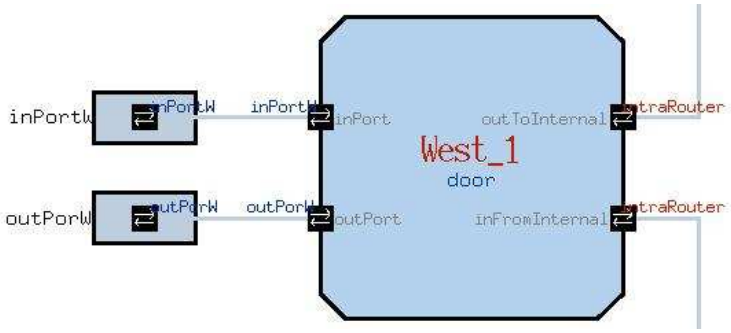

Figura 21 – Representação esquemática do módulo door, usado para dar suporte ao fluxo de comunicação de dados em um roteador HERMES.

5.2.4 Módulos inDoor e outDoor

O nível mais interno da hierarquia do roteador é composto por dois módulos primitivos: inDoor e outDoor. Ambos estão inseridos dentro do módulo hierárquico door. Estes módulos implementam o mecanismo que dá suporte à transferência de dados que chegam a uma porta de entrada do roteador. Este último deve encaminhar estes dados à lógica de controle que os enviará a uma porta de saída. Para permitir este fluxo de dados, os módulos citados realizam chamadas de métodos virtuais que residem dentro do módulo lógica de controle. A Figura 22 mostra a representação esquemática destes dois módulos fundamentais: ao centro se encontra o retângulo chanfrado que representa o código systemC que contém a funcionalidade e à esquerda e à direita as pseudo-portas usadas para interconectar portas systemC destes módulos com o módulo hierárquico onde serão inseridos.

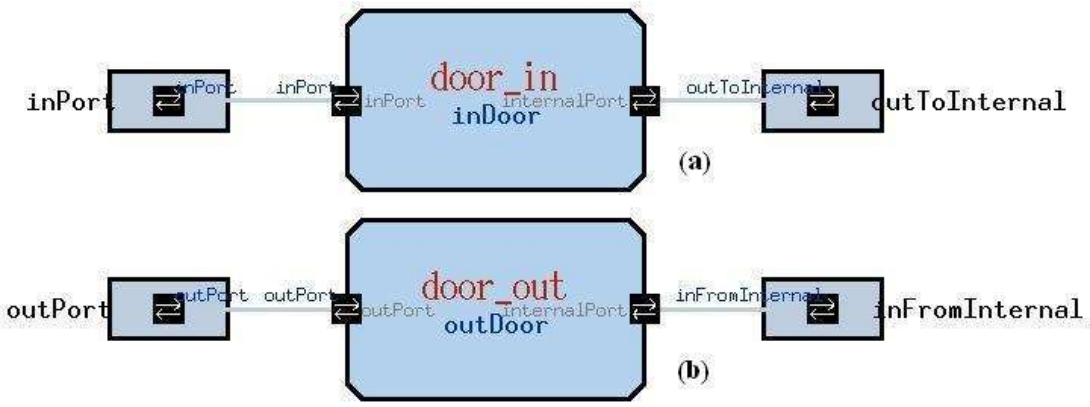

Figura 22 – Portas primitivas de entrada (a) e saída (b).

O módulo inDoor, item (a) da Figura 22, corresponde ao componente envolvido no fluxo de entrada de dados do roteador, e contém duas portas SystemC (inPort e internalPort) descritas a seguir vinculadas a duas interfaces intoRouterIf e outToInternalIf,:

- inPort: atua como meio de entrada, comunicando-se com o lado externo do roteador, de onde são recebidos os dados, transferindo-os para uma área de armazenamento temporário;
- internalPort: atua como repassador dos dados recebidos através de inPort,

para o interior do roteador, onde são tratados pela lógica de controle.

O módulo outDoor, item (b) da Figura 22, corresponde ao componente envolvido no fluxo de saída de dados do roteador, e contém duas portas SystemC (outPort e internalPort) descritas a seguir vinculadas a duas interfaces outFromRouterIf e inFromInternalIf:

- outPort: atua como meio de saída, comunicando-se com o lado externo do roteador, para onde são enviados os dados provindos do interior do roteador;
- internalPort: atua como repassador dos dados previamente tratados pela lógica de controle do roteador, ou seja, que estão sendo roteados.

5.2.5 Módulo interRouterChl

O módulo interRouterChl representa um canal hierárquico entre módulos roteadores, que implementa o mecanismo de interconexão entre portas para comunicação entre roteadores ou de um roteador com um núcleo IP. Este módulo estabelece a estratégia de controle do fluxo através da rede. Toda a transferência de dados entre roteadores na rede e entre roteador e módulo IP local é gerenciada pelo módulo interRouterChl. Ou seja, este componente corresponde a um canal hierárquico que realiza a interconexão entre portas para comunicação de roteadores vizinhos ou entre roteador e núcleo IP. A Figura 23 mostra a representação esquemática do módulo interRouterChl.

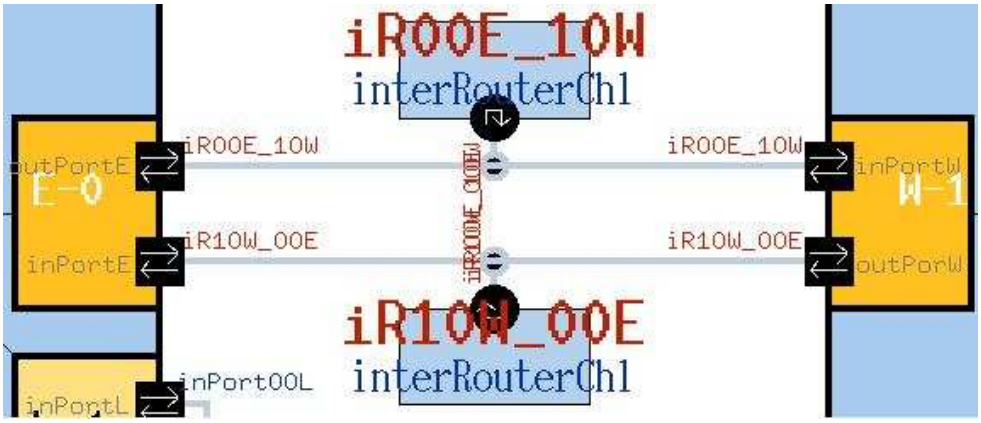

Figura 23 – Módulo interRouterChl, um canal hierárquico de interconexão entre roteadores e entre roteador e núcleo IP.

Este módulo interconecta dois roteadores através de suas portas leste e oeste. A funcionalidade ocorre pela chamada de métodos virtuais implementados no canal hierárquico realizados pelas portas de interface do roteador. Este módulo implementa os métodos virtuais definidos nas interfaces intoRouterIf e outFromRouterIf, que realizam o controle de envio e confirmação de recebimento dos dados transmitidos entre nodos da rede. Assim, estes permitem o fluxo de dados entre as portas dos roteadores e núcleos IP.

5.2.6 Modelo da rede HERMES

Uma vez descritos todos os módulos componentes do roteador, é possível construir uma rede HERMES. Para gerar uma rede HERMES, utilizam-se dois módulos básicos: router e interRouterChl, instanciados dentro de um módulo hierárquico que dará origem a uma instância particular da rede HERMES. A quantidade de módulos router e sua disposição define as dimensões da rede. A quantidade de módulos interRouterChl é estabelecida conforme o número de pares de portas de comunicação a serem interconectadas e portas não utilizadas. A Figura 24 mostra a representação esquemática do System Studio de uma rede cuja dimensão é 2x2, que contém quatro roteadores.

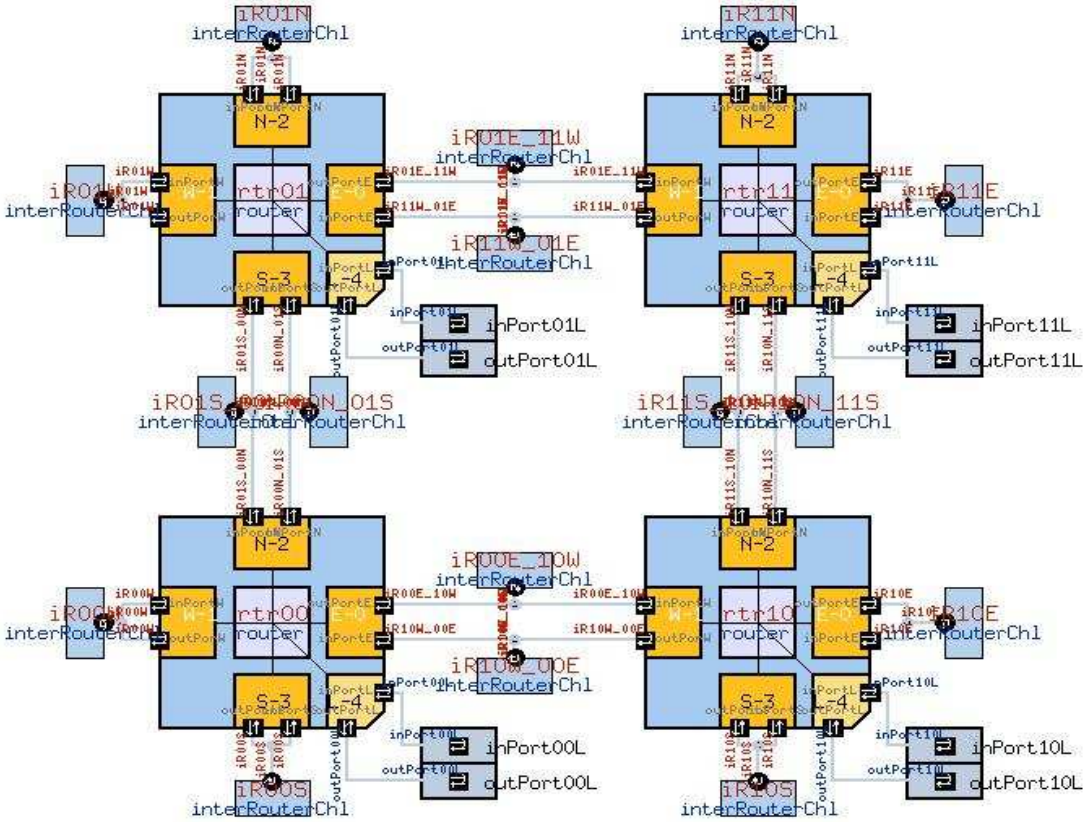

Figura 24 – Rede intrachip HERMES com dimensão 2x2, instanciada em um módulo hierárquico que representa uma rede intra-chip HERMES 2x2.

Nesta Figura pode-se visualizar os dois tipos de módulo básico, além das pseudo-portas usadas para interconectar as portas de comunicação de IP local de cada roteador às portas para comunicação do módulo hierárquico onde estes módulos básicos estão inseridos. O todo corresponde à rede intra-chip, conforme segue:

- roteadores: quatro instâncias do módulo router, representadas pelos quadrados de canto chanfrado onde toda a funcionalidade inerente ao roteamento introduzida anteriormente está programada;
- canais hierárquicos interRouterChl: quatorze instâncias, representadas pelos retângulos de mesmo nome; metade destes módulos são usados apenas para “fechar” as conexões das portas de comunicação não utilizadas (periféricas);

esta internconexão é necessária uma vez que nos roteadores existem métodos virtuais declarados os quais estão implementados no módulo interRouterChl;

- pseudo-portas: oito instâncias, representadas pelos retângulos denominados inPortXXL e outPortXXL, onde XX representa o endereço do roteador na rede e L indica a porta para um núcleo IP local, usadas para interconectar as portas SystemC da porta de comunicação às portas SystemC do módulo hierárquico de nível imediatamente superior, a rede HERMES.

Todos os módulos integrantes da instância da rede HERMES apresentada na Figura 24 possuem uma representação esquemática no ambiente System Studio. Eles também possuem uma vista gráfica simbólica, que representa o módulo visto de fora, ou seja, uma espécie de representação gráfica do módulo tipo “caixa preta”. A Figura 25 mostra esta representação esquemática da rede HERMES da Figura 24. Além da infra-estrutura de rede representada pelo retângulo chanfrado externo, destacam-se quatro pares de portas SystemC usada para interconectar módulos IP locais, portas estas que estão diretamente conectadas às portas de comunicação locais de cada roteador existente no interior da rede, através das pseudo-portas citadas anteriormente.

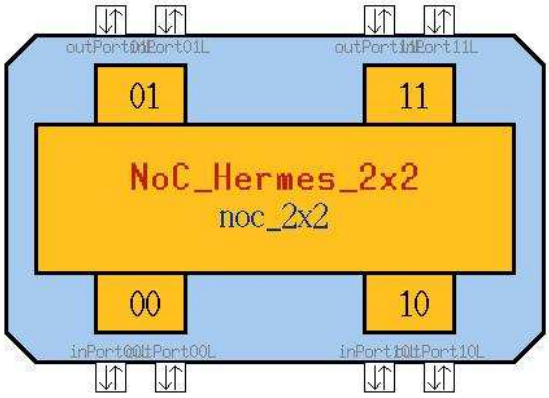

Figura 25 – Representação esquemática de uma rede HERMES 2x2, módulo hierárquico noc_2x2.

Para completar a construção da rede HERMES basta inserir o módulo hierárquico noc_2x2 dentro de outro módulo hierárquico e interconectar as portas para comunicação dos núcleos IP a canais hierárquicos interRouterChl devido ao uso de métodos virtuais, conforme já explicado anteriormente. A Figura 26 apresenta a representação esquemática da rede 2x2 juntamente com os canais hierárquicos. A rede intra-chip HERMES 2x2 está representada pelo retângulo chanfrados maior, os retângulos menores representam os módulos interRouterChl pelos quais os núcleos IP locais podem ser conectados.

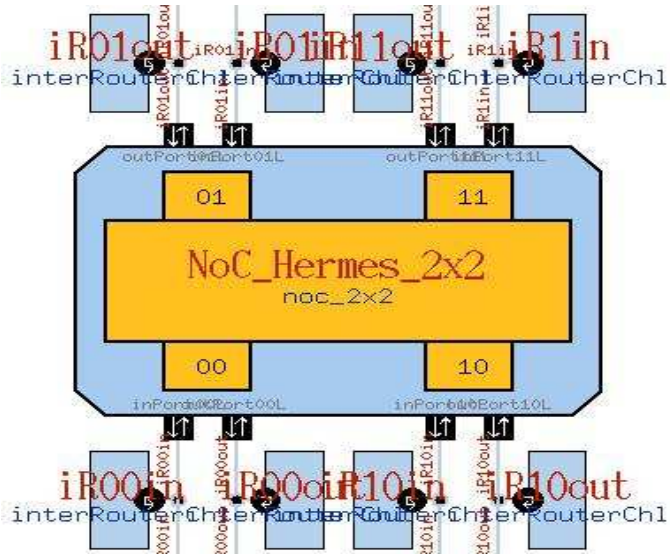

Figura 26 – Módulo noc_2x2, rede HERMES 2x2 juntamente com os canis hierárquicos que permitem conectar a rede aos núcleos IP locais.

