Reconfiguração parcial e remota em dispositivos Virtex

Daniel Gomes Mesquita[1] Fernando Gehm Moraes[2]

Resumo: *Este trabalho contempla reconfiguração parcial e remota de FPGAs, tendo como principal objetivo prover a base para desenvolvimentos no campo da reconfiguração dinâmica. A idéia central é criar mecanismos de hardware semelhantes aos utilizados na memória virtual, ou seja, "virtualizar" o hardware. Apresenta análise de dispositivos FPGAs (*Virtex e AT40k*) e ferramenteas (*JBits*) que habilitam reconfiguração parcial. São mostrados resultados do processo de reconfiguração completa e remota, obtido com a utilização de um software desenvolvido no âmbito do Grupo de Apoio ao Projeto de Hardware (GAPH). Uma extensão dessa ferramenta está sendo criada para possibilitar reconfiguração parcial.*

Palavras-chave: Reconfiguração parcial, reconfiguração dinâmica, FPGA.

I Introdução

Muitas aplicações emergentes em telecomunicações, multimídia e eletrônica necessitam que suas funcionalidades permaneçam flexíveis mesmo depois de o sistema ter sido manufaturado [HAD95]. Tal flexibilidade é fundamental, uma vez que requisitos dos usuários, características dos sistemas, padrões e protocolos podem mudar durante a vida do produto. Essa maleabilidade também pode prover novas abordagens de implementação voltadas para ganhos de desempenho, redução dos custos do sistema ou redução do consumo geral de energia.

A flexibilidade funcional é comumente obtida através de atualizações de software mas desta forma a mudança é limitada somente à parte programável dos sistemas. Desenvolvimentos recentes na tecnologia de matrizes de portas programáveis no campo (*Field-Programmable Gate Arrays*, ou FPGAs) têm introduzido suporte para modificações rápidas e em tempo de execução do hardware do sistema [XIL00].

Essas modificações referem-se a mudanças em circuitos digitais via reconfiguração com ou sem a interrupção da operação do circuito. A implementação de sistemas que demandam flexibilidade, alto desempenho, alta taxa de transferência de dados e eficiência no consumo de energia são possibilitadas por essas tecnologias. Isto inclui aplicações de televisão digital, comunicação sem fio reconfigurável, sistemas de computação de alto desempenho, processamento de imagens em tempo real, produtos para consumo atualizáveis remotamente, entre outros.

Além das características citadas acima, a reconfigurabilidade também contribui para a economia de recursos: quando uma dada tarefa pode ser realizada em várias fases, uma diferente configuração pode ser carregada para cada fase seqüencialmente [VIL97], [DEH00]. Desta forma o tamanho do sistema pode ser menor, o que implica na redução de preço. Reconfigurabilidade também faz do desenvolvimento e teste de hardware tarefas mais rápidas e mais baratas. E pode ainda ser usada como tecnologia para construção de sistemas tolerantes a falhas: tais sistemas podem realizar auto-checagem e reconfigurar a si mesmos, substituindo elementos defeituosos por elementos reserva disponíveis [SAN99].

Este trabalho insere-se no contexto de que é importante identificar e documentar a respeito de dispositivos que permitam a implementação de sistemas digitais reconfiguráveis (SDR); e nisto é inédito, pois não foi encontrada na literatura pesquisa que demonstrasse a reconfigurabilidade parcial em dispositivos Virtex.

O artigo está organizado da seguinte forma: a Seção 2 resume o estado-da-arte em sistemas digitais reconfiguráveis; a Seção 3 apresenta dispositivos FPGAs que suportam reconfiguração parcial e que estão disponíveis comercialmente, dando ênfase para a família de FPGAs Virtex. A Seção 4 enfoca a geração de arquivos de configuração parcial para FPGAs Virtex. E a última Seção trata das dificuldades encontradas e dos trabalhos que ainda precisam ser realizados para que seja possível realizar uma reconfiguração parcial num FPGA Virtex.

II Estado-da-arte

As arquiteturas reconfiguráveis podem ser analizadas temporalmente, em função dos problemas a que se dispuseram resolver. A partir do amadurecimento da tecnologia habilitadora para esses sistemas (FPGAs),

[1]Mestrando, bolsista PARKS - dmesquita@inf.pucrs.br
[2]Orientador - moraes@inf.pucrs.br

alguns centros de pesquisa criaram as primeiras arquiteturas reconfiguráveis, com o intuito principal de aumentar o desempenho de algoritmos que até então eram executados em software. Dentro desta primeira geração estão projetos como DECPeRLe [BER89], PRISM [ATH93] e Splash [GOK90]. Alguns sistemas mais modernos ainda utilizam essa abordagem, como o Transmogrifier-2 [LEW98], o RPM-2 [DUB98] e o SPYDER [SAN99]. Tais sistemas podem ser vistos como a primeira geração dos sistemas digitais reconfiguráveis, conforme a Figura 1.

Já neste primeiro momento verificou-se a eficiência da utilização de FPGAs no domínio de aplicação em questão, tanto em termos de desempenho com relação a abordagens em software quanto no que tange ao critério econômico, quando comparada a soluções ASIC. Contudo também alguns problemas foram levantados. Em geral esses sistemas possuíam um gargalo de comunicação entre microprocessadores e FPGAs e apresentavam um tempo de reconfiguração muito alto, além de poderem ser reconfigurados apenas totalmente. Essa última desvantagem significa que o sistema precisava necessariamente ser parado para que pudesse ser reconfigurado.

Em função disso novas propostas de arquiteturas surgiram. O problema de comunicação entre microprocessadores e FPGAs contou com o avanço da tecnologia habilitadora para ser resolvido, formando uma segunda geração de SDR (Figura 1). Com o aumento do número de transistores por circuito integrado (CI) tornou-se possível o desenvolvimento de um sistema composto por microprocessadores, FPGA(s) e memória em um único CI (System-on-Chip, ou simplesmente SoC). Foram desenvolvidos SoCs de granularidade baixa (FIPSoC [SID99], TRUMPET [PER99]) e com granularidade média (Garp [CAL00], RAW [WAI97]).

