

Introdução à computação reconfigurável *

Iouliia Skliarova, António B. Ferrari

Resumo – Graças à invenção de novos tipos de dispositivos lógicos programáveis (PLDs – *Programmable Logic Devices*), o processo de projecto de sistemas digitais sofreu grandes alterações durante as últimas décadas. Actualmente, muitos sistemas digitais são implementados com a ajuda de PLDs de densidade elevada (em particular, com a ajuda de FPGAs - *Field-Programmable Gate Arrays*). O mercado de FPGAs continua a crescer o que resulta numa grande variedade de dispositivos disponíveis. Neste contexto surgiu um novo método de computação - a computação reconfigurável à qual dedicamos este artigo. Os objectivos principais que se pretende alcançar consistem em introduzir a noção de computação reconfigurável, revelar as suas capacidades mais importantes e inovadoras e auxiliar os alunos das disciplinas relevantes na concepção das ideias básicas da computação reconfigurável. Para tal apresenta-se uma revisão de vários tipos de organização de sistemas reconfiguráveis, nomeadamente consideram-se modos de reconfiguração, mecanismos de interacção entre os componentes principais, modelos de programação de todo o sistema, etc. A seguir, são analisadas técnicas utilizadas na computação reconfigurável para atingir desempenho elevado. Apresentam-se também alguns exemplos de sistemas reconfiguráveis mais conhecidos e as áreas típicas de aplicação da computação reconfigurável.

Abstract – With the advent of new types of programmable logic devices (PLDs), the process of digital system design has undergone a notable revision during the past few decades. Actually, many digital systems are implemented with the aid of high-capacity PLDs (FPGAs, in particular). The FPGA market continues to grow resulting in a wide variety of available devices. In this context the new concept of reconfigurable computing, to which this paper is devoted, has arisen. The main goal of this paper is to introduce the notion of reconfigurable computing, revealing its most important and innovative capacities, and helping the students of relevant disciplines to understand the basic ideas of reconfigurable computing. In order to accomplish this goal we represent a revision of various types of reconfigurable systems organization, namely different

reconfiguration modes, coupling mechanisms, the programming model, etc. Then, some techniques that are usually employed in reconfigurable computing in order to achieve good performance are analyzed. Finally, a number of practical examples are discussed and typical application areas are presented.

I. INTRODUÇÃO

Os computadores convencionais podem ser utilizados para uma ampla gama de aplicações. Um computador de uso geral possui uma arquitectura e um conjunto de instruções fixos e aplicações diferentes são programadas dentro de restrições predefinidas. A programação destes computadores é bastante simples dado que existem muitas ferramentas disponíveis para o efeito. Tudo isto resulta num bom desempenho atingido para muitas aplicações a preço razoável. A vantagem principal desta abordagem é um grande nível de flexibilidade. Qualquer alteração no algoritmo é facilmente incorporável no código (especialmente, se o código foi desenvolvido com base em tecnologia orientada por objectos). Contudo, se considerarmos uma aplicação específica, os computadores convencionais não são capazes de assegurar o melhor desempenho.

Quando os requisitos de uma dada aplicação excedem as capacidades dos computadores de uso geral, recorre-se a abordagens diferentes destinadas a criar sistemas computacionais de desempenho elevado. Uma das técnicas utilizadas é a computação paralela. Ao decompor um problema em tarefas menores e processar estas tarefas em paralelo, pode-se atingir resultados bastante bons. Contudo, a aplicação deve possuir estrutura compatível com tal modelo de computação. Para muitas aplicações a computação paralela oferece pouca aceleração por cada unidade de processamento adicional, sendo a razão disso parcialmente explicada pela lei de Amdahl¹ [1]. Sistemas deste tipo sofrem também de grande sobrecarga (*overhead*) de comunicação entre vários processadores.

Uma abordagem diferente consiste em construir circuitos orientados à aplicação que são capazes de fornecer um

* Trabalho financiado com a bolsa da FCT-PRAXIS XXI/BD/21353/99

¹ O aumento de desempenho que é possível atingir com a ajuda de qualquer modo de execução mais rápido é limitado pela fracção de tempo durante a qual este modo pode ser utilizado

bom desempenho para essa aplicação específica. Por exemplo, é possível projectar e fabricar um circuito integrado específico (i.e. um ASIC - *Application Specific Integrated Circuit*) com o controlo fixo e as unidades funcionais personalizadas e optimizadas para uma dada aplicação. Com esta técnica pode-se atingir resultados muito bons com menos recursos de hardware. Contudo os custos de projecto e de implementação de tal sistema são demasiado elevados e só podem ser justificados no caso de produções de alto volume. Quando um circuito integrado é produzido em grandes quantidades, o custo inicial é amortizado dado que cada pastilha de silício fica responsável só por uma pequena parte do custo inicial. Por esta razão os computadores de uso geral são vendidos a um preço relativamente baixo. Uma outra dificuldade da abordagem orientada à aplicação é o tempo de desenvolvimento longo enquanto o desempenho de computadores de uso geral aumenta bastante rapidamente. Por este motivo, o hardware de uso especial pode ficar obsoleto passado pouco tempo. É de salientar também que as soluções baseadas em ASIC são completamente inflexíveis dado que a sua funcionalidade não pode ser modificada após o fabrico.

Levando em consideração o custo e a curta duração da vida útil dos computadores dedicados, estes não representam uma abordagem suficientemente atractiva a não ser que a necessidade deste hardware seja muito forte e justificada. Contudo, se for possível diminuir os custos e o tempo de desenvolvimento, a técnica orientada à aplicação torna-se bastante viável.

A computação reconfigurável é uma abordagem que combina as duas técnicas descritas acima possibilitando deste modo eliminar as desvantagens associadas com software “puro” (implementado num computador de uso geral) e hardware “puro” (ASIC) (ver fig. 1).

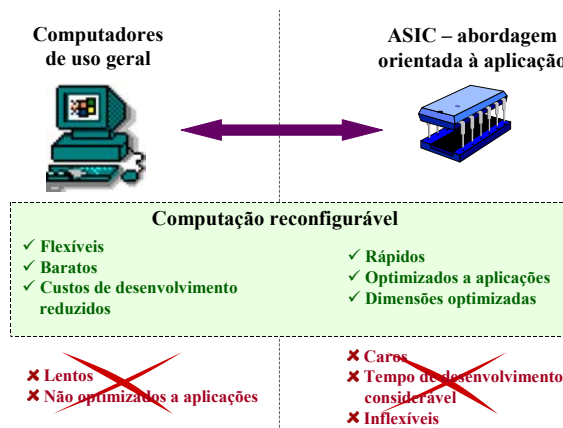


Fig. 1 – Computação reconfigurável combina vantagens principais de computadores de uso geral e de circuitos integrados orientados à aplicação e permite eliminar as suas desvantagens principais

A computação reconfigurável baseia-se em dispositivos lógicos reprogramáveis que podem atingir um desempenho elevado e, ao mesmo tempo, fornecer a flexibilidade da programação a nível de portas lógicas.

Um dispositivo de hardware típico utilizado em computação reconfigurável são as FPGAs (*Field-Programmable Gate Arrays*). A velocidade e os recursos disponíveis em FPGAs recentes são comparáveis com os dos ASICs, enquanto gozam de muita da flexibilidade inerente às implementações em software.

