

CANAIS VIRTUAIS EM REDES INTRA-CHIP – IMPLEMENTAÇÃO NA REDE HERMES

Everton Alceu Carara, Aline Vieira de Mello, Ney Laert Vilar Calazans, Fernando Gehm Moraes

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) - Faculdade de Informática (FACIN)
Av. Ipiranga, 6681 - Prédio 30 / Bloco 4 - CEP 90619-900 - Porto Alegre - RS - BRASIL
Fone: +55 51 3320-3611 - Fax: +55 51 3320-3621

{carara, alinev, moraes}@inf.pucrs.br

ABSTRACT

Recent works propose Networks on Chip (NoC) as the communication architecture that will be able to provide scalability and performance for communication in future SoCs. Even if NoC performance easily exceeds the performance of buses, NoCs have throughput limited to a fraction of the nominal network capacity. This limitation comes from phenomena like packet collision during routing, and buffers space scarcity. One method to increase NoC throughput is to employ virtual channels. Virtual channels are a time division multiplexing approach that enables the concurrent transmission of different packets over the same physical channel. Each packet belonging to a given virtual channel is stored in an individual buffer, improving throughput. This work presents a general discussion of the use of virtual channels intra-chip environments and introduces the implementation of virtual channels for a specific NoC infrastructure, called Hermes. Preliminary results concerning functional validation of the approach are presented as well.

1. INTRODUÇÃO

A interconexão de um SoC (*System on Chip*) por barramento é simples, sob o ponto de vista de implementação, apresentando entretanto diversas desvantagens [1]: (i) apenas uma troca de dados pode ser realizada por vez; (ii) existe necessidade de desenvolver mecanismos inteligentes de arbitragem do meio físico para evitar desperdício de largura de banda; (iii) a escalabilidade é limitada; (iv) o uso de linhas globais em um circuito integrado com tecnologia submicrônica impõe sérias restrições. Estas desvantagens podem ser parcialmente contornadas através do uso de, por exemplo, hierarquia de barramentos, onde o problema continua existindo, sendo apenas minimizado.

Uma arquitetura de interconexão que pode solucio-

nar os problemas relacionados ao uso de barramento, simples ou hierárquicos, são as redes intra-chip [1][2], um conceito denominado *network on chip* – NoC. NoCs herdaram das redes de computadores e de sistemas distribuídos as características das camadas de protocolos e o conceito de ligação de nodos à rede. A comunicação entre os nodos ocorre pela troca de mensagens transferidas por meio de roteadores e canais intermediários até atingir o seu destino [1]. O throughput das redes de interconexão é limitado a uma fração (tipicamente 20%-50%) da capacidade da rede por causa da alocação de recursos acoplados [3]. Redes de interconexão são compostas por dois tipos de recursos: canais e buffer. Tipicamente, um único buffer é associado a cada canal. Canais virtuais desacoplam a alocação de recursos por prover múltiplos buffers para cada canal na rede, permitindo aumentar o grau de liberdade na alocação de recursos por pacote.

Neste cenário, o presente trabalho tem como objetivo apresentar os conceitos e as vantagens de canais virtuais, tendo como estudo de caso a rede Hermes [4].

Este artigo está organizado da seguinte forma. A Seção 2 apresenta os canais virtuais. A Seção 3 aborda a implementação de canais virtuais na rede Hermes. A seção 4 apresenta conclusões e direções para trabalhos futuros.

2. CANAIS VIRTUAIS

Esta Seção apresenta conceitos básicos relacionados a canais virtuais para redes diretas, topologia *mesh* e chaveamento por pacotes utilizando política *wormhole*.

Nas redes diretas, cada nodo de chaveamento possui um nodo de processamento associado, e esse par pode ser visto como um elemento único dentro do sistema, tipicamente referenciado pela palavra nodo. Nas redes indiretas os nodos de processamento possuem uma interface para uma rede de nodos de chaveamento. Somente alguns nodos de chaveamento possuem conexões para

nodos de processamento e apenas esses podem servir de fonte ou destino de uma mensagem.

As redes de interconexão são compostas por: nodos de chaveamento e canais físicos. Em geral, cada nodo de chaveamento em uma rede de interconexão possui um conjunto de *buffers* e um *crossbar*. Para cada canal físico, há um *buffer* de entrada associado, o qual armazena os *flits* recebidos para que depois os mesmos sejam posteriormente enviados. Enquanto um pacote A tem um *buffer* alocado, nenhum outro pacote poderá utilizar o canal a ele associado até que o pacote A o libere. Se o pacote A ficar bloqueado na rede enquanto ocupa o *buffer*, o canal associado fica ocupado e nenhum outro pacote pode usá-lo.

Canais virtuais dissociam a alocação de um único *buffer* por canal físico, dividindo-o em vários canais virtuais, proporcionando assim a associação de vários *buffers* para cada canal físico. Se um pacote A aloca um *buffer* associado a um canal virtual, outro pacote pode alocar um outro *buffer* associado a outro canal virtual do mesmo canal físico. A Figura 1 ilustra a adição de dois canais virtuais para cada canal físico à rede de interconexão. Enquanto o pacote A é transmitido por um dos canais virtuais do canal físico N, o pacote B pode ser transmitido pelo outro canal virtual do mesmo canal físico.

