Hierarquia de Memória em MPSoC Baseado em NoC

Implementação e Avaliação

Tales Marchesan Chaves
DECC – Dept. das Engenharias e Ciência da Computação
URI – Universidade Regional Integrada
Santo Ângelo, Brasil
tales.chaves@gmail.com

Fernando Gehm Moraes, Carlos Alberto Petry
FACIN - Faculdade de Informática
PUCRS – Pontifícia Universidade Católica
Porto Alegre, Brasil
fernando.moraes@pucrs.br, carlos.petry@pucrs.br

***Abstract*—The increase in the number of transistors on a single chip has brought new design challenges, among them, how to design new circuits using this available number of transistors. The reuse of Intellectual Property cores enables the design of such complex systems. Multiprocessor Systems on Chip (MPSoCs) are systems where such modules are processors. One key design structure to define the system performance is the memory hierarchy. Applications need to access large objects, as images, audio, and video, and the local memory of processors might not provide enough space to store such objects. This paper proposes the implementation of a memory hierarchy in a NoC-based MPSoC. The hierarchy is composed by a global shared-memory, which is connected to the NoC, and two memories per processor: one cache memory for caching global data and a private memory for private data and instructions. A hybrid implementation of a directory-based cache coherence policy is proposed and discussed.**

Keywords: MPSoC; NoC; memory hierarchy; cache coherence protocol.

I. INTRODUÇÃO

Desde o início da década de 70 a indústria tem explorado soluções para o aumento do desempenho em sistemas computacionais. Essa exploração se dava principalmente no aumento do paralelismo em nível de instrução e na freqüência de operação. Paralelismo no nível de instrução atingiu os limites práticos de ganho de desempenho com as máquinas super-escalares, dada a natureza seqüencial das aplicações. Aumento da freqüência de operação também atingiu os limites práticos de ganho, devido à dissipação de potência.

Dado o aumento exponencial da capacidade de integração de transistores em um mesmo circuito integrado, a forma encontrada pela indústria para o aumento de desempenho é a exploração de paralelismo com a inserção de um maior número de núcleos de processamento dentro de um chip [1] [2]. Estes sistemas são denominados *Multiprocessor Systems-on-Chips* (MPSoCS), constituídos por processadores, blocos de hardware com função específica, elementos de memória, e um meio de interconexão. Genericamente, componentes do MPSoC são denominados, exceto o meio de interconexão, por elementos de processamento (EPs).

A interconexão entre os EPs é geralmente realizada com barramentos, barramentos hierárquicos, conexões ponto-a-ponto ou redes intra-chip. A utilização de barramentos é ampla, porém esse tipo de interconexão não é escalável [3], suportando apenas algumas dezenas de EPs, além de possuírem ausência de paralelismo, uma vez que em um barramento compartilhado, apenas uma conexão pode ser estabelecida por vez. Conexões ponto-a-ponto são também pouco escaláveis. Redes intra-chip [5][8] têm sido adotadas como uma solução de meio de interconexão que endereça esses problemas de forma distribuída.

Quando o EP é um processador, este é geralmente associado a um módulo de memória, chamado de memória *cache*. Essa memória é responsável pelo armazenamento de dados que são freqüentemente utilizados pelo processador, de forma a aumentar o desempenho.

Em sistemas MPSoCs onde existe uma memória compartilhada, geralmente existe uma estratégia para diminuir a largura de banda requerida para a memória, caso contrário a latência de acesso à memória pode se tornar um gargalo no sistema. De acordo com [7], a organização de memória é um dos componentes mais críticos que podem determinar o sucesso de arquiteturas baseadas em MPSoCs. Esta afirmação se justifica pelo fato de que muitas aplicações consomem uma quantidade significativa de ciclos, durante sua execução, na hierarquia de memória. O uso de memórias com bancos intercalados, memórias *page-mode* e memórias *cache*, são exemplos de estratégias para tratar este problema. O uso de memórias *cache* para dados globais pode tornar o sistema incoerente, pois um mesmo dado pode estar modificado em duas *caches* distintas, o que requer o uso de uma política de coerência.

Em MPSoCs baseados em NoCs, a implementação de políticas de *snooping* é custosa pois exige que, a cada acesso de um EP à memória compartilhada, uma mensagem de notificação seja propagada por todo o sistema (mensagem de *broadcast*). Nestes sistemas podem ser empregadas políticas de coerência baseadas em diretório, onde se estima que o custo da implementação da estrutura de diretório possa atingir até 20% do total da memória [14].

