INTERFACE DE COMUNICAÇÃO DE *CORES* EM FPGAS

José Carlos Palma, Aline Vieira de Melo, Ney Calazans, Fernando Gehm Moraes
{jpalma, alinev, calazans, moraes}@inf.pucrs.br

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

ABSTRACT

The use of predesigned and preverified hardware modules, also called IP cores, is an important part of the effort to design and implement complex systems. However, many aspects of IP core manipulation are still to be developed. This paper proposes one solution to one such problem, namely the dynamic interconnection of hard IP cores inside FPGA system on a chip systems. The paper proposes a communication interface that allows exchange IP cores inside an FPGA during circuit operation, through reconfiguration process. The same interface also allows the communication among distinct IP cores to take place.

1 INTRODUÇÃO

Um *core* é um módulo de hardware, digital ou analógico, podendo ser descrito em diferentes níveis de abstração. Estes *cores* são pré-projetados, pré-verificados (por simulação funcional e *back annotation*) e prototipados em hardware pelo menos uma vez. Somente através do re-uso destes componentes é possível atingir às exigências do mercado atual e reduzir a distância entre a quantidade de recursos disponíveis e a produtividade das equipes de projeto. A Tabela 1 resume as características dos *cores*.

O objetivo deste trabalho é possibilitar que usuários busquem módulos de hardware já sintetizados (*hard cores*) na Internet e os utilizem em dispositivos programáveis (FPGAs), de acordo com suas necessidades. Deve, também, possibilitar que se tenha alguns *cores* em execução no dispositivo, comunicando-se uns com os outros, e que determinado *core* possa ser removido, quando seu uso não for mais necessário, para que outro seja inserido em seu lugar.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta barramentos utilizados para a interconexão de *cores* em ASICs. A Seção 3 apresenta os requisitos necessários para que o re-uso de *hard cores* em dispositivos FPGAs comerciais seja possível de ser realizado. A Seção 4 apresenta nossa proposta de interconexão de cores, e a Seção 5 resultados preliminares. Finalmente, na Seção 6 são apresentadas nossas conclusões e propostas de trabalhos futuros.

Tabela 1 - Três classificações de IP cores baseado em cinco critérios.

	Hard core	**Firm core**	**Soft core**
Rigidez	A organização é predefinida.	Combinação de código fonte e netlist dependente de tecnologia.	Apresenta um código fonte comportamental independente da tecnologia.
Modelagem	Modelado como um elemento de biblioteca.	Combinação de blocos sintetizá-veis fixos. Permite com-partilhar recursos com outros cores.	Sintetizável com diversas tecnologias.
Flexibilidade	Não pode ser modificado pelo projetista.	A personalização de funções específicas é de-pendente da tecnologia.	O projeto pode ser modificado e independe da tecnologia.
Previsibilidade	Temporização é garantida.	Caminhos críticos com temporizações fixas.	A temporização não é garantida, podendo não atender os requisitos do projeto.
Proteção da propriedade intelectual	Alta. A descrição normalmente corresponde a um *layout*.	Média	Muito pequena. Código fonte aberto.
Exemplo de Descrição	Bitstream de FPGA Formatos CIF ou GDS2 para layout de CI	EDIF	VHDL, VERILOG

2 INTEGRAÇÃO DE *CORES*

Nos estágios iniciais do projeto de um SoC (*system on a chip*), os cores são projetados com muitas e diferentes interfaces e protocolos de comunicação. Para contornar o problema de integração entre cores, foram criados padrões para estruturas internas ao circuito integrado. Atualmente, existem poucas arquiteturas de barramento publicamente disponíveis para guiar os fabricantes, tais como a CoreConnect [1] da IBM, AMBA [2] da ARM e WISHBONE [3] da Silicore. Estas arquiteturas de barramento são geralmente vinculadas à arquitetura de um processador [4], tal como o PowerPC ou o ARM.

A arquitetura CoreConnect da IBM fornece três barramentos para interconectar cores e lógica personalizável:

- Barramento Local do Processador (*Processor Local Bus - PLB*): usado para interconectar *cores* com alto desempenho, grande largura de banda, tais como o PowerPC, controladores DMA e interfaces de memória externa.
- Barramento Periférico (*On-Chip Peripheral Bus - OPB*): usado para interconectar periféricos que trabalham com baixas taxas de dados, tais como portas seriais, portas paralelas, UARTs (Universal Assynchronous Receiver Transmiter), e outros *cores* com pequena largura de banda.
- Barramento de Registradores de Controle de Dispositivos (*Device Control Register Bus - DCR*): caminho de baixa velocidade, usado para passar configuração e informações de estado entre o *core* processador e outros *cores*.