Adicionar canais virtuais a uma rede de interconexão é análogo a adicionar pistas (*lanes*) em uma estrada. Uma rede sem canais virtuais é formada por estradas de uma só pista. Nesse tipo de rede, um pacote bloqueado ocupando um canal bloqueia todos os demais pacotes que precisam desse canal. Adicionando canais virtuais diminui-se o número de pacotes bloqueados, ou seja, reduz-se o congestionamento na rede.

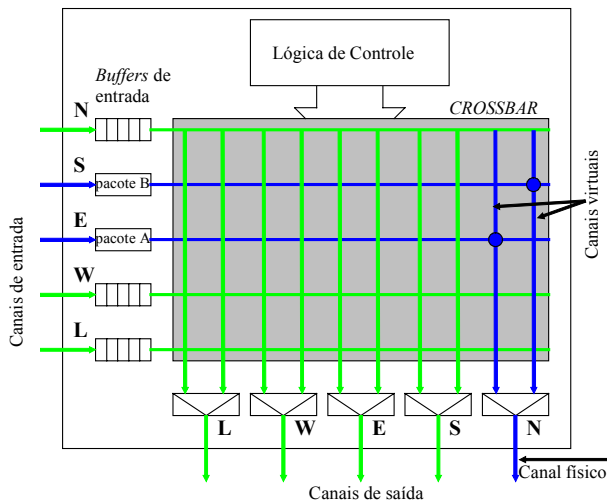


Figura 1 - Canais físicos divididos em canais virtuais.

Os canais virtuais foram apresentados pela primeira vez em [5], com o intuito de prevenir *deadlocks* que podem ocorrer em redes que utilizam chaveamento

wormhole e roteamento adaptativo. O *deadlock* ocorre quando existe uma dependência cíclica de recursos na rede. A Figura 2 mostra parte de uma rede de interconexão com quatro roteadores (1, 2, 3 e 4) possuindo cada um quatro portas bidirecionais (N, S, E, e W) conectadas a um núcleo *crossbar*. Cada um dos roteadores tem um pacote a ser transmitido.

- O roteador 1 tem um pacote A que tem como destino o roteador 4.
- O roteador 2 tem um pacote B que tem como destino o roteador 3.
- O roteador 3 tem um pacote C que tem como destino o roteador 2.
- O roteador 4 tem um pacote D que tem como destino o roteador 1.

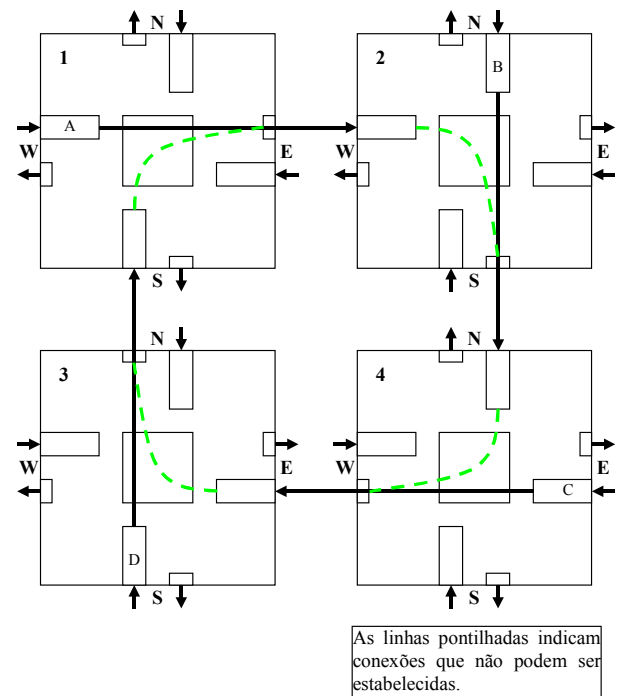


Figura 2 – Dependência cíclica que origina o *deadlock*.

A dependência cíclica ocorre porque existem quatro pacotes sendo transmitidos simultaneamente na rede, e cada um dos pacotes utiliza um canal que é requisitado por outro pacote para que este último atinja seu destino. Por exemplo, os *flits* do pacote A começam a ser transmitidos para o roteador 2 pela porta de saída E. Chegando no roteador 2 pela porta de entrada W, o pacote requisita a porta de saída S para chegar ao seu destino (roteador 4), porém esta porta já está em uso pelo pacote B. O mesmo ocorre com os demais pacotes que estão sendo transmitidos, fechando o ciclo, dando origem ao *deadlock*.

A solução para o problema apresentado na Figura 2 pode ser obtida dividindo cada um dos canais físicos em dois canais virtuais, quebrando assim a dependência cí-

clica, como ilustra a Figura 3. Retomando o exemplo do pacote A, assim que seus *flits* chegam pela porta de entrada W do roteador 2, ele requisita a porta de saída S. Essa porta já está com um dos canais virtuais ocupado pelo pacote B, porém, o outro canal virtual está disponível, logo o pacote A passa a utilizá-lo para atingir seu destino.