Outro avanço tecnológico ocorrido foi a possibilidade de reconfiguração dinâmica. Isto permitiu que as arquiteturas em questão pudessem ser reconfiguradas sem que precisassem parar totalmente de desempenhar suas funções. Essa reconfiguração dinâmica pode ser realizada por chaveamento de contexto, como o que ocorre com os sistemas derivados do DPGA. Este também é o caso do DISC, FireFly, FIPSoC e Garp. Tais sistemas encontram-se na segunda geração de SDRs, conforme a Figura 1. Splash2 apresenta uma reconfiguração alternativa, na sua rede de interconexão externa [J. M92].

Além das abordagens derivadas de avanços tecnológicos e resolução de antigos problemas, arquiteturas reconfiguráveis tem sido utilizadas para implementação de algoritmos genéticos. FireFly é um exemplo de "hardware evolutivo" criado a partir desse enfoque. Demonstra uma utilização exótica de reconfiguração parcial e dinâmica.

Informações adicionais a respeito de arquiteturas reconfiguráveis totalmente funcionais podem ser encontradas no site do pesquisador Steave Guccione [GUC00].

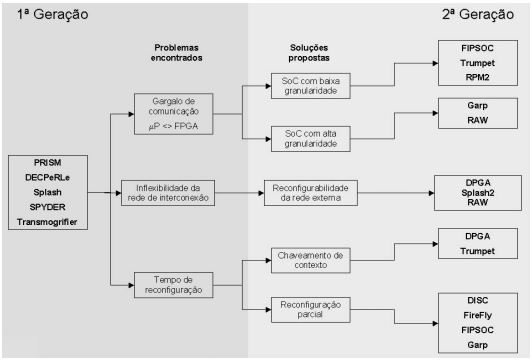

Figura 1: Evolução das arquiteturas reconfiguráveis.

III Dispositivos reconfiguráveis

Os primeiros dispositivos que suportaram reconfiguração parcial foram criados pelas empresas National, Algotronix e Xilinx. O resultado dessa criação foram famílias de FPGAS Clay [NAT98], Cal1024 [ALG89] e XC6200 [XIL99], respectivamente. Tais FPGAs não lograram grande sucesso comercial principalmente pelo fato de não terem sido produzidas ferramentas eficientes de projeto, de roteamento e de posicionamento.

Outro fabricante de FPGAs, a Altera, alega que a partir da família APEX permitiu reconfiguração parcial [ALT01]. Contudo isto ocorre de forma muito limitada. A reconfiguração parcial dessa família dá-se através do projeto de lógica em RAM, criando uma LUT onde podem ser implementadas funções com 7 entradas e 16 saídas. Depois dessa lógica ser implementada no bloco de RAM o sistema pode reescrevê-la em qualquer tempo, mudando a configuração de parte do sistema. A grande limitação desta abordagem é que em algum lugar do circuito deve-se armazenar todas as configurações possíveis que irão modificar a RAM, isto porque não há como fazer a carga externa de novas configurações.

Duas empresas - Atmel e Xilinx - comercializam famílias FPGAs que permitem reconfiguração parcial. Nas próximas Subseções são feitas breves descrições da família AT40k da Atmel e da família Virtex, da Xilinx.

A Atmel

Os FPGAs da família AT40K foram especialmente projetados para suportarem *Cache Logic*, que é uma técnica para construir sistemas e lógica adaptáveis. Num sistema de *Cache Logic* somente as porções da aplicação que estão ativas em um dado momento realmente estão implementadas no FPGA, enquanto funções inativas são armazenadas externamente numa memória de configuração barata. Se novas funções

se fazem necessárias, as antigas são sobrescritas, como mostrado no diagrama contido na Figura 2. Este procedimento aproveita-se da latência funcional inerente a muitas aplicações - em qualquer tempo dado, somente uma pequena proporção da lógica está de fato ativa.

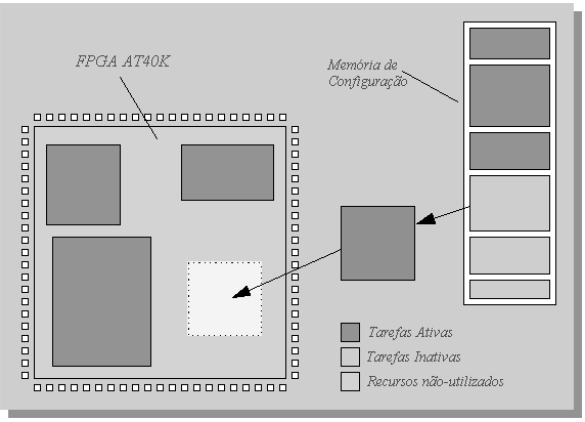

Figura 2: Diagrama da Cache Logic, onde *cores* armazenados em memória configuram o FPGA em tempos diferentes

A série AT40K implementa *Cache Logic*, à medida que pode ser parcialmente reconfigurável, sem interromper a operação da lógica remanescente no dispositivo. Isto permite que funções sejam substituídas em tempo de execução no FPGA, enquanto o sistema continua a operar [ATM00].

A.1 Implementação da Cache Logic

A implementação do *Cache Logic* é conceitualmente semelhante à organização de caches em subsistemas de memória [ATM00a]. No cache de memória uma memória de velocidade mais rápida (normalmente uma SRAM) é utilizada para armazenar os dados ativos, enquanto um maior volume de dados reside numa memória de menor custo, tais como DRAM, EPROM ou disco magnético, etc. *Cache Logic* trabalha de forma semelhante. Na *Cache Logic* somente uma pequena porção do circuito - aquelas funções que são nela carregadas - estão ativas no sistema num dado momento, enquanto as funções não utilizadas permanecem numa memória de custo mais baixo. À medida que novas funções são requeridas, elas podem ser carregadas na *Cache Logic*, substituindo ou complementando a lógica já presente.

