PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE ENGENHARIA
FACULDADE DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

DESENVOLVIMENTO DE UM AMBIENTE DE EXECUÇÃO DE APLICAÇÕES EMBARCADAS PARA A PLATAFORMA MULTIPROCESSADA HEMPS

CEZAR RODOLFO WEDIG REINBRECHT
GERSON SCARTEZZINI
THIAGO RAUPP DA ROSA

Porto Alegre
13 de Julho de 2009

CEZAR RODOLFO WEDIG REINBRECHT

GERSON SCARTEZZINI

THIAGO RAUPP DA ROSA

DESENVOLVIMENTO DE UM AMBIENTE DE EXECUÇÃO DE APLICAÇÕES EMBARCADAS PARA A PLATAFORMA MULTIPROCESSADA HEMPS

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Engenheiro de Computação da Faculdade de Engenharia da Pontifícia Universidade Católica do Rio Grande do Sul.

ORIENTADOR: PROF. DR. FERNANDO GEHM MORAES

Porto Alegre

13 de Julho de 2009

AGRADECIMENTOS

Eu, Cezar Rodolfo Wedig Reinbrecht, gostaria de agradecer, principalmente, à minha mãe e irmã, que sempre tiveram paciência e me apoiaram durante todas as etapas da minha vida. Além disso, gostaria de agradecer a todos os amigos que estiveram presentes durante minha formação acadêmica, sendo de grande ajuda e importância.

Eu, Gerson Scartezzini, gostaria de agradecer aos meus pais por todo o empenho dedicado a me propiciar a oportunidade de estudar, aos meus irmãos pela paciência e ajuda durante momentos de tensão. Além disso, agradeço aos meus avós e padrinhos pelo apoio e incentivo dado em toda a minha caminhada até este momento.

Eu, Thiago Raupp da Rosa, gostaria de agradecer especialmente minha mãe e irmã que foram quem sempre estiveram ao meu lado, me ajudando e me dando forças sempre que precisei. Aos meus tios, Nilton e Marta, que me ajudaram em um período difícil de minha vida, no início da faculdade, e continuam me ajudando até hoje. Também à minha namorada, que teve paciência e soube entender os momentos que precisei me dedicar integralmente para que esse trabalho fosse concluído, e não foram poucos.

O “Grupo CTG” agradece a Deus, pelas oportunidades que nos deu. A todos os colegas da faculdade, que não estão se formando conosco, mas que também estiveram sempre presentes em nossas vidas. Agradecemos também aos membros do grupo GAPH que nos ajudaram em momentos que precisamos de conhecimento técnico mais aprofundado. Também aos avaliadores Ney Calazans e César Marcon, pela disponibilidade e paciência de corrigir e avaliar nosso trabalho.

Agradecemos em especial ao Moraes, pela confiança, ajuda e orientação que nos deu em todas as etapas do desenvolvimento desse trabalho.

Reunir-se é um começo, permanecer juntos é
um progresso, e trabalhar juntos é um sucesso.

Henry Ford

RESUMO

Sistemas Multiprocessados em um único chip (MPSoC) estão sendo cada vez mais utilizados em sistemas embarcados. Devido à complexidade e grande espaço de projeto destes sistemas, ferramentas de CAD e frameworks para o desenvolvimento de projetos são requisitos básicos.

A principal contribuição deste Trabalho de Conclusão é apresentar o ambiente para desenvolvimento e avaliação de sistemas multiprocessados denominada HeMPS Station. HeMPS Station é um ambiente dedicado para MPSoC e seu conceito define uma estrutura de sistema, capaz de avaliar o desempenho de aplicações embarcadas distribuídas em determinada arquitetura executando em um FPGA. Esse ambiente contém ferramentas dedicadas executando em um hospedeiro, interface de comunicação rápida entre hospedeiro e MPSoC para viabilizar a avaliação do sistema durante a execução, e uma estrutura de monitoração inserida no MPSoC, permitindo a captura de dados de desempenho e de depuração.

Para a infra-estrutura de hardware, utilizou-se três arquiteturas de referência, sendo elas a Plataforma HeMPS, a Plataforma ComEt e o Sistema ConMe, sendo as duas últimas completamente desenvolvidas neste trabalho. O atual estado da Plataforma HeMPS consiste em uma plataforma (rede-intrachip, processadores, DMAs e NIs), softwares embarcados (microkernel e aplicações) e uma ferramenta CAD dedicada para gerar os códigos-objeto necessários e possibilitar depuração pós-simulação.

Este projeto vem para suprir a tendência de que aplicações comuns estão adquirindo cada vez mais complexidade, principalmente no ramo do entretenimento, e que diferentes áreas estão sendo automatizadas, necessitando poder de processamento. Futuramente, sistemas baseados nesses dispositivos poderão gerenciar recursos, controlar sistemas industriais ou residenciais ou até mesmo auxiliar computadores de qualquer natureza a realizar o processamento de aplicações específicas em diferentes campos da ciência.

Palavras-chave: aplicações embarcadas, MPSoCs, sistemas multiprocessados, ferramentas CAD, ambiente de depuração.

ABSTRACT

Multi-Processor Systems-on-Chip (MPSoCs) are increasingly popular in embedded systems. Due to their complexity and huge design space to explore for such systems, CAD tools and frameworks to customize MPSoCs are mandatory.

The main contribution of this end-of-term project is to present the environment for development and evaluation of multiprocessors systems named HeMPS Station. HeMPS Station is a dedicated MPSoC environment whose concept encompasses a structure that allows to evaluate distributed embedded applications running in a given architecture in an FPGA. This environment includes dedicated tools running in a host, a fast communication link between host and MPSoC enabling system evaluation and a monitoring structure to capture performance and debug data.

For the hardware infrastructure, three reference architectures are used: HeMPS Platform, ComEt Platform and ConMe System. The last two platforms were completely developed in the context of this work. The present state of the HeMPS MPSoC includes the platform (NoC, processors, DMA, NI), embedded software (microkernel and applications) and a dedicated CAD tool to generate the required binaries and perform post-simulation debugging.

This Project comes to supply the trend that common applications are acquiring even more complexity, mainly in entertainment field. Besides, different areas have being automated, requiring power processing. In a near future, systems based in these devices will manage resources, control industrial and domestic systems or support any computer to process specific applications in different science areas.

Keywords: embedded applications, MPSoCs, multiprocessors systems, CAD tools, debug environment.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AL	Additional Latency
API	Application Programming Interface
ARCNET	Attached Resource Computer Network
ARP	Address Resolution Protocol
bps	Bits per second
BRAM	Block RAM
CAD	Computer Aided Design
CC-NUMA	Cache-Coherent Non-Uniform Memory Access
COMA	Cache-Only Memory Access
CAN	Controller Area Network
CAS	Column Address Strobe
ComEt	Comunicação Ethernet
ConMe	Conexão com Memória
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA-CD	Carrier Sense Multiple Access with Collision Detection
DCE	Data Communication Equipment
DCR Bus	Device Control Register Bus
DDR	Double Data Rate
DDR2	Double Data Rate 2
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DSM	Distributed Shared Memory
DSP	Digital Signal Processing
DTE	Data Terminal Equipment
DVD	Digital Vídeo Disk
EMAC	Embedded Media Access Control
E/S	Entrada e Saída
FCS	Frame Check Sequence
FDDI	Fiber Distributed Data Interface
FIFO	First In First Out

FPGA	Field Programmable Gate Array
GAPH	Grupo de Apoio ao Projeto de Hardware
GB	Giga Byte
Gbps	Giga bits per second
GPP	General Purpose Processor
HD	Hard Disk
HeMPS	Hermes Multiprocessor System
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISS	Instruction Set Simulator
ISO	International Standards Organization
JPEG	Joint Photographic Experts Group
Kbit	Kilo bits
KB	Kilo bytes
LAN	Local Area Network
LED	Light Emitting Diode
LLC	Logical Layer Control
LUT	LookUp Table
MAC	Media Access Control
MB	Mega Byte
Mbps	Mega bits per second
MHz	Mega Hertz
MIG	Memory Interface Generator
MIPS	Microprocessor without Interlocked Pipeline Stages
MPI	Message Passing Interface
MPSoC	Multiprocessing System on a Chip
NDIS	Network Driver Interface Specification
NI	Network Interface
NoC	Network-on-Chip
NORMA	NO-Remote Memory Access
NUMA	Non-Uniform Memory Access
ODT	On Die Termination

OSI-RM	Reference Model for Open Systems Interconnection
PCI-e	Peripheral Component Interconnect-express
pH	Potencial Hidrogeniônico
PVM	Parallel Virtual Machine
RAM	Random Access Memory
RFC	Request for Comments
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SATA	Serial Advanced Technology Attachment
SDRAM	Synchronous Dynamic Random Acess Memory
SDR	Single Data Rate
SoC	System on a Chip
SO-DIMM	Small Outline Dual in-line Memory Module
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
UMA	Uniform Memory Access
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large-Scale Integration
WAN	Wide Area Network

SUMÁRIO

INTRODUÇÃO

Single processors may be sufficient for low performance applications that are typical of early microcontrollers, but an increasing number of applications require multiprocessors to meet their performance goals.

Ahmed Jerraya

Os avanços tecnológicos das últimas décadas [ITR07] propiciaram a evolução de diversas áreas da ciência. Problemas científicos complexos puderam ser modelados, gerando assim aplicações com alto custo computacional. Muitos destes problemas são tão complexos que sistemas monoprocessados convencionais gastariam um tempo muito elevado para atingir o completo processamento da aplicação. Primeiramente, foram propostos como solução supercomputadores, cujas arquiteturas possuem alta velocidade de processamento e grande capacidade de memória. No entanto, essas características acarretam um alto custo financeiro, tornando sua utilização restrita.

Observando tais aplicações, nota-se uma característica comum a todas elas, a viabilidade de particioná-las em tarefas menores. Esta característica propiciou uma nova proposta de solução, que é a distribuição destas tarefas em diferentes sistemas monoprocessados, interagindo de maneira a executar a aplicação completa. A princípio, este método de processamento paralelo foi adotado na forma de *clusters*. Essa seria uma nova etapa evolutiva, pois o custo financeiro diminuiu, enquanto que o poder computacional pôde ser mantido ou mesmo superado.

Seguindo esta tendência evolutiva de diminuir o custo dos recursos sem perder o poder computacional, um novo patamar de hardware pôde ser almejado. A evolução nas tecnologias VLSI (do inglês, *Very Large-Scale Integration*), tornou possível aumentar a densidade dos circuitos integrados, e assim, favorecer uma nova estratégia de arquitetura, o desenvolvimento de sistemas multiprocessados em um único chip (MPSoC, do inglês *Multiprocessor System-on-Chip*) [TAN06] [JER04] [JER05] [WOL05]. Um MPSoC consiste de uma arquitetura composta por recursos heterogêneos, incluindo múltiplos processadores embarcados, módulos de hardware dedicados, memórias e um meio de interconexão [WOL04]. Portanto, é um projeto complexo com alto poder computacional, possibilitando o desenvolvimento de dispositivos para fins de processamento de aplicações distribuídas.

Devido à disponibilidade de processamento de alto desempenho, os MPSoCs estão sendo também utilizados em aplicações embarcadas. Isso se deve dada a tendência de que

aplicações comuns estão adquirindo cada vez mais complexidade, principalmente no ramo do entretenimento, e que diferentes áreas estão sendo automatizadas, necessitando poder de processamento. Futuramente, sistemas baseados nesses dispositivos poderão gerenciar recursos, controlar sistemas industriais ou residenciais ou até mesmo auxiliar computadores de qualquer natureza a realizar o processamento de aplicações específicas em diferentes campos da ciência.

Este Trabalho de Conclusão de Curso tem por objetivo desenvolver uma estação de trabalho multiprocessada para executar aplicações embarcadas. O principal objetivo é possibilitar a execução de aplicações de níveis de complexidade e áreas distintas. Dessa forma, visa-se dar suporte aos projetistas no desenvolvimento de arquiteturas multiprocessadas, mapeamentos heurísticos e escalonadores de tarefas, monitoração de características em tempo de execução, entre outros. Estes objetivos foram alcançados através do desenvolvimento de ambiente baseado na plataforma multiprocessada HeMPS [WOS07][CAR09], denominado HeMPS Station [REI09]. A HeMPS Station executa aplicações reais e apresenta seus resultados em tempo de execução, possibilitando ajustar as características da aplicação em menos tempo e com maior precisão.

O presente documento está organizado da seguinte maneira. O Capítulo 1 apresenta o referencial teórico. O Capítulo 2 descreve as arquiteturas que são definidas como referência para o ambiente HeMPS Station, que são a Plataforma HeMPS, a Plataforma ComEt e o Sistema ConMe. O Capítulo 3 expõe a integração dos módulos de referência para se alcançar o objetivo deste projeto. O Capítulo 4 explana o ambiente HeMPS Station. O Capítulo 5 apresenta a validação do ambiente.

1 REFERENCIAL TEÓRICO

System-level design and
extensible processors can bridge
the gap between silicon technology
and actual SoC complexities.
Jörg Henkel

Neste Capítulo é apresentado o embasamento teórico do trabalho proposto. Nele são citadas as aplicações embarcadas e os conceitos dos principais temas a serem desenvolvidos neste trabalho, sendo eles: os sistemas multiprocessados intra-chip, o armazenamento de dados em memórias DDR2 SDRAM e a comunicação de rede.

A primeira seção descreve os sistemas embarcados, bem como as aplicações. A segunda define o conceito de sistemas computacionais de alto desempenho. A terceira seção apresenta o conceito de sistemas multiprocessados em um único chip. A quarta embasa as características da memória alvo. A última seção define conceitos básicos de redes de computadores e expõe as características do controlador MAC provido pela empresa *Xilinx*.

1.1 Aplicações Embarcadas

Segundo dados estimados por pesquisas em alta tecnologia, mais de 90% dos microprocessadores fabricados mundialmente são destinados a máquinas que usualmente não são chamadas de computadores. Entre estes dispositivos pode-se citar: telefones celulares, fornos microondas, controladores automotivos, tocadores de DVD e computadores. O que diferencia este conjunto de dispositivos de um computador convencional é o seu projeto baseado em um conjunto dedicado e especialista constituído por hardware, software e periféricos – um sistema embarcado [REC04].

A denominação embarcada (em inglês, embedded system) vem do fato de que estes sistemas geralmente estão inseridos em ambientes não eletrônicos [GUP96]. As principais características de classificação deste sistema são a sua capacidade computacional e a sua independência de operação. Outros aspectos relevantes dependem dos tipos de sistemas, modos de funcionamento e itens desejados em aplicações embarcadas.

Sistemas embarcados estão inseridos hoje em todo tipo de domínio de atuação humana. Exemplos destas áreas são ilustrados na Figura 1: monitoramento médico, automação industrial e residencial, sistemas automotivos, eletrodomésticos, aparelhos de multimídia, periféricos, celulares, entre outros.

Figura 1 - Exemplos de Dispositivos com Sistemas Embarcados

A seguir, discutem-se algumas das áreas de emprego de sistemas embarcados, visando ilustrar os diferentes compromissos de desempenho, custo e emprego destes elementos.

1.1.1 Setor Automotivo

Centenas de sensores fornecem informações sobre todo o funcionamento de um automóvel moderno. Unidades de processamento distintas e independentes atuam em regiões diferentes e se comunicam entre si, captando os sinais de sensores e fazendo com que as ações referentes a cada caso sejam tomadas.

Esta comunicação geralmente se dá através de redes, onde o protocolo CAN tem se tornado o padrão. Isto acontece desde a central que pode memorizar a posição dos bancos, espelhos e volante para cada usuário do veículo até a central que gerencia o correto funcionamento do motor.

1.1.2 Aquisição de Dados – Data Logger

Consistem de sistemas que através de sensores (temperatura, umidade, pH e outros) capturam informações do ambiente e as armazenam em memória para consultas posteriores. O Sistema além de monitorar o ambiente, pode com adição de atuadores ao projeto, ter a capacidade de controlar as variáveis do ambiente com base em um critério estabelecido pelo projetista do sistema.

1.1.3 Sistemas de Controle

Geralmente são aplicações mais robustas, com placas dedicadas e múltiplos sensores de entrada e atuadores de saída. Muitas vezes fornecem pouca interação com o usuário, mostrando sinalizações através de diodos emissores de luz, LEDs (do inglês: *Light Emitting Diode*). Usados nos motores de automóveis, processos químicos, controladores de vôo, usinas nucleares, aplicações aeroespaciais e monitoramento e controle de variáveis de ambiente (temperatura, umidade, pH do ar).

1.1.4 Processamento de Sinais

Envolve um grande volume de informação a ser processada em curto espaço de tempo. Os sinais a serem tratados são digitalizados através de conversores Analógico/Digital, processados e novamente convertidos em sinais analógicos por conversores Digital/Analógico. Exemplos de processamento de sinais: tratamento de áudio, filtros, modems, compressão de vídeo, radares e sonares, etc.

1.1.5 Comunicações, Redes e TV Digital

Hubs, Switchs e Roteadores são dotados de microprocessadores e de microcontroladores para controle digital de sinais. Na TV Digital estes controladores digitais têm um núcleo para processamento digital de sinais, instalado na antena e no receptor da TV Digital, com objetivo de selecionar o melhor foco do canal e eliminar sinais ruidosos.

1.2 Sistemas Computacionais de Alto Desempenho

Baseado nos conceitos descritos na seção anterior surge a idéia de sistemas multiprocessados, nos quais cada núcleo de hardware pode ser um processador de propósito geral (*General Purpose Processor* - GPP). Teoricamente, a junção de N processadores pode conduzir a uma melhoria do desempenho em N vezes, atingido uma capacidade de processamento superior a qualquer sistema monoprocessado. Pode se definir duas classes gerais de multiprocessadores: de memória compartilhada e de troca de mensagens [HWA93] [KUM94] [PAT96], discutidas a seguir.

1.2.1 Multiprocessadores de memória compartilhada

Nos multiprocessadores de memória compartilhada, todos os processadores compartilham um espaço de endereçamento global, o qual pode ser fisicamente centralizado ou distribuído. A sincronização e a comunicação entre os processos de uma mesma aplicação ocorrem através do acesso a variáveis compartilhadas em memória. O compartilhamento de memória pode ser realizado segundo os modelos UMA, NUMA ou COMA.

No modelo UMA *(Uniform Memory Access)*, os múltiplos processadores são interligados por meio de uma rede de interconexão a uma memória global centralizada. Essa memória central é formada por módulos disjuntos, os quais podem ser acessados independentemente um do outro por diferentes processadores da máquina. Para reduzir o tráfego na rede, a cada processador pode ser associada uma memória local privativa para o armazenamento de dados locais e instruções de programa. Essas máquinas recebem esse nome porque o tempo de acesso à memória compartilhada é igual para todos os processadores.

No modelo NUMA *(Non-Uniform Memory Access)*, cada processador possui uma memória local, a qual é agregada ao espaço de endereçamento global da máquina. Dessa forma, podem existir até três padrões de acesso à memória compartilhada. O primeiro, e o mais rápido, é aquele onde a variável compartilhada está localizada em uma memória local do processador. O segundo padrão refere-se ao acesso a um endereço na memória central. Já o terceiro, e o mais lento, diz respeito ao acesso a uma posição localizada em uma memória local de outro processador. Dois modelos alternativos de máquina NUMA são mostrados na Figura 2.

O modelo mostrado na Figura 2b é também chamado de multiprocessador de memória compartilhada distribuída (DSM - *Distributed Shared Memory*), pois toda a memória do sistema é distribuída entre os processadores da máquina, não havendo uma memória central.

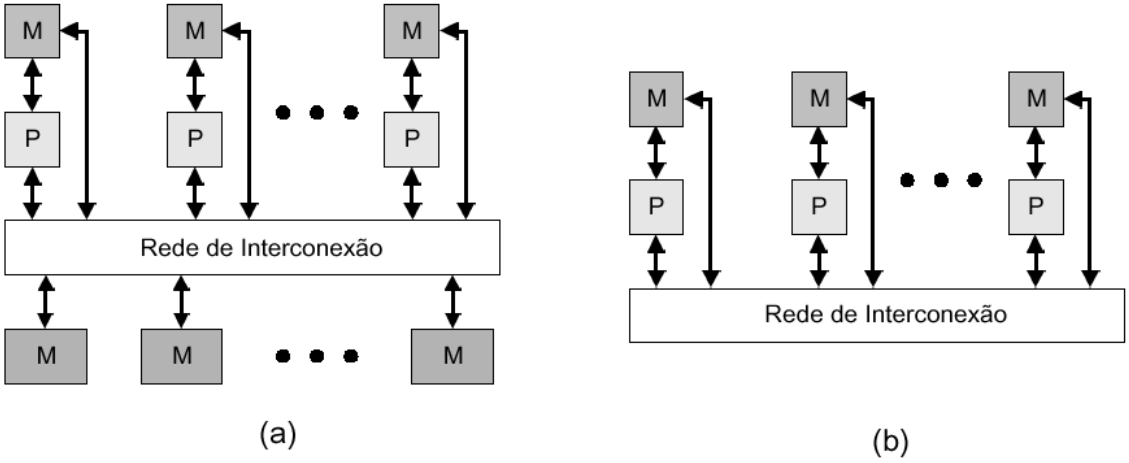

Figura 2 - Modelo NUMA: (a) com memória central (b) sem memória central

As máquinas COMA (Cache-Only Memory Access) são um caso particular das arquiteturas NUMA onde as memórias locais dos processadores são convertidas em caches. Todas as caches formam o espaço de endereçamento global e o acesso as caches remotas é auxiliado por meio de diretórios distribuídos. Já as máquinas CC-NUMA (Cache-Coherent Non-Uniform Memory Access) utilizam memória compartilhada distribuída e diretórios de cache, como, por exemplo, a máquina DASH de Stanford [LEN90]. Essas máquinas têm sido alternativa para a construção de máquinas escaláveis com memória compartilhada.

1.2.2 Multicomputadores

Multicomputadores são constituídos por múltiplos processadores com memória local privativa e sem acesso à memória remota. Não existe compartilhamento de memória e a comunicação entre os processadores ocorre exclusivamente pela troca de mensagens através da rede de interconexão, normalmente através do uso de bibliotecas de comunicação como PVM ou MPI. Por esses motivos, essa arquitetura também recebe o nome de modelo NORMA (NO-Remote Memory Access), ou seja, sem acesso à memória remota. Um exemplo de arquitetura de troca de mensagem é ilustrado na Figura 3.

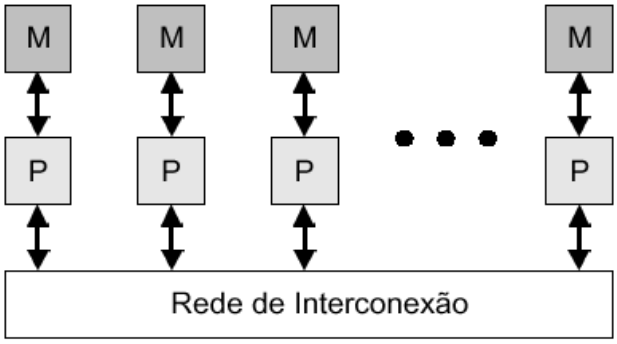

Figura 3 - Modelo NORMA

1.3 MPSoCs

Sistemas intra-chip multiprocessados, MPSoCs (do inglês: *Multiprocessor Systems-on-chips*), são hoje uma das principais aplicações da tecnologia de sistemas integrados [JER04]. MPSoCs incorporam sistemas complexos e permitem o crescimento de grandes nichos de mercado, impulsionando os investimentos necessários para linhas avançadas de fabricação de sistemas integrados. Muitas aplicações, como em dispositivos móveis, exigem a implementação do sistema em um único chip, para atingir as especificações desejadas de tamanho e consumo de energia. Outras aplicações necessitam de projetos em grandes chips para a redução de custos. Sistemas intra-chip, SoC (do inglês: *System-on-chip*) provêem soluções em um único chip em ambos os casos. SoCs são muitas vezes desenvolvidos especificamente para uma determinada aplicação com o objetivo de melhorar seu consumo energético, desempenho ou custo.

Enquanto sistemas monoprocessados podem ser suficientes para aplicações que necessitam baixo desempenho, típico dos primeiros microcontroladores, um crescente número de aplicações necessita de sistemas multiprocessados para atingir seus requisitos de desempenho. Com isso, MPSoCs estão sendo cada vez mais utilizados para o desenvolvimento de sistemas integrados complexos. Um sistema intra-chip multiprocessado não é somente um conjunto de processadores empacotados em um único chip. Tanto os requisitos da aplicação quanto a sua implementação levam o desenvolvedor a construir uma arquitetura específica.

As aplicações de projetos de SoCs apresentam muitas vezes a seguinte combinação de restrições: (i) não somente taxas de computação elevadas, mas também desempenho de tempo real; (ii) baixo consumo de energia e dissipação de potência; (iii) baixo custo.

Cada uma destas restrições já possui um alto grau de dificuldade e as suas combinações tornam o projeto um grande desafio.

MPSoCs equilibram essas necessidades, adaptando a arquitetura do sistema de acordo com os requisitos da aplicação. Colocar poder computacional onde é necessário torna possível atingir os requisitos de desempenho e remover elementos de hardware desnecessários. Como conseqüência temos uma redução no consumo de energia e custo.

MPSoCs não são apenas chips multiprocessados. Chips multiprocessados são componentes que tiram vantagem da crescente densidade dos circuitos integrados para colocar mais processadores em um único chip. Os MPSoCs, além de chips multiprocessados, são arquiteturas especificas que equilibram as restrições do projeto de sistemas integrados com aquilo que a aplicação realmente precisa [JER04].

Na tabela Tabela 1 pode-se observar o estado da arte de alguns MPSoCs presentes na indústria.

Tabela 1 - Estado da Arte dos MPSoCs

Name	*Manufacture*	*Multiprocessing*	*Communication infrastructure*	*#PEs*	*PE type*	*Frequency*	*Throughput*
PC102 (2005)	*picoChip*	Homogeneous	Hibrid (picoBus + switches)	322	RISC (16bits)	150 MHz	-
CELL (2006)	*IBM*	Heterogeneous	Hibrid (Ring + star network)	9	64bits + 8 accelerators	3.3 GHz	204 GFlops/s
Am2045 (2006)	*Ambric*	Homogeneous	NoC mesh	360	RISC (32bits)	333 MHz	1 TOPS
- (2007)	*Intel*	Homogeneous	NoC mesh	80	2 floating-point units	4 GHz	1.28 TFlops/s
Tile64 (2007)	*Tilera*	Homogeneous	NoC mesh	64	3-way VLIW MIPS	866 MHz	443 BOPS

Pode-se observar que no geral, os MPSoCs estão utilizando arquiteturas homogêneas, com redes intra-chips como meio de intercomunicação e com diversos elementos de processamento. Essa análise justifica a escolha deste trabalho pelo MPSoC HeMPS, descrito no Capítulo 2.

1.4 DDR2 SDRAM

A DDR2 SDRAM ou simplesmente DDR2 é uma evolução ao padrão DDR SDRAM, e essa tecnologia têm por objetivo ser melhor no desempenho, no consumo elétrico, na temperatura de operação, na densidade do circuito e no tratamento de interferências eletromagnéticas.

DDR2 SDRAM ou D*ouble Data Rate 2 Synchronous Dynamic Random Access Memory* (memória de acesso aleatório dinâmica síncrona de dupla taxa de transferência 2) é um circuito integrado de memória utilizado em sistemas computacionais, derivada da SDRAM e combinada com a técnica DDR, que consiste em transferir dois dados por ciclo de relógio, obtendo assim, teoricamente, o dobro de desempenho em relação à técnica tradicional de transferência de dados quando operando sob a mesma freqüência de relógio.

A memória DDR2 SDRAM alcança uma largura de banda maior que a da SDR SDRAM (*Single Data Rate Synchonous Dynamic Random Access Memory*) por usar tanto a borda de subida quanto a de descida do relógio para transferir dados, realizando efetivamente duas transferências por ciclo de relógio. Isto efetivamente quase dobra a taxa de transferência sem aumentar a freqüência do barramento externo.

Em relação à velocidade, também é necessário considerar o latência do CAS (do inglês: *Column Address Strobe*). Esse termo se refere ao tempo que a memória leva para fornecer um dado solicitado. Nas memórias DDR, a latência pode ser de 2, 2,5 e 3 ciclos de relógio, enquanto que nas memórias DDR2, a latência vai de 3 a 5 ciclos. Isso significa que, nesse aspecto, a memória DDR2 é mais lenta que a DDR, porém as características do padrão DDR2, especialmente seus valores de freqüência, compensam essa desvantagem.

Há ainda um recurso nas memórias DDR2 que deve ser citado, a latência adicional AL (do inglês: *Additional Latency*). Esta é usada para permitir que os procedimentos ligados às operações de leitura e escrita sejam feitos até expirar o tempo da latência do CAS mais a latência adicional, garantindo uma margem a mais de tempo para as operações. Assim, a medição da latência deve considerar a soma desses dois parâmetros para se obter um total.

Outra característica das memórias DDR é a presença de uma terminação resistiva, para evitar possíveis problemas de ruídos na transmissão do sinal. Nas memórias DDR esse problema foi tratado por meio de resistores externos a memória, que são adicionados à placa conectada a ela. Porém no padrão DDR2, essa terminação resistiva externa não se mostrou eficiente, pelas características físicas desse tipo de memória. Por isso, nessa tecnologia utiliza-

se o ODT (do inglês, *On Die Termination*), onde a terminação resistiva é implementada dentro do próprio circuito integrado de memória. Com isso, o caminho percorrido pelo sinal é menor e há menos ruídos, isto é, menos perda de dados.

Em projetos realizados em dispositivos programáveis, é necessário o desenvolvimento de controladores, para atuarem com a memória. Uma importante ferramenta que gera esses controladores é o Core Generator da empresa Xilinx [KAR08]. Essa ferramenta possui um recurso que cria uma interface com memórias externas, chamada de Gerador de Interfaces de Memória (MIG, do inglês: *Memory Interface Generator*).