Ainda é necessário realizar a configuração de quatro parâmetros, a fim de que a rede possa funcionar corretamente. Para configurar estes parâmetros no ambiente System Studio é necessário fazer acesso ao módulo hierárquico onde são inseridos os módulos noc_2x2 e interRouterChl, e selecionar o módulo noc_2x2. Em seguida, a partir do menu Model deve-se acionar o comando Push in e para cada instância de módulo roteador existente neste módulo hierárquico de nível imediatamente inferior deve-se abrir a opção Configure Object. Em seguida, aplica-se um duplo clique sobre o módulo router e seleciona-se a guia Constructor onde são inseridos os quatro parâmetros separados por vírgula, conforme a **Erro! Auto-referência de indicador não válida.**.

Tabela 5 – Parâmetros usados para configurar cada instância do módulo roteador HERMES.

Parâmetro	Comentário	Exemplo
name	**Nome de identificação do módulo *router*, que permite individualizar cada módulo instanciado**	**"rtr00"**
localAddr	**Endereço XY do roteador na rede (0xXXYY)**	**0x0000**
nocSize	**Tamanho da rede em dimensões cartesianas X, Y (0xXXYY)**	**0x0022**
algorRot	**Especifica o algoritmo de roteamento escolhido para ser executado no roteador**	**XYPuro**

6 MODELAGEM DO SISTEMA MULTIPROCESSADO HEMPS

De acordo com o terceiro objetivo específico descrito na Seção 1.3, este trabalho propõe a geração de um modelo do hardware do sistema HeMPS em alto nível de abstração. Conforme introduzido na Seção 3.1, já existem implementações do sistema HeMPS realizadas pelo GAPH. Contudo estas implementações foram desenvolvidas em RTL, ou seja, abaixo do nível de abstração pretendido. Esta Seção se destina a apresentar o processo de modelagem do hardware do sistema HeMPS desenvolvido no contexto deste trabalho.

O software do sistema HeMPS, e mais especificamente seu microkernel, não faz parte dos objetivos de modelagem propostos deste trabalho. Entretanto, para permitir uma melhor compreensão do processo de modelagem do hardware, bem como para dar suporte à validação da funcionalidade do modelo desenvolvido, o microkernel existente será explorado na próxima seção, porém em nível substancialmente superficial. Maiores detalhes do mesmo podem ser obtidos em [WOS07]. Também, o objetivo de validar o modelo abstrato da HeMPS necessitou que uma parte do microkernel fosse agregado ao primeiro, gerando uma validação parcial também do software da HeMPS em alto nível de abstração, conforme será descrito na Seção 7.4.

6.1 Componentes de software

Basicamente, o software da plataforma HeMPS pode ser dividido em duas classes: o microkernel, que corresponde ao sistema operacional; e tarefas, que compõem as aplicações. Ambos as classes são executadas nos processadores da HeMPS.

6.1.1 O *microkernel* HeMPS

O sistema operacional da plataforma HeMPS, denominado microkernel, foi desenvolvido considerando-se duas classes de aplicações: uma destinada a ser executada exclusivamente no processador mestre, e outra a ser executada nos demais processadores do sistema, os escravos. A associação de cada tipo de microkernel com cada um dos processadores ocorre automaticamente pala ferramenta HeMPS Editor, em tempo de projeto.

Considerando que a estratégia de paginação foi adotada para mapeamento de memória no sistema HeMPS, destina-se a página 0 (zero) de memória para a execução do microkernel. Uma vez que este pode ser maior que o espaço de uma página, como ocorre no caso do microkernel escravo, pode ser reservada mais de uma página para sua execução, estendendo-se até a página 1 (um). O gerenciamento da página ativa (segmento de código selecionado para execução) é realizado pelo próprio microkernel, a partir de um registrador de controle, denominado PAGE.

O escalonamento de tarefas é preemptivo, fazendo uso de um algoritmo round robin como estratégia de eleição da página a ser executada e por conseqüência do segmento de código (microkernel ou tarefa) a ser executado. As tarefas executam segundo o paradigma de fatias fixas de tempo (em inglês, time slices), onde cada tarefa executa por um tempo fixo antes de concorrer a um novo processo de escalonamento.

Uma vez que o paradigma de acesso à memória adotado segue o padrão NORMA (do inglês, no remote memory access) a comunicação entre processadores ocorre exclusivamente através de troca de mensagens, fazendo uso de serviços de comunicação, conforme será detalhado na Seção 6.2.2.3. Mensagens são escritas e lidas em uma área de memória local denominada pipe, mantida pelo microkernel e considerada um canal de comunicação entre tarefas. Na prática, um pipe é uma estrutura de dados contendo um registro para cada mensagem, este último composto por informações de controle e pela mensagem em si. Caso a comunicação ocorra entre tarefas localizadas em um mesmo processador, o processamento se restringe ao uso do pipe, não sendo necessário o uso de serviços especiais do microkernel. Por conseguinte, este tipo de troca de mensagens prescinde do processo de geração de pacotes para realizar a comunicação entre processadores distintos.

6.1.1.1 *Microkernel* mestre

Durante a fase inicial (em inglês, boot), o sistema operacional do processador mestre começa criando estruturas de dados. Estas estruturas são usadas para:

- controle centralizado de páginas livres em processadores;
- controle de alocação de tarefas;
- controle de processadores ativos no sistema.
- As operações básicas realizadas pelo microkernel mestre são as seguintes:
- atribuição inicial de valores às estruturas de controle, conforme definido em tempo de projeto na ferramenta HeMPS Editor;
- alocação estática de tarefas – consiste em enviar o código de máquina destas, armazenado inicialmente em memória externa denominada repositório, para cada um dos processadores escravos. Para tanto, faz-se uso do módulo DMA;
- confirmação de finalização da alocação estática – corresponde ao momento em que todos os processadores escravos tomam conhecimento que já estão alocadas todas as tarefas definidas como estáticas nos seus respectivos destinos. A partir deste ponto ativa-se o suporte a interrupções e inicia-se a execução de um laço eterno, representado uma tarefa ociosa (em inglês, idle task).

Quando ocorrer alguma interrupção, a tarefa ociosa é interrompida e inicia-se o tratamento daquela interrupção. O restante do código existente no microkernel dá suporte

às interrupções, tratando serviços de requisição dinâmica de tarefas, controle de término de execução de tarefa e dar suporte a mensagens de depuração do sistema. Não existe interrupção de software no processador mestre, uma vez que nenhuma tarefa lá executa, apenas o microkernel mestre.

6.1.1.2 Microkernel escravo

Durante a fase inicial, o microkernel do processador escravo cria uma estrutura do tipo fila circular, denominada ni_queue, utilizada para armazenamento temporário de mensagens a serem enviadas quando a interface com a rede estiver indisponível. A partir de um tipo definido pelo usuário, denominado TCB (abreviatura do inglês Task Control Block), são criadas várias outras estruturas para o controle de execução das tarefas. Estas estruturas mantêm o contexto das tarefas a serem executadas, como o valor do contador de programa (em inglês Program Counter ou PC), o endereço inicial em memória e o estado de execução (ou seja, o conteúdo do Banco de Registradores, do EPC, etc.).

As operações realizadas na fase inicial pelo microkernel escravo podem ser resumidas nas seguintes ações:

- desativa-se o tratamento a interrupções;
- atribui-se valores iniciais à fila ni_queue
- executa-se a rotina OS_Init, onde são realizados diversos procedimentos adicionais e reativa-se o tratamento de interrupções
- realiza-se o escalonamento de uma tarefa a ser executada.

A rotina OS_Init executa as seguintes ações:

- atribui-se valores iniciais às estruturas de controle tipo TCB. Estas são: system_tcb, destinada ao controle do microkernel; idle_tcb, usada para controle da tarefa “ociosa”, executada quando não existem tarefas de usuário disponíveis para entrar em execução; tcbs, que corresponde a um vetor contendo estruturas TCB em quantidade igual ao número de tarefas de usuário existentes, que tenham sido alocadas no processador escravo; pipe_slots, que representa a estrutura tipo pipe mencionada na Seção 6.1.1, compreendendo um vetor contendo tipos pipe; tasks_location, vetor que contém a lista global de todas as tarefas existentes e seus respectivos processadores; tasks_request, vetor usado para controlar mensagens enviadas às tarefas locais, que ainda não haviam sido alocadas quando do daquele envio; requestMessage, vetor usado para controlar requisições de mensagens a tarefas locais;
- ocorre o registro do endereço das rotinas que tratam as três possíveis interrupções de hardware, provindas da interface de rede, do DMA e do

gerenciador de tempo para execução de tarefas (time slice).

Em cada processador, após o escalonamento, uma tarefa entrará em execução. O escalonamento ocorre ao final de cada fatia de tempo, segundo o já mencionado algoritmo round robin. Cada tarefa é suspensa quando da ocorrência de uma interrupção, que pode ser de software ou de hardware. Neste momento, a rotina específica para cada tipo de interrupção será invocada. No caso de interrupção de hardware, chama-se a rotina específica para um dado dispositivo, sendo a tarefa em execução suspensa. Posteriormente ao tratamento da interrupção, esta retoma à execução, a não ser que a interrupção tenha sido gerada por motivo de término da sua fatia de tempo.

Interrupções de software são geradas para dar suporte à troca de mensagens entre tarefas (macros ReadPipe e WritePipe). Interrupções de hardware são geradas:

- pelo fim da fatia de tempo da tarefa, onde outra será escalonada, caso exista;
- pela interface de rede, para solicitar tratamento de pacotes;
- pelo DMA, para informar o término da sua operação previamente programada.

Interrupções de hardware suspendem a execução da tarefa atualmente em execução, a qual pode ser ter oportunamente retomada sua execução.

6.1.2 Aplicações

Na plataforma HeMPS tarefas correspondem a trechos de programas escritos pelo usuário em linguagem C. Estas tarefas são criadas para serem executadas nos processadores escravos, podendo existir tantas tarefas executando simultaneamente quantas páginas disponíveis existirem em cada processador.

As tarefas podem ser designadas diretamente para um determinado processador, de forma estática, ou podem permanecer sem associação específica, sendo alocadas de forma dinâmica ao serem invocadas por alguma outra tarefa, devido à execução de uma macro WritePipe. Neste caso tarefas são executadas sob demanda. A definição de qual tarefa será alocada de forma estática ou dinâmica é decidida em tempo de projeto através da ferramenta HeMPS editor. Quando atribuídas a um processador escravo serão alocadas estaticamente, quando restarem atribuídas ao processador mestre (esta é a situação inicial de qualquer tarefa na ferramenta HeMPS Editor) serão alocadas dinamicamente.

Conforme mencionado anteriormente, a comunicação entre tarefas ocorre por meio de mensagens, fazendo-se uso de um canal de comunicação (o pipe). Para dar suporte à comunicação, tarefas executam macros WritePipe, para enviar uma mensagem a um destino e ReadPipe para ler uma mensagem enviada por uma origem. Adicionalmente, tarefas podem executar duas outras macros: GetTick, que retorna o número de ciclos de relógios executados, contabilizados desde o primeiro ciclo gerado e Echo, que é definida como serviço de depuração, tendo como efeito o envio da uma

mensagem para o módulo UART do processador mestre.

Considerando que este trabalho realiza a modelagem em alto nível de abstração, detalhes como operação por ciclo de relógio não estão presentes. Na modelagem realizada a macro GetTick foi modificada para retornar o número de instruções executadas desde o início do processamento, contabilizando todas as instruções executadas, não apenas das tarefas, mas também do microkernel e de tarefas idle. Desta forma, esta medida serve como parâmetro para aproximar uma métrica de tempo. Na organização Plasma, devido ao uso de pipeline, a vazão de instruções por ciclo de relógio é igual a um, com exceção das instruções load e store que consomem 2 ciclos e das instruções de multiplicação e divisão que consomem 32 ciclos, quando não estiver ativada a otimização de cálculo no hardware [RHO07].

6.2 Componentes de hardware complementares ao módulo Plasma

Além dos modelos abstratos desenvolvidos para o processador Plasma, introduzidos na Seção 4.2, foram desenvolvidos outros cinco módulos complementares para completar o modelo do processador da plataforma HeMPS. Estes módulos podem ser divididos em dois grupos. O primeiro é composto pelos módulos de armazenamento desenvolvidos para dar suporte ao repositório de tarefas e pelos registradores usados para controle ou estado. O segundo grupo é composto pelos três módulos usados para dar suporte à comunicação. As duas Seções a seguir apresentam a modelagem dos componentes destes grupos.

6.2.1 Módulos complementares: MEM_EXT e CP0

Existem três classes de dispositivos de armazenamento a serem consideradas no módulo Plasma: memória principal (RAM), memória externa (MEM_EXT) e o co-processador 0 (CP0).

A Figura 27 mostra a representação esquemática do módulo Plasma, contendo estes dispositivos. Na Figura podem ser visualizados quatro módulos: o processador MLite, a memória interna RAM; a memória externa MemExt, que corresponde ao repositório de tarefas, e o co-processador 0, CP0, que contêm registradores de controle do Plasma. Os dois primeiros já foram abordados nas seções 4.2. Todos os quatro módulos são interconectados através de um canal hierárquico de comunicação, que aparece na parte inferior da Figura.

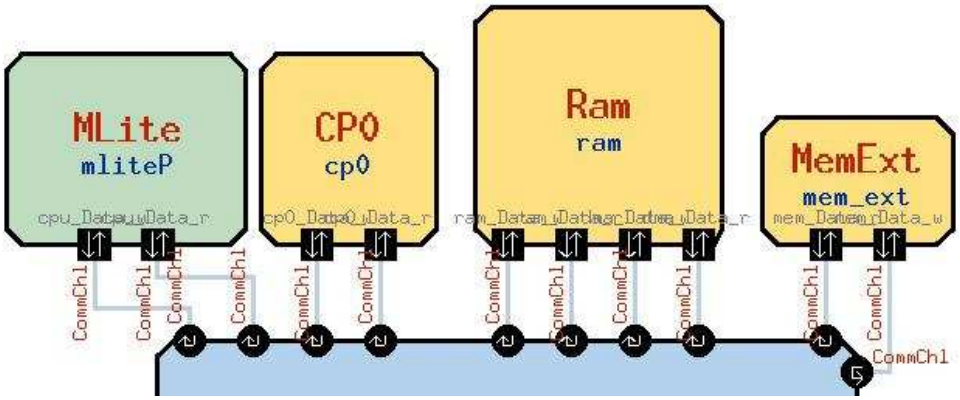

Figura 27 – Modelo Plasma, contendo o processador MLite e seus três dispositivos de armazenamento, interconectados por um canal hierárquico.

6.2.1.1 Módulo MEM_EXT: o repositório

O módulo MEM_EXT implementa a memória externa (secundária). Esta memória tem por função armazenar tarefas que deverão ser distribuídas e executadas nos diversos processadores do sistema. Ou seja, constitui o repositório de tarefas a distribuir nos respectivos elementos de processamento, onde serão alocadas nas páginas da memória principal do processador. A gerência da página a ser considerada é responsabilidade do microkernel.

Na versão RTL da HeMPS a memória externa é implementada a partir de um arquivo VHDL, gerado pela ferramenta HeMPS Editor, que descreve a imagem de memória contendo um cabeçalho e o código das tarefas.

Considerando que as tarefas a serem executadas são definidas em tempo de projeto, a ferramenta HeMPS Editor gera um arquivo cabeçalho C (com extensão .h). Este arquivo contém chamadas de função e respectivos parâmetros, em número necessário para realizar a carga de todas as tarefas previamente armazenadas no repositório. Também se gera um arquivo compatível com a imagem de memória, denominado repository.txt. Este arquivo contém a representação hexadecimal, em formato texto, de todas as tarefas, estando dividido em duas partes: cabeçalho e código. A Tabela 6 apresenta a organização dos dados do repositório.