A habilidade de implementar *Cache Logic* requer do FPGA a capacidade de ser reconfigurado dinamicamente. Outro requisito é a simetria da arquitetura. Isto é necessário para tornar possível o posicionamento arbitrário de blocos genéricos, em uma localização que esteja disposível no tempo requerido.

Existem dois tipos de *Cache Logic*: predeterminada e dinâmica. A primeira envolve o uso de funções predeterminadas, gravadas numa memória externa não volátil (EPROM, por exemplo). Essas funções são previamente roteadas e posicionadas, e o arquivo de configuração correspondente a elas foi previamente gerado. A implementação dessas funções é previamente controlada por um gerenciador residente na Cache Logic. Novas funções devem ser carregadas na *Cache Logic* em *background*, sem promover uma parada na operação da cache.

A *Cache Logic* dinâmica, por outro lado, é a base para construção de um hardware evolutivo. Cache dinâmico envolve a determinação da lógica, posicionamento e roteamento, geração do arquivo de configuração, e configuração da *Cache Logic* em tempo de execução. Os principais aspectos abrangidos no desenvolvimento dessa capacidade incluem o escalonamenteo e alocação de funções, coleta de lógica aleatória e detecção de colisão.

B Xilinx

Os FPGAs da família Virtex contém blocos lógicos configuráveis (do inglês *Configurable Logic Blocks* - 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 [XIL00].

A funcionalidade deste dispositivo é determinada através do arquivo de configuração, denominado *bitstream*. Arquivos de configuração contém uma mescla de comandos e dados. Eles podem ser lidos e escritos através de uma das interfaces de configuração da Virtex.

Os dispositivos Virtex têm a arquitetura interna organizada em colunas, podendo estas serem lidas ou escritas individualmente. Logo, é possível reconfigurar parcialmente esses dispositivos através da modificação destas colunas no arquivo de configuração [XIL00].

B.1 Arquitetura interna de um FPGA Virtex

A regularidade na distribuição dos elementos computacionais que compõem a arquitetura interna da família Virtex permite ações de relocação e desfragmentação que são importantes para reconfiguração parcial [COM99]. Na Figura 3 pode ser visto que CLBs, IOBs, BlockRAMs, recursos de relógio são distribuídos pelo dispositivo de forma regular, em colunas de elementos configuráveis.

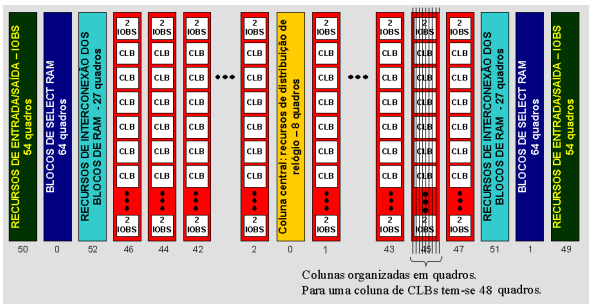

Figura 3: Disposição em colunas dos elementos do FPGA Virtex XCV300.

Ainda na Figura 3 podem ser vistos os quadros, sendo o quadro a unidade mínima de reconfiguração. Cada CLB é composta por 48 quadros sucessivos. Como a disposição das CLBs é em colunas, a modificação de uma CLB implica na alteração de todas as CLBs da coluna a que pertence, logo não é possível reconfigurar uma CLB individualmente. Note-se ainda na mesma Figura que as colunas são numeradas a partir do 0 (zero) - atribuído à coluna central. As demais colunas são numeradas em ordem crescente, com valores pares à esquerda da coluna central, e ímpares à sua direita.

Cada CLB possui dois conjuntos de funções idênticos, denominados slices. A Figura 4 exibe a estrutura interna de uma CLB, onde cada slice contém duas tabelas de busca (*LookUp Tables*, ou LUTs) denominadas F e G, dois *flip-flop*s, um *buffer tri-state*, circuito de propagação rápida de vai-um, além de recursos de roteamento. Cada LUT é composta por 16 bits de configuração, podendo ser configurada como qualquer função booleana de até 4 variáveis, ou memória de porta simples ou porta dupla (neste caso a LUT é denominada LUTRAM).

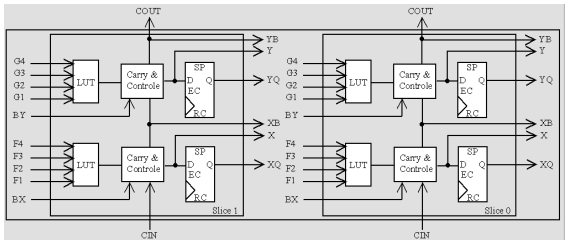

Figura 4: Estrutura interna de um CLB do FPGA Virtex XCV 300.

Os bits da memória de configuração são agrupados em quadros verticais com um bit de largura, que se estendem do topo à base da matriz. Quadros são lidos e escritos seqüencialmente, com endereços crescentes para cada operação. Múltiplos quadros consecutivos podem ser lidos ou escritos com um único comando de configuração, permitindo assim reconfiguração parcial. Cada CLB é cortado por 48 quadros, utilizando 18 bits de cada um. Desta forma, um CLB é completamente configurado por 864 bits.

Nos arquivos de configuração dos FPGAs da família Virtex as informações são gravadas em palavras de 32 bits. Por exemplo, o dispositivo XCV300 possui 48 linhas de CLBs, o que corresponde a 20 palavras de 32 bits. Como deve haver uma palavra vazia ao final de cada quadro, tem-se então 21 palavras de 32 bits por quadro.