Este trabalho tem por objetivo desenvolver um sistema de memória, composto por uma memória compartilhada e memórias *cache* associadas à EPs, no MPSoC HeMPS [15]. Além disso, para manter a coerência de *cache* é desenvolvida uma implementação modular, de software e de hardware, da política de diretório. O diretório, implementado em hardware, é responsável por armazenar o estado de cada bloco da memória, e o *microkernel* de cada processador tem a responsabilidade de requisitar exclusividade, como também

executar a operação de substituição de blocos no estado *dirty*. Essa abordagem torna o módulo de memória modular, como também simplifica o projeto de hardware da memória compartilhada e simplifica o controlador de *cache*.

O presente artigo está organizado como segue. A Seção 2 apresenta trabalhos relacionados à implementação de políticas de coerência em arquiteturas de MPSoC. A Seção 3 descreve a plataforma MPSoC utilizada por este trabalho. A Seção 4 apresenta a hierarquia de memória proposta. A Seção 5 apresenta a implementação da política por diretório. A Seção 6 apresenta os resultados obtidos, e a Seção 7 apresenta as conclusões e as direções para trabalhos futuros.

II. TRABALHOS RELACIONADOS

Protocolos de coerência podem ser implementados apenas em hardware, em software ou utilizando uma abordagem híbrida. Em Girão et. al [9] é apresentada uma plataforma MPSoC onde a interconexão dos elementos é feita com a utilização de uma NoC. A coerência de *cache* é implementada completamente em hardware com a utilização de uma memória de diretório. Cada módulo de memória na plataforma tem associado um módulo diretório que gerencia a sua comunicação com a NoC e a coerência de *cache*.

Em Kim et. al [10] é apresentada uma solução baseada em diretório para um MPSoC baseado em NoC, utilizando uma memória compartilhada distribuída. Nesta solução o diretório é desenvolvido dentro do roteador, o que, segundo os autores, melhora o desempenho.

Em Petrot et. al [11] é proposta a existência de espaços de endereçamento diferentes para dados compartilhados e privados. Dados privados são transferidos para a memória *cache*, pois estes são locais a um processador específico. Dados compartilhados são acessados diretamente da memória compartilhada, eliminando o problema de coerência de *cache*, entretanto isto pode degradar o desempenho geral do sistema.

Em Jerger et. al [12] é proposta uma solução para a coerência de cache denominada *Virtual Tree Coherence* (VTC). Esta é baseada em uma interconexão ordenada virtualmente que mantém um histórico dos nodos que compartilham uma região de memória. Para cada região é criada um árvore virtual de nodos que compartilham o acesso a essa região. Cada vez que essa região é acessada, é enviada uma mensagem de requisição para o nodo raiz da árvore que requisita o dado para o nodo que possui o dado através de uma mensagem *multicast*. Mensagens de requisição em *multicast* são realizadas na forma de uma árvore, visando diminuir a latência na rede.

III. PLATAFORMA HEMPS MPSOC

O MPSoC HeMPS, ilustrado na Figura 1, em uma topologia malha 2x3, é um ambiente multiprocessado homogêneo composto por EPs e uma infra-estrutura de interconexão de elementos.

EPs da plataforma são constituídos por processadores RISC denominados PLASMA [13], os quais são baseados na arquitetura do ISA MIPS-I. Além disso, cada EP contém uma memória RAM privada, um módulo DMA e uma interface para memória externa.

A infra-estrutura de interconexão é constituída por uma rede intra-chip, denominada Hermes [6]. Cada EP é conectado à rede Hermes através de uma interface de rede (*network interface* ou NI), a qual é responsável por mapear sinais da rede intra-chip para as entradas e saídas deste.

Os EPs da plataforma HeMPS dividem-se em duas classes: mestre e escravo. O mestre é responsável pela execução de uma aplicação de alocação de tarefas, a qual pode ser estática ou dinâmica, e a gerência dos recursos do sistema. Tarefas são lidas do repositório de tarefas (do inglês, *Task Repository*), o qual está conectado na interface para memória externa do processador PLASMA, e distribuídas para os EPs escravos de acordo com a configuração da plataforma. Em qualquer configuração da plataforma existe apenas um processador mestre.