Tabela 6 – Organização do repositório de tarefas. O valor "m" representa "nº de tarefas*12" (3*4: três campos de 4 bytes) e "t" o tamanho total do código de todas as tarefas menos 1, em bytes.

	Cabeçalhos				Códigos			
Posição	**0 11**	**12 23**	**…**	**m-12 m-1**	**m**	**m+4**	**…**	**m+t**
Conteúdo	**Cabeçalho da 1ª tarefa**	**Cabeçalho da 2ª tarefa**	**… →**	**Cabeçalho da nª tarefa**	**1ª instrução da 1ª tarefa**	**2ª instrução da 1ª tarefa**	**… →**	**nª instrução da nª tarefa**

Para interpretar esta Tabela lembra-se que a memória do MIPS é endereçada a byte. O cabeçalho é composto de três campos de 32 bits para cada tarefa (12 bytes), e é replicado para cada tarefa. A interpretação de cada palavra (32 bits) do cabeçalho é: (i) identificação numérica da tarefa, (ii) tamanho do código de máquina da tarefa, em número de palavras e (iii) endereço inicial da tarefa no repositório, onde o código desta inicia

(offset). Após os cabeçalhos, segue-se a região onde são armazenados os códigos de máquina de cada tarefa, na mesma ordem dos cabeçalhos.

Esta memória ocupa o segmento de endereços que inicia em 0x1000000 e vai até 0x1FFFFFFF. Assim, o dado armazenado no índice 0 da Tabela acima ocupa o endereço 0x10000000. Toda operação realizada sobre esta faixa de endereços é dirigida ao módulo MEM_EXT.

O conteúdo desta memória é previamente definido (em tempo de projeto) e não muda ao longo da execução de aplicações na plataforma HeMPS. Logo, apenas a operação de leitura é necessária. Existem três processos que implementam a funcionalidade do repositório:

- load_task_code: este processo abre o arquivo repository.txt, cria dinamicamente uma estrutura para conter os dados do repositório, carrega nesta os cabeçalhos e o código das tarefas. Como as informações contidas no arquivo estão em formato texto, onde cada linha representa uma palavra, lê-se nove caracteres para gerar cada palavra a ser armazenada na memória (vetor): oito caracteres texto, representando aos dígitos hexadecimais, e o caractere nova linha;
- cpu_read_service: disponibiliza ao processador o serviço de leitura de palavras armazenadas na memória externa. Como a única operação sobre o módulo MEM_EXT é a leitura das palavras de tarefas, não existe necessidade de suporte a acesso para meia palavra ou byte;
- dma_read_service: disponibiliza o serviço de leitura de palavras armazenadas na memória externa realizada pelo DMA. Este procedimento ocorre quando o DMA executa a transferência do código de máquina de cada tarefa a ser enviada para a rede.

6.2.1.2 Módulo CP0: co-processador 0

A arquitetura MIPS dá suporte nativo a até quatro co-processadores. O co-processador 0 (CP0) é conhecido como o co-processador de controle, sendo implementado em praticamente toda versão do MIPS com maior ou menor funcionalidade. Este módulo desempenha duas funções principais para a CPU: gerenciar a memória virtual do sistema e controlar o processamento de exceções. Embora no Plasma a entidade co-processador 0 não exista explicitamente, considera-se aqui a modelagem de um módulo específico para representar a funcionalidade do CP0. O critério usado para esta abordagem baseia-se no agrupamento de registradores usados para controle em um local específico, semelhante ao que ocorre na definição da arquitetura MIPS.

Dentro do CP0, podem existir até 32 registradores para propósitos específicos. No sistema HeMPS são usados três destes apenas:

- SR ($12): registrador de estado (status). Quando ativado (com valor 1), indica que o processador irá tratar interrupções ocorridas. Quando desativado (com valor 0), o processador ignora interrupções;
- EPC ($14): registrador contador de programa de exceção (exception PC). Armazena o endereço da instrução seguinte àquela onde uma exceção foi gerada. Usado para permitir o tratamento de exceções e posteriormente permitir o retorno ao fluxo de execução anterior à ocorrência da exceção;
- PAGE ($10): implementado no contexto da plataforma HeMPS. Armazena o endereço inicial da página de memória. Usado para compor o endereço físico de acesso à memória.

Além dos registradores acima, foram inseridos mais quatro registradores neste módulo. O critério para estes registradores estarem localizados no CP0 foi a natureza de seu conteúdo. Como estes registradores são usados para propósitos específicos (controle), o módulo CP0 é o local mais lógico para estarem localizados. A lista abaixo introduz estes registradores:

- irq_status: registrador de requisição de interrupções. Armazena as interrupções geradas pelos dispositivos externos, registradas da seguinte forma: cada bit da palavra possui um dispositivo associado, ativar esse bit indica que aquele dispositivo gerou (registrou) uma interrupção;
- irq_mask: armazena a máscara de solicitação de interrupções. Semelhante ao registrador irq_status, possui um bit da palavra associado a cada dispositivo. Porém, o objetivo deste registrador é permitir ao processador controlar quais interrupções irá tratar, ou seja, o processador pode, via software, decidir que interrupções serão verificadas escrevendo neste registrador. As interrupções a serem tratadas serão cuja operação lógica E (And) entre os bits respectivos dos registradores irq_mask e irq_status produzir o valor 1;
- time_slice: registrador para contagem da fatia de tempo. Na implementação RTL do sistema HeMPS este registrador é incrementado a cada ciclo de relógio. Ao atingir o valor 16384 (4000 em hexadecimal) é gerada uma interrupção de hardware, de forma que o microkernel possa assumir o controle do processador e escalonar uma nova tarefa a ser executada, zerando então este registrador para iniciar uma nova contagem. Como a implementação desenvolvida neste trabalho é em alto nível de abstração, este valor é incrementado não a cada ciclo de relógio, mas sim a cada instrução executada, uma vez que a modelagem aqui considerada não contabiliza ciclos de relógio.
- tick_counter: registrador de 32 bits para contagem de ciclos de relógio. Na implementação RTL do sistema HeMPS contabiliza-se o número total de

ciclos. Novamente, como a modelagem aqui realizada não modela ciclos de relógio, a contagem é considerada contabilizando-se o número de instruções executadas.

6.2.2 Módulos complementares: UART, DMA e NI

Existem três dispositivos que dão suporte à comunicação ao processador Plasma na plataforma HeMPS. A Figura 28 mostra a representação esquemática parcial do módulo Plasma onde ocorrem estes dispositivos A controladora UART originalmente realiza a comunicação com um dispositivo serial, sendo aqui considerada como um console de comunicação com o ambiente System Studio. A interface de rede (em inglês Network Interface ou NI) atua como meio de interconexão entre a rede HERMES e os componentes internos do Plasma. O controlador de DMA realiza a transferência de dados entre a rede e a memória interna do Plasma, sem necessidade de interação do processador durante todo o processo. Todos os quatro módulos são interconectados através de um canal hierárquico de comunicação (na Figura denominado CommChl).

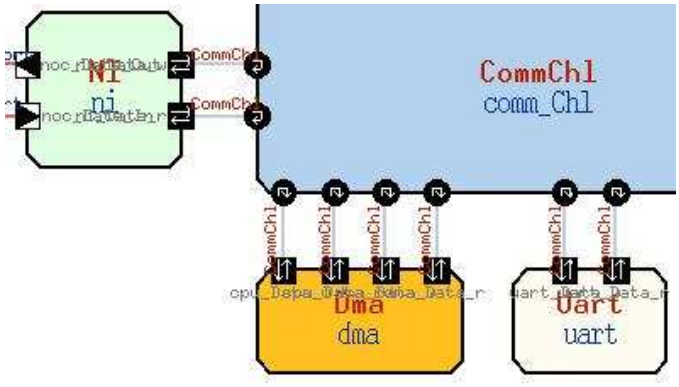

Figura 28 – Visão parcial do modelo do Plasma apresentando a interface externa dos três dispositivos relacionados à comunicação.

6.2.2.1 Módulo UART

O módulo UART é utilizado no sistema HeMPS basicamente para apresentar massagens de depuração enviadas pelos processadores escravos. No contexto deste trabalho, as mensagens geradas pelo módulo UART são apresentadas na janela de console do System Studio.

A UART é constituída por dois processos, com duas portas SystemC para escrita e leitura no/do módulo.

O processo write_service constitui o serviço de escrita no módulo. Mensagens de depuração recebidas pelo processador mestre são redirecionadas para a saída padrão. O segundo processo, read_service, permite receber valores do console, normalmente via teclado, e enviá-los ao processador. Este último serviço não é utilizado por nenhuma função desempenhada pelo microkernel ou pelo hardware. Entretanto esta função foi implementada, prevendo algum tipo de interatividade que possa vir a ser necessária

durante a simulação do modelo abstrato.

A Figura 29 apresenta uma ampliação da representação esquemática do módulo UART. Nesta Figura pode-se observar a UART como um retângulo chanfrado, as duas portas SystemC para uso em entrada e saída de dados e a conexão destas portas as interfaces do canal de comunicação, o que permite a integração do módulo ao restante do modelo Plasma.

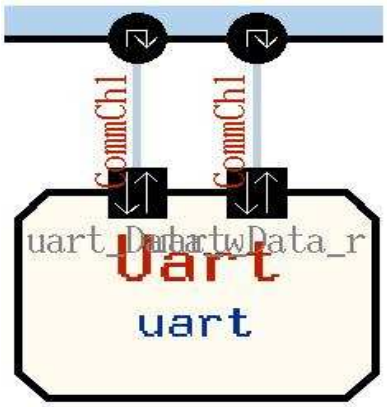

Figura 29 – Módulo UART: usado para envio de mensagens recebidas pelo processador mestre para o console.

Como o módulo UART opera com largura de dados de 8 bits (um byte) os dados a serem enviados ou eventualmente recebidos estão limitados a este tamanho, em cada operação.

6.2.2.2 Módulo DMA

O módulo DMA tem a função de realizar a transferência de tarefas armazenadas em memória externa (no caso da HeMPS no repositório de tarefas) a partir do processador mestre e enviá-las à rede, conforme distribuição de tarefas especificada em tempo de projeto na ferramenta HeMPS Editor. Complementarmente, nos processadores escravos, o DMA tem a função de receber o código objeto das tarefas e transferi-lo para uma página livre de memória. O controle de páginas livres em cada processador escravo é realizado de forma centralizada pelo microkernel mestre.

Para realizar transferências, o DMA é programado pelo processador através de um mecanismo de escrita em registradores mapeados em memória. Na versão HeMPS considerada neste trabalho, de agosto de 2008, existem cinco registradores mapeados em memória, descritos a seguir:

- DMA_SIZE: mapeado no endereço 0x20000200. A função deste registrador é permitir ao microkernel, tanto do processador mestre como do escravo, informar ao DMA o tamanho do código que deverá ser transferido, da memória externa para a rede ou da rede para uma página da memória principal;
- DMA_ADDRESS: corresponde ao endereço 0x20000210. A função deste registrador é informar ao DMA o endereço a partir do qual deverá iniciar a

transferência de dados. No processador mestre, corresponde ao endereço inicial da tarefa, na memória externa, a ser lida e enviada para o processador destino. No processador escravo indica o endereço a partir do qual o código deve ser armazenado na memória principal;

- DMA_OP: corresponde ao endereço 0x20000220. Este registrador tem como função indicar ao DMA em que sentido se dá a operação de transferência do código, podendo ser: origem o repositório de tarefas e destino a rede (código 0) ou a rede como origem e como destino a memória local (código 1);
- DMA_START: corresponde ao endereço 0x20000230. Escrita neste registrador permite ao processador sinalizar o início da transferência. A partir de uma escrita neste registrador o DMA passa a realizar a transferência do código no sentido previamente definido no registrador DMA_OP, liberando o processador para executar outra atividade. Ao final da transferência, o DMA gera uma interrupção de hardware para indicar ao processador que a tarefa foi concluída;
- DMA_ACK: corresponde ao endereço 0x20000240. A função deste registrador é receber a confirmação de término da transferência conforme reconhecida pelo processador. Ou seja, assegura-se ao DMA que o processador reconheceu e tratou a interrupção gerada. Na prática, esta ação faz com que o DMA desative a interrupção gerada anteriormente.

Os cinco registradores descritos acima estabelecem o protocolo de comunicação entre processador e DMA.

A Figura 30 mostra a reapresentação esquemática do módulo DMA. Nela pode ser visto o módulo DMA, representado pelo retângulo chanfrado, além de quatro portas SystemC usadas no fluxo de entrada e saída de dados, representadas pelos quadrados contendo setas duplas, e a conexão destas portas às portas do canal de comunicação que habilita a integração do módulo ao restante do sistema Plasma.

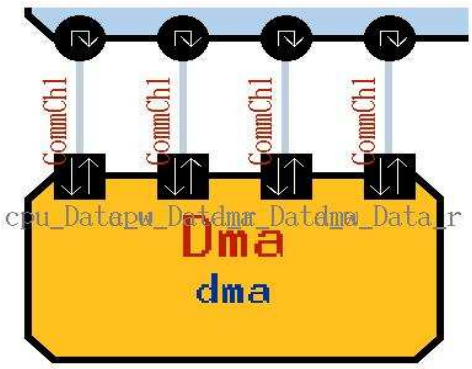

Figura 30 – Módulo de acesso direto à memória (DMA): usado na alocação de tarefas.

As duas portas mais à esquerda destinam-se à comunicação com o processador,

uma para escrita e outra para leitura. Embora o processador apenas escreva no DMA, foi disponibilizado suporte à leitura, possibilitando implementar esta operação de forma relativamente fácil caso seja necessário no futuro.

As duas portas mais à direita destinam-se a operações de escrita e leitura do DMA sobre as memórias local e externa. As operações atualmente implementadas são apenas leitura da memória externa e escrita na memória local.

O funcionamento do DMA pode ser dividido em três fases: programação, operação e finalização. Para funcionar, o DMA precisa ser previamente programado pelo processador. A fim de dar suporte a esta funcionalidade, foram implementadas duas funções e um processo. O processo, cpu_write_service, habilita o serviço de escrita do processador no DMA. Existe suporte para escrita nos cinco registradores acima mencionados.

Na primeira fase, programação, realizam-se operações de escrita pelo processador (mestre ou escravo) nos três primeiros registradores do DMA (tamanho, endereço e operação), armazenando os valores recebidos localmente em cada operação de escrita.

Na segunda fase, operação, ocorre a escrita no quarto registrador (iniciar), o que causa a chamada de uma das duas funções de apoio, dependendo do código da operação recebida. Ocorrendo o código for 0, será chamada a função transfer_from_mem, que realiza a transferência do código de máquina da tarefa, a partir da memória externa (repositório), para o módulo interface de rede (NI). Se a operação for 1, será chamada a função transfer_to_mem, que realiza a operação inversa: receber, da interface de rede, o código da tarefa e escrevê-lo na memória principal. Uma vez invocada uma das funções, aquela permanecerá em execução até a transferência de todo o código, em paralelo com o processador.

Na última fase, finalização, é gerada (registrada) uma interrupção do DMA no registrador irq_status, localizado no módulo CP0, assim que ocorrer o término da transferência dos dados. Esta interrupção será desativada pelo módulo DMA quando o processador sinalizar o reconhecimento da interrupção através da escrita no quinto registrador, finalizando então a operação de DMA.