Calculando-se o total de quadros, pode-se chegar ao número de bits necessários à configuração de um FPGA Virtex. Este total é dado pela Equação 1 :

$$T_{quadros} = (IOB + RAM + RAMConnect) \times 2 + (Chip_{cols} \times 48) + relogio + 3 \quad (1)$$

Onde:

- IOB é o número de quadros por coluna de IOBs (54);
- RAM é o número de quadros por coluna de RAM (64);
- RAMConnect é o número de quadros por coluna de RAMConnect (27);
- A constante 2 significa que há dois conjuntos de IOB+RAM+RAMConnect (um à esquerda e outro à direita do FPGA);
- CHIPCols é o número de colunas de CLBs do dispositivo (48 para o dispositivo XCV300);
- Relógio é a coluna central da Virtex, que possui 8 quadros;
- A constante 3 significa que há um quadro de preenchimento depois de cada coluna de RAM, e outro quadro para complementar as colunas de CLBs.

Substituindo as constantes na Equação 1, tem-se que o total de quadros do FPGA XCV300 é de 2605. Como cada quadro possui 21 palavras de 32 bits, o XCV300 possui 1.750.560 bits de configuração, mais palavras de comando e sincronização.

B.2 Arquivo de configuração e endereçamento dos elementos internos

O *bitstream* é composto por um fluxo de palavras que segue o protocolo de configuração determinado pelo fabricante para o dispositivo [XIL00]. Este protocolo de configuração é composto por uma coleção de registradores de 32 bits. A lógica de configuração é controlada e acessada através desses registradores.

A identificação dos registradores de comandos e do formato da palavra de dados para cada um deles é necessária para que seja possível a modificação dos parâmetros para uma reconfiguração parcial. Para que seja factível uma reconfiguração parcial, além do domínio da escrita nos registradores de um FPGA, faz-se necessária a possibilidade de localizar determinados bits no

arquivo de configuração. A tarefa a princípio é árdua, pois que somadas todas as palavras de um *bitstream* e multiplicadas por sua largura em bits (32), tem-se que localizar um bit dentre 1.751.808 bits!

Analisando as equações apresentadas em [XIL00], é possível estabelecer um roteiro para localização de um determinado bit de uma CLB no arquivo de configuração. Como conseqüência disto, é possível a leitura de um conjunto de elementos. Por exemplo, dada uma aplicação qualquer, pode ser necessária a modificação do bit 14 de uma F-LUT. A Figura 5 mostra que esse bit está dentro da fatia 0 (S0) da CLB situada intersecção entre a primeira linha (R1) e a primeira coluna (C1) de um FPGA XCV100. O posicionamento de uma CLB em coordenadas pré-definidas pode ser obtido através da edição do arquivo de restrições de usuário (UCF, do inglês User Constraints File). As coordenadas de uma CLB são dadas, pois, por Linha (Row), Coluna (Column) e Fatia (Slice), no seguinte formato: R1C1.S0.

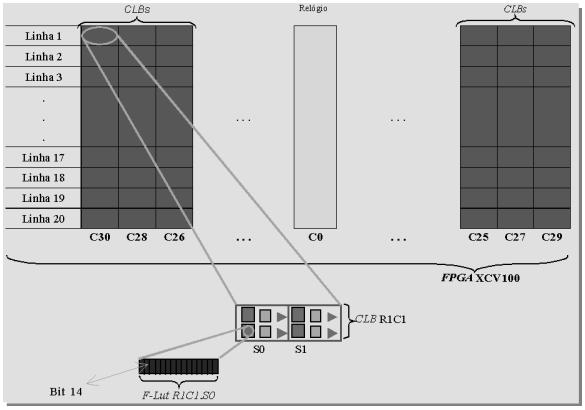

Figura 5: Localizando o bit 14 na CLB dada por R1C1.S0 de um FPGA XCV100.

A localização de uma dado elemento, por exemplo um bit dentro de uma LUT, é feito através dos seguintes passos:

- localização da coluna onde se encontra a CLB;
- localização do quadro onde se encontra o bit desejado, em função do slice e do número do bit;
- localização do bit no quadro, em função da linha (row), e se é LUT G ou LUT F;
- localização de qual palavra no quadro o bit pertence;
- localização de qual bit na palavra corresponde ao bit que se deseja manipular.

Depois de localizado o bit desejado, pode-se proceder a reconfiguração de forma total ou parcial. Reconfigurar o FPGA totalmente é tarefa trivial, uma vez que, após localizar o bit (ou os bits) que se deseja(m) mudar, se faz necessário apenas o recálculo de CRC para geração do novo *bitstream*, sem qualquer modificação no protocolo de configuração. Contudo, para uma reconfiguração parcial, o fluxo de configuração para geração do *bitstream* é diferente, e não está completamente descrito na documentação do fabricante. Porém, depois de dominado o novo protocolo a tarefa torna-se igualmente simples. Atualmente o GAPH está desenvolvendo uma ferramenta que gere automaticamente o *bitstream* parcial.

IV Geração do *bitstream* parcial

A *JBits*

JBits é um conjunto de classes Java que fornecem uma API (Application Program Interface) que permite manipular o arquivo de configuração da família de FPGAs Virtex da Xilinx. Esta interface opera tanto em arquivos de configuração gerados pelas ferramentas de projeto da Xilinx quanto em arquivos de configuração lidos do hardware [MCM99].

O modelo de programação utilizado pelo JBits também usa a abstração de uma matriz de CLBs. Assim, todos os recursos configuráveis na CLB selecionada podem ser configurados ou analisados. Além disso, o controle de todo o roteamento adjacente a CLB selecionada torna-se disponível.

Esta API pode ser utilizada como base para a construção de outras ferramentas. Isto inclui ferramentas de projeto tradicionais para executar tarefas como posicionamento e roteamento do circuito, bem como ferramentas de aplicação específica, como por exemplo um configurador de *cores*[1].

O suporte à orientação a objetos da linguagem Java permite que *cores* parametrizáveis sejam implementados. É necessário que o projetista tenha conhecimento do seu circuito e dos detalhes de configuração do dispositivo, pois, caso contrário, o JBits pode gerar dados que o danifiquem. O JBits fornece uma abordagem de linguagem de alto nível para desenvolvimento de sistemas reconfiguráveis incluindo reconfiguração em tempo de execução.