O resto deste artigo está organizado da maneira seguinte. A secção II inclui uma introdução às FPGAs. Na secção III apresenta-se uma revisão de vários tipos de organização de sistemas reconfiguráveis, nomeadamente consideram-se modos de reconfiguração, mecanismos de interacção entre os componentes principais, modelos de programação de todo o sistema, etc. A seguir, na secção IV, analisam-se técnicas utilizadas na computação reconfigurável para atingir desempenho elevado. A secção V dá alguns exemplos de sistemas reconfiguráveis mais conhecidos e a secção VI revela as áreas típicas de aplicação da computação reconfigurável. Finalmente, as conclusões estão na secção VII.

II. INTRODUÇÃO ÀS FPGAs

As FPGAs foram introduzidas há já bastante tempo e actualmente constituem componentes típicos de muitos produtos comerciais. São também utilizadas intensivamente na investigação a decorrer no Departamento de Electrónica e Telecomunicações da Universidade de Aveiro. Contudo poucos alunos, até passar pelas disciplinas que estudam FPGAs a fundo (tais como “*Computação reconfigurável*”, “*Sistemas Digitais Avançados*”, “*Sistemas Reconfiguráveis Avançados*”, etc.) têm alguma noção sobre as FPGAs.

As FPGAs são os dispositivos lógicos programáveis de maior capacidade disponível hoje em dia. O número de portas de sistema² em FPGAs comerciais tem crescido bastante rapidamente desde a sua introdução nos meados dos anos 80, e atinge actualmente 10M em dispositivos da Xilinx [2]. A arquitectura típica de uma FPGA está representada na fig. 2.

Uma FPGA inclui um *array* de blocos lógicos interligados por recursos de encaminhamento e cercado por um conjunto de blocos de entrada/saída, sendo todos estes componentes programáveis pelo utilizador. Os blocos lógicos contêm elementos combinatórios e sequenciais possibilitando a implementação de funções lógicas bem como de circuitos sequenciais. Os recursos de encaminhamento incluem segmentos de pistas de ligação pré-fabricadas e interruptores programáveis. Um circuito lógico é implementado em FPGA ao distribuir a lógica entre os blocos individuais e interligá-los posteriormente com os interruptores programáveis. É de notar que os atrasos resultantes são fortemente influenciados pela distribuição da lógica e pela estrutura de encaminhamento. Portanto, o desempenho de circuitos

² Esta medida não é muito adequada para expressar exactamente a capacidade lógica, mas é amplamente utilizada por vários vendedores de FPGAs

mapeados em FPGA depende bastante da eficácia das ferramentas CAD (*Computer Aided Design*) utilizadas para a implementação.

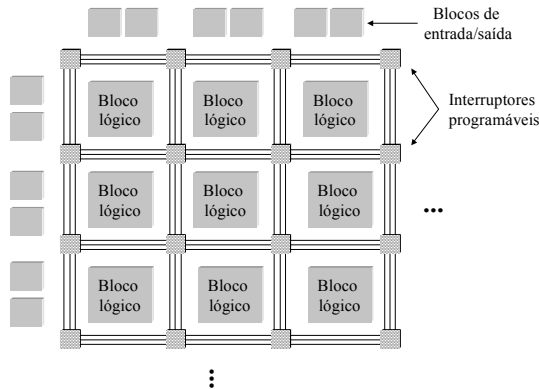


Fig. 2 – Arquitectura típica de FPGA

Uma das maiores vantagens das FPGAs é a sua arquitectura flexível que serve muito bem para uma ampla gama de aplicações. Estas aplicações incluem implementação de controladores de dispositivos, circuitos de codificação, lógica arbitrária, prototipagem e emulação de sistemas, etc. As FPGAs recentes passaram a incorporar várias estruturas heterogêneas tais como blocos de memória, o que possibilita a implementação de sistemas completos num único encapsulamento.

A. Sequência de projecto

No caso de FPGAs, o desenvolvimento de um circuito inclui três fases principais: especificação, implementação e verificação (ver fig. 3). Descrevemos cada uma destas fases em mais detalhe.

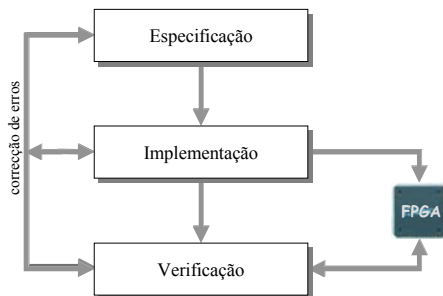


Fig. 3 – Sequência de passos necessários para projectar e implementar um circuito em FPGA

A.1. Especificação

A especificação do circuito a ser implementado numa FPGA, pode ser feita com a ajuda de linguagens de descrição de hardware (VHDL, Verilog, ABEL, etc.) ou através de criação de um diagrama esquemático com uma ferramenta gráfica de projecto assistido por computador.

Os blocos básicos utilizados no diagrama esquemático são descritos numa HDL (*Hardware Description Language*), extraídos das bibliotecas específicas para cada família de FPGA ou criados pelos geradores de componentes parametrizáveis (tais como *LogiBLOX* e *CORE Generator* da Xilinx). A utilização de componentes IP (*Intellectual Property*) aumenta significativamente a produtividade. Existem também ferramentas gráficas que permitem descrever o funcionamento em forma de diagramas de transição de estados.

Recentemente, tornou-se possível especificar a funcionalidade de circuitos numa linguagem de alto nível, tal como Handel-C [3] ou SystemC [4]. Por exemplo, SystemC é uma biblioteca de classes que possibilita a modelação de sistemas digitais recorrendo à tecnologia orientada por objectos. O compilador *CoCentric* da Synopsys sintetiza a descrição em SystemC na descrição RTL (*Register Transfer Level*) em VHDL ou Verilog [5]. O ambiente DK1 da Celoxica permite a compilação directa da descrição em Handel-C para VHDL estrutural ou para EDIF (*Electronic Design Interchange Format*). Note-se que os circuitos descritos a alto nível e criados com a ajuda de ferramentas automáticas ocupam normalmente uma maior área e são mais lentos que os projectados à mão. Mas as ferramentas automáticas facilitam e aceleram significativamente o desenvolvimento de sistemas digitais. Para além disso, as linguagens de alto nível geram descrições mais portáveis possibilitando a implementação mais simples em várias arquitecturas de FPGAs.

Exemplos de uso de várias ferramentas para projecto de sistemas reconfiguráveis podem ser encontrados em [6, 7].

A.2. Implementação

No caso das FPGAs o processo de implementação incorpora três etapas. Primeiro, efectua-se o *mapeamento* das estruturas do projecto nos blocos lógicos existentes em FPGA. A seguir, os blocos resultantes devem ser *colocados* no hardware reconfigurável. Cada um destes blocos fica atribuído a um local específico da FPGA, procurando os algoritmos de colocação que este fique junto daqueles blocos lógicos com os quais comunica.

Uma das técnicas que é utilizada para aliviar os custos de colocação (com o aumento de capacidade de FPGAs esta operação passa a consumir muito tempo) é o *floorplanning* [8]. O algoritmo de *floorplanning* primeiro organiza em grupos aqueles blocos lógicos que requerem uma comunicação muito intensiva. Os grupos são colocados em FPGA e, a seguir, efectua-se a colocação detalhada de blocos lógicos individuais dentro dos limites atribuídos ao grupo. Assim, a colocação completa reduz-se a um conjunto de problemas locais de dimensões menores facilitando deste modo a resolução da tarefa.