1.5 Redes de Comunicação

A necessidade de diminuir custos, aumentar a confiabilidade, disponibilizar o compartilhamento de recursos físicos (HD, impressoras,...) e informações (banco de dados, programas,...) fez surgir as redes de comunicação. Uma rede permite a troca de informações (envio e recebimento) entre diferentes dispositivos. A Internet é o maior exemplo de rede de computadores, com milhões de máquinas conectadas ao redor do mundo.

As tecnologias de rede podem ser divididas em dois grupos básicos: (1) r**ede local**: na qual as tecnologias LAN (*Local Area Network*) conectam muitos dispositivos que estão relativamente próximos, geralmente no mesmo ambiente físico; (2) **rede de longa distância**, na qual as tecnologias WAN (*Wide Area Network*) conectam um número menor de dispositivos que podem estar separados por muitos quilômetros.

Em comparação às WANs, as LANs são mais rápidas e confiáveis. A tecnologia, porém, se desenvolve rapidamente, e as diferenças entre WAN e LAN estão cada vez menores. Os cabos de fibra ótica permitem a conexão de dispositivos LAN separados por quilômetros de distância. Esses cabos também melhoram a velocidade e a confiabilidade das redes WAN.

1.5.1 Elementos de rede

As redes de comunicação consistem basicamente de dois elementos, nodos e um meio de interconexão. Os componentes de uma rede de nodos podem ser divididos em duas classes:

- Equipamentos terminais de dados (DTE, do inglês, Data terminal equipment):

dispositivos que são a fonte ou o destino dos dados. DTEs são tipicamente dispositivos como computadores pessoais, servidores de arquivos, ou servidores de impressão que, como um grupo, todos são muitas vezes referidos como estações finais.

- Equipamento de comunicação de dados (DCE, do inglês, Data communication equipment): dispositivos intermediários que recebem e transmitem dados através da rede. DCEs pode ser dispositivos autônomos como os repetidores, comutadores de rede e roteadores.

1.5.2 Ethernet

Em 1973, os pesquisadores Bob Metcalfe e David Boggs, do *Xerox Corporation's Palo Alto Research Center* (mais conhecido como PARC), criaram e testaram a primeira rede Ethernet. Metcalfe tentava conectar o computador "Alto" da Xerox a uma impressora e acabou desenvolvendo um método físico de cabeamento que conectava os dispositivos na Ethernet. Ele também criou os padrões de comunicação em cabos. Desde então, a Ethernet se tornou a tecnologia de redes mais popular do mundo.

Ethernet é uma tecnologia de interconexão para redes locais - *Local Area Networks* (LAN) - baseada no envio de pacotes. Ela define cabeamento e sinais elétricos para a camada física, e o formato de pacotes e protocolos para a camada de controle de acesso ao meio (*Media Access Control* - MAC). Baseou-se originalmente no protocolo *Carrier Sense Multiple Access with Collision Detection* (CSMA-CD). A Ethernet foi padronizada pelo IEEE (*Institute of Electrical and Electronics Engineers*) como 802.3. Desde os anos 90, ela vem sendo a tecnologia de LAN mais amplamente utilizada e tem tomado grande parte do espaço de outros padrões de rede como *Token Ring*, FDDI (*Fiber Distributed Data Interface*) e ARCNET (*Attached Resource Computer Network*).

O modo de transmissão é uma característica importante do padrão Ethernet, podendo ser:

- **Simplex**: Durante todo o tempo apenas uma estação transmite, a transmissão é feita unilateralmente;
- **Half-duplex**: Cada estação transmite ou recebe informações, não acontecendo transmissão simultânea;
- **Full-duplex**: Cada estação transmite e/ou recebe, podendo ocorrer transmissões simultâneas.

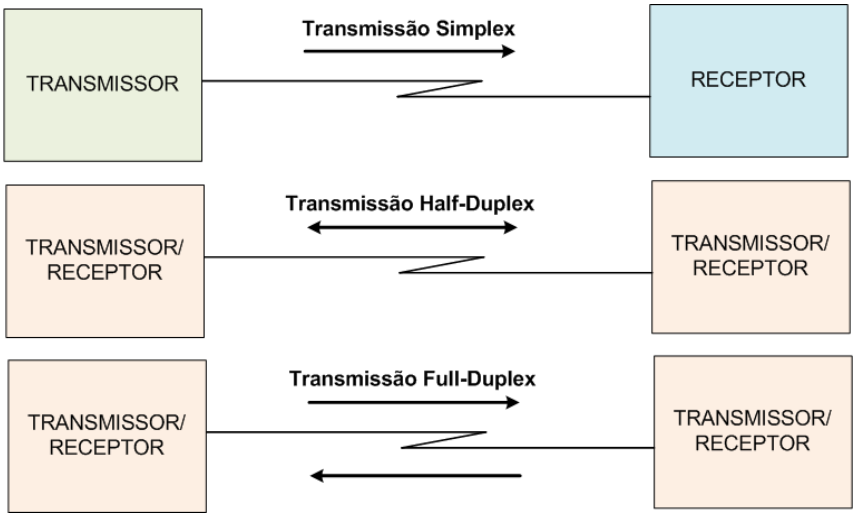

Figura 4 - Modos de Transmissão

Atualmente o protocolo Ethernet está definido nos seguintes padrões: 10 Mbps: 10Base-T Ethernet (IEEE 802.3); 100 Mbps: Fast Ethernet (IEEE 802.3u); 1Gbps: Gigabit Ethernet (IEEE 802.3z) e 10 Gbps: 10 Gigabit Ethernet (IEEE 802.3ae).

1.5.2.1 Estrutura e topologias de redes Ethernet

As LANs podem assumir diferentes configurações quanto a sua topologia de rede, mas, independentemente da sua dimensão ou complexidade, todas elas serão uma combinação de apenas três estruturas básicas de interconexão de rede: (i) conexão ponto a ponto, (ii) barramento compartilhado e (iii) topologia em estrela.

A estrutura mais simples é a interconexão ponto a ponto. Apenas duas unidades de rede estão envolvidas, podendo elas ser do tipo: DTE-DTE, DTE-DCE, ou DCE-DCE. O cabo nas interconexões ponto a ponto é conhecido como *link* de rede. O máximo de comprimento admissível da ligação depende do tipo de cabo e o método de transmissão que é usado.

A segunda estrutura baseia-se nas primeiras implementações de redes Ethernet, onde diferentes DTEs são conectados através de uma estrutura de barramento, como mostrado na Figura 5. O comprimento dos barramentos é limitado a 500 metros, e até 100 estações pode ser conectadas a um único segmento. Segmentos individuais podem ser interligados através do uso de repetidores (DCE), desde que não haja dois caminhos de comunicação distintos entre diferentes segmentos.

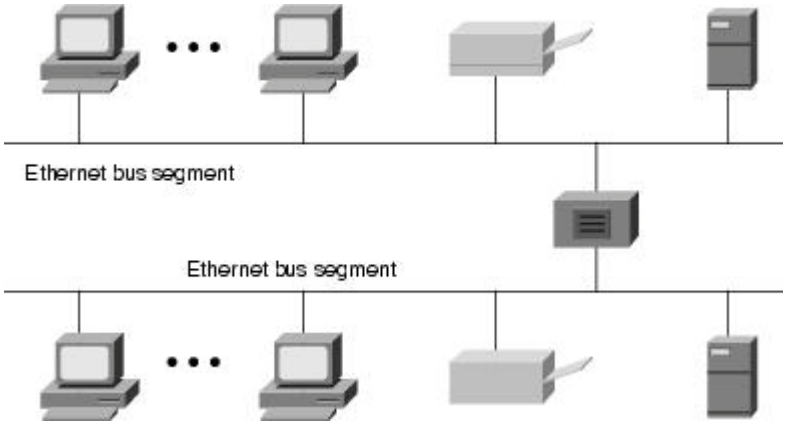

Figura 5 - Exemplo de topologia de barramento

Desde o início da década de 1990, a topologia de rede mais utilizada é a topologia em estrela, onde a unidade de rede central é um repetidor multiporta (podendo ser hubs ou switches de rede) e a ele são conectados DTEs. Todas as conexões entre os DTEs e o DCE central utilizam interconexões ponto a ponto.

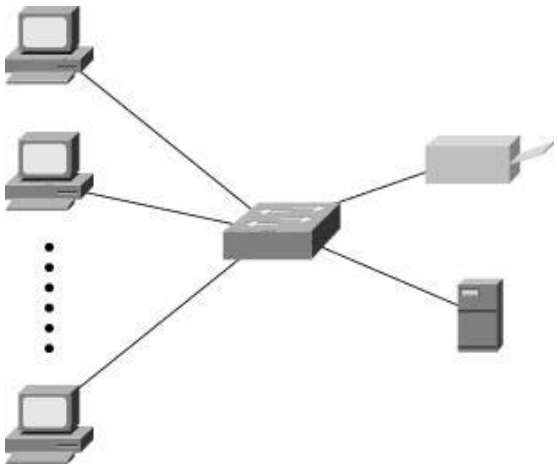

Figura 6 - Exemplo de topologia em estrela

1.5.2.2 Relação entre o Modelo de Referência OSI e o padrão Ethernet

O modelo de referência para Interconexão de Sistemas Abertos (em inglês, *Reference Model for Open Systems Interconnection* ou OSI-RM), proposto pela *International Standards Organization* (ISO), permite a estratificação, o projeto e a implementação de protocolos padronizados em redes de comunicação. O OSI-RM divide o tratamento de informações de rede em sete níveis de abstração. A hierarquia proposta pelo OSI-RM é detalhada no lado esquerdo da Figura 7. A direita ilustra-se o padrão Ethernet bem como a relação entre ele e o modelo de referência OSI. Nota-se que a camada de enlace do modelo OSI é dividida no

padrão Ethernet em duas subcamadas, o Controle de acesso ao meio (MAC) e o controle de enlace lógico. O controle lógico ou subcamada LLC age como uma interface entre a subcamada MAC (*Media Access Control*) e a camada de rede, sendo ela a responsável por especificar os mecanismos para endereçamento de estações conectadas ao meio e para controlar a troca de dados entre dispositivos da rede.

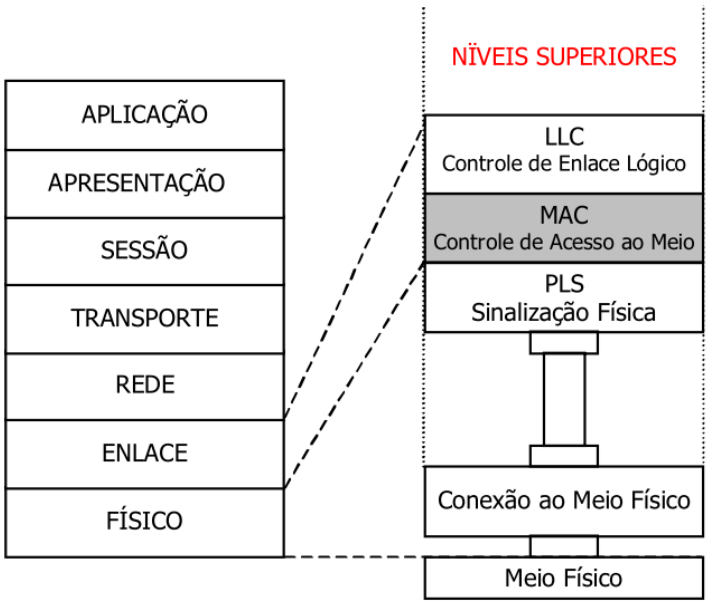

Figura 7 - Relação entre o modelo de referência OSI e o Ethernet

1.5.2.3 Quadros Ethernet e o protocolo CSMA/CD

A Figura 8 detalha o formato de quadro usado na transmissão de dados via tecnologia Ethernet. A natureza das informações contidas em cada um dos campos do quadro padrão (802.3) pode ser claramente identificada a partir da mesma figura. No quadro Ethernet, todos os campos são de tamanho fixo, menos o de dados, que deve conter um número inteiro de bytes.

Preâmbulo 7 bytes	Delimitador de Início de Quadro (SFD) 1 byte	Endereço Destino 6 bytes	Endereço Fonte 6 bytes	Tamanho do quadro 2 bytes	Dados 46 – 1.500 bytes	FCS (CRC) 4 bytes

Figura 8 - Estrutura do quadro Ethernet básico (IEEE 802.3)

O quadro Ethernet é definido como não válido quando o mesmo não possuir um número inteiro de bytes, ou se as partes de um quadro recebido não gerarem um valor de CRC idêntico ao CRC recebido, ou se o tamanho do quadro for menor que o tamanho mínimo. Para que os quadros Ethernet sejam adequadamente transmitidos ou recebidos, o subnível MAC deve desempenhar diversas tarefas, visando garantir a transmissão dos dados com integridade. Na transmissão de quadros, o MAC aceita dados do subnível LLC e monta um quadro. A seguir, transmite um fluxo de dados serial para o nível físico. Sempre que o meio físico está ocupado, o MAC adia a transmissão de dados. O MAC também calcula e acrescenta o FCS para os quadros de saída, e verifica o alinhamento de byte completo. Para garantir o intervalo de tempo mínimo entre quadros, o MAC retarda a transmissão dos dados por um período adequado. Ele também é responsável por detectar colisões e reter a transmissão quando isto ocorre. Após uma colisão, o MAC reforça esta por algum tempo, para garantir a detecção por toda a rede, e programa a retransmissão depois de uma colisão para um instante futuro calculado. Finalmente, o MAC acrescenta preâmbulo, delimitador de início de quadro e FCS, para todos os quadros de saída. Por outro lado, na recepção, o MAC verifica erros de transmissão nos quadros recebidos por meio do FCS e verifica o alinhamento de byte completo. Também descarta quadros menores que o mínimo, os que possuem erros de CRC. O MAC ainda remove o preâmbulo, o delimitador de início de quadro e o FCS de todos os quadros recebidos. Finalmente, o MAC transfere a informação útil do quadro aos níveis superiores [CAL01].

1.5.2.4 Controle de fluxo em redes Ethernet

O controle de fluxo consiste basicamente em permitir que um nó de recepção (como um computador pessoal) ao detectar que seu buffer está se congestionando, solicite ao nó de transmissão (como um servidor de arquivos) para parar de enviar quadros por um determinado período de tempo. O controle é realizado MAC a MAC, através da utilização de um quadro de pausa que é gerado automaticamente pelo MAC do dispositivo de recepção. Se o congestionamento for desfeito antes do período de pausa expirar, uma segunda mensagem de pausa pode ser enviada ao transmissor com o campo de período de tempo preenchido com zero, assim o transmissor poderá voltar a transmitir quadros ao nó de recepção. Uma visão geral do funcionamento do controle de fluxo é mostrada na Figura 9.

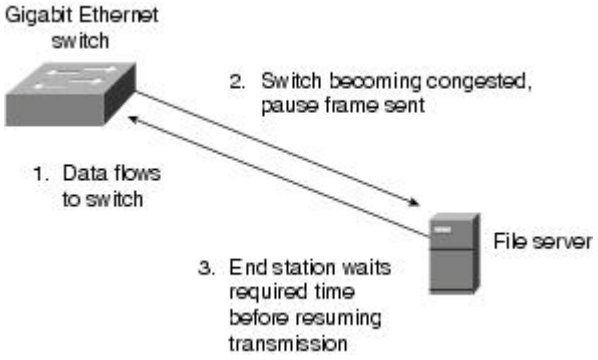

Figura 9 - Visão geral da seqüência de controle de fluxo

1.5.2.5 Protocolos de Rede

O Protocolo Internet (IP, do inglês, *Internet Protocol*) é um protocolo descrito na RFC 791, usado entre duas ou mais máquinas em rede para encaminhamento de dados. Os dados em uma rede IP são enviados em blocos referidos como pacotes ou datagramas. O formato do datagrama IP é ilustrado na Figura 10.

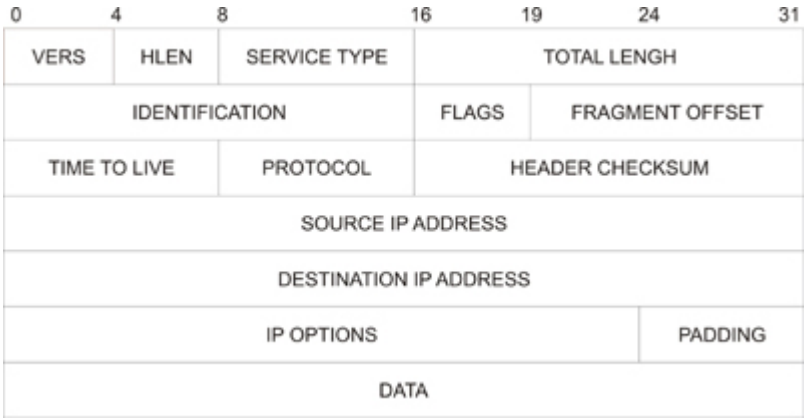

Figura 10 - Formato do datagrama IP

- ***VERS****:* Versão do protocolo *IP* que foi usada para criar o datagrama (4 bits).
- ***HLEN****:* Comprimento do cabeçalho, medido em palavras de 32 bits (4 bits). Ele indica o início dos dados.
- ***TOTAL-LENGTH****:* Este campo proporciona o comprimento do datagrama medido em *bytes*, incluindo cabeçalho e dados.
- ***SERVICE-TYPE****:* Este campo especifica como o datagrama poderia ser manejado.

- ***IDENTIFICATION, FLAGS e FRAGMENTS****:* Estes três campos controlam a fragmentação e a união dos datagramas. O campo de identificação contém um único inteiro que identifica o datagrama, é um campo muito importante porque quando um *gateway* fragmenta um datagrama, ele copia a maioria dos campos do cabeçalho do datagrama em cada fragmento, então a identificação também deve ser copiada, com o propósito de que o destino saiba quais fragmentos pertencem a quais datagramas. Cada fragmento tem o mesmo formato que um datagrama completo.
- ***FRAGMENT OFFSET:*** Especifica o início do datagrama original dos dados que estão sendo transportados no fragmento. É medido em unidades de 8 *bytes*.
- ***FLAG****:* Controla a fragmentação.
- ***TTL(Time To Live):*** Especifica o tempo que o datagrama está permitido permanecer no sistema *Internet*. *Gateways* e *hosts* que processam o datagrama devem decrementar o campo *TTL* cada vez que um datagrama passa por eles e devem removê-lo quando seu tempo expirar.
- ***PROTOCOL****:* Especifica qual protocolo de alto nível foi usado para criar a mensagem que está sendo transportada na área de dados do datagrama.
- ***HEADER-CHECKSUM****:* Assegura integridade dos valores do cabeçalho.
- ***SOURCE AND DESTINATION IP ADDRESS****:* Especifica o endereço *IP* de 32 bits do remetente e receptor.
- ***OPTIONS:*** É um campo opcional. Este campo varia em comprimento dependendo de quais opções estão sendo usadas. Algumas opções são de um *byte*, e neste caso este campo é chamado de *Option Code*, e está dividido em três campos.

Protocolo de Resolução de Endereços *(ARP, do inglês, Address Resolution Protocol*) é um protocolo definido na RFC 826, usado para encontrar um endereço da camada de enlace de rede (endereço MAC, por exemplo) a partir do endereço da camada de rede (como um endereço IP). O emissor difunde em *broadcast* um pacote ARP contendo o endereço IP de outro dispositivo e espera uma resposta com um endereço MAC respectivo. Cada máquina mantém uma tabela de resolução em cache (ARP Cache) para reduzir a latência e carga na rede.

Protocolo de Datagramas do Usuário (UDP, do inglês, *User Datagram Protocol*) é um protocolo simples da camada de transporte. Ele é descrito pela RFC 768 e permite que a aplicação escreva um datagrama encapsulado num pacote IP, e então enviado ao destino. Mas não há qualquer tipo de garantia que o pacote irá chegar ou não. O protocolo UDP não é confiável. Diz-se que o UDP é um serviço sem conexão, pois não há necessidade de manter um relacionamento longo entre cliente e o servidor. A Figura 11 ilustra o cabeçalho do datagrama UDP, logo abaixo segue uma explicação de cada um de seus campos.

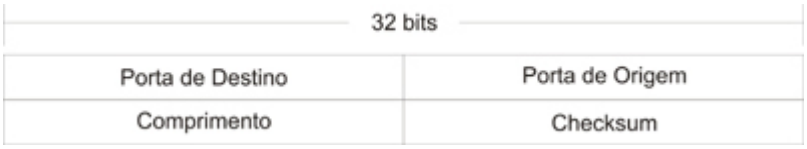

Figura 11 - Formato do datagrama UDP

- **Porta Origem**: Trata-se do número da porta que corresponde à aplicação emissora do segmento UDP. Este campo representa um endereço de resposta para o destinatário.
- **Porta destino**: Contém a porta que corresponde à aplicação da máquina destinatário à qual se dirige.
- **Comprimento**: Especifica o tamanho do cabeçalho UDP.
- **Checksum**: Proporciona uma verificação da integridade no cabeçalho e nos dados.

Protocolo de Controle de Transmissão *(TCP, do inglês,* *Transmission Control Protocol***)** é um dos protocolos sob os quais assenta o núcleo da Internet. A versatilidade e robustez deste protocolo tornaram-no adequado a redes globais, já que este verifica se os dados são enviados de forma correta, na seqüência apropriada e sem erros, pela rede. A Figura 12 ilustra o datagrama TCP.

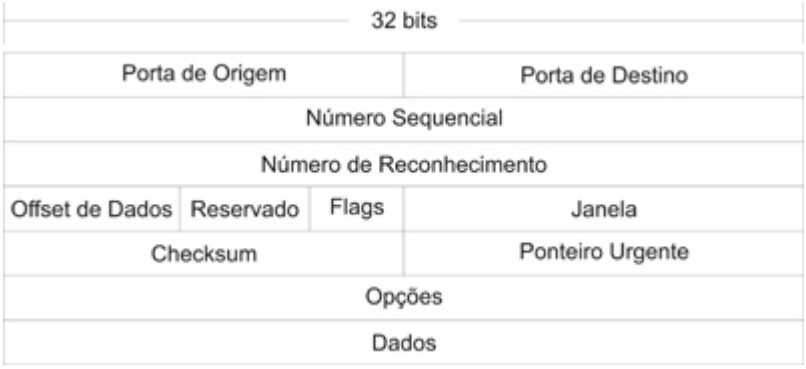

Figura 12 - Formato do datagrama TCP

- **Porta de Origem**: Identifica os pontos em que os processos de origem e de destino da camada superior recebem serviços TCP.
- **Número Seqüencial**: Normalmente especifica o número atribuído ao primeiro byte de dados na mensagem atual. Na fase de estabelecimento da conexão, esse campo também pode ser empregado para identificar um número seqüencial inicial a ser utilizado em uma transmissão de entrada.
- **Número de Reconhecimento**: Contém o número seqüencial do próximo byte de dados que o remetente do pacote espera receber.
- **Off-Set de Dados**: Indica o número de palavras de 32 bits encontradas no cabeçalho do Protocolo TCP.
- **Reservado:** Permanece reservado para uso futuro.
- **Flags:** Carrega várias informações de controle, inclusive o SYN e os bits ACK utilizados para o estabelecimento da conexão, além do bit FIN, utilizado para o encerramento da conexão.
- **Janela:** Especifica o tamanho da janela de recebimento do remetente.
- **Checksum**: Indica se o cabeçalho foi danificado durante o trânsito.
- **Ponteiro Urgente:** Aponta o primeiro byte de dados urgentes do pacote.
- **Opções:** Especifica várias opções do Protocolo TCP.
- **Dados:** Contém informações da camada superior.

Protocolo de Configuração Dinâmica de Cliente (DHCP, do inglês, *Dynamic Host Configuration Protocol*) é um protocolo de serviço TCP/IP que oferece configuração dinâmica de terminais, com concessão de endereços IP e outros parâmetros de configuração para clientes de rede. O DHCP surgiu como padrão em Outubro de 1993. O RFC 2131 contém

as especificações mais atuais deste protocolo. O DHCP oferece três tipos de alocação de endereços IP:

- **Atribuição manual**: Onde existe uma tabela de associação entre o endereço MAC do cliente (que será comparado através do pacote broadcast recebido) e o endereço IP (e restantes dados) a fornecer. Esta associação é feita manualmente pelo administrador de rede; por conseguinte, apenas os clientes cujo MAC consta nesta lista poderão receber configurações desse servidor;
- **Atribuição automática:** Onde o cliente obtém um endereço de um intervalo de endereços possíveis, especificado pelo administrador. Geralmente não existe vínculo entre os vários MAC habilitados a esse espaço de endereços;
- **Atribuição dinâmica:** O único método que dispõe a reutilização dinâmica dos endereços. O administrador disponibiliza um intervalo de endereços possíveis, e cada cliente terá de ser configurado para requisitar um endereço por DHCP assim que a máquina inicialize. A alocação utiliza um mecanismo de aluguel de endereços, caracterizado por um tempo de vida. Após a máquina se desligar, o tempo de vida naturalmente irá expirar, e da próxima vez que o cliente se ligue, o endereço provavelmente será outro.

1.5.3 MAC embarcado em Dispositivos Virtex

Alguns dispositivos da família Virtex 5 disponibilizam aos projetistas um módulo MAC Ethernet Tri-Mode embarcado [XIL09] capaz de implementar as camadas de baixo nível do modelo OSI. O Módulo MAC implementa o método CSMA/CD de acesso ao meio (padrão IEEE 802.3), controla todos os aspectos de transmissão e recepção dos quadros Ethernet e inclui: tratamento de colisão, geração de preâmbulo, detecção e geração de CRC. Diz-se Tri-Mode visto que o mesmo opera nos padrões 10Mbps (IEEE 802.3), 100Mps (IEEE 802.3u) e 1Gbps (IEEE 802.3z).

O módulo embarcado chama-se EMAC e pode ser incluído em projetos de hardware tanto na forma de instanciação manual, a partir da instanciação da primitiva TEMAC, ou facilmente gerado através do software *core Generator* da fabricante *Xilinx*. A primeira opção requer um maior cuidado do projetista, visto que uma série de componentes deve ser incluída ao projeto para prover um funcionamento adequado deste módulo. Gerando o MAC a partir

do *core Generator*, tais cuidados não são necessários visto que o primeiro cria toda a estrutura necessária para o correto funcionamento do EMAC, e o configura, através da atribuição de *generics*, como especificado pelo projetista.

A Figura 13, ilustra o diagrama de blocos do MAC embarcado e suas interfaces: no lado do cliente (lado esquerdo), barramentos de RX e TX, barramento de controle de fluxo e a interface padrão para configuração do MAC (Generic Host Bus e DCR Bus). Do lado do PHY (lado direito) o MAC oferece três formas interfaces de conexão: GTP/GTX, MII/GMII e RGMII, de forma a conectar-se ao componente físico.

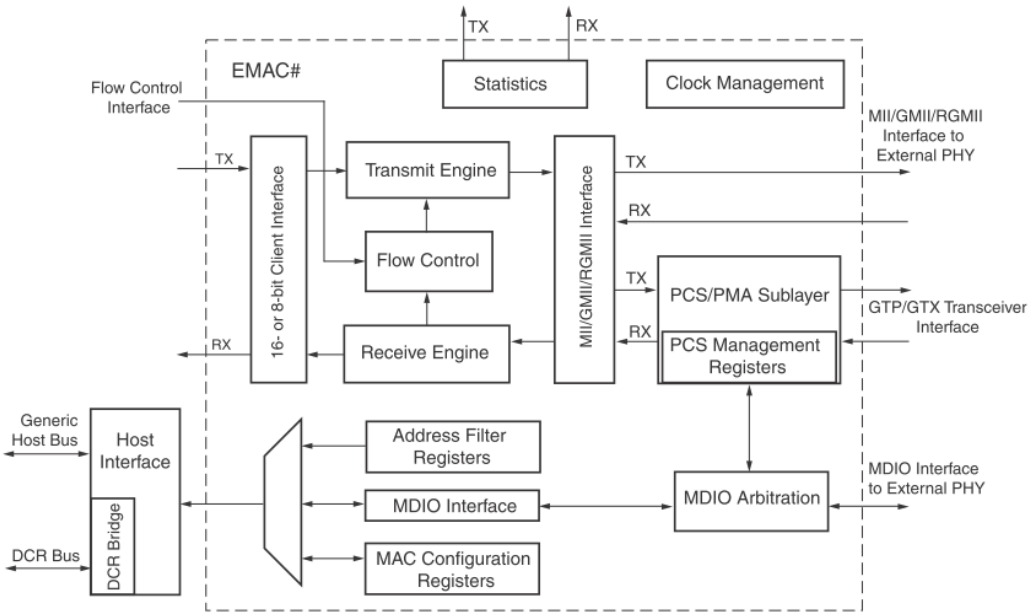

Figura 13 - Diagrama de blocos funcionais do EMAC Ethernet [VIR09]

2 MÓDULOS DE REFERÊNCIA

Este Capítulo tem por objetivo detalhar os três módulos que compõe a HeMPS Station: MPSoC HeMPS, módulo de comunicação Ethernet (ComEt) e o módulo de controle da memória externa DDR2 (ConMe). A Plataforma ComEt e o Sistema ConMe foram desenvolvidos no escopo deste trabalho.

2.1 MPSoC HeMPS

A plataforma HeMPS [WOS07] é um MPSoC homogêneo baseado em NoC, com arquitetura mestre-escravo. A Figura 14 apresenta um exemplo da plataforma, utilizando uma NoC 2x3 com topologia malha. Os principais módulos que compõem o sistema são a NoC HERMES [MOR04] e os diversos processadores Plasma [PLA09], baseados na arquitetura MIPS. O elemento de processamento, chamado de Plasma-IP, contém o processador Plasma e conecta-o à NoC. Esse IP também possui uma memória local, uma interface com a NoC e um módulo para acesso direto à memória (DMA).

