Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

Estudo da viabilidade da utilização de dispositivos Virtex para implementação de sistemas digitais parcialmente reconfiguráveis

Daniel Gomes Mesquita

Orientador: Professor Fernando Gehm Moraes

Trabalho Individual II

Porto Alegre, Dezembro de 2000

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Considerações Iniciais

Muitas aplicações emergentes em comunicação, computação 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 flexibilidade 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 pode ser conseguida 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 Array*s, ou FPGAs) têm introduzido suporte para modificações rápidas em tempo de execução do hardware do sistema [XIL00]. Essas modificações referem-se a mudanças em circuitos digitais via reconfiguração 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 e sinais digitais adaptáveis, 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]. 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 há ainda o uso de reconfigurabilidade 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.

Neste contexto é importante identificar e documentar a respeito de dispositivos que permitam a implementação de sistemas digitais reconfiguráveis. Este trabalho objetiva situar o leitor a respeito de reconfiguração parcial (que é explicada na Seção 1.3), suas vantagens e problemas; bem como mostrar a viabilidade do uso de um dispositivo FPGA específico para implementar reconfiguração parcial.

Para atingir seus objetivos esta monografia está organizada da seguinte forma:

- Capítulo 1: introdução a reconfiguração parcial, motivação para este trabalho e definições relativas a reconfiguração;
- Capítulo 2: revisão do estado-da-arte em reconfiguração parcial. Aborda diversas publicações recentes que tratam de aspectos como novas tecnologias para FPGAs, CAD, e algumas aplicações que utilizam reconfiguração parcial;

- Capítulo 3: relato sobre tecnologias em hardware disponíveis comercialmente que permitem esse tipo de reconfiguração;

- Capítulo 4: detalhamento da arquitetura de um FPGA da família Virtex da Xilinx e descrição de uma aplicação sobre reconfiguração parcial utilizando este dispositivo;

- Capítulo 5: conclusões e proposta de trabalhos futuros.

1.2 Problemas inerentes a sistemas digitais reconfiguráveis

A despeito do alto desempenho demonstrado pelos sistemas de computação reconfigurável, esses sistemas apresentam algumas limitações comuns:

- *Baixa largura de banda e alta latência de interface:* Como os sistemas reconfiguráveis são comumente anexados a um computador hospedeiro como um periférico através de um barramento com banda limitada e alta latência, ocorre uma alta sobrecarga de comunicação entre o sistema reconfigurável e o processador do hospedeiro. Isto limita as acelerações possíveis e evita uma cooperação mais próxima entre sistemas fixos e reconfiguráveis. A exceção a esta desvantagem são as arquiteturas compostas de processador, memória e lógica reconfigurável em um só circuito integrado (*System-on-chip,* ou SoC); como por exemplo as arquiteturas do FIPSOC [SID99] e Trumpet [PER99] .

- *Alta sobrecarga de reconfiguração*: Uma vez que os custos de reconfiguração são altos em quase todas tecnologias reconfiguráveis disponíveis, o tempo de reconfiguração deve ser amortizado sobre uma grande quantidade de processamento para justificar o sistema computacional [WIR98]. Por exemplo, um dispositivo Virtex XCV300, que possui aproximadamente 1,5Mbits de configuração, a uma taxa de 100MHz demora 15ms para ser completamente configurado. Freqüentemente isto significa que uma única configuração deve ser mantida durante toda uma dada aplicação, mesmo quando porções diferentes da aplicação devam ser aceleradas com diferentes lógicas especializadas. Sistemas que utilizam o paradigma de reutilização seqüencial já não possuem esse problema, tais como o DPGA [DEH94].

Apesar da tecnologia FPGA reconfigurável ter sido comercialmente disponibilizada a mais de uma década [ALG00], o número de ferramentas capazes de suportar projeto de sistemas reconfiguráveis é ainda muito limitado. Muitas das ferramentas existentes são baseadas no fluxo de projeto de FPGAs convencionais, e requerem altos níveis de conhecimento e improvisação, no sentido de produzir um sistema reconfigurável funcional. Alguns fatores como ferramentas de projeto de sistemas reconfiguráveis relativamente imaturas e requisitos de projetos conflitantes (devido ao grande número de tecnologias reconfiguráveis) tornam o projeto destes sistemas reconfiguráveis uma tarefa desafiadora.

Uma das áreas para a qual inexiste ferramenta eficiente é reconfiguração parcial [GUC99]. Um exemplo deste tipo de reconfiguração foi descrito em [XIL00a], e utiliza um *script* escrito em Perl para a escrita de um arquivo de configuração parcial. Tal forma de reconfiguração é limitada a esta aplicação e possui interface pouco amigável. Esta aplicação é analisada no Capítulo 4. Além da falta de ferramentas de *CAD* adequadas, para a reconfiguração parcial, bem como para a reconfiguração dinâmica, ainda não existem aplicações comerciais que justifiquem o esforço para sua utilização.

Contudo, a possibilidade de usar hardware reconfigurável da mesma forma que o microprocessador utiliza a memória virtual conduz ao conceito de hardware sob demanda, que pode ter aplicações interessantes em processamento de imagens [VIL97] e em telecomunicações.

1.3 Definições

Por ser uma área relativamente nova, a computação reconfigurável introduz alguns neologismos, e altera o significado de algumas expressões. A seguir são apresentados conceitos, palavras e expressões relevantes ao entendimento deste trabalho.

FPGA: *Fiel Programmable Gate Array* - dispositivo que consiste em uma matriz de blocos lógicos cercada de blocos de entrada e saída, e conectada por fios programáveis de interconexão.

Grão-grosso: Os FPGAs de grão grosso possuem como grão unidades lógicas e aritméticas (ULAs), pequenos microprocessadores e memórias. Como exemplos desse tipo de arquitetura podem ser citadas as máquinas RAW [WAI97] e a arquitetura GARP [CAL00].

Grão-médio: Os FPAGAs que tem grão médio consistem em blocos lógicos bastante grandes, freqüentemente contendo duas ou mais tabelas de busca (*look-up tables* ou *LUTs*) e dois ou mais *flip-flops*. A maioria das arquiteturas de FPGAs implementa a lógica em *LUTs* de quatro entradas. Como exemplos de dispositivos que atualmente possuem grão médio podem ser citados as famílias Spartan e Virtex, da Xilinx; Flex e Apex, da Altera; e AT40K, da Atmel.

Grão-fino: Nos dispositivos com grão fino há um grande número de blocos lógicos simples. Os blocos lógicos normalmente contêm uma função lógica de duas entradas ou um multiplexador 4 para 1 e um *flip-flop*. As famílias SPGA (Actel) e AT6000 (Atmel) são exemplos atuais de dispositivos de grão fino.

Reconfiguração total: É a forma de configuração onde o dispositivo reconfigurável é inteiramente alterado .

Reconfiguração parcial: É a forma de configuração que permite que somente uma porção do sistema reconfigurável seja reconfigurada. Uma reconfiguração parcial pode ser *não-disruptiva* - onde as porções do sistema que não estão sendo reconfiguradas permanecem completamente funcionais durante o ciclo de reconfiguração; ou *disruptiva* - onde a reconfiguração parcial afeta outras partes do sistema, tipicamente necessitando de uma parada no sistema inteiro. Reconfiguração parcial não-disruptiva é freqüentemente abreviada para *reconfiguração parcial*.

Reconfiguração dinâmica: Também chamada de *run-time reconfiguration* (*RTR*), *on-the-fly reconfiguration* ou *in-circuit reconfiguration*. Todas essas expressões podem ser traduzidas também como reconfiguração em tempo de execução. Reconfiguração dinâmica é outra forma de expressar a reconfiguração parcial não-disruptiva. Não há necessidade de reiniciar o circuito ou remover elementos reconfiguráveis para programação.

DPGA: *Dinamically Programmable Gate Array*. É uma terminologia proposta em [DEH94], que denota um hardware que pode ser programado em execução, durante a operação do sistema, em função de um conjunto de arquivos de configuração pré-carregados. Pode suportar reconfiguração parcial ou total.

Cache Logic: É um termo usado para indicar hardware reconfigurável dinamicamente através de chaveamento de contexto, como o DPGA. É uma marca registrada da empresa ATMEL [ATM00].

1.4 Aplicações em RTR

O campo da computação reconfigurável avançou amplamente na década passada, utilizando FPGAs como a base para sistemas reprogramáveis de alto desempenho [GUC00]. Muitos desses sistemas alcançaram altos níveis de performance e demostraram sua aplicabilidade à resolução de uma larga variedade de problemas. Contudo, apesar dos autores desses sistemas o classificarem como reconfiguráveis, eles são tipicamente configurados uma vez antes de iniciarem a execução da aplicação. Sistemas RTR são distintos daqueles citados acima por permitirem especialização da lógica e do roteamento em tempo de execução, conforme pode se visto na Figura 1.1.

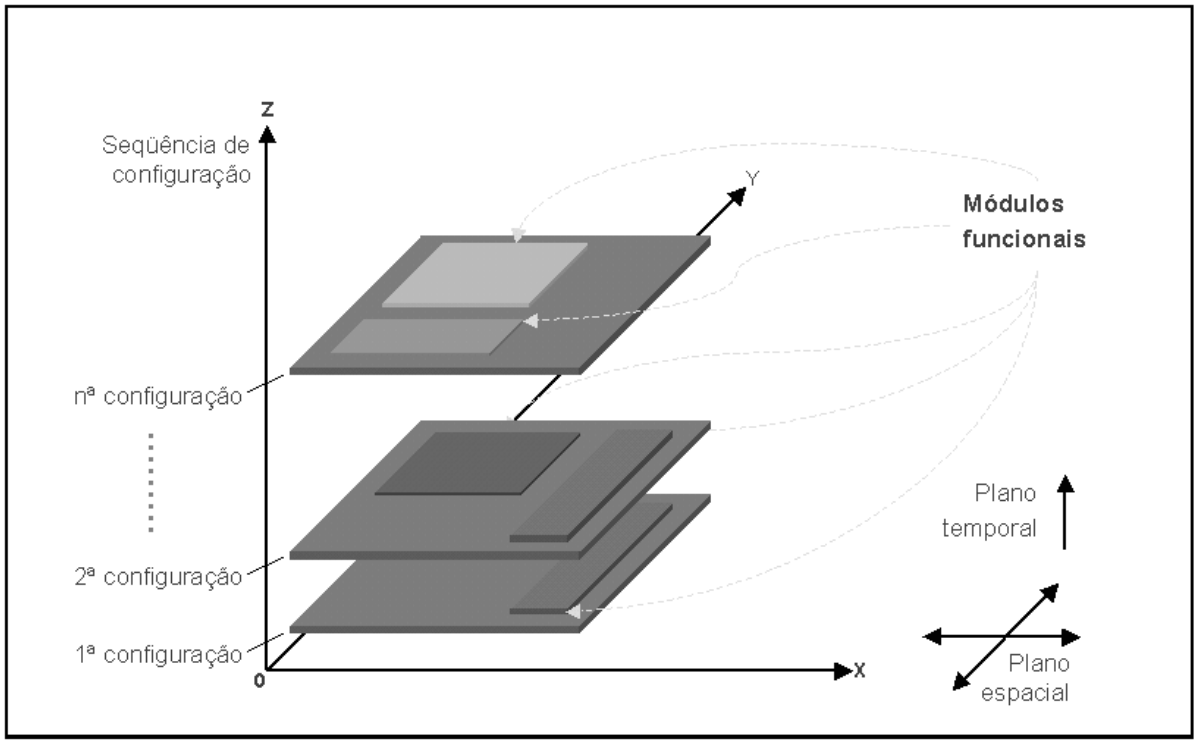

Figura 1.1: RTR

As áreas de um programa que podem ser aceleradas através do uso de hardware reconfigurável freqüentemente são muito numerosas ou complexas para serem carregadas simultaneamente no FPGA disponível. Para esses casos, é interessante que seja possível trocar entre diferentes configurações no mesmo dispositivo.

RTR é semelhante ao conceito de memória virtual, e pode-se até utilizar o termo "hardware virtual" em alguns casos. Conforme essa idéia, o hardware físico é muito menor que do que o somatório dos recursos requeridos para cada uma das configurações. Então, ao invés de reduzir o número de configurações que são mapeadas, apenas ocorre uma troca entre o hardware necessário e o hardware implementado fisicamente.

Apesar da aparente vasta aplicabilidade para sistemas RTR, existem poucos sistemas que de fato implementam esta característica. No Capítulo 2 há um resumo do estado da arte, abrangendo propostas de tecnologia para dispositivos, métodos para projeto/implementação de sistemas RTR, aplicações em RTR e ferramentas de CAD atualmente disponíveis.

2 Estado da Arte

Por tratar-se de uma área bastante nova, há poucos trabalhos relacionados com reconfiguração dinâmica. A seguir serão citados os trabalhos mais importantes a respeito de reconfiguração dinâmica. Logo após há um resumo de cada trabalho citado.