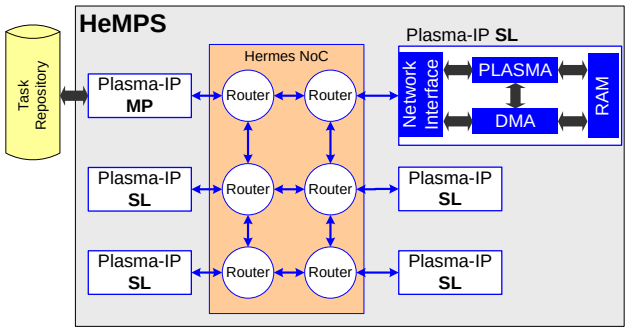

Figure 1. Plataforma HeMPS MPSoC utilizando uma Malha 2x3.

A memória RAM local de cada EP é dividida em páginas. A primeira página armazena o *microkernel* (pequeno sistema operacional, contendo apenas os serviços de alocação de tarefas, escalonamento, comunicação entre tarefas, tratamento de interrupções e chamadas de sistema), e as páginas subseqüentes armazenam as tarefas de usuário.

Cada EP mantém uma tabela de tarefas com a localização das tarefas locais e remotas que é utilizado nas rotinas de comunicação. Os processadores escravos (Plasma-IP SL) são responsáveis pela execução das tarefas de cada aplicação.

IV. HIERÁRQUIA DE MEMÓRIA

Aplicativos que manipulam objetos, como por exemplo, vídeos e imagens, podem conter um volume de dados maior do que a capacidade de armazenamento da memória privada local de um EP. Este fato justifica o desenvolvimento de uma hierarquia de memória onde exista uma memória maior e mais distante do processador, de custo menor, que armazena objetos maiores; e memórias *cache*, com latência menor, porém com menor capacidade que armazenam partes do objeto mais comumente utilizadas pela aplicação para prover dados para o processador em uma taxa mais próxima de sua freqüência de operação.

A hierarquia de memória proposta por este trabalho, ilustrada na Figura 2, é baseada na implementação de uma memória compartilhada entre os EPs, a qual tem por objetivo armazenar dados globais. Além disso, cada processador irá conter duas memórias: uma para dados privados e outra para dados globais. A memória para dados privados contém o código dos aplicativos e dados privados do EP. A memória para dados globais é uma memória *cache* da memória compartilhada.

Como a memória *cache* tem menor capacidade do que a

memória compartilhada existe uma disputa natural de blocos da *cache* por blocos da memória compartilhada. Estes são mapeados na memória *cache* de acordo com o tipo de mapeamento de memória utilizado, o qual pode ser: direto, associativo por conjuntos ou totalmente associativo. No mapeamento direto um bloco da memória compartilhada tem como destino um bloco fixo da memória *cache*, o que torna este mapeamento interessante para sistemas embarcados devido a sua simplicidade e bom desempenho. No mapeamento associativo por conjuntos um bloco da memória compartilhada pode ser destinado a qualquer bloco pertencente a um conjunto da *cache*, e no último o bloco da memória compartilhada pode ser colocado em qualquer bloco da *cache*. Estes dois últimos mapeamentos requerem a utilização de estruturas de comparação para determinar, a cada acesso de memória, a posição do bloco dentro de um conjunto, o que pode aumentar o custo e o consumo de energia no sistema.

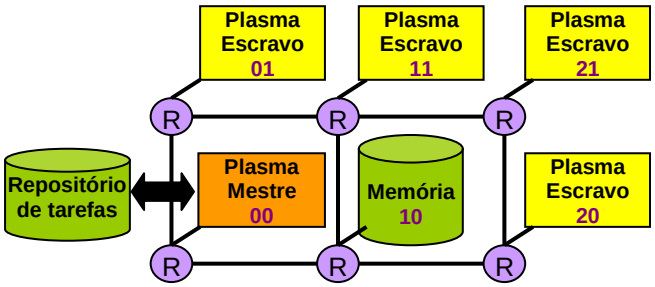

Figure 2. Configuração da plataforma HeMPS com módulo de memória compartilhada.

Escritas à memória *cache* devem atualizar a memória compartilhada em algum instante, de modo a manter a coerência do sistema. A atualização pode acontecer no mesmo momento da operação de escrita, caracterizando a política de *write-through*, ou no momento em que o bloco da memória *cache* é substituído por outro bloco da memória compartilhada, caracterizando a política de *write-back*.

Manter a coerência de *cache* no sistema implica ainda em permitir que um bloco da memória compartilhada seja modificado por apenas um EP a cada vez. De acordo com a política de coerência de *cache* por diretório, para alterar um bloco o EP com este bloco deve possuir exclusividade sobre o mesmo. Para que um bloco possa ser lido por um EP da memória compartilhada este deve estar no estado compartilhado. Vários EPs podem compartilhar um determinado bloco.