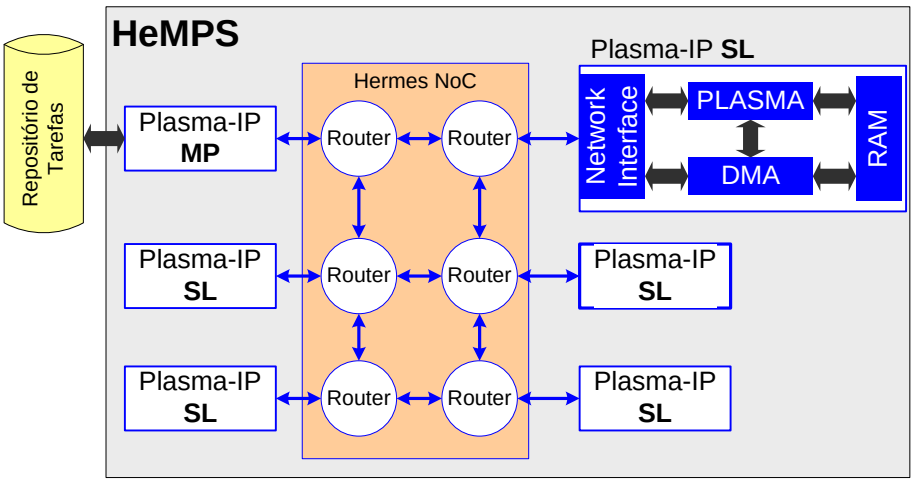

Figura 14 – Instância da plataforma HeMPS utilizando uma NoC 2x3

As aplicações executadas na HeMPS são modeladas usando grafos de tarefas, e cada aplicação deve conter ao menos uma tarefa inicial. O repositório de tarefas é uma memória externa ao MPSoC, que armazena todos os códigos objeto das tarefas necessários para a execução das aplicações. O sistema contém um processador mestre (Plasma-IP MP), responsável por gerenciar recursos do sistema. Esse é o único processador que tem acesso ao

repositório de tarefas. Quando a HeMPS inicia sua execução, o processador mestre aloca a(s) tarefa(s) inicial(ais) nos processadores escravos (Plasma-IP SL). Durante a execução, as tarefas são dinamicamente carregadas do repositório para os processadores escravos, conforme os pedidos de tarefas pelos mesmos. Além disso, recursos podem ficar disponíveis quando uma determinada tarefa terminar sua execução.

Cada processador escravo possui um *microkernel* que suporta execução multitarefa e comunicação entre tarefas. O *microkernel* segmenta a memória em páginas, alocando a primeira delas para si e as demais para as tarefas. Cada Plasma-IP possui uma tabela de tarefas, com as informações das tarefas locais e remotas. Um escalonamento, implementado utilizando algoritmo *Round Robin*, provê suporte multitarefa.

De modo a atingir alto desempenho nos elementos de processamento, a arquitetura do módulo Plasma-IP visa a separação entre computação e comunicação. A interface de rede e o módulo DMA são responsáveis por enviar e receber pacotes, enquanto o processador Plasma realiza a computação de tarefas e gerenciamento de entre os demais módulos. A RAM local é uma memória dupla porta, permitindo acesso simultâneo do DMA e processador, evitando assim hardware extra para elementos como exclusão mútua e roubo de ciclo.

O ambiente inicial para avaliar aplicações distribuídas executadas na HeMPS é chamado *HeMPS Generator* (Figura 15 e Figura 16).

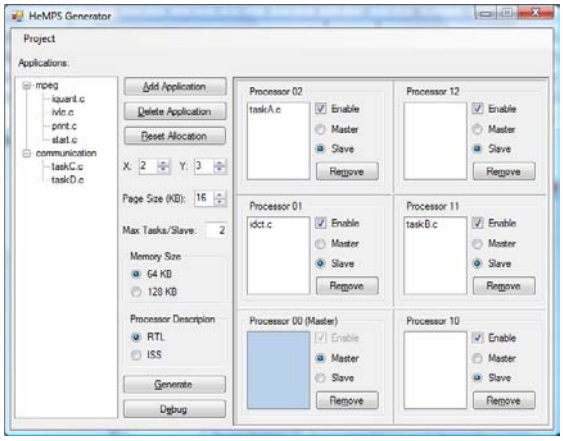

Figura 15 - Ambiente HeMPS Editor

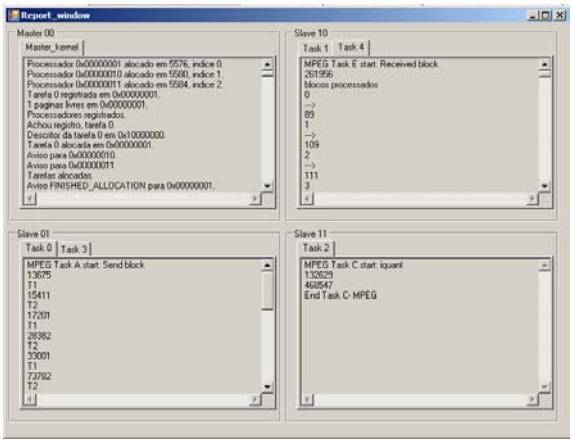

Figura 16 - Ferramenta de Depuração do HeMPS Editor

O ambiente *HeMPS Generator* permite:

1. Configuração da Plataforma: número de processadores conectados a NoC, através dos parâmetros *X* e *Y*; número máximo de tarefas simultâneas por escravo; tamanho de página; nível de abstração da descrição do processador. O

usuário pode escolher entre as descrições na linguagem *C* (ISS) e VHDL. Ambas são equivalentes funcionalmente, sendo a descrição ISS mais rápida em tempo de simulação e a VHDL com mais informações para eventual depuração mais detalhada.

2. Incluir aplicações no sistema. A Figura 15 à esquerda mostra as aplicações *mpeg* e *communication*¸ inseridas no sistema, cada uma com um conjunto de tarefas.
3. Definir o mapeamento inicial das tarefas. Reparar na Figura 15 a tarefa *idct* (tarefa inicial do *mpeg*) alocada no processador 01, e as tarefas *taskA* e *taskB* (tarefas iniciais da aplicação *communication*) alocadas nos processadores 11 e 02 respectivamente.
4. Executar a geração hardware-software, através do botão *Generate*. Isso irá fazer com que (1) as memórias sejam inicializadas (*microkernel*, API, *drivers*) e (2) o repositório seja preenchido com todos os códigos objeto das tarefas.
5. Depuração via simulação, através do botão Debug. A avaliação do sistema é realizada utilizando um simulador comercial, tanto para descrição ISS quanto para descrição VHDL. Ao clicar no botão Debug HeMPS é acionada a ferramenta gráfica de depuração (Figura 16). Essa ferramenta contém uma janela para cada processador. Cada janela possui diversas abas, uma para cada tarefa executada no processador correspondente (na Figura 16 os processadores Slave 10 e Slave 01 executam duas tarefas cada). Desse modo, as mensagens ficam separadas, permitindo ao usuário visualizar os resultados da execução de cada tarefa.

2.1.1 NoC HERMES

Para a interconexão dos núcleos de processamento do MPSoC utiliza-se a NoC HERMES [MOR04]. A escolha do uso da NoC como meio de interconexão é devido ao fato da vantagens que esta proporciona ao projeto de MPSoCs, tais como escalabilidade e paralelismo na comunicação.

A NoC HERMES possui um mecanismo de comunicação denominado chaveamento de pacotes, no qual pacotes são roteados individualmente entre os nodos 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 através dos roteadores. A HERMES utiliza o modo de roteamento *wormhole*, no qual um pacote é transmitido entre os roteadores em *flits*. Apenas o *flit* de cabeçalho possui a informação de roteamento. Assim, os *flits* restantes que compõem o pacote devem seguir o mesmo caminho indicado pelo cabeçalho.

A NoC HERMES utiliza a topologia malha, ilustrada na Figura 17. Essa topologia é justificada em função da facilidade de desenvolver o algoritmo de roteamento, inserir núcleos e gerar *layout* do circuito. A interface de comunicação entre os roteadores vizinhos é composta pelos seguintes sinais: (1) *clock_tx*: sincroniza a transmissão de dados; (2) *tx*: indica disponibilidade de dado; (3) *data_out*: dado a ser transmitido; (4) *credit_in*: informa a disponibilidade de *buffer* no roteador vizinho; (5) *lante_tx*: indica o canal virtual sendo utilizado (sinal não utilizando na HeMPS).

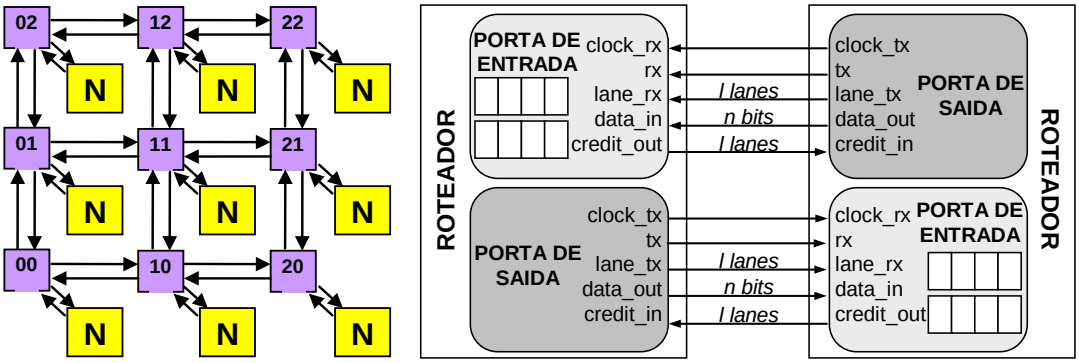

Figura 17 - Exemplo de NoC HERMES e Interface entre os Roteadores

O roteador possui uma lógica de controle de chaveamento centralizada e 5 portas bidirecionais: *East, West, North, South* e *Local*. A porta Local estabelece a comunicação entre o roteador e seu núcleo local. As demais portas ligam o roteador aos roteadores vizinhos.

A cada porta de entrada é adicionado um *buffer* para diminuir a perda de desempenho com o bloqueio de *flits*. A perda de desempenho acontece porque quando um *flit* é bloqueado em um dado roteador os *flits* seguintes do mesmo pacote também são bloqueados em outros roteadores. Com a inserção de um *buffer* o número de roteadores afetados pelo bloqueio de *flits* diminui. O *buffer* do roteador da NoC HERMES funciona como uma fila FIFO (*First In First Out*) circular, e possui tamanho parametrizável. A porta de entrada é responsável por receber os *flits* de pacotes e armazená-los no *buffer*. Para cada *flit* armazenado é verificada a disponibilidade de recepção do *buffer* e a disponibilidade de crédito é informada através do sinal *credit_o*.

A lógica de controle de chaveamento implementa uma lógica de arbitragem e um algoritmo de roteamento. Quando um roteador recebe um *flit* de cabeçalho, a arbitragem é executada e se a requisição de roteamento do pacote é atendida, um algoritmo de roteamento é utilizado para conectar o *flit* da porta de entrada à porta de saída correspondente. Cada roteador deve possuir um endereço único na rede. Para simplificar o roteamento, este endereço é expresso nas coordenadas XY, onde X representa a posição horizontal e Y, a posição vertical do roteador na rede, sendo a posição 00 no canto inferior esquerdo.

A arbitragem utilizada pelo roteador da NoC HERMES é *Round-Robin*. Essa política utiliza um esquema de prioridades dinâmicas, proporcionando um serviço mais justo que a prioridade estática.

O algoritmo utilizado para o roteamento é XY. O algoritmo XY compara o endereço atual ($X_L Y_L$) com o endereço do roteador destino ($X_T Y_T$) do pacote, armazenado no *flit* de cabeçalho. Os *flits* devem ser roteados para a porta Local quando o endereço $X_L Y_L$ do roteador atual é igual ao endereço $X_T Y_T$ do pacote. Se não for o caso, então primeiro é comparado o endereço X_T com o endereço X_L, até que ambos sejam iguais, fazendo assim com que o pacote esteja alinhado horizontalmente com o destino. Em seguida, é realizada a mesma operação para os endereços Y_T e Y_L até que ambos sejam iguais e o pacote seja movido para a porta Local. Se a porta escolhida está ocupada, o *flit* de cabeçalho e também todos os *flits* subseqüentes do pacote serão bloqueados.

Depois que todos os *flits* do pacote forem transmitidos, a conexão deve ser encerrada. Isto pode ser realizado de dois modos diferentes: por um *trailer* ou usando um contador de *flits*. Um *trailer* requer um ou mais *flits* para serem usados como terminador de pacote e uma lógica adicional para detectar o *trailer*. O roteador da NoC HERMES utiliza um contador de *flits* em suas portas de entrada. Este contador é inicializado quando o segundo *flit* do pacote é recebido, indicando o número de *flits* que compõem o *payload*. O contador é decrementado a cada *flit* transmitido com sucesso. Quando o valor do contador chega à zero significa que todos os *flits* foram enviados.

2.1.2 Processador Plasma

O Plasma é um processador RISC de 32 bits com um subconjunto de instruções da arquitetura MIPS [HEN98][WOS07]. Seu código (VHDL) é aberto, disponível através do *OpenCores* [OPE06][WOS07]. O *pipeline* de instruções do Plasma contém três estágios:

busca, decodificação e execução. Diferentemente da definição original do processador MIPS, a organização da memória do Plasma é Von Neumann e não Harvard. Além disso, o Plasma oferece suporte ao complicador C (*gcc*) e tratamento de interrupções.

O Plasma do MPSoC HeMPS possui algumas modificações em relação ao Plasma original. Essas modificações dizem respeito à criação do mecanismo de interrupção, exclusão de módulos e registradores mapeados em memória, além de inclusão de novos módulos e novos registradores mapeados em memória.

Para tornar possível a execução de múltiplas tarefas sobre a mesma CPU, esse Plasma possui um mecanismo de paginação. Dado que o Plasma original não oferece suporte a interrupções de software, foi necessária a inclusão da instrução *syscall*.

A HeMPS tem implementado mais dois módulos: *Network Interface* (*NI*), responsável por realizar a comunicação entre o processador e a NoC; *DMA*, desenvolvido para transferir o código-objeto de tarefas que chegam na *NI* para a memória do processador. A Figura 18 mostra o diagrama do módulo Plasma presente na HeMPS.

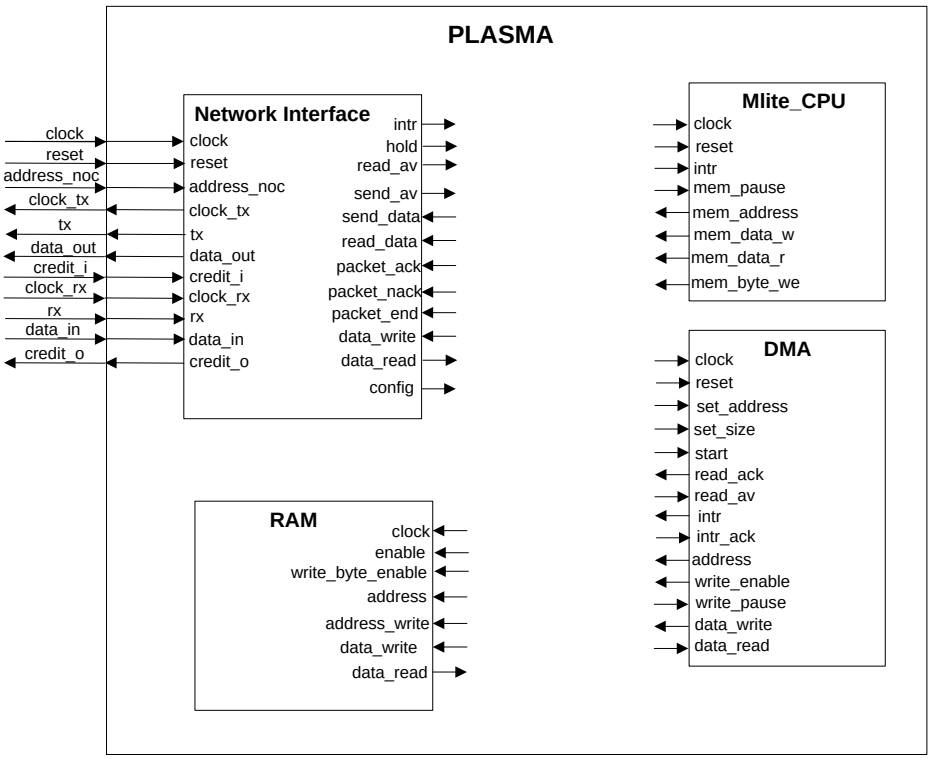

Figura 18 - Módulo Plasma presente na HeMPS

2.1.2.1 Network Interface (NI)

A NI realiza a interface entre o processador e a NoC HERMES e é responsável por:

1. Enviar pacotes para a rede, segmentando os dados deste pacote em *flits*. A NI recebe do processador um pacote cujos dados possuem 32 bits de tamanho e os divide em segmentos de 16 bits (tamanho do *flit* da NoC);
2. Receber pacotes da rede armazenando-os em um *buffer*. Quando existir um pacote completo no *buffer* ou quando o *buffer* estiver cheio, a NI interrompe o processador para que este receba os dados. Os segmentos de 16 bits referentes ao conteúdo do pacote recebido da rede são agrupados e repassados para o processador com palavras de 32 bits;
3. Repassar o código-objeto de tarefas recebido da rede, através do DMA, para a memória;
4. Informar ao processador (*microkernel*) qual a sua localização na rede (*netAddress*).

Um pacote que trafega na rede possui o seguinte formato:

<*target*><*size*><*payload*>

onde *target* indica o destino do pacote; *size* indica o tamanho, em *flits*, do conteúdo do pacote; *payload* indica o conteúdp do pacote. Os campos *target* e *size* possuem tamanho de 16 bits. O campo *payload* é constituído por:

<*service*><*service_parameters*>

onde *service* é o serviço solicitado e *service_parameters* são os parâmetros necessários a este serviço. O serviço reflete a ação a ser tomada pelo *microkernel* depois de ter recebido o pacote. Os serviços que um pacote pode carregar são descritos na Tabela 2.

Tabela 2 - Descrição dos serviços disponíveis.

Serviço	Código	Descrição
REQUEST_MESSAGE	0x00000010	Requisição de uma mensagem.
DELIVER_MESSAGE	0x00000020	Entrega de uma mensagem previamente solicitada.
NO_MESSAGE	0x00000030	Aviso de que a mensagem solicitada não existe.

Serviço	Código	Descrição
TASK_ALLOCATION	0x00000040	Alocação de tarefas: uma tarefa deve ser transferida pelo DMA para a memória do processador.
ALLOCATED_TASK	0x00000050	Aviso de que uma nova tarefa está alocada no sistema.
REQUEST_TASK	0x00000060	Requisição de uma tarefa.
TERMINATED_TASK	0x00000070	Aviso de que uma tarefa terminou sua execução.
DEALLOCATED_TASK	0x00000080	Aviso de que uma tarefa terminou a sua execução e pode ser liberada.
FINISHED_ALLOCATION	0x00000090	Aviso que o nodo mestre terminou a alocação inicial das tarefas (alocação estática).

2.1.2.2 Direct Memory Access (DMA)

O DMA é responsável por transferir o código-objeto de uma tarefa para a memória do processador permitindo que este continue sua execução. O código-objeto de uma tarefa é enviado pela rede para determinado processador.

Assim como qualquer outro pacote, quem recebe o código-objeto da tarefa é a NI, armazenando-o no *buffer* de recebimento. Uma vez o pacote recebido, as seguintes operações são realizadas:

1. A NI imterrompe a CPU informando a chegada de um pacote;
2. O *microkernel*, que executa na CPU e faz parte da infra-estrutura de software do sistema, interpretará o pedido de interrupção como nova tarefa a ser alocada. O *microkernel* obtém o identificador da tarefa e o tamanho do código objeto da mesma e verifica a disponibilidade de página livre na memória. A CPU informa ao DMA o endereço da memória a partir do qual o código objeto deve ser transferido e o tamanho do código-objeto;
3. O DMA faz acessos à NI para ler o código objeto e acessos à memória para escrever o mesmo. Quando o código já estiver alocado, o DMA interrompe a CPU informando que uma nova tarefa está na memória;
4. O *microkernel* faz as inicializações da tarefa, e a partir disso a tarefa executará quando for escalonada.

2.1.2.3 Estrutura do microkernel

A estrutura do *microkernel* multitarefa é mostrada na Figura 19. Ela é composta por diferentes níveis de serviços. No nível mais baixo (Nível 1) encontra-se o serviço de inicialização do sistema (boot), onde são inicializados os ponteiros para dados globais e para pilha, a seção de dados estáticos e as estruturas de dados para gerenciamento das tarefas. Após a inicialização, o serviço de boot aciona o escalonador. No nível acima do boot (Nível 2), encontram-se os drivers de comunicação. O último nível (Nível 3) é composto pelos serviços de tratamento de interrupções, escalonamento, comunicação entre tarefas e chamadas de sistema.

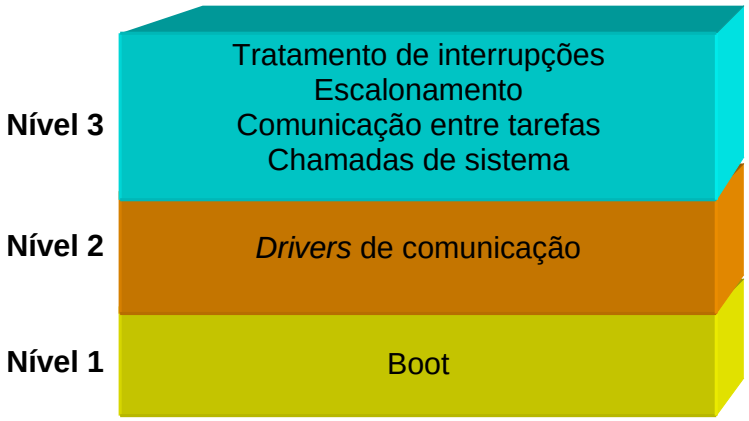

Figura 19 - Estrutura em níveis no *microkernel*

2.2 Plataforma ComEt

A plataforma ComEt é o ambiente desenvolvido neste trabalho para comunicar um dispositivo FPGA com um computador hospedeiro através de redes de computadores. Sua estrutura envolve um software e uma implementação da pilha TCP/IP. Diferentemente da maioria dos projetos deste tipo, toda a pilha foi desenvolvida em hardware, sem a necessidade da utilização de um processador inteiramente dedicado ao processamento das informações do padrão Ethernet. Esta estratégia foi adotada devido à necessidade de alto desempenho e baixa ocupação de área do dispositivo.

O ComEt implementa atualmente três protocolos, sendo eles: UDP, IP e ARP, pertencentes, respectivamente, as camadas de transporte, rede e enlace, além de encapsulá-los em quadros Ethernet para em seguida repassá-los ao EMAC.

A Estrutura do ComEt divide-se em três blocos funcionais: (*i*) Transmissor, (*ii*) Receptor e (*iii*) MAC, além de uma estrutura de armazenamento, a ARP *Cache*, e um software para interação com o usuário, conforme ilustrado na Figura 20.

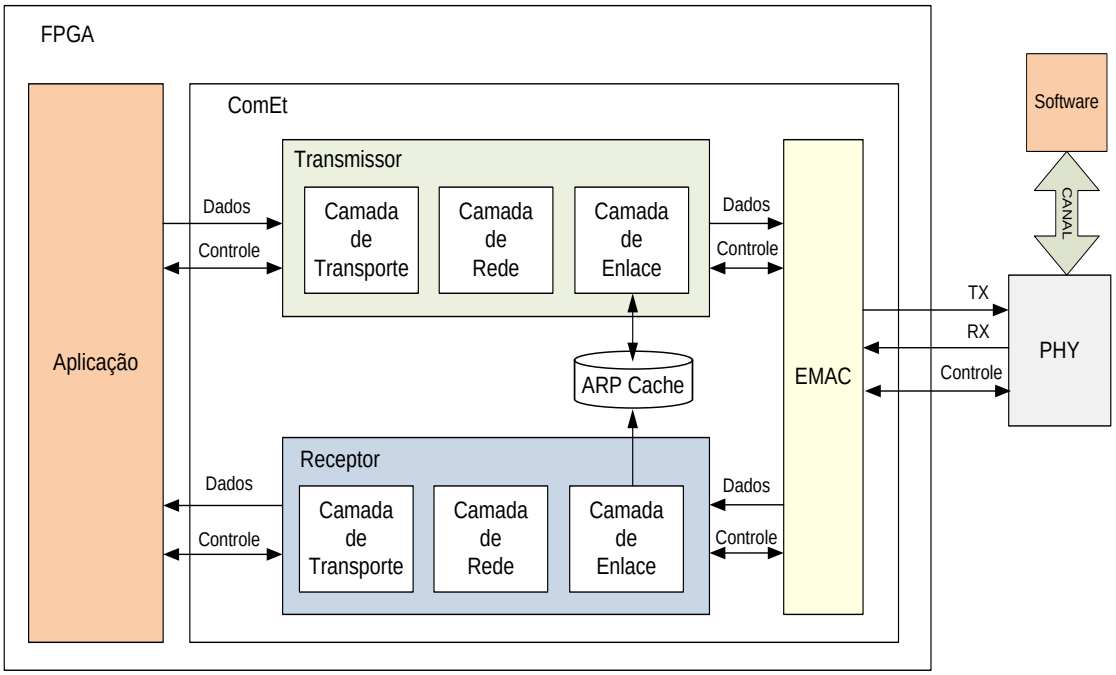

Figura 20 - Diagrama de blocos funcionais da Plataforma

Atualmente a atribuição de um endereço IP para o dispositivo acontece de forma estática, isto é, atribui-se um IP fixo ao ComEt e o mesmo mantém este endereço até ser reprogramado. Como um objetivo de curto prazo, pretende-se implementar o protocolo DHCP, assim o dispositivo poderia obter seu endereço IP automaticamente ao ser conectado a uma rede, não necessitando de uma configuração prévia.

2.2.1 Organização da Pilha ComEt

A Figura 21 apresenta a organização em camadas do ComEt (lado direito), comparando ao modelo de referência TCP/IP (lado esquerdo).

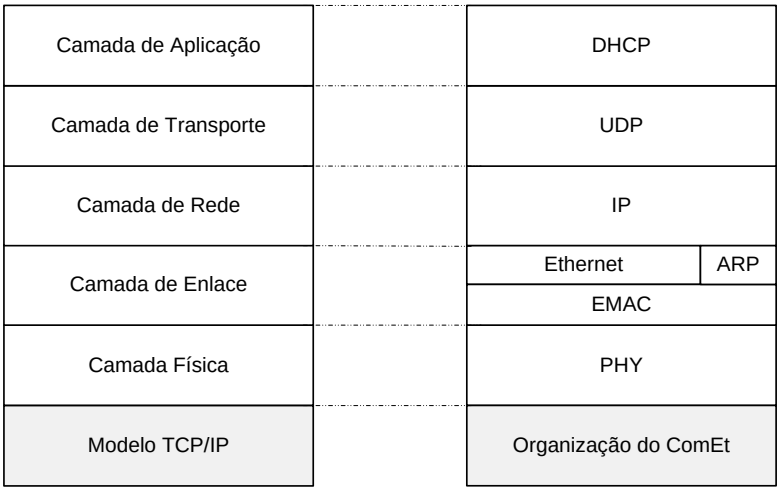

Figura 21 - Organização das camadas da Plataforma ComEt

A Tabela 3 faz uma breve descrição das cinco camadas presentes na pilha TCP/IP, especificando os protocolos implementados pelo ComEt.

Tabela 3 - Descrição das Camadas da pilha TCP/IP

Camada	Descrição	Protocolos Implementados
Camada Física	A camada física provê características físicas, elétricas, funcionais e procedimentos para ativar, manter e desativar conexões entre duas partes.	-
Camada de Enlace	Esta camada é dividida em duas subcamadas, Ethernet/ARP e EMAC. Na primeira, os pacotes vindos da camada superior são encapsulados em um quadro Ethernet, ou seja, são atribuídos os endereços MAC origem e destino e o tamanho do quadro. Caso o módulo Ethernet não encontre o endereço MAC na ARP cache, um pacote ARP é criado e encaminhado a subcamada inferior. A segunda subcamada especifica como os dados serão fisicamente transmitidos através da rede. Esta última não será implementada pelo módulo ComEt, visto que o dispositivo alvo já dispõe de um MAC embarcado, implementando esta subcamada.	Ethernet e ARP
Camada de Rede	É a responsável por enviar e receber dados da rede e decidir como os pacotes de dados serão transmitidos de uma rede a outra, isto é, como será realizado o roteamento do pacote. Utiliza-se de um endereço de 32 bits (IP) para identificar a origem o destino do pacote.	IP
Camada de Transporte	Esta camada é responsável para garantir a segurança na entrega dos dados de um dispositivo a outro e também por gerenciar outras atividades de conexão. A partir de um campo porta, esta camada identifica a qual aplicação entregar os dados vindos das camadas inferiores.	UDP
Camada de aplicação	Define como as aplicações TCP/IP se comunicarão com a camada de transporte de maneira a garantir a comunicação através da rede.	DHCP (a ser implementado)

2.2.2 Transmissor

O módulo de transmissão tem o objetivo de encapsular os dados provenientes da aplicação, de forma a entregar ao MAC um quadro Ethernet válido, isto é, a aplicação disponibiliza dados ao transmissor, este por sua vez, adiciona aos dados um cabeçalho UDP, IP e Ethernet, respectivamente nas camadas de Transporte, Rede e Enlace, e o entrega ao MAC. A seguir segue uma breve descrição das características que de cada camada, na plataforma ComET:

- **Camada de Transporte:** Tem como principais características identificar quais aplicações estão se comunicando. Isto é, a partir de uma porta origem e uma destino, a camada de transporte identifica as aplicações que estão se comunicando. Além disso, realiza o calculo do *checksum* do pacote, de forma ao destinatário poder verificar a integridade dos dados ali contidos.
- **Camada de Rede:** Utiliza-se de endereço IP para identificar o dispositivo destino a quem está se comunicando. Assim como na camada de transporte, a camada de rede também realiza o cálculo de *checksum*, no entanto, somente do cabeçalho inserido por ela, não mais de todo a carga útil do pacote.
- **Camada de Enlace:** Sabendo que a camada de enlace utiliza-se de um endereço físico, único para cada equipamento, e que a camada de rede trabalha com endereçamento IP. Resta a camada de enlace a função de “transformar” o endereço IP fornecido pela camada superior em endereço físico. Com esse intuito, ele consulta a ARP *cache*, a fim de obter o endereço físico referente àquele IP. Se encontrá-lo, inclui aos dados vindos da camada de rede, o endereço físico de origem e destino e identifica o encapsulamento da camada inferior, isto é, informa que a camada inferior encapsula um pacote IP. Caso o endereço IP não seja encontrado na ARP *cache*, um pacote ARP é gerado e enviado as camadas inferiores. Este processo é bloqueante, ou seja, a camada de enlace fica aguardando o recebimento de um pacote ARP de resposta, para só assim, encapsular a informação da camada superior e o entregar as camadas inferiores.