A arquitetura WISHBONE. [3] é análoga a um barramento de microcomputador, sendo que: (i) oferece uma solução flexível para integração que pode ser facilmente adaptada à uma aplicação específica; (ii) oferece uma variedade de ciclos de acesso ao barramento e de larguras de caminhos de dados para atender a diferentes sistemas; e (iii) permite que os cores sejam projetados por vários fornecedores. É um barramento flexível, simples, e o único completamente aberto atualmente, pois sua criadora, a empresa Silicore Corporation, tornou-o aberto ao domínio público [5].

Ao contrário de arquiteturas como a CoreConnect, o WISHBONE tem uma estrutura simples, composta de um único barramento, como mostra a Figura 1. Um sistema com muitos componentes pode incluir duas interfaces WISHBONE: uma para blocos que exigem alto desempenho e outro para periféricos de baixo desempenho.

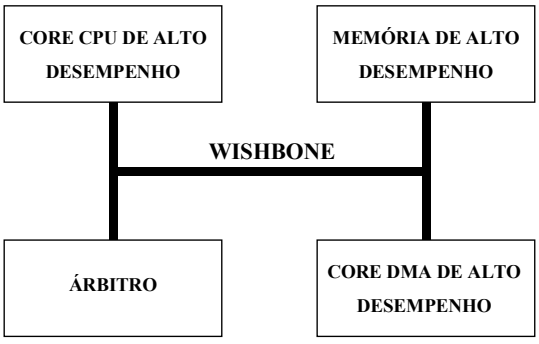

Figura 1 – Arquitetura WISHBONE.

Estas iniciativas de interconexão de *cores* aplicam-se **durante** o projeto dos SoCs. Uma vez o sistema implementado, não há forma de conectar um novo *core* ao sistema, a não ser que seja feita uma nova síntese do hardware.

Não há até o presente momento, ao conhecimento dos autores, uma proposta de interconexão de cores que permita dinamicamente inserir e remover módulos de hardware, *cores*, em tempo de execução. Esta inserção/remoção de *cores*, em tempo de execução, resulta em um equivalente em hardware de sistemas de memória virtual, ou seja, *hardware virtual*.

3 REQUISITOS PARA RE-USO DINÂMICO DE HARD CORES

A motivação para que se distribua (gratuitamente ou não) hard cores é permitir que outros projetistas possam utilizar módulos de hardware já sintetizados, prontos para serem integrados à novos projetos. Não só outros projetistas podem ser beneficiados por esta abordagem, mas sim usuários comuns, mesmo que estes não tenham conhecimento algum sobre FPGAs ou qualquer outro tipo de hardware programável.

Digamos que este usuário tenha em seu computador uma placa aceleradora com um FPGA, e que num determinado momento ele precise de um hardware para acelerar um processo gráfico ou qualquer outra função crítica em tempo de processamento. Este usuário poderia conectar-se à Internet, fazer o download do módulo de hardware que ele deseja, e carregar o módulo no FPGA da sua placa, tendo assim o acelerador que necessita.

Entretanto, a distribuição de propriedade intelectual na forma de módulos já sintetizados (*hard cores*) para FPGAs somente é útil se houver a possibilidade de se "encaixar" novos *cores* em um sistema sendo executado em um FPGA sem interromper ou afetar o funcionamento do mesmo. Para que esta nova abordagem seja factível de ser implementada, é necessário preencher uma série de requisitos:

i) FPGAs com arquitetura regular e disponibilidade de reconfiguração parcial. Alguns FPGAs, como por exemplo os da família *Virtex*, possuem arquitetura interna baseada em colunas, com endereçamento individual. Este fato permite que o dispositivo possa ter sua configuração modificada dinâmica e parcialmente.

ii) Existência de ferramentas que permitam fixar a forma e a posição relativa dos *cores* no dispositivo programável. Exemplo de ferramenta que permite fixar a posição dos circuitos no interior do FPGA é o *Floorplanner* (*Foundation – Xilinx*) Erro! A origem da referência não foi encontrada..

iii) Existência de ferramentas que permitam a geração de *bitstreams* parciais. O conjunto de classes JBITS [7]

permite, teoricamente, gerar um arquivo parcial.

iv) Existência de ferramentas que permitam o *download* de *bitstreams* parciais.

v) Estrutura de conexão entre o *core* inserido e os demais *cores* já em operação no FPGA.

vi) Virtualização dos pinos de entrada e saída.