O estado do bloco na memória de diretório pode ser utilizado também para indicar se o bloco encontra-se atualizado na memória compartilhada. No estado exclusivo, um bloco está modificado na *cache* de um EP, sendo necessário atualizá-lo caso chegue uma operação de leitura na memória por outro processador. No estado compartilhado, o bloco encontra-se atualizado na memória principal.

A. Memória Compartilhada

A memória compartilhada é composta por três entidades principais: (i) memória; (ii) interface de rede - NI; (iii) memória de diretório. As operações na memória são realizadas sobre todo o bloco. A NI é responsável por receber e tratar requisições de leitura, escrita, e pedido de exclusividade, os quais ocorrem em formato de pacotes. A memória de diretório mantém o estado dos blocos, com a finalidade de manter a coerência entre as memórias *cache*.

O formato do pacote de requisição para a memória compartilhada é ilustrado na Figura 3. Cada campo do pacote é composto por dois *flits* (32 bits), exceto o campo **Payload** que contém o tamanho necessário para conter um bloco da memória compartilhada. O campo **Header** é divido em dois campos: o endereço destino do pacote se encontra no primeiro *flit* e o tamanho do pacote, em *flits*, no segundo. O campo **Service** indica o serviço a ser executado. O campo **SourceNetAddr** indica o endereço de rede do EP originário do pacote. O campo **TargetAddress** indica o endereço inicial do bloco onde será executada uma determinada operação. O campo **SourceTask** contém o identificador da tarefa origem. O campo **Length** indica o tamanho do **Payload**.

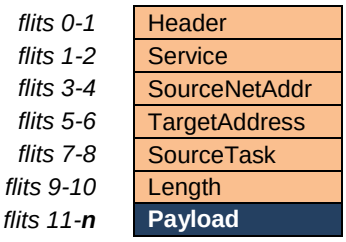

Figure 3. Formato do pacote de requisição para memória compartilhada.

A *NI* contém três máquinas de estados: RSM, SSM e a CSM. A RSM (*Receiving State Machine*) é responsável pelo recebimento de pacotes da rede, os quais podem ser pacotes que contém um bloco a ser escrito na memória compartilhada ou apenas mensagens de controle. A SSM (*Sending State Machine*) é responsável pelo envio de um bloco ao EP requisitante pela leitura. A CSM (*Coherence-control State Machine*) é responsável pelo envio de mensagens para controle de coerência: pedidos de *write-back* e mensagens de invalidação de bloco.

A memória de diretório contém para cada bloco de dados uma entrada. Os bits mais significativos do endereço da memória compartilhada correspondem ao endereço de entrada de um bloco no diretório (mapeamento direto). O estado de cada bloco é representado por três bits: *valid* indica se aquela entrada é válida; *shared* indica se o bloco está compartilhado; *exclusive* indica se o bloco é exclusivo. Para cada EP do sistema, há um bit que indica se o bloco está compartilhado, caso o bloco esteja no estado compartilhado, ou exclusivo, caso o bloco esteja no estado exclusivo.

A figura 4 ilustra um exemplo da memória de diretório onde o *bloco 0* está exclusivo no EP 1, e o *bloco 2* está compartilhado entre os EPs 0 e 3. A cada EP é associado um endereço de rede, necessário para envio de mensagens.

TABELA 1 – ESTRUTURA DA MEMÓRIA DE DIRETÓRIO

	Valid	*Shared*	*Exclusive*	*EP0*	*EP1*	*EP2*	*EP3*
0	1	0	1	0	1	0	0
1	0	0	0	0	0	0	0
2	1	1	0	1	0	0	1
N	0	0	0	0	0	0	0

B. *Memória cache*

O processador PLASMA contém uma interface para memória externa, a qual é utilizada pelo processador mestre para a leitura do código objeto de tarefas do repositório. Nos processadores escravos esta interface é utilizada para a comunicação com a memória *cache*.

A interface para memória externa é composta pelos seguintes sinais: *address* informa o endereço a ser acessado na memória *cache*; *data_write* utilizado para envio de dados a serem escritos na *cache*; *data_read* utilizado pelo processador para acessar dados lidos da *cache*; *write_byte_enable* informa o tipo da instrução (leitura/escrita); *mem_pause_in* utilizado pela *cache* para pausar o processador enquanto esta está fazendo a leitura ou escrita de um dado.

O espaço de endereçamento das memórias do processador é distinto. Acessos a memória com endereços na faixa **0x00000000** à **0x0000FFFF** são destinados a memória privada, enquanto que acessos a endereços na faixa **0x10000000** à **0x100FFFFF** são destinados à memória *cache*.