2.2.2.1 Interface

A Figura 22 apresenta a interface do transmissor ComEt com a Aplicação.

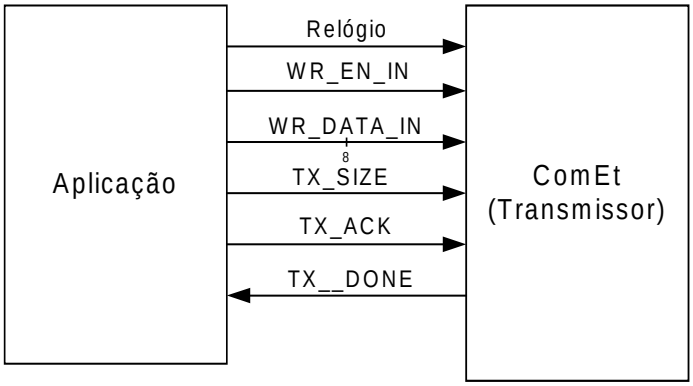

Figura 22 - Interface entre Transmissor ComEt e Aplicação

O sinal de *WR_EN_IN* e *WR_DATA_IN* são sinais conectados diretamente a uma memória no módulo de transmissão. É a partir deles que a aplicação envia dados ao módulo de transmissão do ComEt. O primeiro, *WR_EN_IN*, habilita a escrita na memória, e o segundo, *WR_DATA_IN*, é um barramento de 8 bits utilizado para transferência dos dados. O sinal *TX_SIZE* informa ao transmissor a quantidade de bytes escritos em sua memória. Os últimos dois sinais, *TX_ACK* e *TX_DONE* são utilizados para realizar a sincronia entre os módulos. O primeiro informa ao transmissor que ele já pode iniciar o empacotamento dos dados, e o segundo é um sinal do transmissor indicando que iniciará o empacotamento dos dados armazenados.

2.2.2.2 Protocolo

Sempre que a aplicação desejar enviar dados a partir do ComEt, ela deverá seguir o protocolo descrito a seguir. Primeiramente ativa-se o sinal *WR_EN_IN* indicando que irá se escrever na memória do módulo de transmissão, paralelamente dados deverão ser colocados no barramento *WR_DATA_IN*, sendo um dado por ciclo de relógio. O sinal *relógio* é responsável pela sincronia no momento da escrita. O sinal de *WR_EN_IN* deve permanecer em nível lógico alto enquanto a aplicação estiver escrevendo dados no transmissor. Ao término da escrita dos dados, a aplicação deve atribuir ao barramento *TX_SIZE* a quantidade de bytes escrita na memória do transmissor, ao mesmo tempo em que atribui ao sinal *TX_ACK*

o nível lógico alto. Tanto o sinal *TX_ACK* quanto o *TX_SIZE* devem permanecer altos até que o sinal de *TX_DONE* esteja em nível lógico alto, informando que o transmissor leu o barramento *TX_SIZE* e iniciará o empacotamento dos dados. A Figura 23 ilustra este protocolo.

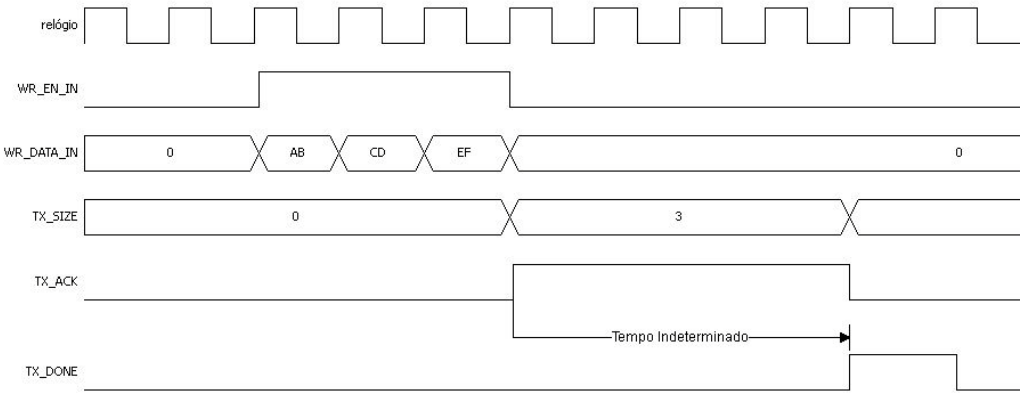

Figura 23 - Protocolo de transmissão

2.2.2.3 Receptor

Este módulo é o responsável por identificar possíveis erros de transmissão, descartar os pacotes inadequados e desempacotar os corretos. Assim como o módulo de transmissão, o de recepção é dividido em três camadas: Transporte, Rede e Enlace. A seguir descrevem-se as funcionalidades de cada uma delas:

- **Camada de Transporte:** Recebe os dados da camada de rede, e verifica a qual aplicação os dados pertencem. Se pertencerem a uma aplicação em execução, ou seja, se a porta for reconhecida, calcula-se o *checksum* e o compara com o vindo no pacote, se correto o desencapsula e envia somente os dados para a camada de aplicação. Caso a porta não for identificada, ou o *checksum* estiver errado, o pacote completo é descartado. No ComEt, é esta a camada responsável por identificar uma mensagem de conexão e desconexão, isto é, para iniciar uma transmissão de dados entre um dispositivo e uma aplicação, uma mensagem de conexão deve ser transmitida. Da mesma forma, para liberar a aplicação, deve-se enviar um pacote de desconexão.
- **Camada de Rede:** Confere se o endereço destino do pacote é igual ao endereço IP do dispositivo, se for, verifica se ele já foi conectado. Estando

conectado, verifica através do endereço IP, se aquele pacote foi transmitido pelo mesmo dispositivo que o conectou, se for, confere o *checksum*, desencapsula o pacote o transmite a camada de transporte. Caso contrário, o pacote é descartado.

- **Camada de Enlace:** Esta camada tem duas funções distintas, a primeira é receber as informações do MAC, desencapsulá-las e transmitir para a camada de rede. A segunda é identificar pacotes ARP de resposta, interpretá-lo e salvar seu conteúdo, endereço IP e físico (MAC), na ARP *cache*.

2.2.2.4 Interface

A Figura 24 apresenta a interface do receptor ComEt com a Aplicação.

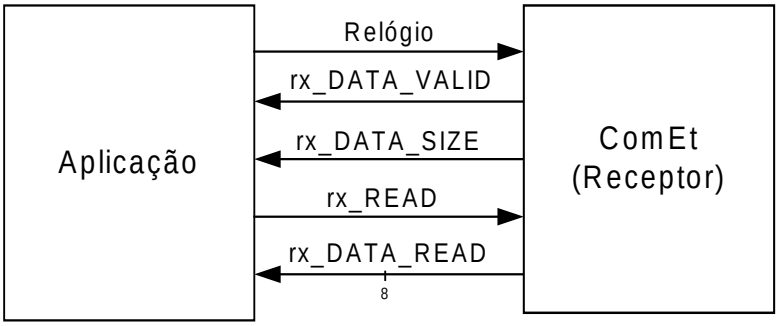

Figura 24 - Interface entre Receptor ComEt e Aplicação

A interface entre o módulo da aplicação e o receptor do ComEt possui basicamente 5 sinais, como ilustrado pela Figura 24. O sinal de relógio, que é transmitido pela aplicação e serve como referência de tempo no momento da leitura dos dados armazenados no receptor pela aplicação. O sinal *rx_DATA_VALID* informa à aplicação quando há dados a serem lidos. O sinal de *rx_DATA_SIZE* informa a quantidade de bytes a serem lidos. O sinal *rx_READ* informa ao módulo de recepção que a aplicação deseja ler dados, ou seja, serve como um sinal de ativação de leitura. O último destes sinais, *rx_DATA_READ* é um barramento de 8 bits, que retorna dados do receptor.

2.2.2.5 Protocolo

Sempre que o módulo de recepção apresenta dados disponíveis para a aplicação, o sinal *rx_DATA_VALID* é colocado em nível lógico alto. O sinal *rx_DATA_SIZE* conterá o número total de bytes armazenados para leitura, assim, a aplicação saberá a quantidade de dados disponíveis para a leitura, estes sinais mantém seu valor até que o sinal de *rx_READ* atinja um nível lógico alto, ou seja, até que a aplicação detecte a presença de dados e comece a ler-los. Um ciclo de relógio após a requisição de leitura, os dados começam a ser escritos no barramento *rx_DATA_READ*. O sinal de relógio é alimentado pela aplicação, de maneira a sincronizar os dados do barramento *rx_DATA_READ*. A Figura 25 ilustra este protocolo.

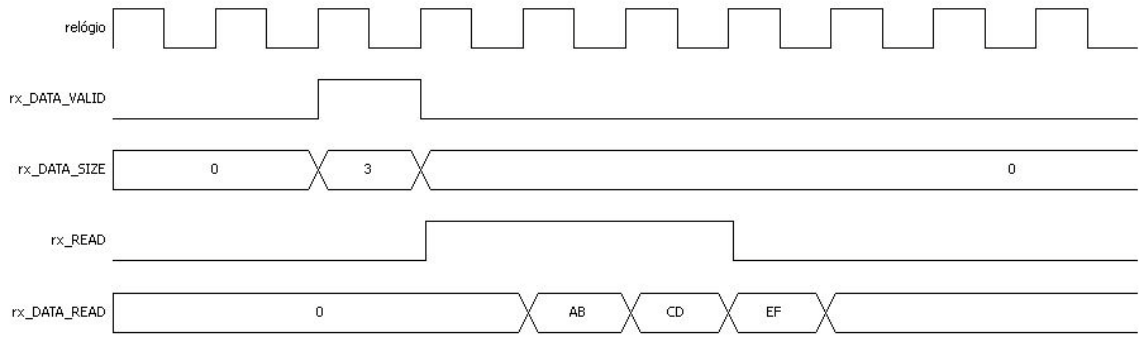

Figura 25 - Protocolo de Recepção

2.2.3 MAC

O MAC utilizado na plataforma ComEt, é um MAC embarcado, configurado através da ferramenta *Core Generator* da empresa *Xilinx* para dispositivos *Virtex-5*. É este o módulo responsável por completar o cabeçalho do quadro Ethernet, e o transmiti-lo à camada física. Apesar deste módulo oferecer a opção de trabalhar sob o padrão Gigabit Ethernet (l Gbps), atualmente tem-se ele configurado somente para o padrão Ethernet (10 Mbps) e Fast Ethernet (100 Mbps). Por ser um bloco bem definido dentro da plataforma ComEt. Sua substituição por outro módulo MAC é bastante simples e rápida. Dessa forma, a plataforma ComEt deixa de ser exclusivamente para dispositivos Virtex-5 e passa a possibilitar sua implementação em qualquer dispositivo, desde que o projetista possua implementado um MAC proprietário, e

logicamente, desde que o dispositivo disponibilize uma camada física que suporte o padrão Ethernet.

2.2.4 Software ComEt

O software ComEt é a ferramenta responsável por interagir com o usuário da Plataforma ComEt. Esse aplicativo possibilita enviar e receber pacotes UDP para qualquer dispositivo conectado na rede que seja capaz de interpretar pacotes UDP. O objetivo final desse aplicativo é ser incorporado ao HeMPS Editor, assim, formando um Framework do HeMPS Station.

A interface gráfica do software pode ser observada na Figura 26.

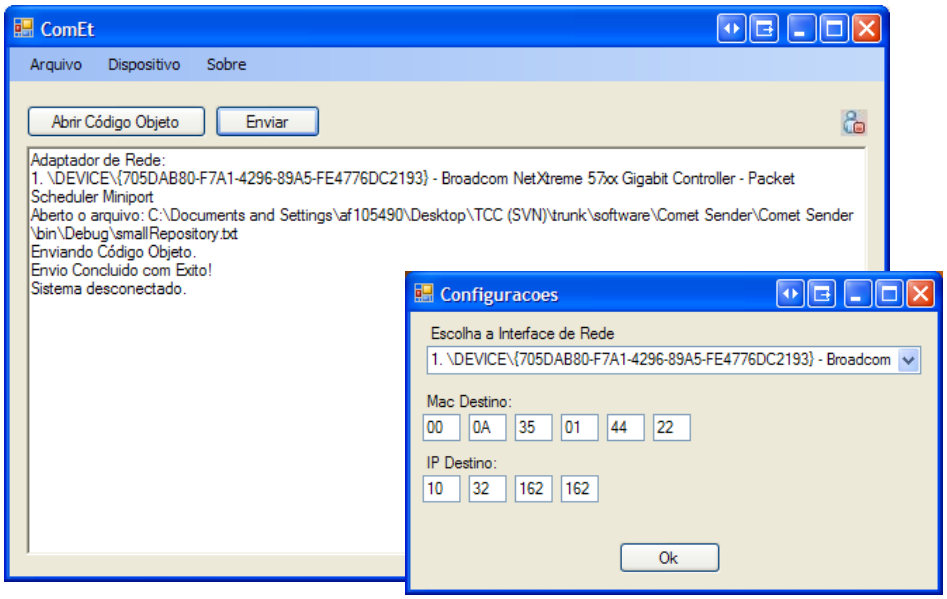

Figura 26 - Interface gráfica do software ComEt.

O software, desenvolvido em linguagem C#, utiliza o *driver* de adaptador de rede NDIS [WOL03] para ter acesso ao dispositivo de rede do computador. A estrutura dessa ferramenta está ilustrada na Figura 27.

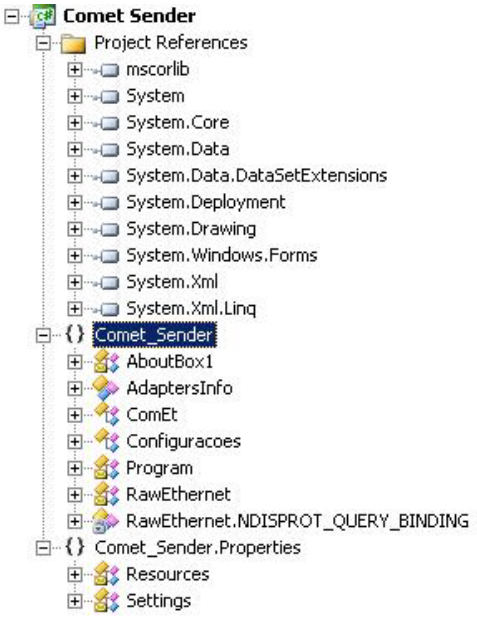

Figura 27 - Estrutura do Software ComEt

A estrutura é apresentada em forma de diferentes símbolos, cada um com seu significado e importância no projeto. A “pasta” indica as bibliotecas utilizadas, as “chaves” o projeto desenvolvido e os demais símbolos são as classes ou estrutura de dados. As classes mais importantes são ComEt e RawEthernet.

A Classe Comet_Sender tem como principais funções *conectar*, *enviarPacote* e *receberDados*, presentes na Figura 28.

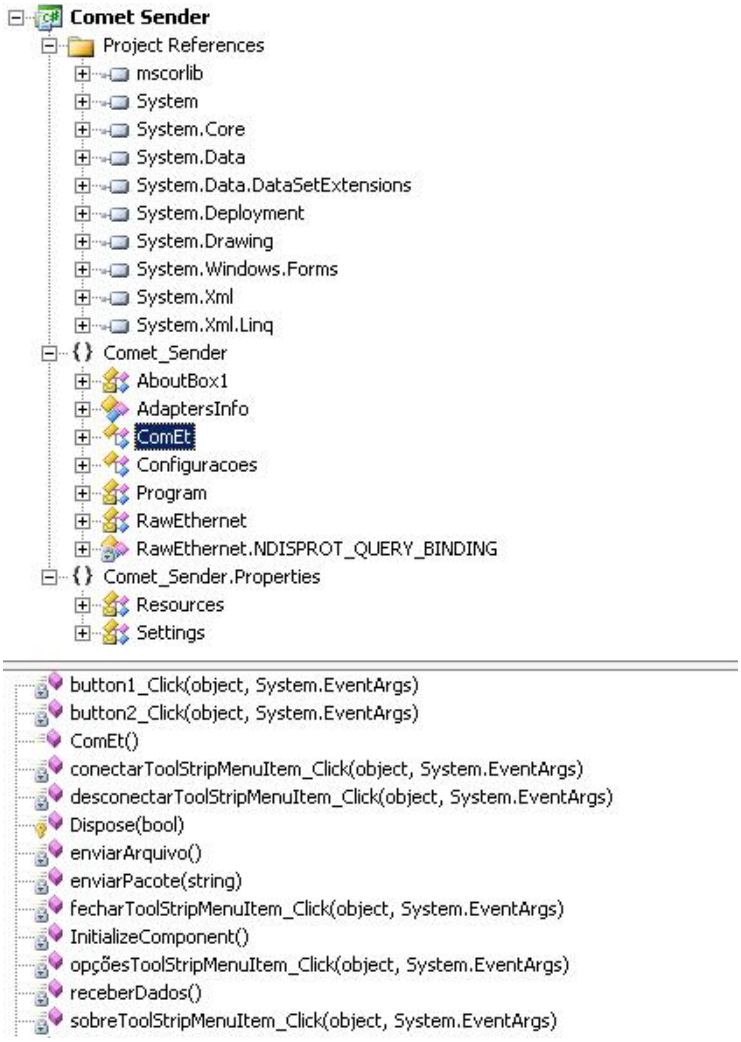

Figura 28 - Classe ComEt

A função *conectar* instancia um adaptador de rede através da classe RawEthernet e conecta no adaptador configurado pelo usuário do software. Esse objeto permite o envio de dados pela rede. A função *EnviarPacote* realiza o empacotamento UDP e utiliza uma primitiva do adaptador para enviar esses bytes. Por fim, a função *receberDados* abre um *socket* UDP, aguarda os pacotes e apresenta na caixa de texto da interface gráfica (Figura 26).

A classe RawEthernet realiza a interação com o driver NDIS, e está representada na Figura 29. As suas funções realizam a conexão com o adaptador da rede, através das funções *OpenDriver* e *BindAdapter*, e o envio dos bytes pela rede, através das funções *WriteFile* e *DoWrite*.

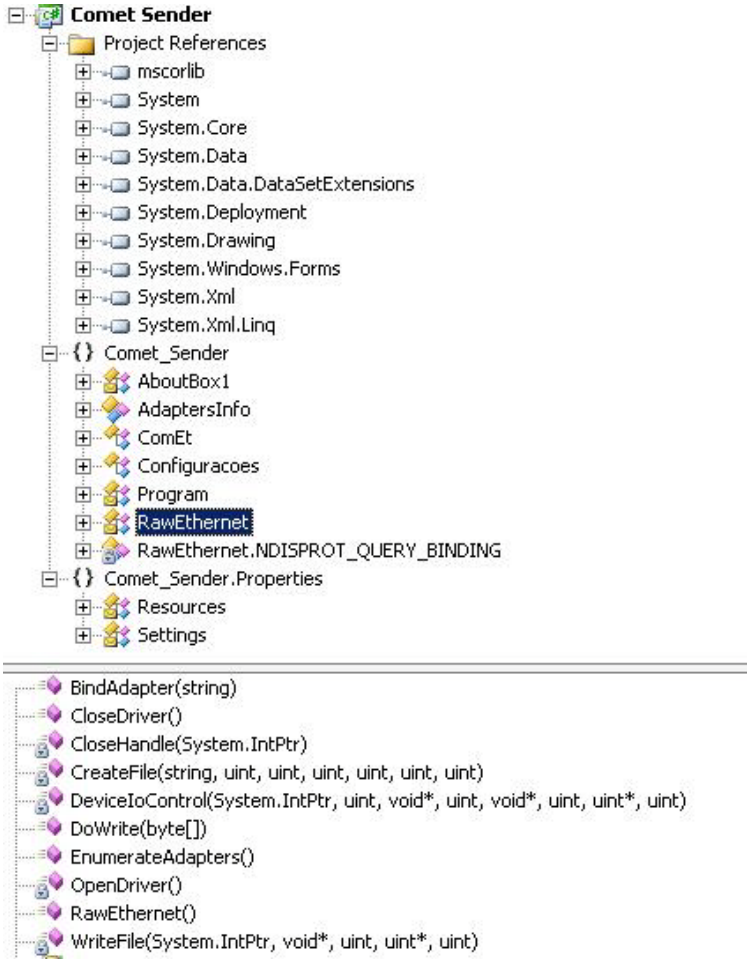

Figura 29 - Classe RawEthernet

2.3 Sistema ConMe

O Sistema ConMe tem por objetivo ser um agente intermediário entre aplicação e a memória DDR2, ou seja, um sistema responsável por realizar a interação entre uma aplicação em hardware e memórias do tipo DDR2 SDRAM. Sua estrutura, conforme a Figura 30, é baseada em dois blocos, uma interface e um controlador. A interface tem por função simplificar o protocolo de comunicação com o controlador e também auxiliar no endereçamento dos dados para aplicações do projetista. O controlador, gerado pela ferramenta Core Generator da Xilinx [KAR08], é o responsável por interagir com a memória fisicamente.

A principal vantagem desse sistema é a sua flexibilidade, pois o bloco Interface é capaz de interagir com qualquer controlador de memórias DDR2 SDRAM gerado pelo Core Generator. Isso permite trabalhar com diferentes memórias DDR2 SDRAM, sem alterar o projeto, na qual esse sistema está inserido.

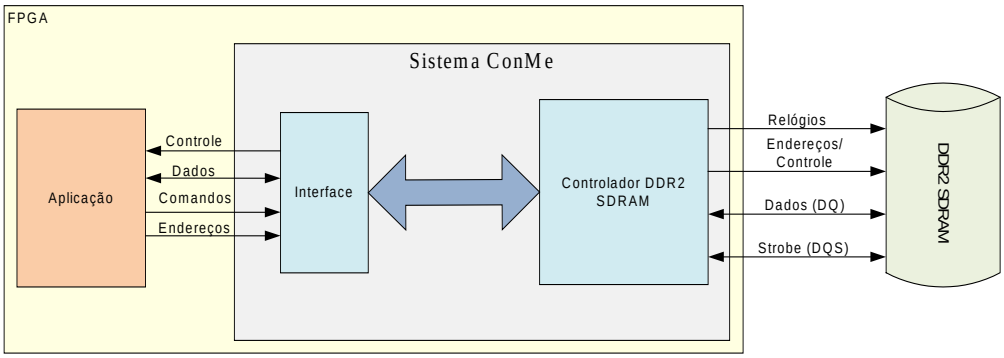

Figura 30 – Diagrama de Blocos do Sistema ConMe

2.3.1 Bloco Interface

O Bloco Interface tem duas funções principais, traduzir o endereçamento da aplicação para o controlador, e simplificar a comunicação com o controlador.

O endereçamento da aplicação possui 27 bits, sendo que em cada endereço é armazenada uma palavra de 512 bits, totalizando uma máximo possível de 8GB de dados. Visto que estamos trabalhando com uma memória de 256 MB, são utilizados apenas os 22 bits menos significativos. Diferentemente, o endereçamento para o controlador possui 31 bits, sendo que em cada endereço é armazenada duas palavras de 128 bits, uma para borda de subida e outra para borda de descida. Esse endereçamento é composto do endereço da aplicação, de 27 bits (22 bits utilizados efetivamente) e mais quatro bits de controle, onde os três bits menos significativos devem permanecer em zero. Assim, temos efetivamente, para este projeto, 23 bits de endereçamento. Além disso, os bits desse barramento de endereço estão divididos em três campos: banco, linha e coluna da memória, conforme ilustrado na Figura 31.

Figura 31 - Organização do barramento de endereço para o controlador.

Para realizar essa tradução de endereços existe uma máquina de estado para a escrita e uma para a leitura, representados pelos fluxogramas da Figura 32 e Figura 33 respectivamente.

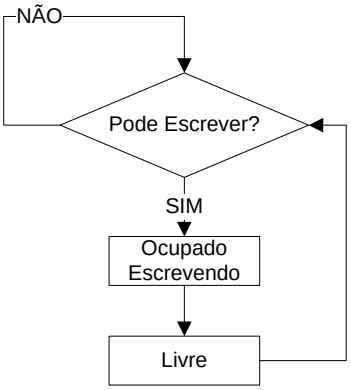

Figura 32 – Fluxograma da máquina de escrita do módulo Interface.

A máquina para a escrita executa os seguintes passos:

- Verifica possibilidade de escrita no controlador;
- Caso esteja disponível indica que este módulo está ocupado;
- Inicia o protocolo de escrita. Esse protocolo utiliza 2 endereços diferentes e 4 palavras de 128 bits, fornecidos pelo barramento de dados provindo da aplicação. Nesse protocolo, o endereço da aplicação é utilizado para todas as quatro palavras, pois este barramento é concatenado com um sinal de controle, o bit mais significativo do campo das colunas. A cada duas palavras esse sinal de controle chaveia, alternando as colunas, conforme especificado pelo controlador.
- Por fim, indica que o módulo já não está ocupado.

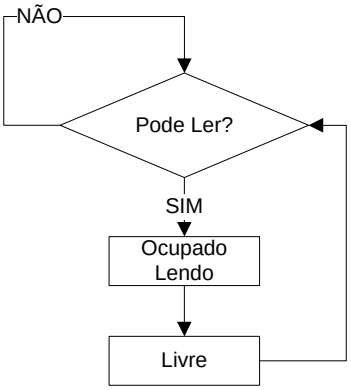

Figura 33 – Fluxograma da máquina de leitura do módulo Interface.

A máquina para a leitura executa os seguintes passos:

- Verifica possibilidade de leitura no controlador;
- Caso esteja disponível indica que este módulo está ocupado.
- Inicia o protocolo de leitura. Esse protocolo utiliza dois endereços diferentes e

4 palavras de 128 bits, representados pelo barramento de dados proveniente da aplicação. Nesse protocolo, o endereço da aplicação é utilizado para todas as quatro palavras, pois este barramento é concatenado com um sinal de controle com um sinal de controle, o bit mais significativo do campo das colunas. A cada duas palavras esse sinal de controle chaveia, alternando as colunas, conforme especificado pelo controlador.

- Por fim, indica que o módulo já não está ocupado.

A interface do Bloco Interface e a aplicação são resumidas em duas etapas, verificar a disponibilidade do sistema e requisitar uma ação. As ações possíveis são escrita, leitura e sem ação (ociosa). Essa comunicação com o ambiente exterior é realizada de maneira síncrona, e sua interface, Figura 34, possui seis sinais.

O sinal *control_ready* representa a disponibilidade do sistema, ou seja, caso esteja livre, esse sinal se encontra em nível lógico alto, porém caso esteja realizando alguma operação (sistema ocupado), se encontra em nível lógico baixo. O sinal *control_rd_data_valid* representa que um dado solicitado está pronto para ser consumido. O barramento *control_cmd*, que possui largura de três bits, representa o comando a ser requisitado ao sistema, podendo ser uma escrita representado pelos bits “000” (em decimal 0), uma leitura representado pelos bits “001” (em decimal 1) ou apenas sem ação representado pelos bits “111” (em decimal 7). É importante ressaltar, que o comando sem ação deve estar sempre presente no barramento quando não for solicitada as outras operações, pois um comando de leitura ou escrita mantida nesse barramento implica em uma nova ação a cada ciclo de relógio. O barramento *control_wr_addr*, de largura de 27 bits, serve para armazenar os endereços durante um comando de leitura ou escrita. Os barramentos *control_wr_data* e *control_rd_data* possuem largura de 512 bits, e são utilizados para enviar e receber os dados, respectivamente.

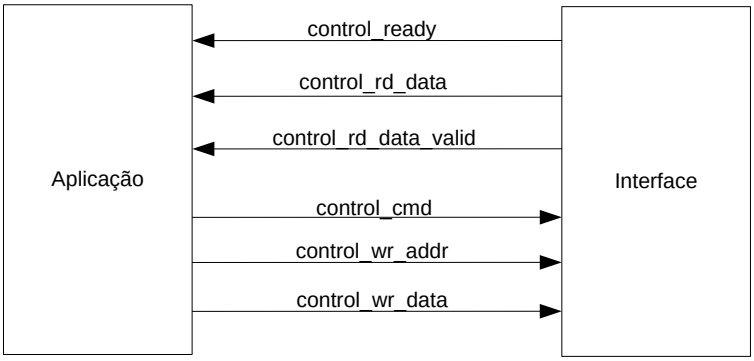

Figura 34 - Interface entre bloco Interface e Aplicação.

2.3.2 Bloco Controlador DDR2 SDRAM

O controlador descrito nesse documento é um controlador gerado através da ferramenta *Core Generator* da empresa *Xilinx* para dispositivos *Virtex-5* [CGG09]. O projeto de controlador gerado é para uma memória DDR2 SDRAM de 256 MB, com freqüência de operação real de 267 MHz, utilizando 64 bits de barramento de dados, e rajadas de 4 palavras por acesso a memória.

Esse controlador utiliza uma interface síncrona para transmissão e recepção de dados. As transmissões são feitas orientadas a rajadas, podendo ser quatro ou oito palavras por rajada. Os acessos a memória iniciam com um comando de ativação, seguido por um comando de leitura ou escrita. Quando é executado o comando de ativação, os endereços registrados estão selecionando o banco e a linha a ser acessado. Quando é executado o comando de leitura ou escrita, os endereços registrados estão selecionando a coluna para o acesso em rajada.