6.2.2.3 Módulo NI

O sistema HeMPS é composto basicamente por dois componentes de hardware: elementos de processamento e a rede de comunicação. O elemento de processamento é implementado pelo módulo Plasma, inserido como um núcleo IP. A rede de comunicação é a rede HERMES, a infra-estrutura de comunicação.

Para que os processadores possam se comunicar é necessário um módulo que permita que os dois elementos possam ser interconectados. Este módulo deve servir de interface entre os dois processadores. O módulo que implementa esta interface é o

módulo denominado NI (do inglês Network Interface). O objetivo deste módulo é prover um fluxo de comunicação entre processador e rede e vice-versa. A NI realiza o envio e recebimento dos dados, tanto para o processador como para o DMA. Adicionalmente a NI informa ao processador qual a localização deste na rede, através de seu endereço de rede.

De forma a permitir um melhor entendimento das funções que a NI implementa, segue uma breve descrição de como funciona o mecanismo de comunicação por mensagens. Conforme mencionado anteriormente, a comunicação entre os processadores ocorre por meio de mensagens. Para serem transmitidas estas mensagens devem ser embutidas em uma unidade de comunicação denominada pacote. Cada pacote que trafega na rede Hermes possui três campos distintos:

- header (cabeçalho): contém o endereço de rede do processador destino do pacote;
- size (tamanho do pacote): contém o tamanho total em flits do campo seguinte, de carga útil do pacote;
- payload (carga útil): contém a informação propriamente dita do pacote, a ser transmitida.

O campo de carga útil, por sua vez, é subdividido, no contexto específico da plataforma HeMPS, em duas partes: código de serviço e parâmetros. O código de serviço indica a ação que se deseja executar, os parâmetros correspondem às informações complementares ao serviço. No sistema HeMPS existem atualmente dez serviços disponíveis, resumidos a seguir:

- MESSAGE_REQUEST (código de serviço 10) - Serviço usado para requisitar a leitura de uma mensagem entre processadores escravos, caracterizado por uma leitura em tarefa remota. A execução deste serviço é disparada pela chamada da macro ReadPipe;
- MESSAGE_DELIVERY (código de serviço 20) - Serviço usado como resposta a uma requisição de leitura de mensagem (MESSAGE_REQUEST). Ele transporta a mensagem em si;
- MESSAGE_UNAVAILABLE (código de serviço 30) - Serviço usado como reposta a uma requisição de leitura de mensagem, indicando que a mensagem solicitada não existe;
- TASK_ALLOCATION (código de serviço 40) - Serviço usado para requisitar a alocação estática de uma tarefa em um processador escravo. O serviço transporta o código da tarefa;
- ALLOCATED_TASK (código de serviço 50) - Serviço usado para informar aos processadores escravos que uma nova tarefa foi alocada;

- REQUEST_TASK (código de serviço 60) - Serviço usado para requisitar a alocação dinâmica de uma tarefa. Gerado em duas situações: (i) ao executar a macro WritePipe e, tendo sido finalizada a alocação estática, caso a tarefa alvo ainda não tenha sido alocada e (ii) ao receber a confirmação de conclusão de alocação estática (FINISHED_ALLOCATION), quando existirem requisições de leitura pendentes para uma tarefa alocada no processador alvo;
- TERMINATED_TASK (código de serviço 70) - Serviço usado para informar ao processador mestre o término da execução de uma tarefa. O mestre informará o término aos demais processadores (via o serviço DEALLOCATED_TASK), que irão atualizar suas tabelas de tarefas existentes respectivas;
- DEALLOCATED_TASK (código de serviço 80) - Serviço usado pelo processador mestre para informar os demais processadores que uma tarefa não existe mais (finalizou). Uma tarefa finaliza quando a função main encerra, momento em que será executada a instrução syscall com parâmetro zero, gerando o retorno ao código bootTask.asm que chamou a tarefa;
- FINISHED_ALLOCATION (código de serviço 90) - Serviço usado informar aos processadores escravos que a alocação de tarefas estáticas foi concluída;
- DEBUG (código de serviço 100) - Serviço usado para envio de mensagens de depuração ao processador mestre, o qual envia esta mensagem para o seu módulo UART.

O software do sistema HeMPS, e mais especificamente seu microkernel, não fazem parte dos objetivos de modelagem propostos deste trabalho. Entretanto, para permitir uma melhor compreensão do processo de modelagem do hardware, bem como para dar suporte à validação da funcionalidade do modelo desenvolvido, o microkernel existente será explorado na Seção 6.2.3, ainda que de forma superficial. Maiores detalhes sobre o mesmo podem ser obtidos em [WOS07]. Também, o objetivo de validar o modelo abstrato da HeMPS necessitou que uma parte do microkernel fosse agregado ao modelo, gerando uma validação parcial também do software da HeMPS em alto nível de abstração, conforme será descrito na Seção 7.4.

Como introduzido na Seção 6.1.1, a comunicação entre processadores é feita através de mensagens, via um canal de comunicação denominado pipe. O microkernel faz acesso a pipes para armazenar ou recuperar mensagens, via as funções WritePipe e ReadPipe. Isto vale tanto para tarefas locais quanto remotas.

Na versão atual do sistema HeMPS, o serviço 30 (MESSAGE_UNAVAILABLE) não é gerado. Quando uma leitura de mensagem em uma tarefa remota é requisitada é gerado um pacote contendo o serviço MESSAGE_REQUEST. A tarefa será alterada para o estado de espera (waiting), sendo reativada quando o serviço MESSAGE_DELIVERY for recebido, contendo a identificação da tarefa relativa à mensagem aguardada. A Figura

31 apresenta os seis protocolos usados pelos serviços contidos nas mensagens usadas para a comunicação entre tarefas de processadores distintos.

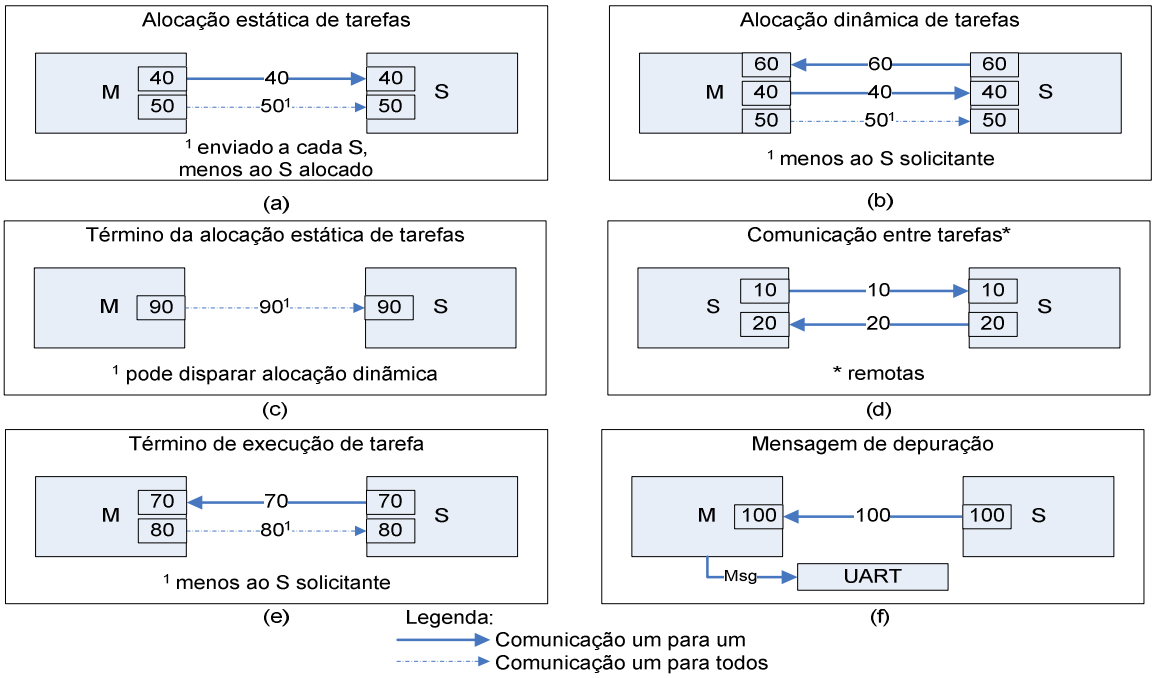

Figura 31 – Protocolo de comunicação usado pelos serviços do microkernel HeMPS, transportados por mensagens na rede. Aqui M corresponde ao processador mestre, S corresponde a algum dos processadores escravos.

O item (a) apresenta o protocolo usado para a alocação estática de tarefas, que ocorre durante o processo de boot do sistema. Para cada tarefa, é gerado e enviado um pacote de serviço 40, contendo o código de máquina da tarefa. Concluída a alocação, outro pacote de serviço 50 é gerado e enviado para cada escravo, menos para aquele onde a tarefa foi alocada.

O item (b) apresenta o protocolo usado para a alocação dinâmica de tarefas, que ocorre sob demanda. Quando uma requisição para alocação de tarefa é gerada, um pacote de serviço 60 e a identificação da tarefa são enviados ao processador mestre. Como resposta, processo semelhante à alocação estática é executado (serviços 40 e 50).

O item (c) apresenta o protocolo usado para indicar o término da alocação estática de tarefas.. Todos os processadores escravos são avisados do término. Neste momento podem ocorrer alocações dinâmicas devido a requisições de mensagem pendentes, geradas para tarefas que ainda não haviam sido alocadas quando da geração da requisição.

O item (d) apresenta o protocolo usado para realizar a comunicação entre tarefas, via troca de mensagens. Um pacote contendo o serviço 10 e a identificação da tarefa onde a mensagem deve ser obtida é enviado ao alvo (processador escravo). No alvo, um pacote contendo o serviço 20 e a mensagem previamente disponibilizada pela macro WritePipe são enviados ao requisitante.

O item (e) apresenta o protocolo usado para informar o término da execução de uma tarefa. No processador escravo onde a tarefa em execução terminou é gerado um pacote, contendo a identificação da tarefa concluída, sendo este enviado ao processador mestre. O mestre por sua vez, gera outro pacote de serviço 80, contendo a identificação da tarefa que terminou e o envia a todos os processadores escravos, menos ao solicitante (onde a tarefa finalizou).

O item (f) apresenta o protocolo usado para mensagens de depuração. Quando uma tarefa enviar uma mensagem de depuração, correspondente à função Echo, esta mensagem será enviada para o processador mestre, juntamente com a identificação da tarefa, sendo posteriormente repassada para o módulo UART do processador mestre.

De forma a permitir a localização facilitada do local onde os serviços são gerados e tratados no microkernel, a Tabela 7 apresenta a correlação entre os serviços de comunicação existentes no sistema HeMPS e as funções relativas a eles, indicando: onde um serviço é gerado e em que local o serviço é tratado, considerando também as duas classes de processadores (mestre/escravo).

Tabela 7 – Correlação entre geração e tratamento de serviços de comunicação. Mestre e Escravo especificam a classe do processador. Gerado significa o local onde é gerado um pacote daquele serviço. Tratado identifica onde o pacote é recebido e tratado. Cód é o código numérico hexadecimal do serviço. Função é o nome da rotina que processa o pacote.

Processador	Mestre		Escravo	
Serviço	Cód	Função	Cód	Função
Gerado	40	DRV_AllocationTask	10	NI_SendMessageRequest
	50	DRV_AllocatedTask	20	NI_SendMessageRequest
	80	DRV_DeallocatedTask	60	NI_SendTaskRequest
	90	DRV_FinishedAllocation	70	NI_SendTaskTerminated
			100	NI_SendDebug
Tratado	60	DRV_Handler	10	Handler_NI
	70	DRV_Handler	20	Handler_NI
	100	DRV_Handler	40	Handler_NI
			50	Handler_NI
			80	Handler_NI
			90	Handler_NI

O mecanismo usado no envio e recebimento de dados pela NI é ilustrado na Figura 32. Nesta Figura existem três componentes básicos:

- portas de entrada e saída: noc_Data_Out e noc_Data_In, que permitem o fluxo de dados em flits entre a NI e a rede HERMES; ni_Data_w e ni_Data_r, que

permitem o fluxo de dados em palavras entre a NI e os componentes internos do Plasma (MLite e DMA);

- registradores de entrada e saída: data_out, que se destina ao envio de dados originados no Plasma para a rede HERMES; data_in, que se destina ao envio de dados provindos da rede HERMES para o Plasma;
- buffer de entrada: dataBuff, constitui-se de uma estrutura tipo fila circular (FIFO), cuja profundidade é parametrizável (por omissão, assume-se uma profundidade de 16 palavras). Neste grupo de registradores são armazenadas as palavras recebidas do registrador data_in. A utilização do buffer se justifica em função do desempenho, evitando gerar interrupções a cada par de flits recebidos (momento em que uma palavra está disponível para ser enviada ao Plasma). Interrupções são geradas apenas quando o buffer estiver cheio, ou quando se detecta o fim de um pacote recebido da rede. Assim, esta abordagem gera número reduzido de interrupções.

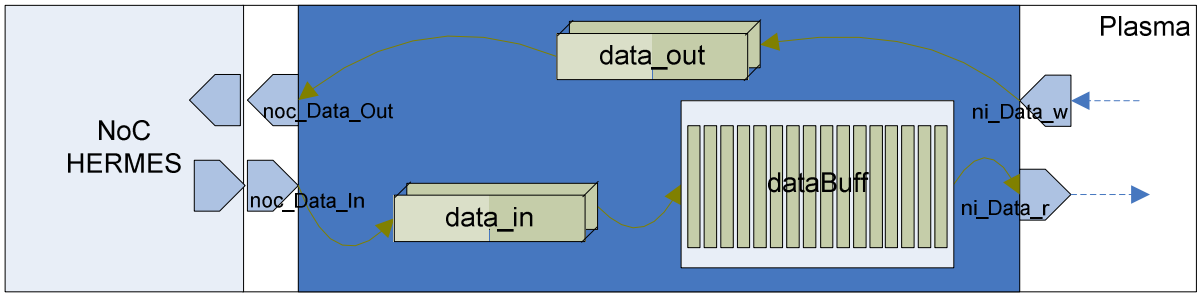

Figura 32 – Componentes da estrutura usada no mecanismo de envio e recebimento de dados através do módulo de interface de rede (NI).

Os dois registradores, data_in e data_out, são definidos em SystemC a partir do uso de uma estrutura do tipo union. Esta estratégia foi escolhida para dar suporte à forma como os dados devem ser recebidos da rede ou enviados para esta: eles são recebidos na forma de flits e armazenados como palavras; são armazenados como palavras e enviados como flits. O tratamento dos flits segue sempre o seguinte padrão: o primeiro flit é armazenado ou retirado da parte alta da palavra e o segundo flit é armazenado ou retirado da parte baixa da palavra. A estrutura union permite atender este mecanismo de forma simples, da seguinte maneira: uma palavra recebida do Plasma é armazenada no registrador data_out em uma única operação e enviada em duas etapas, parte alta e após a parte baixa; dois flits são recebidos da rede e armazenados em duas operações: o primeiro flit é colocado na parte alta do registrador data_in e o segundo flit é colocado na parte baixa, sendo então transferido o seu conteúdo para o buffer de entrada (dataBuff) como uma palavra. A Figura 33 apresenta: a estrutura, a declaração e a atribuição referente ao registrador data_in, para o caso de recebimento de dados provindos da rede.

```
typedef union {
        unsigned long word;
        unsigned short flit[2];        // [0] → LSB (low), [1] → MSB (high)
} dataIN;
. . .
        data_in = new dataIN;
. . .
        noc_Data_In.read(data_in->flit[1]);
        noc_Data_In.read(data_in->flit[0]);
. . .
        dataBuff.write(*data_in);
```

Figura 33 – Estrutura, declaração e forma de leitura relativas à variável data_in, considerando o recebimento de dados provindos da rede.