1. Modelos de hardware para reconfiguração dinâmica: DPGA[DEH94] consiste num dispositivo de hardware que permite reconfiguração dinâmica através de chaveamento de contexto. FIPSOC [SID99] é um circuito que integra lógica programável, memória, processador e células analógicas. Em [PER99] as idéias do DPGA e de integração entre processador, memória e lógica programável são unificadas. [COM99] propõe uma arquiterura de FPGA que permita implementação de sistemas RTR.

2. Métodos para projeto/implementação: em [HAD95] há uma proposta de metodologias de projeto para sistemas com reconfiguração parcial; [WIR98] trata de uma proposta de métrica para avaliar a validade da utilização de um sistema RTR ao invés de um sistema tradicional; e em [DEH00] propõe-se um modelo para projeto de hardware para sistemas baseados em chaveamento de contexto.

3. CAD: a falta de ferramentas de CAD para projeto de sistemas RTR já foi citada como um dos principais motivos para sua não-utilização em larga escala. Contudo há algumas iniciativas para provimento desta necessidade, dentre as quais destacam-se o JHDL [BEL98], o JBITS [GUC00a] e o Dinasty [VAS99].

4. Hardware evolutivo: trata-se de uma abordagem aparentemente exótica, mas tem sido demonstrada eficiente para a resolução de alguns tipos de problemas [SIP00]. É análogo ao conceito de algoritmos genéticos, só que a implementação feita em hardware com características de RTR permite que o dispositivo "cresça" ou "diminua" conforme a necessidade da aplicação. Em [SAN99] há alguns exemplos de sistemas desta natureza.

5. Exemplo em telecomunicação: um dispositivo deve suportar três diferentes tipos de protocolos. Esse dispositivo possui uma interface padrão através da qual os protocolos comunicam-se com o mundo externo. O hardware está localizado numa central, e clientes remotos podem requerer canais no sistema, e o próprio cliente escolhe com qual protocolo de rede ele quer trabalhar. Como cada canal pode estar alocado para um cliente diferente, não seria possível interromper o serviço de um cliente enquanto outro canal está sendo modificado. Isso só é possível através de um sistema que utilize RTR [LO00].

2.1 Modelos de hardware

2.1.1 DPGA

A arquitetura da matriz de portas dinamicamente programável (*Dinamically Programmable Gate Array - DPGA*) é particularmente bem adaptada para computação reconfigurável. Diferentemente de FPGAs normais, onde a função de cada elemento da matriz é determinada por seqüências relativamente lentas de reconfiguração, os elementos da matriz DPGA podem chavear rapidamente entre várias configurações pré-programadas. A reconfiguração rápida permite aos elementos do DPGA pré-carregarem múltiplas personalizações para a matriz, e chavear entre configurações em um ciclo de relógio [DEH94].

Semelhantemente aos FPGAs, DPGAs são compostos de uma cadeia de elementos computacionalmente simples. Cada elemento pode desempenhar uma função lógica simples em vários bits de entrada produzindo um ou mais bits de saída. Muitos FPGAs modernos são modelados como tabelas de busca (*lookup tables - LUTs*) programáveis. A LUT constitui a configuração de cada elemento da matriz, conforme pode ser denotado da Figura 2.1. Entre os elementos da matriz há uma rede de interconexão programável que permite aos elementos serem ligados conforme a aplicação necessita. A interconexão em FPGAs é tipicamente configurada pela programação de portas de passagens e multiplexadores. Cada elemento do DPGA usa uma segunda LUT para armazenar diferentes contextos em uma configuração local. Uma mensagem de configuração global informa a cada elemento do DPGA qual a função que irá desempenhar no próximo ciclo do relógio. A interconexão configurável em um DPGA tem uma tabela de configurações carregadas, e seleciona entre as configurações baseada no corrente identificador de contexto.

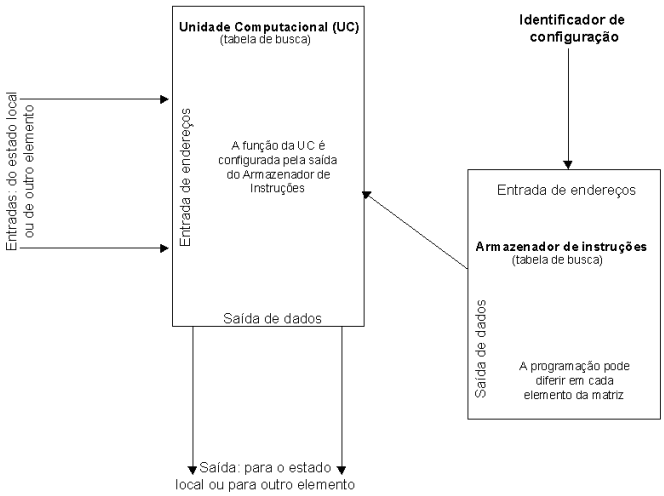

Figura 2.1: Elemento da matriz DPGA

Ao custo de um tamanho maior de elementos de memória e interconexão, o DPGA serve como um FPGA de múltiplos-contextos. Com o espaço dos contextos pré-carregados, o DPGA pode chavear "personalidades" completamente diferentes de um ciclo de relógio para outro. A tabela de configuração do DPGA efetivamente atua como um cache de configurações de elementos da matriz. Fornecendo a cada elemento uma pequena cache de configuração estreitamente acoplada a ele, consegue-se efetivamente uma taxa de reconfiguração alta.

Os múltiplos-contextos carregados permitem a utilização mais eficiente da matriz de elementos. Em aplicações mais pesadas, a rápida reconfiguração permite a uma única matriz DPGA ser carregada com múltiplas configurações simultaneamente. O DPGA pode chavear entre configurações para acelerar diferentes partes de uma aplicação.

2.1.2 FIPSOC

FIPSOC é um SoC desenvolvido pela empresa SIDSA para desenvolvimento rápido de aplicações analógicas e digitais integradas. Cada membro da família FIPSOC possui um microcontrolador 8051 embutido, um FPGA, e um conjunto de células analógicas otimizadas para aquisição de sinais e conversões A/D e D/A, como pode ser visto na Figura 2.2. A SIDSA disponibiliza várias ferramentas de CAD integradas, que possibilitam ao usuário especificar, emular, e mapear todo o projeto em apenas um circuito integrado. A lógica programável do FIPSOC é uma matriz de macro células digitais (*Digital Macro Cells - DMCs*) que podem ser configuradas para funções específicas. As DMCs são de granularidade fina, baseadas em células de RAM estática programáveis, que incluem LUTs de 4 entradas e 4 *flip-flops* para cada uma. O subsistema analógico consiste de blocos analógicos configuráveis (*Configurable Analog Blocks - CABs*) de granularidade grossa. Os CABs permitem configurar diferentes funções analógicas, tais como amplificação diferencial e conversão de dados. A parte digital das células analógicas pode ser independentemente controlada pelo microprocessador ou pelo FPGA [SID99].

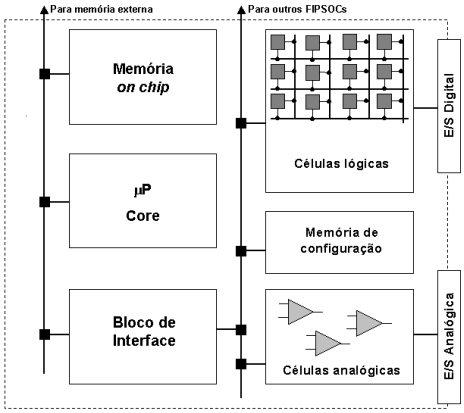

Figura 2.2: Estrutura do FIPSOC

A configuração do circuito integrado é armazenada nos bits de RAM estática. Pode ser feito o carregamento de uma nova configuração para a memória de configuração 2enquanto a célula está em operação. Não há a necessidade de parar o circuito integrado para reconfigurá-lo: duas configurações extras podem ser trocadas em tempo-real.

Por exemplo, um contexto de recuperação pode sempre ser usado para configuração enquanto outro realiza uma computação de propósito geral. Esta reconfiguração dinâmica, parcial ou total, pode também ser colocada em funcionamento pelo próprio hardware reconfigurável, sem a intervenção do microprocessador. A reconfiguração dinâmica pode ser aplicada sobre uma única célula programável, sobre um conjunto selecionado delas, ou sobre toda a lógica reconfigurável.

2.1.3 Trumpet

Trumpet é um circuito integrado de teste que foi projetado para uso no estudo das vantagens envolvidas no projeto de matrizes reconfiguráveis acopladas a bancos de memória DRAM de alta capacidade [PER99]. O ponto alto desta arquitetura consiste em uma rede de "páginas computacionais" (submatrizes configuráveis) e páginas de memória (*Configurable Memory Blocs - CMBs*). Páginas computacionais são baseadas em LUTs de 5 entradas. Páginas de memória e computacionais são interconectadas por uma rede (em forma de árvore) que se estende desde cada submatriz (folhas), alcançando os blocos lógicos individualmente, até uma conexão com um microprocessador (raiz). A Figura 2.3 ilustra essa arquitetura.

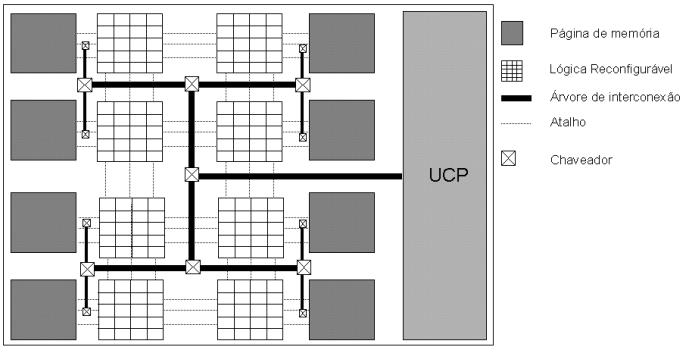

Figura 2.3: Arquitetura Trumpet

Cada submatriz, CMB e chaveador requer um ou mais conjuntos de bits de configuração, dependentes da aplicação. Tais conjuntos de bits podem ser pré-carregados na DRAM, e carregados sob demanda em seus respectivos destinos. Submatrizes e CMBs são arranjados em pares para propósitos de reconfiguração, enquanto a configuração de cada chaveador pode ser assinalada ao CMB mais próximo. Desta forma é possível conseguir reconfiguração total ou parcial no tempo em que se leva para configurar um subarray, um CMB e um ou dois chaveadores. Além disso, conjuntos de bits de estados para CMBs e submatrizes podem ser carregados para inicialização, diagnóstico, ou chaveamento de contexto.

Trumpet é importante por ser uma das primeiras abordagens a valorizar a união de processador, memória e lógica reconfigurável. Grandes bancos de memória tornam possível armazenar várias configurações num mesmo circuito integrado, possibilitando uma rápida reconfiguração em tempo de execução. Ademais, núcleos de aplicações podem beneficiar-se da largura de banda para acessar dados.

Apesar de não haver comprovação de que Trumpet represente o equilíbrio na utilização de recursos de memória e processamento, ainda assim é um passo importante nessa direção.

2.1.4 Arquitetura para RTR de Compton

O trabalho exposto em [COM99] trata de uma proposta de FPGA que permita reconfiguração em tempo de execução, com ênfase na otimização de relocação e defragmentação dos recursos do dispositivo.

Um dos problemas dos sistemas que permitem RTR ocorre quando duas configurações parciais foram sobrepostas durante a compilação na mesma posição física no FPGA. Uma solução para este problema é permitir que o posicionamento final das configurações ocorra em tempo de execução, de forma que ocorra a relocação dinâmica dessas configurações. Utilizando relocação, uma nova configuração pode ser posicionada no FPGA onde ela causar o mínimo conflito com outras configurações já presentes no dispositivo. A Figura 2.4 ilustra uma situação onde relocação e defragmentação são utilizadas.

Mesmo com a relocação, dispositivos que permitam RTR podem ainda sofrer com alguns conflitos de posicionamento que podem ser evitados usando uma otimização de hardware adicional. Como dispositivos que permitem RTR carregam e descarregam configurações no decorrer do tempo, a localização da área desocupada no FPGA é fragmentada, de forma semelhante ao que ocorre com a memória de um sistema genérico. Mesmo que haja área vazia suficiente no FPGA para receber uma determinada configuração, pode acontecer que ela esteja distribuída pelo dispositivo. Uma configuração normalmente requer uma área contígua, então acabará tendo que sobrescrever uma área do CI que contém uma configuração válida. Um dispositivo que pretenda permitir RTR deve incorporar a possibilidade de realizar defragmentação, consolidando áreas não-utilizadas movendo configurações válidas para novas áreas, disponibilizando assim as áreas vazias.

