Sistemas Dinamicamente Reconfiguráveis com Comunicação Via Redes Intra-Chip

Leandro Heleno Möller[1] Fernando Gehm Moraes[2]

Resumo — A constante demanda por novos equipamentos eletrônicos cada vez menores, mais duráveis, com melhores desempenhos e que executem mais funções em um único dispositivo motivam a pesquisa em novos métodos de implementação. A tecnologia atual utiliza microprocessadores quando flexibilidade é almejada e circuitos integrados específicos para uma aplicação (ASIC – Application Specific Integrated Circuit) quando desempenho é requisitado. O problema é que a necessidade destas duas características implica o emprego de ambas em um mesmo equipamento eletrônico. O objetivo deste trabalho é propor um método alternativo a partir da utilização de FPGAs para a implementação de equipamentos eletrônicos, provendo simultaneamente flexibilidade e desempenho em um único circuito integrado. Neste trabalho uma plataforma multiprocessada, dinamicamente reconfigurável construída sobre uma rede intra-chip é apresentada. A construção desta plataforma é a prova de conceito que multiprocessadores podem ter seu hardware modificado em tempo de execução. A principal contribuição do trabalho é a proposta de um novo fluxo de projeto para a implementação de Sistemas Dinamicamente Reconfiguráveis (SDRs).

Palavras Chave —SoCs, NoCs, CoCs, MPSoCs, FPGAs, SDRs, reconfiguração parcial, reconfiguração dinâmica.

I. INTRODUÇÃO

Os dois maiores fabricantes de microprocessadores estão lançando no ano de 2005 processadores com dois núcleos em um único chip [INT05] [AMD05]. Este fato mostra que a evolução dos sistemas microprocessados dar-se-á por multiprocessamento, dado que ganhos de desempenho com um único processador são mínimos com a tecnologia atual. Com esta característica, os computadores pessoais passam a ser máquinas no mínimo biprocessadas, possuindo uma baixa latência de comunicação entre os núcleos, já que eles estão fisicamente implementados no mesmo chip. Nestes chips a comunicação entre os núcleos é feita de forma direta, através de barramentos.

Barramentos, segundo [BEN02], possuem diversas limitações: (i) o número de módulos de hardware que podem ser conectados ao barramento é tipicamente na ordem da dezena; (ii) apenas uma troca de dados pode ser realizada por vez, pois o meio físico é compartilhado por todos os núcleos de hardware, reduzindo o desempenho global do sistema; (iii) necessidade de mecanismos inteligentes de arbitragem do meio físico para evitar desperdício de largura de banda; (iv) o uso de linhas globais em um circuito integrado com tecnologia submicrônica impõe sérias restrições ao desempenho do sistema devido às altas capacitâncias e resistências parasitas inerentes aos fios longos fios. Estas limitações podem ser parcialmente contornadas através do uso de, por exemplo, hierarquias de barramentos, onde o problema continua existindo, sendo apenas reduzido.

Uma maneira de minimizar os problemas oriundos da arquitetura de barramentos é através da utilização de redes de comunicação internas ao circuito integrado [DAL01] [WIN01], no que se denomina hoje de NoC (*Network on Chip*). Uma NoC é uma rede intra-chip [BEN02] composta por módulos de hardware conectados a roteadores, que por sua vez são conectados entre si através de canais de comunicação. Este meio de comunicação possibilita que diferentes módulos da NoC se comuniquem simultaneamente, sendo este um meio de comunicação promissor para futuros MPSoCs (*Multiprocessed SoCs*) [YE03] e CoCs (*Cluster On Chip*) [OTT00].

NoCs podem conectar diversos tipos de núcleos de hardware distintos, como por exemplo processadores, DSPs, interfaces de E/S ou qualquer outro tipo de hardware dedicado. No entanto, pode ser necessário "empacotar" o núcleo que está sendo conectado à NoC caso o mesmo não possua o mesmo protocolo que a rede. Uma vez que o *núcleo* possui o mesmo protocolo de comunicação da rede ele é chamado de *módulo* e pode se comunicar com qualquer outro módulo através da NoC e vice-versa. Este "pacote" do núcleo é chamado de *wrapper* (invólucro).

Uma característica que pode ser agregada a uma NoC, quando a mesma for implementada em dispositivos parcial e dinamicamente reconfiguráveis (e.g. FPGAs), é a possibilidade de substituir módulos conectados a NoC enquanto outros continuam operando. Desta forma é possível ter um hardware virtualmente maior que o fisicamente existente, pois os módulos são configurados no sistema conforme a demanda. Esta é uma característica

[1] Mestrando, bolsista CNPq – moller@inf.pucrs.br
[2] Orientador – moraes@inf.pucrs.br

importante para dispositivos portáteis que necessitam de um tamanho reduzido e não podem consumir potência em excesso.

O objetivo deste trabalho é implementar uma plataforma multiprocessada dinamicamente reconfigurável construída sobre uma rede intra-chip. Uma das contribuições deste trabalho é a proposta de um novo fluxo de projeto para a implementação de SDRs. Este fluxo de projeto se mostrou necessário quando o fluxo modular proposto pela Xilinx [LIM04] demonstrou-se problemático ao tentar expandir o SDR apresentado em [MÖL04]. O fluxo de projeto proposto neste trabalho é totalmente dominado e baseia-se no fluxo de projeto padrão da Xilinx.

O restante deste documento está organizado como segue. A Seção II apresenta o estado da arte na comunicação de módulos reconfiguráveis em SDRs. A Seção III apresenta o fluxo de projeto para a implementação de SDRs proposto por este trabalho. A Seção IV apresenta a família de dispositivos utilizada para embarcar a plataforma multiprocessada, ferramentas de CAD e componentes de hardware para viabilizar a construção de SDRs. A Seção V versa sobre redes intra-chip e a construção de uma rede que suporta reconfiguração parcial e dinâmica de módulos de hardware conectados a ela. Essa rede foi chamada de Artemis[3]. A Seção VI apresenta um estudo de caso que utiliza a rede Artemis como meio de comunicação de um SDR. A Seção VII apresenta o andamento das atividades do presente trabalho. Por fim, as conclusões são apresentados na Seção VIII.

II. Estado da Arte na Comunicação de Módulos em SDRs

Na literatura diversos tipos de sistemas ditos reconfiguráveis foram publicados:

- processadores com unidades funcionais reconfiguráveis: Nano Processor [WIR94], DISC [WIR95], One-Chip [WIT96] , Chimaera [HAU97b];
- processadores conectados a hardware reconfigurável por barramento: PRISM-I [ATH93], PRISM-II [WAZ93], GARP [HAU97a], MORPHOSYS [SIN98];
- processadores conectados por rede estaticamente reconfigurável: RAW [WAI97];
- unidades funcionais reconfiguráveis conectadas a vizinhos: Matrix [MIR96], CHESS [MAR99], ;
- unidades funcionais reconfiguráveis conectadas a vizinhos e a barramento(s): KressArray [HAR95] [HAR97], Dream [BEC00];
- unidades funcionais reconfiguráveis conectadas em pipeline: RaPiD [EBE96], PipeRench [GOL99], Systolic Ring [SAS02];
- módulos reconfiguráveis conectados por barramento: SDR de Dyer [DYE02], SDR de Palma [PAL02], R8NR [MÖL04], SDR de Huebner [HUB04], SDR de Walder [WAL04];
- módulos reconfiguráveis conectados por rede: Gecko [MAR04].

Apesar desta lista não ser exaustiva, é possível separar em dois grupos os trabalhos apresentados até o ano de 2002: o primeiro grupo utiliza sistemas multi-chip para conectar um processador à lógica reconfigurável e o segundo grupo conecta unidades funcionais reconfiguráveis em um único chip. Ambas as abordagens já não atendem mais a demanda dos atuais equipamentos eletrônicos, pois o primeiro grupo possui alta latência para comunicar um processador com a lógica reconfigurável e a granularidade do segundo grupo é pequena para atender o mercado atual.

Os trabalhos publicados após o ano de 2002 tratam os problemas citados nos trabalhos anteriores através da utilização de módulos mais complexos executando em um único chip. Esta abordagem provê baixa latência de comunicação e maior granularidade, sendo o grão um processador ou um hardware dedicado de propósito específico, por exemplo. Dois trabalhos com participação dos Autores que conectam módulos reconfiguráveis através de barramentos são citados na lista: SDR de Palma [PAL02] e R8NR [MÖL04]. Nestes trabalhos três dificuldades para a inserção de um número maior de módulos reconfiguráveis foram detectadas: (i) barramento é um meio de comunicação limitado; (ii) os *tristates* utilizados para comunicação com os módulos reconfiguráveis são componentes escassos no dispositivo reconfigurável; (iii) o fluxo de projeto é problemático.

O sistema Gecko [MAR04] possui uma NoC 3x3 com topologia malha como meio de comunicação, resolvendo o problema (i), mas possuindo os mesmos problemas (ii) e (iii) por utilizar o fluxo de Projeto Modular da Xilinx [LIM04]. Os módulos conectados a esta NoC são: cinco processadores RISC de 16 bits, um bloco 2D IDCT, uma interface para uma CPU externa e dois processadores reconfiguráveis de grão fino. Este sistema foi prototipado em um dispositivo Virtex-II 6000 da Xilinx (XC2V6000).

A NoC do Gecko é composta de três redes, conforme ilustrado na Fig. 1: uma rede de dados, uma rede de controle e uma rede de reconfiguração. Estas três redes distintas foram criadas porque cada rede possui propósitos e necessidades diferentes, permitindo assim tratar cada uma de forma eficiente.

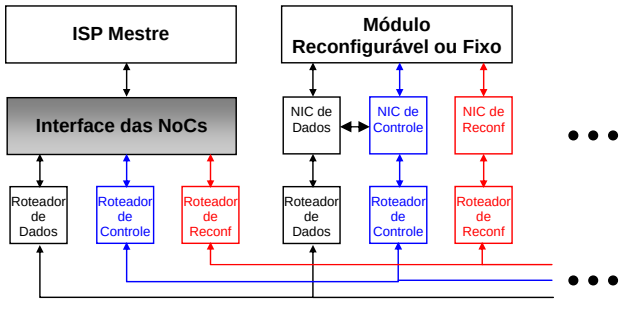

Fig. 1 – O Gecko utiliza três redes: uma rede de dados, uma rede de controle e uma rede de reconfiguração.

[3] Na mitologia grega, Artemis possui o poder de transformar pessoas em animais.