Finalmente, realiza-se o *encaminhamento* do circuito que interliga os vários componentes reconfiguráveis. Os recursos de encaminhamento disponíveis são sempre limitados, portanto o objectivo é minimizar o número de segmentos de pistas de ligação utilizados. Uma colocação

eficiente é essencial para que esta fase seja bem sucedida porque caso os componentes interligados fiquem colocados longe uns dos outros, as ligações respectivas vão precisar de muitos recursos de encaminamento.

Se um sistema utilizar mais que uma FPGA, a sequência de implementação torna-se ainda mais difícil. Neste caso, o projecto deve ser distribuído entre várias FPGAs. Isto é conseguido ao colocar porções menos interligadas do circuito em FPGAs separadas. Depois, o encaminamento global determina as ligações entre as FPGAs que é seguido pelo encaminamento detalhado que atribui segmentos de pistas de ligação a cada sinal. A comunicação entre FPGAs distintas deve ser minimizada para reduzir o número de ligações com atrasos grandes. Para ultrapassar restrições no número de pinos utilizam-se frequentemente as técnicas de multiplexagem no tempo [9].

A.3. Verificação

Existem três tipos básicos de verificação do projecto para FPGAs. Primeiro, realiza-se a simulação funcional que ocorre durante a fase de especificação. O objectivo da simulação funcional é verificar a funcionalidade lógica do circuito.

Depois das fases de mapeamento, colocação e encaminamento, efectua-se a simulação temporal que já pode ter em conta todos os atrasos lógicos e os de encaminamento. Para que isto seja possível, é preciso efectuar a re-anotação da informação física ao projecto lógico.

Finalmente, depois de configurar a FPGA, é possível verificar a implementação física do circuito. Para tal recorre-se normalmente ao uso de um analisador lógico, encaminhando os sinais que se pretende verificar para os pinos externos da FPGA.

Uma metodologia diferente que surgiu recentemente, consiste em incorporar no projecto um núcleo do analisador lógico interno e utilizar os blocos de memória embutidos para guardar valores dos sinais necessários que são posteriormente transferidos para um computador para os analisar. Este método possui algumas vantagens relativamente ao anterior porque a comunicação com o computador efectua-se via interface JTAG (*Joint Test Action Group*) não ocupando deste modo os pinos adicionais da FPGA. Um exemplo desta metodologia são as ferramentas *ChipScope Pro* da Xilinx [10, 11].

B. Classificação de FPGAs

Várias arquitecturas de FPGAs são classificadas de acordo com a granulosidade (i.e. tamanho e complexidade) dos seus blocos lógicos e com a estrutura de ligações programáveis [12].

B.1. Blocos lógicos

Os blocos lógicos da FPGA devem ser capazes de implementar várias funções lógicas e podem ter

complexidade diversa. Normalmente, as arquitecturas de FPGAs são baseadas nos elementos seguintes: pares de transístores, portas lógicas simples, multiplexadores e LUTs (*Look-Up Tables*). Portanto, os blocos lógicos, compostos por estes elementos, diferem bastante em termos do tamanho e das capacidades de implementação de funções lógicas. Estas diferenças são normalmente classificadas por *granulosidade*.

A granulosidade de um bloco lógico pode ser definida de maneiras diversas, tais como número de funções booleanas que este pode implementar, número de portas lógicas *NAND* equivalentes, número total de transístores, ou número de entradas/saídas [12]. De acordo com estas características, as arquitecturas de blocos lógicos são divididas em duas categorias: as de *granulosidade fina* e as de *granulosidade grossa*.

Os blocos lógicos de *granulosidade fina* podem implementar funções elementares servindo deste modo muito bem às manipulações ao nível de bits individuais e atingindo um grande nível de utilização de recursos lógicos disponíveis. Uma desvantagem de blocos lógicos finos é que estes requerem muitos segmentos de pistas de ligação e interruptores programáveis. É de notar também que as FPGAs de granulosidade fina possuem muitos pontos de configuração e precisam portanto de mais bits para serem reconfiguradas [8].

Os blocos lógicos de *granulosidade grossa* têm normalmente muitas entradas e servem bem à implementação de funções complexas. Dado que tais blocos lógicos são optimizados para funções lógicas mais complexas, estas operações são executadas mais rapidamente e consomem menos área do que um conjunto de blocos lógicos elementares interligados de maneira adequada. Contudo, se for necessário realizar operações lógicas elementares, a arquitectura de granulosidade grossa vai sofrer de uma baixa utilização dos recursos disponíveis.

É de salientar que a maioria das FPGAs disponíveis no mercado utilizam blocos lógicos de granulosidade *média* a fim de minimizar as desvantagens inerentes aos dois casos extremos descritos acima.

B.2. Recursos de encaminamento

Os recursos de encaminamento ocupam uma área da FPGA que é muito maior que a usada pelos recursos lógicos [8]. A arquitectura de encaminamento de FPGAs define como os interruptores programáveis e os segmentos de pistas de ligação de comprimentos variáveis são utilizados a fim de interligar os blocos lógicos. O número e a distribuição de segmentos de pistas incorporados afecta a densidade e o desempenho atingíveis na FPGA. Caso este número seja inadequado só uma fracção dos blocos lógicos poderá ser utilizada. Contudo, se o número de segmentos for excessivo, isto pode aumentar o tamanho da FPGA e resultar em eficiência reduzida de utilização do silício.

A estrutura de encaminamento da FPGA deve ser capaz de acomodar todas as ligações de uma aplicação típica e

assegurar a velocidade de funcionamento adequada. Um factor importante no desempenho de FPGAs é o atraso de propagação através dos recursos de encaminhamento atribuídos, dado que algumas ligações requerem caminhos mais compridos que as outras. Para além disso, qualquer interruptor programável (antifusível, SRAM ou transístor EPROM) possui também uma resistência e capacitância que provocam atrasos adicionais.

Existem dois métodos principais de atribuir os recursos de encaminhamento: *segmentado* e *hierárquico* [8]. No encaminhamento segmentado as pistas de ligação curtas servem para as comunicações locais. Estas podem ser interligadas com a ajuda de interruptores a fim de estabelecer ligações longas. Para além disso, existem normalmente pistas de ligação mais compridas que servem para encaminhar sinais a distâncias longas sem ter de passar por interruptores.

Na estrutura de encaminhamento hierárquico os blocos lógicos são organizados em grupos. Primeiro, efectua-se a interligação de blocos pertencentes a um grupo com a ajuda de pistas curtas. A seguir, utilizam-se pistas compridas para interligar vários grupos de blocos lógicos. Esta estratégia pode ser repetida várias vezes através da organização de uma série de grupos em conjuntos maiores e a sua interligação posterior.

B.3. Estruturas heterogéneas

A fim de assegurar melhor desempenho e flexibilidade na computação, as FPGAs recentes passaram a incluir estruturas heterogéneas, i.e. para além dos componentes típicos que fazem parte de qualquer FPGA, utilizam-se blocos específicos.

Por exemplo, é bastante difícil implementar em FPGA a operação de multiplicação de uma maneira eficiente. Portanto, foram criadas unidades de multiplicação optimizadas e embutidas na estrutura de algumas FPGAs.