[COM99] propõe uma nova arquitetura especialmente projetada para explorar os benefícios de re-

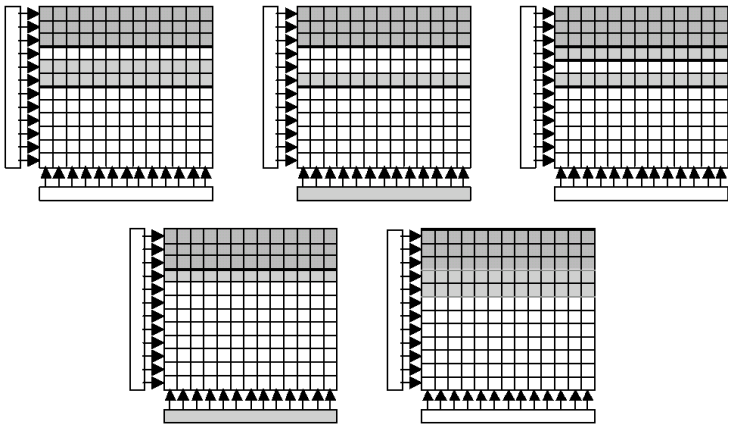

Figura 2.4: Exemplo de relocação e defragmentação

locação e defragmentação. A autora referencia esta proposta de arquitetura como R/D (Relocação / Defragmentação) FPGA.

Usando alguns poucos e simples conceitos na fase de projeto de um FPGA, pode ser assegurado que ele permita relocação e defragmentação:

1. O primeiro é o conceito de configuração parcial, já citado na Seção 1.3. A proposta do R/D FPGA é baseada num *core*[1] genérico parcialmente configurável. A Figura 2.5 (a) mostra um dispositivo cuja lógica programável é composta por elementos configuráveis individualmente, o que permite a reconfiguração parcial.

2. A segunda idéia é a de homogeneidade. Se cada célula na estrutura for idêntica, não haverá obstáculos funcionais para mover uma configuração de uma posição para outra dentro dos limites do dispositivo. Da mesma forma, a estrutura de roteamento deve ser homogênea para evitar limitações de posicionamento.

3. O terceiro conceito é o de Entrada/Saída (E/S) virtualizada. Uma estrutura de E/S baseada em barramento fornece um método independente de posição para ler e escrever dados de configurações individuais. As configurações então não são limitadas por restrições de E/S evitando a preocupação com a localização dos pinos externos do FPGA. Com isto o roteamento de E/S permanece inalterado quando a configuração é mapeada para uma nova localização.

4. A quarta idéia é a de unidimensionalidade. FPGAs comerciais atuais são baseados em uma estrutura bidimensional. O movimento de configurações em duas dimensões para relocação e defragmentação pode ser muito difícil, pois há muitas possibilidades diferentes de posicionamento a serem consideradas. Uma estrutura em linhas, onde cada linha seria uma estrutura atômica para configuração do FPGA, removeria a complexidade da relocação e defragmentação. Isto se dá porque reduz as configurações em objetos unidimensionais, onde a única variação da área de configuração permitida é no número de linhas utilzadas. Operações de rotação horizontal ou vertical, ou deslocamento horizontal não são mais necessárias. A única operação requerida para relocação de uma configuração é a mudança no deslocamento vertical. Um exemplo de relocação/defragmentação por linhas pode ser visto na Figura 2.5 (b).

Por causa da unidimensionalidade, a E/S virtualizada também é simplificada. Ao invés de incluir fios de E/S ao longo de cada coluna e de cada linha do FPGA, esses fios são necessários somente para cada

[1] Módulo de hardware

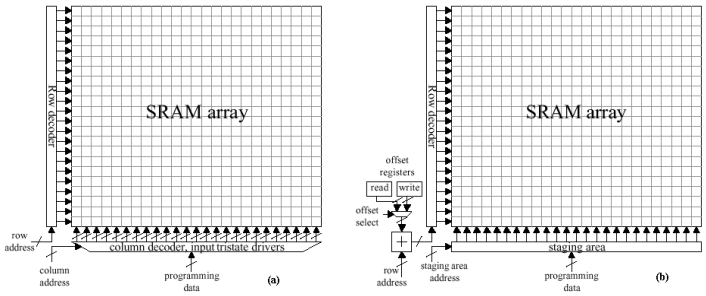

Figura 2.5: Um FPGA com arquitetura básica para permitir RTR (a), e a arquitetura do R/D FPGA (b).

coluna. Apesar da abordagem 2D proporcionar maior flexibilidade de roteamento, acaba por complicar os programas de roteamento e posicionamento. Numa estrutura de roteamento unidimensional, o posicionamento é restrito ao longo de apenas um eixo, o que faz com este posicionamento ocorra muito mais rápido.

2.1.4.1 Arquitetura do R/D FPGA

A memória do R/D FPGA é composta de uma matriz de bits de SRAM. Esses bits são habilitados para leitura/escrita através da decodificação do endereço de linha pelos dados de programação. Contudo, a coluna do docodificador, multiplexador e *drivers* de triplo-estado de entrada foi substituída por uma estrutura denominada área de preparação (*staging*), conforme Figura 2.5 (b).

Essa área de preparação é uma pequena área de armazenamento temporário (*buffer*), o qual é essencialmente um conjunto de células de memória igual ao número de uma linha de programação do FPGA. Cada linha, e por conseguinte a área de preparação, contém palavras de informações. Uma vez que a informação para a linha esteja completa na área de preparação, a área inteira é escrita em uma operação na memória de configuração do FPGA, conforme o endereço contido na linha de endereços. Essa estrutura é semelhante a proposta implementada nos FPGAs da família Virtex da Xilinx.

Na área de preparação do R/D FPGA há um pequeno decodificador que habilita leituras/escritas endereçáveis. A coluna de decodificação determina qual das palavras na área de preparação está sendo referenciada num dado momento.

O decodificador de linha inclui uma sutil modificação, qual seja, a adição de dois registradores, um multiplexador 2 para 1 para escolher entre esses registradores, e um somador. Todas essas estruturas são iguais em largura ao endereço da linha. Isto permite que um deslocamento vertical seja carregado em um ou mais registradores para ser adicionada ao novo endereço de linha, o que resulta num novo endereço de linha relocado. Um dos registradores é o registrador de escrita de deslocamento, o qual armazena o deslocamento de relocação enquanto uma configuração é escrita. O outro é o registrador de leitura, que é usado durante a defragmentação para ler do FPGA uma configuração relocada.

O ponto principal da arquitetura proposta é a possibilidade de posiocinar a lógica em qualquer parte do FPGA, independentemente dos pinos de E/S que são necessários para comunicação com o mundo externo. Somado a isto, a relocação e a defragmentação ocorrem de forma simplificada, criando o ambiente ideal para implementação de sistemas RTR.

2.2 Métodos para projeto, implementação e avaliação de sistemas RTR

2.2.1 Método para projeto de sistemas RTR

Pesquisadores do departamento de engenharia elétrica da Brigham Young University desenvolveram o protótipo de um sistema de rede neural baseada em RTR parcial [HAD95]. O sistema foi chamado de RRANN-2 (*run-time reconfigurable artificial neural network*). Durante o projeto e implementação do RRANN-2, foi desenvolvido um método de projeto para sistemas RTR.

O ponto mais importante deste método é maximizar a lógica que permanecerá inalterada (estática) e minimizar a lógica que precisa ser modificada durante a execução da aplicação (dinâmica). Esta maximização dá-se pelo particionamento da aplicação em blocos funcionais que são em sua maior parte, comuns a todas configurações utilizadas na implementação da aplicação. Esses blocos representam as partes da configuração que não mudam, e portanto podem ser implementadas como um circuito estático. O projeto de recursos para reconfiguração dinâmica só ocorrerá quando não houver unidade funcional correspondente entre a configuração implementada e a que será utilizada. Adicionalmente, a parte estática do circuito é utilizada para reter valores intermediários do dispositivo reconfigurável. Isto elimina circuitos de controle e roteamento que seriam necessários para armazenar valores.

O segundo passo no desenvolvimento de um projeto de RTR parcial é mapear fisicamente os blocos no dispositivo. Além das restrições de implementação e localização necessárias, um bloco lógico também é restringido pelo contexto físico que o cerca. Todos os assuntos referentes a interconexão global, localização global, e posicionamento e interconexão com os blocos próximos tem que ser resolvidos. O problema é que muitas dessas restrições não são conhecidas em tempo de projeto, o que vem a requerer um difícil processo interativo.

Os autores deste método ressaltam a necessidade de pesquisa contínua nessa área, para cristalizar conhecimento a respeito de uma metodologia de projeto para sistemas RTR. Também citam a falta de CAD eficiente para suportar definições de restrições, localização e roteamento.

2.2.2 Métrica para avaliação de sistemas RTR

Os aumentos na eficiência providos através de RTR não são obtidos sem custo [WIR98]. Tempo e largura de banda de memória adicionais são requeridos para transferir os bits de configuração do circuito de uma unidade de armazenamento fora do FPGA para dentro da memória de configuração do dispositivo. Em alguns casos esse tempo extra elimina as vantagens de uma especialização em tempo de execução. Para outros casos, contudo, a relação entre custo de reconfiguração e ganhos em desempenho e economia de recursos compensa em muito o tempo de configuração.

Os autores deste estudo propuseram uma forma de avaliar a validade da utilização de RTR através de um método que calcula o equilíbrio entre ganho de eficiência e custo de configuração. A base da análise que realizaram é a densidade funcional.

A métrica de densidade funcional é definida nos termos de custo da implementação de uma lógica no hardware. Esse custo (C) é tradicionalmente medido como o produto entre a área total requerida para implementar a lógica em hardware (A) e tempo total de operação (T), ou

$$C = AT \tag{2.1}$$

e aplica-se àqueles sistemas onde a vazão (número de operações por segundo) é mais importante que a latência. T inclui o tempo requerido para execução, controle, inicialização e transferência de dados.

Densidade Funcional (D) mede a vazão computacional de uma unidade de hardware, e é definida como o inverso de C:

$$D = \frac{1}{C} = \frac{1}{AT} \tag{2.2}$$

A métrica densidade funcional em (2.2) será utilizada para comparar circuitos modificados estaticamente contra as alternativas em reconfiguração dinâmica.

O incremento (I) na densidade funcional de alguns sistemas RTR (D_r) sobre as alternativas estaticamente reconfiguráveis (D_s) é computada como a diferença normalizada entre D_r e Ds, como segue:

$$I = \frac{\Delta D}{Ds} = D_r - \frac{D_r - D_s}{D_s} = \frac{D_r}{D_s} - 1 \tag{2.3}$$

A porcentagem de incremento é medida multiplicando (2.3) por 100.

A maior diferença entre sistemas RTR e os estáticos é o custo adicional de reconfiguração. Os dispositivos atuais requerem que a configuração do circuito e a execução ocorram separadamente. Isto força a adição do tempo de configuração ao tempo total de operação de um sistema RTR. Para estes sistemas, o tempo total de operação T inclui o tempo total de execução (T_e) e o tempo total de configuração (T_c), ou

$$T = T_c + T_e \tag{2.4}$$

Substituindo T em (2.2) tem-se a densidade funcional acrescida do custo de configuração:

$$D_r = \frac{1}{A(T_c + T_e)} \tag{2.5}$$

A partir de (2.5) fica claro que o tempo de configuração reduz a densidade funcional.

Contudo, apesar do tempo de configuração absoluto ser um importante parâmetro, o tempo de configuração relativamente ao tempo de execução é muito mais informativo. A taxa de configuração

$$f = \frac{T_c}{T_e} \tag{2.6}$$

define este importante parâmetro. O tempo total de operação pode ser expresso como

$$T = T_e(1 + f) \tag{2.7}$$

Substituindo este tempo em (2.2) obtem-se a métrica de densidade funcional nos termos de f:

$$D_r = \frac{1}{AT_e(1 + f)} \tag{2.8}$$

Como sugerido em (2.8), tempos longos de configuração podem ser tolerados se houver um tempo longo de execução correspondente (f pequeno). No limite, como $f \to 0$, a sobrecarga imposta pela configuração é negligenciável. Tais sistemas aproximam-se da máxima densidade funcional provida por RTR. Essa valor máximo (D_{max}) é calculado ignorando-se o tempo de configuração:

$$D_{max} = Lim_{f \to 0} D_r = \frac{1}{AT_e} \tag{2.9}$$

Usando-se D_{max}, o limite superior do incremento (I_{max}) sobre um sistema estático pode ser encontrado. Este parâmetro sugere o benefício máximo de um sistema RTR, e é uma boa indicação da pertinência

da utilização de RTR para uma dada aplicação:

$$I_{max} = \frac{D_{max}}{D_s} - 1 \tag{2.10}$$

Além de propor essas métricas, os autores do artigo sugerem que ao projetar sistemas RTR procure-se aumentar ao máximo o tempo de execução do sistema, para que se justifique o uso de RTR.

2.2.2.1 Exemplo de comparação entre um sistema RTR e um sistema estático

A seguir será justificado o tempo de configuração de uma alicação RTR, segundo a abordagem apresentada acima. A aplicação analisada é a implementação de uma rede neural que usa o algoritmo de de aprendizagem por retro-propagação em FPGAs. Como mostrado na Figura 2.6, o algoritmo é particionado em três estágios: pós-alimentação, retro-propagação e atualização. Esse processo de três estágios é repetido até que o algoritmo de treinamento converja.