Os itens (*iv*), (*v*) e (*vi*) carecem hoje de ferramentas. Não se encontra na literatura referência a ferramentas que manipulam arquivos parciais de configuração.

3.1. Arquitetura Virtex

Existem atualmente no mercado duas famílias de FPGAs que suportam reconfiguração parcial: a família At40k [8], desenvolvida pela empresa *Atmel*, e a família *Virtex* [9], desenvolvida pela empresa *Xilinx*. O presente trabalho utilizará a família *Virtex* por esta apresentar uma maior disponibilidade de ferramentas de CAD e capacidade em termos de portas lógicas.

Os FPGAs da família Virtex contém blocos lógicos configuráveis (do inglês *Configurable Logic Block*s - CLBs), blocos de entrada/saída (*Input/Output Blocks* - IOBs), blocos de RAM, recursos de relógio, roteamento programável e configuração do circuito elétrico. Cada CLB possui recursos para o roteamento local e conexão com a matriz de roteamento geral (GRM). Um anel de roteamento periférico, denominado de *VersaRing* permite um roteamento adicional com os blocos de entrada e saída (IOBs). Esta arquitetura apresenta blocos de memória RAM dedicados (BRAMs) de 4096 bits cada um, e conta com 4 a 8 blocos dedicados, que implementam as funções de DLLs para o controle, distribuição e compensação de atrasos do relógio.

A Figura 2 exibe uma abstração da arquitetura interna de um FPGA Virtex, onde podem ser vistos os elementos citados acima.

Figura 2 - Disposição em colunas dos elementos do FPGA Virtex XCV300.

A funcionalidade deste dispositivo é determinada através do arquivo de configuração, denominado *bitstream*. Os dispositivos Virtex têm a arquitetura interna organizada em colunas (Figura 2), podendo estas serem lidas ou escritas individualmente. Logo, é possível reconfigurar parcialmente esses dispositivos através da modificação destas colunas no arquivo de configuração.

Esta característica de reconfiguração parcial permite, em tempo de execução, modificar o comportamento do hardware através do *download* de uma configuração parcial.

3.2. Ferramenta Xilinx Floorplanner

Para que se possa configurar parcialmente um dispositivo FPGA, sem afetar o restante do sistema em funcionamento no mesmo, é preciso que se tenha conhecimento da forma e da área exata ocupada por cada *core*, permitindo assim que eles sejam "encaixados" sem que haja sobreposição dos mesmos. Está área é definida no momento da síntese física do *core*, sendo necessário o uso de uma ferramenta que manipule a lógica contida no dispositivo, delimitando sua disposição no mesmo.

O *Floorplanner* (gerador de planta-baixa) é uma ferramenta gráfica de localização que permite ao projetista controlar a disposição do seu projeto em um dispositivo FPGA usando um paradigma de "arrastar e soltar". A ferramenta utiliza uma representação hierárquica do projeto, usando estruturas e cores para distinguir os diferentes níveis de hierarquia.

O *Floorplanner* é utilizado para definir manualmente restrições de posicionamento. Uma vez a área do *core* restringida, executa-se o posicionamento e o roteamento. As ferramentas de síntese física dos fabricantes de FPGA não provêm nenhum meio de se obter *bitstreams* parciais. Logo, deve-se desenvolver ferramentas para geração de *bitstreams* parciais a partir de *bitstreams* completos.

3.3. JBITS

JBits [7] é um conjunto de classes Java que fornecem uma API (*Application Program Interface*) que permite manipular o *bitstream* da família de FPGAs Virtex. Esta interface opera tanto em *bitstreams* gerados pelas ferramentas de projeto da Xilinx quanto em *bitstreams* lidos do hardware. Isto fornece a capacidade de reconfigurar os circuitos nos dispositivos FPGAs Virtex à partir do arquivo de configuração. O modelo de programação utilizado pelo Jbits é um array bidimensional de CLBs. Cada CLB é referenciada por uma linha e uma coluna. Assim, todos os recursos configuráveis na CLB selecionada podem ser configurados ou analisados.

Os recursos e os valores que podem ser utilizados estão associados à arquitetura Virtex, e somente valores

constantes pré-definidos nas classes Java correspondentes podem ser utilizados para ajustar estes recursos. A Figura 3 mostra como os métodos get( ) e set( ) acessam as LUTs. O parâmetro valor é usado para especificar a lógica a ser implementada pela LUT. O valor é um array de inteiros (somente "0"s e "1"s) representando os 16 bits na LUT.

