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

Análise de Arquiteturas e Técnicas para Implementação de Sistemas Reconfiguráveis Parcial e Dinamicamente

Autor: Daniel Gomes Mesquita

Orientador: Professor Fernando Gehm Moraes

Trabalho Individual I

Porto Alegre, Setembro de 2000

Resumo

Em áreas como multimídia, telecomunicações e sistemas embarcados, onde o alto desempenho no processamento é essencial, um microprocessador convencional que possui um conjunto predeterminado de unidades funcionais não corresponderá as expectativas de desempenho desejadas. Contudo, combinando microprocessadores convencionais com lógica reconfigurável (FPGAs) é possível obter-se um produto economicamente viável e ainda assim eficiente para uma vasta gama de aplicações. Neste trabalho será apresentada uma análise de diversas arquiteturas reconfiguráveis, bem como a taxonomia desses sistemas; tudo isto com a finalidade de obter-se um ponto de partida para pesquisa na área de sistemas reconfiguráveis dinamicamente.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

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*, ou FPGA) tem 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 em tempo de execução. 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, comunicações sem fio reconfiguráveis, sistemas de computação de alto desempenho, processamento de imagens em tempo real e sinais digitais adaptáveis, produtos ao consumidor 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. 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.

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 um alto *overhead* 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.

- *Alto overhead 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. 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.

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.

A primeira parte deste trabalho propõe uma revisão do estado da arte das tecnologias reconfiguráveis e dos cenários típicos de sistemas reconfiguráveis, com a finalidade de auxiliar na decisão sobre qual abordagem será adotada no desenvolvimento de uma aplicação que utilizará hardware reconfigurável dinamicamente, voltado para área de telecomunicações.

Na seqüência serão analisadas arquiteturas que habilitam a reconfiguração parcial e dinâmica. Ênfase será dada aos FPGAs da família Virtex, fabricada pela Xilinx, sobre a qual há a demonstração de um estudo de caso.

2 Taxonomia de Arquiteturas Reconfiguráveis

Nas próximas seções serão apresentados os critérios de algumas taxonomias, segundo Bozidar [RAD98], Page [PAG97] e Sanchez [SAN99]. Será complementada a taxonomia destes sistemas com a inclusão dos quesitos reconfigurabilidade e abrangência da reconfigurabilidade. A escolha destes critérios busca a elaboração um conjunto de características de classificação cujos elementos sejam ortogonais entre si [CAL98].

2.1 Critérios de Bozidar

Bozidar Radunovic e Veljko Milutinovic propuseram uma taxonomia para arquiteturas reconfiguráveis baseada nos critérios de objetivo da arquitetura, granularidade, integração e reconfigurabilidade da rede de interconexão. A Figura 2.1 exibe no formato de árvore essa proposta de taxonomia [RAD98].

2.1.1 Objetivo da Arquitetura

Uma arquitetura reconfigurável pode ter como meta uma das duas características abaixo:

- *Tolerância a falhas*: É um dos primeiros campos onde a computação reconfigurável foi implementada. Durante a fabricação e a utilização, há uma certa possibilidade de que uma parte do circuito integrado torne-se defeituosa. Em arquiteturas clássicas uma parte defeituosa significa que o circuito integrado inteiro está inutilizado. Contudo, numa arquitetura reconfigurável tolerante a falhas, o sistema ainda poderia continuar operando, pois é capaz de detectar e corrigir certo conjunto de falhas.
- *Acrécimo de velocidade*: O uso de arquiteturas reconfiguráveis para incrementar a desempenho de sistemas é uma tecnologia que emergiu rapidamente, principalmente pela possibilidade de implementação de algoritmos consumidores de tempo diretamente em hardware.

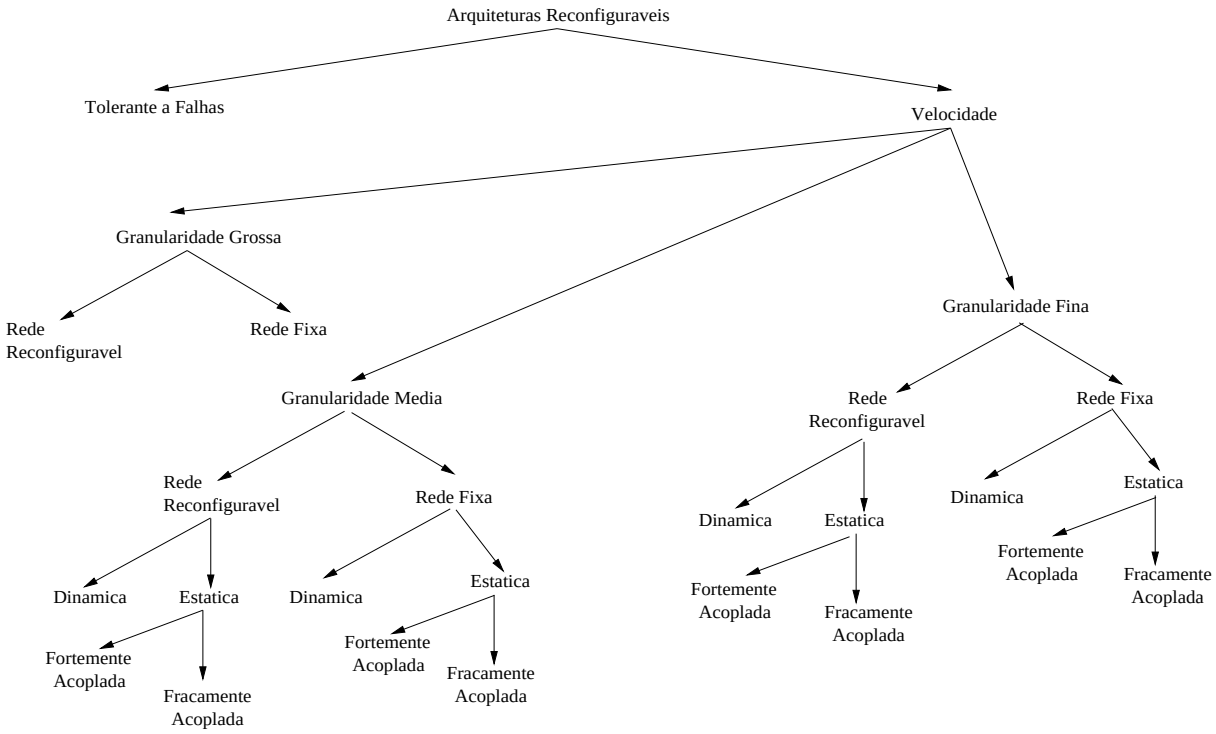

Figura 2.1: Taxonomia de Bozidar

2.1.2 Granularidade

Granularidade pode ser definida de várias maneiras, como por exemplo o número de funções booleanas que um bloco lógico pode implementar, o número de portas $NAND$ de duas entradas equivalente, o número total de transistores, a área normalizada total, ou pelo número de entradas e saídas[ROS93]. Bozidar, et al. tratam granularidade como o menor bloco do qual um dispositivo reconfigurável é feito [RAD98].

- *Granularidade fina:* É o caso em que o bloco lógico mais simples contém poucas funções booleanas. A maioria dos atuais sistemas reconfiguráveis possuem granularidade fina. Seus blocos são ligados através de uma rede de interconexão reconfigurável.

- *Granularidade Média:* Sistemas com esta granularidade são desenvolvidos para aumentar a velocidade de aplicações com cálculos intensivos do mesmo tipo. Para este propósito possuem células básicas relativamente grandes, capazes de fazer partes significantes dos cálculos. O mais freqüente é o caso de uma simples Unidade Lógica e Aritmética (ULA), capaz de realizar operações aritméticas com tipos básicos de dados. Esses blocos básicos são conectados com uma rede reconfigurável que permitem formar diferentes estruturas lógicas.

- *Granularidade grossa:* Essa granularidade acontece quando o bloco lógico mais simples é

baseado em microprocessador completo, com ou sem aceleradores especiais.

2.1.3 Integração

Sistemas reconfiguráveis também podem ser classificados pela forma com que são configurados. Sistemas *dinâmicos* são aqueles que não são controlados por nenhum dispositivo externo [SAN98]. São bio-inspirados, sistemas evolutivos onde uma lista de configurações possíveis não existe.

Por outro lado, existem sistemas que recebem uma configuração de uma lista finita de possíveis configurações. Nessas arquiteturas pode ser feita uma distinção entre a proximidade com que interagem com o processador, originando a classificação *fortemente acoplados* e *fracamente acoplados*.

- *Sistemas Dinâmicos*: O melhor exemplo para essa característica é a simulação de sistemas vivos, por exemplo, implementação de algoritmos genéticos.
- *Sistemas estáticos, fortemente acoplados*: Unidades de reconfiguração atuam como unidades de execução do processador, e podem manipular dados diretamente de seus registradores.
- *Sistemas estáticos, fracamente acoplados*: Unidades de reconfiguração são anexadas ao sistema como co-processadores, comumente em uma placa em separado.

2.1.4 Reconfigurabilidade da rede externa de interconexão

Este quesito trata da rede de interconexão entre unidades reconfiguráveis, que pode ser fixa ou reconfigurável.

- *Arquitetura com rede externa reconfigurável*: Uma rede reconfigurável conecta todas as unidades reconfiguráveis, criando uma grande unidade reconfigurável virtual, garantindo uma maior escalabilidade para o sistema.
- *Arquitetura com rede fixa*: É uma arquitetura mais barata e mais simples. Pode não ser adequada para aplicações com computações intensivas, mas é suficiente para a maioria das aplicações.

2.2 Critérios de Page

Page considera o hardware dinamicamente reconfigurável como a tecnologia que impulsionará o desenvolvimento de sistemas que unam as características de processadores de propósito geral (GPPs) e a flexibilidade de FPGAs, em especial os *Dinamically Programmable Gate Arrays* (DPGAs). Esse autor classifica arquiteturas reconfiguráveis quanto sua atuação como coprocessadors, a respeito da arquitetura da memória, e sobre a execução dos programas pelo sistema reconfigurável[PAG97].

2.2.1 Coprocessamento

Levando-se em consideração um DPGA e um GPP como descritos acima, existe diversas formas de usá-los em um sistema. Uma forma de classificar esses sistemas é através dos diferentes níveis de interação entre os processadores:

- *Coprocessamento*: O microprocessador lança uma instrução, ou seqüência de instruções que pode ser detectada e interpretada pelo DPGA. Na realidade o DPGA atuaria como um coprocessador.
- *Chamada de procedimento remoto*: O microprocessador lança uma instrução, ou seqüência de instruções que é interpretada pelo DPGA como uma chamada de procedimento remoto (*Remote Procedure Call - RPC*). É semelhante ao coprocessamento, exceto pelo fato que é uma interface mais poderosa que usa sincronização explícita sempre que necessário.
- *Modelo Cliente-Servidor*: O algoritmo do DPGA é um processo servidor que atua de forma semelhante ao mecanismo do *RPC,* mas onde comunicações podem chegar de qualquer processo que esteja sendo executado pelo microprocessador.
- *Processos paralelos*: Os processos que são executados pelo DPGA são independentes daqueles que são executados pelo GPP. A comunicação entre processos pode acontecer a qualquer momento, via troca de mensagens.