III. Fluxo de Projeto para SDRs

A carência de dispositivos reconfiguráveis de grão grande que permitam reconfiguração parcial e dinâmica induz a utilização de dispositivos FPGA de grão pequeno. Os dois maiores fabricantes de dispositivos reconfiguráveis de grão pequeno disponíveis no mercado que suportam reconfiguração parcial e dinâmica são a Xilinx [XIL05a] e a Atmel [ATM05]. Dentre estes se optou pelos dispositivos Xilinx por diversos motivos: disponibilidade de documentação, trabalhos relacionados ao assunto tanto no meio acadêmico [HUB04] [DYE02] [MÖL04] quanto na indústria, existência de ferramentas que permitem interagir nas fases intermediárias entre a descrição inicial de hardware até a prototipação em FPGAs, diversidade de plataformas e a alta densidade em termos de portas lógicas dos dispositivos.

No contexto de SDRs faz-se necessário definir os termos *sistema* e *projeto*. Utilizar-se-á *sistema* como sendo o equivalente a SDR e possuindo diversos projetos. Cada *projeto* é um dos possíveis estados atuais de configuração do sistema. O fluxo de projeto proposto para a implementação de SDRs em FPGAs Xilinx é ilustrado na Fig. 2. Este fluxo inicia com um projeto em linguagem de descrição de hardware que compõe o projeto inicial (Fig. 2A) e é finalizado pela geração de bitstreams parciais (Fig. 2L ou Fig. 2M). Este fluxo é divido em 3 fases: (i) geração e validação do projeto inicial que não possui características reconfiguráveis (Fig. 2A a Fig. 2D); (ii) projeto inicial com características reconfiguráveis (Fig. 2E a Fig. 2H); (iii) geração de bitstreams parciais (Fig. 2I a Fig. 2M). É importante conhecer cada um dos passos e as respectivas ferramentas deste fluxo, pois algumas destas farão uso de parâmetros avançados e específicos quando aplicadas a SDRs. As Seções a seguir apresentam o fluxo de projeto proposto para SDRs, possuindo correspondência direta com a Fig. 2.

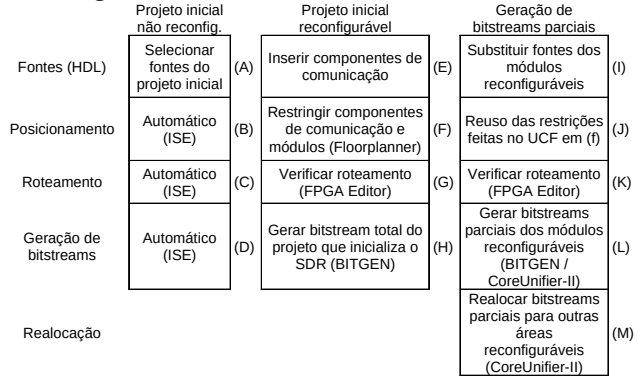

Fig. 2 – Etapas de projeto para SDRs proposto neste trabalho.

A. Definir projeto inicial do SDR

O primeiro passo do fluxo de projeto é definir a estrutura geral do SDR, escolhendo a quantidade de áreas reconfiguráveis que farão parte deste e quais módulos estarão inicialmente configurados nestas áreas reconfiguráveis (Fig. 2A). É aconselhável prototipar este projeto usando configuração total (Fig. 2B a Fig. 2D), de forma a garantir que as restrições temporais deste sejam atendidas e que o mesmo funcione em hardware. O motivo para esta prototipação inicial é garantir a operacionabilidade do ponto de partida do SDR, uma vez que serão posteriormente adicionados novos elementos, agregando complexidade ao todo.

B. Modificar fontes HDL

O passo seguinte é a adição de componentes de comunicação entre os módulos reconfiguráveis e o restante do projeto (Fig. 2E). Estes componentes, também conhecidos como *macros* na literatura, fixam a posição das entradas e saídas entre cada um dos módulos reconfiguráveis instanciados em diferentes projetos do sistema. A construção de projetos com novos módulos reconfiguráveis é feita com o reuso do projeto inicial e a substituição do fonte HDL específico do módulo reconfigurável (Fig. 2I). Para que o sistema funcione corretamente é importante que os módulos reconfiguráveis dos diferentes projetos respeitem a semântica de cada pino da(s) macro(s). Maiores informações sobre macros serão abordadas na Seção IV.B.

C. Restringir posicionamento

O terceiro passo no fluxo de implementação de SDRs é restringir o posicionamento dos módulos reconfiguráveis e das macros (Fig. 2F). Esta restrição pode ser feita pela ferramenta gráfica Floorplanner da Xilinx, que permite selecionar a posição destino dos componentes do projeto dentro do FPGA assim como definir o formato de alocação de lógica dos módulos no mesmo. O Floorplanner gera como saída um arquivo de restrições do usuário (*User Constraints File* - UCF). Este arquivo pode ser depois reutilizado para gerar módulos reconfiguráveis para o sistema (Fig. 2J).

D. Verificar roteamento

Infelizmente, ao restringir a lógica de um módulo a uma determinada área do FPGA, não obrigatoriamente os fios que interconectam os componentes internos deste módulo estarão posicionados dentro dos limites definidos para a lógica. Com freqüência, os fios internos de um módulo invadem a área de outro, como pode ser verificado pela ferramenta FPGA Editor da Xilinx (Fig. 2G e Fig. 2K). Caso um ou mais fios cruzem a área pré-definida, estes ficarão desconectados após a reconfiguração, podendo ocasionar o não funcionamento do projeto e até mesmo um dano físico no dispositivo. A ferramenta FPGA Editor da Xilinx permite visualizar os fios do sistema e rotear manualmente ou automaticamente conexões que apresentem erros.

E. Gerar bitstreams

A ferramenta BitGen da Xilinx é responsável por gerar o bitstream total que inicializa o sistema (Fig. 2H). Três ferramentas são descritas na Seção IV para a geração de

bitstreams parciais de módulos reconfiguráveis (Fig. 2L).

F. Realocar módulos do sistema

Existe ainda um passo opcional para a implementação de SDRs que é a relativização do posicionamento dos módulos reconfiguráveis do sistema (Fig. 2M), que neste trabalho é chamada de realocação. Segundo Compton et. al. [COM00] realocação é a habilidade de determinar em tempo de execução o posicionamento de uma configuração no dispositivo reconfigurável. A realocação pode ser útil quando existir um elevado número de combinações de áreas reconfiguráveis para os módulos do SDR a ser sintetizada ou uma dificuldade de restringir um determinado módulo em uma determinada área reconfigurável. Outro motivo para realocar módulos, ao invés de sintetizá-los novamente para cada nova posição do dispositivo, é que os módulos podem possuir restrições temporais que só podem ser obtidas depois de longas etapas de posicionamento e roteamento. Neste caso, o módulo é sintetizado uma única vez, sendo posteriormente realocado para alguma posição do sistema, com a vantagem de manter as suas características temporais. Módulos que possuem alta taxa de comunicação com módulos distantes do sistema podem ser realocados para ocuparem posições próximas, reduzindo o congestionamento do meio de comunicação, aumentando a eficiência do sistema como um todo.

IV. Suporte a SDRs em Dispositivos Xilinx

A Seção III apresentou o fluxo de projeto para a construção de SDRs, citando ferramentas de CAD específicas para dispositivos Xilinx. Muitas destas ferramentas não foram construídas visando SDRs e algumas são voltadas para uma família de dispositivos específicos da Xilinx. Por este motivo esta Seção primeiramente restringe e justifica para qual família de FPGAs o restante deste documento está sendo endereçado.

A seguir alternativas de macros existentes na literatura e duas macros propostas pelo Autor são estudadas para implementar a etapa descrita na Fig. 2E. As etapas descritas no fluxo de projeto da Fig. 2F a Fig. 2K não são descritas neste documento por utilizarem ferramentas disponibilizadas pelo próprio fabricante. Por último, alternativas de ferramentas para a geração de bitstreams parciais são apresentadas para executar a etapa descrita na Fig. 2L.

A. Família de dispositivos Virtex-II

Dentre as famílias de FPGAs da Xilinx selecionou-se a Virtex-II para embarcar o sistema proposto neste trabalho por possuir uma estrutura de configuração regular e um número maior de portas lógicas equivalentes que a família Virtex. A Fig. 3 apresenta uma visão geral da família de dispositivos Virtex-II.

Os dispositivos Virtex-II são compostos basicamente por blocos lógicos configuráveis (CLBs – *Configurable Logic Blocks*), blocos de entrada e saída (IOBs – *Input/Output Blocks*), blocos de memória (BRAMs – *Block Select RAMs*), gerenciadores digitais de relógios (DCMs – *Digital Clock Manager*), multiplicadores e recursos de roteamento que interconectam todos estes componentes. Os CLBs provêem elementos funcionais para a implementação de lógica combinacional e/ou seqüencial. Cada CLB possui quatro *slices* e dois *tristates*. Cada slice possui dois geradores de função de quatro entradas (Look-Up Tables - *LUTs*), dois flip-flops para armazenamento de um bit e recursos como propagação rápida de vai um. Cada BRAM possui 18Kbits de memória dupla porta, configurável em diversas larguras de dados e endereços.

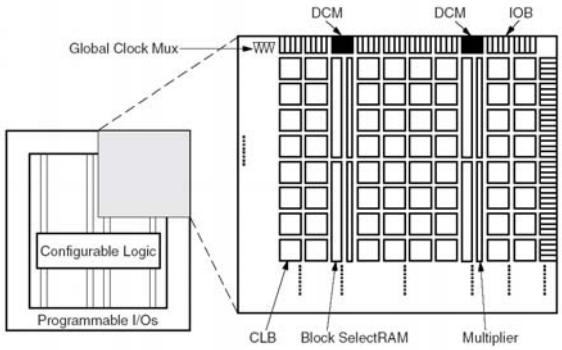

Fig. 3 – Arquitetura básica do dispositivo Virtex-II.

A arquitetura de configuração dos dispositivos Xilinx é organizada como uma matriz bidimensional de bits. No entanto, os dispositivos só podem ser configurados unidimensionalmente em colunas, onde cada coluna é composta por *frames*. O *frame* é a menor unidade que pode ser lida ou escrita do FPGA. Os *frames* são enviados para o FPGA a partir de um arquivo de configuração chamado *bitstream*. Chama-se de *bitstream total* o bitstream que descreve a configuração completa do hardware, e *bitstream parcial* o arquivo de configuração que altera uma parte das funções implementadas no FPGA.

Outra característica dos dispositivos FPGAs Xilinx é que eles são *glitchless*, permitindo que áreas não apresentem nenhum tipo de transitório se os mesmos dados de configuração forem sobrescritos aos existentes. No entanto, caso sejam enviados dados de configuração diferentes dos previamente existentes no FPGA, *glitches* podem ser observados em sinais que ligam a área que está sendo reconfigurada ao restante da lógica que está fixa no dispositivo.