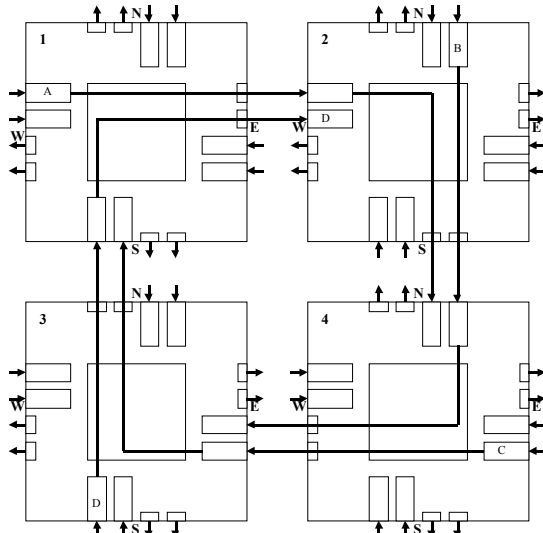


Figura 3 – Quebrando a dependência cíclica com o uso de canais virtuais.

Uma rede convencional organiza os *buffers* de entrada como *FIFOs*, as quais só podem armazenar pacotes inteiros em seqüência. Se um pacote for bloqueado por conta de uma colisão, o canal físico também ficará bloqueado devido ao bloqueio de cabeça de fila (*HOL – head-of-line blocking*) e nenhum outro pacote poderá utilizar esse canal.

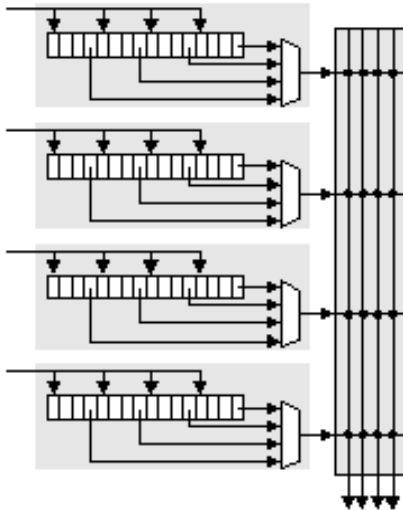


Figura 4 – Buffers SAMQ.

Uma rede com canais virtuais organiza os *buffers* em pequenas filas de profundidade menor. Com essa organi-

zação obtém-se uma coleção de *buffers* que podem ser alocados independentemente uns dos outros. Isso aumenta a flexibilidade na alocação de *buffers*, resolve o problema de *HOL*, melhora a utilização do canal e aumenta o *throughput* global da rede. Três organizações de *buffers* são referenciadas na literatura [6] para a implementação de canais virtuais: *SAFC (Statically Allocated, Fully Connected)*, *SAMQ (Statically Allocated Multi-Queue)* e *DAMQ (Dynamically-Allocated, Multi-Queue)*. A Figura 4 ilustra a organização *SAMQ*, a qual visa simplificar o gerenciamento do *crossbar* através da multiplexação das saídas dos *buffers* de entrada atribuídos a uma mesma porta de saída.

O controle de fluxo com canais virtuais é realizado em dois níveis. No nível de pacote, os pacotes são atribuídos aos canais virtuais (ou *lanes*). No nível de *flit*, os *flits* são atribuídos aos *buffers* [3].

Na comunicação entre dois roteadores adjacentes há um roteador transmissor e um roteador receptor. A atribuição de *lane* é feita pelo roteador transmissor, o qual monitora todos os canais de saída, mantendo a informação sobre a disponibilidade dos *lanes*. Para um determinado pacote no *buffer* de entrada do roteador transmissor, é selecionado um canal de saída particular com base no destino do pacote e no resultado do algoritmo de roteamento. A lógica de controle de fluxo então atribui esse pacote a qualquer *lane* livre no canal físico selecionado. Se todos os *lanes* estão em uso, o pacote é bloqueado.

O controle no nível de *flit* é usado após a atribuição de um *lane* a um pacote. Tipicamente, em redes com controle de fluxo bloqueante, é necessário que o roteador receptor envie ao roteador transmissor uma informação a respeito da disponibilidade de espaço para armazenar um novo *flit*. Essa informação pode ser transferida através de fios dedicados a essa função, como no caso de um controle de fluxo baseado em créditos, ou através de uma banda adicional no canal oposto ao canal de dados [7], o que exige o uso de enlaces bidirecionais. Além disso, no canal de envio, no qual os *flits* de dado são transmitidos para o receptor, a informação relativa ao *lane* selecionado deve ser enviada juntamente com esses *flits*.

Em [3] foram realizadas simulações de redes indiretas multiestágio a fim de medir o efeito dos canais virtuais no desempenho da rede. As simulações foram feitas em nível de *flit* e a quantidade de armazenamento total foi mantida constante em 16 *flits* por nodo, enquanto que o número de canais virtuais foi variado de 1 a 16 em potências de dois. O tamanho dos pacotes foi fixado em 20 *flits*. Os resultados mostraram que o aumento do número de *lanes* pode levar a incrementos da vazão (ou *throughput*) de até 3,5 vezes em relação às redes sem canais virtuais. As simulações também indicaram que o acréscimo de *lanes* tem pouco efeito sobre a latência abaixo da vazão de saturação de uma rede convencional.