2.2.2 Arquitetura de memória

O programa sendo executado no coprocessador DPGA normalmente precisará de certa quantidade de variáveis e de armazenamento temporário para sua operação. Alguns raros sistemas podem, contudo, consistir inteiramente de lógica combinacional. Page sugere três diferentes modelos de memória:

- *DPGA sem acesso a memória externa*: Em alguns casos o algoritmo executado no DPGA pode operar sem memória externa. É uma situação aceitável somente quando o DPGA necessita de poucos estados para sua operação.
- *DPGA compartilha memória com o microprocessador*: O DPGA pode usar qualquer memória associada ao barramento por ele compartilhado. Contudo isto gera sobrecarga de tempo e espaço para que seja negociada a arbitragem de acesso a memória. Pode ser necessária a existência de um controlador de endereçamento, o que tenderá a reduzir a velocidade de acesso a memória para processamento intensivo de dados.
- *DPGA com memória local própria*: O algoritmo da parte reconfigurável é organizado com memória privada, consumindo o mínimo de ciclos de relógio para cada dado, aumentando o desempenho durante um processamento. Contudo pode haver duplicação de dados com relação ao GPP. Por este motivo é também necessário um controle de coerência e um meio para troca de mensagens.

2.2.3 Forma de operação da memória local

Page salienta que os modelos de operação de memória não são necessariamente mutuamente exclusivos, isto é, não são ortogonais.

- *Memória de bloco*: A memória local é suficientemente grande para conter um conjunto de dados completo para o processamento. Por exemplo este poderia ser um quadro de vídeo completo a ser processado por um algoritmo de compressão de imagens.
- *Memória de fila*: A memória age como uma fila enquanto há troca de dados e resultados com o microprocessador. Como exemplo tem-se o armazenamento de uma região completa de suporte para um filtro de tempo real.
- *Cache*: A memória local pode rodar como uma cache sobre uma estrutura de dados maior controlada pelo GPP.

2.2.4 Execução dos programas

Existe também diversas formas de os programas serem programados no DPGA. Estes modelos para suportar execução de programas no DPGA podem explorar diferentes partes da relação *custo/desempenho* das implementações.

- *Hardware puro*: O algoritmo é convertido (por síntese para hardware) em uma descrição de hardware que é carregada no DPGA.
- *Microprocessador de aplicações específicas*: O algoritmo é compilado em um código de máquina abstrato, para um processador abstrato (*Application Specific Instruction Set Processor - ASIP*). Os dois são então co-otimizados para produzir a descrição de um processador de aplicação-específica e o código de máquina para ele. A descrição do microprocessador pode então ser compilada em um DPGA.
- *Reutilização seqüencial*: O algoritmo pode ser muito grande para ser implementado no DPGA, ou por razões de engenharia ou econômicas divide-se o algoritmo em partes, de tal forma que ele rode parcialmente, usando a capacidade de reconfiguração dinâmica do DPGA. Os ganhos relacionados com a reutilização do hardware devem ser balanceados com o tempo que é gasto com a reconfiguração.
- *Uso múltiplo simultâneo*: Se os recursos do DPGA são grandes o bastante, é possível haver um número de algoritmos co-residentes, e cada um deles pode interagir separadamente com o processador.
- *Uso sob demanda*: Existe a possibilidade de cada sistema complexo ser construído onde o hardware não existe ao mesmo tempo, mas cuja demanda de tempo-real do sistema dita qual hardware deve ser construído e qual deve ser destruído. Há uma analogia razoável com sistemas de memória virtual, e por isto esse esquema pode ser chamado de "hardware virtual".

2.3 Critérios de Sanchez

2.3.1 *String* de Configuração

Para Sanchez et. al, existe uma distinção entre dois modos de configurabilidade: uma estática, onde a *string* de configuração de um FPGA é carregada somente uma vez antes da execução de uma tarefa, e o FPGA será reconfigurado somente depois desta execução; e outra dinâmica, onde a *string* de configuração do FPGA pode mudar a qualquer momento [SAN99].

- *Sistema estático*: Possui pouca flexibilidade. A configuração do sistema ocorre antes do início da execução da tarefa a que se propõe. Durante toda execução o FPGA permanece inalterado. Após, pode ser reconfigurado totalmente. É o mais comumente utilizado.
- *Sistema dinâmico*: Envolve uma string de configuração que pode mudar durante a execução de uma tarefa pelo FPGA. Tem como objetivos adaptação dinâmica a mudanças de especificação, bem como manipulação de especificações incompletas.

2.4 Principais arquiteturas classificadas pelos critérios vistos

O principal propósito desta Seção é comparar, através da Tabela 2.1, algumas das mais importantes arquiteturas na área de computação reconfigurável. Foram utilizados os critérios analisados previamente para prover a classificação, por serem aqueles ortogonais o bastante para permitir clareza na comparação entre os sistemas. Os detalhes referentes a cada arquitetura serão vistos no próximo Capítulo.

Critério / Sistema	Grão	Integração	Rede externa	Coprocessamento	Memória / Arquitetura	Memória / Operação	Execução dos programas	Reconfiguração
DECPeRLe [PAT89]	fino	fracamente acoplado	fixa	coprocessador	compartilhada	n/a	hardware puro	estática
DISC [WIR96]	fino	fortemente acoplado	fixa	processador	compartilhada	cache	reutilização	dinâmica
DPGA [DEH94]	médio	fortemente acoplado	reconfigurável	coprocessador	própria / local	cache	uso múltiplo sim.	dinâmica
FIPSOC [SID99]	fino	fortemente acoplado	fixa	processo paralelo	compartilhada	cache	μP de aplicação	dinâmica
FireFly [EDU99]	fino	dinâmico	fixa	processador	própria / local	bloco	hardware puro	dinâmica
Garp [TIM00]	médio	fortemente acoplado	fixa	coprocessador	compartilhada	fila	μP de aplicação	dinâmica
PRISM [ATH93]	fino	fracamente acoplado	fixa	cliente / servidor	compartilhada	fila	μP de aplicação	estática
RAW [ELL97]	grosso	fracamente acoplado	reconfigurável	coprocessador	própria / local	bloco	μP de aplicação	estática
RPM-2 [DUB98]	fino	fracamente acoplado	fixa	processador	própria / local	bloco	hardware puro	estática
Splash[GOK94]	fino	fracamente acoplado	fixa	processador	compartilhada	fila	μP de aplicação	estática
Splash2 [ARN92]	fino	fracamente acoplado	reconfigurável	RPC	própria / local	bloco	μP de aplicação	estática
SPYDER[EDU99]	fino	fortemente acoplado	fixa	coprocessador	compartilhada	n/a	μP de aplicação	estática
Transmogrifier [DAV98]	fino	fracamente acoplado	fixa	cliente / servidor	própria / local	bloco	μP de aplicação	estática
Trumpet [STY99]	fino	fortemente acoplado	fixa	processo paralelo	própria / local	cache	μP de aplicação	dinâmica

Tabela 2.1: Classificação de sistemas reconfiguráveis

3 Análise de arquiteturas reconfiguráveis

As arquiteturas reconfiguráveis podem ser analizadas temporalmente, em função dos problemas a que se dispuseram resolver. A partir do amadurecimento da tecnologia habilitadora para esses sistemas (FPGAs), alguns centros de pesquisa criaram as primeiras arquiteturas reconfiguráveis, com o intuito principal de aumentar o desempenho de algoritmos que até então eram executados em software. Dentro desta primeira geração estão projetos como DECPeRLe, PRISM e Splash. Alguns sistemas mais modernos ainda utilizam essa abordagem, como o Transmogrifier-2, o RPM-2 e o SPYDER.

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

Em função disso novas propostas de arquiteturas surgiram. O problema de comunicação entre $_{\mu}P$ e FPGAs contou com o avanço da tecnologia habilitadora para ser resolvido. Com o aumento do número de transistores por circuito integrado (CI) tornou-se possível o desenvolvimento de um sistema composto por $_{\mu}P$, FPGA(s) e memória em um único CI (*System-on-Chip*, ou simplesmente SoC). Foram desenvolvidos SOCs de granularidade baixa (FIPSOC, TRUMPET) e com granularidade média (GARP, RAW).

Outro avanço tecnológico ocorrido foi a possibilidade de reconfiguração dinâmica. Isto permitiu que as arquiteruras em questão pudessem ser reconfiguradas sem que precisassem parar totalmente de desempenhar suas funções. Essa reconfiguração dinâmica pode ser realizada por chaveamento de contexto, como o que ocorre com os sistemas derivados do DPGA e com o Trumpet, ou por reconfiguração parcial, como no caso do DISC, FireFly, FIPSOC e Garp. Splash2 apresenta uma reconfiguração alternativa, na sua rede de interconexão externa.

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

A Figura 3.1 ilustra essa análise, mostrando a evolução dos sistemas reconfiguráveis nos últimos 15 anos.

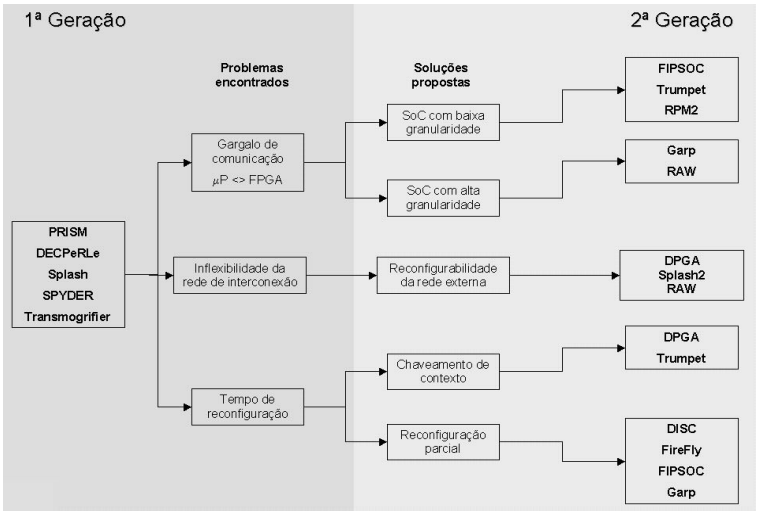

Figura 3.1: Evolução das arquiteturas reconfiguráveis

Nas próximas Seções os sistemas citados na Seção 2.1 serão analisados levando-se em consideração o objetivo da arquitetura, principais características, seus pontos fortes e suas desvantagens.

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

3.1 DECPeRLe

DECPeRLe é um projeto desenvolvido pelo *Paris Research Lab* [BER89]. A placa DECPeRLe é composta por dezesseis FPGAs X3090 da Xilinx, formando uma grade quadrada. Cada linha e coluna desta grade é interconectada com um barramento de 16 bits. A grade é conectada ao computador hospedeiro de forma similar às memórias RAM; em outras palavras, fora os barramentos de entrada e saída, há um barramento de *download*, que o usuário utiliza para descarregar configurações na placa. Além disto, a grade é conectada à memória local e a outros dispositivos através de conectores externos de 32 bits. O sistema inteiro funciona como uma memória ativa programável. A estrutura genérica desta arquitetura é vista na Figura 3.2.