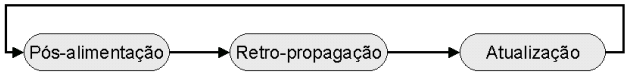

Figura 2.6: Três estágios do algoritmo de treinamento por retro-propagação

A estrutura básica do sistema RRANN-2 foi projetada para explorar a característica de reconfigurabilidade parcial do FPGA Clay. Essa particularidade é usada para redução do tempo de reconfiguração, pois não há mais necessidade de configurar o sistema inteiro. RRANN-2 consiste de um processador de propósito geral e um conjunto de processadores de propósito específico para execução do algoritmo de retro-propagação. Os três processadores neurais foram projetados para compartilhar o máximo de hardware, com a finalidade de reduzir informação de configuração para cada passo de reconfiguração.

Abordagem estática utiliza um FPGA para cada neurônio.

Na Tabela 2.1 são mostrados os parâmetros de comparação entre o sistema estático e o sistema dinâmico, para uma rede de 60 neurônios. Como mostrado, o circuito RTR completa o algoritmo com um terço do hardware necessário para o sistema estático. Essa redução em hardware fornece um incremento máximo na densidade funcional (I_{max}) de 168% sobre a alternativa estática.

	ESTÁTICO	*Máximo*	*RTR Global*	*Parcial*
CONEXÕES	10980	10980	10980	10980
A (células)	50764	18094	18904	18904
T_e (ms)	1,49	1,49	1,49	1,49
T_c (ms)	N/A	0	2,42	1,16
f	N/A	0	1,62	0,78
D	146	391	148	219
I	0	1,68	0,014	0,50

Tabela 2.1: Parâmetros de circuito para uma rede neural com 60 neurônios no sistema RRANN

Para calcular a densidade funcional real do sistema RTR o tempo de configuração deve ser conhecido. Os três passos de reconfiguração requeridos pelo RRANN-2 consomem 2,42ms quando a reconfiguração global é utilizada. Apesar de isto justificar o sistema com uma taxa de reconfiguração de 1,62, o incremento total obtido pela utilização de RTR é de apenas 1,4%. Reconfigurando somente as partes do circuito que precisam ser mudadas, o tempo de reconfiguração é reduzido para 1,16ms. Esta redução no tempo de configuração tem impacto significante na densidade funcional total. Como resumido na Tabela 2.1, a taxa de configuração é reduzida para 0,78 e um incremento na densidade funcional é de 50%.

Como sugerido antes, a densidade funcional é dependente do tempo de execução e aumenta com o tamanho da rede.

2.2.3 Método para projeto de hardware para RTR

Em [DEH94] os autores lembram que apesar de os FPGAS atuais possuírem características de hardware interessantes, como dispositivos com mais de um milhão de portas lógicas equivalentes, altas taxas de relógio, reconfiguração rápida e grande largura de banda, seu uso para reconfiguração dinâmica ainda é limitado. Essa limitada aplicabilidade deve-se a exígüa oferta de ferramentas para automação de projeto e implementação, e também, pela ausência de um modelo computacional que abstraia os recursos fixos dos dispositivos da descrição do hardware a ser projetado.

Para justificar esta hipótese, os autores citam alguns pontos problemáticos do projeto de sistemas RTR:

- Dispositivos-alvos não são compatíveis: Programas desenvolvidos para um dispositivo em particular (ou conjunto de dispositivos) possuem compatibilidade de código-fonte bastante limitada, e nenhuma compatibilidade do arquivo de configuração entre dispositivos de diferentes fabricantes. Organizar um programa para uma nova geração de dispositivos, ou para uma menor e mais barata, ou com consumo mais baixo, tipicamente requer esforço humano substancial.

- Dispositivos-alvos possuem recursos físicos limitados: a limitação de recursos de hardware tende a prejudicar a expressividade desses sistemas e sua ampla aplicabilidade. E tais modelos de projeto de hardware, a escolha de um algoritmo para aplicação está restrita ao tamanho do hardware disponível. Além disso, uma estrutura de computação tem que ser fixa em tempo de execução, sem a permissão de alocação dinâmica de recursos.

Como solução, os autores apresentam a computação organizada em fluxos para execução reconfigurável (*Stream Computations for Reconfigurable Execution* - SCORE). O modelo computacional SCORE tenta resolver o problema da limitação de recursos físicos através virtualização de recursos computacionais, de comunicação e de memória do hardware reconfigurável. Configurações do FPGA são particionadas em páginas de tamanhos fixos que se comunicam: em analogia à memória virtual, o hardware é carregado (*paged in*) sob demanda. A comunicação por fluxo entre as páginas que não estão simultaneamente no dispositivo pode ser transparentemente buferizada[2] através da memória. Este esquema permite uma aplicação particionada explorar mais as páginas disponíveis fisicamente, sem necessidade de recompilação.

Com um projeto adequado de hardware, este esquema permite compatibilidade binária e escalabilidade entre diferentes famílias de dispositivos através da compatibilidade das páginas.

Para os programas beneficiarem-se dos recursos físicos adicionais (páginas), o modelo de programação é uma abstração natural da comunicação que ocorre espacialmente entre blocos de hardware. Isto é, o grafo do fluxo de dados captura os blocos de computação (operadores) e a comunicação (fluxo) entre eles. Uma vez capturado, é possível explorar técnicas para mapeamento desses grafos em hardware de tamanho arbitrário. Além do mais, a composição em tempo de execução dos grafos é suportada, possibilitando uma estrutura de aplicação orientada a dados, alocação dinâmica de recursos e e integração entre módulos de hardware (*cores*) desenvolvidos ou compilados separadamente.

[2]Neologismo baseado no termo em inglês *buffer,* que é um dispositivo para armazenamento temporário na memória

2.3 Ferramentas para RTR

2.3.1 JRTR

Apesar de FPGAs baseados em SRAM poderem ser passíveis de sofrerem inúmeras configurações, existem muito poucas ferramentas de programas, bem como poucas informações disponíveis comercialmente para realização de RTR [MCM99]. Este cenário começou a mudar com a apresentação da família de FPGAs XC6200 [XIL99]da Xilinx, que permitia reconfiguração parcial. Porém, segundo os autores de [MCM99], tal família não logrou sucesso por não ter surgido nenhuma ferramenta comercial que explorasse a característica de RTR da XC6200.

No sentido de criar um conjunto de ferramentas para reconfiguração parcial, a Xilinx iniciou o projeto JERC6K, vinculado à família 6200. Com a descontinuidade desse FPGA, o projeto foi transferido para suportar a família XC4000, e foi renomeado para *Xilinx Bitstream Interface* (ou XBI). Esse programa foi mais tarde renomeado para JBits. Contudo a família XC4000 não continha suporte para reconfiguração parcial. Quaisquer mudanças na configuração do circuito requeriam a parada do dispositivo, e sua reconfiguração era lenta ao ponto de ser inaceitável para determinadas aplicações.

Mais recentemente, o programa JBits foi portado para a família de dispositivos Virtex (ver Capítulo 4). A princípio, o JBits reconfigurava dispositivos Virtex causando a disrupção do sistema, como ocorria com a família XC4000.

A recente adição do programa JRTR (*Java Run-Time Reconfiguration*) à versão Virtex do conjunto de ferramentas JBits resultou no suporte direto à reconfiguração parcial. Este suporte utiliza uma combinação de técnicas de hardware e programas para permitir que pequenas alterações sejam feitas diretamente no arquivo de configuração da Virtex, de forma rápida e sem a interrupção da operação.

2.3.1.1 Interface do JRTR

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 [XIL00b].

O modelo de programação utilizado pelo Jbits é um a matriz bidimensional de blocos lógicos configuráveis (*Configurable Logic Blocks* - CLBs). Cada CLB é referenciada por uma linha e uma coluna. Assim, todos os recursos configuráveis na CLB selecionada podem ser configurados ou analisados. Além disso, o controle de todo o roteamento adjacente à CLB selecionada torna-se disponível. Devido ao código ser escrito em Java, o tempo de compilação é bastante rápido, e pelo controle ser ao nível de CLB, os arquivos de configuração podem ser modificados ou gerados rapidamente.

Esta API tem sido utilizada para construir circuitos completos e para modificar circuitos existentes. O suporte à orientação à objetos da linguagem Java permite que *cores* parametrizáveis sejam implementados. 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*.

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 danifiquem o dispositivo. 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 para este trabalho é o seu uso no desenvolvimento de aplicações Java RTR. 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 da Virtex 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 ajustadas, e somente os dados realmente necessários são escritos no dispositivo, ou lidos dele.

A Figura 2.7 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 JRTR *Bitstream Parser/Generator* é utilizado para analisar o arquivo de configuração, e para manter a imagem dos dados e as informações de acesso.

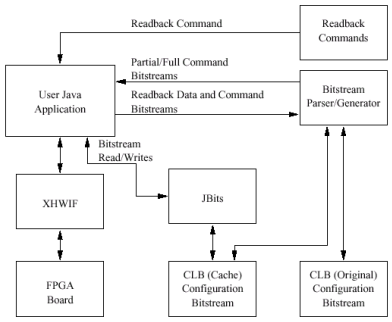

Figura 2.7: Diagrama de blocos do JRTR

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.

2.3.2 JHDL

Segundo [BEL98], durante o desenvolvimento de aplicações para computadores configuráveis (*Configurable Computing Machines* - CCMs), projetistas devem realizar duas tarefas gerais. Primeiro, devem projetar o circuito que implementa a funcionalidade necessária para a aplicação. Isto é tipicamente feito usando ferramentas comerciais de CAD, tais como VHDL, em conjunto com ferramentas *back-end* obtidas junto a fabricantes de FPGAs. Segundo, projetistas devem escrever um programa para supervisionar a operação da aplicação. Nos casos de aplicações RTR de maior complexidade, este programa de controle pode ser igualmente complexo, carregando uma variedade de configurações e dados, sob demanda, conforme a necessidade da aplicação. Atualmente, o programa de controle e a descrição do circuito devem ser desenvolvidos simultanea e independentemente; o projetista é responsável por garantir que esses dois pedaços de programas irão cooperar corretamente, tipicamente através de ciclos de carga, execução e compilação repetidos.

Esta divisão entre descrição do circuito e programa de controle é na realidade a divisão da aplicação entre partes estática e dinâmica: a estática representada pela biblioteca de circuitos, e a dinâmica constituída pelo programa de controle, que escolhe configurações de hardware de uma biblioteca, configura o dispositivo e executa a aplicação. Contudo, devido as vertiginosas mudanças que ocorrem no campo da computação reconfigurável, tratar as partes estáticas e dinâmicas da aplicação é inadequado e limitante. O que é necessário é uma única e integrada descrição que permita ao projetista naturalmente expressar as partes dinâmicas e estáticas da aplicação simultanea e conjuntamente.

Nesse sentido Brad Hutchings e Peter Bellows propuseram uma abordagem de projeto e uma ferramenta de CAD focadas na criação de uma descrição integrada. Seu projeto foi desenvolvido baseado nos seguintes requisitos:

1. A ferramenta deve usar uma liguagem de programação existente, sem extensões. Isto possibilita que um grande número de programadores possam usar a ferramenta.

2. O paradigma de controle da CCM deve ser o de CCM independente. Detalhes de controle da CCM devem ser elevados a um nível mais alto de abstração, com o objetivo de facilitar o processo de redirecionamento das aplicações para vários dispositivos-alvo.

3. O método de descrição deve suportar RTR total e parcial. Espera-se que esse seja o tipo de CCMs que mais necessita ferramentas de CAD.

4. A descrição integrada deve servir para simulação e execução, sem modificações.

O sistema JHDL é implementado como um conjunto de bibliotecas de classes Java, com funcionalidade dividida em duas áreas básicas: simulação de circuito e suporte a execução da CCM. As classes referentes ao suporte da execução provêm acesso transparente às funções de controle da CCM via mecanismos de construção/destruição.

Projetistas desenvolvem circuitos em JHDL selecionando um conjunto de elementos síncronos ou combinacionais, e ligando-os de modo a formar um circuito síncrono arbitrário. Existem três classes diferentes que podem ser utilizadas para implementar um circuito: *CL* (combinacional), *Synchronous* (síncrono) e *Structural* (interconexão entre elementos síncronos e combinatórios). No uso de cada classe, o projetista define uma nova classe que herda características da classe apropriada e implementa a funcionalidade desejada no construtor e em outros métodos. Circuitos individuais são interconectados instanciando objetos *Wire* e passando esses objetos como argumentos para os construtores.

Maiores informações a respeito do status atual do projeto JHDL podem ser obtidas no seguinte endereço: http://www.jhdl.org/

2.3.3 Dynasty

A ferramenta proposta em [VAS99] implementa uma biblioteca para projeto de sistemas RTR. A implementação é baseada na abstração de RTR no sentido da utilização de uma planta-baixa temporal, que permite manipulação do projeto nas dimensões espacial e temporal.