B. Macros

A interconexão entre módulos reconfiguráveis e o restante do sistema deve ser feita por algum componente intermediário, denominado *macro*. As macros permitem definir pontos de interface entre cada um dos módulos reconfiguráveis instanciados em diferentes projetos. Macros, em dispositivos Xilinx, são criadas com a ferramenta FPGA Editor e são instanciadas no código fonte como componentes. Algumas macros existentes na literatura e uma nova proposta de macro são estudadas a seguir.

B.1. Bus Macro

A Bus Macro [LIM04] [XIL04b] é o componente proposto pela Xilinx para interconectar módulos reconfiguráveis ao restante do sistema. A Bus Macro, ilustrada na Fig. 4, é formada por oito *tristates* e disponibiliza quatro fios de um bit para a comunicação do sistema com o módulo reconfigurável ou vice-versa. Vários trabalhos [MÖL04] [BLO04] [WAL04] já utilizaram com sucesso Bus Macros como componente de interconexão de áreas reconfiguráveis utilizando o fluxo de Projeto Modular.

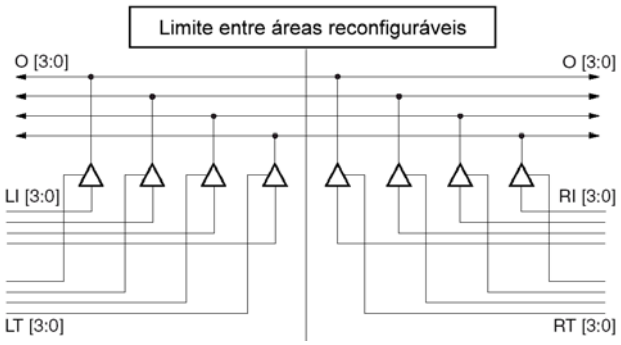

Fig. 4 – Implementação física de uma Bus Macro.

Um ponto negativo da utilização de Bus Macros é o uso de *tristates*, recursos escassos nos FPGAs da Xilinx, existindo apenas dois por CLB. Além disto, Huebner [HUB04] aponta que a ferramenta de roteamento algumas vezes não respeita as restrições feitas pelo projetista e cruza o limite entre as áreas reconfiguráveis na implementação de Bus Macros. A conseqüência deste ato é o mau funcionamento do sistema ou até um curto circuito que pode danificar o FPGA.

B.2. Macro de Huebner

Devido aos problemas supracitados das Bus Macros, Huebner [HUB04] desenvolveu um barramento seguindo a idéia de comunicação em SDRs proposta por Palma [PAL02], onde os elementos que comunicam as diversas áreas reconfiguráveis ficam na parte inferior do FPGA. A vantagem desta abordagem é a possibilidade de isolar a computação na parte superior do FPGA da comunicação, fixada na parte inferior. Desta forma é possível reconfigurar a computação de um módulo sem interromper a comunicação dos outros que estão em execução no sistema. Isto é factível somente porque a reconfiguração de dispositivos FPGAs Xilinx é *glitchless*, conforme explicado na Seção IV.A.

A princípio, Huebner tentou instanciar múltiplas macros iguais lado a lado de forma a implementar um barramento. Devido à arquitetura Virtex-II apresentar mistura de colunas de CLBs com BRAMs e blocos multiplicadores, problemas de roteamento foram detectados. Por este motivo as macros que seriam instanciadas foram todas roteadas manualmente, construindo assim dois barramentos unidirecionais para a comunicação de módulos reconfiguráveis. A Fig. 5A apresenta o barramento de leitura e a Fig. 5B apresenta o barramento de escrita.

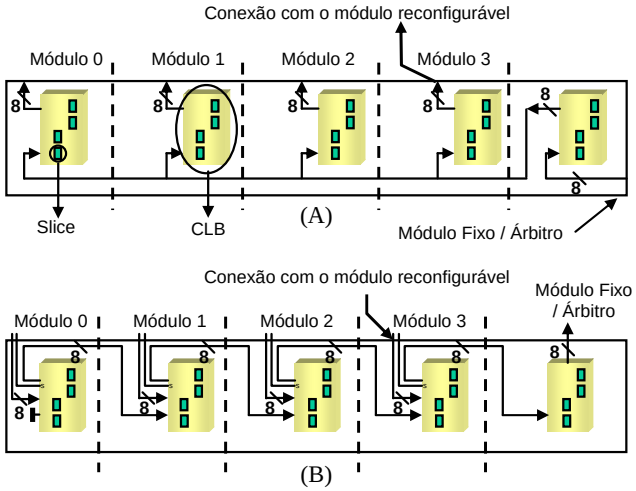

Fig. 5 - Barramentos unidirecionais de Huebner:
(A) barramento de leitura e (B) barramento de escrita.

B.3. Macros de Dyer

Dyer, Plessl e Platzner propuseram em [DYE02] duas macros para solucionar os problemas de roteamento em SDRs para FPGAs Virtex. A macro denominada "CLB macro simples", apresentada na Fig. 6A, é programada como uma macro de passagem de dados e é utilizada para auxiliar no roteamento de conexões que invadem um determinado módulo. Desta forma é possível posicionar uma ou mais macros de passagem na rota a ser seguida pelo fio, forçando assim o uso de um determinado caminho entre a origem e o destino.

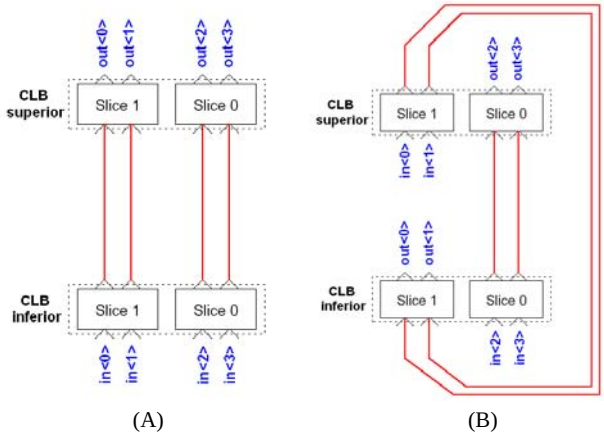

Fig. 6 - Macros de Dyer implementadas para dispositivos Virtex: (A) CLB macro simples e (B) CLB macro dupla.

A macro denominada "CLB macro dupla", apresentada na Fig. 6B, é utilizada para isolar um módulo reconfigurável do restante do sistema. Ela é formada por duas CLBs interconectadas e programadas como passagem

de dados. A CLB macro dupla permite comunicar quatro bits e possui uma rota fixa entre duas CLBs.

B.4. Macros Propostas

Além de fixar a comunicação dos módulos reconfiguráveis nos diferente projetos do SDR, uma das macros propostas neste trabalho possui a função de isolar o restante do sistema enquanto um módulo está sendo substituído. Este isolamento é necessário porque a reconfiguração instantaneamente modifica o módulo, podendo assim ocorrer chaveamentos transitórios em sua saída e por conseqüência interferir no funcionamento do restante do sistema que continua operando. A Fig. 7 apresenta as duas macros propostas neste trabalho para SDRs que possuem a área fixa à esquerda e a área reconfigurável à direita. A macro LR, apresentada na Fig. 7A, possui apenas a função de fixar a comunicação entre a área fixa e a área reconfigurável nos diferentes projetos do SDR. A macro RLS, apresentada na Fig. 7B, possui a função de fixar a comunicação entre a área reconfigurável e a área fixa, e a função de não permitir o envio de dados para a área fixa durante a reconfiguração.

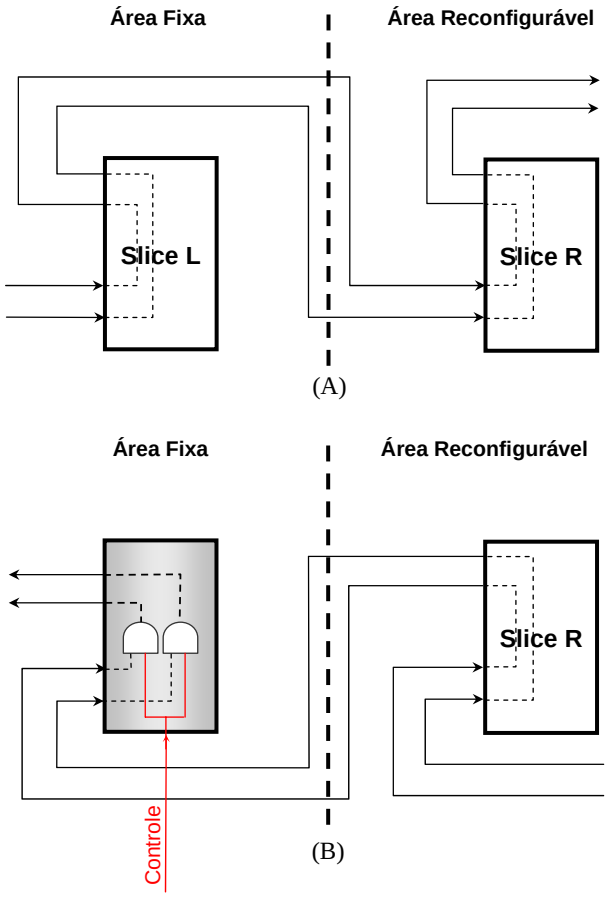

Fig. 7 - Macros propostas para habilitar SDRs apresentadas parcialmente, pois o diagrama de blocos em (A) e (B) deve ser quadruplicado para ocupar uma CLB inteira da Virtex-II. Nestas, a área fixa fica à esquerda e a área reconfigurável à direita. Em (A), a macro LR passa dados da esquerda para a direita. Em (B), a macro RLS passa dados da direita para esquerda apenas quando o módulo não estiver sendo reconfigurado.

C. Geração de bitstreams parciais

O processo de geração de *bitstreams* é um processo específico até mesmo entre dispositivos distintos de uma mesma família de um mesmo fabricante. Com freqüência, apenas o próprio fabricante desenvolve ferramentas para a geração de bitstreams parciais, pois para efetuar este passo é necessário conhecer em profundidade a organização do bitstream e sua relação com a arquitetura de reconfiguração do dispositivo. Como os fabricantes em geral não têm o interesse de divulgar detalhes sobre estas arquiteturas, apenas eles possuem informações suficientes para prover ferramentas capazes de configurar parcialmente os seus dispositivos. Outro motivo que inibe os fabricantes de dispositivos reconfiguráveis de fornecer mais informações de suas arquiteturas é que seja feita engenharia reversa de algum projeto, podendo assim colocar em risco a propriedade intelectual de produtos dos clientes. A seguir são apresentados alguns programas que permitem gerar bitstreams parciais para dispositivos Virtex-II.