O sistema tem sido testado em diversas aplicações, tais como aritmética de inteiros longos, criptografia RSA, biologia molecular, redes neurais, compressão de vídeo, física de alta energia, etc.

Assim como outros sistemas baseados em FPGAs, DECPeRLe-1 tem uma velocidade de reconfiguração lenta. Ele pode ser reconfigurado por inteiro 50 vezes por segundo. Apesar de o sistema compreender 16 FPGAs independentes, ele não pode ser reconfigurado parcialmente.

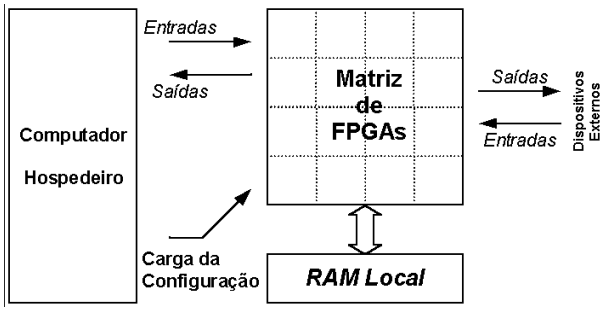

Figura 3.2: Arquitetura genérica de um DECPeRLe

O potencial gargalo de memória é aliviado pela memória RAM distribuída. Um total de 4Mb de RAM é composto por quatro bancos independentes com barramento de 32 bits. Todas as aplicações que utilizam RAM rodam em torno de 25MHz, resultando um uma largura de banda de 100Mb/s.

O DECPeRLe-1 é programável em C ou C++. Apesar destas linguagens serem convenientes para os desenvolvedores de software, a metodologia em si é similar para os projetistas de circuitos. Depois de escrever expressões e particioná-las em porções que serão mapeadas no sistema, o usuário tem que especificar informações de roteamento e localização. Portanto, esta abordagem ainda carece de automação.

3.2 DISC

A principal idéia desta abordagem é a reconfiguração parcial. O processador tem um conjunto de instruções dinâmico [WIR96]. Todavia, este o conjunto é normalmente maior do que a unidade de configuração disponível é capaz de manipular. Então, um algoritmo de cache é implementado; independentemente de quando uma instrução seja requisitada, ela é colocada no lugar da última instrução utilizada. Esta técnica reduz significantemente o tempo de reconfiguração, e o fato de que somente a instrução requisitada é carregada de cada vez, é possível buscar antecipadamente a próxima instrução (*prefetching*).

O processador consiste em um controlador global e módulos de instruções adaptadas. O controlador global é composto de registradores de propósito geral e sinais de controle, e supervisiona a busca e execução de intruções. A Figura 3.3 mostra a estrutura geral da arquitetura.

Módulos de intruções adaptadas contém as instruções. São projetados para prover todas informações necessárias para cada instrução, sem levar em consideração sua localização física. Isto aumenta o problema de mapeamento de instruções, uma vez que a biblioteca de instruções tem que possuir o mapeamento de todas instruções para todos os locais possíveis. De outra forma, o uso total de módulos de instruções adaptadas será baixo.

Um conjunto de ferramentas de software foi desenvolvido para esta plataforma. A plataforma de desenvolvimento usa um compilador C baseado no MIPS, simulando registradores de propósito geral para compensar a abordagem baseada em acumuladores do DISC. Operadores

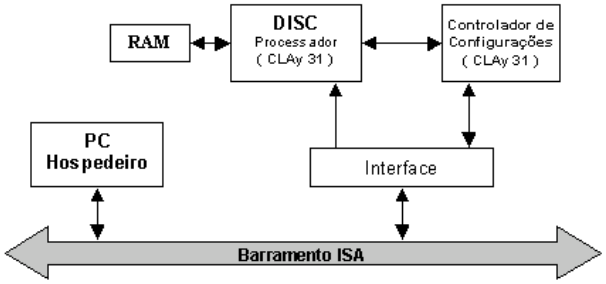

Figura 3.3: Estrutura geral do sistema DISC

C padrão, bem como rotinas de hardware definidas pelo usuário são mapeadas para instruções adaptadas. Um depurador e um montador dinâmico também foram desenvolvidos.

Esta plataforma é atualmente implementada numa tecnologia de integração de média escala; portanto ainda não pode ser demonstrado seu potencial total. DISC também sofre com o alto número de acessos à memória. Assim como outros sistemas reconfiguráveis, sua deficiência está na falta de uma ferramenta de software integrada para geração de código de hardware e software. Apesar disso, o conceito tem mostrado aumento de velocidade significante para certas aplicações, e a idéia apresentada nesta abordagem direciona para amplas áreas de pesquisa futuras.

3.3 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 3.4. 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.

Ao custo de um tamanho maior de elementos e interconexão, o DPGA serve como um FPGA de múltiplo-contexto. 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. Dando a cada elemento uma pequena cache de configuração estreitamente acoplada a ele, consegue-se efetivamente uma taxa de reconfiguração alta.

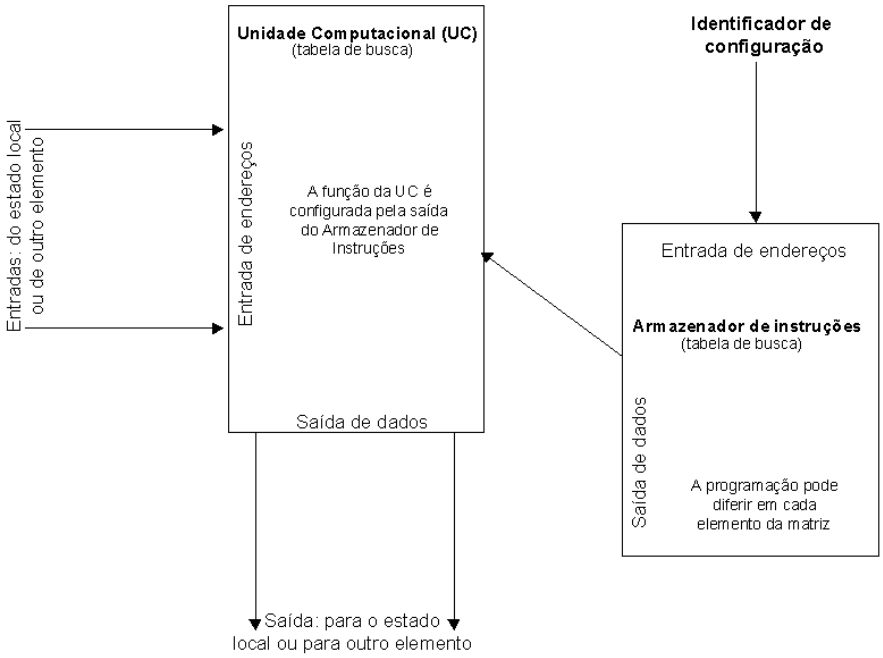

Figura 3.4: Elemento da matriz DPGA

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.

3.4 FIPSOC

FIPSOC é um dispositivo do tipo *Sistem-on-Chip (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 3.5. 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 células digitais de macros (*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*) 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 $_{\mu}P$ ou pelo FPGA [SID99].

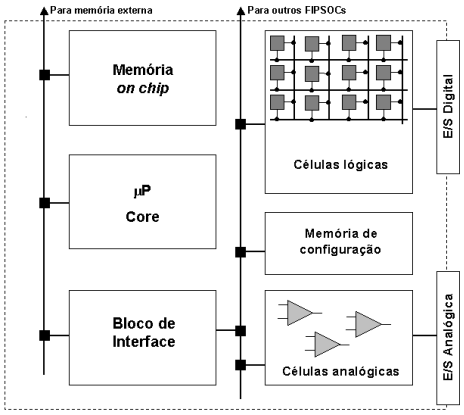

Figura 3.5: Estrutura do FIPSOC

A configuração do circuito integrado é armazenada nos bits de RAM estática. Cada característica programável é suportada por dois outros bits de recuperação, que são mapeados no espaço de endereços do microprocessador. Pode ser feito o *dowload* de uma nova configuração para a memória de recuperação enquanto 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 $_{\mu}P$. A reconfiguração dinâmica pode ser aplicada sobre uma única célula programável, sobre um conjunto selecionado delas, ou sobre toda a lógica reconfigurável.

3.5 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 no modelo celular autônomo. 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 *configurations* do algoritmo de programação celular, um indicador de sincronização, um gerador de pulso de *clock* ajustável manualmente, um *display* LCD para mostrar valores obtidos pela evolução e um cabo para suprimento de energia. Nota-se na Figura 3.6 que o cabo de força é de fato a única ligação do FireFly com o mundo externo.

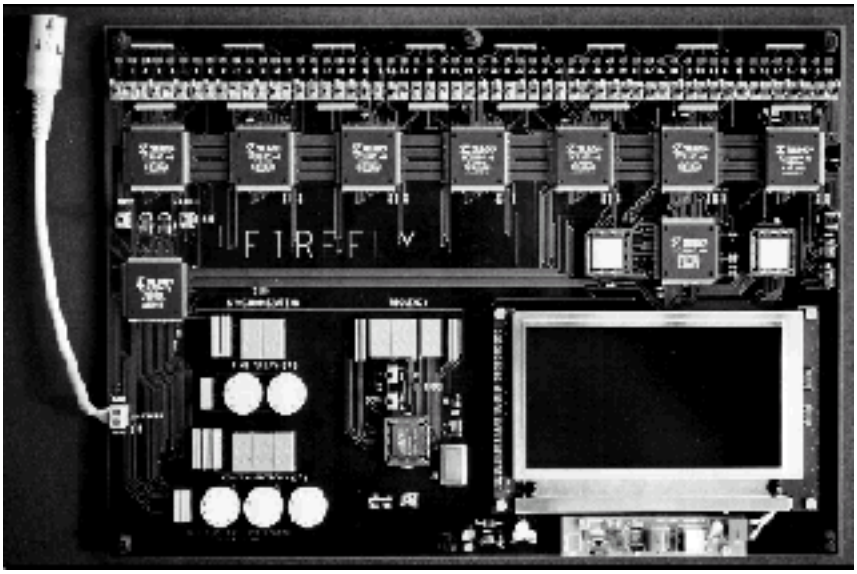

Figura 3.6: Placa " evolutiva" FireFly

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

3.6 Garp

Garp é um processador baseado no conjunto de instruções do processador MIPS-II. Foi projetado com o intuito de acelerar laços de programas de propósito geral [CAL00]. Garp atua como um coprocessador junto ao MIPS, e consiste em uma matriz bidimensional de blocos lógicos

programáveis (*Configurable Logic Blocks* - CLBs), interconectadors por fios programáveis. A célula lógica básica do vetor reconfigurável é uma unidade de 2 bits com quatro entradas de 2 bits cada. A matriz tem pelo menos 24 blocos lógicos em uma coluna, dos quais 16 blocos a partir da primeira posição são conectados ao barramento do processador. A configuração da matriz é carregada da memória. Uma vez que um tempo significante é necessário para carregar a configuração inteira, muitas abordagens tem sido tentadas para encurtar a latência da reconfiguração. Uma dessas é a abilidade de reconfigurar parcialmente o vetor. Uma unidade de memória cache é embarcada junto ao sistema, e armazeana as configurações usadas recentemente para um rápido recarregamento. Na Figura 3.7 pode ser observado um esquema da matriz Garp, onde é mostrado o caminho de dados reconfigurável. Barramentos de memória provêem uma reconfiguração com alta largura de banda e alta taxa de transferência de dados entre a matriz e a memória.