A característica mais importante do JBits é o seu uso no desenvolvimento de aplicações Java RTR[2]. Neste fluxo, os circuitos podem ser configurados durante a execução através de uma aplicação Java que se comunica com a placa contendo o dispositivo Virtex.

No sentido de obter maior vantagem do suporte à reconfiguração parcial, a API JBits foi estendida com a API JRTR. Esta interface provê um modelo de cache onde as mudanças dos dados de configuração são ajus-

[1] Representação eletrônica de um componente em nível de processamento, como um microprocessador, ou um periférico [VAH01]. Também pode ser tratado como módulos pré-projetados e pré-validados de hardware.

[2] Do inglês *Run-Time Reconfiguration* - reconfiguração em tempo de execução.

tadas, e somente os dados realmente necessários são escritos no dispositivo, ou lidos dele. A Figura 6 mostra o diagrama de blocos de alto nível para o código JRTR. A interface do JBits existente é ainda utilizada para ler e escrever arquivos de configurações do disco, ou de outro dispositivo externo. Mas o "Gerador de Arquivos de configuração" do JRTR é utilizado para analisar o arquivo de configuração, e para manter a imagem dos dados e as informações de acesso.

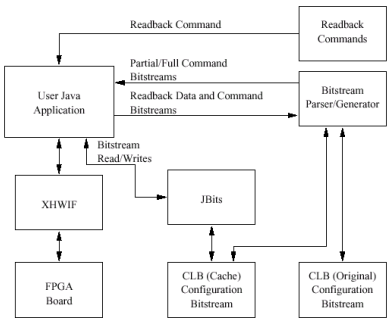

Figura 6: Diagrama de blocos do JRTR

A API atual provê controle simples, mas completo, do cache de configurações. O usuário pode produzir configurações parciais em qualquer tempo, e então carregá-las no hardware.

B Ferramenta para reconfiguração remota e completa

Todo o trabalho de reconfiguração é feito sobre o *bitstream*, ou seja, com *hard cores* e não *soft cores*. A primeira ferramenta desenvolvida para reconfiguração de dispositivos FPGA teve três objetivos, plenamente alcançados:

1. Permitir a alteração de valores armazenados em memória interna do FPGA (LUTRAM). A alteração de valores armazenados em LUTRAM permite modificar o comportamento do circuito pela alteração dos parâmetros de configuração do circuito.

2. Que esta alteração pudesse ser feita de forma remota. Se a configuração do FPGA pode ser realizada remotamente, isto pode significar uma grande redução de custos para o fabricante do dispositivo. Por exemplo, um determinado fabricante detecta um erro em seu circuito. Se reconfiguração remota é utilizada, ele pode atualizar o hardware de todos os seus clientes a distância, sem que estes percebam que o sistema foi modificado.

3. "Esconder" do usuário final a arquitetura do sistema. Devido à complexidade na manipulação de *bitstreams* uma camada de software é acrescentada, de forma que a aplicação final (na forma de uma Applet) mostre ao usuário apenas os parâmetros que devem ser alterados. A Figura 7 ilustra em (a) os comandos para gerar a aplicação de configuração de uma determinada aplicação, e em (b) a aplicação final vista pelo usuário. Na aplicação de reconfiguração tem-se basicamente os parâmetros a inserir, o formato dos parâmetros e a leitura e download do *bitstream*.

A ferramenta para reconfiguração remota e completa de FPGAs foi desenvolvida em Java. Ela permite acessar remotamente os *bitstreams* e visualizar suas configurações. Possibilita modificação de LUTs, procura de uma LUT com uma configuração específica e visualização da listagem de todas as LUTs com configurações diferentes da configuração padrão. Após modificar o *bitstream* é possível salvá-lo e baixá-lo remotamente para a placa sem a necessidade de nenhum software extra. Essa ferramenta foi implementada seguindo o paradigma cliente-servidor.

O lado servidor da aplicação é responsável por atender os clientes e executar todas as funções citadas anteriormente e passar via *sockets* as respostas para os clientes. Também é responsabilidade do servidor salvar um arquivo de configuração e realizar a chamada ao programa *JTAGProgrammer*, que realiza o *download* da configuração para o dispositivo alvo.

Os requisitos para a instalação do servidor é ter o conjunto de classes JBits na máquina, um servidor de páginas para disponibilizar o Applet (Apache), o *JTAGProgrammer* e o JDK 1.2 (ou versão superior) para rodar a aplicação em Java.

Existem duas versões de clientes: uma aplicação Java e outra na forma de Applet. A vantagem da Applet é que não necessita ter na máquina cliente o JDK instalado; basta um navegador de internet que provenha uma máquina virtual JAVA. Em contrapartida, em função das restrições para acesso a recursos locais do cliente pelo Applet [MAT98], não é possível transmitir via esta ferramenta um arquivo de configuração do cliente para o servidor. Portanto o Applet possibilita que remotamente se façam modificações em *bitstreams* previamente colocados na máquina servidora.

Através da versão aplicação é possível transmitir um arquivo ao servidor, e realizar todas funcionalidades possíveis da ferramenta. Contudo, há necessidade de ter o JDK instalado na máquina do cliente, além de a aplicação ter de ser disparada via linha de comando (DOS).

A Tabela 1 ilustra os principais métodos disponíveis no JBits para manipular um *bitstream*.

Uma observação importante é quanto ao endereçamento das CLBs por parte das classes JBits e da documentação da Xilix. Por exemplo, no FPGA XCV300 as CLBs são numeradas de (1,1) a (32,48) conforme [XIL00]. Entretanto, no JBits as CLBs são numeradas de (31,0) a (0,47).