A Figura 34 mostra a representação esquemática do módulo NI. Nesta Figura podem ser visualizados: o módulo NI ao centro, representado pelo retângulo chanfrado; portas SystemC externas à esquerda, que interconectam o módulo à rede HERMES e por conseqüência ao restante do sistema HeMPS; portas SystemC, à direita, conectadas às portas de interface localizadas na extrema direita, para ligação ao Plasma local.

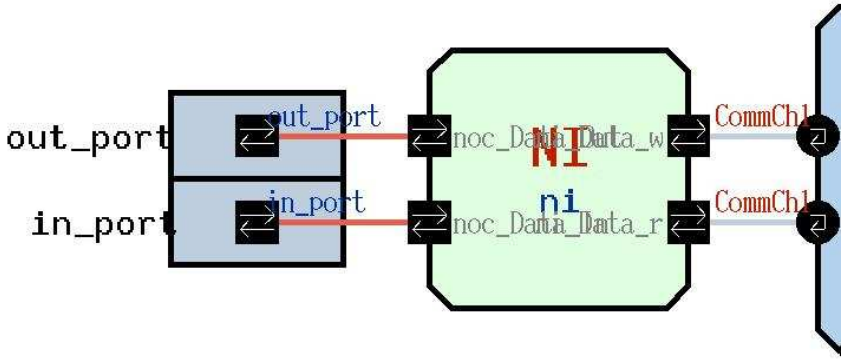

Figura 34 – Módulo de interface de rede (NI).

A funcionalidade do módulo NI é composta de quatro processos, divididos em três grupos, conforme especificado a seguir:

- fluxo de comunicação no sentido do Plasma para a rede HERMES, implementado pelo processo plasma_write_service;
- fluxo de comunicação no sentido da rede HERMES para o Plasma, implementado pelos processos noc_read_service e plasma_read_service;
- controle de geração de interrupções de hardware, implementado pelo processo noc_intr_service.

O processo plasma_write_service recebe uma palavra enviada pelo processador MLite ou pelo DMA e a armazena no registrador data_out. Na seqüência são transmitidos para a rede dois flits, primeiro a parte alta do registrador e logo após a parte baixa.

O processo noc_read_service recebe dois flits da rede e armazena-os na parte alta e parte baixa, respectivamente, do registrador data_in. Logo após, os flits recebidos

são transferidos para dataBuff como uma palavra. O processo plasma_read_service, por sua vez, retira do buffer a palavra armazenada e a envia para o processador MLite ou módulo DMA.

Por fim, o processo noc_intr_service controla a geração de interrupção, de forma a sinalizar ao processador que trate os dados chegando pela rede.

6.2.3 O modelo completo do sistema multiprocessado HeMPS

Uma vez introduzidos os componentes que constituem o sistema HeMPS, é possível apresentar a modelagem completa deste. O módulo HeMPS é composto por três componentes: rede intrachip, núcleo IP e canais de interconexão. A Figura 35 apresenta a distribuição hierárquica dos módulos existentes no modelo abstrato da HeMPS. Na Figura os retângulos com linhas duplas representam módulos hierárquicos, os retângulos com linhas simples representam módulos primitivos e os retângulos com linhas pontilhadas representam canais hierárquicos SystemC.

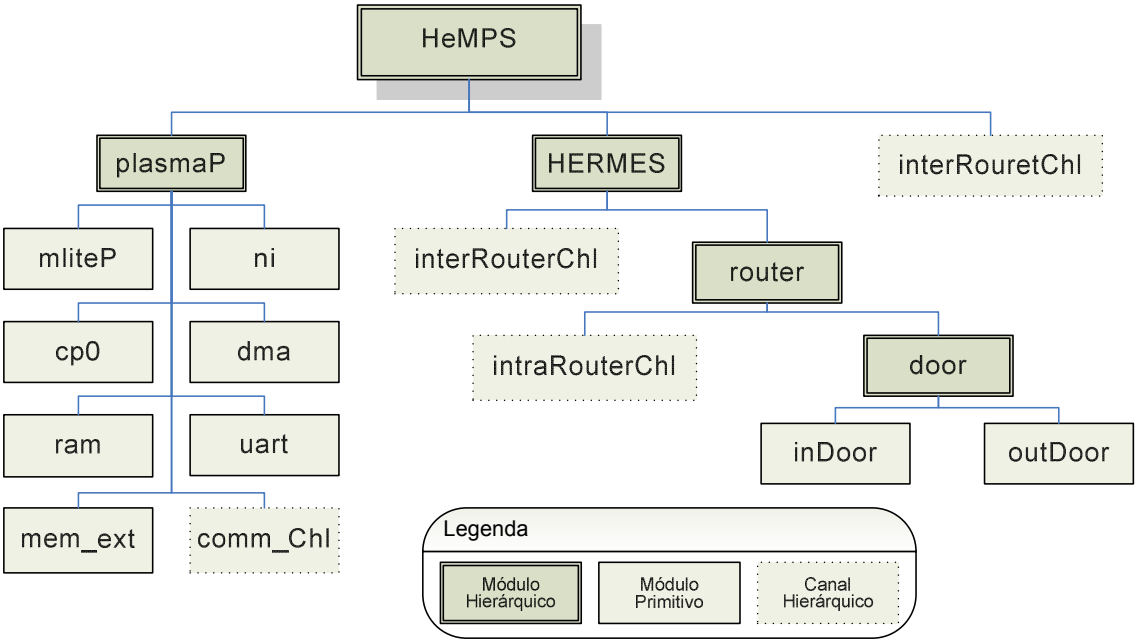

Figura 35 – Organização hierárquica dos módulos componentes do sistema HeMPS.

Na parte superior da hierarquia está o módulo HeMPS, que corresponde à entidade topo de toda a hierarquia. No nível abaixo estão os três módulos fundamentais que compõem o sistema: plasmaP, HERMES e interRouterChl. O módulo plasmaP é o elemento de processamento do sistema, composto por sete módulos primitivos e um canal hierárquico. O módulo HERMES é o elemento de comunicação do sistema, sendo composto por roteadores e canais de interconexão. No nível inferior a este se encontra o módulo router, que corresponde ao elemento de roteamento, composto por portas de comunicação e canais hierárquicos. No nível seguinte aparece o módulo door que implementa os fluxos de comunicação dentro do roteador. Door é composto por dois módulos primitivos, representando as portas de entrada e de saída, que efetivamente

implementam cada fluxo de dados em alguma direção.

A Figura 36 mostra a representação esquemática de uma instância do sistema HeMPS no ambiente System Studio, contendo uma rede HERMES 2x2 e quatro processadores Plasma.

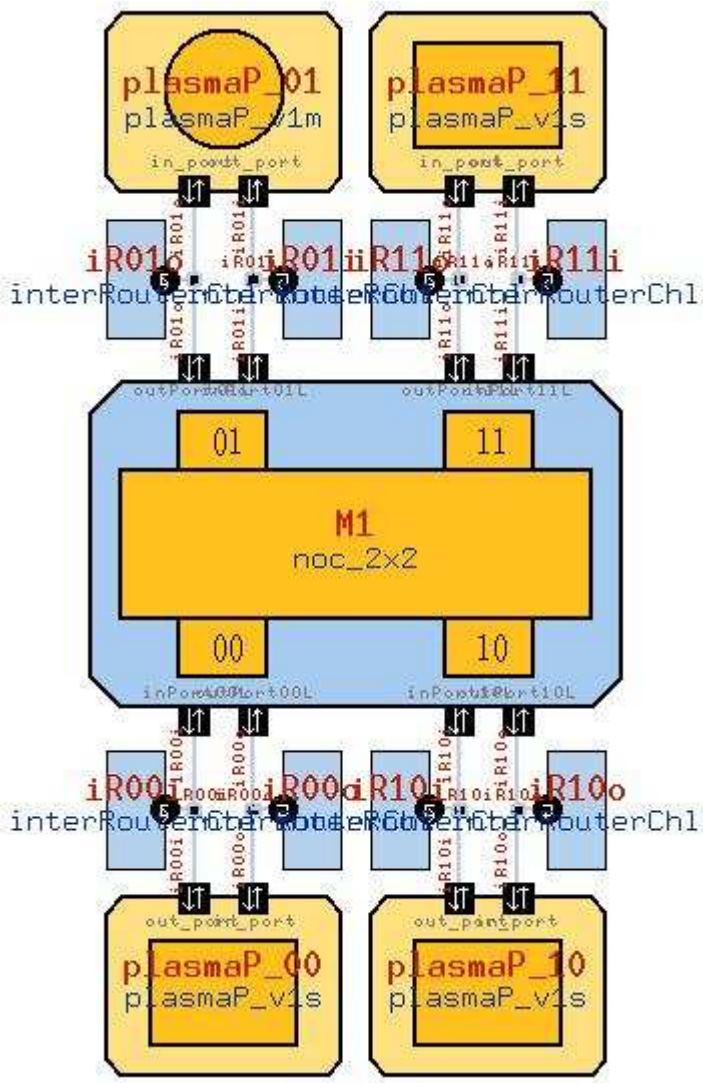

Figura 36 – Uma instância do sistema HeMPS, contendo uma rede HERMES 2x2 e quatro processadores Plasma.

Ao centro está localizada a rede HERMES. Nas extremidades estão localizados quatro módulos processadores Plasma. Um deles é o mestre, identificado por um círculo ao centro, e os demais são escravos, identificados por quadrados no centro do retângulo chanfrado. Entre a rede e os processadores existem canais de interconexão: interRouterChl, que implementam os métodos virtuais da interface.

Existem diversos parâmetros que devem ser configurados para que o sistema possa funcionar adequadamente. O módulo HERMES não possui nenhuma parametrização possível na modelagem realizada no ambiente System Studio. Os módulos plasmaP e seus submódulos, contudo, possui parâmetros especificáveis em tempo de elaboração do modelo. A Tabela 8 detalha os parâmetros existentes,

informando também os valores padrão para cada um deles:

Tabela 8 – Parâmetros e configuração padrão para os módulos plasmaP e seus submódulos.

Módulo	Parâmetro	Comentários	Tipo	Valor Padrão
mliteP	big_endian	Define o formato considerado em operações de memória: *big endian* ou *little endian*. Alterar para falso para little endian.	Lógico	Verdadeiro
	decode_only	Define apenas decodificação do código objeto, não sua execução.	Lógico	Falso
	cycle_show	A cada instrução executada apresenta uma versão *assembly* modificada da instrução.	Lógico	Falso
	print_state	Apresenta o ambiente interno do processador a cada instrução executada	Lógico	Falso
	_show_err	Apresenta mensagens complementares quando um erro em tempo de execução ocorre.	Lógico	Falso
ram	master_ram	Define se o módulo será instanciado para o processador mestre. Alterar para falso em processadores escravos.	Lógico	Verdadeiro
	mem_pages	Indica o número de páginas existentes na memória. Cada página possui 16 Kbytes.	Inteiro	4
	big_endian	Define o formato de armazenamento de dados em memória: *big endian* ou *little endian*.	Lógico	Verdadeiro
mem_ext	master_mem	Define se o módulo será instanciado para o processador mestre. Alterar para falso em processadores escravos.	Lógico	Verdadeiro
	big_endian	Define o formato de armazenamento de dados em memória: *big endian* ou *little endian*.	Lógico	Verdadeiro
	show_header	Apresenta o conteúdo do cabeçalho contido no repositório de tarefas, durante a carga da memória.	Lógico	Verdadeiro
	show_code	Apresenta o código de máquina em hexadecimal contido no repositório de tarefas, durante a carga da memória.	Lógico	Falso
NI	buff_depth	Define o tamanho do buffer para armazenamento de dados que chegam da rede, em nº de palavras.	Inteiro	16

As funções cycle, mem_read e mem_write foram ampliadas no módulo MLite para

dar suporte ao acesso a dispositivos externos mapeados em memória, . O acesso foi implementado através de chamadas de métodos dos canais de comunicação.

7 VALIDAÇÃO E RESULTADOS

7.1 Introdução

A validação dos diversos modelos abstratos apresentados anteriormente constitui uma etapa importante deste trabalho. Este Capítulo apresenta os procedimentos adotados para validar os modelos propostos e para avaliar os resultados obtidos a partir da simulação dos modelos.

As Seções a seguir apresentam o processo de validação do módulo Plasma na Seção 7.2, da NoC HERMES na Seção 7.3 e do sistema multiprocessado HeMPS na Seção 7.4. Por fim, a Seção 7.5 apresenta resultados preliminares obtidos pela simulação do modelo abstrato do processador Plasma.

7.2 Validação do módulo Plasma

Esta Seção está subdividida em duas partes. A primeira apresenta a validação considerando o módulo processador MLite. A segunda parte apresenta a validação do processador considerando a memória como um componente separado. No módulo Plasma, existem outros cinco componentes além do processador e da memória. Considerando que estes componentes se destinam ao suporte do funcionamento do Plasma no escopo do sistema HeMPS, a validação destes componentes será apresentada somente na Seção 7.4.

7.2.1 Módulo MLite

A validação do módulo MLite foi realizada a partir da simulação de aplicações de software compiladas para a arquitetura MIPS-I e executadas sobre dois cenários, ambos implementados no ambiente System Studio.

O primeiro cenário, mostrado na Figura 37, executa a simulação sobre um único processador, denominado mlite1. O objetivo deste cenário foi validar a funcionalidade do módulo quanto à execução de código de máquina. C1 e C2 são canais de comunicação usados para interligar os módulos. M1 representa a instância do processador MLite e data_in e data_out são portas SystemC de entrada e saída. M2 constitui um módulo destinado a registrar os valores produzidos e enviados por M1 de forma a compará-los com os valores previstos. Ambos os módulos são instanciados dentro de um módulo hierárquico.

No cenário da Figura 37, o módulo M1 executa aplicações do tipo produtor/consumidor. Dados são gerados em M1 e consumidos e registrados por M2. A porta SystemC C1 não é utilizada neste cenário.

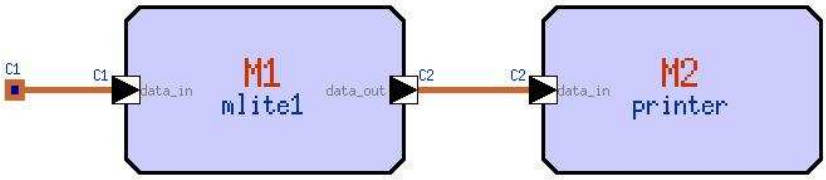

Figura 37 – Cenário de simulação utilizado para validar o módulo MLite no ambiente System Studio.

Duas aplicações foram executadas sobre o modelo monoprocessado:

- um gerador seqüencial de inteiros: esta aplicação simples teve por objetivo a validação inicial do módulo. Trata-se de um contador que gera números inteiros de 32 bits de forma crescente e decrescente. Cada valor gerado é enviado pelo canal C2 e registrado por M2;
- um algoritmo de ordenação tipo bubble sort: esta aplicação teve por objetivo realizar um processamento mais exigente em termos de uso de recursos e tempo de simulação, se comparado ao gerador de inteiros. Inicialmente, atribui-se uma seqüência de valores decrescentes a um vetor de inteiros, cujo tamanho é parametrizável no código C. Cada valor do vetor é enviado pelo canal C2 e registrado por M2. O vetor é então ordenado em ordem inversa e seus valores novamente enviados e registrados por M2. Esta aplicação foi executada considerando diversas variações, tanto no tamanho do vetor quanto no número de ordenações executadas.