3. CANAIS VIRTUAIS NA REDE HERMES

A rede Hermes [4] possui um único *lane* por canal físico, o que permite a ela transmitir somente um pacote por vez em cada um dos seus canais físicos. Esta Seção apresenta o projeto e a implementação de canais virtuais sobre a rede Hermes. Nesta implementação, o objetivo é permitir futuramente o uso de algoritmos de roteamento adaptativos, e principalmente aumentar o desempenho da rede. Adotou-se arquitetura *SAMQ* para os *buffers* de entrada, havendo dois *buffers* independentes, um para cada *lane*.

Os roteadores realizam a transferência de pacotes entre os núcleos. Os componentes de um roteador são: *buffers de entrada*, apresentados na Seção 3.1, *lógica de controle*, que engloba arbitragem e roteamento, apresentada na Seção 3.2.

3.1 Buffers

A Figura 5 ilustra a nova arquitetura do módulo *buffer*, segundo a arquitetura *SAMQ*, contendo dois *buffers* independentes para o armazenamento dos *flits* dos pacotes recebidos. Ao chegar em uma das portas de entrada do roteador, o *flit* é demultiplexado e armazenado no *buffer* correspondente ao *lane* de saída da porta que o está enviando. Por exemplo, se o *flit* é enviado pelo *lane 2* de uma porta de saída, ele será armazenado no *buffer* relativo ao *lane 2*. Assim quando o roteador está recebendo dois pacotes simultaneamente pela mesma porta de entrada, os *flits* de cada pacote são armazenados em *buffers* diferentes. Note que a porta de entrada L (local) não possui canais virtuais, visto que esta porta está conectada ao nodo de processamento.

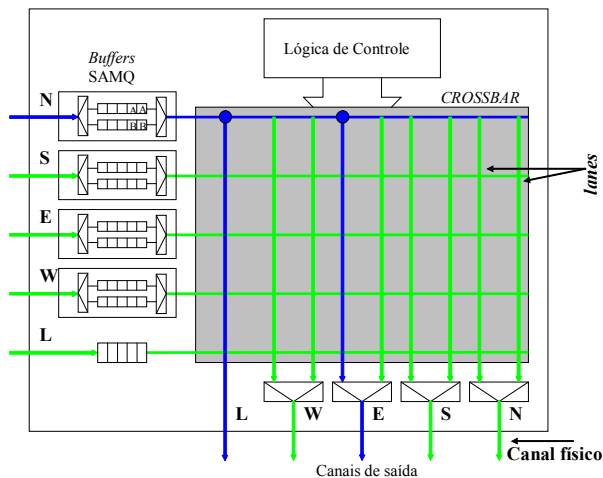


Figura 5 – Roteador com *buffers* multiplexados.

Os *buffers* são organizados como *FIFOs* circulares, possuindo três ponteiros de controle (*first*, *prox* e *last*) e o contador de *flits* *counter_flit*, o qual determina o fim da conexão.

O armazenamento dos *flits* (escrita nos *buffers*) é controlado pelo controle de fluxo, e utiliza-se o método baseado em créditos. Para o envio dos *flits* (leitura dos *buffers*) tem-se uma máquina de estados independente para cada um dos *buffers*, bem como ponteiros de controle e contador de *flits*. A Figura 6 mostra a máquina de estados, e em seguida a explicação da mesma.

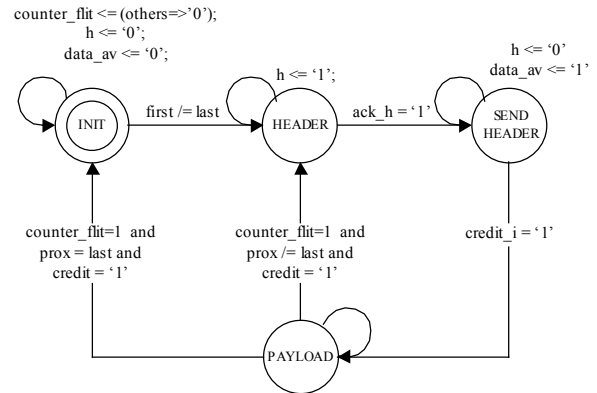


Figura 6 – Máquina de estados para o envio dos *flits*.

- Quando sinal reset é ativado a máquina de estados avança para o estado INIT. No estado INIT os sinais *counter_flit* (contador de *flits* do corpo do pacote), *h* (que indica requisição de roteamento) e *data_av* (que indica a existência de *flit* a ser transmitido) são inicializados com zero. Quando houver algum *flit* armazenado, ou seja, os ponteiros *first* e *last* apontam para posições diferentes, a máquina de estados avança para o estado HEADER.
- No estado HEADER é requisitado o chaveamento ($h='1'$), pois o primeiro *flit* de um pacote é o *header*. A máquina permanece neste estado até que receba a confirmação do chaveamento ($ack_h='1'$), quando então avança para SENDHEADER.
- Em SENDHEADER é indicado que existe um *flit* a ser transmitido ($data_av='1'$). A máquina de estados permanece em SENDHEADER até receber a confirmação da transmissão ($credit_i='1'$), então o ponteiro *first* aponta para o segundo *flit* do pacote e avança para o estado PAYLOAD.
- No estado PAYLOAD é indicado que existe um *flit* a ser transmitido ($data_av='1'$). Quando é recebida a confirmação da transmissão ($credit_i='1'$) é verificado qual o valor do sinal *counter_flit*. Se *counter_flit* é igual a um, a máquina avança para o estado INIT se não há mais nenhum *flit* armazenado, caso contrário avança para HEADER. Caso *counter_flit* seja igual a zero, o sinal *counter_flit* é inicializado com o valor do *flit*, pois este corresponde ao número de *flits* do corpo do pacote. Caso *counter_flit* seja diferente de um e de zero o mesmo é decrementado e a máquina de estados permanece em PAYLOAD enviando os *flits* do pacote.