A interface do controlador está representada no diagrama de blocos da Figura 35. O projeto do controlador é dividido em cinco blocos, o *Phy, o Controller, o User Interface, a Infrastructure e o Testbench.*

O *Phy* é responsável pela interface direta com a memória DDR2 SDRAM. Sendo assim, suas funções são instanciar os recursos para gerar os sinais de relógios, de controle, de endereçamento e de dados para a memória; realizar a seqüência de inicialização da memória; calibrar o tempo para captura de dados utilizando a tecnologia *ChipSync* (IDELAY). [XIL09]

O *Controller* é responsável por gerar os comandos para a memória, baseado nos comandos requisitados pelo bloco *User Interface*. Opcionalmente, pode-se implementar um esquema para a gerência de bancos, a fim de melhorar o desempenho.

O *User Interface* dispõe de uma FIFO para interagir com a aplicação específica do usuário. Nesse nível, são executados os comandos de leitura e escrita. Os demais comandos e controles envolvidos em cada processo de leitura ou escrita são realizados pelo bloco Controller.

A *Infrastructure* é responsável por gerar os sinais de relógio necessários utilizando um módulo *Digital Clock Manager*. Também realiza a sincronização dos sinais de reset perante os diferentes domínios de relógio presentes no projeto. Pode-se definir que esse módulo não gerará os sinais de relógio, tornando necessário algum módulo externo ao controlador gerar tais sinais.

O *Testbench* é um módulo que tem por função escrever e ler da memória, apenas para testá-la.

.

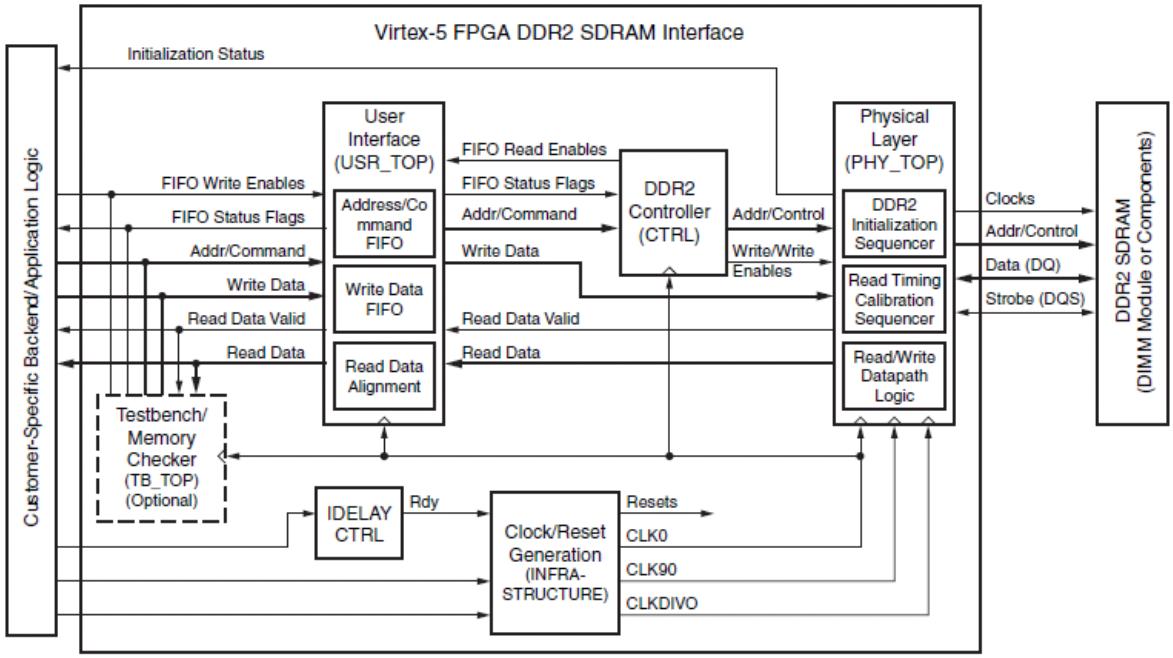

Figura 35 - Diagrama de Blocos da Interface do Controlador DDR2 SDRAM [KAR08]

2.3.3 Comunicação entre o Bloco Interface e Controlador

A comunicação entre o bloco do Controlador e o bloco Interface (Figura 35) é realizada em duas etapas - verificação e requisição. A verificação avalia se o controlador está disponível para atender a requisição. Sua interface com o ambiente externo pode ser observada na Figura 36, que apresenta doze sinais, sendo três deles utilizados para dados, um para endereço e oito para controle.

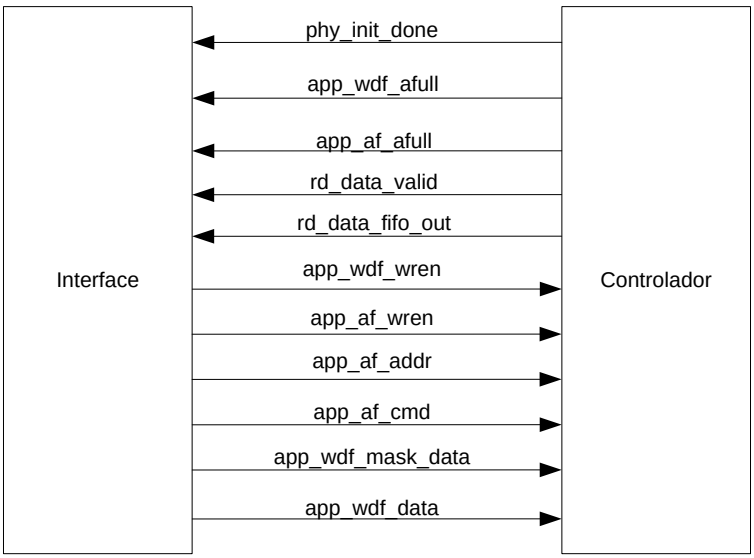

Figura 36 - Interface entre Bloco Interface e Bloco Controlador

O sinal *phy_init_done* identifica se a memória está pronta para o uso, ou seja, se ela já foi inicializada pelo controlador, sendo nível lógico alto para representar esse estado. Os sinais *app_wdf_afull* e *app_af_afull* são controles para informar se as FIFOs do controlador estão cheias, a primeira para identificar a FIFO que armazena os dados e a segunda a que armazena os endereços. Quando cheias, representado por nível alto, é necessário esperar o controlador esvaziar a FIFO para realizar a operação de escrita. O controlador utiliza FIFOs devido ao fato de que existem muitos processos durante a escrita, sendo o tempo desta operação incerto, caso necessite retransmissão, mudar de banco, etc. O sinal *rd_data_valid* representa quando um resultado de leitura é colocado no barramento de dados de leitura, *rd_data_fifo_out*. Quando o sinal *rd_data_valid* fica em nível lógico alto, é iniciada uma rajada de dados, sendo escrita no barramento *rd_data_fifo_out* uma palavra por ciclo de relógio.

Os sinais *app_wdf_wren e app_af_wren* habilitam, respectivamente, a escrita na FIFO de endereços e de dados do controlador. O barramento *app_aff_addr*, de largura de 32 bits, é utilizado para endereçamento. O barramento *app_af_cmd*, de largura de 3 bits, é utilizado para atribuir o comando a ser executado pelo controlador. Existem apenas duas operações possíveis de comando, leitura e escrita, pois as demais operações são reservadas ao controlador. Outros valores não exercem influencia no sistema. O barramento *app_wdf_data* é utilizado para armazenar os dados a serem escritos na memória. Este sinal possui 128 bits, justificado pelo fato de que o controlador escreve na memória duas palavras por ciclo. Posteriormente, o controlador, divide esse barramento em dados da borda de subida e dados da borda de decida (*rising data e falling data*). O barramento *app_wdf_mask_data* tem por função mascarar os dados escritos na memória. Porém, essa função não é utilizada nesse projeto.

A Figura 37 ilustra o protocolo de requisição de escrita na memória.

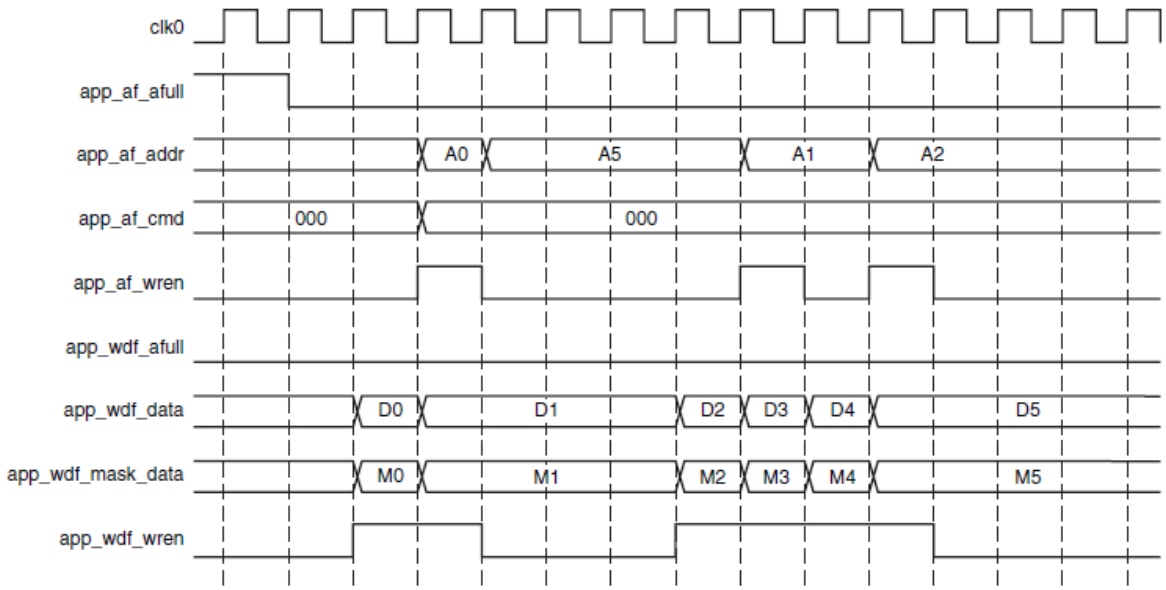

Figura 37 - Protocolo de requisição de escrita na memória. [KAR08]

Antes de realizar esse protocolo é fundamental verificar se a memória já está inicializada, ou seja, se o sinal *phy_init_done* se encontra com nível lógico alto, e se tanto a FIFO para dados quanto para endereços não estão cheias, isto é realizado analisando se os sinais *app_af_afull* e *app_wdf_afull* estão em nível lógico baixo. Sendo assim, o protocolo para requisitar uma escrita na memória pode ser executado. Primeiramente, deve-se avisar o comando a ser executado para o controlador. Isto é feito atribuindo o comando desejado no barramento *app_af_cmd*, e colocando o sinal *app_af_wren* em nível alto por um ciclo. Após

esta ação, é realizada a escrita, onde o endereçamento e os dados são controlados de maneira independente. No mesmo instante de tempo, coloca-se o dado no barramento *app_wdf_data* e o sinal *app_wdf_wren* em nível alto, enquanto que o endereço é colocado no barramento *app_af_addr* e o sinal *app_af_wren* em nível alto. O sinal *app_wdf_wren* deve ficar em nível alto por 4 ciclos, pois a cada ciclo ocorre uma escrita, caracterizando a rajada de 4 palavras. Sempre são definidos dois endereços a cada quatro palavras. Dessa forma, o sinal *app_af_wren* fica alto apenas durante o primeiro ciclo, definindo o primeiro endereço, e durante o terceiro ciclo, definindo o segundo endereço.

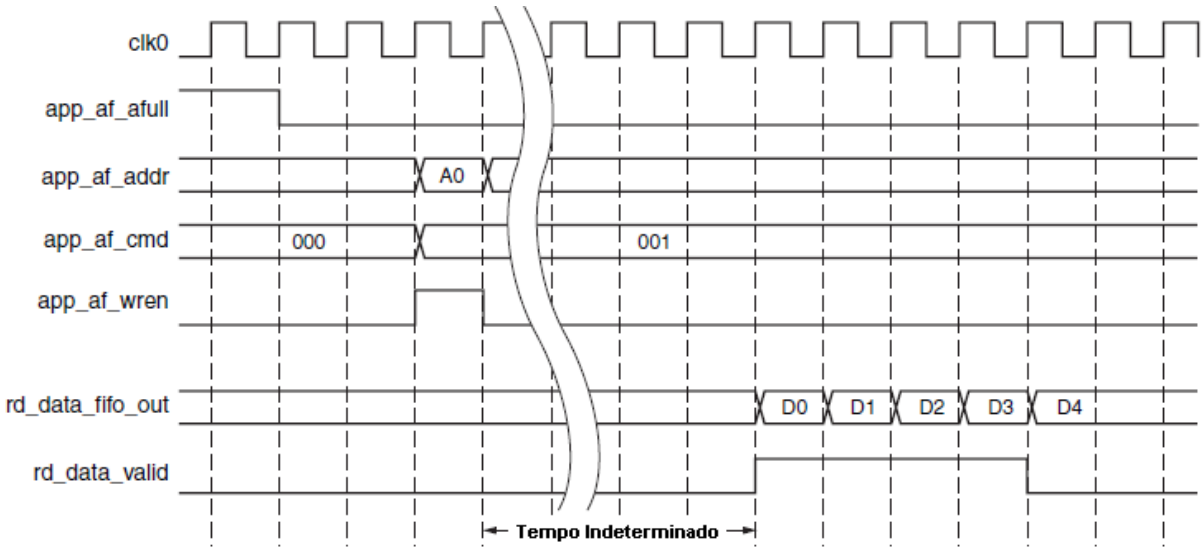

Figura 38 - Protocolo de requisição de leitura da memória. [KAR08]

Antes de realizar esse protocolo é fundamental verificar se a memória já está inicializada, ou seja, se o sinal *phy_init_done* se encontra com nível lógico alto, e se tanto a FIFO para dados quanto para endereços não estão cheias, isto é realizado analisando se os sinais *app_af_afull* e *app_wdf_afull* estão em nível lógico baixo. Sendo assim, o protocolo para requisitar uma escrita na memória pode ser executado. Primeiramente, deve-se avisar o comando a ser executado para o controlador. Isto é feito atribuindo o comando desejado no barramento *app_af_cmd*, e colocando o sinal *app_af_wren* em nível alto por um ciclo. Tendo em vista que o processo de leitura da memória possui um atraso indeterminado, é necessário verificar o sinal *rd_data_valid* ir para nível lógico alto. Quando isso ocorre o barramento *rd_data_fifo_out* disponibiliza quatro palavras, uma por ciclo.

2.3.4 Comunicação com Memória DDR2 SDRAM

O bloco Controlador executa sete comandos na memória: *Precharge*, Auto *Refresh*, *Active*, *Write*, *Read*, *Load* e *Idle*.

O comando de *Precharge* é utilizado para desativar uma linha aberta em um determinado banco. O banco estará disponível para as próximas ativações somente após um determinado tempo depois da execução desse comando.

O comando Auto *Refresh* se deve pelo fato que a DDR2 SDRAM precisa ser atualizada a cada 7,8 microssegundos. Este controle de tempo das atualizações é feito pelo controlador da memória.

O comando *Active* é necessário para ativar uma linha de um determinado banco da memória. Apenas é possível executar os comandos *Read* e *Write* em linhas já ativadas. É necessário observar que se uma linha já estiver ativada, ocorre um conflito. Caso isso ocorra, o controlador precisa executar o comando *Precharge*, e depois, o *Active*. Para reduzir o número de acessos a diferentes bancos a todo o momento, o controlador possui um sistema para gerenciar os bancos.

O comando *Read* é utilizado para iniciar uma rajada de acesso para leitura da linha ativada. O protocolo pode ser observado na Figura 39.

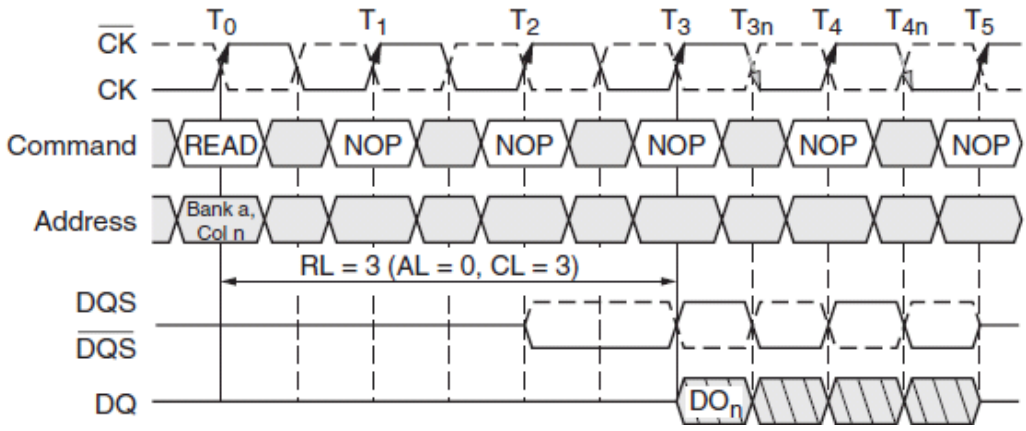

Figura 39 - Protocolo de Leitura DDR2 SDRAM. [KAR08]

O protocolo de leitura inicia atribuindo o endereço da coluna desejada no barramento *Adress*. Logo, atribui-se o comando de leitura no barramento *Command*. Após isso, é necessário aguardar o tempo de latência da leitura, dado pela latência do CAS (*CAS latency*) somado com a latência adicional (AL – *Aditional Latency*). Depois dessa espera, os dados são

recebidos pelo barramento DQ, em rajada (duas palavras por ciclo de relógio). O barramento DQS é utilizado para sincronia durante a leitura.

O comando *Write* é utilizado para iniciar uma rajada de acesso para a escrita na linha ativada. O protocolo pode ser observado na Figura 40.

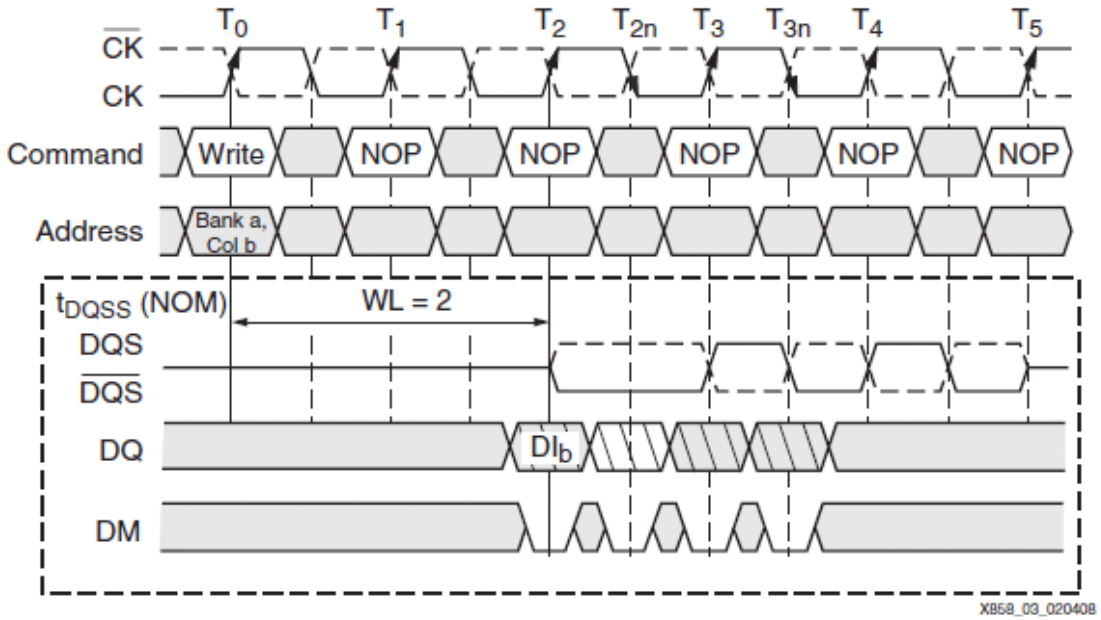

Figura 40 - Protocolo de Escrita DDR2 SDRAM. [KAR08]

O protocolo de escrita inicia atribuindo o endereço da coluna desejada no barramento *Address*. Logo, atribui-se o comando de escrita no barramento *Command*. Após isso, é necessário aguardar o tempo de latência da escrita, dado pela latência de leitura menos um ciclo de relógio. Depois dessa espera, os dados são escritos no barramento DQ, em rajada (duas palavras por ciclo de relógio), durante as bordas do relógio. O barramento DM é utilizado para atribuir máscaras aos dados e o barramento DQS é utilizado para a sincronia durante a escrita.

O comando *Load* é utilizado para realizar a inicialização. E o comando *Idle*, representa nenhum comando a ser executado.

2.3.5 Comunicação com Aplicação

A Figura 41 ilustra o protocolo de escrita executado pela aplicação na interface desse sistema.

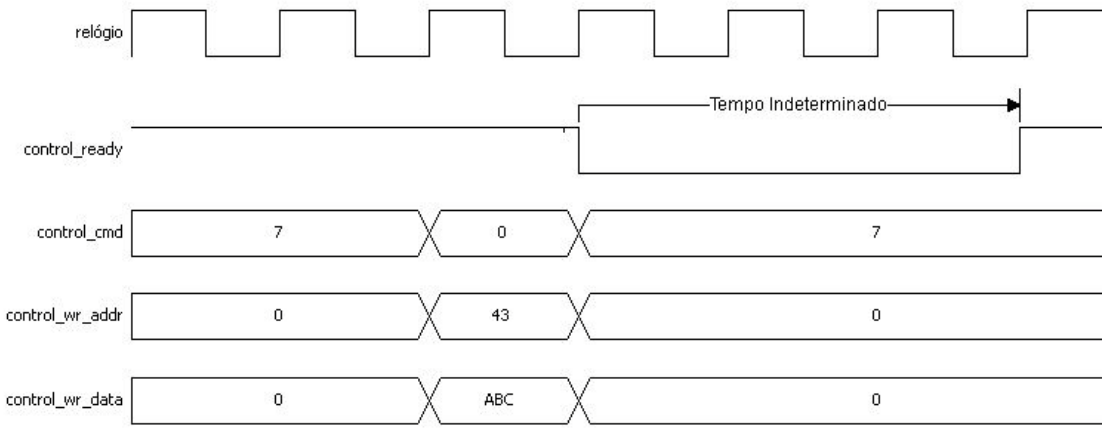

Figura 41 – Protocolo de Escrita de dados na Interface do Sistema ConMe.

Primeiramente é necessário observar se o sinal *control_ready* se encontra com nível lógico alto, representando disponibilidade. Logo, atribui-se ao barramento *control_wr_data* o dado a ser escrito, ao barramento *control_wr_addr* o endereço a ser guardado o dado e ao barramento *control_cmd* o comando de escrita. O *control_cmd* é mantido no barramento apenas um ciclo de relógio, sendo necessário atribuir novamente o comando sem ação a ele. Os outros sinais, relativos a endereço e dados, não importam quando o comando selecionado é o sem ação. Após esse protocolo, o bloco Interface coloca nível baixo no sinal *control_ready*, significando que a requisição está sendo processada, voltando a ser nível lógico alto depois de um tempo indeterminado.

A Figura 42 ilustra o protocolo de leitura na interface desse sistema.

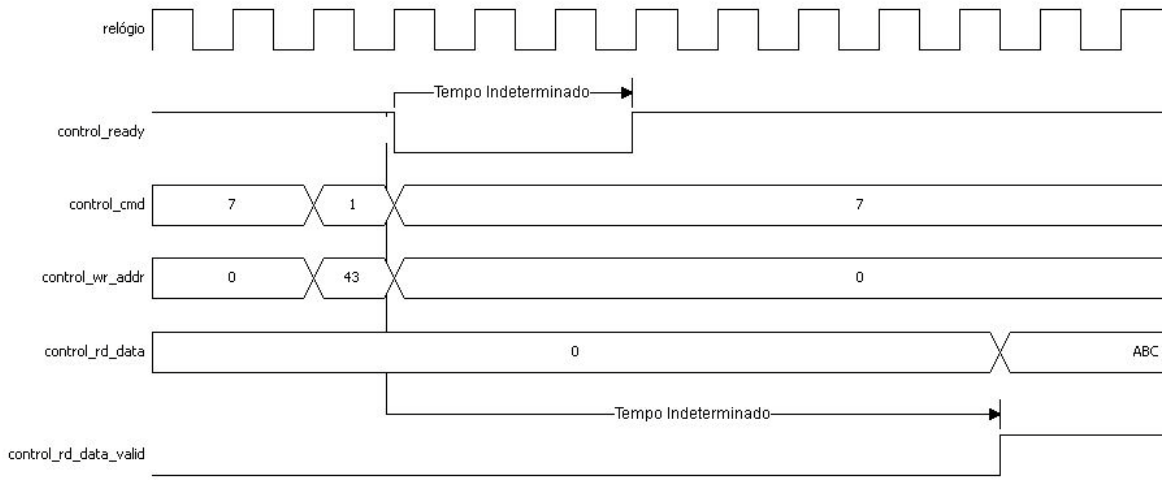

Figura 42 - Protocolo de Leitura de dados da Interface do Sistema ConMe.

Primeiramente é necessário observar se o sinal *control_ready* se encontra com nível lógico alto, representando disponibilidade. Logo, atribui-se ao barramento *control_wr_addr* o endereço a ser lido na memória e ao barramento *control_cmd* o comando de leitura. O *control_cmd* é mantido no barramento apenas um ciclo de relógio, sendo necessário atribuir novamente o comando sem ação a ele. Após esse protocolo, é necessário observar dois sinais, o *control_ready*, que vai para nível lógico alto quando for possível realizar novas operações, e o sinal *control_rd_data_valid*, que vai para nível lógico alto quando o dado requisitado pelo comando de leitura se encontra no barramento *control_rd_data*.

3 INTEGRAÇÃO DOS MÓDULOS DE REFERÊNCIA

Este capítulo descreve a integração dos módulos de referência, e as adaptações necessárias para isso. Essas modificações vão desde a infra-estrutura de software da plataforma HeMPS até a criação de protocolos para comunicação entre o MPSoC e os demais módulos. Ao final, pretende-se obter a comunicação entre os módulos validada funcionalmente para posterior integração.

3.1 Plataforma HeMPS com Repositório em BRAMs e Interface Serial

A primeira atividade de integração compreendeu a prototipação da plataforma HeMPS, sem os módulos ComEt e o Sistema ConMe. Esta prototipação é necessária para avaliar as eventuais modificações na plataforma HeMPS versão 3.7. O que difere esta prototipação de outras realizadas nessa plataforma é as mudanças no microkernel, não realizadas nesta, afim de reduzir a área do projeto em hardware.

O repositório da HeMPS versão 3.7 é gerado em forma de um vetor de dados, onde em cada posição é armazenada uma palavra de 32 bits. Para iniciar o processo de transição da HeMPS original para uma HeMPS com o repositório em memória, a primeira modificação compreendeu a substituição deste vetor por 32 BRAMs, de 16 KB cada. Nesse conjunto de BRAMs, cada uma representa 1 bit, dos 32 que compõem a palavra. O acesso para leitura de dados é o mesmo utilizado anteriormente (Figura 43), visto que a BRAM entrega a palavra no barramento de dados com um ciclo de latência.

Nota-se na Figura 43, que o dado fica disponível no barramento *mem_read* um ciclo após a mudança do barramento de endereços *mem_addr*. Desse modo, não foi necessária nenhuma mudança no acesso ao repositório pelo processador mestre. Portanto, a primeira modificação realizada no MPSoC HeMPS foi a substituição do repositório de tarefas

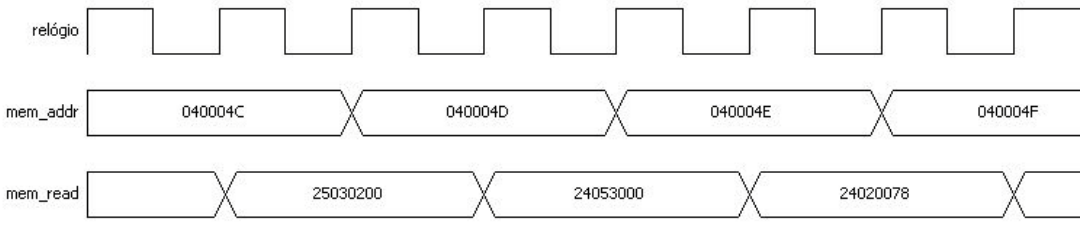

Figura 43 - Processo de leitura da memória externa.

Para geração do repositório em forma de uma cadeia de BRAMs foi desenvolvida uma aplicação que transforma o código objeto das tarefas, gerado pelo HeMPS Generator, em um arquivo que instancia as BRAMs e define as suas respectivas inicializações.

A segunda modificação realizada na HeMPS original compreendeu a validação do módulo serial (UART), para fins de *debug* enquanto os módulos ComEt e ConMe não estivessem prontos.

Para esta ação, foram modificados sinais de controle e o contador responsável por ajustar o módulo à taxa de transferência (*baud rate*) desejada. Devido ao fato de anteriormente o módulo da serial não ser utilizado no ambiente de simulação, foi necessária a validação funcional desse módulo através de simulações RTL.

Para que o módulo serial operasse sem erros foram necessárias algumas modificações. Primeiramente, foi adicionado um sinal de habilitação do *buffer* ao controle de escrita do sinal que indica que a serial está escrevendo o dado no barramento. Outra modificação realizada foi o valor do registrador para o ajuste de *baud rate*. Como o objetivo dessa etapa é apenas validar o sistema em FPGA, optou-se por utilizar o sinal de relógio operando em freqüência baixa, no caso 12,5 MHz. Em seguida, definiu-se o *baud rate* a ser utilizado, no caso, 9600 bps. Com isso calculou-se quantos ciclos de relógio seria necessário aguardar até que o próximo bit pudesse ser enviado para a serial, Equação 1.