O segundo cenário, mostrado na Figura 38, usa duas instâncias do modelo abstrato do processador. Este cenário tem por objetivo testar a comunicação entre processadores, funcionalidade posteriormente utilizada quando um número maior de instâncias é inserido no sistema HeMPS. C1, C2 e C3 representam os canais de comunicação, M1 e M2 representam instâncias dos processadores MLite e M3 o modelo destinado a registrar os valores recebidos de M2. Todos os módulos são instanciados dentro de um módulo hierárquico. M1 executa uma aplicação do tipo produtor/consumidor e M2 executa uma aplicação do tipo consumidor. Os dados produzidos em M1 são consumidos por M2, que então os envia a M3 para serem mostrados.

Figura 38 – Cenário de validação da comunicação entre dois processadores MLite.

Sobre este cenário foi executada uma aplicação gerador/consumidor seqüencial de inteiros. Esta aplicação teve como objetivo criar um fluxo de comunicação entre dois módulos distintos do processador, aspecto fundamental para validar a comunicação entre processadores. Compreende dois programas executados em paralelo nos dois processadores: um produtor e um consumidor. O produtor, executado em M1, gera

números inteiros de 32 bits de forma crescente e decrescente e os envia ao segundo programa através do canal C2. Os valores recebidos por M2 são repassados por este processador ao módulo M3 onde são registrados pela aplicação consumidora que executa neste módulo.

7.2.2 Módulo de memória

A validação do módulo de memória foi realizada executando os mesmos programas da versão com um processador. Este terceiro cenário pode ser visto na representação esquemática já introduzida na seção 4.2.2, Figura 18.

O principal objetivo desta validação foi garantir que a execução do código fosse realizada corretamente, fazendo uso do módulo de memória externa ao processador conectado ao último através de um canal de comunicação.

7.3 Validação da infra-estrutura de comunicação HERMES

A validação do módulo HERMES foi realizada através da inserção de módulos do tipo produtor/consumidor nas portas locais dos roteadores.

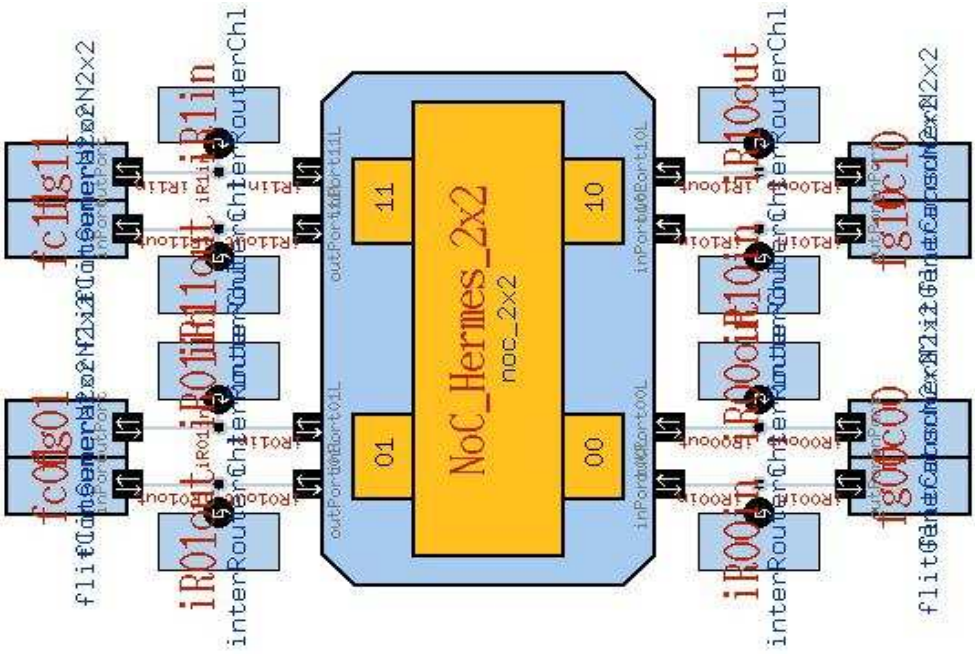

Figura 39 – Rede HERMES 2x2 instanciada junto com quatro pares de módulos produtor/consumidor.

A Figura 39 mostra a representação esquemática de uma rede HERMES 2x2, juntamente a quatro módulos produtores e quatro consumidores conectados às portas locais de cada roteador através de canais hierárquicos. Ao centro da Figura encontra-se a rede HERMES, com 4 roteadores. Nas extremidades estão os pares de módulos produtor/consumidor interligados à rede através de canais hierárquicos. Cada módulo produtor monta um pacote contendo dados de teste e envia estes a cada consumidor

localizado em outro ponto da rede. Cada módulo consumidor por sua vez recebe e registra os pacotes enviados pelos demais produtores. Este mecanismo permite gerar um cenário de comunicação onde cada produtor envia pacotes de dados para todos os demais consumidores e cada consumidor recebe pacotes de todos os produtores, garantindo com isto a validação da comunicação entre todos os roteadores existentes na rede.

7.4 Validação do sistema multiprocessado HeMPS

Duas abordagens foram consideradas para realizar a validação do sistema HeMPS. A primeira é através da escrita de trechos de código especificamente criados para validar partes ou conjuntos de partes do sistema. A segunda é através do uso do próprio microkernel, usado como software de validação dos diversos módulos do sistema. A escolha final recaiu sobre o uso do microkernel. Os principais motivos que determinaram esta escolha foram:

- a existência prévia deste software, desenvolvido especificamente para o sistema HeMPS;
- o fato do microkernel já ter sido executado sobre a versão RTL original da HeMPS;
- a possibilidade de validar o modelo abstrato desenvolvido neste trabalho, a partir de software já validado previamente;

O processo de validação da versão abstrata do sistema HeMPS foi realizado em duas etapas, descritas nas Seções a seguir.

7.4.1 Validação do módulo HeMPS: primeira etapa

A primeira etapa tem por objetivo realizar a validação da funcionalidade dos submódulos do elemento de processamento Plasma. Executa-se a fase inicial do microkernel (boot), tanto para o processador mestre como para os escravos. Também considera-se uma aplicação exemplo compondo o repositório de tarefas, consistindo de um conjunto de quatro tarefas numeradas de 0 a 3. A alocação das tarefas foi realizada de forma estática como segue: tarefa 0 no processador 10, tarefa 1 no processador 11, tarefa 2 no processador 20 e tarefa 3 no processador 21. O processador 01 executa o microkernel mestre e o processador 00 não é usado.

A Tabela 9 apresenta o resumo das operações executadas pelo microkernel mestre durante o boot, sob a forma de uma lista de tarefas. Esta etapa de validação executa com sucesso todas as operações desta tabela, com exceção da operação 11, uma vez que o objetivo foi validar o processo de boot e não a execução de tarefas em si.

Tabela 9 – Lista de tarefas da fase inicial do microkernel mestre.

Nº	Arquivo	Descrição da operação
1	boot.S	Ajusta os valores iniciais dos registradores de pilha fp e sp, e do registrador que aponta para a área de dados estáticos, gp.
2	boot.S	Atribui o valor zero para toda a área de memória que contém dados estáticos.
3	boot.S	Invoca a execução da função principal, main.
4	kernel.c	Atribui valores iniciais às estruturas de dados: free_pages, task_locations e processors.
5	kernel.c	Invoca a execução da função InitializeProcessors, que registra o número (endereço) dos processadores na estrutura processors.
6	kernel.c	Para cada tarefa existente, invoca a execução da função InitializeTasks, que registra a identificação da tarefa e o endereço de rede do processador onde esta será executada na estrutura task_locations.
7	kernel.c	Invoca a execução da função TaskAllocation que, para cada registro existente em task_locations (uma tarefa), envia ao processador alvo da tarefa um pacote contendo o serviço 40 e o código objeto da tarefa via o driver DRV_AllocationTask (boot.S). Adicionalmente envia aos demais processadores um pacote contendo o serviço 50 junto à identificação e endereço do processador da tarefa alocada.
8	kernel.c	Invoca a execução do driver DRV_FinishedAllocation, que envia um pacote contendo o serviço 90 para cada processador escravo existente, indicando o fim da alocação estática de tarefas.
9	kernel.c	Invoca a execução da função OS_InterruptMaskSet, que insere o bit referente à interrupção da NI no registrador máscara de interrupção, irq_mask (20000010) do módulo CP0.
10	kernel.c	Invoca a execução da função OS_AsmInterruptEnable, que ativa o registrador SR ($12) do módulo CP0, indicando que o processador mestre está apto a tratar interrupções.
11	kernel.c	Inicia a execução de um laço sem conteúdo e condição de parada, correspondendo à tarefa ociosa (idle task).

Para facilitar a análise e validação do processador mestre, a simulação é dividida em duas partes. Na primeira parte, o módulo Plasma mestre executa dois procedimentos:

- load_code, implementado no módulo RAM, carrega o microkernel mestre na memória principal;
- load_task_code. implementado no módulo MEM_EXT, carrega na memória externa (repositório) uma imagem contendo o cabeçalho e código de máquina das tarefas.

A Figura 40 apresenta o processo de registro da captura gerada pela primeira parte da simulação, que corresponde aos dois procedimentos citados. Na parte superior da Figura pode ser vista a indicação de início da simulação. Logo a seguir observa-se a saída gerada pelo módulo MEM_EXT, onde se mostra os três campos do cabeçalho de cada tarefa: identificação, tamanho e endereço inicial do código de máquina. Embora o espaço de endereçamento deste módulo esteja entre 0x10000000 e 0x1fffffff, o endereçamento local inicia na posição de memória 0x00000000. Por exemplo, o endereço 0x00000030 hachurado na Figura 40 indica o início do código de máquina das tarefas, logo após o seu cabeçalho. Isto se deve ao fato de que internamente ao receber um acesso ao endereço 0x10000030, deste será subtraído 0x10000000, gerando o endereço local usado na estrutura de memória do módulo MEM_EXT. Na parte inferior pode ser vista a confirmação da carga de 1108 palavras a partir do arquivo repository.txt, imagem da memória externa (repositório), indicando as quatro tarefas da aplicação selecionada. Na seqüência nota-se a carga de 1483 palavras referente ao código de máquina do microkernel mestre.

```
================================================
(CPU) Plasma Instruction Set Simulator (ISS) - Starting ...

----------------------------------------
(MEM_EXT) load tasks code  - Starting ...

TaskID   Size     InitAddr (hex)
-------- -------- --------
00000000 00000149 00000030
00000001 00000191 00000554
00000002 000000cb 00000b98
00000003 000000a3 00000ec4

(MEM_EXT) Read 4 task(s) (1108 words) from 'repository.txt' - Big
Endian Selected

(MEM_EXT) load tasks code  - ... finished
----------------------------------------

-----------------------------------
(RAM) load Kernel code - Starting ...

(RAM) Read 1483 words from 'kernel_master.txt' - Big Endian Selected

(RAM) load Kernel code - ... finished
-----------------------------------
```

Figura 40 – Saída gerada na primeira parte da simulação do módulo Plasma mestre: carga na memória externa (MEM-EXT) das quatro tarefas da aplicação selecionada e carga do microkernel na memória principal (RAM).

Concluída a primeira parte da simulação, inicia-se a segunda parte onde o processador MLite passa a executar o microkernel mestre. Neste momento, começa o processamento das operações listadas na Tabela 9. A execução das operações 7 e 8 em particular geraram grande fluxo de dados a partir do processador mestre, correspondendo à distribuição das tarefas para os processadores escravos. O fluxo de dados produzido é registrado de forma a comparar os dados enviados com os valores previstos, o que

permite validar a execução a segunda parte da simulação. A Figura 41 e a Figura 42 apresentam a captura do fluxo de dados gerado pelas operações 7 e 8, respectivamente.

```
1.  (NoC)    10        298 - flit seq: 2
2.  (NoC)     0         40 - flit seq: 4
3.  (NoC)     0          0 - flit seq: 6
4.  (NoC)     0        149 - flit seq: 8
5.  (NoC) 3c1c          0 - flit seq: a
6.  (NoC) 379c          0 - flit seq: c
...
7.  (NoC) 6973       6865 - flit seq: 298
8.  (NoC) 642e          0 - flit seq: 29a
...
9.  (NoC)    11        328 - flit seq: 2b4
10. (NoC)     0         40 - flit seq: 2b6
11. (NoC)     0          1 - flit seq: 2b8
12. (NoC)     0        191 - flit seq: 2ba
13. (NoC) 3c1c          0 - flit seq: 2bc
14. (NoC) 379c          0 - flit seq: 2be
...
15. (NoC) 6973       6865 - flit seq: 5da
16. (NoC) 642e          0 - flit seq: 5dc
...
```

```
17. (NoC) 6973       6865 - flit seq: 8f6
18. (NoC) 642e          0 - flit seq: 8f8

    (NI) Header received from CPU, sent
19. (NoC)    10         6 - flit seq: 8fa
20. (NoC)     0        50 - flit seq: 8fc
21. (NoC)     0        21 - flit seq: 8fe
22. (NoC)     0         3 - flit seq: 900

    (NI) Header received from CPU, sent
23. (NoC)    11         6 - flit seq: 902
24. (NoC)     0        50 - flit seq: 904
25. (NoC)     0        21 - flit seq: 906
26. (NoC)     0         3 - flit seq: 908

    (NI) Header received from CPU, sent
27. (NoC)    20         6 - flit seq: 90a
28. (NoC)     0        50 - flit seq: 90c
29. (NoC)     0        21 - flit seq: 90e
30. (NoC)     0         3 - flit seq: 910
```

Figura 41 – Fluxo de dados gerado pela execução da operação 7 da Tabela 9.

```
    (NI) Header received from CPU, sent
1.  (NoC)    10         2 - flit seq: 912
2.  (NoC)     0        90 - flit seq: 914

    (NI) Header received from CPU, sent
3.  (NoC)    11         2 - flit seq: 916
4.  (NoC)     0        90 - flit seq: 918

    (NI) Header received from CPU, sent
5.  (NoC)    20         2 - flit seq: 91a
6.  (NoC)     0        90 - flit seq: 91c
```

```
    (NI) Header received from CPU, sent
7.  (NoC)    21         2 - flit seq: 91e
8.  (NoC)     0        90 - flit seq: 920

9.  (CPU) Plasma Instruction Set
    Simulator (ISS) - ... finished
    ===================================

10. 0.14user 0.24system 0:12.47elapsed
    3%CPU
```

Figura 42 – Parte da saída gerada pela execução da operação 8 da Tabela 9.

Para facilitar a compreensão do fluxo gerado, a Tabela 10 e a Tabela 11 apresentam respectivamente uma descrição dos conteúdos gerados pelas operações 7 e 8.

Cada uma das Tabelas possui 3 colunas: a primeira indica a linha ou linhas correspondentes da Figura 41 ou da Figura 42; a segunda coluna indica o valor ou valores assumidos para cada flit representado em hexadecimal; a terceira coluna introduz um comentário sobre o significado do flit ou flits referenciados.

Reticências indicam a existência de uma seqüência contendo mais de dois de flits. Na Figura 41 e na Figura 42 reticências indicam uma seqüência de flits omitidos; os números de linhas não correspondem à seqüência de flits enviados, mas sim à seqüência de linhas apresentadas nas Figuras.