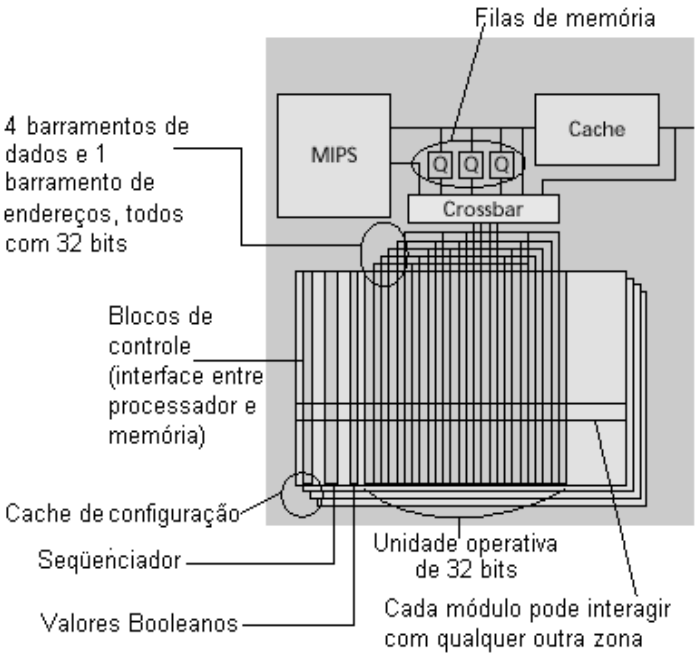

Figura 3.7: A Matriz Garp

Um suporte de software em baixo nível foi introduzido através de um conjunto de instruções de controle. Como diferentes configurações da matriz requerem diferrentes tempos de execução, uma instrução que inicia a execução contém o parâmetro de contagem de relógio, que especifica o comprimento da execução no processador em ciclos de relógio. A plataforma de desenvolvimento de software consiste de um compilador C e uma linguagem de configuração de matriz, que é utilizada para descrever a configuração a ser carregada na matriz do Garp.

Seguindo a idéia de unidades reconfiguráveis fortemente acopladas a um processador, o projeto Garp coloca um microprocessador completo com uma unidade reconfigurável num mesmo circuito integrado, e demonstra o incremento no desempenho se comparado a um microproces-

sador convencional. Seu problema é o custo envolvido na programação do sistema, que ainda é muito alto.

3.7 PRISM

A abordagem do projeto PRISM (*Processor Reconfiguration through Instruction-Set Metamorphosis*) contempla um compilador que recebe um programa de entrada e gera como saída uma imagem de hardware e software [ATH93]. A imagem de hardware consiste nas especificações físicas usadas para programar uma plataforma reconfigurável. A imagem de software, de forma semelhante a uma imagem executável gerada por um compilador convencional, é um código de máquina pronto para execução em um processador junto com o código que integra o elemento recém sintetizado dentro do fluxo de execução de intruções. A Figura 3.8 ilustra os relacionamentos entre esses componentes no PRISM.

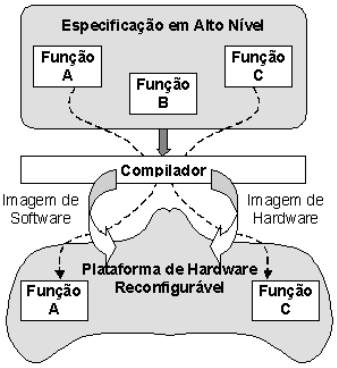

Figura 3.8: Arquitetura reconfigurável PRISM

A base da arquitetura PRISM é uma placa que contém quatro FPGAs 3090 da Xilinx, controlados por um processador Motorola 68010. No topo desta plataforma de hardware há um compilador C com um conjunto de ferramentas que transformam parte do código C em implementação de hardware. Na primeira parte da compilação o sistema separa porções do programa que serão candidatas a serem implementadas em software e aquelas que serão sintetizadas em hardware. Mas cabe ao programador a tarefa de analizar quais dos candidatos serão sintetizados de fato. Os segmentos de código selecionados para hardware são transformados num grafo de fluxo de dados. Para cada elemento desse grafo o compilador encontra uma implementação comportamental numa biblioteca. O passo seguinte é transformar a descrição comportamental do grafo num conjunto de expressões Booleanas, otimizadas para velocidade e espaço no circuito integrado. Outra tarefa é unir o projeto de hardware gerado com o restante do código. Isto pode ser feito pelo compilador, através da inserção de chamadas de funções da biblioteca para carregamento e inicialização de FPGAs. Depois que o processador avalia uma instrução sintetizada, ele move os operandos apropriados para o FPGA e quase imediatamente coleta os resultados.

Este projeto foi um dos primeiros a apresentar as possibilidades de reconfiguração. Apesar da lentidão da reconfiguração que apresenta, o sistema tem obtido acelerações significantes, na ordem de 50 vezes em certas aplicações. Como a maioria das outras arquiteturas reconfiguráveis, o PRISM carece de ferramentas de automação, principalmente para mapear o algoritmo em hardware, diminuindo a interação humana no sistema.

3.8 RAW

O microprocessador RAW consiste de um conjunto de unidades de processamento replicadas e altamente interconectadas, conforme pode ser observado na Figura 3.9 [WAI97]:

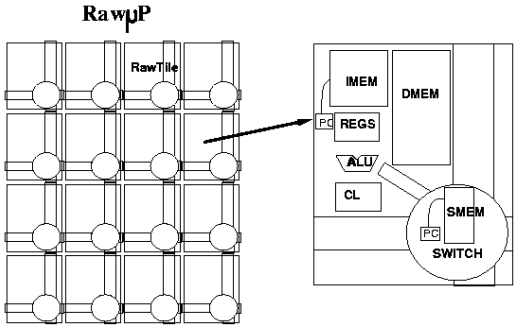

Figura 3.9: Composição do $_{\mu}P$ RAW

Cada uma dessas unidades contém um processador simples, semelhante a processadores RISC, uma porção de memória para instruções e dados, lógica configurável, e chaves programáveis.

A arquitetura RAW tem duas partes reconfiguráveis. A primeira na camada mais baixa: uma lógica reconfigurável de granularidade fina em cada unidade. A outra parte é a rede que interconecta essas unidades. RAW usa software para implementar operações como renomear registradores, escalonamento de instruções e verificação de dependências. Esta abordagem diminui o suporte de hardware para estas operações, o que disponibiliza mais área do circuito integrado para memória e lógica, resultando num *clock* mais rápido, e reduz a verificação da complexidade do circuito integrado. A granularidade do RAW é mais grossa que a maioria dos sistemas baseados em FPGAs, pois o nó corresponde a um μP, e não a LUTs. Ainda comparando, RAW tem um tempo de configuração menor, atrasos de propagação melhores para operações comuns, requisitos de roteamento mais baixos, e menor área por operação semelhante a operações de ULAs. Contudo, RAW pode ainda manter o mesmo nível de paralelismo de granularidade fina que um sistema FPGA.

O sistema inteiro é distribuído sem a necessidade de uma unidade de controle central. Cada unidade é independente com suas próprias memórias de dados, intruções e chaves - sem um gargalo potencial de acesso à memória. A maior parte dos recursos do RAW estão sob o controle de um compilador. Ao invés de construir uma lógica especializada para cada operação

dinâmica, tal como renomear registradores, ou prever saltos, a arquitetura RAW introduz um enfoque muito mais simples. Conseqüentemente o compilador é muito mais complexo, e tende a eliminar intervenções do usuário durante o tempo de compilação. Contudo, ainda há a necessidade de implementação de um compilador que manipule uma quantidade maior de programas, e explore agressivamente a capacidade de paralelismo de RAW.

3.9 RPM-2

RPM (*Rapid Prototyping Engine for Multiprocessor Systems*) foi desenvolvido para emular sistemas multiprocessadores. RPM-2 consiste de 9 processadores Sparc conectados a um *Futurebus* mais *backplane* [DUB98]. A Figura 3.10 mostra um diagrama de blocos de uma das placas. O barramento transfere pacotes entre os nodos processadores. A interface do barramento consiste de um CI *Newbridge Life*, o qual implementa o protocolo de barramento, e um conjunto de transceptores de barramento. 8 FPGAs 4013 da Xilinx controlam as três memórias de cada placa. As portas SCSI e serial permitem cada placa ser configurada como um nodo de E/S (Entrada/Saída). A interface serial RS232 é utilizada para depuração.

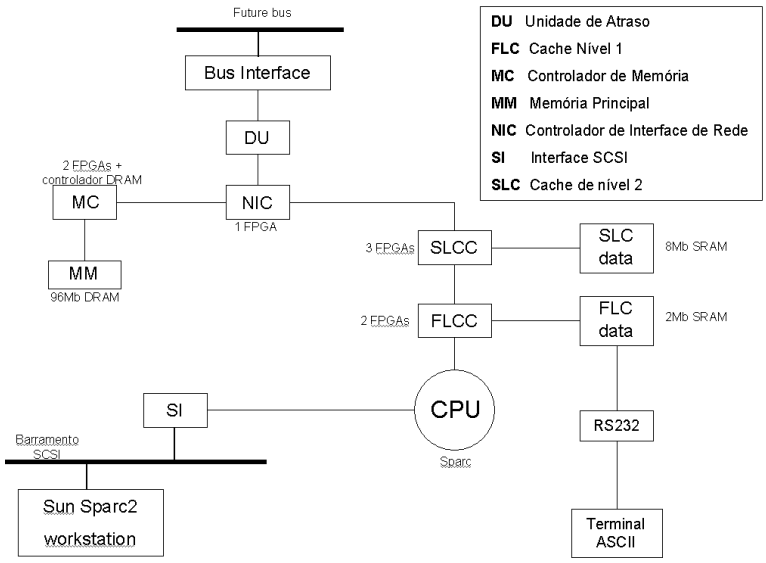

Figura 3.10: Diagrama de blocos do nodo processador RPM-2

RPM-2 apresenta grande flexibilidade e eficiência com relação a simuladores em software, podendo emular diferentes arquiteturas DSM (*Distributed Shared Memory*), como COMA, memória virutal compartilhada, DSMs híbridas e CC-NUMAS. Todas essas máquinas podem ser comparadas no mesmo substrato de hardware.