A documentação do JBits menciona a possibilidade e indica classes e métodos para geração de um *bitstream* parcial. Contudo, experimentos realizados mostra-

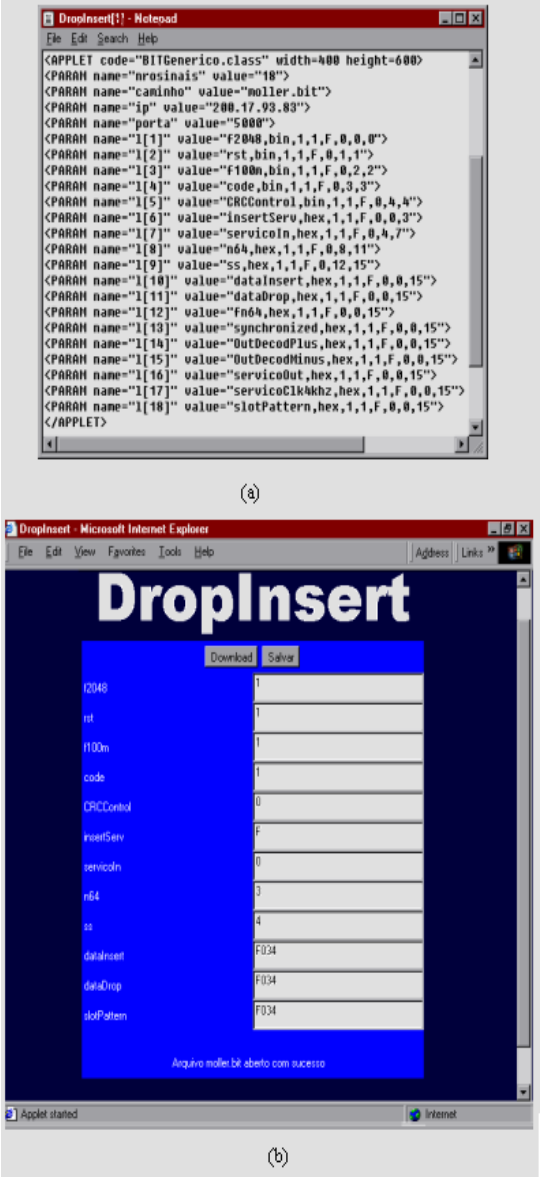

Figura 7: (a) Comandos para geração da Applet e (b) aplicação de reconfiguração.

Comando	Descrição
JBits jbits	Criar instância de JBits para posterior chamadas dos métodos
JBits jbits.read("entrada.bit")	Abrir um *bitstream*
jbits.get(row, col, LUT.SLICE0_F)	Capturar o valor de uma LUT
jbits.set(row, col, LUT.SLICE0_F, val)	Gravar um novo valor em uma LUT
jbits.write("saida.bit")	Salvar um *bitstream*

Tabela 1: Comandos JBits.

ram que a estrutura do arquivo *bitstream* parcial gerado estava incorreto em relação ao protocolo padrão do fabricante, o que impediu de continuar a utilizar JBits para a geração de arquivos parciais.

C *Ferramenta para reconfiguração parcial*

A reconfiguração completa é o primeiro passo para a "virtualização" do hardware. Este termo significa que módulos de hardware podem ser alternadamente inseridos e removidos de um FPGA em operação, exatamente como ocorre com sistemas de memória virtual. Para isto é necessário que o dispositivo FPGA permita reconfiguração parcial e dinâmica.

Encontra-se em fase de desenvolvimento no âmbito do grupo de pesquisa GAPH (PUCRS) uma ferramenta escrita em JAVA, sem a utilização das classes JBits, que permite reconfiguração parcial de FPGAs da família Virtex.

Baseada em um estudo minucioso da arquitetura interna de FPGAs da família Virtex, e de seu arquivo de configuração, essa ferramenta realiza as mesmas operações citadas na Subseção 4.2, além de gerar um *bitstream* parcial. Esse *bitstream* parcial foi pré-validado através de sua inserção em um arquivo de configuração completo, onde substituiu parte da lógica do *bitstream* original (completo).

As etapas de análise do formato do *bitstream* [XIL00] [XIL00a], estudo do protocolo de download, geração do *bitstream* parcial estão concluídas. O trabalho que segue a este é a validação do *bitstream* parcial na placa de prototipação. Esta fase ainda não foi realizada devido à ausência de ferramenta de download que permita a configuração parcial sem reinicialização do dispositivo programável.

V Conclusões e trabalhos futuros

A *Dificuldades encontradas*

Atualmente somente dois fabricantes produzem dispositivos que permitem reconfiguração parcial, mas a documentação referente a como proceder esse tipo de reconfiguração não é clara em muitos casos e não é sequer correta em outros.

Com relação ao endereçamento de elementos em FPGAs da família Virtex, as equações encontradas na documentação do fabricante não são devidamente explicadas. Gastou-se três meses nesta tarefa. Porém, como fruto deste trabalho, elaborou-se um documento didático explicando os pontos obscuros do *Application Note* 151 [XIL00].

Se houveram dificuldades relativas a prolixidade do documento citado acima, o problema foi ainda maior quanto ao protocolo de configuração do dispositivo. A informação contida no *Application Note* 151 a esse respeito estavam imcompletas. Somente analisando esse documento em conjunto com o *Application Note* 138 [XIL00a] (que trata exatamente do protocolo de configuração), e completando as informações obtidas

com o que foi encontrado na lista de discussão mantida pela Xilinx é que foi possível cheagar ao protocolo de reconfiguração parcial de dispositivos Virtex.

Depois de definido o protocolo, outro obstáculo encontrado foi sua validação. Esta ainda não ocorreu porque os softwares utilizados para configurar o FPGA não permitem reconfiguração parcial. Todos programas (*Hardware Debugger e JTagProgrammer*) resetam o FPGA antes do início do download, o que inviabiliza a reconfiguração parcial. Este problema ainda não foi solucionado, mas através de contato telefônico com o representante do fabricante no Brasil soube-se que um novo software que permite download parcial está sendo desenvolvido e deverá estar disponível no início de agosto.

A descrição desse protocolo, bem como o endereçamento dos elementos fogem ao escopo deste artigo, mas serão descritos com detalhes na dissertação.