ESCREVENDO VALORES DAS LUTS
SLICE **0** **F** LUT: jbits.set (linha, coluna, LUT.SLICE0_F, valor)
SLICE **0** **G** LUT: jbits.set (linha, coluna, LUT.SLICE0_G, valor)
SLICE **1** **F** LUT: jbits.set (linha, coluna, LUT.SLICE1_F, valor)
SLICE **1** **G** LUT: jbits.set (linha, coluna, LUT.SLICE1_G, valor)

LENDO VALORES DAS LUTS
SLICE0 **F**LUT: int [] returnVal = jbits.get (linha, coluna, LUT.SLICE0_F)
SLICE0 **G**LUT: int [] returnVal = jbits.get (linha, coluna, LUT.SLICE0_G)
SLICE0 **F**LUT: int [] returnVal = jbits.get (linha, coluna, LUT.SLICE0_F)
SLICE0 **G**LUT: int [] returnVal = jbits.get (linha, coluna, LUT.SLICE1_G)

Figura 3 – Acesso às LUTs do FPGA Virtex.

Este conjunto de classes pode vir a ser utilizado para construir ferramentas de projeto tradicionais, como suporte a posicionamento e roteamento, quanto novas ferramentas para suporte à reconfiguração parcial.

4 PROPOSTA DE INTERFACE DE COMUNICAÇÃO DE *CORES* EM FPGAS

A proposta é implementar no FPGA uma estrutura análoga a uma interface PCI em um PC, onde se conectam dispositivos sem a modificação do sistema. A estrutura também deve ser responsável pela virtualização dos pinos de entrada/saída. Esta virtualização é necessária para que o *core* seja portável, pois os pinos de entrada/saída diferem de um dispositivo para outro. Esta estrutura está relacionada aos itens (*v*) e (*vi*) dos requisitos citados no Seção 3.

Como dito anteriormente, até o presente momento não foi encontrada na literatura referência sobre uma proposta de interface intra-FPGA para que dispositivos reconfiguráveis possam receber cores de forma modular. Essa modularidade consiste em conectar e remover cores sem que haja necessidade de alterações na lógica pré-existente no FPGA. Há técnica semelhante, voltada para o projeto de cores para ASICs. Como exemplo podem ser citados o Amba, o CoreConnect e o WISHBONE, já mencionados.

O ponto de partida para a definição da interface de comunicação é estabelecer uma porção de hardware fixo (estático) no FPGA, chamado de controlador. O controlador é responsável pela comunicação com o mundo externo (pinos de entrada/saída do dispositivo) e pela comunicação com os *cores*. Isto significa que os módulos de hardware a serem conectados, os *cores*, somente comunicar-se-ão com o mundo externo através desta interface.

O FPGA deve ser inicialmente carregado apenas com o controlador. Os *cores* são carregados em tempo de execução, ou seja, em nosso hardware ter-se-á apenas os módulos que precisam ser executados naquele momento. Os demais estão armazenados externamente, para uso posterior.

A comunicação entre o controlador e os demais *cores* é feita através de pinos virtuais. Estes pinos virtuais são implementados através de *buffers tristate*, presentes na arquitetura de FPGAs da família Virtex.

A proposta inicial utilizaria *buffers* na fronteira entre o controlador e o *core*, servindo de conector entre os mesmos. Estes *buffers* fariam parte tanto da descrição do controlador quanto da descrição do *core*. Haveria, assim, uma sobreposição destes *buffers* no momento da reconfiguração parcial do FPGA, como mostra a Figura 4.

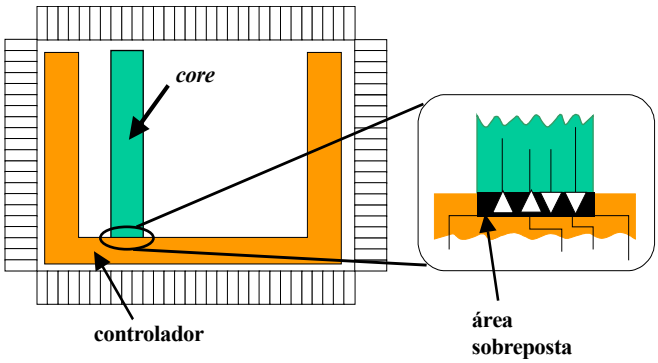

Figura 4 – Interface de comunicação entre *cores*.

Esta proposta não é factível de ser realizada. Isto deve-se a fato das CLBs compartilharem LUTs, *flip-flops* e roteamento. Logo, uma dada CLB do controlador que faria a interface com o *core*, conteria também recursos adicionais de lógica, os quais seriam destruídos no momento da inserção do *core*. As ferramentas de síntese não possuem recursos para proibir o uso de roteamento em uma área.