As máquinas de estado não trabalham simultaneamente, pois a saída do módulo *buffer* é compartilhada pelos *buffers* internos. Ou seja, a interface do módulo *buffer* só pode estar associada a um dos *buffers* por vez. Dessa maneira o módulo *buffer* habilita o funcionamento de uma das máquinas e desabilita a outra, criando um esquema de *time sharing*. O controle de habilitação é feito através do sinal *clock* (*clk*) das máquinas de estado, o qual é independente para cada uma delas. A máquina ativa tem seu *clock* pulsando enquanto a outra o tem mantido em nível alto. A Figura 7 apresenta um exemplo de código fonte em VHDL para o controle de habilitação. O sinal *sel_lane* indica qual máquina está habilitada no momento. Cada uma das máquinas de estado fica selecionada por *SLICE* ciclos de *clock*, desde que o *buffer* correspondente tenha *flits* a enviar.

```
01 clk(L1) <= clock when sel_lane = L1 else '1';
02 clk(L2) <= clock when sel_lane = L2 else '1';
```

Figura 7 – Exemplo de código para o controle de habilitação das máquinas de estado.

3.2 Lógica de Controle

A lógica de controle contém dois módulos: árbitro e lógica de roteamento. O roteador ao receber pacotes utiliza uma política de arbitragem para determinar qual pacote será roteado. Em seguida executa um algoritmo de roteamento para determinar a porta de saída e o *lane* (canal lógico) por onde o pacote deve ser enviado.

Quando o algoritmo de roteamento retorna uma porta de saída e um *lane* livre, a conexão entre a porta de entrada e a porta de saída é estabelecida e são preenchidas as tabelas de chaveamento *port_in*, *port_out*, *lane_in* e *free*. Todas as tabelas possuem dois níveis de indexação.

- A tabela *port_in* é indexada pela porta de entrada e pelo *lane* de entrada, sendo preenchida com a porta de saída da conexão.
- A tabela *port_out* é indexada pela porta de saída e pelo *lane* de saída, sendo preenchida com a porta de entrada.
- A tabela *lane_in* é indexada pela porta de entrada e pelo *lane* de entrada, sendo preenchida com o *lane* de saída.
- A tabela *free* é indexada pela porta de saída e pelo *lane* de saída, sendo preenchida com o estado da porta: livre (1) ou ocupada (0).

A Figura 8 apresenta um possível chaveamento e como as tabelas são preenchidas, considerado que as portas de entrada W, E e L receberam dados pelo *lane* L1. A porta *Local* recebe dados sempre pelo L1, pois ela possui somente um *lane*.

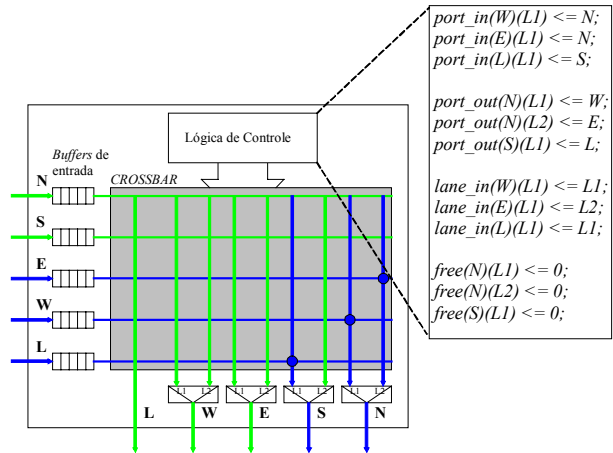


Figura 8 –Chaveamento e as tabelas correspondentes.

Como ilustra a Figura 8:

- A porta de saída *North* está com os dois *lanes* ocupados ($free(N)(L1) = 0$ e $free(N)(L2) = 0$), recebendo dados da porta *West* pelo *lane* L1 ($port_out(N)(L1) = W$) e da *East* pelo *lane* L2 ($port_out(N)(L2) = E$).
- A porta de entrada *West* utiliza o *lane* L1 ($lane_in(W)(L1) = L1$) da porta de saída *North* ($port_in(W)(L1) = N$) enquanto que a porta de entrada *East* utiliza o *lane* L2 ($lane_in(E)(L1) = L2$) da mesma porta de saída ($port_in(E)(L1) = N$).
- A porta de entrada *Local* utiliza o *lane* L1 ($lane_in(L)(L1) = L1$) da porta de saída *South* ($port_in(L)(L1) = S$), logo a porta de saída *South* tem o *lane* L1 ocupado ($free(S)(L1) = 0$) pela porta de entrada *Local* ($port_out(S)(L1) = L$).