B Conclusões

Este trabalho mostrou como reconfiguração pode ser utilizada em dispositivos FPGA. A reconfiguração permite a redução de custos no momento da manutenção/atualização do hardware e a redução de custos pela utilização de dispositivos menores através da virtualização do hardware. Para que seja possível a utilização de reconfiguração é necessário preencher dois requisitos: existência de hardware que suporte reconfiguração e ferramentas de projeto para reconfiguração. Apresentou-se um dispositivo que permite reconfiguração, FPGAs Virtex, e desenvolveu-se um conjunto de ferramentas para reconfiguração. Um reconfigurador remoto e completo está operacional, enquanto que a reconfiguração parcial está em desenvolvimento. A reconfiguração, total ou parcial, executada remotamente abre novas perspectivas para o projeto de sistemas digitais, pois passa a permitir praticamente a mesma flexibilidade no hardware quanto a que existe no software.

Trata-se agora de reconfigurar módulos completos de hardware, não simples bits de memória. A configuração de módulos completos permitirá a inserção e remoção de diferentes blocos funcionais, da mesma forma que um sistema de gerenciamento de memória virtual opera com partes de um determinado programa.

C Trabalhos futuros

O trabalho que deve ser iniciado imediatamente trata-se de validar um *bitstream* parcial sem, contudo, realizar a reconfiguração parcial no FPGA.

Esta tarefa consiste na geração de um arquivo de configuração completo, que deverá conter pelo menos um *core*. Conforme pode ser visto na Figura 8- (a), o *core* 1 servirá como um barramento para ligação de outros *cores* com o mundo externo, cuja utilidade foi descrita em [MES01]. Ainda na Figura 8- (a), a área pontilhada mostra onde será inserido o novo *core*. A parte do arquivo de configuração original, denotado na figura pela intersecção entre a área pontilhada e a área hachurada, deverá ser lida, pois será copiada para o novo *bitstream*.

Com o uso de ferramenta de síntese de alto nível (*FloorPlanner*, que faz parte do pacote Foundation, do fabricante Xilinx) é possível restringir o posicionamento de LUTs em uma área pré-determinada. Com isto, gera-se o *core* 3 (área quadriculada da Figura 8- (b)). Através do programa descrito na Subseção 4.3 pode ser anexada a parte "copiada" do primeiro *core*, gerando o módulo que corresponderá à área pontilhada na Figura 8- (a).

Assim, ter-se-á um *bitstream* parcial dentro de um arquivo de configuração completo. Se o download desse novo *bitstream* for realizado com sucesso, então a geração do arquivo de configuração parcial terá sido feita de forma correta.

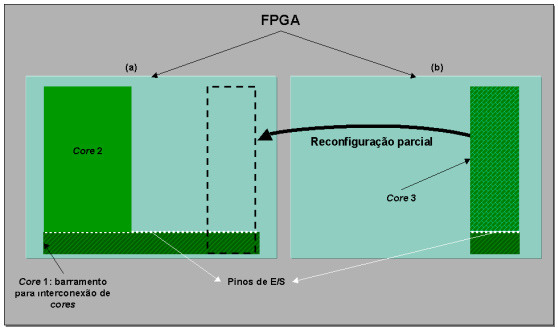

Figura 8: Reconfiguração parcial: adição de um *core* em um dispositivo com um barramento para interconexão entre *cores*.

Infelizmente, somente no momento em que o software responsável pelo download de *bitstreams* para o FPGA permitir download de *bitstreams* parciais é que poder-se-á ter certeza que o protocolo de configuração parcial está correto.

A médio prazo entretanto, o principal trabalho que deve ser desenvolvido para viabilizar a reconfiguração parcial e dinâmica de *cores* em FPGAs, na opinião dos autores deste artigo, é o aprofundamento da pesquisa a respeito de uma interface para conexão de *cores* [MES00]. Uma visão primitiva desta idéia pode ser denotada da Figura 8- (a), onde a área hachurada corresponde ao barramento para conexão entre *cores*.

Esse barramento, denominado interface, seria também um módulo de hardware, porém estático. O FPGA seria inicialmente carregado apenas com esta interface, que possui um controlador. O controlador comunica-se externamente com os pinos de entrada/saída do FPGA, e internamente com os sinais de entrada/saída do(s) *core*(s), através de pinos virtuais. Isto significa que os módulos de hardware a serem conectados, somente comunicar-se- ão com o mundo

externo através desta interface. Os *cores* poderão ser carregados em tempo de execução, possuindo conexões aos sinais da interface. O controlador da interface será 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.

Referências

[ALG89] ALGOTRONIX, Equipe de documentação. **Cal1024 datasheet**. Disponível por WWW em http://dec.bournemouth.ac.uk/drhw_lib/archive/cal1024.ps.gz (1989).

[ALT01] ALTERA, Equipe de documentação da. **Configuration (in-circuit reconfiguration)**. Disponível por WWW em http://www.altera.com/support/devices/programming/sup-configuration.html (2001).

[ATH93] ATHANAS, Peter M.; SILVERMAN, Harvey F. Processor reconfiguration through instruction-set metamorphosis. **IEEE Computer**, p.11–18, 1993.

[ATM00] ATMEL, Equipe de documentação da. **At40k series configuration**. Disponível por WWW em http://www.atmel.com/atmel/ postscript/doc1009.ps.zip (2000).

[ATM00a] ATMEL, Equipe de documentação da. **Implementing cache logic with fpgas**. Disponível por WWW em http://www.atmel.com/atmel/ postscript/doc0461.ps.zip (2000).

[BER89] BERTIN, Patrice; RONCIN, Didier; VUILLEMIN, Jean. **Introduction to programmable active memories**. [S.l.: s.n.], 1989. (PRL Report 3).

[CAL00] CALLAHAN, Timothy J.; HAUSER, John R.; WAWRZINEK, John. The garp architecture and c compiler. **Computer Magazine**, n.4, p.62–69, 2000.