2.3.3.1 Projeto de planta-baixa temporal

Projeto de planta-baixa (*floorplanning*) no nível de disposição (*layout*) é uma técnica comum em fluxos de projetos para FPGAs/ASICs, a qual é utilizada para definir ou modificar posições espaciais de módulos de projeto, no sentido de incrementar desempenho ou eficiência na implementação do projeto. Para fins de diferenciação, este tipo de projeto de planta-baixa será doravante denominado planta-baixa espacial. Uma planta-baixa espacial de um projeto estático (que não sofrerá reconfiguração) permanece inalterado durante todo o tempo de vida do projeto. Em sistemas RTR, contudo, a presença e a posição espacial de cada módulo de projeto reconfigurado podem mudar com o tempo. Em cada instante durante o tempo de execução, o projeto de planta-baixa espacial é determinado pela sua coordenada num espaço de projeto temporal. Um plano definido pela coordenada temporal corresponde ao projeto de planta-baixa temporal para uma configuração de projeto. Este conceito é ilustrado na (Figura 1.1).

Os autores desta ferramenta chamaram o processo de transformação do projeto de alto-nível na forma comportamental para sua implementação em projetos de planta-baixa espacial posicionados num espaço temporal de *projeto de planta-baixa temporal*. Este processo combina dois problemas de fluxo de projeto em uma fase de projeto: (i) particionamento e seqüenciamento de módulos de projeto em configurações

de projeto (também chamadas de particionamento temporal), e (ii) posicionamento temporal de módulos de projeto e conexão com a área reconfigurável de cada configuração.

No sentido de permitir completa exploração de projeto espacial das porções RTR do projeto, o projeto de planta-baixa temporal precisa resolver outros problemas relacionados: (iii) escalonamento (particionamento do passo de controle), (iv) alocação de unidades funcionais, e (v) alocação de registradores.

O projeto de planta-baixa temporal fornece ao projetista uma melhor abstração de projeto de espaço para sistemas RTR. O fluxo de projeto convencional não provê visualização do espaço, além de que torna sua exploração uma tarefa difícil. Por outro lado, projeto de planta-baixa temporal é bem ajustado para visualização de projeto para dimensões espacial ou temporal, em 2 ou 3D.

2.3.3.2 O *framework* Dynasty

Dynasty é um *framework* extensível e portável, projetado para pesquisa de técnicas e métodos para projeto de sistemas RTR.

No sentido de suportar a geração de plantas-baixas espaciais tardiamente no fluxo de projeto (necessário no caso de sistemas RTR), a representação interna do projeto disponibiliza visões de projeto em vários níveis de abstração: (i) comportamental, (ii) estrutural, (iii) projeto de planta-baixa temporal, e (iv) disposição (*layout*). Uma representação robusta de projeto implementada como um a coleção de objetos C++ está sendo usada para capturar soluções de projeto durante todo o fluxo do projeto. Outro conjunto de estruturas de dados em C++ é utilizado para implementar o gerenciamento de biblioteca baseado num servidor comum de bibliotecas.

A representação interna do projeto permite combinação de visões de projeto durante o desenvolvimento do sistema RTR. Por exemplo, durante o projeto de planta-baixa temporal várias porções do projeto podem estar disponíveis nas visões arquiteturais e comportamentais. Representações incompletas do projeto são também suportadas, com o intuito de suportar inserção posterior de controladores, ou outros circuitos estáticos. A representação interna do projeto também facilita a análise de métricas de projeto e armazenamento de resultados analisados.

O projeto de planta-baixa temporal implementado no *framework* Dynasty é descrito na Figura *2.8.*

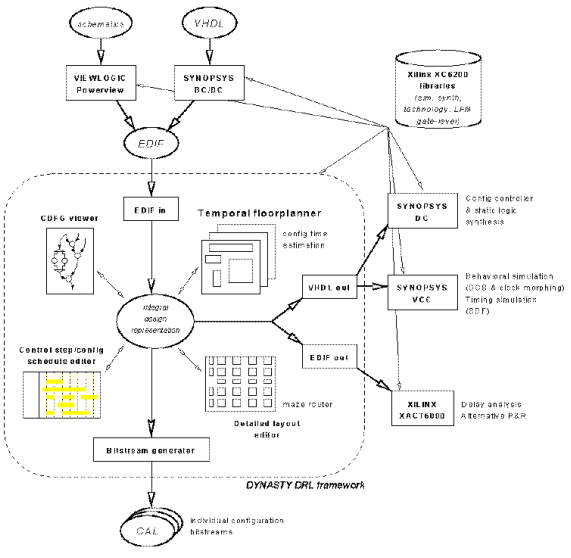

Figura 2.8: Projeto de planta-baixa temporal baseado no modelo Dynasty

Até a data de publicação do artigo, o *framework* suportava somente a família XC6200 da Xilinx de FPGAs passíveis de serem reconfigurados dinamicamente.

A descrição abaixo mostra as ferramentas do Dynasty que permitem a exploração do espaço de soluções de um projeto de sistema RTR:

- *Visualizador do grafo de dados e controle (CDFG)*: provê informação sobre comportamento do sistema, o qual será implementado durante o fluxo de projeto.
- *Navegador de estrutura e hierarquia do projeto*: fornece detalhes de todos os elementos do projeto e suas relaçãoes hierárquicas. Isto permite alocação de elementos/módulos para operadores CDFG.
- *Editor de controle de passo e escalonador de configurações*: é uma ferramenta de análise que permite ajuste e controle e escalonamento de configurações durante o espaço de soluções do peojeto.
- *Gerador de planta-baixa 3D*: permite visualização e manipulação de relacionamentos entre módulos, com o projeto de planta-baixa espacial e entre duas ou mais configurações.
- *Editor de disposição detalhada*: permite manipulação de elementos de disposição detalhados, tais como chaves lógicas e roteamento.

Os parâmetros e estruturas de uma solução em RTR podem ser diretamente manipulados via interface gráfica do Dynasty. Uma mudança em qualquer visão será propagada para representações de projeto relevantes em outras visões. Por exemplo, quando um módulo é colocado em uma configuração cujos resultados acarretam uma violação na dependência de dados do CDFG, o passo de controle e o escalonador de configurações são automaticamente recalculados para refletir a latência de projeto e a sobrecarga de configuração resultantes.

A ferramenta central do *framework* é o *3D Floorplanner*, que é um gerador de plantas-baixas. Ele utiliza visualização de projetos 2D para mostrar projetos de plantas-baixas espaciais para cada configuração, as quais podem ser inseridas no tempo (terceira dimensão) usando-se controles de navegação entre configurações. O *3D Floorplanner* usa um número de características que facilitam a manipulação de múltiplas e parciais configurações:

1. amostragem simultânea de múltiplos projetos de planta-baixa espaciais, que pode também ser utilizada para verificar conflitos de compartilhamento de recursos entre configurações.
2. estimador de tempo de configuração calcula o número de bits de configuração necessários para mudar entre duas configurações ou para o projeto inteiro.
3. é possível demarcar regiões como "não utilizáveis", o que é útil para posterior inserção de módulos, ou controle do posicionamento e do roteamento do projeto.

Com relação a simulação, o *framework* Dynasty suporta dois níveis de abstração:

1. O *3D Floorplanner* pode gerar um modelo de simulação em VHDL para o projeto RTR. O usuário pode selecionar entre duas técnicas de simulação:
   (a) O projeto finalizado pode ser importado para o XACT6000 (ferramenta de roteamento e posicionamento da Xilinx), onde um modelo de temporização detalhado pode ser gerado e simulado por ferramentas de terceiros.
   (b) Transformação de relógio (*Clock Morphing - CM*): A implementação de CM é baseada no pacote IEEE std_logic_1164, que implementa um novo sinal virtual e suporta sua propagação em bibliotecas de tecnologias-alvo.

2. Chaveamento dinâmico de circuito (Dynamic Circuit Switching - DCS): A implementação de DCS no *framework* Dynasty usa multiplexadores virtuais, chaves e demultiplexadores que são implementados como procedimentos VHDL paramétricos, em um pacote VHDL separado.

O *framework* Dynasty fornece interfaces para terceiros através de interfaces EDIF e VHDL.

A síntese automática de um controlador de configurações não é diretamente suportada pelo *framework* Dynasty. Contudo, o *3D Floorplanner* pode gerar um escalonador de configurações em um arquivo texto (nos formatos VHDL e pseudo-código), a partir do qual tal controlador pode ser construído, através da utilização de ferramentas de projeto e compilação padrões.

Módulos de projeto que não são reconfigurados durante o tempo de execução podem ser sintetizados fora do *framework* Dynasty, e então importados para dentro de posições reservadas na planta-baixa, e conectadas com as partes remanescentes do projeto através de rótulos únicos de rede.

2.4 Hardware evolutivo

2.4.1 Conceitos e histórico

Em [SIP00] os autores citam biólogos no sentido de mostrar que a natureza, com suas características pouco precisas (acaso, mutações aleatórias), pratica uma "engenharia" muito mais eficiente que o homem. Partindo desse pressuposto questionam a viabilidade de simular a natureza, utilizando computadores para resolver problemas. Esse tipo de computação foi denominada computação evolutiva.

Essencialmente, computação evolutiva é desempenhada através de programas que rodam em processadores genéricos. Contudo, com o advento dos FPGAs e mais recentemente, dos FPGAs reconfiguráveis parcial e dinamicamente, surgiu uma nova classe de sistemas, qual seja, o hardware evolutivo, ou *evolware*.

Nas seções seguintes são apresentados três sistemas que utilizam o conceito de *evolware* na resolução de problemas.

2.4.2 Exemplos de sistemas que implementam hardware evolutivo

2.4.2.1 FireFly

Um exemplo de hardware evolutivo tem sido apresentado pelos pesquisadores do Instituto Federal Suiço de Tecnologia, que chamam seu sistema de FireFly [SAN99].

O Firefly é um sistema evolutivo baseado em modelos genéticos. Ele consiste de células básicas conectadas a uma linha. Cada célula tem seu próprio estado representado por um dígito binário, e este por sua vez é alterado conforme o estado das células adjacentes, de acordo com certas regras. O ponto principal desta máquina é conseguir oscilações uniformes do estado de cada célula. A configuração inicial dos estados do sistema e das regras são gerados aleatoriamente. Depois disso, os estados das células mudam de acordo com suas regras. Após um certo número de transições, o resultado é avaliado segundo metas predefinidas. Regras de transição podem ser mudadas de acordo com os resultados obtidos.

A placa contem LEDs que indicam os estados das células, chaves para definição manual dos estados iniciais, circuitos integrados FPGAs, *display* e botão para controlar os parâmetros *time steps* e *configuration* do algoritmo de programação celular, um indicador de sincronização, um gerador de pulso de relógio ajustável manualmente, um visor LCD para mostrar valores obtidos pela evolução e um cabo para suprimento de energia. Nota-se na Figura 2.9 que o cabo de força é de fato a única ligação do FireFly com o mundo externo.

Os princípios de evolução já vinham sendo utilizados em abordagens de projeto de programas. Contudo este é um dos primeiros trabalhos que implementaram essas idéias em hardware. As maiores vantagens

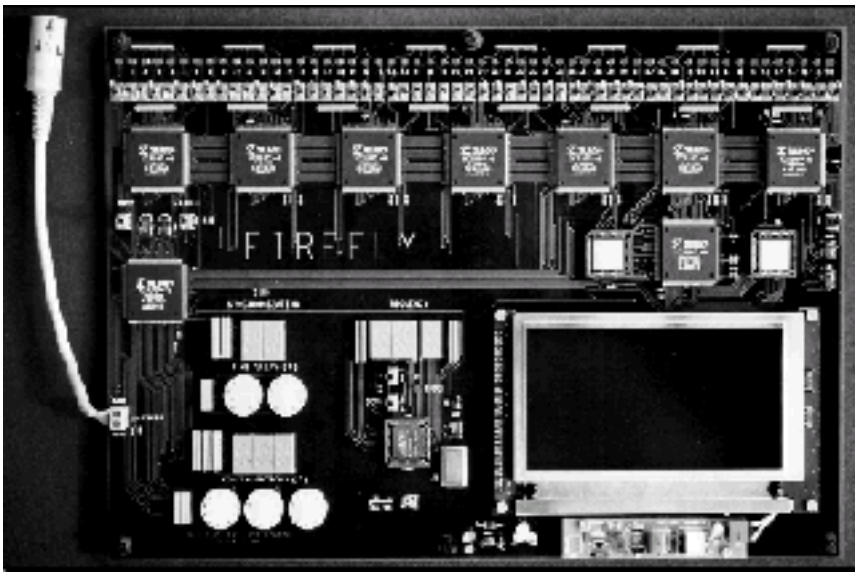

Figura 2.9: Placa " evolutiva" FireFly

de um sistema evolutivo como este é o paralelismo massivo, aproveitamento do princípio da localidade nas interações celulares e a simplicidade das células. Estas propriedades tornam quase ideal a implementação em FPGAs. Uma aceleração de três ordens de magnitude já foi alcançada comparando com implementações em programas rodando sobre GPPs.