Visando contornar este problema, adotou-se como solução a utilização de duas camadas de *buffers*, uma camada pertencendo ao controlador e outra pertencendo ao *core*, como mostra a Figura 5.

Devido às limitações impostas no número de *buffers tri-states* existentes no FPGA Virtex, adotou-se largura de barramento igual a 1. É importante lembrar que o objetivo do nosso trabalho é mostrar que é possível utilizar reconfiguração dinâmica de *hard cores* em FPGAs comerciais, não propondo uma nova arquitetura de FPGAs.

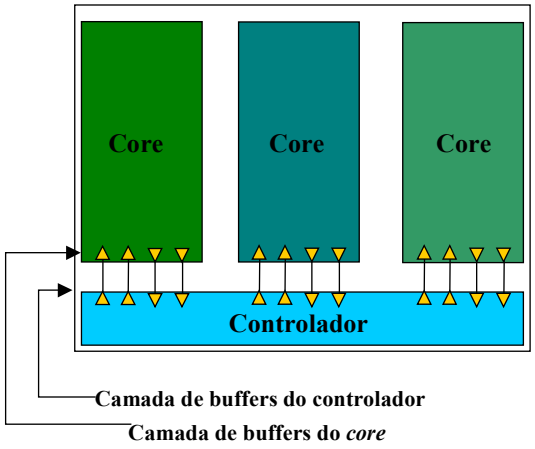

Figura 5 – Implementação com duas camadas de *buffers*.

O controlador da interface deve ser o árbitro deste barramento, gerenciando o aceite ou a rejeição de conexões, bem como o controle de conflitos relativos ao acesso aos pinos de entrada/saída do FPGA. Ainda no controlador, é definido um *core* mestre, também estático, que controla a interface com o mundo externo. O *core* mestre pode ser modificado, de acordo com a aplicação utilizada.

A Figura 6 ilustra a utilização da interface de comunicação, visando sua validação. Esta aplicação é composta por 3 módulos: (1) controlador, composto pelo mestre, árbitro e *buffers* de comunicação; (2) *core* escravo número 1; (3) *core* escravo número 2. A interface com o mundo externo é feita na parte fixa do FPGA, ou seja, pelo controlador. Esta interface é composta por um barramento de 32 bits (*display*), e os sinais *clock*, *start*, *reset*.

A comunicação entre o controlador e cada core é feita por 7 sinais: (1-2) *data in* e *data out*, responsáveis pelo tráfego de dado; (3-4) *request* e *grant*, responsáveis pela solicitação de envio de dados e permissão para enviá-los; (5) *start*, que indica o início de uma transmissão na linha de dados; (6-7) *clock* e *reset*.

Quando um *core* deseja enviar dados para outro módulo do sistema, o sinal "request" é enviado pelo *core,* e este fica aguardando o sinal "grant" O árbitro recebe os sinais de "request" dos *cores* e atende aos pedidos de acesso ao barramento (um de cada vez) enviando o sinal "grant" para o *core* que tem permissão de acesso naquele momento. Este tipo de controle permite que haja *cores* com prioridade mais alta que outros e, ao mesmo tempo, garante que nenhum *core* com alta prioridade seja atendido mais de uma vez, enquanto um de baixa prioridade fica em estado de espera.

Quando o árbitro envia o sinal "grant" a um core, o sinal que habilita o *buffer tristate* de escrita na linha de transmissão ("dataOut") é ativado apenas para este *core*. Neste momento, o sinal "start" é enviado a todos os *cores*, para avisar que um novo dado será transmitido. A transmissão dura exatamente 40 ciclos de clock. Definiu-se que o pacote de dados é composto de 40 bits, sendo 8 para endereçar *cores*, e 32 para dados. Após estes 40 ciclos, o árbitro volta a verificar a existência de outro sinal "request" ativo.