Apesar disto, os protótipos de hardware que podem ser construídos a partir de RPM-2 tem algumas limitações básicas:

1- Não é possível prototipar sistemas com caches de três níveis, ou sistemas com caches compartilhadas;

2- Não é possível prototipar sistemas com mais de oito processadores;

3- Não é possível implementar interconexões específicas;

4- Protótipos RPM não são apropriados para avaliar os efeitos da arquitetura de um processador no desempenho do sistema multiprocessador.

RPM não pode prototipar quaisquer arquiteturas multiprocessadoras, mas é muito mais flexível que protótipos em hardware tradicionais, e pode ser usado para explorar um grande número de arquiteturas.

3.10 Splash

Splash é um dispositivo reconfigurável desenvolvido para otimizar a resolução de problemas que envolvem algoritmos sistólicos [GOK90]. Como existem métodos para mapear matrizes sistólicas em matrizes lineares, Splash tem uma organização de matriz linear. Essa matriz consiste de 32 FPGAs 3090 da Xilinx, cada um acoplado a blocos de 128kb de memória RAM.O sistema possui interconexão entre cada estágio adjacente através de um barramento de 68 *bits*. O primeiro e o último estágios são ligados ao computador hospedeiro através de um vetor do tipo *FIFO*, conforme pode ser visto na Figura 3.11. A placa em si é conectada ao computador hospedeiro através de dois barramentos: uma para transferência de dados e outro para configuração.

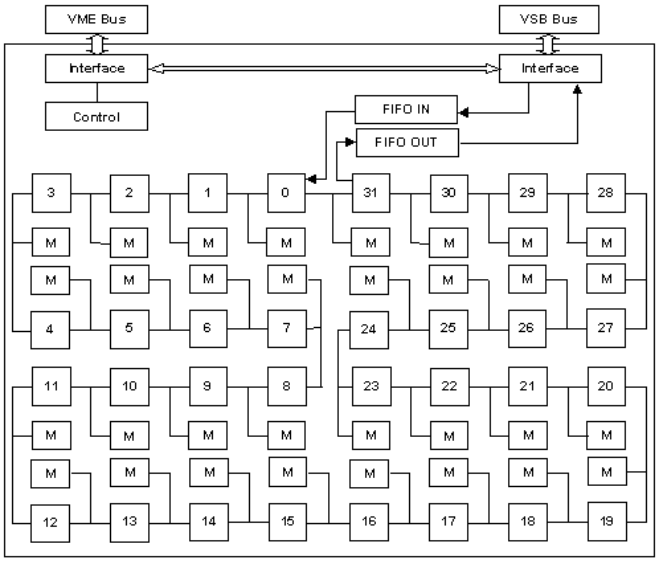

Figura 3.11: Interconexões do Splash

O ambiente de desenvolvimento de Splash é composto por várias ferramentas de desenvolvimento. Contudo todas as ferramentas permitem somente uma programação de baixo nível

dos FPGAs. Um ambiente chamado *Runtime* para o sistema Splash foi desenvolvido com um depurador e um conjunto de rotinas em C, para acesso direto à plataforma Splash através de programas em C.

Splash tem vários pontos negativos, entre eles suas ferramentas rudimentares de software. Sua importância deve-se ao fato de este sistema ter mostrado vantagens significantes da computação reconfigurável sobre arquiteturas convencionais.

3.11 Splash2

O Splash2 é baseado em 17 FPGAs 4010 da Xilinx, cada um acoplado a blocos de 512Kb de memória. Dezesseis desses FPGAs são conectados em um vetor, e também a uma rede *crossbar* que introduz uma flexibilidade maior que se fosse tratado como uma matriz linear [J. M92]. O décimo sétimo FPGA é conectado a rede *crossbar* e atua como difusor dos dados que recebe do barramento SIMD (*Single Instruction Multiple Data*). A placa Splash2 é ligada ao resto do sistema por cinco barramentos. Além do barramento SIMD, usado para transferir dados do computador hospedeiro à placa, existem dois barramentos de extensão, um barramento da dados de saída e um barramento de configuração. Um esquema da arquitetura pode ser visto na Figura 3.12.

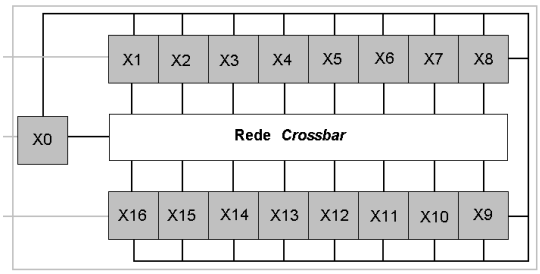

Figura 3.12: Arquitetura do Splash2

O sistema Splash2 apresenta ferramentas de software baseadas em VHDL, linguagem de nível mais alto que a utilizada no Splash. Um simulador de Splash2 compreende um conjunto de descrições VHDL para cada um dos componentes do sistema, e pode simular o comportamento do sistema. Uma outra parte da plataforma de software é um compilador que compila código VHDL em uma *netlist* Xilinx, e a otimiza para o sistema Splash2. Softwares da Xilinx fazem o roteamento e a localização (*placement*). Além disto, um C de alto nível e com características de paralelismo no nível de bit (dbC) tem sido adaptado para Splash2.

Splash2 foi desenvolvido para incrementar certos aspectos do sistema Splash: escalabilidade, largura de banda para E/S e programabilidade. Tem conexões mais flexíveis que seu antecessor, apresenta difusão e rede *crossbar*. O sistema inteiro é controlado em um nível mais alto, através de uma linguagem com tipos definidos por usuários, verificação forte de tipos, sobrecarga de operadores, funções, procedimentos e pacotes. Assim como seu antecessor, Splash2 sofre

pela baixa velocidade de integração, mas tem a vantagem da reconfiguração parcial. Esta propriedade é devida a possibilidade de reconfiguração da rede de interconexão.

3.12 SPYDER

SPYDER é um anagrama de *REconfigurable Processor Development SYstem*. Este sistema é um coprocessador reconfigurável que foi criado com o objetivo de auto-adaptar-se a uma dada aplicação de uma forma transparente ao usuário [SAN99].

Um processador consiste de uma unidade de controle e uma unidade de processamento (ou *data path*). SPYDER atua como uma unidade de processamento, que consiste de três FPGAs conectados a um banco de registradores. Cada FPGA mantém um acesso independente aos registradores no sentido de permitir processamento paralelo de dados e, por conseguinte, a implementação de arquiteturas superescalares. A Figura 3.13 exibe a arquitetura SPYDER.

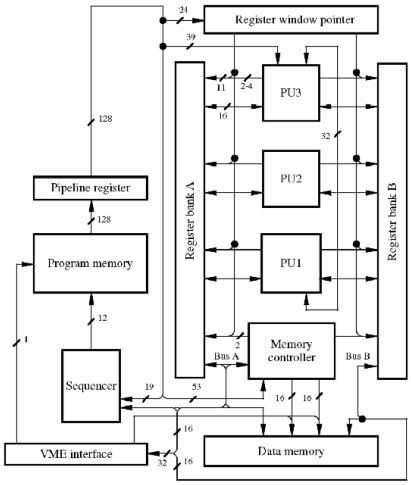

Figura 3.13: Arquitetura Spyder

Dada a complexidade do problema que essa arquitetura propos-se a resolver, foi implementada uma solução intermediária: o usuário determina os operadores e os descreve em uma linguagem de alto nível (C++). O compilador então gera a configuração correspondente aos FPGAs. Finalmente a aplicação é escrita pelo usuário através do uso do código correspondente e determina as operações de tal forma que consiga o máximo grau de paralelismo.

Como um coprocessador comum, SPYDER é conectado a um computador hospedeiro, que manipula entradas e saídas, e executa os programas de desenvolvimento. Mas apesar disto, atua como um processador von Neumman [RAD98].

Uma das grandes vantagens desta abordagem é a idéia de colocar unidades reconfiguráveis em uma unidade de processamento von Neumman. Isto combina as vantagens de um processador von Neumman em aplicações de propósito geral com a alta eficiência de arquiteturas reconfiguráveis em aplicações de domínio específico.

Contudo, mais uma vez, o problema desta arquitetura é a falta de uma ferramenta capaz de esconder detalhes de hardware do usuário.

3.13 Transmogrifier-2

Transmogrifier-2 (TM-2) é um sistema multi-FPGA de prototipação rápida, que oferece alta capacidade, altas taxas de relógio, e é flexível o bastante para implementar uma grande variedade de sistemas [LEW98]. O sistema pode compreender dezesseis placas com dois FPGAs 10K50 da Altera cada, quatro circuitos integrados I-Cube para interconexão, e 8 Mb de memória. Uma fotografia de um sistema duplo TM-2 é mostrada na Figura 3.14. A arquitetura de roteamento inter-FPGA usa uma nova estrutura: uma *crossbar* parcial não uniforme que provê um atraso constante entre qualquer dupla de FPGAs do sistema. A arquitetura é modular e escalonável, o que significa que sistemas de tamanhos variados podem ser construídos a partir de cópias da mesma placa, mantendo a capacidade de roteamento e a característica de atraso constante.

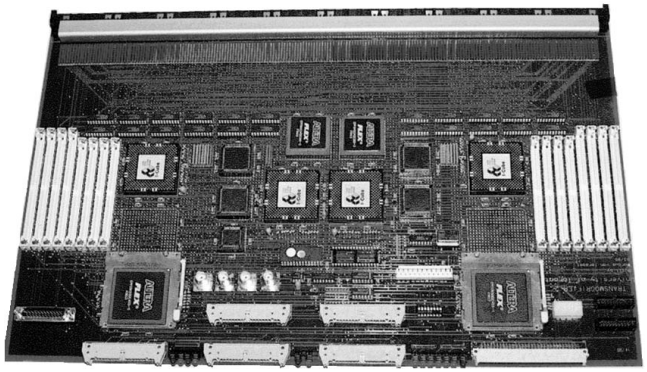

Figura 3.14: Placa de circuito impresso Transmogrifier 2

Os pontos principais do TM-2 são os seguintes:

- *Modularidade e escalabilidade*: É escalonável porquanto pode ser utilizado em uma ampla faixa de tamanhos, unindo placas do mesmo tipo. É modular exatamente porque as placas são construídas da mesma forma, para permitir a escalabilidade.
- *Alta velocidade*: Pode ser implementado com taxas razoavelmente altas de relógio (10MHz).
- *Alta largura de banda para memória e E/S* (o autor não menciona o tamanho dessa banda);
- *Baixo tempo de configuração*: O sistema pode ser reconfigurado em menos de 10 segundos ;
- *Manufaturabilidade:* O TM-2 tem custo baixo, usa tecnologia encontrada no comércio e nenhuma sinalização não-convencional é utilizada para interconexão.

O software de CAD para TM-2 provê um fluxo automatizado de projeto que inclui junção de múltiplos arquivos de projetos (gerados por HDLs ou esquemáticos), roteamento e geração do arquivo *".bit"*. Configuração e depuração são feitas de forma interativa. O particionamento do projeto, até o presente momento, é feito manualmente.