A maioria das arquitecturas actuais incluem blocos de memória dedicados que servem para guardar dados que são utilizados frequentemente. Para além disso, com a ajuda dos blocos de memória embutidos pode-se depurar circuitos em *run-time* conforme descrito na secção II.A.3.

As FPGAs recentes incluem também transmissores/receptores de alto débito (múltiplos Gbits) e núcleos de processadores o que aumenta a gama de aplicações possíveis nas áreas de telecomunicações e de processamento de sinal e imagem.

Dado que as FPGAs são uma tecnologia relativamente recente, as suas arquitecturas estão sujeitas a uma evolução constante [2, 13-17]. Algumas das características das FPGAs disponíveis comercialmente estão sumariadas na tabela 1.

Tabela 1. Características principais de FPGAs disponíveis comercialmente

Companhia	Família	Tecnologia de programação	Granulosidade	Densidade (portas de sistema)	Estruturas heterogéneas
Actel	Axcelerator	antifusível	fina	até 2M	SRAM embutida (até 330 Kbits)
	ProASIC ^{PLUS}	Flash	fina	até 1M	SRAM embutida (até 198 Kbits)
Altera	Stratix	SRAM	média	até 114K de elementos lógicos ³	SRAM (até 10 Mbits), blocos DSP
Atmel	AT40K	SRAM	média	até 50K	--
Lattice	ORCA	SRAM	grossa	até 900K	RAM embutida (até 148 Kbits)
Quick-Logic	pASIC3	antifusível	média	até 75K	--
	Eclipse-II	antifusível	média	até 370K	SRAM (até 55Kbits) e unidades computacionais embutidas
Xilinx	XC4000XL	SRAM	média	até 180K	--
	Virtex-EM	SRAM	média	até 3M	SRAM embutida (até 1120 Kbits)
	Virtex-II	SRAM	média	até 8M	SRAM (até 3 Mbits), multiplicadores
	Virtex-II Pro	SRAM	média	--- ⁴	SRAM (até 10 Mbits), multiplicadores, processadores PowerPC, transmissores/receptores de múltiplos Gbits embutidos

³ Dados sobre as portas de sistema estão indisponíveis

⁴ Por causa de existência de várias estruturas heterogéneas não faz sentido expressar a densidade em termos de portas de sistemas

C. Recentes famílias de FPGAs da Xilinx

Os líderes de vendas de FPGAs são Xilinx [2] e Altera [14]. Como no Departamento de Electrónica e Telecomunicações da Universidade de Aveiro são utilizados exclusivamente os dispositivos da Xilinx, descrevemos as famílias mais avançadas produzidas por esta companhia em mais detalhe.

Actualmente a Xilinx produz FPGAs de séries Spartan, Virtex e Virtex-II, nomeadamente as famílias Spartan, Spartan-XL, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-EM, Virtex-II e Virtex-II Pro, todas baseadas em SRAM [2].

A série Spartan tem como objectivo principal fornecer desempenho elevado, memória embutida e suporte robusto para IP a preço reduzido. As séries Virtex e Virtex-II abrangem dispositivos mais complexos que possibilitam a implementação de sistemas completos num único encapsulamento (ver fig. 4). A densidade elevada combinada com os blocos de memória embutidos, as tecnologias avançadas de gestão de relógio e suporte para uma ampla gama de normas de I/O (*Input/Output*), permitem aos utilizadores atingir níveis de desempenho que anteriormente só eram viáveis com os ASICs.

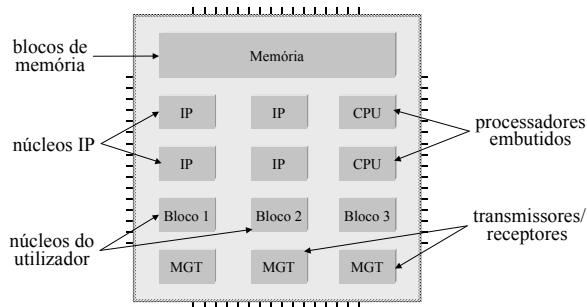


Fig. 4 – Implementação de um sistema num único encapsulamento com a ajuda de FPGAs da família Virtex-II Pro

Dado que a família Virtex-II Pro inclui os dispositivos mais poderosos descrevemo-la em mais detalhe. Os componentes básicos da arquitectura Virtex-II Pro estão ilustrados na fig. 5. As FPGAs desta família possuem as seguintes características principais [2]:

- Até 14416 blocos lógicos (CLBs - *Configurable Logic Blocks*) organizados num *array* de 136×106. A densidade das FPGAs da família Virtex-II Pro atinge 10M de portas de sistema. É de notar que o número de portas de sistema não reflecte as capacidades das estruturas heterogéneas existentes tais como processadores e transmissores/receptores embutidos.

- Até 10008 Kbits de memória embutida e até 1738 Kbits de memória distribuída. A memória embutida consiste em 556 (no máximo) blocos *SelectRAM* de 18 Kbits cada. Os blocos são programáveis em várias configurações entre 16K×1 bit e 512×36 bits.

- Suporte para mais que 25 normas de I/O (tecnologia *Select I/O-Ultra*); até 1200 entradas/saídas.

- Entre 4 e 12 blocos de gestão de relógio - DCMs (*Digital Clock Managers*). Os DCMs asseguram a síntese flexível da frequência do relógio, a diferença precisa de fases, etc.

- Até 556 multiplicadores dedicados (18 bits×18 bits). Os multiplicadores estão associados a cada bloco de memória embutido e são optimizados para operações baseadas no seu conteúdo.

- Até 24 transmissores/receptores MGT (*Multi-Gigabit Transceivers*) que suportam velocidades de transferência de dados entre 622 Mbits/s e 3.125 Gbits/s.

- Até 4 núcleos de processadores IBM PowerPC 405 embutidos que podem funcionar a 300 MHz. Os processadores possibilitam a implementação de sistemas embutidos complexos e permitem partilhar o projecto de uma maneira óptima atribuindo à lógica da FPGA porções computacionalmente intensivas da aplicação ficando os processadores com as porções orientadas ao controlo. Deste modo, software e hardware podem ser desenvolvidos em paralelo, o que não tem acontecido tradicionalmente, quando era impossível começar o desenvolvimento de software antes de ter um protótipo de hardware.

- Suporte para reconfiguração parcial. Todos os dados de configuração bem como os conteúdos de todos os *flip-flops/latches*, LUTs e blocos de memória embutidos podem ser lidos da FPGA possibilitando deste modo a depuração em *run-time*.

- Suporte completo pela ferramenta ISE 5.x da Xilinx incluindo o sistema *CORE Generator* com uma série de núcleos de IP.

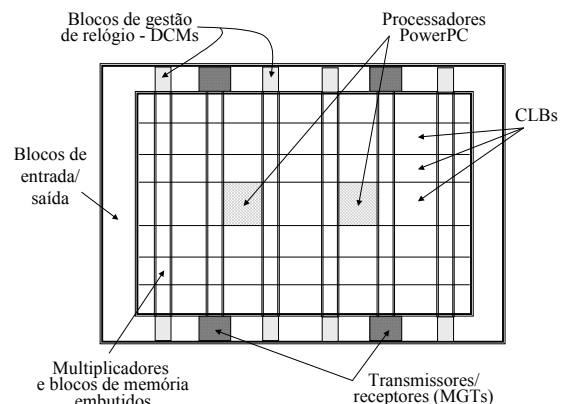


Fig. 5 – Componentes básicos da arquitectura Virtex-II Pro

III. SISTEMAS RECONFIGURÁVEIS

Embora o conceito de computação reconfigurável exista há já bastante tempo [18, 19], apenas recentemente surgiram tecnologias que possibilitaram a sua implementação e aplicação na prática. O interesse

começou no início dos anos 90 quando a densidade das FPGAs ultrapassou as 10K portas lógicas [20]. Desde aí, a computação reconfigurável, devido à possibilidade de acelerar várias aplicações, tornou-se objecto de investigação intensiva. A sua característica mais importante é a capacidade de realizar computações em hardware com o objectivo de incrementar o desempenho, ficando ao mesmo tempo com a flexibilidade do software [21].