O *crossbar* realiza os chaveamentos correspondentes às tabelas contidas na lógica de controle.

3.2.1 Arbitragem

O roteador suporta até nove conexões simultâneas, mas apenas uma pode ser estabelecida a cada instante. Logo, existe a necessidade de um árbitro para determinar qual o pacote deve ser roteado quando mais de um *header* chegar ao roteador em um mesmo instante de tempo. Após atender uma solicitação, a arbitragem aguarda até que o algoritmo de roteamento seja executado e somente após este período volta a atender solicitações. Se o algoritmo de roteamento não consegue estabelecer uma conexão, a porta de entrada volta a solicitar roteamento ao árbitro. A arbitragem utilizada é o de prioridade dinâmica rotativa, *Round Robin*. O código fonte simplificado, em linguagem VHDL, é apresentado na Figura 9.

```

01ask <= '1' when h(LOCAL)='1' or h(EAST)='1' or
02   h(WEST)='1' or h(NORTH)='1' or h(SOUTH)='1'
03   else '0';
04
05incoming <= CONV_VECTOR(sel);
06header <= data(CONV_INTEGER(incoming));
07
08process(sel,h)
09  begin
10    case sel is
11      when LOCAL=>
12        if h(EAST)='1' then prox<=EAST;
13        elsif h(WEST)='1' then prox<=WEST;
14        elsif h(NORTH)='1' then prox<=NORTH;
15        elsif h(SOUTH)='1' then prox<=SOUTH;
16        else prox<=LOCAL; end if;
17      when EAST=>
18        if h(WEST)='1' then prox<=WEST;
19        elsif h(NORTH)='1' then prox<=NORTH;
20        elsif h(SOUTH)='1' then prox<=SOUTH;
21        elsif h(LOCAL)='1' then prox<=LOCAL;
22        else prox<=EAST; end if;
23      when WEST=>
24        if h(NORTH)='1' then prox<=NORTH;
25        elsif h(SOUTH)='1' then prox<=SOUTH;
26        elsif h(LOCAL)='1' then prox<=LOCAL;
27        elsif h(EAST)='1' then prox<=EAST;
28        else prox<=WEST; end if;
29      when NORTH=>
30        if h(SOUTH)='1' then prox<=SOUTH;
31        elsif h(LOCAL)='1' then prox<=LOCAL;
32        elsif h(EAST)='1' then prox<=EAST;
33        elsif h(WEST)='1' then prox<=WEST;
34        else prox<=NORTH; end if;
35      when SOUTH=>
36        if h(LOCAL)='1' then prox<=LOCAL;
37        elsif h(EAST)='1' then prox<=EAST;
38        elsif h(WEST)='1' then prox<=WEST;
39        elsif h(NORTH)='1' then prox<=NORTH;
40        else prox<=SOUTH; end if;
41    end case;
42end process;

```

Figura 9 – Exemplo de código fonte em linguagem VHDL para a seleção da porta de entrada que terá permissão de chaveamento.

3.2.2 Roteamento

O roteamento é o passo executado logo após a arbitragem, pois ele determina por qual porta de saída o pacote selecionado pela arbitragem será enviado. Utiliza-se neste trabalho o algoritmo XY determinístico.

O algoritmo de roteamento XY faz a comparação do endereço do roteador atual com o endereço do roteador destino do pacote (armazenado no primeiro *flit* do pacote). O pacote deve ser enviado para a porta *Local* do roteador quando o endereço $xLyL^1$ do roteador atual for igual ao endereço $xTyT^2$ do roteador destino do pacote. Caso contrário, é realizada, primeiramente, a comparação horizontal de endereços. A comparação horizontal determina se o pacote deve ser enviado para o leste ($xL < xT$), para o oeste ($xL > xT$), ou se o mesmo já está horizontalmente alinhado ao roteador destino ($xL = xT$). Caso esta última condição seja é realizada a comparação vertical que determina se o pacote deve ser enviado para o sul ($yL < yT$), para o norte ($yL > yT$), ou se o mesmo já

¹ $xLyL$ é o endereço da chave atual, onde xL é o endereço horizontal e yL o endereço vertical.

² $xTyT$ é o endereço da chave destino dos dados, onde xT é o endereço horizontal e yT o endereço vertical.

está verticalmente alinhado ao roteador destino ($yL = yT$). Caso esta última condição seja verdadeira ou a porta vertical escolhida esteja com os dois *lanes* ocupados, é realizado o bloqueio dos *flits* do pacote em todos os roteadores intermediários até que algum *lane* da porta de saída escolhida seja liberado.

4. VALIDAÇÃO FUNCIONAL

A Figura 10 ilustra um exemplo de transmissão onde o roteador 20 envia dados para o roteador 21. O roteador 20 envia os dados pela porta de saída *North* (índice 2) e o roteador 20 os recebe pela porta *South*.

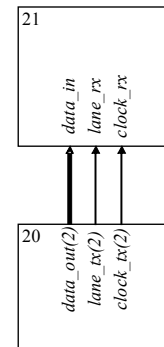


Figura 10 – Transmissão de dados.