O mapeamento da memória compartilhada para a memória *cache* utilizado é o mapeamento direto. A escolha deste mapeamento é justificada pela simplicidade na implementação, o que facilita o projeto de hardware da hierarquia de memória. A política de atualização da memória compartilhada é *write-back*.

Acessos a memória de dados globais por tarefas são feitos através de chamadas a funções do *microkernel ReadShared* e *WriteShared*. A configuração do bloco de memória a ser acessado é feito através da estrutura *MessageMP*, utilizada como parâmetro das funções acima. Esta estrutura é composta por três campos: bloco (*block*), tamanho (*length*) e deslocamento dentro do bloco (*offset*). Requisições para a memória compartilhada, através das funções, são interceptadas pelo *microkernel*. Este é responsável por fazer o acesso à memória *cache* e detectar a ocorrência de *hit* ou *miss*. Caso ocorra um *hit* a tarefa continua sua execução. Em caso de *miss*, o *microkernel* faz uma requisição para a memória compartilhada para leitura ou escrita do bloco requisitado.

V. PROTOCOLO DE COERÊNCIA

A política de coerência de *caches* baseada em diretórios proposta para a plataforma HeMPS segue uma abordagem híbrida. Parte da implementação desta política é dada em hardware, na memória compartilhada, e outra parte em software, no *microkernel*. A memória compartilhada contém um controlador responsável pelo controle do acesso aos blocos da memória baseado no conteúdo da memória de diretório que mantém o estado atual de cada bloco. O *microkernel* é responsável pela execução das seguintes tarefas: detecção de *hit*, detecção de *miss*, execução da atualização da memória compartilhada quando solicitado (*write-back*) e pedido de exclusividade de um bloco.

Nas subseções seguintes são detalhadas as diferentes situações que podem ocorrer no acesso à memória e as respectivas ações determinadas pela política por diretório a fim de manter a coerência do sistema.

A. *SERVIÇOS DA MEMÓRIA COMPARTILHADA*

Cada pacote enviado a memória compartilhada possui um campo serviço o qual define a operação a ser executada na mesma. Os serviços definidos indicam operações básicas da memória (escrita e leitura de blocos), como também operações para auxiliar no controle da coerência de *caches*, como solicitação de exclusividade de bloco, invalidação de bloco e *write-back*. Os serviços suportados pela plataforma são descritos na Tabela 2.

TABELA 2. SERVIÇOS SUPORTADOS PELA MEMÓRIA COMPARTILHADA.

Nome	Código
WRITE_MEMORY	00000110
READ_MEMORY	00000120
ASK_EXCLUSIVITY	00000130
INVALIDATE_BLOCK	00000140
WRITE_BACK	00000150
EXCLUSIVITY_GRANTED	00000160
EXCLUSIVITY_DENIED	00000170

B. *ACESSO A MEMÓRIA COM HIT*

Operações de leitura na memória compartilhada que tem como destino um bloco que está presente e válido na *cache* local do EP não afetam a memória compartilhada. Neste caso, nenhuma ação é requerida por parte da política de coerência, pois estas não afetam o sistema imediatamente. Caso a operação seja de escrita, o bloco deve estar no estado exclusivo para que não seja necessário o acesso a memória compartilhada.

C. *ESCRITA EM BLOCO COM MISS*

Quando um bloco não se encontra na memória *cache*, é necessário que este seja trazido da memória compartilhada para a memória *cache* do EP requisitante. Caso o bloco a ser atualizado na *cache* tenha seu endereço mapeado para uma linha da *cache* que contém um bloco no estado modificado (*dirty*), o *microkernel* deverá escrever este bloco na memória compartilhada antes de buscar o novo bloco, caracterizando uma operação de *write-back*.

Após o *write-back*, o *microkernel* solicita para a memória compartilhada a leitura do bloco que não estava presente na *cache*. Existem dois cenários possíveis na leitura de um bloco da memória, os quais são ilustrados na Figura 4 e 5.

O bloco solicitado para a memória compartilhada pode estar em um dos seguintes estados: compartilhado ou exclusivo. Caso o bloco esteja no estado compartilhado (Figura 4) este pode ser lido da memória imediatamente, pois esta contém a versão atualizada deste.