A fim de atingir desempenho elevado e suportar uma ampla gama de aplicações, os sistemas reconfiguráveis são normalmente compostos pela lógica reconfigurável e um processador de uso geral. Para executar a aplicação de uma maneira mais eficiente, aquelas partes que não são facilmente mapeadas na lógica reconfigurável, são executadas num processador hospedeiro, enquanto as partes que requerem computações muito intensivas e podem beneficiar da sua implementação em hardware, são realizadas em FPGAs. Assim, um sistema reconfigurável típico consiste em lógica reconfigurável (denominada por *componente variável*) e num processador hospedeiro (referenciado como *componente fixo*) conforme representado na fig. 6.

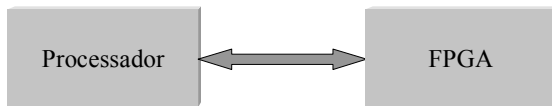


Fig. 6 – Estrutura de um sistema reconfigurável típico

Descrevemos seguidamente as propriedades mais importantes que caracterizam os sistemas reconfiguráveis.

A. Modos de reconfiguração

O tempo de uma operação realizada num sistema reconfigurável é a soma do tempo de configuração e do tempo de execução. A configuração é feita normalmente pelo processador hospedeiro que envia os dados de configuração que definem o funcionamento actual do hardware. A configuração pode ser carregada só no início de execução ou periodicamente a fim de variar o funcionamento do sistema. O modelo de execução também varia de sistema para sistema [21]: em alguns sistemas o processador hospedeiro fica suspenso enquanto o hardware reconfigurável está activo; em outros é permitida a execução simultânea.

No domínio da computação reconfigurável podemos distinguir entre dois modos de configuração: *estático* e *dinâmico* [22]. A reconfiguração *estática* pressupõe o funcionamento permanente da FPGA depois desta ser configurada (ver fig. 7). De facto, o modo estático não concede grande flexibilidade mas permite atingir um bom desempenho via utilização do hardware optimizado para uma dada aplicação [23].

Às vezes é preciso activar configurações diferentes de acordo com a necessidade corrente de uma aplicação. Isto possibilita mais secções da aplicação a serem mapeadas

em hardware do que o que pode “caber” numa FPGA. Deste modo, a maior parte da aplicação pode potencialmente ser acelerada num sistema reconfigurável resultando num desempenho mais elevado. Este conceito é frequentemente referido por reconfiguração dinâmica a que está associada a noção de *hardware virtual*. Neste caso, os recursos disponíveis na FPGA são bastante menores que o conjunto de recursos necessários para todas as configurações. Mas em vez de reduzir o número de configurações mapeadas, estas são transferidas para a FPGA de acordo com a necessidade actual [8]. Uma das vantagens dos sistemas reconfiguráveis dinamicamente comparando-os com os sistemas programáveis só no início da aplicação, é a potencialidade de efectuar optimizações do hardware com base na informação determinada durante a execução.

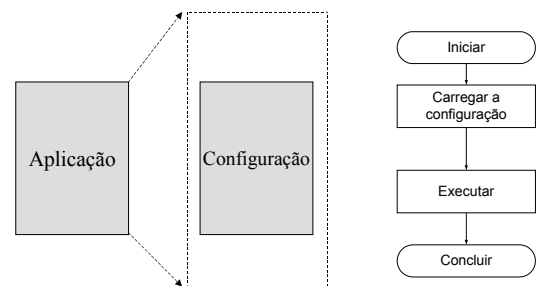


Fig. 7 – Reconfiguração estática

A reconfiguração dinâmica pode, por sua vez, ser dividida em reconfiguração global e reconfiguração parcial. A reconfiguração *global* reserva todos os recursos de hardware para cada fase de execução. Depois de uma fase ser concluída, todos os recursos da FPGA são reconfigurados para a fase seguinte (ver fig. 8).

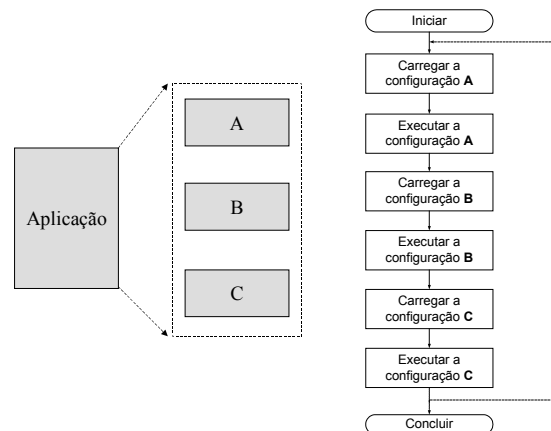


Fig. 8 – Reconfiguração dinâmica global

A reconfiguração *parcial* recorre à modificação selectiva dos recursos de hardware ao longo de execução de uma aplicação (ver fig. 9). Esta flexibilidade permite que o hardware seja mais personalizado às necessidades correntes da aplicação. A reconfiguração parcial exige que

só os recursos seleccionados sejam reprogramados o que resulta num *overhead* de configuração menor do que no caso anterior.

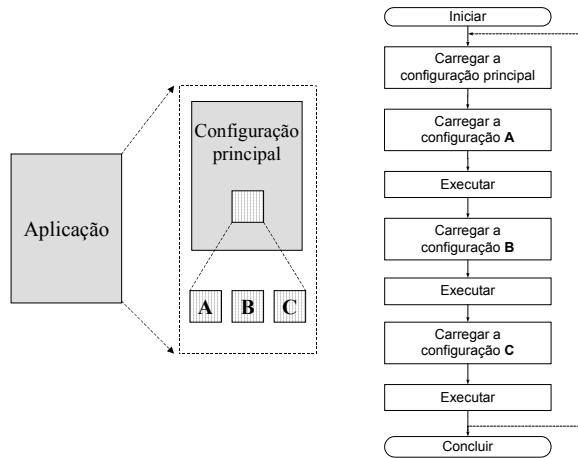


Fig. 9 – Reconfiguração dinâmica parcial

É de notar que existem outros termos que são utilizados para designar reconfiguração estática e dinâmica, por exemplo “ligação durante o projecto” (*design-time binding*) e “ligação durante a execução” (*implementation-time binding*) [24]; “configuração em *compile-time*” e “configuração em *run-time*” [25].

A fim de possibilitar a reconfiguração dinâmica existem três tipos de dispositivos disponíveis [21]:

1) *Dispositivos de contexto único* requerem uma reconfiguração completa para alterar qualquer um dos bits de programação (ver fig. 10a). A maioria das FPGAs disponíveis no mercado pertencem a esta classe. É de notar que existe uma técnica especial que permite a reprogramação parcial deste grupo de FPGAs (esta baseia-se em modelos - *hardware templates* [26]).