A Figura 11 mostra as formas de onda do módulo *buffer* da porta entrada *South* recebendo os dados enviados pela porta de saída *North* do roteador 20. O roteador 20 está enviando dois pacotes simultaneamente utilizando os dois *lanes* de saída da porta *North*.

A entrada *lane_rx* (conectada a *lane_tx(2)*) indica em qual dos *buffers* (*flit_buff*) o dado contido em *data_in* deve ser armazenado. O roteador 20 controla a entrada dos dados nos *buffers* através do sinal *clock_tx(2)* (conectado a *clock_rx*), pois trata-se de um controle de fluxo baseado em créditos. Os dados são armazenados na borda de subida do sinal *clock_tx(2)*.

A Figura 12 mostra as formas de onda da lógica de controle do roteador 20 realizando a arbitragem e o roteamento. Os eventos destacados na Figura 12 são:

1. A lógica de controle recebe um pedido de chaveamento da porta de entrada *West*.
2. A lógica de controle recebe outro pedido de chaveamento, agora da porta de entrada *Local*.
3. É realizada a arbitragem e o roteamento para a porta de entrada *West*.
4. As tabelas de chaveamento são preenchidas e a confirmação de chaveamento é sinalizada para a porta de entrada *West*.
5. É realizada a arbitragem e o roteamento para a porta de entrada *Local*.
6. As tabelas de chaveamento são preenchidas e a confirmação de chaveamento é sinalizada para a porta de entrada *Local*.

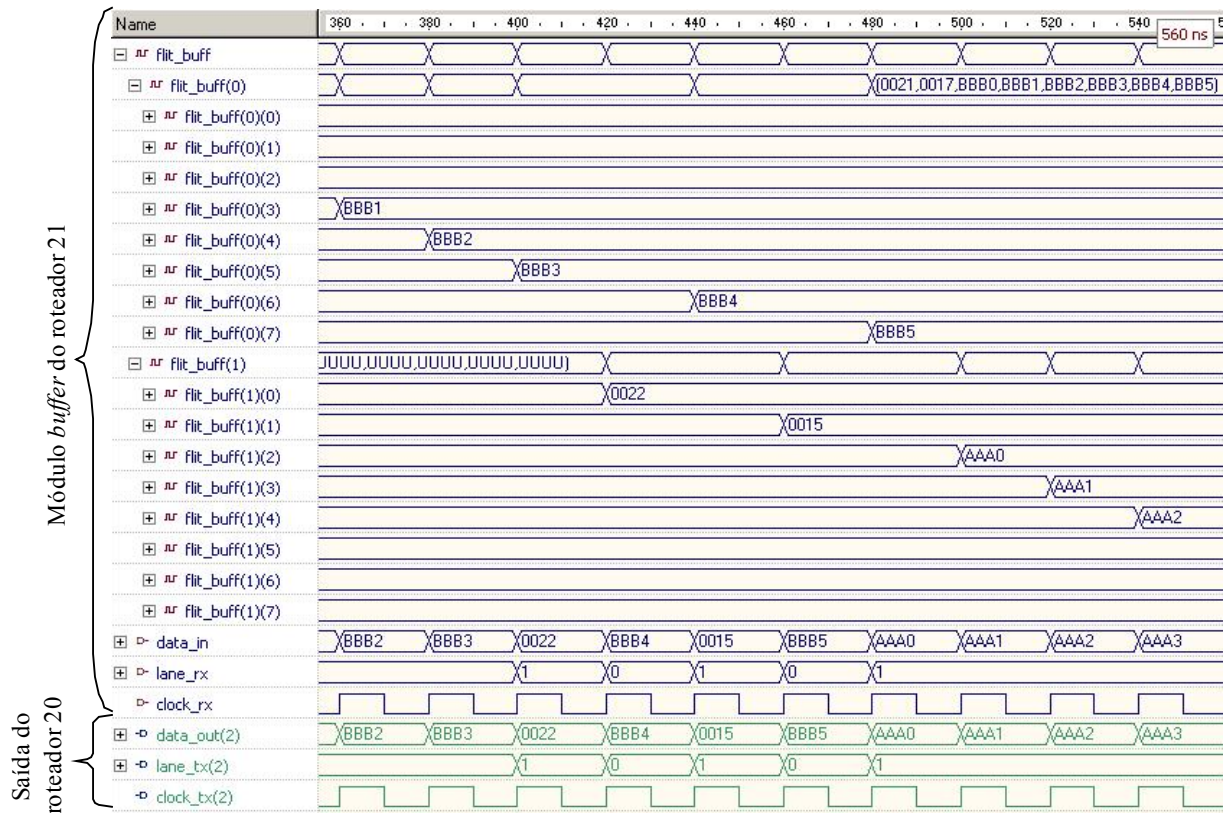


Figura 11 - Entrada de dados no módulo buffer do roteador 21 e saída de dados do roteador 20.