C.1. BitGen

A ferramenta BitGen é uma ferramenta que opera por linha de comando e permite gerar bitstreams parciais e totais a partir de arquivos NCD (*Native Circuit Description* – Descrição Nativa do Circuito). Um bitstream total pode ser gerado a partir da linha de comando **`bitgen arquivo.ncd`**, onde **`arquivo.ncd`** é o nome do projeto gerado pela ferramenta de síntese física ou modificado pela ferramenta FPGA Editor da Xilinx.

Bitstreams parciais com pequenas modificações podem ser gerados a partir do comando **`bitgen -g ActiveReconfig:Yes -g persist:yes -r in.bit mod.ncd par.bit`** que permite gerar um bitstream parcial (**`par.bit`**) a partir de um bitstream inicial (**`in.bit`**) e o projeto inicial com alguma modificação (**`mod.ncd`**) feita através do FPGA Editor. O parâmetro **`ActiveReconfig:Yes`** informa que uma reconfiguração parcial está em andamento e por isso não se deve modificar os valores dos flip flops. O parâmetro **`persist:yes`** é utilizado para manter os pinos de configuração externos/internos do FPGA após a configuração atual, assim habilitando futuras reconfigurações.

Quando o fluxo de projeto modular é utilizado, bitstreams parciais de módulos são gerados a partir do comando **`bitgen -f param.ut top.ncd`**, onde **`top.ncd`** é o módulo já roteado e **`param.ut`** é um arquivo com parâmetros a serem utilizados pela ferramenta BitGen. Para que o bitstream parcial **`top.bit`** seja corretamente gerado, é necessário que o arquivo **`top.ncd`** seja sintetizado com alguns parâmetros adicionais.

Também existe uma forma pouco conhecida para a geração de bitstreams parciais, através do parâmetro **`-g PartialMask`**. Este parâmetro aceita uma máscara que informa quais colunas do FPGA devem ser reconfiguradas. Mais informações sobre este parâmetro não podem ser disponibilizados devido a um termo de cooperação de pesquisa assinado com regras de sigilo (*Non Disclosure*

Agreement - NDA) entre o GAPH e a Xilinx.

C.2.JBits

JBits [XIL05b] é um conjunto de classes Java que provê ao usuário acesso em alto nível de abstração a bitstreams da família Virtex e Virtex-II. Atualmente a versão 3 do JBits provê suporte apenas aos bitstreams da família Virtex-II, enquanto as versões anteriores provêm acesso apenas à família Virtex.

Entre as principais características do JBits pode-se citar a leitura e a escrita de valores de componentes internos como LUTs, CLBs, BRAMs e IOBs, e a geração de bitstreams parciais ou totais com as modificações efetuadas. Uma vantagem do JBits é a sua velocidade de geração e modificação de bitstreams quando comparado à execução de síntese para a geração de um bitstream total.

Uma abordagem utilizada por Huebner [HUB05] para gerar bitstreams parciais a partir do JBits é marcar os frames de um bitstream total como modificados e gerar um bitstream parcial a partir deste. A Fig. 8 apresenta o trecho de código comentado em linguagem Java utilizando a API JBits 3 responsável por gerar bitstreams parciais segundo este método.

```
01 //capturando qtde de linhas e colunas
02 int rows = device.getTileRows();
03 int cols = device.getTileCols();
04
05 //instanciando uma matriz de recursos
06 int[][] res = new int[rows][cols];
07
08 //cria a instancia de um bitstream parcial
09 byte[] partial = jbits.generatePartial();
10
11 //percorre todos os frames de um CLB
12 for (int x=startColumn; x<endColumn; x++) {
13    for (int y=0; y<22; y++) {
14       //atribuindo um valor para o MinorFrame
15       res[0][1] = y;
16
17       //captura a primeira linha do frame
18       int value[] = jbits.getCLBBits(0,x,res);
19
20       //escreve o valor lido no frame
21       jbits.setCLBBits(0,x,res,value);
22    }
23 }
24 //habilita o calculo de CRC
25 jbits.enableCrc(true);
26
27 //gera o bitstream parcial
28 jbits.writePartial(outFileName);
```

Fig. 8 - Trecho de código Java/JBits para geração de bitstreams parciais de módulos.

C.3. CoreUnifier

O CoreUnifier [MES03] [MOR03b] é uma ferramenta gráfica desenvolvida pelo Autor em linguagem Java, e possui o objetivo de gerar bitstreams parciais a partir da seleção de áreas de bitstreams totais. Ela foi implementada a partir das equações de endereçamento do dispositivo Virtex disponibilizadas no Application Note 151 [XIL04a].

O CoreUnifier-II, desenvolvido para a geração de bitstreams parciais do dispositivo Virtex-II, também foi implementado em linguagem Java e ainda não possui uma versão gráfica do mesmo. Ele foi construído com base no conhecimento obtido do CoreUnifier e de algumas informações adicionais obtidas via NDA com a Xilinx. Tais informações foram importantes para o seu desenvolvimento porque a organização do bitstream em relação a sua arquitetura não foi aberta ao público, como feito para o dispositivo Virtex.

A versão 050408 do CoreUnifier-II pode ser utilizada com os seguintes parâmetros:

- **`-p prototipo.txt`**: utiliza o arquivo **`prototipo.txt`** como protótipo para a construção de bitstreams parciais.
- **`-o parcial.rbt`**: cria o bitstream parcial **`parcial.rbt`** segundo o protótipo de geração de bitstream parcial.
- **`-c comentado.rbt`**: gera o arquivo **`comentado.rbt`**, descrevendo linha a linha o bitstream parcial ou total passado como entrada.
- **`-t saida.rbt`**: corrige o CRC de bitstreams com erro e salva como **`saida.rbt`**.

V.REDE INTRA-CHIP PARA SDRS

O trabalho de Marescaux [MAR04], apresentado na Seção II, possui três redes distintas: uma para tráfego de dados, uma para controle e uma exclusiva para reconfiguração. Por um lado, três redes distintas garantem uma melhor qualidade de comunicação e menor latência. Por outro lado, a sobreposição de redes de comunicação aumenta o número de fios entre os módulos do sistema, conseqüentemente aumentando a complexidade para a conexão de módulos reconfiguráveis e a área para o controle de cada uma destas redes em cada uma das interfaces dos módulos.

A proposta deste trabalho é utilizar a própria rede intra-chip de comunicação de dados para controlar também a reconfiguração. Para a implementação desta, a rede Hermes é utilizada como base, sendo apresentada na Seção V.A. Na seqüência, duas políticas de acesso a módulos reconfiguráveis são analisadas na Seção V.B. As modificações necessárias na rede Hermes para a conexão de módulos reconfiguráveis são apresentadas na Seção V.C. Essa rede intra-chip modificada para permitir a substituição de módulos de hardware em tempo de execução foi chamada de Artemis. A Seção V.D apresenta como as macros propostas na Seção IV são conectadas na interface da rede Artemis com um módulo reconfigurável. Por último, a Seção V.E apresenta a validação por simulação funcional da Artemis.

A. Rede Hermes

A topologia de uma NoC é definida pela estrutura de conexão dos seus roteadores. Na topologia malha utilizada neste trabalho, roteadores distintos podem possuir um número de portas diferentes, dependendo de sua posição na rede, como mostrado na Fig. 9C. Por exemplo, o roteador central de uma NoC Hermes 3x3 possui cinco portas,

conforme apresentado na Fig. 9B. As cinco 5 portas bidirecionais do roteador são: *East*, *West*, *North*, *South* e *Local*. Cada porta possui uma fila de entrada para o armazenamento temporário de *flits*. A porta *Local* estabelece a comunicação entre o roteador e seu módulo local. As demais portas ligam o roteador aos roteadores vizinhos. O modo de chaveamento adotado pelo roteador da NoC Hermes é *wormhole*, onde cada pacote que trafega pela rede é passado *flit* a *flit* pelos canais físicos. O formato dos pacotes que trafegam pela rede é apresentado na Fig. 9A. A lógica de controle implementa a *lógica de arbitragem* e o *algoritmo de chaveamento por pacotes*.

Um roteador pode ser requisitado a estabelecer até 5 conexões simultâneas (uma conexão por porta de entrada do roteador). A *lógica de arbitragem* é utilizada para escolher qual das portas de entrada será roteada primeiro. Para isto, é utilizado um algoritmo de arbitragem dinâmica rotativa (round-robin), onde a prioridade de uma determinada porta é dada em função da última porta a ter a requisição de arbitragem atendida. Este algoritmo previne que uma mesma porta de entrada tenha sempre a maior prioridade para requisitar uma porta de saída, não permitindo acesso das outras portas de entrada às portas de saída.

O *algoritmo de chaveamento por pacotes* utilizado pelo roteador é o XY determinístico, no qual os pacotes trafegam pela rede sempre em uma mesma ordem: (*i*) trafegam pela rede no eixo X até chegar na coluna do roteador destino; (*ii*) trafegam pela rede no eixo Y até chegar na linha do roteador destino; (*iii*) estando no roteador destino, o pacote é encaminhado para a porta local para alcançar o módulo destino.

Quando um *flit* é bloqueado em um determinado roteador, o desempenho da rede é afetado, pois os *flits* do mesmo pacote podem ser bloqueados em roteadores intermediários da rede dependendo dos tamanhos do pacote e das filas. Para diminuir esta perda de desempenho, uma fila circular é adicionada a cada porta de entrada do roteador, reduzindo o número de roteadores afetados pelos *flits* bloqueados. O tamanho da fila é parametrizável em tempo de projeto. Mais detalhes sobre a NoC Hermes podem ser obtidos em [MOR04].

B. Políticas de Acesso a Módulos Reconfiguráveis

A posição de um dado módulo reconfigurável na NoC pode variar ao longo da execução do sistema. Isto ocorre porque estes saem do sistema para dar espaço a outros e até mesmo para melhorar o desempenho do sistema como um todo (i.e. reposicionar módulos origem e/ou destino de forma que fiquem mais próximo quando estes se comunicam com bastante freqüência ou quando transferem grande quantidade de informações). Para isso é necessária uma entidade que gerencie a qual roteador um módulo reconfigurável vai ser conectado e que módulo reconfigurável pode ser retirado para a entrada de outro. A esta entidade dá-se o nome de Controlador de Configurações (CC).