2) *Dispositivos de contexto múltiplo* possuem várias camadas de bits de programação estando em cada instante activa apenas uma camada (ver fig. 10b). A vantagem principal destes dispositivos é a possibilidade de mudar de contexto de uma maneira extremamente rápida. Para além disso, é permitida a configuração nos bastidores (*background*) possibilitando reprogramar um contexto enquanto um outro contexto está activo.

3) *Dispositivos parcialmente reconfiguráveis* permitem que áreas pequenas da FPGA sejam modificadas sem necessidade de reprogramar todo o dispositivo (ver fig. 10c). As reconfigurações parciais requerem bastante menos tempo do que a reconfiguração total. Frequentemente, as zonas da FPGA que não são perturbadas podem continuar a execução permitindo que a reconfiguração seja realizada em simultâneo com a execução [8].

Uma característica importante a ter em conta em sistemas reconfiguráveis dinamicamente, é que o tempo gasto em reconfiguração não deve absorver a aceleração ganha. Portanto, aplicam-se várias técnicas destinadas a

reduzir o *overhead* da reconfiguração [21]. Estas incluem utilização de métodos de compressão de dados de configuração; carregamento dos ficheiros de configuração em *background*, enquanto o processador principal está a executar uma outra parte da aplicação; etc.

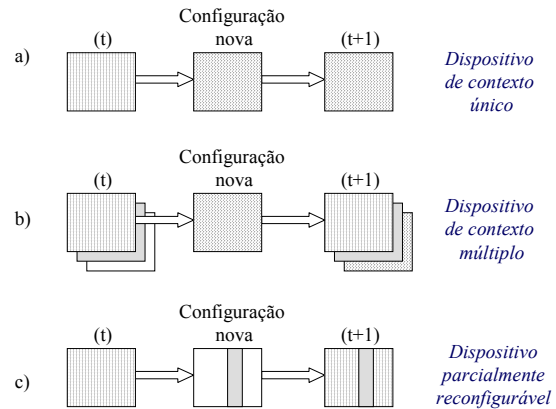


Fig. 10 – Vários tipos de dispositivos reconfiguráveis

B. Mecanismo de interacção entre o hardware reconfigurável e o processador hospedeiro

O mecanismo de interacção define o nível de “proximidade” dos componentes fixo e variável dum sistema reconfigurável [8, 27, 28]. No caso mais simples, o hardware reconfigurável pode representar uma unidade de processamento externa que comunica com o processador hospedeiro raramente ou até funciona de uma maneira independente (ver fig. 11a). Se a unidade reconfigurável for utilizada como um acelerador, então o processador hospedeiro inicializa o hardware e envia-lhe todos os dados necessários [29-31]. O acelerador efectua todas as computações independentemente do processador principal e no final retorna os resultados, podendo ambos os componentes executar em simultâneo.

Existem arquitecturas em que ambos os componentes principais (i.e. a lógica reconfigurável e o processador) residem na mesma pastilha de silício e são interligados muito fortemente (ver fig. 11b) [32-34]. O hardware reconfigurável é usado apenas para implementar unidades funcionais reconfiguráveis dentro do processador hospedeiro. Neste caso as unidades reconfiguráveis actuam como unidades funcionais no *datapath* do processador sendo os operandos guardados em registos. Este modelo permite estender a programação tradicional com a capacidade de definir e adicionar instruções personalizadas. Arquitecturas deste tipo beneficiam do nível próximo de interacção resultando num baixo *overhead* de comunicação. Contudo, tais sistemas são de projecto e implementação difíceis e sofrem frequentemente de limitações no acesso à memória.

A fim de explorar todo o potencial da lógica reconfigurável é necessário assegurar a largura de banda adequada na comunicação com a memória (ver fig. 11c).

Sistemas nesta categoria permitem que a lógica reconfigurável tenha acesso directo à memória em vez de transferir todos os dados necessários via registos do processador hospedeiro [27, 35-37]. A fim de possibilitar isso normalmente cria-se uma memória local (semelhante a *cache*) que tem ligação directa eficiente com a memória principal. A integração directa da memória e da lógica reconfigurável aumenta a quantidade de dados disponíveis para os recursos reconfiguráveis possibilitando deste modo explorar melhor o paralelismo e aumentar o desempenho. Como foi descrito na secção II.B.3, todas as FPGAs recentes incorporam grandes blocos de memória embutidos que podem ser utilizados para estes fins.

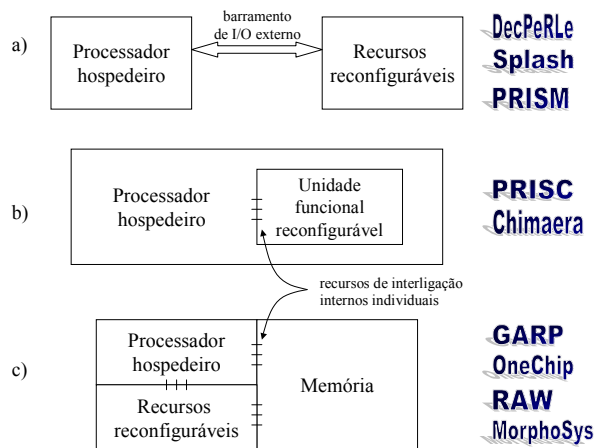


Fig. 11 – Diferentes tipos de mecanismos de interação entre o hardware reconfigurável e o processador hospedeiro

O nível de “proximidade” dos componentes principais de um sistema reconfigurável influencia a largura de banda suportada na comunicação e afecta o modelo de programação do sistema. Assim, sistemas reconfiguráveis fortemente interligados servem bem para a aceleração em geral (execução de ciclos internos, etc.). Sistemas com fraca interligação são adequados para processamento idêntico de grandes volumes de dados. Portanto, aceleradores e coprocessadores são muito importantes para algumas classes de aplicações [28].

C. Capacidade lógica

O número de FPGAs envolvidas num sistema reconfigurável pode variar de 1 a algumas centenas. A capacidade lógica determina a complexidade das aplicações que podem ser mapeadas em hardware. Por exemplo, o detector *DZero* (utilizado na área de física subatómica) inclui um *array* de 582 FPGAs das famílias Spartan, Virtex e Virtex-E da Xilinx, capaz de processar 1.5 *terabytes* de dados por segundo [38].

A utilização de sistemas compostos por múltiplas FPGAs requer a definição de uma boa topologia de encaminhamento e a existência de ferramentas de projecto assistido por computador capazes de dividir circuitos em porções interligadas de uma maneira eficiente, que

permitam atingir um nível adequado de utilização dos recursos lógicos.

D. Modelo de programação

Existem várias maneiras de mapear uma aplicação num sistema reconfigurável. No caso ideal, um compilador devia compilar automaticamente um programa escrito numa linguagem de programação de alto nível (assim como C++) numa configuração a ser carregada na FPGA. É de notar contudo que linguagens de programação convencionais não podem ser utilizadas directamente para estes fins por não suportarem noções como processamento paralelo e ser bastante difícil identificar o paralelismo automaticamente. Contudo, há sistemas reconfiguráveis que são programados deste modo [35]. A vantagem principal desta técnica é que o utilizador não precisa de ter conhecimentos sobre a arquitectura do sistema bem como sobre o hardware em geral, e a utilização da lógica reconfigurável fica transparente.