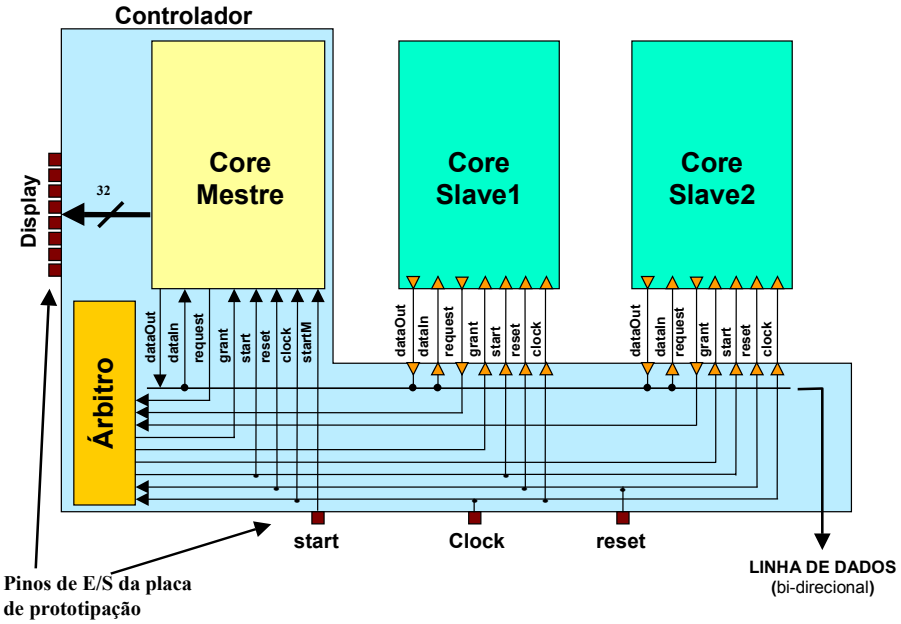

Figura 6 – Integração de *cores* utilizando a proposta de barramento de comunicação.

4.1. Interface entre o Core e o Barramento de Comunicação

Para que os *cores* sejam compatíveis com a interface de comunicação mostrada na Figura 6 foram criados dois módulos, "Send" e "Receive" (Figura 7), que devem ser conectados entre a lógica do usuário (*hw-core*) e o barramento de comunicação.

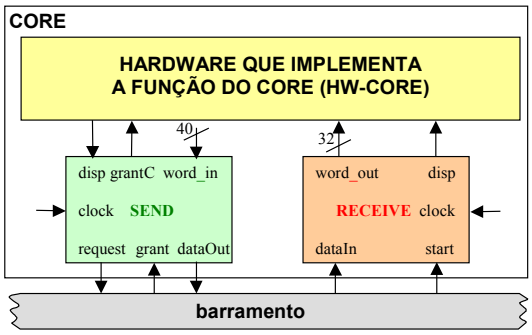

Figura 7 - Módulos Send e Receive.

Quando um dado está disponível para ser enviado ao barramento, o módulo "Send" recebe o sinal "disp". Isto faz com que este módulo envie um sinal "request" para o árbitro e fique esperando pelo sinal "grant". Ao receber o sinal "grant", o módulo recebe o pacote de 40 bits do *hw-core*, através do sinal "word_in". Neste momento, o sinal "grantC" é enviado ao *hw-core* para avisar que o dado já está no módulo "Send", e inicia-se a transferência serial do pacote (através do sinal "dataOut") para o barramento de comunicação, durante os 40 ciclos de clock.

O módulo "Receive" é ativado a partir do momento que recebe o sinal "start", avisando que um pacote vai ser transmitido por um outro *core*. O módulo passa então a armazenar os dados da linha de transmissão (sinal "dataIn"). Após armazenar os 8 primeiros bits, é feita a verificação do endereço de destino. Se o endereço de destino corresponde ao endereço do *core* que está recebendo o dado, a recepção dos 32 bits restantes continua. Do contrário, o restante do pacote é ignorado. Após receber todo o pacote, o módulo avisa ao *hw-core*, através do sinal "disp", e disponibiliza este pacote através do sinal "word_out".

5 RESULTADOS PRELIMINARES

5.1. Exemplo de simulação com comentários

Antes de ser prototipado em hardware, o sistema foi simulado através do ambiente Active-HDL, da Aldec [10]. A Figura 8 ilustra um trecho da simulação.

Inicialmente, a linha de transmissão está em alta impedância (1, na Figura 8). O core mestre envia o sinal "send" ao módulo "Send", avisando que deseja transmitir um pacote (2). O módulo "Send", então, envia o sinal "request" ao árbitro (3), e fica esperando pelo sinal "grant".

O sinal "grant" é enviado ao módulo "Send" do mestre (4) e, logo após, o árbitro avisa aos módulos "Receive", através do sinal "start" (5), que um dado será transmitido.

A transmissão inicia em (6). O módulo "Receive" do core *escravo1* recebe este aviso e inicia a recepção (7). Neste momento, todos os módulos "Receive" de todos os cores estão recebendo o pacote. Após receber os oito primeiros bits (endereço do destino), estes módulos verificam se este endereço corresponde ao seu. Caso não corresponda, o módulo para de receber dados, como aconteceu com o módulo "Receive" do mestre (8). Do contrário, a recepção continua.