[COM99] COMPTON, Katherine. **Programming architectures for run-time reconfigurable systems**. Washington, Estados Unidos: http://www.ee .washington.edu/faculty/hauck/publications/ KatiThesis.pdf, 1999. Dissertação de Mestrado.

[DEH00] DEHON, André; CASPI, Eylon; CHU, Michael; HUANG, Randy; YEH, Joseph; MARKOVSKY, Yury; WAWRZYNEK, John. Stream computations organized for reconfigurable execution (score): introduction and tutorial. In: PROCEEDINGS OF FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS, 2000. **Anais...** 2000.

[DUB98] DUBOIS, Michael; JEONG, Jaeheon; SONG, Yong Ho; MOGA, Adrian. Rapid hardware prototyping on rpm 2. **IEEE Design and Test of Computers**, n.3, p.112–118, 1998.

[GOK90] GOKHALE, Maya; AL. et. Splash, a reconfigurable linear logic array. **International Conference on Parallel Processing**, v.9, p.219–314, 1990.

[GUC00] GUCCIONE, Steve. **List of fpga-based computing machine**. Disponível por WWW em http://www.io.com/ğuccione/HW_list.html (Agosto 2000).

[HAD95] HADLEY, John D.; HUTCHINGS, Brad L. Design methodologies for partially reconfigured systems. In: IEEE WORKSHOP ON FPGAS FOR CUSTOM COMPUTING MACHINES, 1995, California, Estados Unidos. **Anais...** 1995. p.78–84.

[J. M92] J. M, Arnold; AL. et. The splash 2. In: SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES, 1992. **Anais...** 1992. p.316–324.

[LEW98] LEWIS, David M.; GALLOWAY, David R.; IERSSEL, Marcus van; ROSE, Johnathan; CHOW, Paul. The transmogrifier-2: a 1 million gate rapid prototyping system. In: IEEE TRANSACTIONS ON VLSI SYSTEMS, 1998, Nova Iorque, EUA. **Anais...** 1998. p.188–198.

[MAT98] MATAYOSHI, Cláudio; RUGGIERO, Wilson. Modelo de segurança da linguagem java. In: XVI SIMPóSIO BRASILEIRO DE REDES DE COMPUTADORES, 1998. **Anais...** 1998.

[MCM99] MCMILLAN, Scott; GUCCIONE, Steven A. Partial run-time reconfiguration using jrtr. In: PROCEEDINGS OF FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS, 1999, Glasgow, Escócia. **Anais...** 1999.

[MES00] MESQUITA, Daniel Gomes; MORAES, Fernando Gehm. **Implementação**

de sistemas digitais reconfiguráveis parcial, remota e dinamicamente. Disponível por WWW em http://www.inf.pucrs.br/dmesquita/interest/itens/pep.ppt (Novembro 2000).

[MES01] MESQUITA, Daniel; MORAES, Fernando Gehm; MOLLER, Leandro; CALAZANS, Ney Laert Vilar; PALMA, José Carlos Sant'anna. Reconfiguração parcial e remota de cores fpgas. In: PROCEEDINGS DO VII WORKSHOP IBERCHIP, 2001. **Anais...** 2001.

[NAT98] NATIONAL, Equipe de documentação da. **Navigator**. Disponível por WWW em http://www.national.com/appinfo/ milaero/files/f61.pdf (Julho 1998).

[PER99] PERISSAKIS, Stylianos; JOO, Yangsung; AHN, Jinhong; DEHON, Adré; WAWRZYNEK, John. Embedded dram for a reconfigurable array. In: PROCEEDINGS OF THE SYMPOSIUM ON VLSI CIRCUITS, 1999, USA. **Anais...** 1999.

[SAN99] SANCHEZ, Eduardo; SIPPER, Moshe; HAENNI, Jacques-Olivier; BEUCHAT, Jean-Luc; STAUFFER, André; PEREZ-URIBE, Andrés. Static and dinamic configurable systems. In: IEEE TRANSACTIONS ON COMPUTERS 99, 1999, Nova Iorque, Estados Unidos. **Anais...** 1999. p.556–564.

[SID99] SIDSA, Equipe de documentação da. **Fipsoc mixed signal system-on-chip**. Disponível por WWW em http://www.sidsa.com/FIPSOC/fipsoc1.pdf (Setembro 1999).

[VAH01] VAHID, Frank; GIVARGIS, Tony. Platform tuning for embedded systems design. **IEEE Computer**, California, USA, p.112–114, 2001.

[VIL97] VILLASENOR, John; MANGIONE-SMITH, William. Configurable computing. **Scientific American**, v.19, p.54–58, 1997.

[WAI97] WAINGOLD, Elliot; TAYLOR, Michael; SRIKRISHNA, Devabhaktuni; SARKAR, Vivek; LEE, Walter; LEE, Victor; KIM, Jang; FRANK, Matthew; FINCH, Peter; BARUA, Rajeev; BABB, Jonathan; AMARASINGHE, Saman; AGARWAL, Anant. Baring it all to software: raw machines. **IEEE Computer**, New York, USA, p.86–93, 1997.

[XIL00] XILINX, Equipe de documentação da. **Frequently asqued questions about partial reconfiguration**. Disponível por WWW em http://www.xilinx.com/xilinxonline/partreconfaq.htm (2000).

[XIL99] XILINX, Equipe de documentação. **XC6200 datasheet**. Disponível por WWW em http://dec.bournemouth.ac.uk/drhw_lib/archive/xc6200.pdf (Junho 1999).

[XIL00] XILINX, Equipe de documentação. **Virtex series configuration architecture user guide**. Disponível por WWW em http://www.xilinx.com/xapp/xapp151.pdf (Setembro 2000).

[XIL00a] XILINX, Equipe de documentação. **Virtex fpga series configuration and readback**. Disponível por WWW em http://www.xilinx.com/xapp/xapp138.pdf (Fevereiro 2000).