2.4.2.2 Bio-Watch

O BioWatch é uma das aplicações que fazem parte do projeto Embryonics da Escola Politécnica de Lausanne, Suíca. Tal projeto tem como objetivo final desenvolver circuitos VLSI capazes de auto-reparar-se e auto-replicar-se. Essas duas propriedades inspiradas em seres vivos são obtidas através da transposição de certas características de organização celular em um mundo bidimensional de circuitos integrados em silício [SAN99].

O BioWatch é um organismo artificial projetado para contar minutos (de 00 a 59) e segundos (de 00 a 59); isto é, um módulo de 3600. Este organismo é unidimensional e compreende quatro células organizadas em um vetor, com conexões físicas idênticas, e um conjunto de recursos também idêntico.

A célula mais a esquerda conta dezenas de minutos, a célula a sua direita conta minutos, a seguinte conta desenas de segundos, e a que está mais a direita conta os segundos. Então, no BioWatch cada célula realiza uma de duas tarefas específicas: contador módulo-6 ou módulo-10. A organização é multicelular, como nos seres vivos, com cada célula realizando uma única função, descrita por um subprograma chamado gene da célula. Uma reconfiguração dinâmica da tarefa executada por uma das células ocorre durante a ocorrência do processo de auto-reparação do organismo artificial.

O genoma é o conjunto de todos os genes do BioWatch, onde cada gene é um subprograma caracterizado por um conjunto de instruções e sua coordenada no eixo X. Armazenando em cada célula todo o genoma, cada célula é capaz de realizar as funções de um determindo gene. Esta é outra propriedade bio-inspirada: cada célula do ser humano contém todo o genoma, apesar de que apenas parte dele é utilizado para célula realizar uma determinada função. Dependendo da sua posição no organismo, cada célula interpreta o genoma e extrai e executa os genes que a configuram. O BioWatch faz o que é conhecido na biologia como diferenciação celular (ver Figura 2.10).

A auto-reparação de um organismo permite a reconstrução parcial do dispositivo original em caso de uma falha menor. no sentido de implementar um processo de auto-reparação no BioWatch, algumas

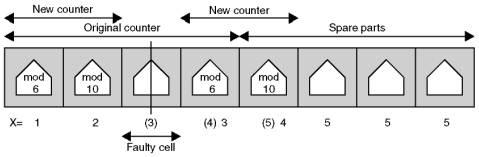

Figura 2.10: Estrutura do BioWatch

células sobressalentes são necessárias e são colocadas à direita do vetor de células. No exemplo da Figura 2.10 são usadas 4 células sobressalentes. Este processo é obtido ignorando-se (*bypassing*) a célula com falha, e deslocando para direita todo restante do vetor celular original. A nova coordenada, então definida, faz com que haja a reconfiguração dinâmica da tarefa realizada pela célula.

2.4.2.3 Artificial Hand

Diversos circuitos integrados para aplicações específicas foram desenvolvidos no Laboratório de Sistemas Evolutivos (LSE), no Laboratório Eletrotécnico de Tsukaba, Japão. Lá foram construídos diversos circuitos, tanto digitais como analógicos, para fins comerciais, dentre eles destacam-se: um CI analógico para telefones celulares, um CI para redes neurais capaz de reconfiguração autônoma, e um CI para compressão de dados para impressoras eletrofotográficas [KAJ99].

Um dos trabalhos mais impressionantes que está sendo realizado naquele laboratório, é um CI de propósito geral para hardware evolutivo que está sendo utilizado para implementar um controlador para uma mão artificial que é controlada por impulsos elétricos capturados a partir dos nervos do braço (Figura 2.11).

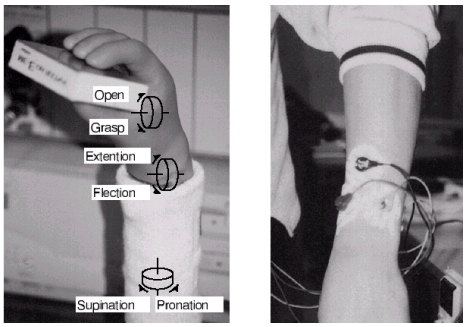

Figura 2.11: Prótese de mão que captura impulsos mioelétricos

Em condições normais, uma pessoa comum pode manipular uma mão protética com facilidade em menos de um mês de treino. Revertendo este cenário, a equipe do LSE criaram uma mão protética que se auto-adapta à pessoa, ao invés da pessoa adaptar-se à mão. A idéia foi que um controlador deveria aceitar sinais advindos de nervos do braço e mapeá-los para as ações desejadas da mão. Como esses sinais variam enormemente entre os indivíduos, é impossível projetar previamente tal circuito. Mas com um controlador de hardware evolutivo, a mão normalmente requer menos de 10 minutos para adaptar-se ao seu proprietário, através de um treinamento no qual a pessoa repete vários movimentos da mão. A Figura 2.12 mostra a arquitetura do controlador da mão protética.

Essa abordagem é chamada de *evolware on-line*, pois o circuito adapta-se a um ambiente-alvo enquanto é utilizado. Outra aplicação poderia ser, por exemplo, um robô enviado para pesquisar o solo de Júpter não teria como ser totalmente programado previamente, seria necessária sua adaptação *on-line*.

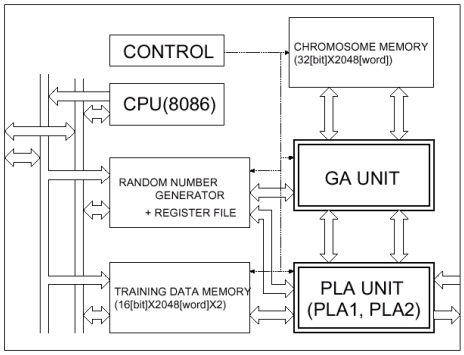

Figura 2.12: Estrutura do controlador da mão protética

2.5 Exemplo de RTR em telecom

Hwang Li é engenheiro na ATMEL. Em [LO00]relata que trabalhou em um projeto para um cliente que provê serviços de comunicação de dados. A aplicação requeria um dispositivo que suportasse três diferentes protocolos de rede, e que tivesse uma interface comum. O dispositivo seria colocado numa central, e diferentes clientes poderiam adquirir canais no sistema, e escolher o protocolo que quisesse rodar. Como cada canal pode pertencer a um cliente diferente, rodando protocolos diferentes, o sistema não poderia ser interrompido por causa de um cliente para que seu canal fosse mudado.

A solução proposta utiliza o FPGA ATMEL AT40K, que permite reconfiguração parcial através de *Cache Logic*. A lógica genérica foi arranjada na base do FPGA, enquanto a área restante foi eqüanimente dividida entre os três canais. Um arquivo de configuração é gerado para cada canal. E no sentido de poder programar os três protocolos em qualquer ordem, também é necessário um arquivo de configuração vazio para cada porção do dispositivo. Assim é possível programar o dispositivo sempre a partir de um estado conhecido.

O autor menciona alguns problemas durante a fase de projeto. Por exemplo, quando se está fazendo a disposição dos módulos há que se negociar o uso de relógios globais. A porção genérica usou um relógio diferente do relógio dos canais.

O segundo aspecto trata do roteamento. A ferramenta de posicionamento e de roteamento da ATMEL permite restringir áreas pra implementação de lógica. Isso permite designar áreas específicas para cada módulo. A única coisa que não é possível neste ponto é a restrição de roteamento a uma porção fechada. Isto resulta numa alta penalidade, que implica na diminuição de densidade. Caso densidade seja importante para o sistema, o roteamento deve ser feito manualmente.

O último aspecto refere-se ao teste dos arquivos de configuração para detectar conflitos. Primeiramente é necessário simular cada módulo individualmente, usando o fluxo normal de simulação. O passo seguinte é simular o sistema inteiro. Isto é mais fácil de ser feito depois do hardware implementado, pois essa simulação não é totalmente suportada pelo programa.

3 Tecnologias que habilitam sistemas digitais parcialmente reconfiguráveis

3.1 Hardware

A reconfiguração parcial foi implementada primordialmente pelas empresas National, Algotronix e Xilinx; que produziram as 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, 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 suportam reconfiguração parcial. A seguir será feita uma breve descrição da família AT40k da Atmel e das características da família Virtex, da Xilinx.

3.1.1 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 3.1. 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.

A série AT40K implementa *Cache Logic*, pois 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].

- Implementação da *Cache Logic*

A implementação do *Cache Logic* é conceitualmente semelhante ao cache 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,

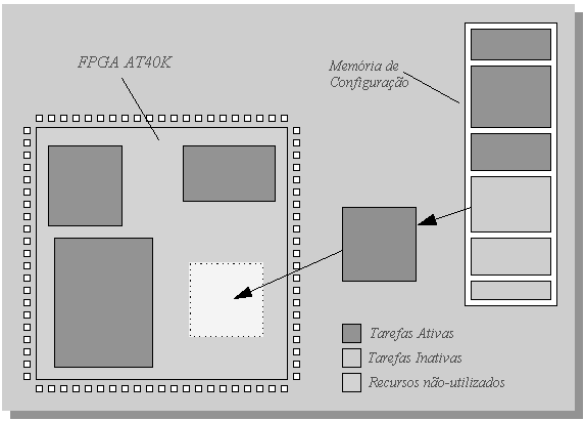

Figura 3.1: Diagrama da *Cache Logic*

tais como DRAM, EPROM eu 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. É possível compilar variações de um projeto em tempo real. 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* a predeterminada e a 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.

O segundo tipo de *Cache Logic*, dinâmica, é a base para construção de um hardware adaptativo. Cache dinâmico envonve 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.

3.1.2 Xilinx

A arquitetura Virtex suporta reconfiguração parcial [XIL99a]. Cada dispositivo Virtex contém blocos lógicos configuráveis (*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. Essas funcionalidades lógicas são configuradas através do arquivo de configuração. 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.

A memória de configuração da Virtex pode ser vista como uma matriz retangular de bits. Estes bits são agrupados em quadros verticais com um bit de largura, e se estendem do topo à base da matriz. Um quadro é a unidade atômica de configuração: é a menor porção de memória de configuração que pode ser

lida ou nela escrita.3.2.

Figura 3.2: Exemplo de colunas de configuração para o FPGA XCV50

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. A menor quantidade de informações que pode ser lida ou escrita com um único comando é um único quadro. A matriz de CLBs inteira, mais o bloco de interconexão de SelectRAM podem ser lidos ou escritos com apenas um comando. Cada bloco de conteúdo de SelectRAM deve ser lido ou escrito separadamente.

Como os membros da família Virtex possuem uma estrutura em colunas, onde os quadros podem ser lidos ou escritos individualmente, é possível reconfigurar parcialmente esses dispositivos através da modificação desses quadros no arquivo de configuração.

4 Detalhamento da arquitetura interna do FPGA Virtex XCV300

4.1 Características gerais

Como visto na Seção 3.1.2, FPGAs da família Virtex são organizados em colunas (rever a Figura 3.2). Será dado enfoque nas colunas de CLBs, por que são estas que importam para o estudo de caso em reconfiguração parcial que será esplanado na Seção 4.3. Vale relembrar que as colunas são numeradas do centro (coluna 0) para as extremidades, com colunas pares à esquerda da coluna central, e ímpares à sua direita (ainda conforme a Figura 3.2)

Cada CLB é dividida em duas fatias (*slices*), e cada fatia contém duas LUTs (F-LUT e G-LUT) Figura 4.1. As CLBs são organizadas em colunas, e cada coluna é composta por 48 quadros.

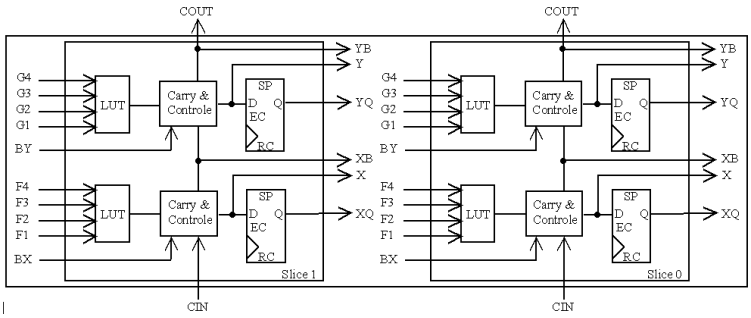

Figura 4.1: Esquema da CLB da Virtex XCV300

Apesar de encontrar-se organizado verticalmente no dispositivo, para fins de entendimento, um quadro pode ser horizontalmente visualizado, como uma matriz unidimensional de bits. Para colunas de CLBs, os 18 primeiros bits correspondem aos dois IOBs do topo da coluna, enquanto cada subgrupo de 18 bits subseqüentes corresponde às informações inerentes a uma CLB. Esses subgrupos são de número igual ao número de linhas do dispositivo, e a eles seguem-se 18 bits correspondentes aos 2 IOBs da base da coluna. A Figura 4.2 ilustra a organização de um quadro correspondente a uma coluna de CLBs. Ressalte-se que o número de bits do quadro deve ser múltiplo de 32 (tamanho da palavra, ou *wd*), e que para isso seja mantido há um número suficiente de bits de preenchimento.