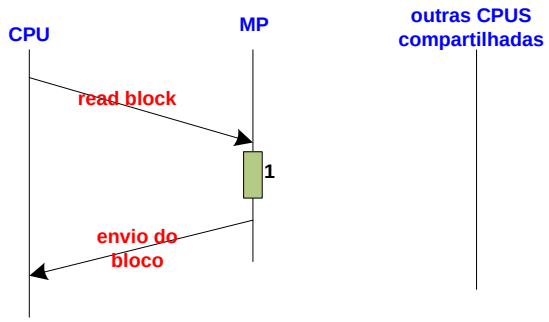

Figure 4. Leitura na memória principal (MP): requisição de substituição de um bloco compartilhado na memória compartilhada.

Caso o bloco esteja no estado exclusivo, é necessário que seja enviado ao EP detentor do direito de exclusividade, um pedido de *write-back*. Este pedido tem por objetivo informar de que outro EP quer escrever no respectivo bloco. O EP receptor do pedido de *write-back* envia o bloco para o EP solicitante diretamente (Figura 5). A escrita em memória do bloco é dispensada, pois o bloco seria imediatamente alterado por outro EP. Esta estratégia evita uma operação de escrita e uma operação de leitura da memória compartilhada, economizando vários ciclos de operação. Quando o EP detentor do direito de exclusividade conclui o envio do pacote para o EP solicitante, este não detém mais o direito de exclusividade sobre o bloco.

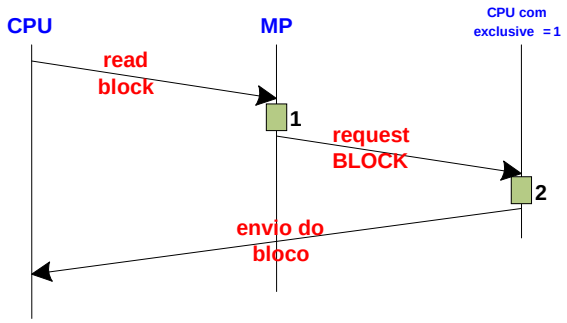

Figure 5. Leitura na memória: requisição de substituição de um bloco exclusivo.

Em operações de escrita onde o bloco destino se encontra no estado compartilhado na memória compartilhada, após o término da operação de leitura do bloco da memória compartilhada, o *microkernel* requisita exclusividade sobre o bloco. A memória compartilhada envia neste momento uma mensagem de invalidação do bloco para todos os processadores que tinham uma cópia deste (Figura 6). A mensagem de invalidação tem por conseqüência a invalidação do bloco na *cache* do EP receptor da mensagem, obrigando o EP a buscar o bloco da memória compartilhada no próximo acesso.

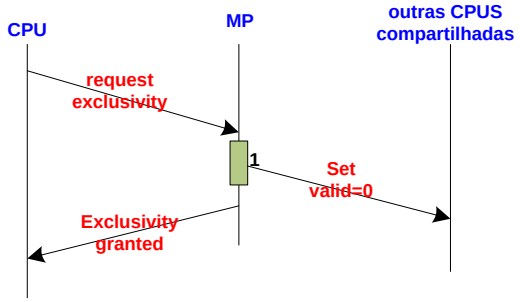

Figure 6. Escrita na memória: pedido de exclusividade sobre o bloco implica no envio de mensagens de invalidação para os outros EPs.

D. LEITURA DE BLOCO COM MISS

De maneira similar a operações de escrita, caso o bloco não se encontre na *cache* este deve ser buscado na memória compartilhada, e operações de atualização da memória compartilhada subseqüentes devem ser executadas. Entretanto, para executar uma operação de leitura o EP não precisa requisitar a exclusividade sobre o bloco.

VI. RESULTADOS

A validação da plataforma proposta deu-se através da utilização do simulador RTL Modelsim. A configuração da plataforma HeMPS MPSoC utilizada para a simulação é descrita a seguir: NoC Hermes com topologia de malha 2x3, largura de flits igual a 16 bits; algoritmo de roteamento XY; e controle de fluxo baseado em crédito.

Os EPs são constituídos por: 1 processador mestre (cujo endereço na rede é 0x0001), 4 processadores escravos e 1 memória compartilhada (endereço na rede 0x0010). A memória compartilhada possui 512 blocos de 128 palavras e a memória *cache* possui 16 blocos de 128 palavras.

A demonstração da correta operação do protocolo de coerência é realizada através de uma operação de leitura. O bloco a ser lido da memória compartilhada encontra-se no estado exclusivo, de acordo com a Tabela 3, o que indica que a cópia do bloco armazenada na memória *cache* do EP escravo cujo endereço na rede é 0x0021 está modificada com relação à cópia deste bloco na memória compartilhada.