Tabela 10 – Descrição do conteúdo dos dados enviados à NoC pela operação 7 da Tabela 9.

Linhas	Valor(es)	Comentário
1	10 e 298	*Header (target)*: identificação do processador alvo, para onde será enviado o pacote neste caso para o processador escravo de número 10. *Header (size)*: quantidade de *flits* a serem enviados no campo *payload*, resultado de: 6 + code size, (tamanho do cabeçalho) + (tamanho do código objeto x 2).
2	0 e 40	*Payload (service)*: código do serviço para alocação estática de tarefa.
3	0 e 0	*Payload (task id)*: identificação da tarefa que está sendo enviada para alocação, neste caso a tarefa de número 0 (zero).
4	0 e 149	*Payload (task size)*: tamanho do código de máquina em número de palavras.
5 a 8	3c1c ... 0	*Payload (task code)*: código de máquina da tarefa 00, identificada na linha 3.
9	11 e 328	*Header: size* e *target;* porém para a tarefa 01.
10 a 12	0 ... 191	Payload: service, task id e task size; porém para a tarefa 01.
13 a 16	3c1c ... 0	*Payload: task code*, porém para a tarefa 01.
17 e 18	6973 ... 0	Últimos dois *flits* do p*ayload (task code)* da última tarefa enviada para alocação, no caso a 3.
19	10 e 6	*Header: target* e *size* do pacote gerado depois de finalizado o envio de cada tarefa para alocação, neste caso para a última tarefa (3).
20	0 e 50	*Payload (service):* serviço usado para informar que uma tarefa foi alocada em um processador escravo. Este pacote é enviado pelo processador mestre a todos os processadores escravos, menos àquele onde a tarefa foi alocada, para que a tabela local de tarefas alocadas de cada processador seja atualizada.
21	0 e 21	*Payload*: indica em que processador a tarefa foi alocada, no caso o 21.
22	0 e 3	*Payload*: indica o número da tarefa que foi alocada, no caso a 3.
23 a 26	11 ... 3	O mesmo conteúdo das linhas 19 a 22, porém enviado para o processador 11.
27 a 30	20 e 6	O mesmo conteúdo das linhas 19 a 22, porém enviado para o processador 20

A Figura 42 apresenta o fluxo de dados gerado pela operação 8 e a Tabela 11 a descrição dos dados deste fluxo.

Tabela 11 – Descrição do conteúdo de dados enviados para a NoC para a operação 8 da Tabela 9.

Linha	Valor(es)	Comentário
1	10 e 2	*Header (target)*: identificação do processador alvo para onde será enviado o pacote, neste caso para o processador escravo de número 10 *Header (size)*: quantidade de *flits* a serem enviados no campo *payload.*
2	0 e 90	*Payload (service)*: serviço usado para informar o fim da alocação estática de

		tarefas.
3 e 4	**11 ... 90**	**O mesmo conteúdo das linhas 1 e 2, porém enviado para o processador 11.**
5 e 6	**20 ... 90**	**O mesmo conteúdo das linhas 1 e 2, porém enviado para o processador 20.**
7 e 8	**21 ... 90**	**O mesmo conteúdo das linhas 1 e 2, porém enviado para o processador 21.**
9	**–**	**Fim do processo de simulação da primeira fase de simulação.**
10	**–**	**Tempo consumido e utilização da CPU pela simulação nesta etapa.**

A Tabela 12 apresenta o resumo das operações executadas pelo microkernel escravo durante o boot, sob a forma de uma lista de tarefas.

Tabela 12 – Lista de operações referente à execução do processo inicial do microkernel escravo.

Nº	Arquivo	Descrição da operação
1	**boot.S**	**Ajusta os valores iniciais dos registradores de pilha fp e sp, e do registrador que aponta para a área de dados estático, gp.**
2	**boot.S**	**Atribui o valor zero para toda a área de memória que contém dados estáticos.**
3	**boot.S**	**Invoca a execução da função principal, main.**
4	**kernel.c**	**Invoca a execução da função *ASM_SetInterruptEnable* (boot.S), que desativa (insere 0) no registrador SR ($12) do módulo CP0, indicando que o processador não irá tratar interrupções.**
5	**kernel.c**	**Invoca a execução da função *NI_Init* (ni.c), que atribui zeros a duas variáveis de controle.**
6	**kernel.c**	**Invoca a execução da função *OS_Init*, que realiza as seguintes funções:** **→ obtém os endereços dos registradores *gp* e *sp* e os armazena na estrutura *system_tcb*;** **→ define a variável *current* que é um ponteiro para o TCB da tarefa *idle* e define o seu PC como o endereço inicial desta tarefa;** **→ armazena em *net_address* o endereço do nodo da rede a partir do acesso ao módulo NI;** **→ atribui valores iniciais às estruturas: pipe_slot, task_location, tcbs, task_requests e request_message;** **→ invoca a execução da função *OS_InterruptMaskClear*, que desativa (escreve 0) todos os bits do registrador *irq_mask* (20000010) do módulo CP0, ou seja, desativa o reconhecimento de todas as interrupções;** **→ registra o endereço das funções que irão tratar as três interrupções de hardware possíveis quando estas foram registradas (*Timer*, *DMA* e *NI*);** **→ invoca a execução da função *OS_InterruptMaskSet*, que ativa (escreve 1) nos**

		bits do registrador *irq_mask* (20000010) do módulo CP0 correspondentes às três interrupções de hardware possíveis, ou seja, ativa o reconhecimento de todas as interrupções.
7	**kernel.c**	**Invoca a execução da função ASM_RunSchedulerTask (boot.S) passando como parâmetro o ponteiro current (endereço da tarefa Idle), que realiza o seguinte:** **→ acessa o TCB da tarefa current para definir o escalonamento;** **→ restaura o contexto da tarefa selecionada, no caso Idle;** **→ desvia a execução do processador para a 1ª instrução da tarefa escalonada.**

Todas as operações da Tabela 12 são executadas com sucesso nesta etapa de validação, exceto a operação 7, uma vez que o objetivo é validar o processo de boot e não a execução das tarefas.

Para facilitar a análise e validação do processador escravo, a simulação foi dividida em duas partes. Na primeira parte da simulação o processador mestre executa o procedimento load_code, implementado no módulo RAM que carrega o microkernel escravo na memória principal;

A Figura 43 apresenta o registro da primeira parte da simulação, que corresponde ao procedimento citado acima. Na parte superior da Figura pode ser vista a indicação de início da simulação. Logo a seguir, observa-se a saída gerada pelo módulo RAM, indicando a carga de 4680 palavras, referente ao código de máquina do microkernel escravo. A carga do repositório de tarefas não é executada por nenhum processador escravo, uma vez que apenas o mestre tem acesso a este repositório.

```
============================================================
(CPU) Plasma Instruction Set Simulator (ISS) - Starting ...

-------------------------------------
(RAM) load Kernel code - Starting ...

(RAM) Read 4680 words from 'kernel_slave.txt' - Big Endian selected

(RAM) load Kernel code - ... finished
-------------------------------------

(CPU) Plasma Instruction Set Simulator (ISS) - ... finished
============================================================

0.13user 0.21system 0:10.36elapsed 3%CPU
```

Figura 43 – Saída gerada pelo procedimento inicial executado pela simulação do processador escravo: carga do microkernel.

Concluída a primeira parte da simulação, inicia-se a segunda parte, onde os processadores escravo passam a executar o microkernel escravo. Neste momento, inicia-se o processamento das operações da Tabela 12. A execução destas operações não gera fluxo de dados do processador escravo para a NoC, apenas processamento local.

Concluída a execução de todas as operações, a simulação termina, o que pode

ser notado no destaque da parte inferior da Figura 43. Por fim, a última linha apresenta o tempo consumido e a utilização de CPU por esta etapa da simulação.

Esta Seção apresentou o processo de validação do módulo HeMPS, considerando tanto o processador mestre como os processadores escravo juntamente com trechos do microkernel relacionado a cada um deles. Esta primeira fase de validação permitiu verificar a funcionalidade de seis dos sete módulos existentes no módulo Plasma. A exceção foi o módulo UART, que é usado apenas quando uma tarefa em um dos processadores escravos envia uma mensagem de depuração ao mestre. A Figura 44 apresenta a representação esquemática dos módulos mestre e escravo usados para a primeira etapa da validação. O módulo mestre está conectado a um módulo Loger usado para gerar o registro (log) do fluxo de dados que seria enviado à NoC HERMES.

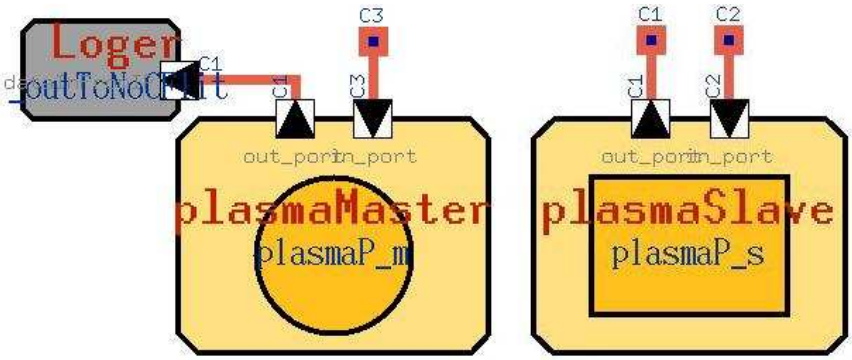

Figura 44 – Módulos Plasma mestre e escravo, usados na primeira fase de validação.

7.4.2 Validação do módulo HeMPS: segunda etapa

A segunda etapa de validação tem por objetivo simular a funcionalidade da infra-estrutura de rede HERMES em conjunto com os elementos de processamento Plasma. Esta etapa é executada fazendo uso de parte do microkernel, o mesmo usado na fase anterior, incluindo também a mesma aplicação de teste escrita para ser usada como repositório de tarefas. A Figura 45 mostra a estrutura geral do modelo usado nesta etapa. Esta estrutura encontra-se em processo de validação e será alvo de relatório futuro.

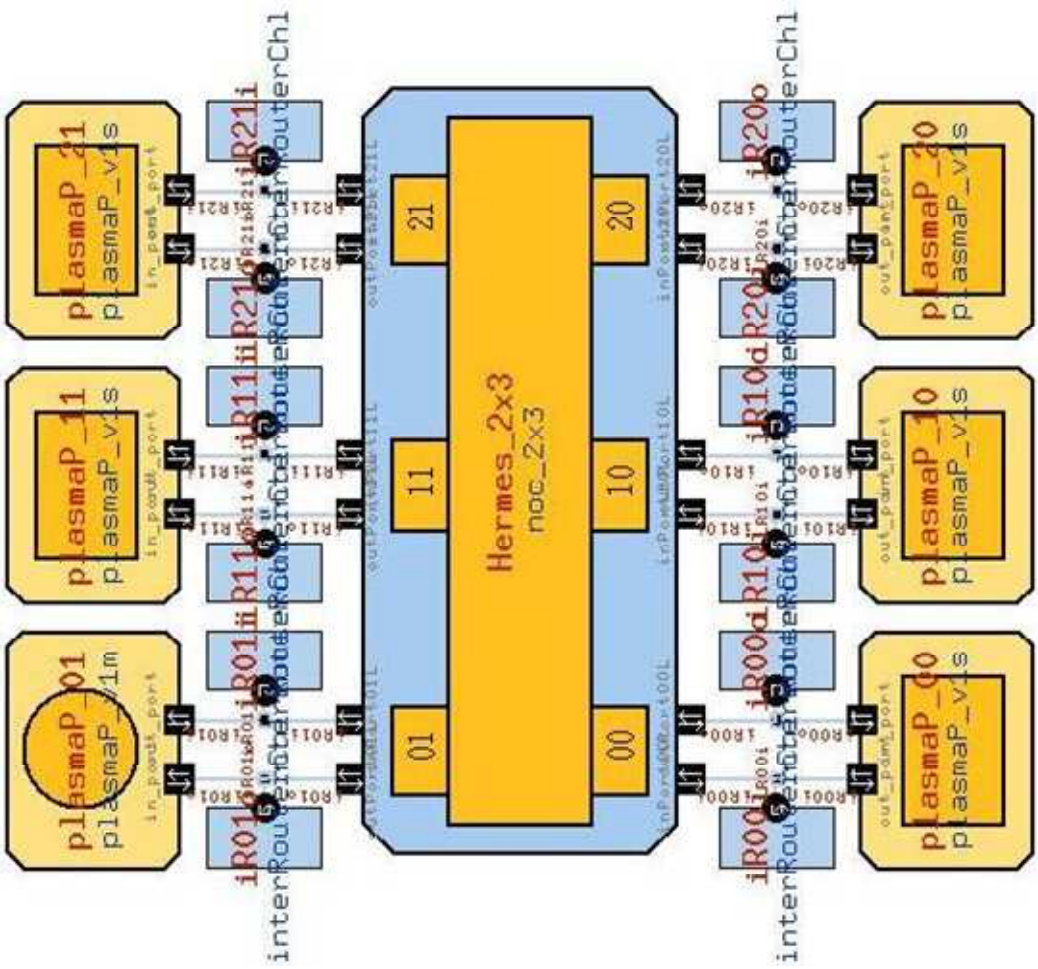

Figura 45 – Módulo HeMPS de dimensão 2 x 3 composto de uma rede HERMES 2x3 e seis processadores Plasma, um mestre e cinco escravos.

7.5 Resultados quantitativos preliminares

Foram realizadas diversas simulações do modelo abstrato do Plasma, a fim de compará-lo à descrição RTL existente do Plasma. O desempenho foi medido considerando o tempo total consumido para realizar uma simulação completa. Uma simulação completa é aquela que executa todos os procedimentos de uma aplicação. O final da execução do código e conseqüentemente da simulação ocorre quando uma instrução break é reconhecida.

As Figura 46 e Figura 47 mostram gráficos comparativos de simulações executadas respectivamente a partir da versão RTL do projeto Plasma e da versão abstrata produzida no escopo deste trabalho.

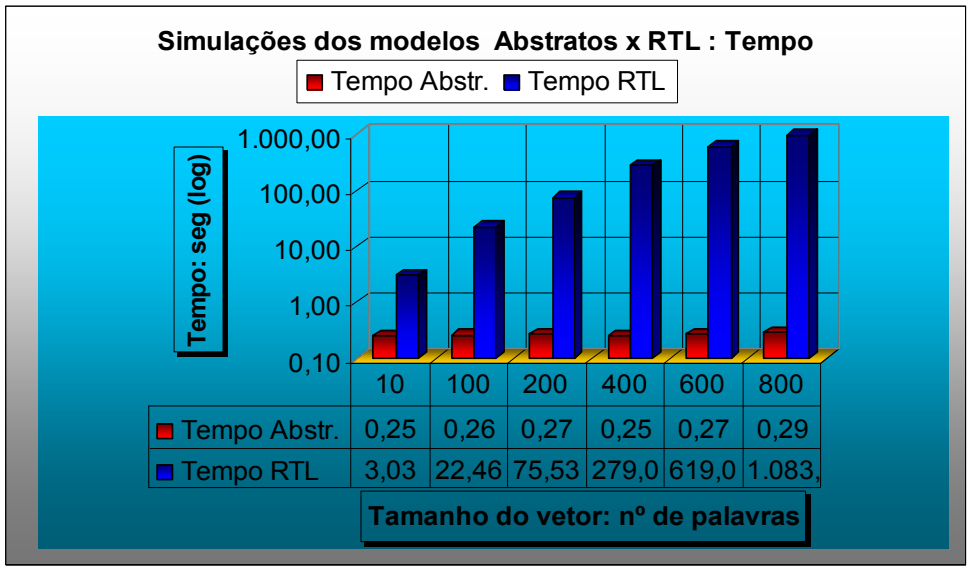

Figura 46 – Comparação de tempos de simulação entre um modelo abstrato do Plasma baseado em ISS e um modelo RTL. Simulação abstrata realizada no ambiente System Studio, e simulação RTL realizada no ambiente Modelsim da Mentor.