Uma abordagem diferente consiste no desenvolvimento de hardware reconfigurável de acordo com as necessidades de uma dada aplicação. Para tal utiliza-se o fluxo de projecto típico descrito na secção II.A. Esta técnica requer que o utilizador tenha experiência em projecto de circuitos de hardware. Embora o tempo total de implementação de uma aplicação seja neste caso maior, os resultados podem ser melhores dado que o projectista tem mais flexibilidade e mais opções a explorar.

É de salientar que cada uma das técnicas descritas acima possui as suas vantagens e desvantagens. Normalmente a compilação automática utiliza-se em arquitecturas orientadas para aceleração de aplicações gerais, enquanto para aplicações específicas podem ser atingidos melhores resultados com a segunda abordagem. A fig. 12 ilustra as três maneiras possíveis de mapear uma aplicação em arquitecturas convencionais e sistemas reconfiguráveis.

E. Modelo de execução

Uma aplicação pode ser inteiramente implementada em hardware reconfigurável (cabendo ao processador hospedeiro apenas tarefas de inicialização da FPGA) ou pode ser partilhada entre o hardware e o software. Existem vários métodos de partilhar a aplicação. Uns implementam em FPGA instruções específicas estendendo deste modo o conjunto básico de instruções do processador hospedeiro [32, 34]. Outros métodos efectuam a partição de acordo com a complexidade da função a executar. Por exemplo, partes computacionalmente intensivas do programa são atribuídas à FPGA enquanto as partes que exibem pouco paralelismo são processadas pelo processador de uso geral [39]. Sistemas deste tipo baseiam-se na regra 90/10, que diz que 90% do tempo de execução dum programa é despendido por 10% do seu código. A fim de atingir resultados significativos tenta-se acelerar esta pequena porção do código com a ajuda de FPGA. Os 90% restantes do código possuem pouco paralelismo e são

melhor executados num processador de uso geral. Sendo assim, o processador é responsável pelas tarefas sequenciais enquanto à FPGA são atribuídas tarefas computacionalmente intensivas [40]. Num terceiro tipo de métodos a aplicação é partilhada de acordo com a capacidade lógica do componente variável do sistema

[23]. Neste caso, se uma aplicação (ou alguma parte desta) não “cabe” na FPGA disponível, deve ser primeiro processada pelo processador hospedeiro e depois transferida para a FPGA. A partição pode ser executada de uma maneira manual (ver fig. 12b) ou automática (ver fig. 12c).

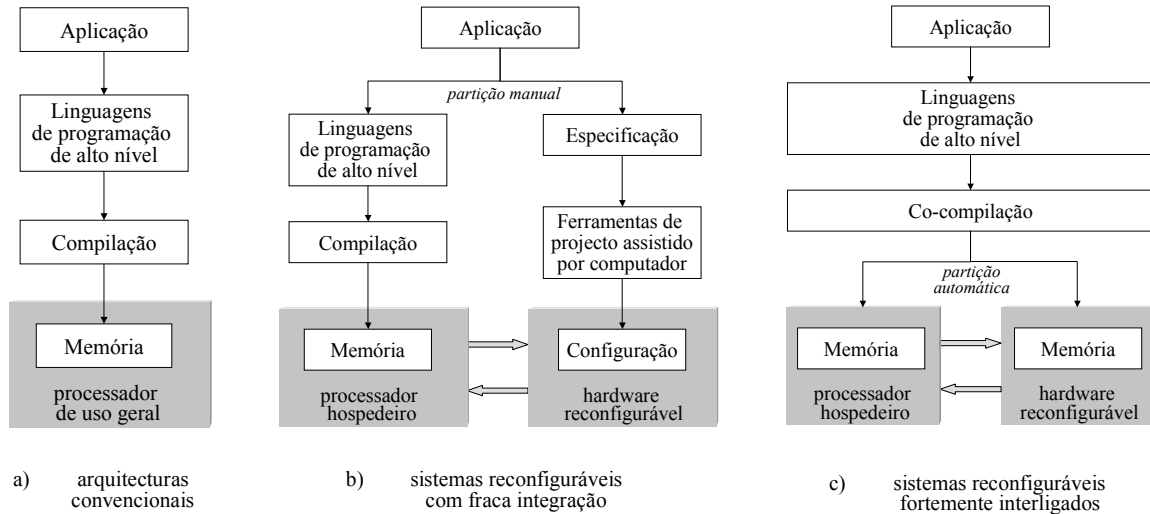


Fig. 12 – Modelos de programação e de execução de uma aplicação em várias plataformas computacionais

IV. TÉCNICAS BÁSICAS UTILIZADAS NA COMPUTAÇÃO RECONFIGURÁVEL A FIM DE ATINGIR UM DESEMPENHO ELEVADO

Note-se que a frequência de relógio suportada por FPGAs recentes é aproximadamente uma ordem de grandeza menor do que a frequência utilizada em computadores de uso geral, que actualmente se situa em alguns GHz. Isto explica-se parcialmente pela necessidade de acomodar interligações programáveis. Portanto, o mapeamento directo de uma aplicação do software para uma FPGA não irá assegurar um desempenho mais elevado que o atingido num computador de uso geral. A fim de conseguir um bom desempenho, são normalmente aplicadas as três técnicas seguintes [41]:

1) *Largura de banda elevada no acesso à memória.* Tradicionalmente, os computadores de uso geral organizam a memória em forma de um conjunto de palavras de tamanho fixo. Os dados de um problema podem não caber exactamente numa só palavra, portanto são precisos vários acessos à memória para os processar. Contudo, em FPGA é possível organizar a memória de acordo com a dimensão actual dos dados, permitindo que estes sejam processados numa única operação.

2) *Uso de unidades funcionais específicas e optimizadas.* O conjunto de instruções básicas utilizado em computadores de uso geral foi desenvolvido para servir uma ampla gama de aplicações. Portanto, algumas operações específicas podem precisar de muitas instruções para serem expressas, e levar muitos ciclos de relógio para serem concluídas. Frequentemente, as aplicações

envolvem operações bastante simples mas estas não são bem suportadas por ALUs convencionais. Com FPGAs torna-se possível criar unidades funcionais optimizadas para certas operações e tamanhos de dados (necessários para uma aplicação particular). Como resultado, a unidade é capaz de executar num ciclo de relógio algumas operações que requerem vários ciclos num computador de uso geral. Bons exemplos de operações deste tipo são as que funcionam a nível de bits individuais e envolvem expressões lógicas relativamente complexas.

3) *Exploração de paralelismo e de pipelining.* As unidades funcionais descritas acima são normalmente simples e ocupam poucos recursos de hardware, portanto é possível reproduzi-las para explorar o paralelismo. Para além disso, as FPGAs típicas incorporam um grande número de *flip-flops* e LUTs distribuídos por todo o dispositivo que podem ser utilizados para permitir uma gestão flexível de registos construídos com base nestes elementos de memória. Isto possibilita a implementação de técnicas de escalonamento (*pipelining*) eficientes que resultam em melhor desempenho, aumentam o nível de utilização dos recursos de hardware e reduzem acessos à memória externa.