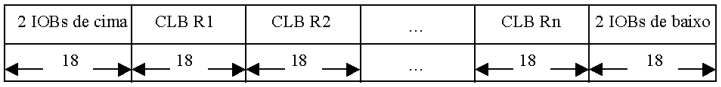

Figura 4.2: Visão horizontalizada de um quadro da Virtex

Por exemplo, no caso do FPGA XCV300 o número de bits por quadro correspondente às CLBs é 576.

A estes são somados os 36 bits referentes aos 4 IOBs, o que totaliza 612 bits. Como 612 não é divisível por 32 (daria 19,125 palavras), adicionam-se 28 bits de forma que o total passa a ser 640 bits, equivalente a 20 palavras. Como a organização dos frames da Virtex exige uma palavra de preenchimento ao final de cada quadro, o FPGA XCV300 possui 21 palavras, ou 672 bits de altura.

Ainda exercitando a matemática, e utilizando o mesmo dispositivo citado acima, podemos chegar ao número total de bits necessários para configuração total daquele FPGA. O número total de palavras por quadro já foi definido para a XCV300, e é igual a 21. O cálculo do número total de quadros é definido pela equação (4.1):

$$T_{quadros} = (IOB + RAM + RAMConnect) * 2 + (Chip_{cols} * 48) + relogio + 3 \quad (4.1)$$

onde:

- IOB é o número de quadros por coluna de IOB (54 para XCV300);
- RAM é o número de quadros por coluna de RAM (64 para XCV300);
- RAMConnect é o número de quadros por coluna de RAMConnect (27 para XCV300);
- A constante 2 signicica que há dois conjuntos de IOB+RAM+RAMConnect (à esquerda e à direita do FPGA);
- $Chip_{cols}$ é o número de colunas de CLBs do dispositivo (48 para XCV300);
- *Relogio* é a coluna central da Virtex, que tem 8 quadros;
- 3 significa que há 1 frame de preenchimento depois de cada coluna de RAM, e outro para IOB + RAMConnect + CLB;

Então, substituindo-se as constantes em (4.1) tem-se que $T_{quadros} = 2605$.

Como há 2605 quadros, e cada quadro possui 21 palavras de 32 bits, o FPGA Virtex XCV300 necessita de **1.750.560** bits para ser configurada.

Esses 1.750.560 bits são organizados dentro de um arquivo de configuração. Para estudo e entendimento dessa organização, trabalhou-se com o formato ASCII do arquivo de configuração (.RBT), que possui o número de linhas equivalentes ao número de palavras existentes no dispositivo, acrescido de 7 linhas iniciais de cabeçalho (comentários). Na Figura 4.3 pode-se ver o início de um arquivo de configuração.

```
Xilinx ASCII BitStream
Created by xecomp v1.0; comentado por Fernando G. Moraes
Designed name: SEMAFORO
Architecture: Virtex
Part: XCV300bg352
Date: 12 de Dezembro de 200 20:44
Bits: 1.750.560
111 11 11111111111111 11 11111111111     FFFFFFFF    dummy word
101 01 01010011001010 10 10101100110     AA995566    sincronização
001 10 00000000000100 00 00000000001     30008001    seleciona registrador CMD
000 00 00000000000000 00 00000000111           7     reset CRC
001 10 00000000001011 00 00000000001     30016001    seleciona registrador FLR
000 00 00000000000000 00 00000010100          14     frame lenght (14H=20 em decimal, OK)
001 10 00000000001001 00 00000000001     30012001    seleciona registrador COR
000 00 00010100000001 11 11100101101               --> variável
001 10 00000000000110 00 00000000001     3000C001    seleciona registrador MASK
000 00 00000000000000 00 00000000000           0     valor default da máscara
001 10 00000000000100 00 00000000001     30008001    seleciona registrador CMD
000 00 00000000000000 00 00000001001           9     switch clock - para jtag
001 10 00000000000001 00 00000000001     30002001    seleciona registrador FAR  - frame address
000 00 00000000000000 00 00000000000           0     start frame address
001 10 00000000000100 00 00000000001     30008001    seleciona registrador CMD
000 00 00000000000000 00 00000000001           1     WCFG - escreve configuração
001 10 00000000000010 00 00000000000     30004000    registrador FDRI
010 10 00000000000110 01 01100000111     5000CB07    CB07 data words  (51975)
.
.
.
```

Figura 4.3: Exemplo de um arquivo .RBT

O arquivo de configuração é composto por uma palavra de preenchimento seguida por uma palavra de sincronização e de alguns comandos que implementam o protocolo de configuração descrito em [XIL00]. Essa seqüência de inicialização consome 18 palavras. Após estas 18 palavras encontram-se as palavras que

compoem os quadros que formam CLBs, cujo número varia para cada dispositivo. No FPGA XCV300 são 51975 palavras. Logo depois vem 3 palavras de comando que antecedem às palavras correspondentes à primeira coluna de RAM. A seguir, mais três palavras de comando antes da segunda coluna de RAM. logo antes do final do arquivo há um quadro de preenchimento, e por último, 10 palavras para fechar o arquivo. A Figura 4.4 mostra a estrutura descrita neste parágrafo.

Figura 4.4: Estrutura do arquivo de configuração

O caso tratado na Seção 4.3 trata de reconfiguração parcial, através da leitura e gravação em arquivos de configuração. Cabe ressaltar que o cálculo de endereço de bits de LUT SelectRAM é fundamental. Esse cálculo, e aqueles necessários para chegar nele serão vistos na próxima Seção.

4.2 Endereçamento de elementos

A tarefa mais intrincada e exaustiva desta pesquisa até o momento foi o cálculo para endereçamento de LUTs no FPGA.

A seguir determinaremos a posição para leitura do bit 14 da F-LUT situada na fatia 0 ($slice = 0$, ou S0) da CLB encontrada na primeira linha (R1), e primeira coluna (C1) do FPGA XCV100 (R1C1.S0).

Para que se determine a posição inicial de uma LUTRAM, as informações iniciais devem ser:

- Linha onde se encontra a CLB que contém a LUT desejada (CLB_{row});
- Coluna onde se encontra a CLB que contém a LUT desejada (CLB_{col});
- Bit, de 0 a 15, desejado (lut_bit);
- Se o arquivo de configuração será lido ou escrito ($RW = 1$ para leitura, ou $RW = 0$ para escrita);
- Número de palavras por quadro (FL depende do dispositivo).

De posse dessas informações, primeiramente calcula-se o endereço da coluna desejada (*Major Adress*, ou MJA), conforme o algoritmo abaixo:

se ($CLB_{col} \leq \frac{Chip_{cols}}{2}$) então
$Chip_{cols} - CLB_{col} * 2 + 2$ senão
$2 * CLB_{col} - Chip_{cols} - 1$
Algoritmo para cálculo do *MJA*

Para este caso, o MJA será igual a:

$$MJA = 30 - 1 * 2 + 2 = 30$$

Depois de encontrada a coluna, deve-se achar o endereço do quadro (*Minor Adress*, ou MNA) onde se inicia a LUTRAM, através da equação (4.2):

$$MNA = lut\_bit + wd - slice * (2 * lut\_bit + 17) \tag{4.2}$$

onde a diferença entre $lut\_bit + wd$ e o restante da equação deve-se à mudança de quadro, dependendo do bit encontrar-se na fatia 0, ou na fatia 1.

Aplicando-se os valores do exemplo na equação (4.2), obtém-se:

$$MNA = 14 + 32 - 0 = 46$$

Até o momento sabe-se a coluna onde encontra-se a CLB, e o quadro onde ela começa. O próximo passo é determinar em que palavra do arquivo de configuração começa o quadro (fm_st_wd). A equação para cálculo do fm_st_wd é 4.3:

$$fm\_st\_wd = FL * (8 + MJA - 1) * 48 + MNA) + RW * FL \tag{4.3}$$

sendo que $8 + MJA - 1$ significa que está se ignorando um quadro de preenchimento (-1) e saltando 8 quadros relativos a coluna central, que é o relógio do dispositivo ($+8$). A multiplicação por 48 quer dizer que a coluna tem 48 quadros, e a multiplicação por 21 indica que cada um desses quadros tem 21 palavras. A soma disto com MNA mostra exatamente a palavra onde se inicia o quadro, caso seja uma operação de escrita. Para uma operação de leitura, há que se deslocar 32 bits, ou seja, uma palavra, para iniciar a leitura do frame. Isto acontece por que a entrada do *flip-flop* encontra-se numa palavra anterior à sua saída.

Aplicando-se os valores do exemplo temos:

$$fm\_st\_wd = 14 * (8 + 30 - 1) * 48 + 46 + 1 * 14 = 24.924$$

As penúltima informação necessária para que se encontre um bit de LUTRAM são a posição (ou altura, se o frame for visualizado verticalmente) do bit em relação ao quadro (fm_bit_idx). Para encontrar-se fm_bit_idx deve-se utilizar a equação 4.4:

$$fm\_bit\_idx = 3 + 18 * CLB_{row} - FG + RW * 32 \tag{4.4}$$

Em 4.4, multiplica-se a linha onde encontra-se a CLB que contém o bit desejado por sua altura. Diminui-se 1 caso se deseje a LUT-G 3 indica que o bit inicial da LUT fica 3 bits abaixo do início da CLB. No caso da escrita, uma palavra de preenchimento é colocada ao final de cada quadro. Contudo, para leitura, a palavra de preenchimento vem antes das palavras com informações válidas do quadro. Portanto a multiplicação por 32 em caso de leitura perfaz esse deslocamento necessário.

Mais uma vez, aplicando-se resultados obtidos anteriormente tem-se:

$$fm\_bit\_idx = 3 + 18 * 1 - 0 + 1 * 32 = 53$$

O último dado para encontrar um determinado bit de LUTRAM dentro do FPGA é sua posição em relação à palavra ($fm\_wd\_bit\_idx$), dentro do quadro. Contudo, antes é necessária a verificação da palavra relativa ao quadro, o que se obtém por 4.5:

$$fm\_wd = int(\frac{fm\_bit\_idx}{32}) \tag{4.5}$$

onde int retorna o valor inteiro do resultado da divisão.

Para o exemplo:

$$fm\_wd = int(\frac{53}{32}) = 1$$

Sabendo-se o valor de 4.5 e de 4.4, calcula-se finalmente 4.6:

$$fm\_wd\_bit\_idx = 31 + 32 * fm\_wd - fm\_bit\_idx \quad (4.6)$$

Entrando-se com os valores, finaliza-se:

$$fm\_wd\_bit\_idx = 31 + 32 * 1 - 53 = 10$$

E com isto tem-se a posição exata do bit 14 da F-LUT da CLB R1C1.S0, do FPGA XCV100, família Virtex. Para ler, contudo, uma LUTRAM completa, deve-se ler seqüencialmente os 16 bits que a compõem (0 a 15).

4.3 Estudo de caso: Application Note 138 da Xilinx

Sistemas avançados que empregam múltiplos FPGAs freqüentemente utilizam CPUs embutidas para tarefas em nível de sistema operacional, incluindo configuração dos FPGAs: a CPU monitora a operação dos subsistemas do FPGA ou provê controle em tempo-real.

Através de um mecanismo de semáforo, é possível ler e escrever na lógica configurada, o que é equivalente ao monitoramento e controle de status que a CPU provê.

Semáforos contam com a tecnologia de reconfiguração parcial da Virtex para sua implementação. Essa reconfiguração permite que um conjunto de registradores de Status/Controle permaneça visível a CPU mesmo quando diferentes projetos são carregados no FPGA [XIL99a].

Semáforos são entidades especiais em projetos VHDL que auxiliam na comunicação entre a lógica do FPGA e as portas de comunicação. Através de uma dessas portas, chamada SelectMAP, uma CPU embutida pode escrever valores no semáforo (controle) e ler valores do semáforo (status). Unidades semáforo, conectadas aos sinais lógicos em um projeto HDL, são ligadas através do roteamento normal de um FPGA, como pode ser visto na Figura 4.5.

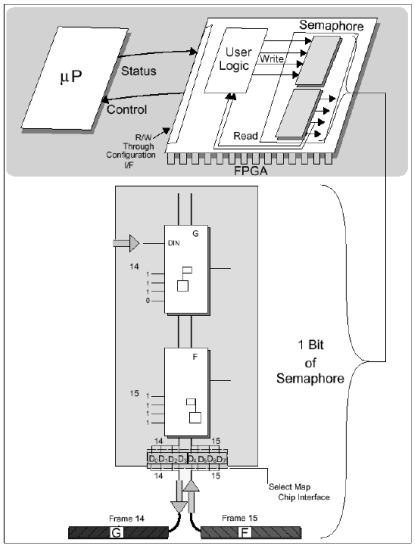

Figura 4.5: Método do semáforo

Esta Seção descreve um semáforo de 16 bits usa células de RAM16x1D (porta dupla), e o procedimento para ler e escrever de/para este módulo através de uma porta de configuração. Este conceito pode ser usado para modificar ou ler status para um CI.