Equação 1 - Cálculo de intervalo de tempo entre 2 bits enviados para serial

$$Valor = \frac{Freqüência}{baudrate}$$

Na Figura 44 e na Figura 45 pode-se observar o funcionamento do módulo serial modificado, já inserido dentro do MPSoC HeMPS, utilizado para o envio de mensagens de depuração ao mundo externo.

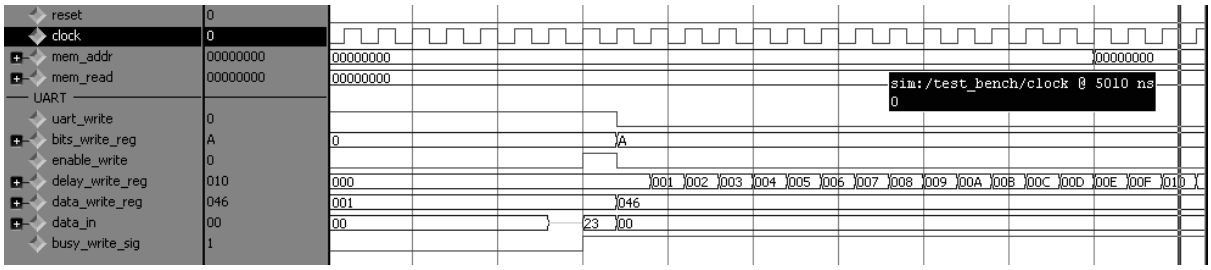

Figura 44 - Início da escrita na serial

Na Figura 44 nota-se a inicialização da escrita na serial pelo sinal *enable_write*. Logo após o sinal *uart_write*, que é o sinal ligado diretamente à serial, vai para o nível lógico zero,

indicando o bit inicial da palavra (*start bit*). Em seguida, o contador de ciclos começa a ser incrementado, de modo a esperar o tempo exato para o envio do próximo bit. Esse comportamento pode ser observado na Figura 45.

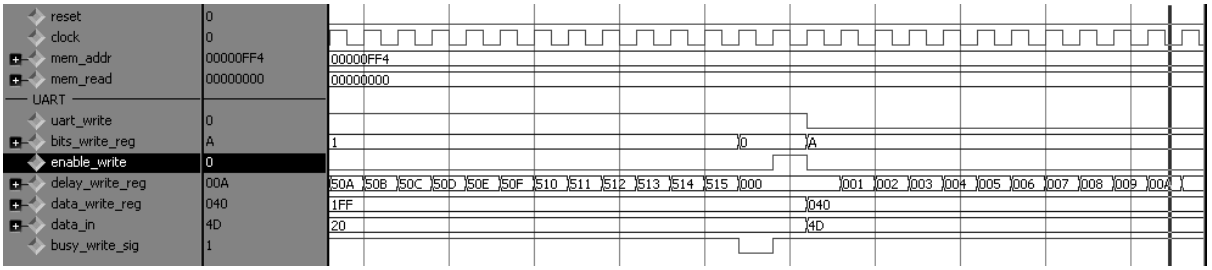

Figura 45 - Finalização da escrita de uma palavra na serial.

Conforme os bits vão sendo escritos na serial, o registrador *bits_write_reg* vai sendo decrementado, até que seja identificado que não existe nenhum bit restante a ser enviado, nesse caso, quando o registrador chega à zero. Na Figura 45 nota-se que existe uma operação de escrita logo em seguida à que foi executada, pelo sinal *enable_write*. Nesse momento o registrador *delay_write_reg* é reinicializado para que seja possível enviar o próximo bit no intervalo de tempo correto. Nota-se também que no momento em que o último bit foi escrito a serial indica que está livre através do sinal *busy_write_sig* em nível lógico zero.

Ou seja, quando existe dado na fila para ser enviado ao mundo externo, o módulo serial entra em funcionamento, e envia um *byte* por vez.

Após a validação funcional foi realizada a prototipação. A configuração escolhida foi um MPSoC de tamanho 2x2, com 1 processador desabilitado. Essa configuração foi escolhida devido ao alto número de BRAMs necessárias no projeto, restringindo a escolha a uma Virtex-2 Pro XC2VP30, que possui 136 BRAMs. Como cada processador Plasma ocupa 32 BRAMs para a memória RAM, o repositório também ocupa 32 BRAMs e mais 1 BRAM utilizada como *buffer* do módulo serial, esse projeto necessita de 129 BRAMs. A aplicação escolhida para o teste foi a *communication*, com 4 tarefas. Assim, processamento, alocação dinâmica de tarefas e comunicação entre os processadores seriam testados.

A placa foi conectada a um computador hospedeiro, e utilizando a ferramenta *HyperTerminal* foi possível capturar os dados oriundos da serial e armazená-los em um arquivo. Para que o software HeMPS Generator pudesse ler o arquivo e assim validar também a prototipação, foi desenvolvida uma ferramenta que adapta os dados provenientes da serial para o padrão esperado pelo HeMPS *Generator*.

Ao final obteve-se a mesma resposta no HeMPS *Generator*, com o arquivo proveniente da simulação funcional e com o arquivo proveniente do FPGA. Assim, pôde-se validar a prototipação do MPSoC HeMPS em FPGA.

3.2 Plataforma Comet e Sistema ConMe

A segunda atividade de integração compreendeu a união da Plataforma ComEt e o Sistema ConMe, a fim de possibilitar escritas e leituras em uma memória DDR2 remotamente. Ambas as arquiteturas não precisaram ser modificadas, pois foi desenvolvido um módulo para controlá-las. Esse módulo, denominado Main Control, traduz as mensagens provindas da ComEt, interpreta os comandos e os executa através do ConMe. A Figura 46 apresenta a arquitetura que integra os módulos ComEt e ConMe.

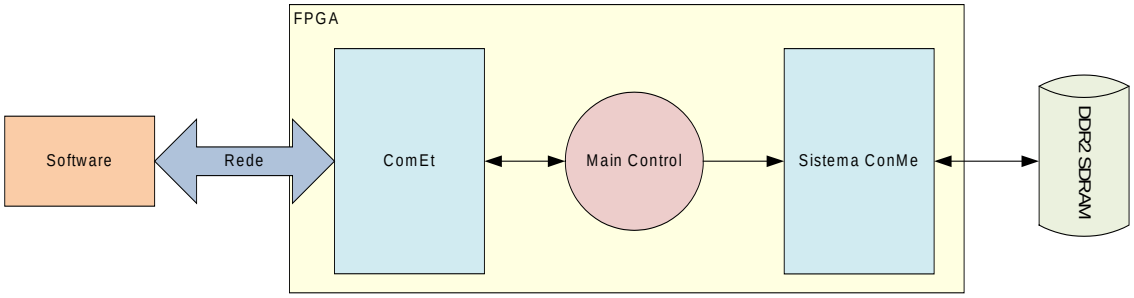

Figura 46 - Arquitetura de validação dos módulos ComEt/ConMe

O computador remoto envia dado através do software, que por sua vez encaminha pacotes UDP pela rede, que chegam à placa de prototipação, onde a Plataforma ComEt desempacota e entrega os dados para o Main Control. Este interpreta os dados e requisita a ação ao Sistema ConMe. Após isso, dados indicando que a ação foi realizada são enviados de volta, até chegar ao software novamente.

Para que o módulo Main Control interprete corretamente os dados do computador remoto, foi elaborada uma estrutura de dados conforme a Figura 47.

Comando	Endereço	Dados
4 bytes	4 bytes	62 bytes

Figura 47 - Estrutura de dados elaborada para o módulo Main Control.

O campo de comando pode ser uma requisição de escrita, decodificado como *0000AADD* em hexadecimal, ou leitura, decodificado como *AAAAAAAA* em hexadecimal. O

campo de endereço indica o endereço a ser lido ou escrito na memória. Visto que no Sistema ConMe o barramento para endereçamento utiliza 27 bits, são desconsiderados os 5 bits mais significativos desse campo de 32 bits. Por último, o campo de dados é utilizado apenas quando é executada uma escrita. Os dados lidos da memória são enviados diretamente ao software sem nenhum padrão, ou seja, sem controle adicional.

Após a recepção dos dados, inicia-se o seguinte processo no Main Control:

- Verifica-se o comando existente no seu respectivo campo;
- Se for de escrita, utiliza o campo de endereço e de dados recebidos para realizar o protocolo de escrita no Sistema ConMe. Caso seja leitura, utiliza apenas o campo de endereço.
- Se escrita, espera sinal do ConMe indicando que não está mais ocupado. Se for de leitura espera um sinal de dado pronto para consumo.
- Envia pacote com dados da leitura, representando que a ação foi realizada. Escrita não possui pacote de retorno.
- Aguarda novo pacote.

Ao final obteve-se uma aplicação de escrita e leitura em memórias DDR2 que executa remotamente. Assim pode-se validar a prototipação do controle para o armazenamento de dados na memória através de um FPGA que se comunica pela rede.

3.3 Plataforma HeMPS e Plataforma Comet

Essa abordagem tem por objetivo integrar a Plataforma ComEt e a plataforma HeMPS, a fim de possibilitar a execução de aplicações na Plataforma HeMPS remotamente, utilizando como repositório de armazenamento módulos Block RAM (BRAM). Para possibilitar tal integração, modificações na estrutura original da HeMPS tiveram de ser realizadas. Além disso, houve a necessidade de desenvolver um módulo intermediário entre o ComEt e a HeMPS, denominado Main Control, sendo este derivado do módulo de mesmo nome, da seção 3.2, no entanto com algumas modificações para suportar as funcionalidades desejadas. Este módulo é capaz de interpretar os dados vindos da HeMPS e os encaminhar ao ComEt (Figura 48).

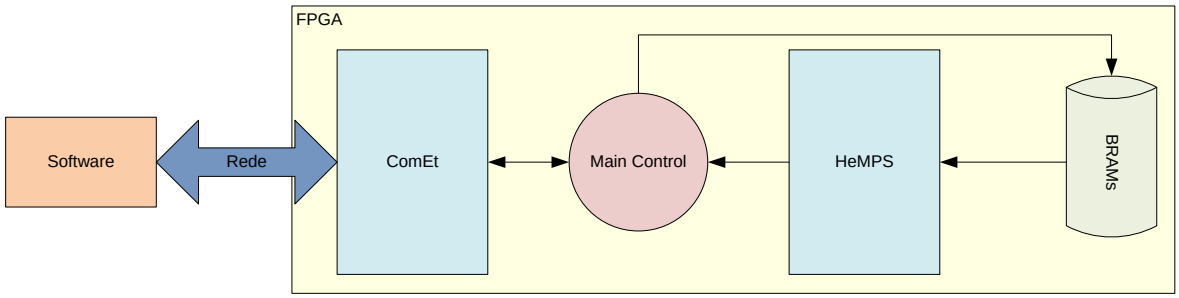

Figura 48 – Arquitetura ComEt/HeMPS

Nesta arquitetura manteve-se a mesma estrutura de dados utilizada na seção 3.2, que apresentava a integração entre a plataforma ComEt e a ConMe. Somente modificou-se a forma como o módulo Main Control interpreta os comandos. O campo de comando continua podendo ser uma requisição de escrita, decodificado como *0000AADD* em hexadecimal. No entanto, não existe mais a requisição de leitura, e sim a solicitação de inicialização da HeMPs, decodificada como *AAAAAAAA* em hexadecimal e uma de solicitação de parada, decodificada como 0000CCAA.

Nesta abordagem, por padrão a HeMPs está em estado de reset. Esta situação só é modificada quando o módulo Main Control interpreta o comando de solicitação de inicialização (AAAAAAAA). O módulo Main Control também instancia Block RAMs (BRAMs) a fim de que o repositório se torne dinâmico, para que seja possível executar diferentes códigos-objetos, trocando o conteúdo do repositório em tempo de execução.

Assim sendo, o software encaminha um pacote UDP destinado ao FPGA. Este por sua vez o interpreta, através da Plataforma ComEt, e encaminha somente os dados necessários ao módulo Main Control. O Main Control interpreta estes dados e caso o pacote de dados contenha um comando de escrita, o módulo Main Control armazena os dados na posição de memória informada pelo campo de endereço. Caso o comando seja de solicitação de inicialização da HeMPs, o Main Control coloca o sinal de *reset* em nível lógico baixo, fazendo com que a HeMPS inicie sua execução. Caso o comando seja de parada, o sinal de *reset* é colocado em nível lógico alto, sendo mantido assim até o próximo comando de inicialização.

3.3.1 Modificações do MPSoC HeMPS

A primeira etapa da adaptação do MPSoC HeMPS é a preparação para a comunicação com o módulo Main Control. Observa-se que, como o repositório poderá ser alterado em

tempo de execução, o número total de tarefas irá ser variável. Anteriormente, esse número era definido em uma biblioteca utilizada em tempo de compilação do *microkernel* dos processadores (mestre e escravo). Semdo assim era um número fixo e era dependente de uma aplicação ou conjunto de aplicações.

Outra modificação necessária é a forma de como o processador mestre envia os dados de depuração para o mundo externo. Antes, o processador tinha apenas uma interface com a serial, sendo um pino para a leitura de dados (*uart_read*) e um pino para escrita de dados (*uart_write*). Com a substituição da serial pelo módulo de comunicação ComEt, foi necessária a retirada do módulo serial e também da fila que armazenava os dados antes deles serem enviados para a serial.

Para que o *microkernel* ficasse independente do número total de tarefas que devem ser executadas pelo sistema, foi necessário também mudar a estrutura do repositório. Para as estruturas que dependiam do número total de tarefas, adotou-se uma nova estratégia. Agora, as estruturas são do tamanho do número máximo de tarefas que podem estar sendo executadas simultaneamente, ou seja, o número total de processadores escravos multiplicado pelo número de páginas da memória disponível para as tarefas em um processador escravo. Exemplo:

HeMPS 3x3; memória de 64KB; tamanho de página de 16KB.

Ou seja: $64 \div 16 = 4\,páginas$ e $3 \times 3 = 9\,roteadores$

Número máximo de tarefas sendo executadas simultaneamente: (9-1) x (4-1) = 24

Com isso garante-se que as estruturas de controle nunca terão sua capacidade máxima excedida, visto que quando uma tarefa é desalocada ela libera o espaço que ocupava nas estruturas de controle.

Já para o *microkernel* do processador mestre é necessário um cuidado maior, visto que alguns mecanismos de controles são realizados baseados no número total de tarefas existentes. A solução adotada foi modificar a estrutura do repositório, colocando no primeiro endereço o número total de tarefas que irão ser executadas e para cada tarefa o endereço do processador no qual ela deve ser alocada (Figura 49).

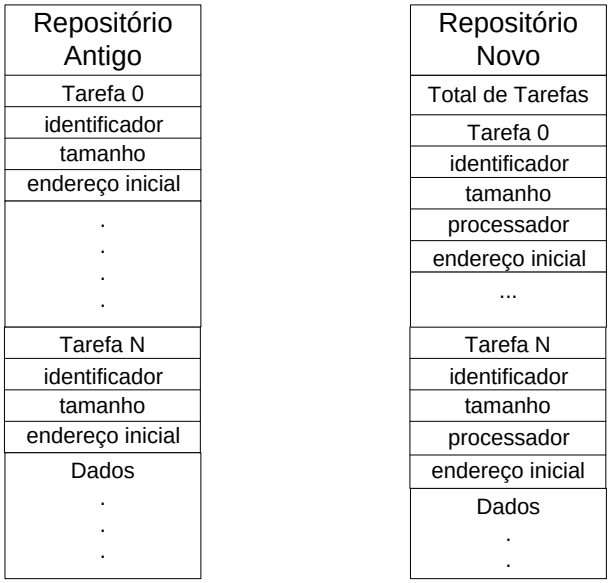

Figura 49 - Estruturas do repositório de tarefas

Quando a tarefa deve ser alocada dinamicamente o campo *processador* deve vir preenchido com ‘1’s.

Dessa forma, o processador mestre lê do repositório primeiramente o número total de tarefas, e em seguida, utiliza o segundo endereço como base para a leitura das tarefas. Assim, quando o repositório é atualizado com um novo conjunto de aplicações e o sistema reinicializado, o processador mestre consegue adaptar-se e realizar as operações de transferência de códigos objetos, escalonamento, depuração e serviços normalmente.

Além disso, foi desenvolvida uma fila assíncrona, que utiliza o relógio da HeMPS na escrita e o relógio do Main Control na leitura. Foi criada também uma nova interface entre o processador mestre e o mundo externo. A Figura 50 apresenta os sinais de entrada e saída da fila e a Tabela 4 descreve cada um desses sinais.

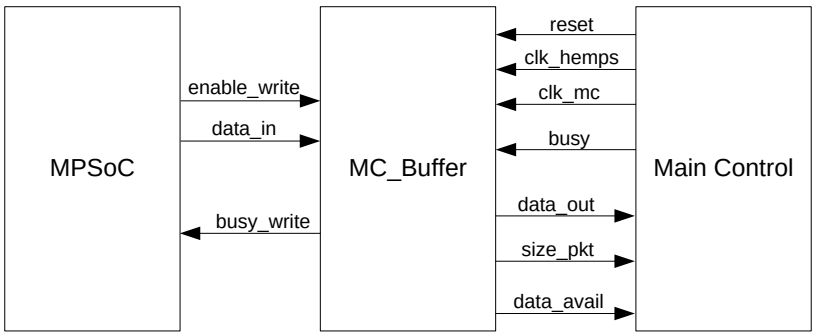

Figura 50 - Interfaces do módulo MC_Buffer

Tabela 4 - Sinais do módulo MC_Buffer

Sinal	Tipo	Descrição
reset	Entrada	Inicializa ou reinicializa o módulo.
clk_hemps	Entrada	Relógio vindo do MPSoC. Utilizado para escrita na fila.
clk_mc	Entrada	Relógio vindo do mundo externo. Utilizado para leitura da fila.
enable_write	Entrada	Habilita a escrita de dados na fila.
busy	Entrada	Informa se o mundo externo está preparado para recepção dos dados.
data_in	Entrada	Dado à ser escrito na fila.
data_out	Saída	Dado lido da fila.
size_pkt	Saída	Informa ao mundo externo a quantidade de dados que irá ser lida e enviada.
busy_write	Saída	Informa ao MPSoC que a fila está cheia.
data_avail	Saída	Informa ao mundo externo a existência de dados disponíveis para leitura.

Os processos de leitura e escrita são dependentes. O processo de leitura entra em ação somente quando o processo de escrita identifica uma determinada condição, no caso, a chegada de uma nova linha (\n) ou quando a quantidade de dados existentes na fila chega ao tamanho máximo do pacote que pode ser enviado para ao Main Control. Quando isso acontece, o processo de leitura é disparado. No início do processo de leitura é calculada a quantidade de dados que será enviada, em *bytes*, e é informada a disponibilidade de dados para Main Control através do sinal *data_avail*. Em seguida, o processo fica esperando a resposta de que os dados podem começar a ser enviados através do sinal *busy*. Quando isso acontece é habilitada a leitura da fila pelo sinal *enable_read* e o processo fica em *loop* até que a quantidade de dados, calculada anteriormente, seja enviada. Após isso, o processo de leitura fica esperando novamente pelo processo de escrita. O Main Control então transfere este dados ao módulo ComEt que empacotará os dados e os transmitirá ao computador remoto.

Ao final, o MPSoC HeMPS, adaptado de maneira a interagir com a plataforma ComEt, obteve um meio rápido de comunicação entre o hospedeiro e o FPGA. Assim, permitindo a realização da depuração das aplicações em tempo de execução.

3.4 Plataforma de Prototipação

A prototipação em FPGA foi realizada em uma plataforma desenvolvida pela empresa *Hitech Global*, Figura 51. Esta plataforma dispõe de um FPGA *lx330t*, da família *Virtex 5* fabricado pela *Xilinx* [XIL09]. Este dispositivo tem o equivalente a 51.840 *slices* (onde, cada *slice* possui quatro LUTs, e quatro *flip-flops*) e permite a inserção e remoção de núcleos de

hardware em tempo de execução (reconfiguração parcial e dinâmica). Este FPGA também é dotado de 324 blocos de BRAMs de 36 Kbits cada, totalizando 11.664 Kbits de memória interna. A plataforma conta também com dois MACs Ethernet, memória externa SO-DIMM DDR2 e barramentos SATA e PCI-e. Um dos motivos da escolha desta plataforma foi a existência deste dispositivo no GAPH, e principalmente, pela grande área de prototipação oferecida pelo mesmo.

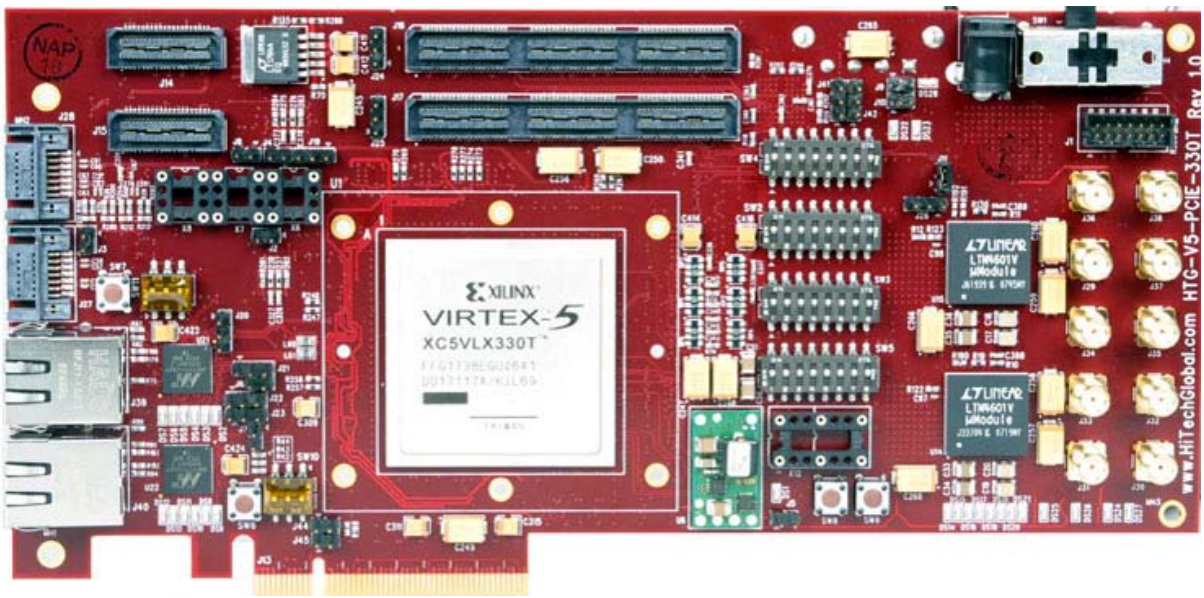

Figura 51 - Plataforma de Prototipação HTG-LX330T, com FPGA Virtex-5 330T

4 ARQUITETURA HEMPS STATION

Este Capítulo descreve a principal contribuição do presente trabalho, o ambiente HeMPS Station. Isso é atingido através da integração das arquiteturas base, descritas no Capítulo 2, com as demais adaptações realizadas, descritas no Capítulo 3, e o desenvolvimento do módulo de controle central, chamado Main Control. Além disso, foram necessárias outras adaptações no MPSoC HeMPS, detalhadas no decorrer desse Capítulo.

4.1 HeMPS Station

O conceito HeMPS Station define um ambiente dedicado para MPSoC, basicamente um conceito para a estrutura de um sistema, capaz de avaliar o desempenho de aplicações embarcadas distribuídas em determinada arquitetura rodando em um FPGA. Esse ambiente contém ferramentas dedicadas executando em um determinado hospedeiro, interface de comunicação rápida entre hospedeiro e MPSoC para viabilizar avaliação do sistema durante a execução, e uma estrutura de monitoração inserida no MPSoC, permitindo captura de dados de desempenho. Portanto, HeMPS Station é composto por um MPSoC, uma memória externa e um sistema de interface hospedeiro/MPSoC, ilustrado na Figura 52. Na prática, o HeMPS Station foi implementado através da Plataforma HeMPS (MPSoC), Sistema ConMe (memória externa) e Plataforma ComEt (interface rápida de comunicação entre hospedeiro e MPSoC) interligados pelo módulo de controle Main Control, conforme mostrado na Figura 53.

O ambiente inicia a partir de comandos enviados pelo software. Esses comandos trafegam em pacotes UDP pela rede. Ao chegar no FPGA, a Plataforma ComEt realiza o processamento para entregar os dados ao Main Control. Este por sua vez, interpreta as funções e realiza as ações necessárias, atuando com o Sistema ConMe ou com a HeMPS. Durante a execução, o Main Control recebe dados da HeMPS, que são repassados a Plataforma ComEt, chegando novamente ao software.

As funções possíveis são conectar, carregar dados na memória, iniciar execução das tarefas e desconectar. Ao conectar o sistema armazena o endereço do computador hospedeiro e abre uma sessão. Em seguida é necessário carregar alguma aplicação para ser executada pelo MPSoC. Isso é feito com o comando carregar dados, onde um arquivo com a aplicação é enviado pelo software. Após carregar a aplicação, o software executa, automaticamente, o

comando de iniciar. Durante a execução, resultados da aplicação são enviados pelo ambiente ao software, sendo exibidos em uma janela de depuração. Realizado esse processo, deve ser executado o comando de desconectar, de modo a liberar o ambiente para que outros usuários possam utilizá-lo.

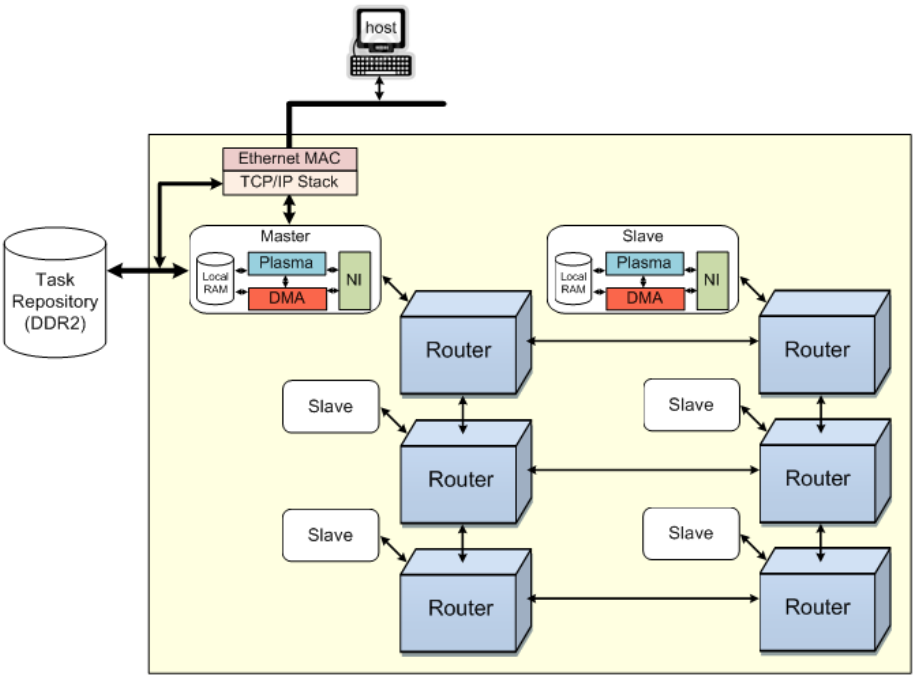

Figura 52 - HeMPS Station – Estrutura Teórica

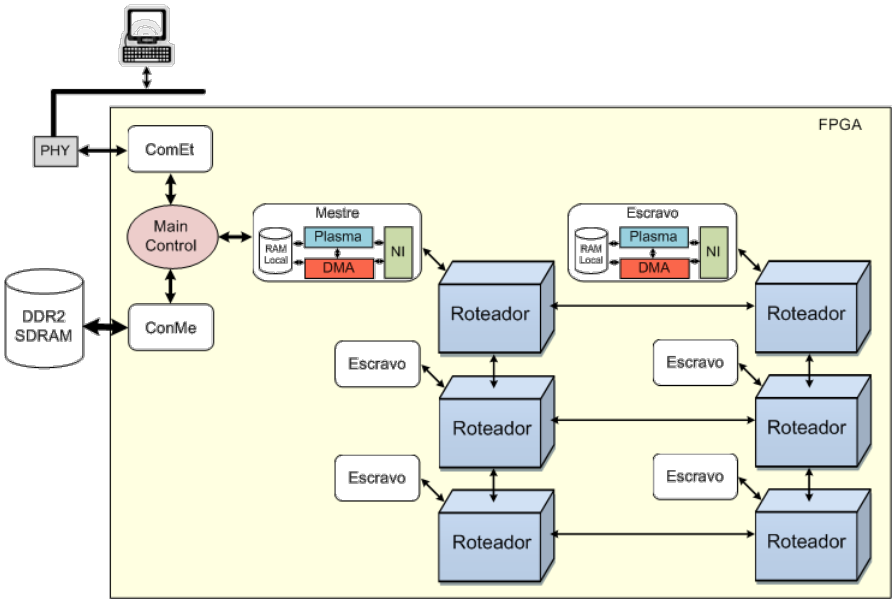

Figura 53- HeMPS Station – Estrutura Prática

Para que o módulo Main Control interprete corretamente os dados do computador remoto, foi elaborada uma estrutura de dados conforme a Figura 54.

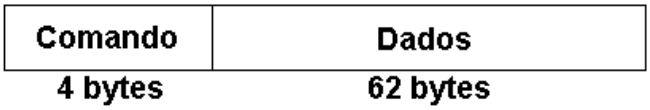

Figura 54 - Estrutura de dados do HeMPS Station

Diferentemente da estrutura elaborada para arquitetura entre Comet e ConMe, nesse contexto não é necessário o elemento endereço, pois as aplicações são organizadas seqüencialmente no repositório. Isso se justifica pelo fato de que os dados de aplicações da HeMPS estão localizadas na memória em ordem seqüencial, ou seja, iniciando no endereço zero até o tamanho da aplicação.