Futuramente, a equipe de projeto do Transmogrifier-2 pretende utilizar FPGAs Altera10K100 para suportar 2 milhões de portas. Também pretendem investir em melhorias do CAD.

3.14 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 3.15 ilustra essa arquitetura.

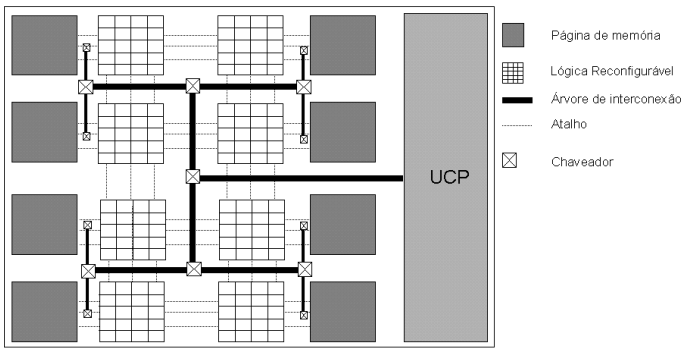

Figura 3.15: Arquitetura Trumpet

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. Também vale ressaltar que a complexidade de desenvolvimento para DRAM é ocultada do programador, através da utilização de uma interface semelhante a SRAM.

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

4 Tecnologias habilitadoras

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 *placement*.

Outro fabricante de FPGAs, a Altera, somente a partir da família APEX permitiu reconfiguração parcial, mas 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 o *download* externo de novas configurações.

A empresa Atmel possui duas famílias de FPGAs que permitem reconfiguração parcial [ATM00], cujas características gerais serão analisadas na seção 4.1.

Na seção 4.2 desste capítulo será abordada a arquitetura interna da família Virtex de FPGAs.

4.1 Famílias AT6000 e AT40K

A série AT40K [ATM00a] de FPGAs da Atmel possui de 5 a 50 mil portas lógicas equivalentes, e foi projetada para alta densidade e computação intensiva, além de outros projetos lógicos. A Atmel tenta resolver o problema da relação Lógica *versus* SRAM através de uma RAM rápida, flexível e distribuída, que foi chamada de *FreeRAM*, que não utiliza valiosos recursos lógicos. Funções lógicas estruturadas, incluindo multiplicadores de matrizes, podem ser implementados diretamente em células *core* sem o uso de recursos de barramento, alcançando grandes ganhos em termos de velocidade, utilização (área), consumo de energia e custo.

O coração do FPGA AT40K é uma matriz simétrica de células idênticas. A matriz é contínua de uma borda a outra, exceto pelo barramento repetidor, que regenera sinais e conecta um barramento a qualquer outro. Os repetidores dividem os recursos de barramento em uma área que é matriz 4x4 de células, chamadas de setores. No canto inferior direito de cada setor há 32 blocos de *FreeRAM* de 4 bits, acessíveis através dos barramentos adjacentes. A RAM pode ser configurada como simples ou dupla-porta, com operações síncronas ou assíncronas. A

arquitetura do FPGA AT40K pode ser observada na Figura 4.1.

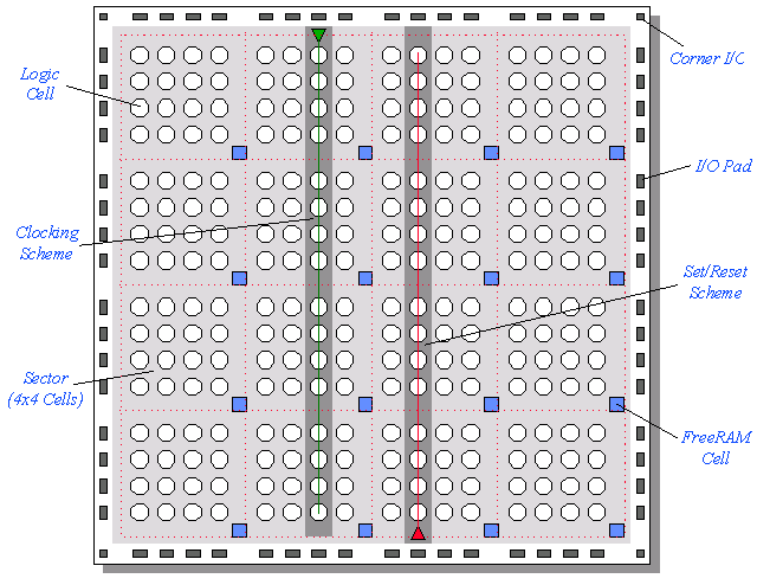

Figura 4.1: Arquitetura da AT40K

A família AT6000 [ATM00b] foi projetada para acelerar o desempenho de sistemas baseados em processadores, enquanto diminui o custo e o consumo de energia. O número massivo de registradores (de 1.024 a 6.400) permitem seu uso como co-processadores reconfiguráveis para DSP. Pode incrementar a velocidade do sistema em uma ordem de grandeza eliminando o gargalo de funções DSP em software.

Ambas famílias (AT40K e AT6000) tem a possibilidade de implementação de projeto de *Cache Logic*, onde parte de um FPGA pode ser reprogramada sem perda de dados de registradores, e enquanto o restante do FPGA continua a operar sem interrupção.

A configuração de um FPGA da Atmel é classificada em cinco níveis. O primeiro nível trata da configurabilidade comum a todos os FPGAs. O nível 2 é a habilidade de reprogramar todo o dispositivo durante a operação do sistema. O terceiro nível é a capacidade de reprogramar apenas algumas áreas do FPGA durante a execução. O penúltimo nível é a possibilidade de escrever valores em registradores usando uma ferramenta chamada *Quick Change*. Nível 5 é a habilidade de escrever qualquer componente ou bit no dispositivo através da escrita no *Bitstream*[1].

Esta seção enfocará o nível 3 de configurabilidade, que trata da reconfiguração parcial da família AT40K.

[1] Coleção de comandos e informações de configuração.

4.1.1 Reconfiguração parcial na família AT40K

Os FPGAs desta família 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 4.2. Este procedimento aproveita-se da latência funcional inerente a muitas aplicações - em qualquer tempo dado, somente uma pequena proporção da lógica está de fato ativa.

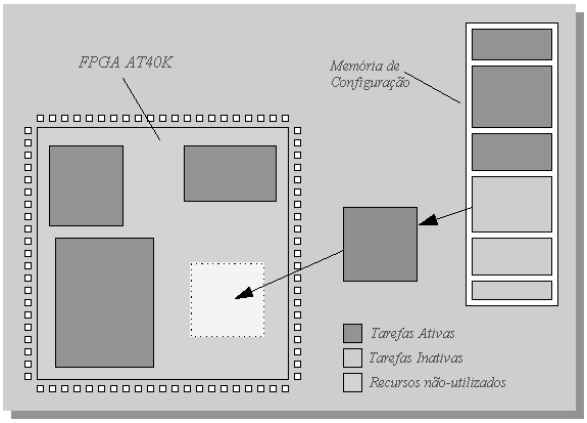

Figura 4.2: Diagrama da *Cache Logic*

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. Através da multiplexação de tempo um único FPGA AT40K pode implementar muitos projetos, o que de outra forma requeriria muitos FPGAs estáticos.

A reconfiguração parcial pode ocorrer de duas formas diferentes, que são dependentes do projeto que está sendo implementado [ATM00a].

A primeira maneira ocorre quando dois projetos separados são implementados em tempos diferentes num único FPGA. Por exemplo, um circuito contendo um *flip-flop T* deve ser implementado primeiro numa metade do FPGA, enquanto um circuito contendo um *flip-flop D* deve ser implementado na outra metade, e este último é totalmente independente do primeiro (como visto na Figura 4.3). Isto implica que os *flip-flops T* e *D* tem relógios separados, bem como *clears, sets,* entradas e saídas separadas.

A segunda forma de reconfiguração parcial acontece quando um segundo projeto é implementado depois que outro projeto já foi implementado no FPGA. Por exemplo, *flip-flops T* já foram implementados e *flip-flops D* são requeridos posteriormente. Contudo, ao invés de usar um segundo relógio, foi escolhido usar a saída $\overline{Q}$ do *flip-flop T,* conforme a Figura 4.4.

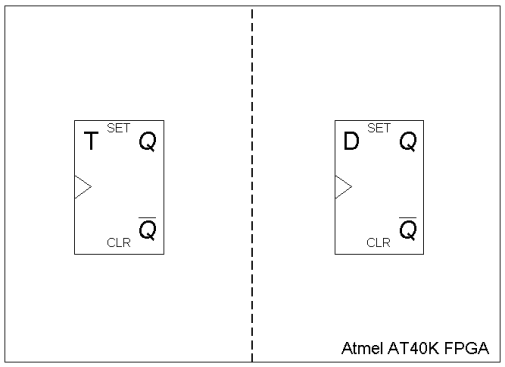

Figura 4.3: AT40K contendo *flip-flops* T e D independentes

Isto mostra que é possível que um novo circuito conecte-se em qualquer saída de circuitos já implementados. Até o presente momento este tipo de reconfiguração parcial parece ocorrer somente nesta família de FPGAs. Pesquisas posteriores serão realizadas com o intuito de verificar essa possibilidade em FPGAs da família Virtex.

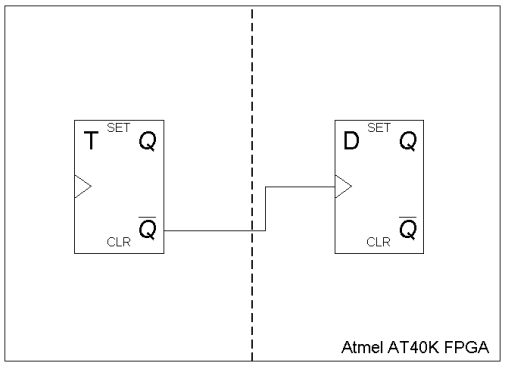

Figura 4.4: AT40K contendo *flip-flops* T e D acoplados

A ferramenta de CAD da Atmel permite reconfiguração à medida que torna possível, em sua janela de Compilação, a escolha da porção do FPGA onde a lógica será implementada, e a geração de mais de um *Bitstream* por dispositivo. É possível projetar, usando o exemplo da Figura 4.3, primeiramente a parte do FPGA referente ao *flip-flop* T , gerar o respectivo *Bitstream,* escolher o local que ocupará no FPGA e carregá-lo na placa; e, posteriormente, projetar e implementar o *flip-flop* D*,* repetindo o processo.

4.2 Arquitetura de um FPGA Virtex

A arquitetura Virtex suporta novos modos de configuração, incluindo reconfiguração parcial. Na Seção seguinte será abordada a arquitetura da Virtex, com ênfase na locação dos bits

de dados no *Bitstream*. A Seção 4.4 tratará de um estudo de caso de reconfiguração parcial utilizando-se um FPGA Virtex.