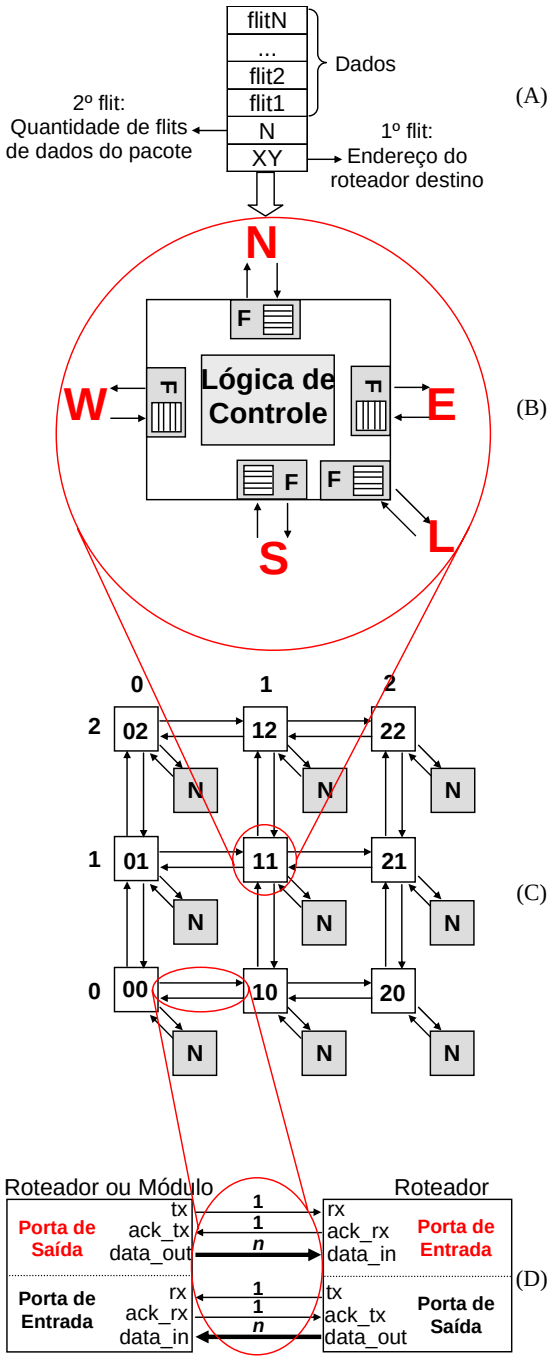

Fig. 9 – Estrutura geral da NoC Hermes: (a) formato dos pacotes que trafegam pelos roteadores; (b) diagrama de blocos do roteador; (c) exemplo de NoC Hermes 3x3. (d) interface roteador-roteador ou módulo-roteador.

Dois CCs foram propostos como trabalhos acadêmicos [BLO03] [CUR04], tendo sido estes analisados por Carvalho [CAR04]. Carvalho implementou um CC completamente em hardware chamado RSCM. Encontra-se em andamento a implementação de um CC em software [SOA05] para que seja feita a análise dos prós e contras em relação ao trabalho de Carvalho. Não pertence ao escopo deste trabalho conectar um CC ao SDR proposto, ficando esta atividade como uma sugestão de trabalho futuro. No

entanto, este trabalho se preocupa em prover um suporte de hardware e de comunicação entre os módulos reconfiguráveis e o CC.

Logo, para efetuar um acesso a um módulo reconfigurável da rede deve-se acessar antes o CC para se descobrir onde o módulo está configurado. A Fig. 10 apresenta duas políticas de acesso ao CC para posterior acesso ao módulo reconfigurável.

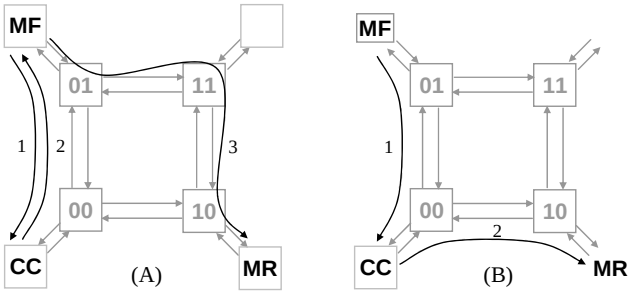

Fig. 10 – Dois exemplos de políticas de acesso a módulos reconfiguráveis. MF representa um módulo fixo, MR representa um módulo reconfigurável e CC é o controlador de configurações. Em (a) o CC é acessado apenas para descobrir o endereço de MR, em (b) o CC é acessado toda vez que MR for acessado.

Os passos sugeridos na Fig. 10A são descritos a seguir:

1. MF, que deseja acessar MR, acessa CC para obter o endereço de MR.
2. CC identifica se MR realmente se encontra configurado ou se o mesmo deve ser configurado e depois responde de volta para MF o endereço de MR.
3. MF acessa diretamente MR.

O passo 3 se repete para cada acesso de MF a MR.

A seguir são descritos os passos sugeridos na Fig. 10B:

1. MF envia pacote destinado a MR para CC enviá-lo.
2. CC identifica se MR realmente se encontra configurado ou se o mesmo deve ser configurado e depois encaminha o pacote de MF para MR.

A política descrita na Fig. 10B é a mais simples de ser implementada, pois quando deseja-se acessar um MR, o pacote é enviado para CC e este gerencia qual é o endereço que corresponde ao MR. No entanto, esta política possui uma latência maior, pois CC é sempre acessado antes de acessar um MR. A necessidade de sempre acessar o CC antes de acessar um MR pode criar também um gargalo no sistema quando existirem muitos MRs.

A política descrita na Fig. 10A é a que mais se aproxima à política de comunicação já implementada na NoC, pois uma vez que o endereço de destino do pacote é fornecido pelo CC, os acessos a MRs são semelhantes aos acessos a MFs. Esta política permite efetuar a comunicação com MRs com uma baixa latência de comunicação, já que não é necessário acessar o CC sempre antes de acessar o MR. Em contraponto, cada módulo conectado ao meio de comunicação deve possuir uma tabela local que relacione o endereço dos MRs utilizados pelo módulo. Esta é uma tabela limitada, contendo apenas um subconjunto de registros da tabela total do CC. O número de registros existentes na tabela de CC é igual ao número de MRs do sistema, enquanto a tabela existente em cada módulo é parametrizável. A vantagem desta implementação é que um módulo pode acessar mais de um MR sem consultar o CC. A desvantagem é que criar tabelas nos módulos que podem acessar MRs pode trazer, além de um aumento significativo em área, a necessidade de controle da validade do endereço de MR.

C. Modificações na rede Hermes

Com o objetivo de suportar o controle da reconfiguração pela mesma rede intra-chip de dados, quatro modificações na rede Hermes são necessárias: (1) descarte de pacotes enviados para módulos reconfiguráveis durante a reconfiguração do módulo; (2) criação de pacotes de controle; (3) inserção de dois sinais de controle para cada porta de cada roteador; (4) inserção de lógica nas portas de entrada dos roteadores para recepção/repasse de pacotes de controle.

C.1. Descarte de pacotes

Independente da política de acesso a módulos reconfiguráveis utilizada, deve-se garantir que nenhuma modificação feita no sistema para suportar reconfiguração parcial e dinâmica de módulos leve a NoC a um estado de bloqueio irrecuperável. O caso citado na Fig. 10A é um destes exemplos. O problema ocorre quando MR está sendo reconfigurado e um pacote chega de MF no roteador de MR (passo 3 da Fig. 10A), ficando este bloqueado porque o hardware do módulo reconfigurável ainda não está presente para receber os *flits* do pacote enviado. Neste caso duas questões devem ser analisadas: (1) este pacote foi destinado ao módulo que existia antes da reconfiguração ou ao módulo que está sendo configurado? (2) o que fazer com o pacote recebido durante o processo de reconfiguração?

Um fato preocupante é que durante o processo de reconfiguração o pacote em questão fica aguardando na fila do roteador do módulo reconfigurável e, dependendo do tamanho do pacote, em outros roteadores intermediários da NoC. Devido a este motivo e à dúvida de não saber se o pacote foi destinado ao módulo que estava previamente configurado ou ao módulo que está sendo configurado, decidiu-se descartar os pacotes recebidos pelo roteador de um módulo reconfigurável durante o processo de configuração do módulo. Fica a cargo da política de comunicação evitar que pacotes não sejam descartados pelo roteador do módulo que está sendo reconfigurado, ou tratar deste caso usando e.g. uma estratégia de reenvio.

Para que o roteador de um módulo reconfigurável possa descartar pacotes enviados durante a reconfiguração do módulo é necessário que o roteador tome conhecimento que uma reconfiguração ocorrerá. Isto é feito na rede Artemis através de *pacotes de controle*.

C.2. Pacotes de controle

Para que a rede Artemis possa garantir o correto

funcionamento da NoC em SDRs, pacotes de controle foram criados para informar os roteadores de módulos reconfiguráveis se o seu módulo local está sendo reconfigurado. Estes pacotes de controle são identificados por sinais extras nas entradas dos roteadores, detalhados na Seção seguinte. Dois serviços habilitados pela chegada de pacotes de controle foram adicionados aos roteadores de módulos reconfiguráveis da Artemis: os serviços de estabelecimento e interrupção de comunicação entre o roteador e o módulo reconfigurável. A Fig. 11A e a Fig. 11B apresentam, respectivamente, o pacote de estabelecimento e o pacote de interrupção da comunicação entre o roteador e seu módulo reconfigurável.

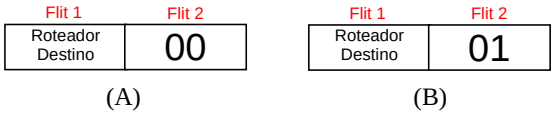

Fig. 11 – Pacotes de controle da rede Artemis: (a) estabelecimento da comunicação entre roteador e módulo reconfigurável; (b) interrupção da comunicação entre roteador e módulo reconfigurável.

Os pacotes de controle possuem apenas dois *flits*. O primeiro é o destino do pacote e o segundo é a operação que o roteador deve executar. Com esta estrutura de pacotes de controle a NoC pode possuir até $2^{(\text{tamanho do flit em bits})}$ serviços disponibilizados pelos roteadores.

C.3. Inserção de sinais nas portas dos roteadores

Foi adicionado nas portas dos roteadores da rede Artemis um sinal para diferenciar pacotes de controle de pacotes de dados. Desta forma, a cada *flit* enviado pela NoC através dos sinais *data_out* e *tx*, adiciona-se o sinal *ctrl_out*, o qual informa se o *flit* é de controle. O roteador destino do *flit* recebe *flits* de forma análoga pelos sinais *data_in*, *rx* e *ctrl_in*. A Fig. 12 apresenta a interface entre dois roteadores da rede intra-chip Artemis.