4.2 Modificações na HeMPS

A última etapa da adaptação da HeMPS diz respeito à comunicação entre o MPSoC e o módulo Main Control. As modificações comentadas anteriormente, em relação à comunicação com o módulo ComEt, continuam fazendo parte da versão final da HeMPS, visto que o módulo Main Control irá respeitar o protocolo comentado anteriormente. Porém ainda é necessária uma adaptação entre MPSoC e memória DDR.

Atualmente, com o repositório implementado em forma de um vetor ou em forma de conjunto de BRAMs, quando o barramento de endereços muda, a palavra fica pronta no barramento de dados um clico de relógio depois, tanto quando acessada pelo processador mestre como pelo DMA. Isso não ocorre na DDR, pois existe uma latência desconhecida a cada acesso feito à memória. Logo, é necessário um protocolo para a comunicação entre memória e processador/DMA.

Como o processador e DMA lêem do repositório, a solução encontrada foi ler o dado somente após o sinal *data_valid,* que indica que o dado no barramento está válido, ficar em nível lógico um. O DMA somente realiza o acesso ao repositório após ser configurado pelo processador mestre, assim, quando ele inicia a leitura no repositório, o processador mestre fica liberado para realizar outras atividades. Logo, o repositório é acessado pelo DMA ou pela CPU, nunca por ambos ao mesmo tempo.

É importante lembrar que o DMA também acessa a memória RAM do processador mestre. Então, é necessário tomar o cuidado de esperar pela confirmação de dado válido somente quando o DMA necessitar acessar o repositório.

O DMA independe de dados do processador após ser ativado pelo mesmo, e indica que está em execução pelo sinal *active*. Logo, pode-se modificar o processo de leitura de dados do DMA sem que isso interfira no restante do sistema. A solução adotada foi adaptar o módulo DMA já existente, fazendo com que apenas o processador mestre utilize esse DMA modificado. As modificações realizadas foram:

a) Inclusão do pino de entrada *data_valid.*

b) Inclusão de estados na máquina de estado.

c) Modificações em sinais internos de controle.

A Figura 55 mostra o diagrama de estados do módulo DMA modificado.

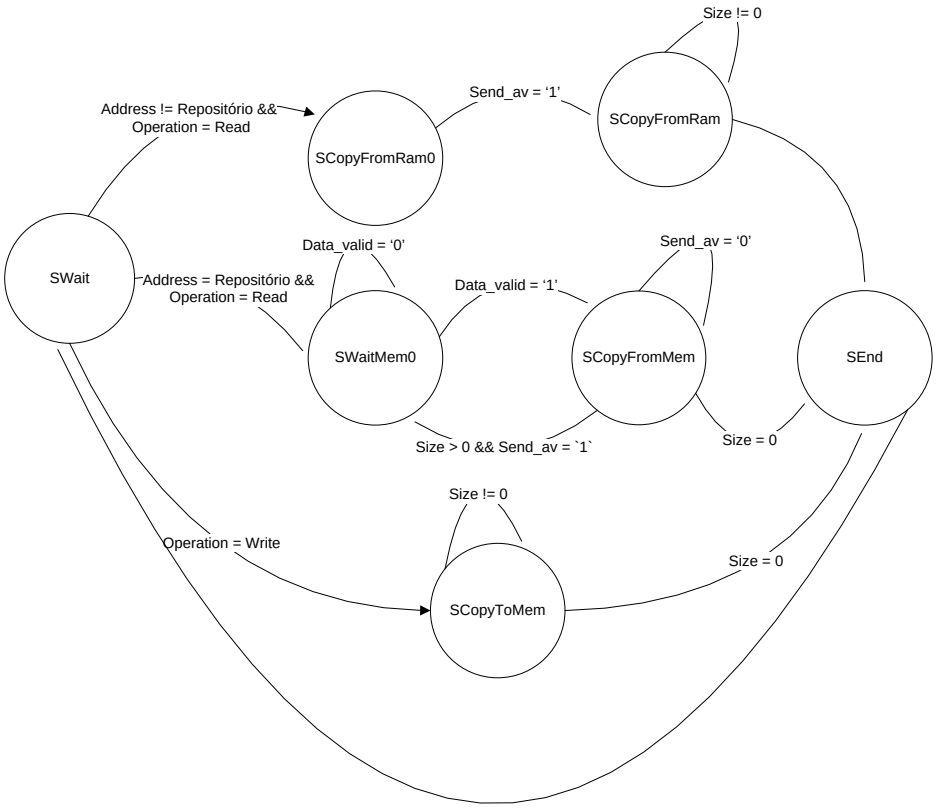

Figura 55 - Máquina de estados do módulo DMA modificado.

No estado *SWait* o DMA fica esperando a sua configuração, realizada pela CPU. Nesse estado é capturada a quantidade de dados que devem ser lidos no registrador *size*, o endereço inicial no registrador *address* e a operação a ser realizada, leitura ou escrita.

Caso a operação seja de escrita, a máquina vai para o estado *SCopyToMem*. Essa operação é realizada somente na memória RAM do processador. Como a memória RAM continua sendo implementada em um conjunto de BRAMs, esse estado não foi alterado. Caso a operação seja de leitura e o endereço não referencie uma posição do repositório, a máquina

de estados vai para o estado *SCopyFromRam0* e segue o fluxo anterior de operação. Caso contrário, a máquina vai para o estado *SWaitMem0* e decrementa o número de palavras lidas. Isso acontece porque quando o barramento de endereços muda, o módulo que gerencia a memória interpreta isso como uma requisição de dados. Ou seja, logo que é recebido o endereço inicial de leitura a memória DDR começa a buscar o dado, por isso a quantidade de dados lidos é decrementada. Quando a máquina chega ao estado *SWaitMem0* ela fica esperando a memória DDR avisar que o dado no barramento está válido através do sinal *data_valid*.

Em seguida a máquina vai para o estado *SCopyFromMem*. Esse estado é responsável por controlar os dados que estão sendo lidos. Caso a quantidade de dados lidos já tenha chegado ao desejado, a máquina vai para o estado *SEnd*, finaliza sua leitura e fica esperando novamente a configuração do processador, caso contrário a máquina vai para o estado *SWaitMem0* e fica esperando novamente a sinalização de dado válido por parte da memória DDR.

Assim consegui-se adaptar o processador Plasma e o módulo DMA para acesso ao repositório de tarefas implementado em uma memória DDR.

4.3 Main Control

Esse módulo é responsável por interagir e controlar todos os módulos do ambiente HeMPS Station. Dentre as suas funções pode-se citar interpretação de comandos, requisição de escrita e leitura da memória externa, inicialização e envio de dados para o MPSoC HeMPS, e envio de dados de depuração.

O Main Control realiza uma gerência dos dados e dos endereços, a fim de organizar o acesso as informações. Primeiramente, observa-se que o ComEt disponibiliza quinhentos e doze bits de dados, representando dezesseis palavras para a HeMPS, que trabalha com trinta e dois bits. Visto que é escrito na memória estes quinhentos e doze bits por endereço, é realizado uma tradução dos endereços da memória externa com os endereços requisitados pelo MPSoC. Assim, surge o conceito de uma pequena memória *cache* nesse módulo, pois durante uma leitura é registrado dezesseis palavras. Nas próximas quinze requisições de leitura da HeMPS, os dados já se encontram no registrador (*cache*). Quando o endereço referenciar um dado que não está na *cache*, o sistema interpreta esse fato como *cache miss*, e realiza uma nova leitura da memória externa.

O Main Control utiliza três estruturas de controle independentes para controlar todo o sistema. A primeira, realiza a leitura do receptor do ComEt, interpreta o comando provindo e realiza a devida ação. A segunda, realiza a entrega de dados da leitura da memória externa para o MPSoC HeMPS. Por fim, a última estrutura consome dados de depuração da fila do MPSoC e os envia ao transmissor do ComEt.

4.3.1 Estrutura de Controle Principal

O controle realizado por essa estrutura representa a parte principal da integração dos módulos de referência. O funcionamento dessa estrutura de controle é implementado conforme o diagrama de estados ilustrado pela Figura 56.

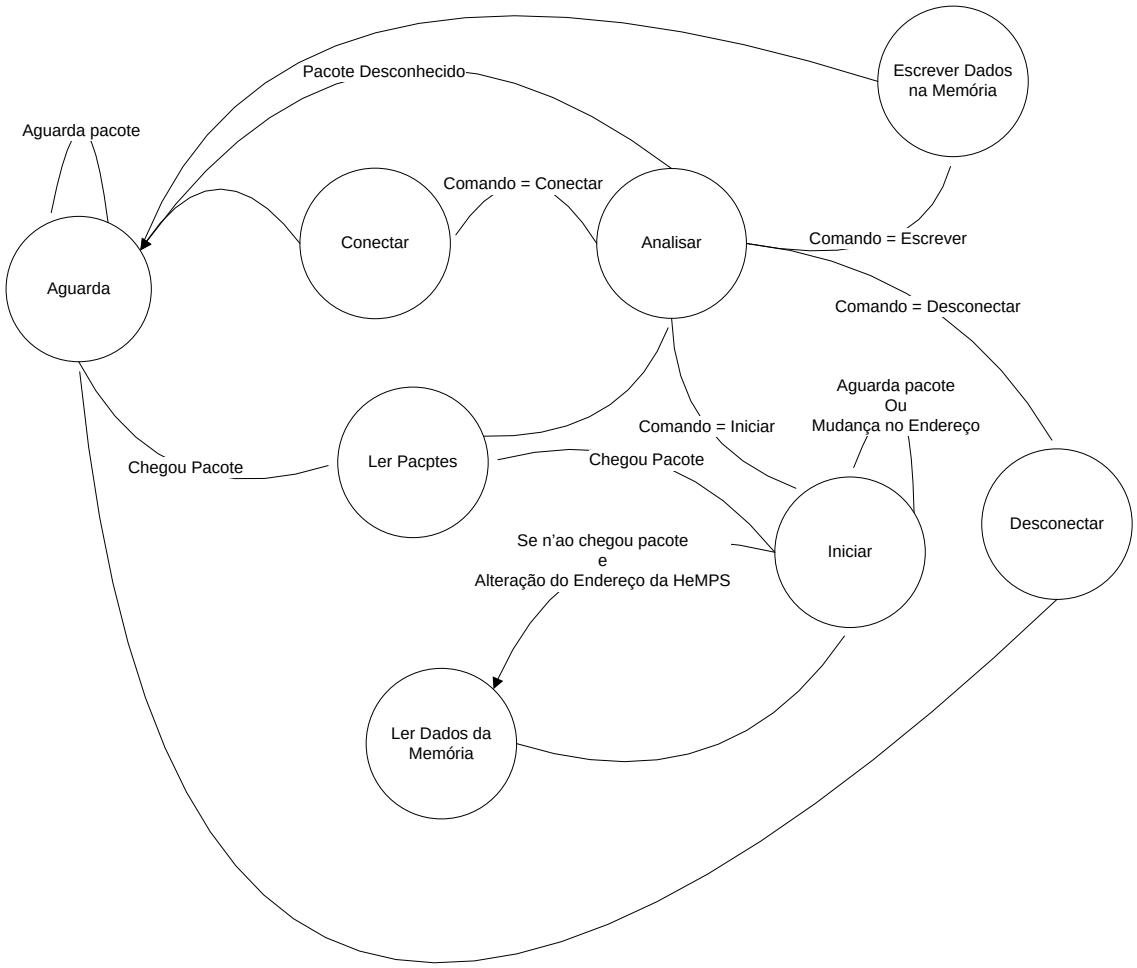

Figura 56 - Diagrama de estados da estrutura principal de controle do módulo Main Control.

O sistema inicia sempre no estado *Aguarda*. Esse estado espera um sinal indicando que chegou um pacote na Plataforma ComEt. Assim, passa-se para o estado *Ler Pacotes*, que faz a leitura dos dados do pacote, seguindo no próximo ciclo para o estado *Analisar*.

O *Analisar* identifica o comando que está inserido nos dados lidos. Caso seja identificado um comando de conectar, o próximo estado é o *Conectar*. Se o comando for escrever, passa-se para o estado *Escrever Dados na Memória*. Se reconhecer o comando iniciar vai para o estado *Iniciar*. E por fim, se for o comando desconectar, o próximo estado é o *Desconectar*. Caso o comando não seja reconhecido, os dados são descartados e volta-se ao estado inicial *Aguarda*.

O estado *Conectar* não realiza nenhuma ação em especial, pois é utilizado pela Plataforma ComEt. Esse estado existe para possibilitar a inserção de algum controle em trabalhos futuros.

O estado *Escrever Dados na Memória* realiza a escrita dos dados na memória. O endereçamento é seqüencial, ou seja, a cada dado escrito o endereço é somado. O endereçamento na memória só é reiniciado quando executado o estado *Desconectar*.

O estado *Iniciar* ativa o MPSoC HeMPS, e aguarda dados tanto do MPSoC quanto do ComEt. Caso tenha um novo pacote no ComEt, volta-se para o estado *Ler Pacotes*. Se não tiver pacotes, é verificado se o endereço anterior é diferente do atual, representando uma requisição de leitura pela HeMPS. Assim passa-se para o estado *Ler Dados da Memória*. Esse estado apenas realiza a ação de requisitar a leitura. Se o dado já estiver registrado (devido a leituras anteriores), um sinal é acionado avisando que o dado está pronto. Caso *o endereço referenciado pela aplicação não aponte para aquele conjunto de dados registrados*, é realizada uma leitura na memória. Retornando em seguida novamente para o estado *Iniciar*.

O estado *Desconectar* realiza a reinicialização de todos os registradores, inclusive o MPSoC HeMPS. Assim, seu próximo estado é o estado inicial de todo o sistema (*Aguarda)*.

4.3.2 Estrutura de Controle para o Envio de Dados à HeMPS

O controle realizado por essa estrutura representa a parte de transmissão dos dados requisitados pelo MPSoC HeMPS. O funcionamento dessa estrutura de controle é apresentado no diagrama de estados, ilustrado na Figura 57.

O estado inicial *Espera* aguarda uma indicação de que o dado de leitura está disponível para ir para o estado *Enviar Dados à HeMPS*. Isso pode ser realizado pela estrutura de

controle principal quando o dado já se encontra registrado no Main Control, ou pelo Sistema ConME, quando é necessário realizar uma leitura da memória externa.

O estado *Enviar Dados à HeMPS* coloca o dado no barramento de dados do MPSoC e aciona um sinal de dado válido. Em seguida, retorna ao estado *Espera*.

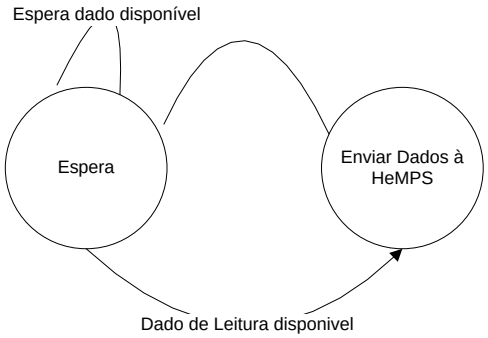

Figura 57 - Diagrama de estados para a estrutura de controle de envio de dados à HeMPS

4.3.3 Estrutura de Controle de Depuração

O controle realizado por essa estrutura representa a parte de transmissão dos dados de depuração ao ComEt, que vem do MPSoC HeMPS. O funcionamento dessa estrutura de controle é representado pelo diagrama de estados, ilustrado pela Figura 58.

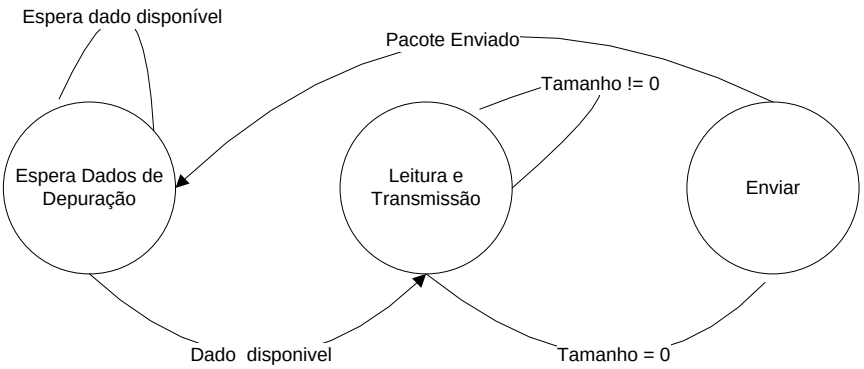

Figura 58 - Diagrama de estados da estrutura de depuração do módulo Main Control.

O estado inicial *Espera Dados de Depuração* aguarda um sinal do MPSoC avisando que existe dado de depuração a ser enviado ao computador hospedeiro. Assim, passa-se para o estado *Leitura e Transmissão*.

O estado *Leitura e Transmissão* realiza a escrita na FIFO do ComEt na medida em que lê a FIFO de depuração da HeMPS. Esse processo é realizado até que o tamanho do pacote de informação seja atingido. Após isso, passa-se para o estado *Enviar*.

O estado *Enviar* define o tamanho do pacote para o ComEt e avisa que deseja enviar dados. Esse estado aguarda um sinal indicando que o pacote foi enviado, assim retornando ao estado inicial *Espera Dados de Depuração*.

5 VALIDAÇÃO

De forma a se validar o hardware desenvolvido, cada módulo do ambiente HeMPS Station foi validado individualmente - plataforma HeMPS, plataforma ComEt e sistema ConMe. Estes módulos foram simulados de forma individual, e posteriormente integrados. Além disso, a validação dos protótipos em FPGA seguiu a mesma metodologia. Para a realização das simulações fez-se uso da ferramenta *ModelSim*, e para o desenvolvimento da validação em hardware, utilizaram-se as ferramentas da fabricante *Xilinx*, sendo elas o software ISE, para a realização da síntese lógica e física, e o *ChipScope* para análise em tempo de execução. Como apresentado no capitulo anterior, ambientes de validação foram/estão sendo desenvolvidos de maneira a verificar o comportamento dos módulos ao serem conectados aos demais.

5.1 Aplicações Embarcadas

As aplicações utilizadas para a validação do ambiente devem apresentar a mesma estrutura das aplicações já existentes na ferramenta *HeMPS Generator*. Para a validação funcional utiliza-se a aplicação *communication*, presente na ferramenta *HeMPS Generator*, pelo fato de ser uma aplicação simples e com uma maior facilidade de depuração. Para a fase de prototipação pretende-se desenvolver uma aplicação com um número maior de tarefas, visto que o sistema alvo é um MPSoC de tamanho 4x5, ou seja, utilizando 20 processadores.

A aplicação *communication* contém um conjunto de 4 tarefas, chamadas de *taskA*, *taskB*, *taskC* e *taskD*. As tarefas inicias dessa aplicação são *taskA* e *taskB*. Essas tarefas iniciam sua execução enviado mensagens para a tarefa *taskC*, que por sua vez, envia cada mensagem recebida à tarefa *taskD*. O grafo dessa aplicação pode ser observado na Figura 59.

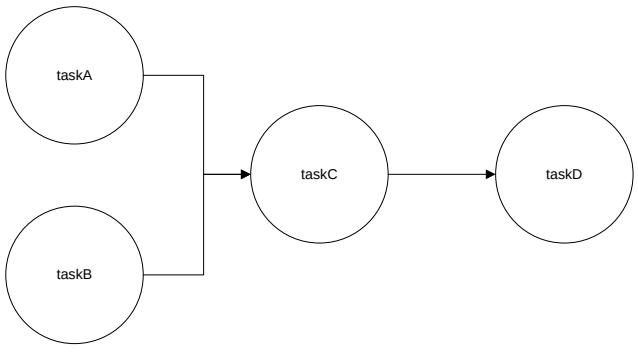

Figura 59 - Grafo da aplicação *communication*.

Com essa aplicação pretende-se validar funcionalmente as seguintes características do ambiente HeMPS Station:

- Leitura de pacotes pelo módulo ComEt.
- Envio do código-objeto do módulo ComEt para a memória DDR.
- Funcionamento do processador mestre com o *microkernel* e repositório modificados.
- Leitura de dados da memória DDR pelo processador mestre e módulo DMA.
- Funcionamento do módulo DMA modificado.
- Envio de mensagens de depuração para o módulo ComEt.

5.2 Validação Funcional

Para realizar a validação funcional de todo o sistema foi necessário elaborar um ambiente completo de simulação, ilustrado pela Figura 60.

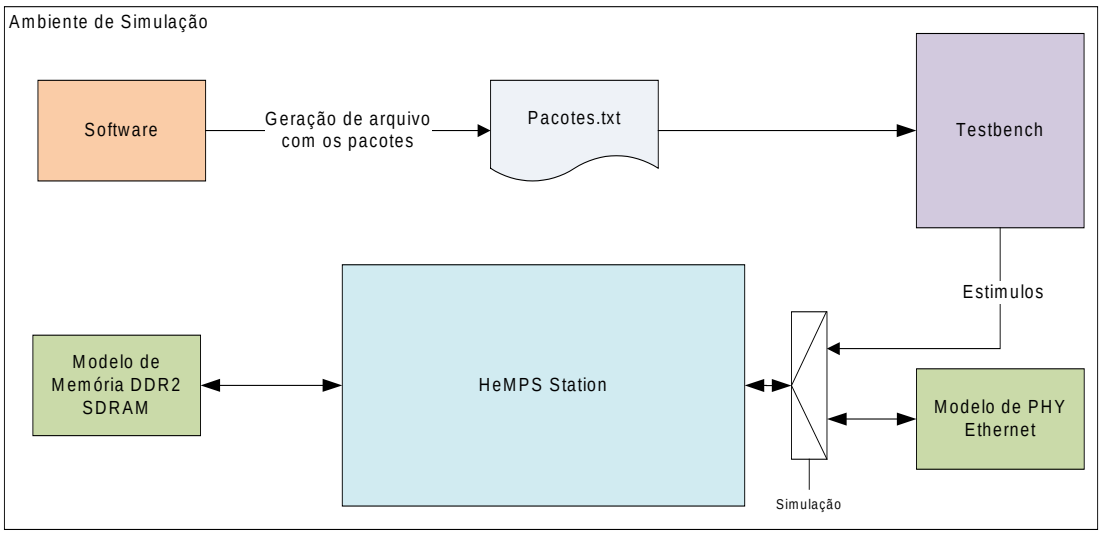

Figura 60 - Ambiente de Simulação do HeMPS Station

Primeiramente, o mesmo software que envia pacotes para rede escreve os pacotes em um arquivo, *pacotes.txt*. Esse arquivo é utilizado pela entidade de teste *Testbench* como referência para criar os estímulos de dados no sistema. Na entrada do HeMPS Station existe um multiplexador que indica se os dados vão ser recebidos pelo PHY Ethernet ou pela entidade de teste. Isso se deve pelo fato de que os controles do pacote no nível físico, não são gerados pelo software. Acessos ao modelo de memória DDR2 SDRAM, providos pelo gerador do controlador de memória da Xilinx, são executados durante a simulação.

O comportamento geral do funcionamento do ambiente HeMPS Station pode ser observado nas formas de onda da Figura 61.

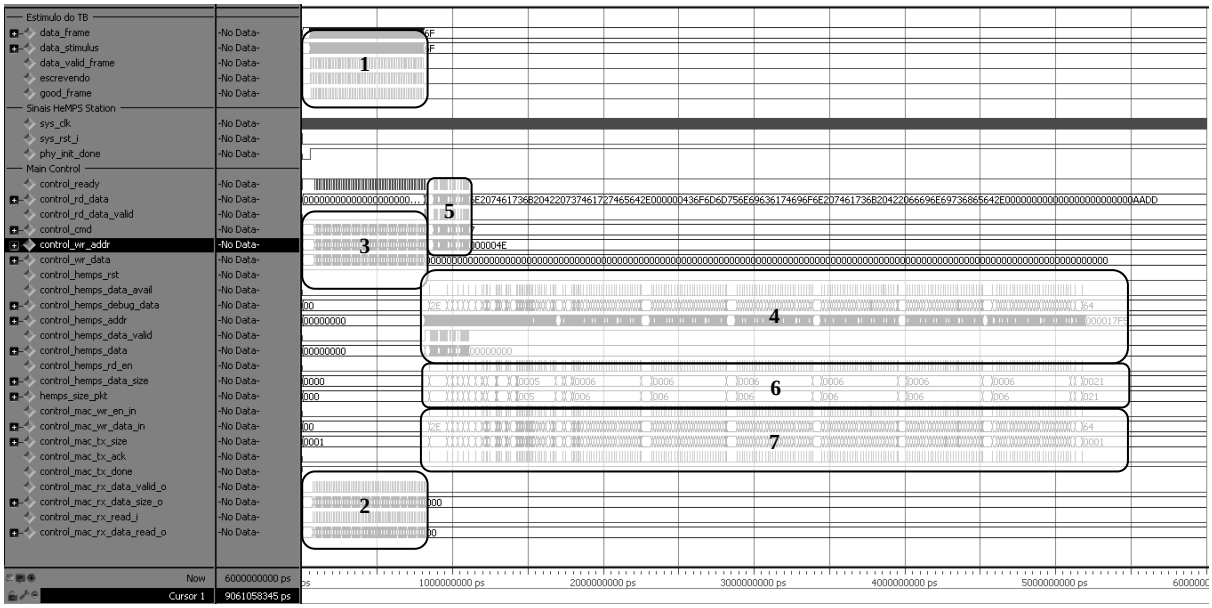

Figura 61 - Formas de onda do funcionamento do ambiente HeMPS Station

A Figura 61 exibe de maneira seqüencial o funcionamento e ilustra exatamente como ocorrem as relações entre os módulos. Na etapa 1, é realizado a inserção dos estímulos no ComEt, ou seja, pacotes UDP completos são enviados a ele. Este por sua vez, realiza seu processamento e repassa em 2 ao Main Control. Como grande parte dessa rajada de pacotes são comandos de escrita na memória (carregamento de uma aplicação), em 3 observa-se a escrita na memória DDR2. O último pacote que vem do ComEt é um comando de iniciar o MPSoC, e sua execução é representada por 4. Nos primeiros instantes dessa execução, o processador mestre e seu DMA acessam o repositório (leitura da memória DDR2) em 5. Durante toda a execução do MPSoC, dados de depuração são enviados ao Main Control, em 6, que por sua vez repassa ao ComEt, etapa 7, para empacotar e transmitir pela rede.

Nas etapas 1 e 2, busca-se validar o módulo de recepção. Para isso, foi simulado o funcionamento dos seus módulos internos (camada de enlace, camada de rede e camada de trasporte), de forma a verificar se o mesmo realiza o correto desempacotamento dos quadros recebidos pelo MAC.

A Figura 62 ilustra as interfaces do módulo de recepção, bem como, suas interconexões internas. Em destaque, ilustra-se o momento de chegada de um quadro Ethernet (1), o desencapsulamento dos pacotes pelas camadas intermediárias (2 e 3) e a entrega dos dados para a aplicação (4). Destaca-se também a abertura e o fechamento de uma sessão com

o computador remoto (5), isto é, a detecção, pela camada de transporte, de uma mensagem de conexão. Esta mensagem faz com que o sinal de *host_connected* seja colocado em nível lógico alto e o sinal de *destiny_ip* registre o endereço IP do computador na qual o dispositivo esta sendo conectado. No final, um pacote de desconexão é recebido, liberando a sessão.

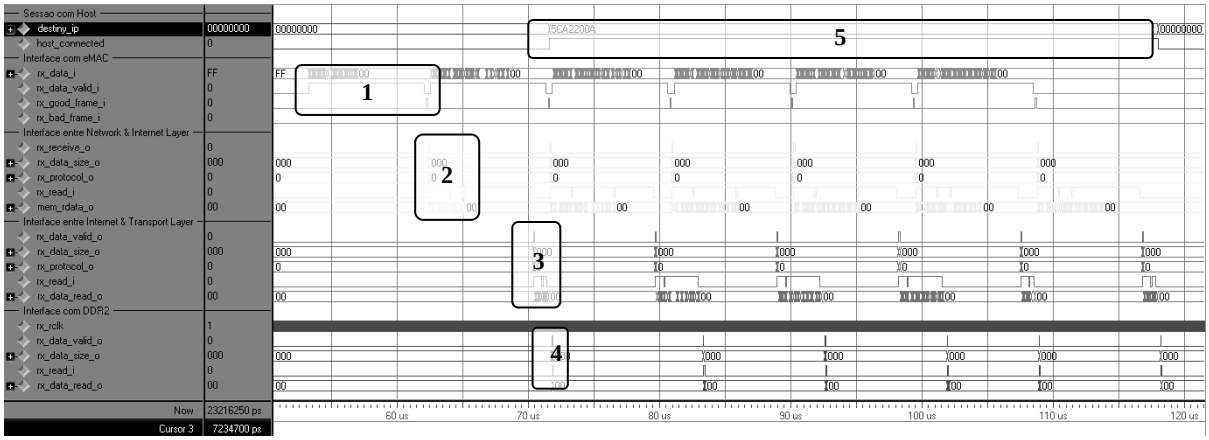

Figura 62 - Formas de onda da interface do módulo de recepção do ComEt.

Na etapa 3 (da Figura 61), têm-se a validação do protocolo de escrita na memória DDR2 SDRAM, demonstrado pela Figura 63.