O conhecimento da locação dos bits é a base para acessar e alterar informações no CI. Podem ser escritas aplicações que utilizam ou modificam as funcionalidades do circuito operacional sem a necessidade de parar o dispositivo [XIL00]. A família Virtex tem como membros os FPGAs Virtex e Virtex-E, e Virtex-E *Extended Memory,* e diferem entre si principalmente pela quantidade de blocos de RAM disponíveis em cada um.

4.3 Estruturas básicas de configuração

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, conforme Figura 4.5. Essas funcionalidades lógicas são configuradas através do *Bitstream*. *Bitstreams* de configuração contém uma mescla de comandos e dados. Eles podem ser lidos e escritos através de uma das interfaces de configuração da Virtex.

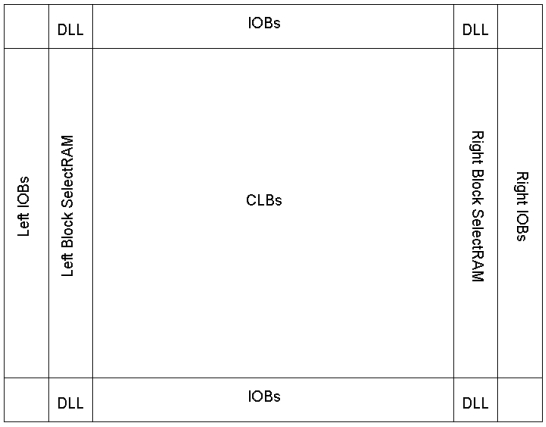

Figura 4.5: Visão geral da arquitetura Virtex

Essas interfaces são SelectMAP, serial mestre/escravo, ou interface Boundary Scan.

A escrita de alguma parte ou toda a configuração é feita através do lançamento de comandos de configuração na interface desejada, seguidos das informações de configuração.

A interface SelectMAP possui 8 bits no dispositivo, com pinos nomeados D[7:0]. O *Bitstream* de configuração pode ser escrito na taxa de 8 bits por ciclo de relógio.Dispositivos Virtex podem ser configurados para reter os seguintes pinos da SelectMAP: *D[7:0], BUSY/DOUT,* $\overline{INIT}$, $\overline{WRITE}$ e $\overline{CS}$. Isto permite posterior reconfiguração através destes pinos.

Quando uma interface serial mestre/escravo ou a interface Boundary Scan é utilizada para configuração ou reconfiguração, apenas um bit por ciclo de relógio é transmitido.

Informações de configuração podem ser lidas usando as interfaces SelectMAP ou Boundary Scan. A interface mestre/escravo não pode ser usada para leitura de configuração. A leitura

total ou parcial de uma configuração é feita através do lançamento de comandos de leitura à interface desejada, e seguinte leitura dos dados a partir desta mesma interface.

4.3.1 Colunas de configuração

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.

Quadros são agrupados em unidades maiores chamadas colunas. Em dispositivos Virtex existem diferentes tipos de colunas, conforme a Tabela 4.1:

Tipo de Coluna de configuração	**# de Quadros**	**# por Dispositivo**
Centro	8	1
CLB	48	# de colunas CLB
IOB	54	2
Interconexão de bloco de SelectRAM	27	# de colunas de bloco de SelectRAM
Bloco de SelectRAM de conteúdo	64	# de colunas de bloco de SelectRAM

Tabela 4.1: Tipos de colunas de configuração

Cada dispositivo contém uma coluna central que inclui configuração para os pinos do relógio global. Duas colunas IOB representam a configuração para todos IOBs nas margens esquerda e direita do dispositivo. A maioria das colunas são CLBs, e cada coluna CLB tem dois IOBs acima e dois abaixo delas. Os dois tipos de colunas restantes envolvem os blocos de RAM: um para conteúdo e outro para interconexão. Um exemplo de colunas do FPGA Virtex modelo XCV50 é mostrado na Figura 4.6.

4.3.2 Endereçamento de configuração

O espaço total de endereços é dividido em tipos de blocos. Existem dois tipos de blocos: RAM e CLB. O tipo RAM contém somente as colunas de blocos de SelectRAM de conteúdo. O tipo CLB contém todos os outros tipos de colunas.

Ambos espaços de endereçamento (RAM e CLB) são divididos em endereços mais e menos significativos. Cada coluna de configuração tem um único endereço mais significativo dentro do espaço de RAM ou CLB. Cada quadro de configuração possui um único endereço menos significativo dentro do espaço da coluna.

O esquema de numeração para os tipos de blocos e endereços menos significativos são idênticos para todos a família Virtex. Contudo a numeração do endereço mais significativo difere entre as famílias Virtex e Virtex-E, conforme pode ser visto na Tabela 4.2.

Para exemplificar o endereçamento mais significativo será tomada como exemplo a família Virtex. O espaço de endereços de CLB inicia com “0” para a coluna central e alterna entre as

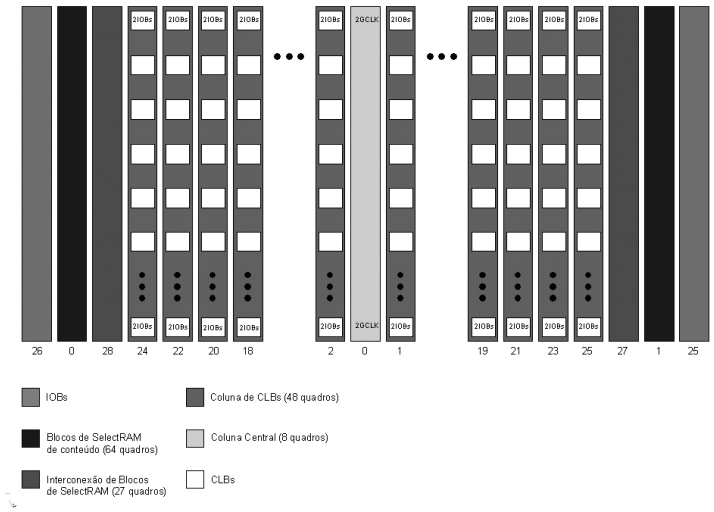

Figura 4.6: Exemplo de colunas de configuração para o FPGA XCV50

metades esquerda e direita para todas colunas de CLBs, depois para colunas IOBs e finalmente para blocos de interconexão das SelectRAM.

O espaço de endereçamento tem um "0" para a coluna do bloco de conteúdo de SelectRAM da esquerda, e "1" para a coluna da direita. A alocação de quadros aos recursos do dispositivo em um FPGA XCV50 pode ser vista na Figura 4.6.

4.3.3 Quadros

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.

O tamanho do quadro depende do número de linhas do dispositivo. O número de bits de configuração em um quadro é 18 x (# linhas_CLB + 2). O quadro é preenchido com zeros à direita (embaixo) para caber em palavras de 32 bits. Uma palavra de preenchimento adicional é necessária para identificar o final de cada quadro, para fins de *pipelining*.

Cada quadro encaixa-se verticalmente no dispositivo, com a "frente" do frame acima. Conforme mostrado na Figura 4.7 é conveniente considerar o quadro horizontalmente quando ele é visto como parte do *Bitstream*. Os IOBs de cima são vistos à esquerda, seguidos pelas CLBs através da coluna, até os IOBs na direita.

Os bits são organizados nos quadros conforme o que segue:

- Para colunas CLB, os primeiros 18 bits controlam os 2 IOBs no topo da coluna; então 18 bits são alocados para cada linha de CLB; em seguida os 18 bits seguintes controlam

Tipo de Coluna	Tipo de Bloco	Virtex	Virtex-E
Primeiro endereço mais significativo	CLB	0	0
	RAM	0	1
Ordem do endereçomais significativo	CLB	1:Centro	1:Centro
		2:CLB	2:Interconexão BRAM/CLB
		3:IOB	3:IOB
		4:Interconexão BRAM	
	RAM	Bloco de Conteúdo da RAM	Bloco de Conteúdo da RAM

Tabela 4.2: Esquema de endereçamento superior por família Virtex

Figura 4.7: Organização dos frames na coluna CLB

os 2 IOBs da base da coluna. O quadro contém bits de enchimento para fazer dele um múltiplo integral de 32 bits.

Para colunas IOB, o quadro aloca 18 bits a cada 3 IOBs, conforme a Figura 4.8. Durante a leitura e a escrita de frames, os bits são agrupados em palavras de 32 bits, iniciando na esquerda (que corresponde ao topo do dispositivo). Se existem palavras que não preenchem totalmente uma palavra de 32 bits, o quadro é completado com zeros a direita.

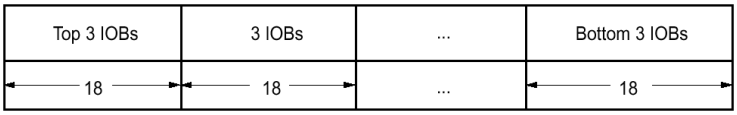

Figura 4.8: Organização dos frames na coluna IOB

Para as colunas de blocos de conteúdo de SelectRAM (Figura 4.9) os primeiros 18 bits são bits de preenchimento; os 72 bits seguintes são alocados para cada linha de RAM; finalmente existem outros 18 bits de preenchimento. O quadro tem bits de preenchimento suficientes para torná-lo múltiplo de 32.

Figura 4.9: Organização dos frames na coluna BRAM

4.3.4 Posicionamento dos bits da LUT SelectRAM

No que diz respeito ao início do frame de configuração, o posicionamento relativo dos bits de LUT SelectRAM dentro do *Bitstream* é o mesmo para cada fatia de CLB. Os bits de LUT SelectRAM são espraiados através do endereço menos significativo de 16 quadros consecutivos. Cada endereço menos significativo de quadro contém todas as instância de um índice daquela coluna. Esses 16 quadros contém todos os 16 bits da LUT SelectRAM para uma coluna de fatias de CLB. Para ler ou escrever uma LUT SelectRAM é necessário ler ou escrever todos esses 16 quadros. A LUT SelectRAM trabalha com lógica negada.

4.3.5 Leitura de dados de configuração

Informações de configuração podem ser lidas dos dispositivos Virtex se a opção de *readback* estiver habilitada para a corrente configuração. Durante a leitura de dados de um dispositivo Virtex, um quadro de preenchimento é lido antes de qualquer quadro com informações válidas. Isto ocorre adicionalmente à palavra de preenchimento de cada quadro. O quadro de preenchimento deve ser incluído na contagem de palavras a serem lidas do dispositivo.

Cada quadro lido do dispositivo consiste do número de palavras mostrado na Tabela 4.3.

Quadro de Preenchimento (16 palavras)	
Palavra de preenchimento	Quadro de dados 0 (15 palavras)
...	...
Palavra de preenchimento	Quadro de dados n (15 palavras)

Tabela 4.3: Quadro de preenchimento para leitura do dispositivo

4.3.6 Escrita de dados de configuração

Informações de configuração podem ser escritas dos dispositivos Virtex se a opção de *write-back* estiver habilitada para a configuração atual. A re-escrita das mesmas informações de configuração não gera quaisquer sinais transientes.

Alterações nas informações de configuração podem gerar sinais transientes, especialmente se os valores da LUT ou sinais de roteamento são alterados. Para este caso, todas as células lógicas e roteamento podem ser colocados em um estado não-contencioso.