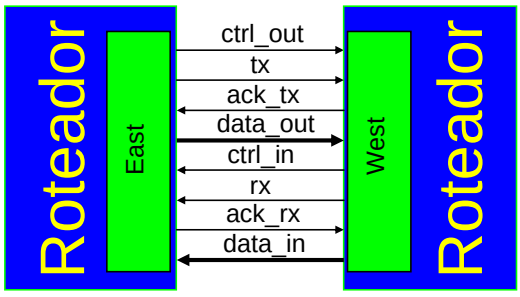

Fig. 12 – Interface entre as portas East e West de dois roteadores da rede intra-chip Artemis.

C.4. Inserção de lógica nas portas dos roteadores

Para que pacotes de controle cheguem ao roteador destino é necessário inserir lógica em cada uma das portas dos roteadores da NoC para repassar ou capturar os pacotes de controle. A primeira tarefa do roteador é verificar o destino (primeiro *flit*) de um pacote recebido e se este é de controle (*ctrl_in* = '1'). Caso o destino do pacote de controle seja igual ao endereço do roteador atual então este pacote será capturado pelo roteador atual. A chegada do segundo *flit* do pacote informa se será estabelecida (*data_in* = x"00") ou interrompida (*data_in* = x"01") a comunicação do roteador com o módulo reconfigurável. Caso o destino do pacote seja um endereço diferente do roteador atual, então este *flit* deve entrar na fila pré-existente (*fila de dados*) e uma segunda fila criada (*fila de controle*) deve armazenar no mesmo índice da fila de dados a informação de que aquele dado é de controle (*ctrl_in* = '1'). A fila de controle possui o mesmo tamanho da fila de dados, largura um bit e sempre opera com o mesmo índice da fila de dados. O segundo *flit* deste pacote, contendo a operação a ser efetuada pelo roteador destino, é igualmente inserido na fila de dados, inserindo também o valor recebido de *ctrl_in* na fila de controle. Uma vez que um pacote de controle se encontra na fila do roteador, o processo de repasse é feito de forma semelhante aos pacotes de dados, desde que levado em consideração o envio de *ctrl_out* com o valor armazenado na fila de controle para cada *flit* enviado.

D. Interface entre Roteador e Módulo Reconfigurável

A interface entre o roteador da Artemis e o módulo reconfigurável é apresentado na Fig. 13. Foram necessárias duas macros LR para conectar os 11 bits da interface do roteador com o módulo reconfigurável e duas macros RLS para conectar os outros 10 bits de interface no sentido inverso de comunicação.

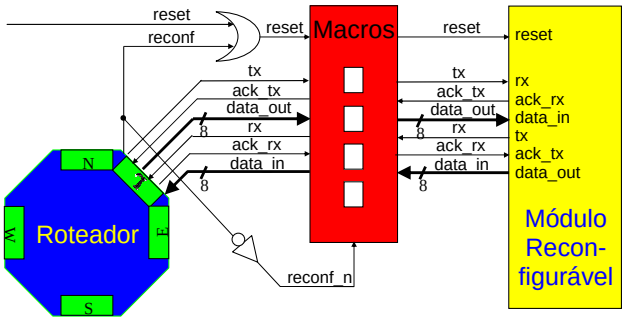

Fig. 13 – Interface da rede Artemis com um módulo reconfigurável passando através das macros propostas. No estudo de caso implementado duas macros LR e duas macros RLS foram utilizadas para comunicar os 21 bits de interface do módulo reconfigurável com a Artemis.

Na interface entre o roteador e um módulo reconfigurável percebe-se que os sinais *reconf* e *reset* foram adicionados e os sinais *ctrl_in* e *ctrl_out* foram suprimidos, quando comparado à interface roteador-roteador apresentada na Fig. 12. Os sinais *ctrl_in* e *ctrl_out* foram suprimidos da interface entre o roteador e o módulo reconfigurável porque módulos reconfiguráveis não enviam nem recebem pacotes de controle. O *reset* é um sinal global que deve ser conectado apenas aos módulos do sistema, sendo utilizado para colocar as máquinas de estados dos módulos no estado inicial. O sinal *reconf* é utilizado assim como *reset* para inicializar as máquinas de

estados, com a exceção que o sinal de *reconf* inicializa apenas o módulo local do roteador a ele conectado. O sinal de *reconf* negado da Fig. 13 é conectado ao sinal *chave* da macro RLS, conforme apresentado na Fig. 7B, permitindo a passagem de dados do módulo reconfigurável para o roteador apenas quando o módulo não estiver sendo reconfigurado.

E. Validação da Rede Artemis

A rede Artemis foi primeiramente validada através de simulação funcional. A Fig. 14 apresenta, por meio de formas de onda, a validação do processo de interrupção da comunicação com um módulo reconfigurável e o posterior restabelecimento desta comunicação após a reconfiguração. Os passos apresentados a seguir possuem correspondência com a numeração indicada na Fig. 14.

1. Envio de pacote de controle, requisitando a interrupção da comunicação do roteador 10 com o seu módulo reconfigurável. Este pacote foi originado pelo CC, que está conectado na porta local do roteador 00.
2. Chegada do pacote de interrupção de comunicação com o módulo reconfigurável no roteador 10. Roteador ativa o sinal de *reconf* informando à macro que não mais repasse os sinais de saída do módulo reconfigurável para o roteador.
3. Teste de chegada de um pacote de dados durante a reconfiguração. Este pacote é totalmente descartado, conforme discutido na Seção V.C.1. O posterior tratamento de outros pacotes por parte do roteador indica que o mesmo garante que pacotes recebidos durante o processo de reconfiguração não o bloqueiam.
4. Chegada do pacote de restabelecimento de comunicação com o módulo reconfigurável. Roteador desativa sinal de *reconf*, informando à macro que volte a repassar os sinais de saída do módulo reconfigurável para o roteador.
5. Chegada de pacote de dados após a reconfiguração do módulo reconfigurável.
6. Envio de pacote de dados do roteador 10 para o módulo reconfigurável.

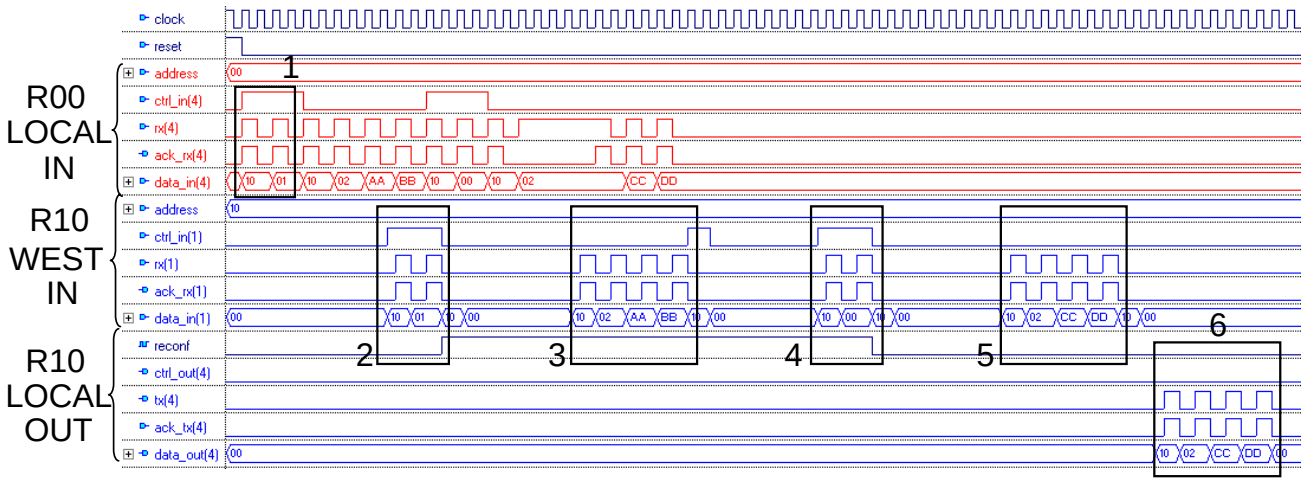

Fig. 14 – Simulação da rede Artemis mostrando o tráfego de pacotes de controle para interromper / restabelecer conexão com os módulos reconfiguráveis e de pacotes de dados repassados ou descartados.

VI. Estudo de Caso

Para validar a rede Artemis conectada a um módulo reconfigurável, o sistema apresentado na Fig. 15 foi prototipado em uma plataforma Memec Insight MB1000 com um dispositivo Virtex-II XC2V1000. O sistema possui conectada a Artemis dois processadores R8 [MOR03a], uma interface RS-232 e um módulo reconfigurável. A interface RS-232 é responsável por comunicar o sistema com um computador hospedeiro e vice-versa. O processador R8 de 16 bits possui 40 instruções, arquitetura *load store* Von Neumann e um banco de registrador 16 x 16 bits. Cada processador utiliza dois blocos de 18 Kbits como memória cache de instruções e dados unificada, provendo um espaço de endereçamento de 2K palavras. Cinco módulos reconfiguráveis foram conectados a Artemis: “mult”, “div”, “sqrt”, “double” e “not”. O módulo “mult” multiplica dois operadores de 16 bits. O módulo “div” divide um operador de 16 bits por outro de 16 bits. O módulo “sqrt” calcula a raiz quadrada de um operador de 32 bits. O módulo “double” multiplica um operador de 16 bits por dois. O módulo “not” inverte todos os bits de um operador de 16 bits.

A tarefa do CC neste sistema é atribuída ao computador hospedeiro, sendo a política de requisição de reconfiguração descrita como segue:

- um processador R8 executa instrução de envio de dados para o computador hospedeiro, informando o identificador do módulo reconfigurável desejado;
- o mesmo processador executa instrução de recepção de dados, ficando bloqueado até que a reconfiguração seja concluída;
- o software SerialAppReconf [MÖL05], que está sendo

executado no computador hospedeiro, recebe a requisição de reconfiguração;

- o SerialAppReconf seleciona uma área reconfigurável para configurar o módulo reconfigurável;
- o SerialAppReconf envia um pacote para interromper a comunicação do roteador com a área reconfigurável selecionada;
- o SerialAppReconf encontra o bitstream específico que configura o módulo reconfigurável na área reconfigurável selecionada;
- o serialAppReconf faz uma chamada de sistema para o software Impact reconfigurar o FPGA com o bitstream parcial selecionado;
- após o final da reconfiguração, o SerialAppReconf envia um pacote para estabelecer a comunicação do roteador com o módulo reconfigurável configurado;
- o SerialAppReconf responde à recepção de dados do processador R8 com o número da área reconfigurável onde o módulo reconfigurável requisitado foi configurado;
- o *wrapper* do processador associa o módulo reconfigurável requisitado ao endereço real do mesmo na rede em uma tabela.