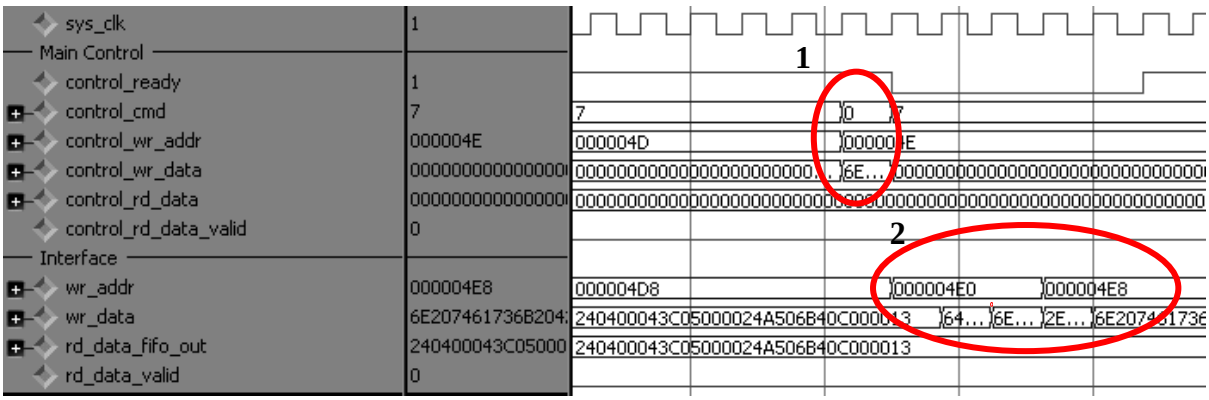

Figura 63 – Escrita de dados na memória.

Na Figura 63, 1 representa o protocolo realizado pelo Main Control, se comunicando com o módulo Interface do sistema ConMe. Nesse protocolo o dado, o endereço e o comando escrita são atribuídos e mantidos por um ciclo de relógio. Em 2, o módulo Interface interpreta a requisição e executa o protocolo com o controlador, utilizando o mesmo endereço repassado pelo Main Control, concatenado ele com os bits de controle de coluna. Pode-se observar que a palavra do barramento *control_wr_data* (512 bits) foi dividida em quatro palavras para colocar no barramento *wr_data* (128 bits), e assim realizar a rajada de quatro palavras.

Nas etapas 4 e 5 (da Figura 61), para a verificação do correto funcionamento do processador mestre, é necessário que o mesmo obtenha do repositório o número total de tarefas. Isso também valida o acesso ao repositório por parte do mestre, ou seja, leitura da memória DDR. A Figura 64 demonstra o primeiro acesso que o processador faz ao repositório, no caso a busca pelo número total de tarefas.

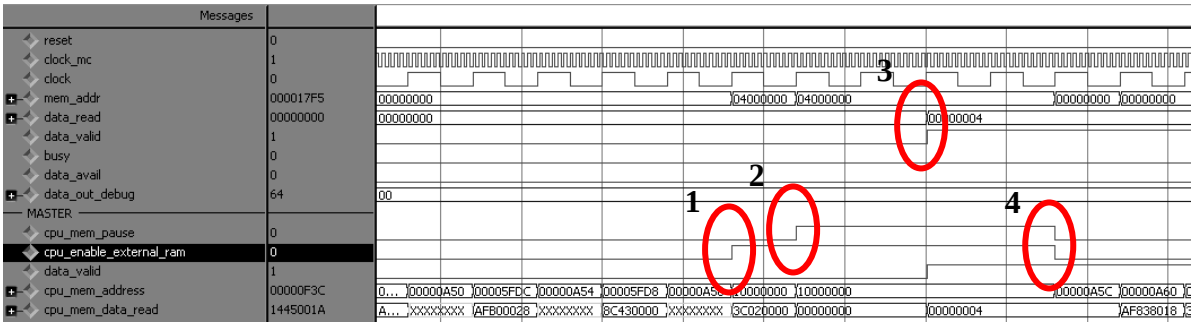

Figura 64 - Acesso ao repositório pelo processador mestre.

Na Figura 64, em (1) ocorre a identificação, por parte da CPU, que o endereço que está no barramento *cpu_mem_address* é referente à uma posição de memória do repositório. Como explicado no Capítulo 4.2, o dado pode ser lido somente após a sinalização de dado válido, indicado pelo sinal *data_valid* (3). Com isso o processador é colocado em espera, através do sinal *cpu_mem_pause*, (2). Após o dado ser lido, o processador é novamente liberado para seguir seu fluxo de execução (4). Como a aplicação utilizada para a validação utiliza quatro tarefas pode-se perceber que a leitura da memória ocorreu corretamente.

Em seguida, para a validação do funcionamento do módulo DMA modificado, é necessário verificar o momento em que o DMA é ativado pela CPU e começa a ler dados do repositório e enviá-los para a NI. Esse comportamento é ilustrado na Figura 65.

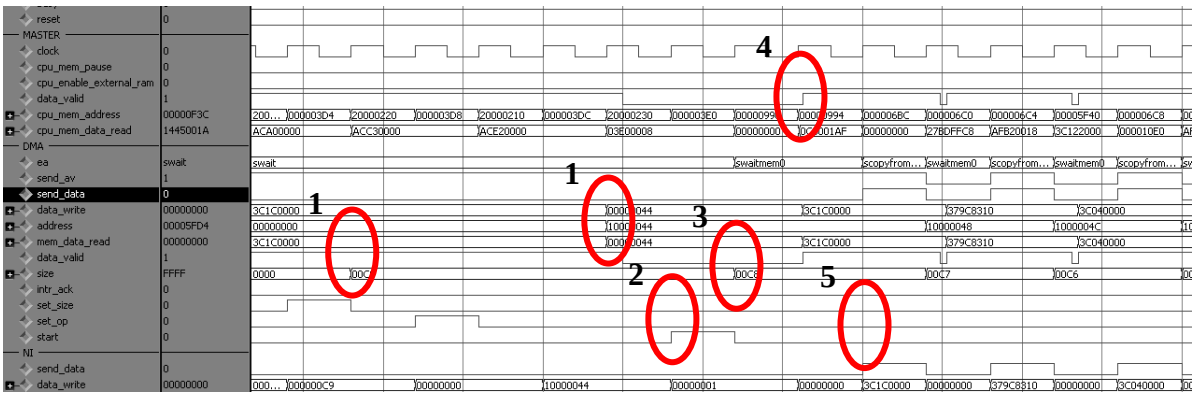

Figura 65 - Inicialização e leitura de dados do repositório do módulo DMA.

Antes da CPU iniciar o DMA, o processador configura o endereço inicial de acesso e a quantidade de dados a serem transferidos pelo DMA (1). Com isso, o DMA é disparado (2) e inicia sua execução. Como devem ser lidos dados do repositório, logo no início o contador já é decrementado (3) (funcionamento detalhado no Capítulo 4.2). Após isso o DMA fica aguardando o sinal *data_valid* para poder capturar o dado vindo da memória (4). Após capturar o dado o DMA verifica a disponibilidade de envio do mesmo para a NI e sinaliza que está enviando uma palavra através do sinal *send_data* (5). Feito isso, a NI se encarrega de enviar os dados empacotados corretamente através da NoC. Esse comportamento se repete até que o DMA tenha enviado todos os dados (Figura 66).

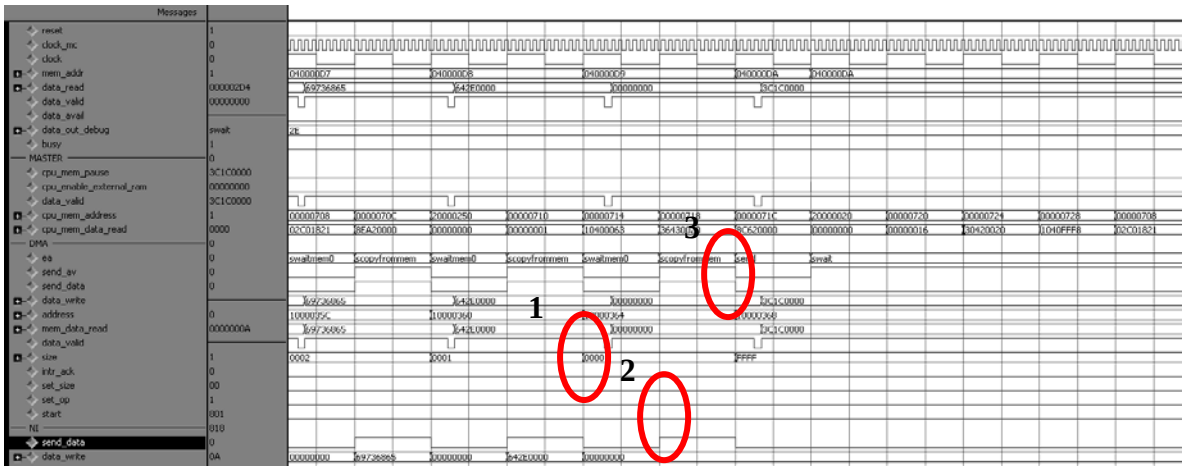

Figura 66 - Finalização do processo de envio de dados pelo DMA.

Na Figura 66 nota-se que o sinal *size* vai sendo decrementado a cada envio de palavra até chegar a zero (1). Quando isso acontece, o DMA envia seu último dado para a NI (2) e vai para o estado de *SEnd* (3), finalizando a sua execução e esperando novamente por outra configuração provinda da CPU.

Na etapa 6, têm-se a validação do protocolo de envio de mensagens de depuração para o módulo ComEt (Figura 67).

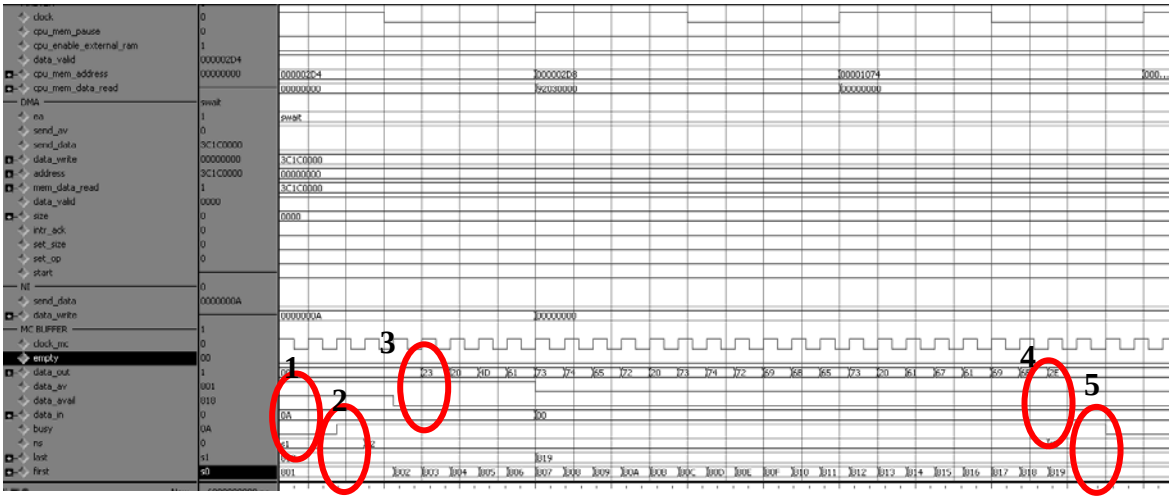

Figura 67 - Protocolo de comunicação para leitura de dados de depuração entre Main Control e HeMPS.

O processo de envio de dados inicia quando um determinado caractere chega à fila do módulo MC_Buffer. Em seguida, é sinalizado ao módulo Main Control que existem dados disponíveis para envio através do sinal *data_avail* (1). Logo após, o sinal *busy* fica sendo aguardado pelo módulo MC_Buffer. Quando isso ocorre (2), a leitura da fila é habilitada, o sinal *data_avail* é colocado em nível lógico baixo e os dados começam a ser enviados (3). Os dados são lidos da fila até que se tenha enviado todos os dados disponíveis (4) e para a finalização do protocolo o módulo Main Control coloca o sinal de *busy* em nível lógico baixo (5).

Na etapa 7 (da Figura 61), pretende-se validar o módulo de transmissão. A simulação desse módulo busca avaliar o empacotamento dos dados entregues pela aplicação e verificar se este por sua vez será transmitindo de forma válida ao MAC. A Figura 68 ilustra as interfaces do módulo de transmissão do ComEt, bem como as interfaces dos módulos internos do transmissor.

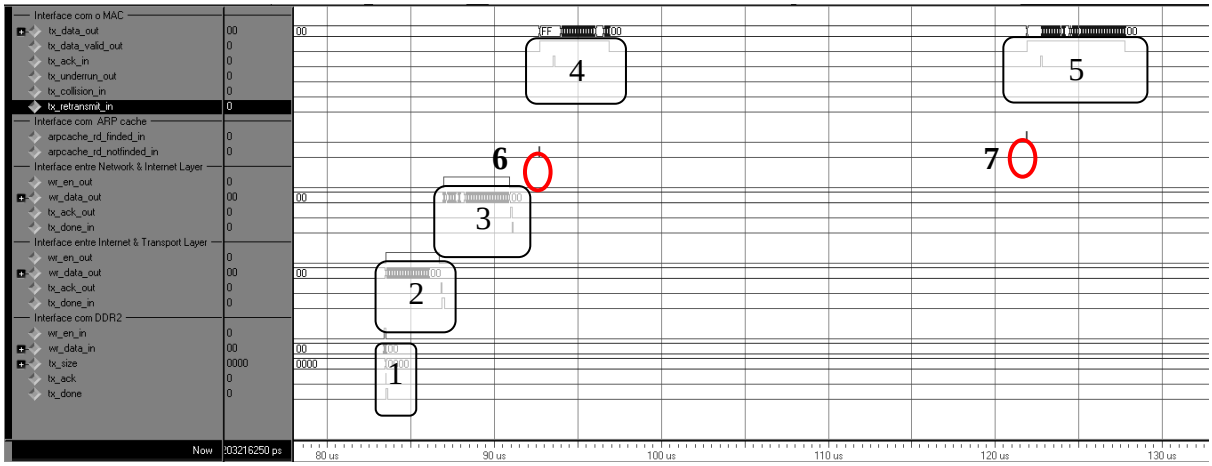

Figura 68 - Forma de ondas da interface do módulo de transmissão do ComEt.

Em destaque, ilustra-se o momento do pedido de transmissão de dados pela aplicação (1), o encapsulamento dos pacotes pelas camadas intermediárias (2 e 3) e a entrega do quadro para o MAC (5). Nota-se também que, a camada de enlace, ao consultar a ARP cache, recebe como resposta um sinal de que o endereço consultado não estava na memória (6). Dessa forma, a camada de enlace monta um pacote ARP e o envia ao MAC, como destacado em (4). Este por sua vez, fica bloqueada até que um pacote ARP de resposta seja recebido (7), e o endereço solicitado possa ser encontrado na ARP cache, a partir deste momento é que a camada de enlace transmite os dados da aplicação, já encapsulados, ao MAC (5).

5.3 Prototipação

Primeiramente foram prototipados os projetos individualmente. Após tais validações, prototipou-se a Plataforma integrada. Assim sendo, apresenta-se nesta seção os relatórios de ocupação de área e de *timing* de cada validação executada.

5.3.1 Plataforma HeMPS

A configuração utilizada para a prototipação da plataforma HeMPS é a mesma descrita no Capítulo 3.1.

Foi utilizado um MPSoC de tamanho 2x2, com 1 processador desabilitado, devido às restrições já comentadas. A Tabela 5 mostra os dados referentes à ocupação de área do dispositivo. O dispositivo alvo desta prototipação foi uma Virtex 2 Pro XC2VP30, visto as

limitações já apresentadas anteriormente. A prototipação foi validada através da comparação entre o arquivo gerado em ambiente de simulação e o arquivo gerado em tempo de execução, com informações provenientes da porta serial do computador e exibidas pela ferramenta *HyperTerminal*, disponível no sistema operacional *Windows*, da fabricante *Microsoft*.

Tabela 5 - Utilização de área do dispositivo Virtex-2 Pro (xc2vp30) para o MPSoC HeMPS. Dados retirados do Framework ISE.

Utilização Lógica	**Utilizado**	**Disponível**	**Utilização (%)**
Número Total de *Slice Registers*	5826	27392	21%
Utilizados como *Flip Flops*	5814		
Utilizados como *Latches*	12		
Distribuição de Lógica			
Ocupação de *Slices*	10379	13696	75%
Número Total de 4 input LUTs	16544	27392	60%
Utilizados como lógica	15648		
Utilizados como *route-thru*	336		
Utilizados como *Dual Port RAMs*	560		
RAMB16s	129	136	94%
BUFGMUXs	3	16	18%
DCMs	1	8	12%

5.3.2 Plataforma ComEt

Para realizar a prototipação da plataforma ComEt, desenvolveu-se um hardware capaz de sincronizar o módulo de recepção e módulo de transmissão. Dessa forma, os dados recebidos pelo módulo de recepção são diretamente transferidos ao de transmissão, fazendo-se uma espécie de *loopback* dos dados. A validação deste protótipo ocorreu através da análise dos pacotes recebidos/transmitidos da/para a rede, através do auxílio de uma versão livre do software para a análise de protocolos de rede *wireshark.* O processo de síntese foi realizado no dispositivo alvo (Virtex-5 lx330t), e a Tabela 6, apresenta o relatório de ocupação de área dos módulos do ComEt.

Tabela 6 - Utilização de área do dispositivo Virtex-5 (lx330t) para a prototipação da plataforma ComEt. Dados retirados do Framework ISE.

Utilização Lógica	**Utilizado**	**Disponível**	**Utilização (%)**
Número Total de *Slice Registers*	1512	207360	1%
Utilizados como *Flip Flops*	1495		
Utilizados como *Memory*	764	54720	1%
Número de route-thrus	413	414720	1%
O6 output only	412		
O5 and O6	1		

Distribuição de Lógica			
Ocupação de *Slices*	1391	51840	2%
Utilização E/S			
Bonded IOBs			
Números de *bonded*	22	960	2%
BUFG/BUFGCTRLs	5	32	15%
BUFGs	5		
TEMACs	1	2	50%

5.3.3 Plataforma ConMe

Para realizar a prototipação do sistema ConMe, foi modificado o projeto de validação do controlador, gerado pela ferramenta Core Generator. Este projeto faz requisições de escrita e leitura de dados no módulo Interface do ConMe. O correto funcionamento foi observado na ferramenta de análise Chipscope Analyser. O processo de síntese foi realizado no dispositivo alvo (Virtex-5 lx330t), e a Tabela 7 apresenta o relatório de ocupação de área dos módulos do ConMe.

Tabela 7 - Utilização de área do dispositivo Virtex-5 (lx330t) para a prototipação da plataforma ConMe. Dados retirados do Framework ISE.

Utilização Lógica	Utilizado	Disponível	Utilização (%)
Número Total de *Slice Registers*	2918	207360	1%
Utilizados como *Flip Flops*	2918		
Utilizados como *Memory*	367	54720	1%
Número de route-thrus	114	414720	1%
O6 output only	103		
O5 and O6	2		
Distribuição de Lógica			
Ocupação de *Slices*	1692	51840	3%
Utilização E/S			
Números de *bonded*	117	960	12%
BlockRAM/FIFO	75	324	23%
BUFG/BUFGCTRLs	8	32	25%
BUFGs	8		
DCM_ADVs	3	12	25%

5.3.4 HeMPS Station

Para a realização da síntese da plataforma completa, sintetizou-se um MPSoC de tamanho 3x2, além do projeto do ComEt e do ConMe. O processo de síntese foi realizado no dispositivo alvo (Virtex-5 lx330t), e a Tabela 8 mostra os dados referentes à ocupação de área do dispositivo.

Tabela 8 - Utilização de área do dispositivo Virtex-5 (lx330t) para a prototipação da plataforma HeMPS Station. Dados retirados do Framework ISE.

Utilização Lógica	**Utilizado**	**Disponível**	**Utilização (%)**
Número Total de *Slice Registers*	17418	207360	8%
Utilizados como *Flip Flops*	17377		
Utilizados como *Memory*	1342	54,720	2%
Número de route-thrus	1110	414720	1%
O6 output only	1096		
O5 and O6	2		
Distribuição de Lógica			
Ocupação de *Slices*	13004	51840	25%
Utilização E/S			
Bonded IOBs			
Números de *bonded*	134	960	13%
BUFG/BUFGCTRLs	12	32	37%
BUFGs	12		
DCM_ADVs	3	12	25%
TEMACs	1	2	50%

CONSIDERAÇÕES FINAIS

Este trabalho apresentou o projeto de uma estação de trabalho para projetos de MPSoCs – HeMPS Station. Este ambiente permite gerar um projeto parametrizável, simulá-lo, prototipá-lo em um dispositivo FPGA e realizar execuções remotas com depuração em tempo de execução. Além disso, ele se mostra uma ferramenta poderosa no campo de pesquisa em projetos de sistemas multiprocessados e aplicações embarcadas.

O desenvolvimento da arquitetura HeMPS Station compreende desde a implementação de softwares, módulos de hardwares e até adaptações do MPSoC alvo. Portanto, dentre as contribuições mais relevantes podem-se citar as adaptações no MPSoC HeMPS, com o intuito de torná-lo mais flexível à prototipação; o desenvolvimento de um IP que realiza uma comunicação pela rede através de um MAC Ethernet, o desenvolvimento de um software que realiza comunicação com este IP e o desenvolvimento de um IP para realizar de forma prática o acesso às memórias DDRs.

As etapas de desenvolvimento ocorreram de forma separada, bem como suas validações. Ao integrar os três módulos referenciais (Plataforma HeMPS, Plataforma ComEt e Sistema ConMe) foi desenvolvido um módulo controlador, para realizar a interação e o correto funcionamento de todo o sistema.

Por fim, foi realizada a validação funcional com o sistema integrado. Conseqüentemente, o sistema foi prototipado, mas apesar de estar rodando em FPGA, o sistema ainda se encontra em fase de testes e validação.

Um fator muito importante a ser ressaltado são as dificuldades que foram encontradas durante o desenvolvimento do projeto, em sua maioria ligadas à etapa de prototipação. Os maiores problemas foram ocasionados pela falta de conhecimento e domínio da placa de prototipação alvo. O primeiro problema ocorreu, pois o circuito integrado responsável pela comunicação com a saída Ethernet não estava estável. Após muito tempo de projeto congelado, descobriu-se que esse CI é reiniciado apenas quando a placa é ligada, e para que tudo ocorra corretamente todos os cabos de rede devem já estar devidamente conectados e configurados. O segundo problema foi o desconhecimento do controle das freqüências de relógios, cujos valores não podiam ser confirmados, pois a placa não possui nenhum pino de teste ou pino simples de saída. Após descobrir o relógio principal utilizou-se módulos DCMs para criar os relógios necessários.

Outra grande dificuldade foi adaptar o controlador da memória DDR2 SDRAM gerado pelo Core Generator para os projetos desenvolvidos. Isso se deve, pois o UCF (do inglês: *User Constrains File*), arquivo que contém requisitos de localização dos pinos, localização dos módulos e rotas de roteamento, é gerado de maneira errônea. O fator positivo é que, após muito estudo bibliográfico, descobriu-se que a última versão desse gerador de controladores de memória possui um recurso de atualização do arquivo UCF, onde o projetista realiza a alocação dos pinos e o controlador atualiza as localizações do módulo e rotas no FPGA.

Encoutrou-se dificuldade também em adaptar o processador mestre e módulo DMA do MPSoC HeMPS para leitura da memória DDR. Isto porque se teve de encontrar uma forma consistente de parar o processador enquanto o dado lido da memória não estivesse válido.

Como objetivos futuros, almeja-se aperfeiçoar o ambiente para que este fique mais automatizado, criar um sistema de monitoramento de hardware inserido no projeto com análises em tempo de execução e realizar experimentos com diferentes tipos de aplicações embarcadas, comparando o desempenho do MPSoC HeMPS.

REFERÊNCIAS BIBLIOGRÁFICAS

[BAD01] Bader, D. A.; Pennington, R. "*Cluster Computing: Applications*". The International Journal of High Performance Computing, vol. 15-2, Maio 2001, pp. 181-185.

[BEN02] Benini, L.; Micheli, G. "*Networks on Chips: A New SoC Paradigm*". IEEE Computer, vol. 35-1, Janeiro 2002, pp. 70-78.

[CAL01] Calazans, N. L. V.; Moraes, F. G.; Torok, D. L.; Andreoli, A. V. "*Projeto para Prototipação de um IP Soft Core MAC Ethernet"*. Revista de Informática Teórica e Aplicada, vol. 8-1, 2001, pp. 23-41.

[CAR05] Carara, E.; Moraes, F. "*MPSoC-H – Implementação e Avaliação de Sistema MPSoC Utilizando a Rede Hermes*". Relatório Técnico, FACIN, PUCRS, Brasil, 2005, 43p.

[CAR09] Carara, E.; Oliveira, R. P.; Calazans, N. L. V.; Moraes, F. G. "*HeMPS – A framework for Noc-Based MPSoC Generation*". Em: IEEE International Symposium on Circuits and Systems - ISCAS´09, 2009, pp. 1345-1348.

[CGG09] XILINX INC. "*CORE Generator Guide*" Capturado em: www.xilinx.com/itp/xilinx6/books/docs/cgn/cgn.pdf, Junho 2009.

[GUP96] Gupta, R. K.; De Micheli, G. "*A Co-Synthesis Approach to Embedded System Design Automation"*. Design Automation for Embedded Systems, vol 1-2, Janeiro 1996, pp. 69-120.

[HAL00] Hall, E. A. "*Internet Core Protocols: The Definitive Guide*". O'Really, 2000, 472p.

[HEN03] Henkel, J. "*Closing the SoC design gap*". IEEE Computer, vol 36-9, Setembro 2003, pp. 119-121.

[HWA93] Hwan, K. "*Advanced Computer Architecture: Parallelism, Scalability, Programmability*". McGraw-Hill, Inc. Computer Science, 1993, 771p.

[ITR07] International Technology Roadmap for Semiconductors. "*System Driver"*. Capturado em: http://www.itrs.net/Links/2007ITRS/Home2007.htm, 2007.

[JER04] Jerraya, A.; Wolf, W. "*Multiprocessors Systems-on-Chips*". Morgan Kaufman Publishers, 2004, 608p.

[JER05] Jerraya, A.; Tenhunen, H.; Wolf, W. "*Guest Editors' Introduction: Multiprocessor Systems-on-Chips*". IEEE Computer, vol 38-7, Julho 2005. pp. 36-40.

[KAR08] Palanisamy, K.; Chiu, R. "*High-Performance DDR2 SDRAM Interface in Virtex-5 Devices - Xilinx Application Note*". Capturado em: http://www.xilinx.com/support/documentation/application_notes/xapp858.pdf, Maio 2008.

[KIS06] Kistler, M.; Perrone, M.; Petrini, F. "*Cell Multiprocessor Communication Network: Built for Speed*". IEEE Micro, vol 26-3, Maio - Junho 2006, pp. 10-23.

[KUM94] Kumar, V.; Grama, A.; Gupta, A.; Karpis, G. "*Introduction to Parallel Computing: Design and Analysis of Algorithms"*. Benjamin/Cummings, 1994, 580p.

[LEN90] Lenoski, D.; Laudon, J.; Gharachorloo, K.; Gupta, A.; Hennessy, J. "*The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor"*. ACM SIGARCH Computer Architecture News, vol 18-3a, Junho 1990, pp. 148-

159.

[LIN05] Lin, L.; Wang, C.; Huang, P.; Chou, C.; Jou, J. "*Communication-driven task binding for multiprocessor with latency insensitive network-on-chip*". Em: ASP-DAC, 2005, pp. 39-44.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Mello, A.; Moller, L.; Ost, L. *"HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip"*. Integration the VLSI Journal, vol. 38-1, 2004, pp. 69-93.

[PAT96] Patterson, D.; Hennessy, J. L. *"Computer Architecture: A Quantitative Approach"*. Morgan Kaufmann, 1996, 760p.

[PLA09] Rhoads, S. *"PLASMA Processor"*. Capturado em: http://www.opencores.org/?do=project&who=mips, 2009.

[REC04] Reis, C. *"Sistemas Operacionais para Sistemas Embarcados"*. ED-UFBA, Brasil, 2004.

[REI09] Reinbrecht, C. R. W.; Scartezzini, G.; Da Rosa, T. R.; Moraes, F. G. *"HeMPS Station: an environment to evaluate distributed applications in NoC-based MPSoCs"*. Em: SIM2009, 2009, pp 161-164.

[SAI07] Saint-Jean, N.; Sassatelli, G.; Benoit, P.; Torres, L.; Robert, M. *"HS-Scale: a Hardware-Software Scalable MP-SOC Architecture for embedded Systems"*. Em: ISVLSI, 2007, pp 21-28.

[STE01] Sterling, T. *"Beowulf Cluster Computing with Linux"*. Em: MIT Press, Outubro 2001. pp. 14-21.

[TAN06] Tanurhan, Y. *"Processors and FPGAs Quo Vadis?"*. Em: Computer, v. 39-11, 2006, pp. 108-110.

[TIL07] Tilera Corporation. *"TILE64™ Processor"*. Capturado em: http://www.tilera.com/pdf/ProBrief_Tile64_Web.pdf, Agosto 2007.

[VAN07] Vangal, S.; Howard, J.; Ruhl, G.; Dighe, S.; Wilson, H.; Tschanz, J.; Finan, D.; Iyer, P.; Singh, A.; Jacob, T.; Jain, S.; Venkataraman, S.; Hoskote, Y.; Borkar, N. *"An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS"*. IEEE International Solid-State Circuits Conference (ISSCC), Fevereiro 2007. pp. 5-7.

[VIR09] XILINX INC. *"Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC - User Guide*", Capturado em: *Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC - User Guide*, Abril 2009.

[WOL03] Wolf, S. *"Testing Network Drivers with the NDIS Test Tool"*. Capturado em: http://www.wd-3.com/archive/NDISTest.htm, Julho 2003

[WOL04] Wolf, W. *"The Future of Multiprocessors Systems-on-Chip"*. Em: DAC, 2004, pp.681-685.

[WOL05] Wolf, W. *"Embedded Computer Architectures in the MPSoC Age"*. Em: 32nd International Symposium on Computer Architecture, 2005.

[WOS07] Woszezenki, R. C. *"Alocação De Tarefas E Comunicação Entre Tarefas Em MPSoCS"*, Dissertação de Mestrado, FACIN, PUCRS, Brasil, Março 2007, 121p.

[XIL09] XILINX INC. *"Virtex 5 Family Overview"* Capturado em: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, Março 2009.