A operação de leitura na memória compartilhada condizente com o estado do bloco requerido é constituída pelas seguintes fases:

1. envio do pacote de requisição de leitura do bloco para a memória compartilhada;
2. pedido de atualização do bloco para o EP que detém o direito de exclusividade sobre o bloco;
3. envio do bloco para atualização da memória compartilhada;
4. envio do bloco para o EP que fez a requisição da operação de leitura.

TABELA 3. ENTRADA DA MEMÓRIA DE DIRETÓRIO REFERENTE AO BLOCO 1.

Valid	Shared	Exc	CPU01	CPU11	CPU21	CPU20
1	**0**	**1**	**0**	**0**	**1**	**0**

A Figura 7 ilustra a chegada de um pacote de requisição de leitura na memória compartilhada. Na Figura 7, cada palavra (dois *flits*) é delimitada por um quadrado e este é associado a um número que corresponde a uma parte do pacote como detalhado a seguir: **1** cabeçalho do pacote; **2** serviço *0x00000120*, que corresponde a um pedido de leitura de bloco; **3** endereço do EP requisitante na rede intra-chip; **4** bloco destino da operação a ser executada na memória compartilhada, neste caso (*0x00000080*) que corresponde ao endereço inicial do bloco 1; **5** indica o identificador (*id*) da tarefa solicitante; **6** especifica o tamanho do bloco a ser lido.

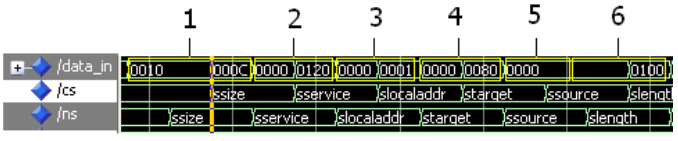

Figure 7. Pacote de requisição de leitura de bloco enviado para a memória compartilhada.

O bloco 1 solicitado pelo EP de endereço *0x0001* se encontra no estado exclusivo, como indicado na Tabela 3. Portanto, é necessário que um pedido de atualização do bloco (*write-back*) seja enviado ao processador detentor do direito de exclusividade. Na figura 8 a palavra **1** indica o destinatário do

pacote (*0x0021*). A palavra **2** indica o serviço a ser executado no bloco determinado por **3**.

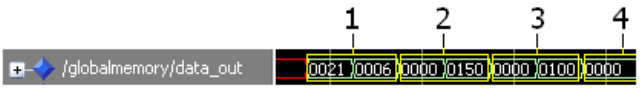

Figure 8. Pacote com pedido de *write-back* do bloco 1.

O EP com endereço de rede igual a *0x0021*, após receber o pedido de *write-back*, copia os dados referentes ao bloco da memória *cache* e monta um pacote de escrita contendo o bloco alterado por este, o que será atualizado na memória compartilhada (Figura 9). Após o envio do pacote, o *microkernel* deste EP altera o estado do bloco requisitado na memória *cache* para inválido. A escrita de um bloco na memória compartilhada faz com que este tenha sua entrada na memória de diretório atualizada para um novo estado: compartilhado.

Na figura 9, a palavra **1** indica o cabeçalho; **2** indica o serviço a ser executado na memória compartilhada; **3** indica o EP origem do pacote; **4** indica o bloco destino da memória compartilhada; **5** indica o identificador da tarefa; **6** indica o tamanho do bloco; **7** contém 128 palavras a serem escritas no bloco indicado por **4** na memória compartilhada.

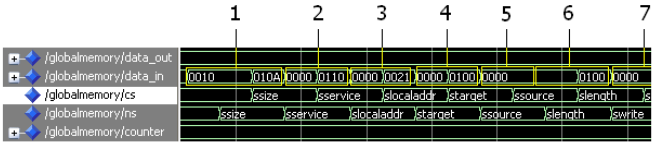

Figure 9. Pacote de escrita referente ao bloco 1 na memória compartilhada.

Após a atualização do bloco na memória compartilhada, este é enviado ao EP requisitante. Ao término da operação de envio do bloco, ocorre uma interrupção na interface de rede do EP e o *microkernel* é solicitado para fazer o tratamento da chegada do pacote. Este por sua vez, programa o DMA para copiar o bloco para a memória *cache*, finalizando a operação de leitura em memória.

A execução desta operação de leitura, que correspondente ao pior caso de desempenho (necessidade de *write-back* pelo processador que detém a exclusividade), consumiu 3.123 ciclos de *clock*. Para fins de comparação, o *timeslice* (período de tempo em que uma tarefa ocupa o processador até ser interrompida) é de 8.194 ciclos de *clock*, o que significa que esta operação executa por 38,1% do período de um *timeslice*. Este é um desempenho muito bom, demonstrando a eficiência do método proposto.