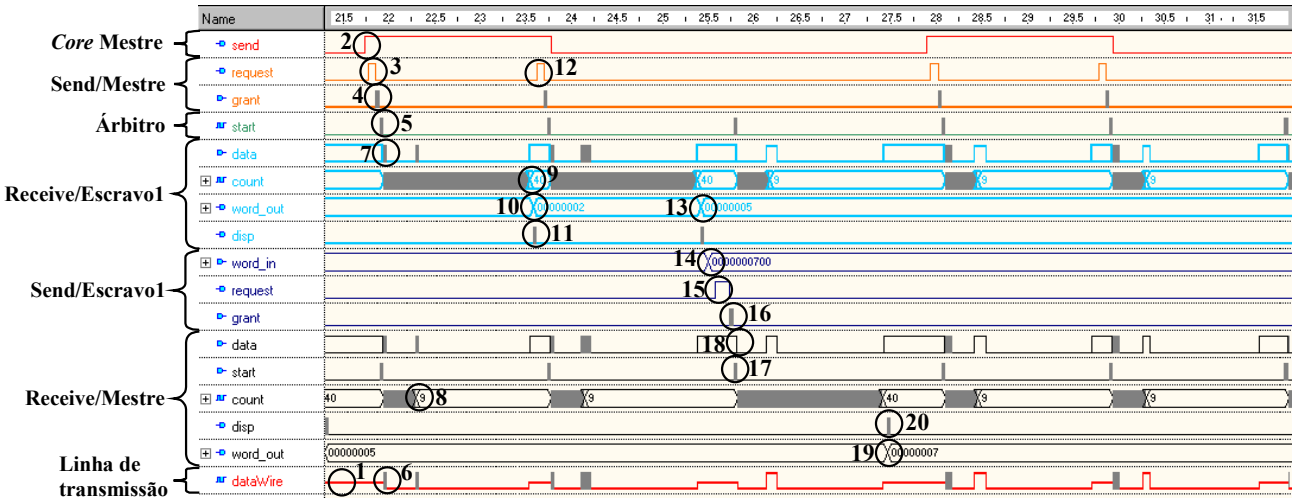

Figura 8 – Simulação da interface de comunicação.

Após 40 ciclos de clock (9), o pacote, contendo o primeiro operando, foi recebido pelo seu destino, o módulo "Receive"do core *escravo1* (10), e este módulo avisa ao *hw-core* que o dado está disponível (11).

Terminada a primeira transmissão, o core mestre permanece com o sinal "send"ativo, pois deseja transmitir o segundo pacote, com om segundo operando. O módulo "Send"do mestre envia novamente o sinal "request" (12), e o processo se repete.

Após ter recebido o segundo operando (13), o core escravo executa a operação para a qual foi desenvolvido (neste exemplo, soma) e disponibiliza o resultado ao seu módulo "Send" (14). Este módulo requisita a linha de transmissão (15) e recebe o sinal "grant" (16), para que inicie a transmissão. Logo em seguida, o módulo "Receive"do core mestre recebe o sinal "start" (17) e começa a receber o pacote, contendo o resultado da operação (18). O pacote é recebido e o dado fica disponível ao core mestre (19), que recebe o aviso (20) do módulo "Receive".

5.2. Prototipação do barramento de comunicação de *cores*

O exemplo da Figura 6 foi prototipado na placa XCV300 [11]. Como exemplo, foi desenvolvida uma aplicação onde foram utilizados dois *cores* escravos, cada um realizando uma operação aritmética diferente com dois operandos.

O *core* mestre , além de determinar quando e qual *core* deve efetuar a operação, também fornece os operandos. Nesta aplicação foram utilizados três operandos (“A”, “B” e “C”) de 32 bits e duas operações (adição e subtração). A forma de depuração utilizada na placa de prototipação foi exibir os valores dos operandos e dos resultados em um display alfa-numérico.

O funcionamento no FPGA foi correto, o que demonstrou que a estrutura de barramento com *buffers* tri-state pode ser utilizada em dispositivos Virtex.

5.3. Reconfiguração dinâmica dos *cores* utilizando a interface de comunicação

Uma vez a interface de comunicação validada por simulação e prototipação, procede-se à geração dos módulos em separado.

Para o exemplo da Figura 6 são gerados 3 *bitstreams*: um para o controlador e um para cada *core* escravo. A posição física de cada *bitstream* é definida pela ferramenta de *floorplaning* (planta-baixa). A Figura 9 ilustra na parte superior os *bitstreams* gerados separadamente.