O código em VHDL para o instanciamento do Módulo Semáforo em um projeto de contador simples é mostrado no algoritmo da Figura 4.6. O Módulo Semáforo em si é uma coluna de 16 células de RAM16x1D, configurada como 1x1, provendo uma interface entre a lógica interna e um hospedeiro externo, através de portas de configuração. As portas são mostradas na Tabela 4.1.

A lógica interna usa as portas mostradas para escrever e ler do semáforo. O hospedeiro externo escreve executando uma reconfiguração parcial de um quadro, modificando os bits de SelectRAM conforme necessário. Uma leitura é feita através de um *readback* parcial, e exame dos bits desejados.

```
library ieee;
USE ieee.std_logic_1164.all;

ENTITY semaphore16 IS
  PORT (
    wclk          : IN   std_logic;
    we            : IN   std_logic;
    data          : IN   std_logic_vector(15 DOWNTO 0);
    rout          : OUT  std_logic_vector(15 DOWNTO 0)
  );
END semaphore16;

ARCHITECTURE behave OF semaphore16 IS
SIGNAL logic0  : std_logic;
SIGNAL logic1  : std_logic;

COMPONENT RAM16X1D
  PORT (
    WCLK          : IN   std_logic;
    WE            : IN   std_logic;
    D             : IN   std_logic;
    A0            : IN   std_logic;
    A1            : IN   std_logic;
    A2            : IN   std_logic;
    A3            : IN   std_logic;
    DPRA0         : IN   std_logic;
    DPRA1         : IN   std_logic;
    DPRA2         : IN   std_logic;
    DPRA3         : IN   std_logic;
    SPO           : OUT  std_logic;
    DPO           : OUT  std_logic
  );
END component;

BEGIN

logic0            <= '0';
logic1            <= '1';
```

```
laco :  for i in 15 downto 0 generate
        RAMBIT: RAM16X1D ( WCLK => wclk, WE => we, D => data(i),
                DPO => rout(i), A3 => logic1, A2 => logic1,
                A1 => rout(i), A0 => logic1, DPRA3 => logic1,
                DPRA2 => logic1, DPRA1 => logic1, DPRA0 => logic0 );
 end generate laco;

END behave;

library ieee;
USE ieee.std_logic_1164.all;

ENTITY top IS
  PORT (
    clk           : IN   std_logic;
    we            : IN   std_logic;
    data          : IN   std_logic_vector(15 DOWNTO 0);
    qout          : OUT  std_logic_vector(15 DOWNTO 0)
  );
END top;

ARCHITECTURE behave OF top IS

COMPONENT semaphore16
  PORT (
    wclk          : IN   std_logic;
    we            : IN   std_logic;
    data          : IN   std_logic_vector(15 DOWNTO 0);
    rout          : OUT  std_logic_vector(15 DOWNTO 0)
  );
END component;

BEGIN

semaphore: semaphore16 PORT MAP ( wclk => clk, we => we, data => data, rout =>
qout );

END behave;
```

Figura 4.6: Semáforo em VHDL

Nome do sinal	Tipo	Descrição
xclk	input	Write Clock
we	input	Write Enable
data[15:0]	input	Write Data
rout[15:0]	output	Read Data

Tabela 4.1: Portas do Módulo Semáforo

O endereço da porta de escrita da SelectRAM dupla-porta é fixado em '1111' (G-LUT), e o endereço da porta de leitura é configurado em '1110' (F-LUT). Isto cria efetivamente duas SelectRAMs 1x1, onde cada uma é usada para enviar informações de volta e adiante, entre a lógica interna e o hospedeiro externo em uma de duas direções.

Um *flag* de atualização (chamado de *Update*) pode ser incluído para permitir a lógica interna saber quando o hospedeiro externo reconfigurou células de SelectRAM. Por exemplo este flag poderia ser configurado em '1' quando dados são escritos no FPGA através da porta de configuração, e alterado para '0' quando dados são lidos do projeto lógico.

Depois do projeto compilado, posicionado e roteado, o próximo passo é determinar a locação dos bits dentro do arquivo RBT, conforme demonstrado na Seção 4.2, que correspondem aos bits de SelectRAM [XIL00].

5 Conclusão

Nesta monografia buscou-se mostrar a possibilidade de realização de reconfiguração parcial e dinâmica em um FPGA disponível comercialmente.

Para tanto realizaram-se as seguintes ações:

1. revisão do estado da arte sobre sistemas que utilizam RTR;
2. estudo de ferramentas que facilitam seu projeto e implementação;
3. análise detalhada da estrutura interna de FPGAs da família Virtex.

A partir destas atividades, concluiu-se que dispositivos pertencentes à família Virtex de fato permitem reconfiguração parcial não disruptiva.

Essa conclusão baseia-se nos seguintes fatos:

- A estrutura interna da Virtex assemelha-se à arquitetura proposta por Comptom [COM99], com blocos lógicos homogêneos compostos por elementos que podem ser acessados individualmente (quadros organizados verticalmente, no caso da Virtex);
- De forma semelhante ao R/D FPGA de Comptom, que é organizado em linhas, a Virtex é organizada unidimensionalmente, mas na forma de colunas;
- O arquivo de configuração da Virtex é documentado (embora mal estruturado, o que complicou seu entendimento);
- A Virtex possui mecanismos elétricos que permitem reconfigurar parte do FPGA sem a necessidade de parada no sistema [XIL00].

Contudo, apesar de a reconfiguração parcial ser factível em dispositivos Virtex, esta não é uma tarefa fácil, à medida que não há ferramentas no mercado que automatizem o processo de projeto e há apenas uma disponível comercialmente que permite a implementação de sistemas RTR (JRTR).

5.1 Objetivos alcançados

O presente trabalho propôs-se originalmente a implementar o exemplo de reconfiguração parcial presente no [XIL99a]. Contudo, devido à falta de um equipamento essencial para realização de parte da tarefa, o objetivo do trabalho foi modificado.

Buscou-se então rever o estado da arte, com foco em RTR (o que difere da revisão feita durante o Trabalho Individual I [MES00]), e analisar o endereçamento de componentes de um dispositivo que presumia-se permitisse a implementação de sistemas RTR.

Com a nova revisão do estado da arte constatou-se que a idéia de RTR ainda é imatura, pois para ela não há método eficiente de projeto, nem ferramenta consolidada para sua implementação. Por outro

lado ficou caracterizado o interesse do assunto tanto no meio acadêmico quanto na indústria, pois ambos investigam a validade e a factibilidade da utilização de sistemas RTR.

A mudança de objetivo logrou sucesso, uma vez que o conhecimento mais detalhado a respeito da Virtex acarretará em economia de tempo para realização de reconfiguração parcial quando estiver disponível o equipamento necessário para tanto.

5.2 Dificuldades encontradas

Conforme já visto, a reconfiguração parcial ocorre a partir da escrita em quadros individualmente. Quando se faz necessária a mudança no valor de uma G-Lut, por exemplo, deve-se escrever em 16 quadros. Porém, essa escrita afeta todas as CLBs que situam-se abaixo e acima da CLB referente à LUT desejada. Para que seja mantida a consistência dessas outras CLBs, a alteração é feita segundo a filosofia Leitura-Modificação-Escrita (*Read-Modify-Write*, ou RMW), onde primeiro lê-se os 16 quadros (para este exemplo), então modifica-se os bits referentes à Lut desejada, e depois grava-se os quadros com as modificações e com os dados das demais CLBs mantidos.

A parte de leitura do RMW é feita através da leitura do status atual do dispositivo, através de um método de leitura do status do FPGA, chamado ReadBack. Para dispositivos da família Virtex, contudo, segundo [DS...] o ReadBack só pode ser realizado com a utilização de um cabo de programação chamado MultiLinx. E o laboratório do GAPH não possui tal equipamento, nem foi possível adquirí-lo em tempo hábil para realização deste trabalho.

Salienta-se que tal cabo para programação de FPGAs já foi encomendado, pois é fundamental para a seqüência desta pesquisa.

Ainda no campo das dificuldades, na nova proposta de trabalho a maior barreira a ser ultrapassada foi a qualidade da documentação fornecida pela empresa Xilinx. O entendimento do arquivo de configuração de um dispositivo Virtex foi tarefa realmente árdua devido a falta de clareza na documentação analisada. Acredita-se até que uma das principais contribuições deste trabalho foi destrinchar o funcionamento do arquivo de configuração da Virtex e documentá-lo, pois isto facilitará o projeto de sistemas RTR com este dispositivo.

5.3 Trabalhos futuros

Pretende-se como trabalho futuro a criação de um sistema reconfigurável parcial, remota e dinamicamente. Para tanto, algumas ferramentas de apoio deverão ser desenvolvidas, bem como um método para conexão de módulos de hardware à parte estática do sistema deve ser elaborado.

Organizando de forma mais clara tem-se que os trabalhos futuros serão os seguintes:

1. Após o estudo do endereçamento de elementos da Virtex realizado neste trabalho, é possível realizar a modificação/programação desses elementos e recálculo do CRC para validar o arquivo de configuração;

2. Desenvolvimento de uma *web-tool* para para ReadBack;

3. Modificação parcial do arquivo de configuração;

4. Consolidação dos trabalhos anteriores em uma *web-tool* para reconfiguração parcial, dinâmica e remota, para implementação do exemplo proposto em [XIL99a];

5. Estudo de uma interface para conexão de módulos de hardware ao hardware estático nos dispositivo.

Das tarefas citadas, a quinta será a que demandará maior tempo e complexidade, mas será a maior contribuição deste trabalho, pois permitirá a generalização da ferramenta desenvolvida no item 4. Esta monografia foi o primeiro passo para o desenvolvimento de um sistema que utilize RTR a ser implementado num dispositivo Virtex. E tal feito abrirá muitas perspectivas para a pesquisa em reconfiguiração parcial e dinâmica.

Referências Bibliográficas

[ALG00] ALGOTRONIX, Equipe de documentação da. **Company profile**. Disponível por WWW em http://www.algotronix.com/profile.htm (2000).

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

[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).

[BEL98] BELLOWS, Peter; HUTCHINGS, Brad. Jhdl-an hdl for reconfigurable systems. In: PROCEEDINGS OF THE IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES, 1998. **Anais...** 1998. p.124–133.

[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.

[DEH94] DEHON, André. **Dpga-coupled microprocessors**: commodity ics for the early 21st century. Disponível por WWW em http://www.ai.mit.edu/projects/transit/tn100/tn100.html (Janeiro 1994).

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

[GUC00a] GUCCIONE, Steven; LEVI, Delon; SUNDARARAJAN, Prasanna. **Jbits**: a java-based interface for reconfigurable computing. Disponível por WWW em http://www.io.com/ guccione/Papers/MAPPLD/JBitsMAPPLD.pdf (Julho 2000).

[GUC99] GUCCIONE, Steven; LEVI, Delon. The advantages of run-time reconfiguration. In: PROCEEDINGS OF SPIE - THE INTERNATIONAL SOCIETY FOR OPTICAL ENGINEERING, 1999, Washington, Estados Unidos. **Anais...** 1999. p.87–92.

[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.

[LO00] LO, Hing Kai. Info about reconfigurability. [S.l.: s.n.], 2000. (Comunicação pessoal via e-mail).

[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; MORAES, Fernando Gehm. Análise de arquiteturas e técnicas para implementação de sistemas reconfiguráveis parcial e dinamicamente. [S.l.: s.n.], 2000. (Trabalho Individual I monografia pré-requisito para dissertação de mestrado PPGCC-FACIN-PUCRS).

[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).

[SIP00] SIPPER, Moshe; RONALD, Edmund M. A. A new species of hardware. In: IEEE SPECTRUM 2000, 2000. **Anais...** 2000.

[VAS99] VASILKO, Milan. Dynasty: a temporal floorplanning based cad framework for dynamically reconfigurable logic systems. In: PROCEEDINGS OF FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS, 1999, Glasgow, Escócia. **Anais...** 1999. p.124–133.

[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.

[WIR98] WIRTHLIN, Michael J.; HUTCHINGS, Brad L. Improving computational density using run-time circuit reconfiguration. In: IEEE TRANSACTIONS ON VLSI SYSTEMS, 1998, Nova Iorque, EUA. **Anais...** 1998. p.247–256.

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

[XIL00a] XILINX, Equipe de documentação da. **Xapp153**: status and control semaphore registers using partial reconfiguration. Disponível por WWW em http://www.xilinx.com/xapp/xapp153.pdf (Novembro 2000).

[XIL00b] XILINX, Equipe de documentação da. **The jbits 2.4 sdk for virtex**. Disponível por FTP em ftp://customer:xilinx@ftp.xilinx.com/download/JBits2_4.exe (Novembro 2000).

[XIL99] XILINX, Equipe de documentação. **X6200 datasheet**. Disponível por WWW em http://dec.bournemouth.ac.uk/drhw_lib/archive/xc6200.pdf (Junho 1999).

[XIL99a] XILINX, Equipe de documentação. **Satus and control semaphore registers using partial reconfiguration**. Disponível por WWW em http://www.xilinx.com/xapp/xapp153.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).