VII. CONCLUSÃO

Este artigo apresentou uma proposta de hierarquia de memória para uma plataforma MPSoC homogênea baseada em NoC. A hierarquia é composta de uma memória compartilhada para dados, além de duas memórias para cada EP: uma memória privada para dados e instruções e uma memória *cache* para dados globais. Além disso, a coerência do sistema é mantida através da utilização de uma política baseada em diretório.

A implementação da política de coerência baseada em diretório possibilita que blocos sejam compartilhados entre os EPs, onde a coerência do sistema é mantida através de uma memória de diretório que armazena o estado corrente de todos os blocos da memória compartilhada, o que garante que cada acesso a um bloco obtenha a cópia mais atualizada deste.

Como contribuições deste trabalho pode-se citar: a disponibilidade de um sistema MPSoC com hierarquia de memória em domínio público, permitindo a comunidade acadêmica realizar experimentos sobre MPSoCs baseados em NoC; e a proposta de uma abordagem modular e híbrida de hierarquia de memória, que visa facilitar o projeto do hardware com a implementação de algumas rotinas de tratamento no *microkernel*.

Este trabalho encontra-se em etapa parcial. Como perspectivas futuras citam-se a sintetização da hierarquia de memória, a fim de validar o correto funcionamento do modelo em hardware, uma vez que o grupo de pesquisa do autor já contém um protótipo da plataforma MPSoC sintetizada. Também, existe como perspectiva a comparação da implementação com outros trabalhos, além da execução de testes com aplicativos como multiplicação de matrizes e detecção de curvas em imagens.

RERFERÊNCIAS

[1] Duller, A.; et al. “picoArray Technology: The Tool's Story”. In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2005, pp.106-111.

[2] Kahle, J.A. , M. N. Day, H. P. Hofstee, et al. “Introduction to the Cell Multiprocessor”. IBM J. Res & Dev., Vol.49(4/5), Jul./Sep. 2005, pp. 589-604.

[3] Jerraya, A; Tenhunen, H; Wolf, W. “ Guest Editors' Introduction: Multiprocessor Systems-on-Chips”. IEEE Computer, Vol. 38 (7), 2005, pp. 36-40.

[4] Gupta, R. K., Zorian, Y. “Introducing Core-Based System Design”. IEEE Design & Test of Computers, Vol.14 (4), Out./Dec. 1997, pp. 15-25.

[5] Guerrier, P.; Greiner, A. “A Generic Architecture for On-Chip Packet-Switched Interconnections”. DATE, 2000, pp. 250-256.

[6] Moraes, et. al. “HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip”. Integration the VLSI Journal, Amsterdam, Vol. 38(1), Out. 2004, pp. 69-93.

[7] Kandemir, M.; Dutt, N. “Memory Systems and Compiler Support for MPSoC Architectures”. Multiprocessor Systems-on-Chips, Kluwer Academic Publishers, pp. 251–281, 2005.

[8] Benini, L.; De Micheli, G. “Networks on Chips: a New SoC Paradigm”. IEEE Computer, Vol. 35 (1), Jan. 2002, pp.70-78.

[9] Girão, et. Al. “Cache Coherency Communication Cost in a NoC-based MPSoC Platform”. SBCCI’07, Set. 2007.

[10] Kim, D.; Kim, M.; Sobelman, G.E. “DCOS: Cache Embedded Switch Architecture for Distributed Shared Memory Multiprocessor SoCs,” Proceedings, IEEE International Symposium on Circuits and Systems, 2006, pp. 979- 982.

[11] Petrot, F.; Greiner, A.; Gomez, P.; “On Cache Coherency and Memory Consistency Issues in NoC Based Shared Memory Multiprocessor SoC Architectures”. 9th EUROMICRO Conference, Aug. 2006, pp. 53 – 60.

[12] Jerger, E. N. D.; Peh, L.; Lipasti, M. H. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In Proceedings of the 2008 41st International Symposium on Microarchitecture, November 08, 2008, pp. 35-46.

[13] PLASMA Processor. Capturado em www.opencores.org/, Agosto 2009.

[14] Lenoski, D; Laudon, J; Gharachorloo, K.; Weber, W; Gupta, A.; Hennessy, J.; Horowitz, M.; and Lam, M. The stanford dash multiprocessor. Computer, Vol. 25(3), 1992, pp. 63-79.

[15] Woszezenki, C. “Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2007, 121p.