Utilizando uma ferramenta desenvolvida localmente ao grupo de pesquisa, estes *bitstreams* são agrupados em um único arquivo de configuração. Esta ferramenta permite a geração de:

- *Bitstream* completo. Neste caso, o *bitstream* gerado contém todo o circuito e a reconfiguração **não** é parcial, mas sim total. No presente momento estamos validando este procedimento, visando provar que é possível a junção de diferentes *bitstreams* em um único, mantendo o funcionamento do sistema.
- *Bitstream* parcial. Neste caso, o *bitstream* gerado contém apenas o novo *core*, mais uma pequena parte do controlador, referente às colunas utilizadas pelo *core* (ver arquitetura Virtex). Este é o objetivo final do trabalho, pois permite a reconfiguração dinâmica do sistema.

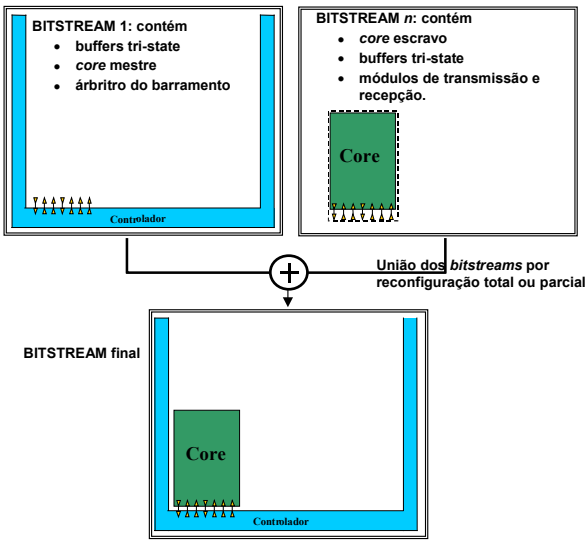

Figura 9 - União dos *bitstreams*.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho enfatizou a importância do re-uso de módulos de hardware (*cores*) na construção de sistemas digitais complexos, bem como a necessidade de distribuição destes módulos (em forma de *bitstreams*) através da Internet. Foram mostrados os requisitos básicos para que esta distribuição seja factível de ser implementada, entre eles, a existência de uma estrutura que permite a comunicação entre *cores* e com o mundo externo. Tal estrutura foi implementada, simulada e prototipada em hardware.

Como trabalhos em andamento, está sendo executada a junção de diferentes *bitstreams* em um único (reconfiguração total), e a geração um bitstream parcial,

contendo apenas parte do circuito (um *core*, por exemplo).

O correto funcionamento da proposta, e a execução de download parcial (trabalho futuro), abrem uma nova perspectiva na distribuição de propriedade intelectual, e no uso de reconfiguração parcial dinâmica em dispositivos FPGA.

Bibliografia

[1] IBM. **The CoreConnect™ Bus Architecture**. Disponível em http://www.chips.ibm.com/products/coreconnect/docs/crcon_wp.pdf (Jun 1999).

[2] ARM Ltda. **AMBA Specification Overview**. Disponível em http://www.arm.com/Pro+Peripherals/AMBA (Jul 2000).

[3] SILICORE Co., **WISHBONE**. Disponível em http://www.silicore.net/pdfiles/wishbone.pdf (Jul 2001).

[4] BERGAMASCHI, R. A., LEE, W. R. **Designing Systems-on-Chip Using Cores.** Design Automation Conference, Los Angeles, California, USA, 2000.

[5] OPENCORES.ORG. **Wishbone SoC Interconnection**. Disponível em http://www.opencores.com/press/pr_8jan2001.shtml (Jul 2001).

[6] Xilinx Inc. **Floorplanner Guide 3.1i**. 2001.

[7] GUCCIONE, S. A., Levi, D., SUNDARARAJAN, P. **JBits: A Java-based Interface for Reconfigurable Computing**. 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD). Julho 2000.

[8] ATMEL, Equipe de documentação da. **At40k Series Configuration**. Disponível em http://www.atmel.com/atmel/postscript/doc1009.ps.zip (Out 2000).

[9] XILINX Inc**. Virtex 2.5V Field Programmable Gate Arrays (DS003)**. Virtex Series datasheet, 2000.

[10] ALDEC, INC. **Code Coverage for Active-HDL.** Disponível em http://www.aldec.com/activehdl/codecoverage.htm (Out 2001).

[11] VCC Inc. **The Virtual Workbench**. Disponível em http://www.vcc.com/vw.html.