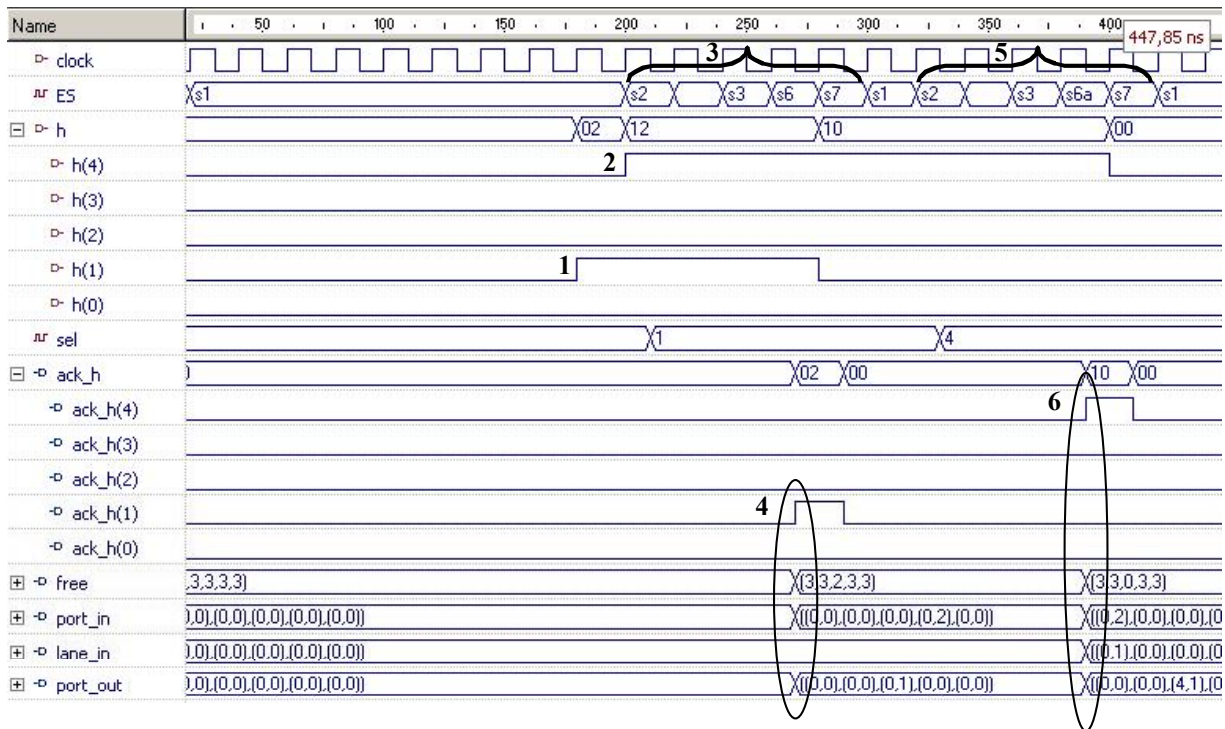


Figura 12 – Arbitragem.

5. CONCLUSÃO

Este trabalho descreveu o conceito de canais virtuais e as suas principais vantagens; (i) quebra da dependência cíclica que origina o *deadlock*; (ii) aumento do desempenho geral de rede. Mostrou também as alterações necessárias na arquitetura do roteador da rede Hermes para a adição de canais virtuais, seguida de sua validação funcional.

Como visto a principal alteração realizada é na organização dos *buffers* de entrada, pois para dar suporte a canais virtuais, eles devem ser capazes de armazenar e enviar *flits* de mais de um pacote simultaneamente. É nessa alteração onde há o maior custo de área envolvido para adição de canais virtuais, pois será necessário mais espaço para armazenamento dos *flits* e mais lógica para o controle dos *buffers*.

A otimização da rede visando à redução do custo de área e o aumento do desempenho serão os próximos trabalhos. Feito isso, novas avaliações serão realizadas com cargas de tráfegos que demonstrem melhor as vantagens de canais virtuais e a utilização de algoritmos de roteamento adaptativos.

6. REFERÊNCIA BIBLIOGRÁFICA

- [1] Benini, L. De Micheli, G. "Networks on chips: a new SoC paradigm". Computer, v.35(1), 2002, pp. 70-78.
- [2] Bergamaschi, R. A.; Bhattacharya, S.; Wagner, R.; Fellenz, C.; Muhlada, M.; White, F.; Daveau, J. M.; Lee, W. R. "Automating the design of SoCs using cores". IEEE Design & Test of Computers, v.18(5), 2001, pp. 32-45.
- [3] Dally, W. J. "Virtual-Channel Flow Control". In: 17th International Symposium on Computer Architecture (ISCA), 1990, pp. 60-68.
- [4] Moraes, F. G.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. "HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip". Integration: the VLSI Journal, v. 38(1), 2004, pp. 69-93.
- [5] Dally, W. J.; Seitz, C. L. "Deadlock-Free Message Routing in Multiprocessors Interconnection Networks". IEEE Transactions on Computers, v.36(5), 1987, pp. 547-553.
- [6] Tamir, Y.; Frazier, L. "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches". IEEE Transactions on Computers, v.41(6), 1992, pp.725-737.
- [7] INMOS. "IMS C004 Programmable Link Switch". In: INMOS Engineering Data, 1989, pp. 479-501. pp. 105-112.