Durante a escrita das informações de configuração em um dispositivo Virtex, seja pela interface SelectMAP ou JTAG, os quadros de dados são escritos com cada quadro seguido por

uma palavra de preenchimento. Depois de todos os quadros de informações escritos, um quadro de preenchimento deve ser escrito (para esvaziar os *pipelines* internos). As palavras e os quadros de preenchimento devem ser incluídos no número de palavras a serem escritas no dispositivo.

4.3.7 Alteração nas informações de configuração

Dispositivos Virtex suportam o método leitura-modificação-escrita (*Read-Modify-Write,* ou*RMW*) para modificação de dados na LUT SelectRAM. Então, é possível ler ou alterar os conteúdos de uma ou mais LUT SelectRAMs através das interfaces JTAG ou SelectMAP. Quando da escrita de dados em uma ou mais LUT SelectRAMs, todos os bits do quadro devem ter informações de configuração válidas. Isto pode ser assegurado através da alteração de configurações válidas de arquivos de *Bitstreams,* ou de quadros lidos de um dispositivo Virtex configurado corretamente. O método RMW deve ignorar o quadro de preenchimento e a palavra de preenchimento do primeiro frame. Isto alinha as informações de configuração remanescentes no formato de escrita do dispositivo. Depois de modificada a configuração, as informações alteradas podem ser escritas no dispositivo Virtex, seguidas de uma palavra e um frame de preenchimento.

LUT SelectRAMs não configuráveis externamente não devem dispôr-se na mesma fatia de LUTs ou LUT SelectRAMs que serão reconfiguradas externamente.Tal mistura pode causar que dados não relacionados a LUT SelectRAMs a serem reconfiguradas sejam modificados quando da escrita dos quadros no dispositivo.

4.4 Estudo de caso: Semáforo de Status e Controle Utilizando Reconfiguração Parcial

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.

A interface SelectMAP da Virtex torna possível configuração total ou parcial em alta velocidade (400Mb/s). Através de um mecanismo de semáforo, a porta SelectMAP pode 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 SelectMAP. Através do 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 4.10.

Esta subseção descreve um semáforo de 16 bits usa células de RAM16x1D, e o procedimento para ler e escrever de/para este módulo através de uma porta de configuração. Este conceito

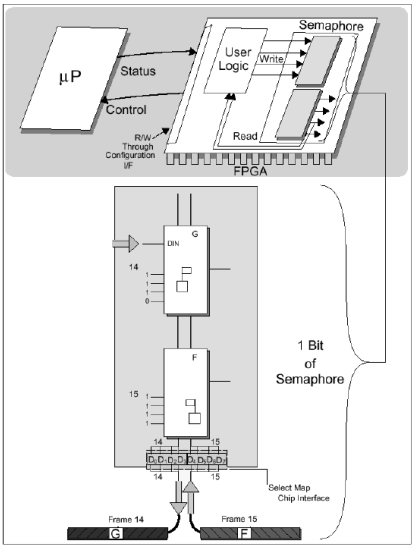

Figura 4.10: Metodologia do semáforo

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 4.11. 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 4.4.

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

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 4.4: Portas do Módulo Semáforo

O endereço da porta de escrita da SelectRAM dupla-porta é fixado em '1111', e o endereço da porta de leitura é configurado em '1110'. 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

```
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 4.11: Semáforo em VHDL

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, localizado (*placed*) e roteado, o próximo passo é determinar a locação dos bits dentro do arquivo RBT, que corresponde aos bits de SelectRAM [XIL00]. O módulo Semáforo pode ser colocado em qualquer lugar do FPGA onde não existem LUT SelectRAMs nem *Shift Register* LUTs na mesma coluna de CLBs. Caso contrário os conteúdos de RAM/SRL seriam sobrescritos.

5 Conclusões e trabalhos futuros

5.1 Conclusões

Este trabalho apresentou conceitos relacionados com computação reconfigurável, especialmente no tocante à reconfiguração dinâmica e parcial de sistemas baseados em FPGAs.

Foi realizado um estudo sobre diversos critérios de classificação para sistemas reconfiguráveis, seguido de uma avaliação de importantes arquiteturas reconfiguráveis, que foram comparadas através dos quesitos derivados do primeiro estudo.

As principais tecnologias habilitadoras a esta forma de computação foram revistas. Um estudo de caso com um dos atuais FPGAs que permitem reconfiguração dinâmica e parcial foi analisado.

O embasamento teórico proporcionado por este trabalho acarretará na proposta de implementação de um sistema reconfigurável com ênfase em telecomunicações.

5.2 Trabalhos Futuros

Como ponto de partida para a utilização de reconfiguração dinâmica será a análise da viabilidade de implementação do módulo proposto no item 4.4. O semáforo poderá ser implementado em um FPGA da família Virtex (trabalho (i)), possivelemente o modelo XVC300.

O objetivo desta tarefa é adquirir a práxis no desenvolvimento de sistemas digitais utilizando uma linguagem de descrição de hardware (VHDL), além da familiarização com a arquitetura Virtex e com as ferramentas de CAD para prototipação de sistemas digitais da Xilinx.

Outro trabalho relacionado com reconfiguração parcial a ser executado é a implementação de um sistema reconfigurável baseado em chaveamento de contexto (trabalho (ii)). FPGAs da família Virtex podem ser reconfigurados parcialmente através da escrita nos quadros de uma coluna de CLBs. Este tipo de reconfiguração impõe que as lógicas a serem executadas estejam implementadas no FPGA, conquanto apenas parte delas esteja ativa em dado momento de tempo. A reconfiguração, neste caso, implica meramente na definição de qual lógica será ativada e qual será desativada num dado instante. Este procedimento exige que a lógica sofra restrições de localização e roteamento. Tais restrições serão realizadas com a utilização da ferramenta Floorplanning.

Na seqüência direta desta tarefa encontra-se a tentativa de reconfigurar uma coluna inteira de CLBs (trabalho (iii)), o que acarretará num estágio de reconfiguração mais amplo e flexível que o chaveamento de contexto. A lógica a ser implementada não mais estará previamente no FPGA com uma parte ativa e outra inativa. Ao contrário, o FPGA estará executando determinada lógica quando receberá um *bitstream* contendo a porção de lógica que será implementada. A reconfiguração dar-se-á pela modificação de colunas inteiras de CLBs. Em paralelo a este desenvolvimento buscar-se-á a definição de uma estrutura padrão para comunicação entre colunas de CLBs, que estaria para o FPGA como barramento PCI está para a placa-mãe de um microcomputador.

Propõe-se ainda, futuramente, um estudo de ferramentas para download remoto de configurações parciais, com o objetivo de possibilitar a atualização remota de sistemas baseados em FPGAs. A Figura 5.1 mostra as tarefas propostas e a aquisição de conhecimentos e experiências decorrentes de seus desenvolvimentos.

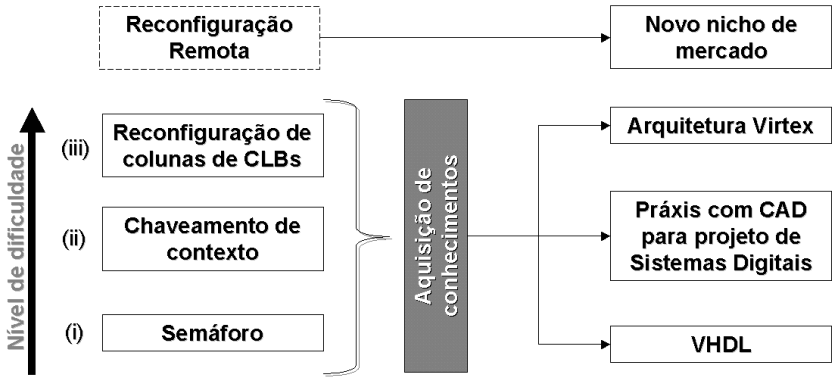

Figura 5.1: Diagrama de trabalhos futuros

Os trabalhos (i), (ii) e (iii) originarão *cores* e técnicas de implementação destes, e serão disponibilizados na Internet a partir do site do Grupo de Apoio ao Projeto de Hardware (GAPH).

Todos essas tarefas serão iniciadas na disciplina de Trabalho Individual II. Contudo, devido o grau de complexidade (ainda desconhecido, mas supostamente alto) inerente a estas, a priori é impossível determinar até onde os trabalhos avançarão durante o período da referida disciplina.

A síntese dos temas trabalhados - reconfiguração parcial, dinâmica e remota de sistemas digitais em telecomunicações, será o mote da dissertação de mestrado.

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).

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

[ATM00] ATMEL, Equipe de documentação da. **Field progammable gate array**. Disponível por WWW em http://www.atmel.com/atmel/products/prod3.htm (2000).

[ATM00a] ATMEL, Equipe de documentação da. **At40k series configuration**. Disponível por WWW em http://www.atmel.com/atmel/postscript/doc1009.ps.zip (2000).

[ATM00b] ATMEL, Equipe de documentação da. **At6000 series configuration**. Disponível por WWW em http://www.atmel.com/atmel/postscript/doc0436.ps.zip (2000).

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

[CAL98] CALAZANS, Ney Laert Vilar. **Projeto lógico automatizado de sistemas digitais seqüenciais**. [S.l.]: 11ł Escola de Computação, 1998. 318p.

[CAL00] CALLAHAN, Timothy J.; HAUSER, John R.; WAWRZINEK, John. Introduction to the special issue on computational linguistics using large corpora. **Computer Magazine**, n.4, p.62–69, 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).

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

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

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

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

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

[LEW98] LEWIS, David M.; GALLOWAY, David R.; IERSSEL, Marcus van; ROSE, Johnathan; CHOW, Paul. The transmogrifier-2: a 1 million gate rapid prototyping system. Nova Iorque, EUA: [s.n.], 1998. p.188–198.

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

[PAG97] PAGE, Ian. Reconfigurable processor architectures. **Internet**, 1997.

[PER99] PERISSAKIS, Stylianos; JOO, Yangsung; AHN, Jinhong; DEHON, Adré; WAWRZYNEK, John. Embedded dram for a reconfigurable array. In: VLSI '99, 1999, USA. **Anais...** 1999.

[RAD98] RADUNOVIC, Bozidar; MILUTINOVIC, Veljko. A survey of reconfigurable computing architectures. In: FPL98, 1998, Tallin, Estonia. **Anais...** 1998.

[SAN99] SANCHEZ, Eduardo; AL. et. Static and dinamic configurable systems. Nova Iorque, EUA: [s.n.], 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).

[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.

[WIR96] WIRTHLIN, Michael J.; HUTCHINGS, Brad L. A dinamic struction set computer. In: FPD96, 1996, Toronto, Canada. **Anais...** 1996.

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

[XIL99] XILINX, Equipe de documentação. **Xc6200 datasheet**. Disponível por WWW em http://dec.bournemouth.ac.uk/drhw_lib/archive/xc6200.pdf (Junho 1999).

[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 (Fevereiro 2000).