A aplicação selecionada para ser executada sobre as duas simulações foi um algoritmo de ordenamento bubble sort. O tamanho do vetor foi variado de 10 a 800 elementos, sendo que a ordenação foi executada duas vezes em cada aplicação. A Figura 46 mostra o tempo de simulação necessário para executar a aplicação completa. A Figura 47 mostra o percentual médio de CPU utilizado para realizar cada simulação na máquina hospedeira.

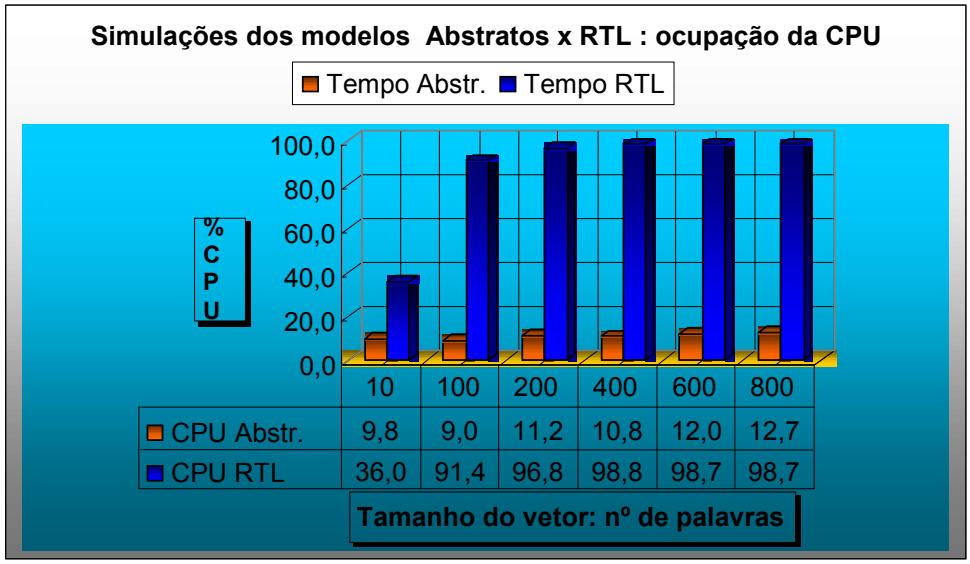

Figura 47 – Comparação ocupação de CPU entre um modelo abstrato do Plasma baseado em ISS e um modelo RTL. Simulação abstrata realizada no ambiente System Studio, e simulação RTL realizada no ambiente Modelsim da Mentor.

Os recursos computacionais usados nas simulações são:

- Sistema operacional Red Hat Enterprise Linux WS release 4;
- Hardware: processador Intel Xeon 3.80GHz e memória RAM DDR2 4 GBytes;

- Ferramentas de software: Synopsys CoCentric System Studio e Mentor ModelSim.

A partir dos gráficos, pode-se concluir que o tempo de simulação no modelo abstrato permaneceu relativamente estável em todas as simulações. O mesmo não ocorreu com a simulação do modelo RTL, o que já era esperado. No melhor caso, vetor com 10 elementos, o tempo de simulação do modelo RTL foi 12 vezes maior que o modelo abstrato, chegando a ser 3698 vezes maior para o pior caso, o vetor de 800 elementos.

Outro aspecto a ser destacado é o nível de ocupação que a simulação gerou sobre o ambiente hospedeiro. A maior taxa ocorrida para o modelo abstrato foi de aproximadamente 13% contra uma taxa próxima a 100% exigida pela simulação RTL.

O menor tempo de simulação e menor uso da CPU se deram pelo aumento do nível de abstração, ao custo da perda de detalhes existentes na descrição RTL. Para os estágios inicias da exploração da arquitetura e para o desenvolvimento antecipado do software do sistema, o modelo abstrato é claramente mais apropriado.

8 CONCLUSÕES

8.1 Conclusões e contribuições

A principal contribuição do presente trabalho foi disponibilizar um modelo funcional parametrizável, descrito em alto nível de abstração, do hardware do sistema multiprocessado HeMPS existente no grupo de pesquisa do autor.

O modelo foi desenvolvido fazendo uso do ambiente comercial para desenvolvimento de sistemas System Studio da Synopsys. Este ambiente permite realizar, com relativa facilidade, a simulação comportamental do sistema de referência, a partir do modelo capturado via descrições arquiteturais implementados em linguagem SystemC, o que insere maior produtividade na fase inicial de projeto.

Os dois componentes básicos do sistema, o elemento de processamento e a rede intrachip, foram modelados e integrados, sendo usados para o processo de validação do sistema HeMPS. Parte do software básico da HeMPS foi usado para validar o modelo desenvolvido.

O Plasma modelado neste trabalho compreende efetivamente uma infra-estrutura de processamento composta por: (i) processador MLite, (ii) interface de rede, (iii) controlador de acesso direto à memória, (iv) memória principal e (v) co-processador de controle. Todos os módulos que compõem o processador foram implementados e testados, restando apenas alguns ajustes e validações a serem realizadas no módulo interface de rede com relação ao tratamento de interrupções.

A infra-estrutura de rede HERMES foi integrada e validada mediante a criação de um cenário contendo pares produtor-consumidor em cada nodo da rede, os quais injetaram dados originados e destinados a todos os roteadores existentes, para configurações de rede 2x2 e 3x3.

Os resultados obtidos com a simulação são considerados promissores. Por exemplo, a simulação do módulo Plasma descrita neste trabalho chegou a atingir ganhos de desempenho em termos de tempo de simulação na faixa de três ordens de magnitude em relação a um modelo RTL clássico do mesmo processador.

Quanto ao modelo como um todo, pretende-se atingir a validação completa pela execução total do software do sistema, compreendendo microkernel e tarefas da aplicação. No escopo deste trabalho, foi considerado o processo inicial do sistema operacional (boot), como critério para realizar a validação do modelo abstrato.

8.2 Trabalhos futuros

A seguir apresentam-se algumas propostas para trabalhos futuros a realizar:

1. Explorar possibilidades de generalização dos modelos desenvolvidos. Esta generalização poderá permitir a geração automática de modelos a partir de parametrizações dos modelos abstratos criados, além de permitir variar cenários da arquitetura a partir de parametrizações.
2. Explorar a possibilidade de implementação de MPSoCs contendo elementos de processamento heterogêneos.
3. Explorar formas de realizar mapeamento e migração de tarefas, de maneira a flexibilizar o modelo abstrato desenvolvido.
4. Explorar a modelagem mista de componentes, com relação ao nível de abstração. Esta abordagem pode levar a definir cenários mais adequados a cada etapa do desenvolvimento para diferentes variações da arquitetura. A idéia é fazer evoluir cada um dos modelos não temporizados de módulos desenvolvidos aqui para modelos parcialmente temporizados, precisos em nível de ciclos e combinações destes tipos de modelos para as principais partes do sistema HeMPS.

REFERÊNCIAS BIBLIOGRÁFICAS

[ANG06] Angiolini, F.; Ceng, J.; Leupers, R.; Ferrari, F.; Ferri, C.; Benini, L. “An Integrated Open Framework for Heterogeneous MPSoC Design Space Exploration”. In: Conference on Design, Automation and Test in Europe (DATE’06), 2006, pp. 1145-1150.

[ARC08] ArchC. “The ArchC Architecture Description Language”. Capturado em: http://www.archc.org, Julho 2008.

[BEL08] Beltrame, G.; Bolchini, C.; Fossati, L.; Miele, A.; Sciuto, D. “ReSP: A Non-Intrusive Transaction-Level Reflective MPSoC Simulation Platform for Design Space Exploration”. In: Design Automation Conference Asia and South Pacific (ASPDAC'08), 2008, pp. 673-678.

[BEL08b] Beltrame, G.; Fossati, L.; Sciuto, D. “High-Level Modeling and Exploration of Reconfigurable MPSoCs”. In: NASA/ESA Conference on Adaptive Hardware and Systems(AHS’08), 2008, pp. 330-337.

[BEN05] Benini, L.; Bertozzi, D.; Bogliolo, A.; Menichelli, F.; Olivieri, M. “MPARM: Exploring the Multi-Processor SoC Design Space with SystemC”. Journal of VLSI Signal, 41(2), July 2005, pp. 169-182.

[CAI03] Cai, L.; Gajski, D. “Transaction Level Modeling: An Overview”. In: 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS’03), 2003, pp. 19-24.

[CAR09] Carara, E. A.; de Oliveira, R. P.; Calazans, N. L. V.; Moraes, F .G. “HeMPS - A Framework for Noc-Based MPSoC Generation”. In: IEEE International Symposium on Circuits and Systems (ISCAS´09), 2009, pp. 1345-1348.

[DÖM08] Dömer, R.; Gerstlauer, A.; Peng, J.; Shin, D.; Cai, L.; Yu, H.; Abdi, S.; Gajski, D. D. “System-on-Chip Environment: A SpecC-Based Framework for Heterogeneous MPSoC Design”. Journal on Embedded Systems (EURASIP'08), 2008, Junho 2008. 13p.

[DON04] Donlin, A. “Transaction Level Modeling: Flow and Use Models”. In: 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS’04), 2004, pp. 75-80,

[GAP08] GAPH - Hardware Design Suport Group. “GAPH Homepage”. Capturado em: http://www.inf.pucrs.br/~gaph, Julho 2008.

[GAP08a] GAPH - Hardware Design Suport Group. “ATLAS - An Environment for NoC Generation and Evaluation”. Capturado em: http://www.inf.pucrs.br/ ~gaph/ATLASHtml/ATLASIndex_us.html, Julho 2008.

[GRO02] Grötker, T.; Liao, S.; Martin, G.; Swan, S. “System Design with SystemC”. Kluwer Academic Publishers, 2002, 236p.

[HEN03] Henkel, J. “Closing the SoC Design Gap”. IEEE Computer, 36(9), September 2003, pp. 119-121.

[IND08] Indrusiak, L. S.; Ost, L.; Möller, L.; Moraes, F.; Glesner, M. “Applying UML Interactions and Actor-oriented Simulation to the Design Space Exploration of Network-on-Chip Interconnects”. In: International Symposium on VLSI (ISVLSI’08), 2008, pp. 491-494.

[JAN04] Jang, H.; Kang, M.; Lee, M.; Chae, K.; Lee, K.; Shim, K. “High-Level System Modeling and Architecture Exploration with SystemC on a Network SoC: S3C2510 Case Study”. In: Conference on Design, Automation and Test in Europe (DATE’04), 2004, pp. 538-543.

[JER05] Jerraya, A.; Wolf, W. “Multiprocessor Systems-on-Chips”. Morgan Kaufmann Publishers, Amsterdam, 2004, 608 p.

[JER05a] Jerraya, A.; Tenhunen, H.; Wolf, W. “Guest Editors’ Introduction: Multiprocessors Systems-on-Chips”. IEEE Computer, 38(7), July 2005, pp. 36-40.

[KAN91] Kane, G.; Heinrich, J. “MIPS RISC Architecture - 2nd Edition”. Prentice-Hall, 1991, 544 p.

[KOG02] Kogel, T; Bussaglia, D. “SystemC Based Design of an IP Forwarding Chip with CoCentric System Studio”. In: Synopsys Users Group (SNUG’02), 2002, 8p.

[LEE07] Lee, S.; Yoon, S.; Lee, J.; Huang, M. L.; Park, S. “Transaction Level Model Simulator for NoC-based MPSoC Platform”. In: 6th WSEAS International Conference on Instrumentation, Measurement, Circuits and Systems (WSEAS’07), 2007, pp. 170-174.

[THE08] The MathWorks, Inc. “Simulink Home Page”. Capturado em: http://www.mathworks.com/products/simulink/, Julho 2008.

[MAA08] Maalej, I.; Gogniat, G.; Philippe, J. L.; Abid, M. “System level design space exploration for multiprocessor system on chip”. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI’08), 2008, pp. 93-98.

[MOR04] Moraes F. G.; Calazans, N. L. V.; Mello, A. V.; Möller, L.; Ost, L. “HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip”. Integration The VLSI Journal, 38(1), October 2004, pp. 69-93.

[MOR04a] Moreno, E. I. “Modelagem, Descrição e Validação de Redes Intrachip no Nível de Transação”, Dissertação de Mestrado, PPGCC-FACIN-PUCRS, 2004, 137 p.

[MOR08] Moreno, E. I.; Popovici, K. M.; Calazans, N. L. V.; Jerraya, A. A. “Integrating Abstract NoC Models within MPSoC Design”. In: International Symposium on Rapid System Prototyping (RSP’08), 2008, pp. 65-71.

[NIK08] Nikolov, H.; Stefanov, T.; Deprettere, E. “Systematic and Automated Multiprocessor System Design, Programming, and Implementation”. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(3), March 2008, pp. 542-555.

[OST08] Ost. L.; Moraes, F. G.; Möller, L.; Indrusiak. L. S.; Glesner, M.; Määttä, S.; Nurmi, J. “A Simplified Executable Model to Evaluate Latency and Throughput of Networks-on-chip”. In: Proceedings of the Symposium on Integrated Circuits and System Design (SBCCI’08), 2008, pp. 170-175.

[PAT05] Patterson, D. A.; Hennessy, J. L. “Organização e Projeto de Computadores, 3ª Edição”. Editora Campus, 2005, 484 p.

[PET07] Petry, C. A. “Projeto Eletrônico em Nível de Sistema: Um Estudo da Ferramenta System Studio da Synopsys”. Trabalho Individual II, PPGCC-FACIN-PUCRS, 2007, 72 p.

[POP07] Popovici, K.; Jerraya, A. “Simulink based Hardware-Software Codesign Flow for Heterogeneous MPSoC”. In: 2007 Summer Computer Simulation Conference (SCSC’07), 2007, pp. 497-504.

[RHO07] Rhoads, S. “Plasma - most MIPS I(TM) opcodes: Overview”. Capturado em: http://www.opencores.org/projects.cgi/web/mips/ overview, Novembro 2007.

[SWE06] Sweetman, D. “See MIPS Run”. Morgan Kaufmann Publishers, 2006, 513 p.

[SYN06] Synopsys, Inc. “System Studio User Guide - Version 2006.12-SP1”, Guia do Usuário, 2006, 595 p.

[SYN08] Synopsys, Inc. “DesignWare Home Page”. Capturado em: http://www.synopsys.com/products/designware/designware.html, Julho 2008.

[SYN08a] Synopsys, Inc. “System Studio Home Page”. Capturado em: http://www.synopsys.com/products/sls/system_studio/ system_studio. html, Julho 2008.

[TIB07] Tibboel, W.; Reyes, V.; Klompstra, M.; Alders, D. “System-Level Design Flow Based on a Functional Reference for HW and SW”. In: 44th Annual Conference on Design Automation (DAC'07), 2007, pp. 23-28.

[WOS07] Woszezenki, C. R. “Alocação de Tarefas e Comunicação entre Tarefas em MPSoCS”. Dissertação de Mestrado, PPGCC-FACIN- PUCRS, 2007, 121 p.