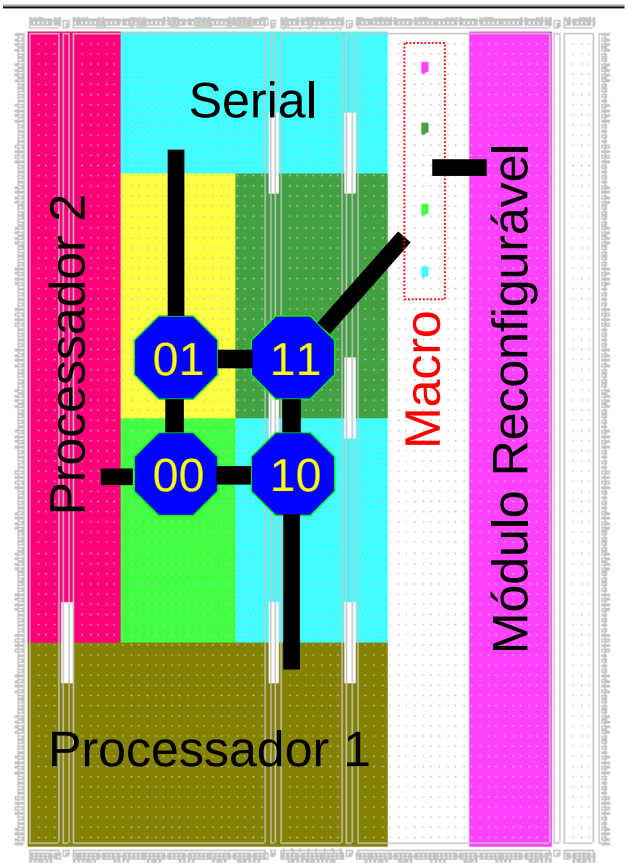

Fig. 15 – Planta baixa do SDR implementado como estudo de caso. O SDR possui dois processadores R8 (a esquerda e abaixo), a interface serial (acima), a NoC Artemis (ao centro), um hardware dedicado reconfigurável a direita e as macros entre o hardware dedicado e a NoC.

A política de escrita de dados em módulos reconfiguráveis é feita da seguinte forma. Primeiro o processador R8 envia dados para o módulo reconfigurável através da instrução *store* no endereço *EPXX* com o valor *YY*, onde *E* é a constante em hexadecimal selecionada para acessar módulos reconfiguráveis, *P* é um valor em hexadecimal para acessar uma das possíveis 16 posições de memória de um módulo reconfigurável, *XX* representa o identificador do módulo reconfigurável a ser acessado e *YY* representa o valor a ser enviado ao módulo reconfigurável. A seguir o *wrapper* pesquisa na tabela qual é o endereço de rede ZZ do módulo reconfigurável identificado por XX. Um pacote de envio de dados é enviado para ZZ com o valor YY. A política de leitura de dados de um módulo reconfigurável é feita de forma análoga à escrita, sendo utilizada a instrução *load* ao invés de *store* e YY não é enviado, mas sim recebido pelo processador R8.

A política de liberação de um módulo reconfigurável XX por parte do processador R8 é feita com a execução da instrução *store* no endereço FFFC com o valor XX. Esta instrução informa ao *wrapper* do processador que este deve enviar um pacote para o computador hospedeiro permitindo que outro módulo reconfigurável utilize a área do módulo XX.

A Tabela I apresenta o resultado de área discriminado para cada módulo do SDR implementado. Embora o módulo reconfigurável que utilize mais LUTs seja o “sqrt” e sua lógica pudesse ser colocada em apenas uma coluna de CLB, o módulo reconfigurável só foi prototipado com sucesso em 8 colunas de CLBs, sendo 2 colunas somente de roteamento e uma reservada para o lado direito das quatro macros utilizadas. O projeto total que utilizou mais LUTs foi o que contém o módulo “sqrt”, ocupando 37,97% (3888/10240) das LUTs disponíveis no FPGA Virtex-II 1000.

TABELA I
RELATÓRIO DE ÁREA DO SDR IMPLEMENTADO

Área	Módulo	LUTs	FFs	BRAM	ASIC mapping	CLB Cols
Fixa	R8	825	313	2	4251	-
	RS-232	403	256	-	2667	-
	NOC	1605	855	-	9163	-
Reconf.	not	75	98	-	859	8
	double	58	95	-	835	8
	mult	164	259	-	2294	8
	div	166	259	-	2297	8
	sqrt	219	269	-	2470	8

VII. ANDAMENTO DAS ATIVIDADES

A Tabela II apresenta o cronograma de atividades para a realização do corrente trabalho. As atividades apresentadas na Tabela II possuem correspondência direta com os itens apresentados a seguir, que discorrem sobre o andamento de cada atividade.

TABELA II
CRONOGRAMA DE ATIVIDADES PARA ANO DE 2005

Atv.	Objetivo	Jan		Fev		Mar		Abr		Mai		Jun		Jul		Ago		Set		Out		Nov		Dez	
1	*CoreUnifier-II*																								
2	*Macro*																								
3	*Escrita Artigo*																								
4	*XDLAnalyzer*																								
5	*Modificações na NoC*																								
6	*Integr. Macro/NoC*																								
7	*Seminário Andamento*																								
8	*Escrita Artigo*																								
9	*Realocação*																								
10	*Escrita Artigo*																								
11	*Escrita Volume*																								

- *Atv 1 – Entender a organização de bitstreams da família Virtex-II, implementar o cálculo de CRC para esta família de dispositivos e desenvolver a ferramenta CoreUnifier-II.* Atividade **realizada**, a ferramenta CoreUnifier-II se encontra funcional, tendo sido utilizada para a geração de bitstreams parciais dos módulos reconfiguráveis conectados à Artemis. Os bitstreams parciais gerados funcionam conforme esperado na plataforma de prototipação. Espera-se ainda implementar a interface gráfica do CoreUnifier-II, semelhante à desenvolvida na ferramenta CoreUnifier.
- *Atv 2 – Desenvolver macros com as funções de isolar a área que está sendo reconfigurada e prover uma interface de comunicação entre esta e o restante do sistema.* Atividade **realizada** e funcionando conforme esperado. As macros propostas foram apresentadas na Seção IV.B.4 e foram validadas pelo estudo de caso da Seção VI e pelo trabalho de [SOA05].
- *Atv 3 – Escrita de artigo sobre as atv 1 e 2.* Atividade **em andamento**.
- *Atv 4 – Implementar a ferramenta XDLAnalyzer para verificar a invasão de fios em módulos reconfiguráveis*. Atividade **parcialmente realizada**. Uma versão inicial do XDLAnalyzer foi construída fazendo a comparação de fios conectados as macros de dois projetos do mesmo SDR. No entanto, a ferramenta provou ser ineficiente quando fios internos de um módulo (não conectados as macros) invadem a área de outro módulo. Este também é um acontecimento que deve ser detectado pelo XDLAnalyzer porque causa o não funcionamento do SDR. Na próxima versão do XDLAnalyzer apenas um projeto será utilizado como entrada e todos os pontos de interconexão de todos os fios do projeto serão analisados, verificando se dois pontos de interconexão se encontram em áreas reconfiguráveis diferentes. Se isto ocorrer o SDR só funcionará se todos os projetos tiverem estes dois pontos de interconexão no mesmo lugar.
- *Atv 5 – Modificar o roteador da NoC Hermes para estabelecer/interromper comunicação com o módulo reconfigurável.* Atividade **realizada**. As modificações no roteador da NoC Hermes, apresentadas na Seção V.C, foram validadas na Seção V.E. e funcionam na plataforma de prototipação conforme apresentado no estudo de caso da Seção VI. Estas modificações no roteador da NoC Hermes deram origem à NoC Artemis.
- *Atv 6 – Conectar o roteador da NoC Artemis com as macros propostas*. Atividade **realizada**. A conexão das macros propostas neste trabalho com o roteador da NoC Artemis foi explicada na Seção V.D. A correta execução do SDR apresentado na Seção VI ilustra que a conexão foi implementada com sucesso.
- *Atv 8 – Escrita de artigo sobre as atv 5 e 6.* Atividade Atividade **a ser realizada**.
- *Atv 9 – Realocação de módulos reconfiguráveis.* Atividade **a ser realizada**.
- *Atv 10 – Escrita de artigo sobre a atv 9.* Atividade Atividade **a ser realizada**.
- *Atv 11 – Escrita do volume*. Atividade **a ser realizada**.

VIII. CONCLUSÕES

A utilização de SDRs para a implementação de equipamentos eletrônicos apresenta as seguintes vantagens sobre a implementação onde o hardware é uma entidade fixa: (i) apenas os módulos necessários à execução necessitam estar configurados, reduzindo o consumo de potência; (ii) menor área total do sistema implementado, reduzindo custo; (iii) possibilidade de atualização sem alteração do produto ou intervenção do usuário, aumentado a vida útil dos sistemas implementados com esta técnica.

Apesar destas vantagens, há fatores que limitam a implementação de SDRs: tecnologia, fluxo de projeto e comunicação. O fator tecnologia está associado aos atuais FPGAs, os quais possuem arquitetura limitada para a implementação de SDRs, e elevado tempo de reconfiguração. O fluxo de projeto presente nas ferramentas de CAD não provê um método suficientemente automatizado, que possibilite a projetistas não especialistas a implementação de SDRs. O fator comunicação implica como a parte fixa de um dado sistema comunica-se com as partes reconfiguráveis. O

presente trabalho contribuiu justamente em dois destes três fatores limitantes para a implementação de SDRs: *comunicação* e *fluxo de projeto*.

A *comunicação* é feita através da rede Artemis: uma plataforma multiprocessada em um único chip que provê flexibilidade para tratar uma vasta gama de problemas em paralelo e baixa latência entre processadores por estes estarem fisicamente próximos. Esta plataforma pode ser ainda especializada em tempo de projeto para obter melhor desempenho, substituindo processadores por módulos de hardware que tratem o problema específico a ser resolvido.

A segunda contribuição deste trabalho foi à proposta de um novo *fluxo de projeto* para SDRs. A principal vantagem deste é utilizar métodos de implementação que não sejam tão propensos a erros quanto o Fluxo de Projeto Modular da Xilinx. Já se encontram funcionais neste fluxo a ferramenta de geração de bitstreams parciais CoreUnifier-II e as macros que permitem a comunicação com módulos reconfiguráveis.

O fluxo de projeto proposto e a rede Artemis foram validados em FPGA através do correto funcionamento do estudo de caso apresentado na Seção VI. Este estudo de caso também mostrou que sistemas que possuem NoCs como meio de comunicação podem ter seus módulos substituídos enquanto o restante do sistema permanece executando normalmente.

AGRADECIMENTOS

Os autores gostariam de agradecer a colaboração e dedicação de Daniel Camozzato, Ismael Grehs, Alzemiro Henrique e Aline Abreu no desenvolvimento deste trabalho.

REFERÊNCIAS

[AMD05] Advanced Micro Devices, Inc. “Multi-Core”. Capturado em: http://multi-core.amd.com, Jul. 2005.

[ATH93] Athanas, P.; Silverman, H; “Processor Reconfiguration through Instruction-Set Metamorphosis”. Computer, vol. 26 (3), Mar. 1993, pp.11-18.

[ATM05] Atmel Corporation. “Atmel Corporation” Capturado em: http://www.atmel.com, Jul. 2005.

[BEC00] Becker, J.; Piontek, T.; Glesner, M.; “DReAM: A Dynamically Reconfigurable Architecture for Future Mobile Communications Applications”. In: Field Programmable Logic and Applications (FPL), 2000, pp. 312-321.

[BEN02] Benini, L.; De Micheli, G. “Networks on Chips: a New SoC Paradigm”. Computer, vol. 35 (1), Jan. 2002, pp. 70-78.

[BLO03] Blodget, B.; McMillan, S.; Lysaght, P. “A Lightweight Approach for Embedded Reconfiguration of FPGAs”. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2003, pp. 399-400.

[BLO04] Blodget, B.; Bobda, C.; Huebner, M.; Niyonkuru, A. “Partial and Dynamically Reconfiguration of Xilinx Virtex-II FPGAs”. In: Field Programmable Logic and Applications (FPL), 2004, pp. 801-810.

[CAR04] Carvalho, E. “RSCM – Controlador de Configurações para Sistemas de Hardware Reconfigurável”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2004, 127 p.

[COM00] Compton, K.; Cooley, J.; Knol, S.; Hauck, S. “Configuration Relocation and Defragmentation for Reconfigurable Computing”. In: Field-Programmable Custom Computing Machines (FCCM), 2000, pp. 279-280.

[CUR04] Curd, D. “Partial Reconfiguration of RocketIO Pre-Emphasis and Differential Swing Control Attributes”. Xilinx Application Note 660 (XAPP660 v2.2), 2004, 9 p.

[DAL01] Dally, W.; Towles, B. “Route Packets, not Wires: On-Chip Interconnection Networks”. In: Design Automation Conference (DAC), 2001, pp. 684-689.

[DYE02] Dyer, M.; Plessl, C.; Platzner, M. “Partially Reconfigurable Cores for Xilinx Virtex”. In: Field Programmable Logic and Applications (FPL), 2002, pp. 292-301.

[EBE96] Ebeling, C.; Cronquist, D.; Franklin, P. “RaPiD - Reconfigurable Pipelined Datapath”. In: Field Programmable Logic and Applications (FPL), 1996, pp. 126-135.

[GOL99] Goldstein, S.; Schmit, H.; Moe, M.; Budiu, M.; Cadambi, S.; Taylor, R.; Laufer, R. “PipeRench: a Coprocessor for Streaming Multimedia Acceleration”. In: International Symposium on Computer Architecture (ISCA), 1999, pp. 28-39.

[HAR95] Hartenstein, R.; Kress, R. “A Datapath Synthesis System for the Reconfigurable Datapath Architecture”. In: Asia and South Pacific Design Automation Conference (ASP-DAC), 1995, pp. 479-484.

[HAR97] Hartenstein, R.; Becker, J.; Herz, M.; Nageldinger, U. “An Innovative Platform for Embedded System Design”. In: Architekturen von Rechensystemen (ARCS), 1997, pp. 143-152.

[HAU97a] Hauser, J.; Wawrzynek, J. “Garp: a MIPS Processor with a Reconfigurable Coprocessor”. In: Field-Programmable Custom Computing Machines (FCCM), 1997, pp. 12-21.

[HAU97b] Hauck, S.; Fry, T.; Hosler, M.; Kao, J. “The Chimaera Reconfigurable Functional Unit”. In: Field-Programmable Custom Computing Machines (FCCM), 1997, pp. 87-96.

[HUB04] Huebner, M.; Becker, T.; Becker, J. “Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration”. In: Brazilian Symposium on Integrated Circuits and Systems Design (SBCCI), 2004, pp. 28-32.

[HUB05] Huebner, M. “Comunicação Pessoal”. Capturado em: http://www-itiv.etec.uni-karlsruhe.de/opencms/opencms/en/institute/staff/huebner.html, Jan. 2005.

[INT05] Intel Corporation. “Multi-Core”. Capturado em: http://www.intel.com, Jul. 2005.

[LIM04] Lim, D.; Peattie, M. “Two Flows For Partial Reconfiguration: Module Based or Small Bit Manipulations”. Xilinx Application Note 290 (XAPP290 v1.2), 2004, 28 p.

[MAR04] Marescaux, T.; Nollet, V.; Mignolet, J.; Bartic, A.; Moffat, W.; Avasare, P.; Coene, P.; Verkest, D.; Vernalde, S.; Lauwereins, R. “Run-Time Support for Heterogenous Multitasking on Reconfigurable SoCs”. Integration The VLSI Journal, vol. 38 (1), Oct. 2004, pp. 107-130.

[MAR99] Marshall, A.; Stansfield, T.; Kostarnov, I.; Vuillemin, J.; Hutchings, B. “A Reconfigurable Arithmetic Array for Multimedia Applications”. In: Field Programmable Gate Arrays (FPGA), 1999, pp. 135-143.

[MIR96] Mirsky, E.; DeHon, A. “MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources”. In: Field-Programmable Custom Computing Machines (FCCM), 1996, pp. 157-166.

[MES03] Mesquita, D.; Moraes, F.; Palma, J.; Möller, L.; Calazans, N. “Remote and Partial Reconfiguration of FPGAs: Tools and Trends”. In: Parallel and Distributed Processing Symposium (IPDPS), 2003, pp. 177.

[MÖL04] Möller, L.; Calazans, N.; Moraes, F.; Brião, E.; Carvalho, E.; Camozzato, D. “FiPRe: An Implementation Model to Enable Self-Reconfigurable Applications”. In: Field Programmable Logic and Applications (FPL), 2004, pp. 1042-1046.

[MÖL05] Möller, L. “Möller Tools”. Capturado em: http://www.inf.pucrs.br/~moller/tools, Jul. 2005.

[MOR03a] Moraes F.; Calazans, N. “R8 Processor Architecture and Organization Specification and Design Guidelines”. Capturado em: http://www.inf.pucrs.br/~gaph/Projects/R8/public/R8_arq_spec_eng.pdf, Feb. 2003.

[MOR03b] Moraes, F.; Mesquita, L.; Palma, J.; Möller, L.; Calazans, N.

"Development of a Tool-Set for Remote and Partial Reconfiguration of FPGAs". In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2003, pp. 1122-1123.

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

[OTT00] Otten, R.; Stravers, P. "Challenges in Physical Chip Design". In: Computer Aided Design (ICCAD), 2000, pp. 84-91.

[PAL02] Palma, J.; Mello, A.; Möller, L.; Moraes, F.; Calazans, N. "Core Communication Interface for FPGAs". In: Brazilian Symposium on Integrated Circuits and Systems Design (SBCCI), 2002, pp. 183-188.

[SAS02] Sassatelli, G.; Torres, L.; Benoit, P.; Gil, T.; Diou, C.; Cambon, G.; Galy, J. "Highly Scalable Dynamically Reconfigurable Systolic Ring-Architecture for DSP Applications". In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2002, pp. 553-558.

[SIN98] Singh, H.; Lee, M.; Lu, G.; Kurdahi, F.; Bagherzadeh, N.; Filho, E. "MorphoSys: a Reconfigurable Architecture for Multimedia Applications". In: Brazilian Symposium on Integrated Circuits and Systems Design (SBCCI), 1998. pp. 134-139.

[SOA05] Soares, R. "Controladores de Configuração para Arquiteturas Reconfiguráveis: Características, Comparação e Evolução". Seminário de Andamento, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2005, 15 p.

[WAI97] Waingold, E.; Taylor, M.; Srikrishna, D.; Sarkar, V.; Lee, W.; Lee, V.; Kim, J.; Frank, M.; Finch, P.; Barua, R.; Babb, J.; Amarasinghe, S.; Agarwal, A. "Baring It All to Software: Raw Machines". IEEE Computer, vol. 30 (9), Sep. 1997, pp. 86-93.

[WAL04] Walder, H.; Platzner, M. "A Runtime Environment for Reconfigurable Hardware Operating Systems". In: Field Programmable Logic and Applications (FPL), 2004, pp. 831-835.

[WAZ93] Wazlowski, M.; Agarwal, L.; Lee, T.; Smith, A.; Lam, E.; Athanas, P.; Silverman, H.; Ghosh, S. "PRISM-II Compiler and Architecture". In: Field-Programmable Custom Computing Machines (FCCM), 1993, pp. 9-16.

[WIN01] Wingard, D. "MicroNetwork-Based Integration for SoCs". In: Design Automation Conference (DAC), 2001, pp. 673-677.

[WIR94] Wirthlin, M.; Hutchings, B.; Gilson, K.; "The Nano Processor: a Low Resource Reconfigurable Processor". In: Field-Programmable Custom Computing Machines (FCCM), 1994, pp. 23-30.

[WIR95] Wirthlin, M.; Hutchings, B. "A Dynamic Instruction Set Computer". In: Field-Programmable Custom Computing Machines (FCCM), 1995, pp. 99-107.

[WIT96] Wittig, R.; Chow, P. "OneChip: an FPGA Processor with Reconfigurable Logic". In: Field-Programmable Custom Computing Machines (FCCM), 1996, pp. 126-135.

[XIL04a] Xilinx, Inc. "Virtex Series Configuration Architecture – User Guide". Xilinx Application Note 151 (XAPP151 v1.7), 2004, 45 p.

[XIL04b] Xilinx, Inc. "Development System Reference Guide – Modular Design". Xilinx, 2004, pp. 81-116.

[XIL05a] Xilinx, Inc. "Xilinx: The Programmable Logic Company". Capturado em: http://www.xilinx.com, Jul. 2005.

[XIL05b] Xilinx, Inc. "JBits 3.0 SDK Documentation". Capturado em: http://www.xilinx.com/labs/projects/jbits, Jul. 2005.

[YE03] Ye, T.; Benini, L.; De Micheli, G. "Packetized On-Chip Interconnect Communication Analysis for MPSoC". In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2003, pp. 344-349.