V. EXEMPLOS DE SISTEMAS RECONFIGURÁVEIS

Durante os últimos 10-15 anos foram propostas muitas arquitecturas de sistemas reconfiguráveis. Estas diferem bastante em termos de mecanismos de interacção dos componentes principais e dos modelos de programação e de execução. Descrevemos alguns exemplos de sistemas

mais conhecidos agrupando-os de acordo com o mecanismo de interacção adoptado.

A. Sistemas reconfiguráveis com fraca interligação

A.1. DECPeRLe

Um dos primeiros trabalhos que se enquadram no paradigma de computação reconfigurável é o DECPeRLe-1 também conhecido sob o nome mais geral de PAM (*Programmable Active Memories*) [29]. DECPeRLe-1 é um coprocessador ligado ao barramento de I/O de um processador hospedeiro. O processador hospedeiro deve carregar um ficheiro de configuração no PAM e a seguir, pode ler e escrever no PAM como num módulo de memória, enquanto o PAM processa os dados entre as instruções de leitura e escrita.

DECPeRLe-1 foi implementado numa placa que contém 16 FPGAs XC3090 da Xilinx organizadas num *array* de 4x4 (ver fig. 13). Cada FPGA tem ligações directas às 4 FPGAs-vizinhas. Para além disso existem barramentos comuns. DECPeRLe-1 inclui também 4 bancos de memória (4MB no total). As aplicações implementadas que utilizaram a memória incorporada funcionaram a uma frequência de ~25 MHz, enquanto para as aplicações que não necessitavam aceder à memória, foram atingidas frequências maiores. DECPeRLe-1 possui 4 conectores, três dos quais servem para estabelecer ligações com os dispositivos externos e o último destina-se a estabelecer a interface com o processador hospedeiro. Para descrever circuitos foi escolhida a linguagem C++ complementada com uma biblioteca proprietária.

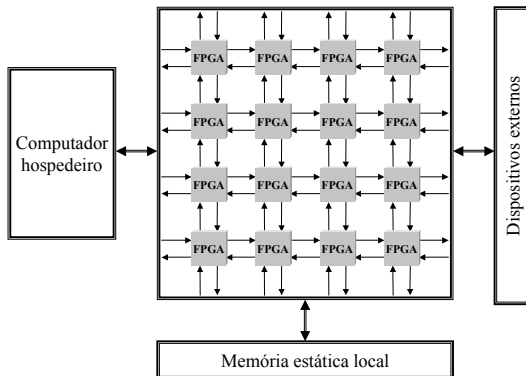


Fig. 13 – Arquitectura DECPeRLe-1

A.2. Splash

A série de sistemas reconfiguráveis Splash, que vieram a ser explorados como uns dos recursos computacionais do *Center for Computing Sciences* (anteriormente, *Supercomputing Research Center*) em Maryland, EUA, também é muito conhecida [30]. Splash-2 foi implementado numa placa com 17 FPGAs XC4010 da Xilinx interligadas num *array* linear. Para além das

ligações entre as FPGAs adjacentes existem interruptores para comunicações gerais limitadas entre as FPGAs que não são vizinhas. Cada FPGA foi ligada a um banco de RAM com a configuração 256Kx16 bits. Uma ou mais placas com as FPGAs podem ser ligadas a um computador hospedeiro via um barramento especial. Aplicações executadas em Splash-2 incluem processamento de sinal, pesquisa no texto, processamento de padrões de ADN, etc.

A.3. PRISM

Uma das primeiras arquitecturas na qual FPGAs interferiram directamente com um processador foi o PRISM (*Processor Reconfiguration through Instruction Set Metamorphosis*) [31]. PRISM liga um microprocessador 68010 da Motorola que funciona a 10 MHz a um *array* de 4 FPGAs XC3090 da Xilinx. O processador é complementado com um conjunto de instruções de hardware que representam um subconjunto de funções de C do programa executado no sistema. A aceleração é limitada pela interface processador-coprocessador embora a comunicação fosse restringida à passagem de argumentos e dos resultados respectivos.

B. Sistemas em que a lógica reconfigurável implementa unidades funcionais no datapath do processador

B.1. PRISC

A arquitectura PRISC (*Programmable Instruction Set Computers*) [32] baseia-se num processador RISC (*Reduced Instruction Set Computer*) complementando a funcionalidade do *datapath* com a lógica reconfigurável sendo esta integrada no processador através de adição de uma unidade funcional programável – PFU (*Programmable Functional Unit*) às unidades funcionais fixas (ver fig. 14). A PFU é implementada com base em LUTs. A complexidade das funções realizadas na PFU é de molde a que a sua latência não excede o tempo de um ciclo do processador. Várias imagens de PFU são pré-compiladas e activadas através de comutação de contextos. Deste modo o conjunto básico de instruções do processador é estendido com as instruções de hardware implementadas em PFU. A arquitectura requer que uma ferramenta de software extraia automaticamente a função de hardware e gere imagens de hardware adequadas.

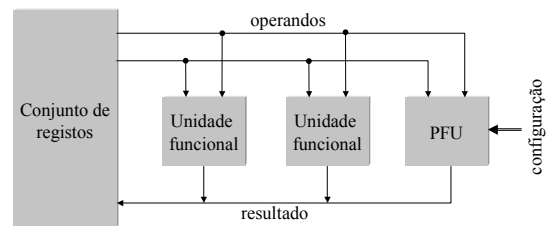


Fig. 14 – Arquitectura PRISC

B.2. Chimaera

Chimaera [34] é um sistema parcialmente reconfigurável em *run-time*. A interface entre o processador e a lógica reconfigurável é semelhante à implementada no PRISC. Contudo, Chimaera permite que o processador e a lógica executem em simultâneo. Componente principal do Chimaera é o *array* reconfigurável, onde todas as computações são realizadas. O *array* é composto por linhas de células lógicas entre as quais passam canais de encaminhamento horizontais. O *array* obtém as suas entradas directamente dos registos do processador hospedeiro. A cada linha do *array* reconfigurável corresponde uma posição da CAM (*Content Addressable Memory*) que indica quais as instruções que já concluíram a sua execução. A CAM analisa todas as instruções a serem executadas recolhendo apenas aquelas que devem ser processadas pelo *array* reconfigurável. Se a instrução lida já está presente no *array*, o seu resultado é enviado para um registo. Caso contrário, o processador é parado enquanto a lógica de controlo carrega a instrução adequada da memória para o *array* reconfigurável.

C. Sistemas reconfiguráveis fortemente interligados

C.1. MorphoSys

A arquitectura MorphoSys [37] consiste num componente reconfigurável (*array* de células reconfiguráveis) combinado com um processador *TinyRISC* e na interface à memória de largura de banda elevada, todos integrados numa mesma pastilha de silício (ver fig. 15). O *array* é organizado em 8×8 células cada uma das quais inclui uma ALU de 28 bits, um multiplicador de 16×12 bits, uma unidade de deslocamento, dois multiplexadores e registos. As células são configuradas por palavras de contexto guardadas numa memória de contexto. A memória de contexto é capaz de armazenar múltiplos conjuntos de dados de configuração (até 32) diminuindo deste modo o tempo despendido em reconfiguração. Uma palavra define a configuração para toda a linha ou toda a coluna do *array* seguindo deste modo o modelo de computação SIMD (*Single Instruction stream, Multiple Data streams*). O funcionamento do *array* reconfigurável é controlado pelo processador através de instruções especiais adicionadas ao conjunto de instruções básico. Para assegurar a interface adequada com a memória MorphoSys inclui um *buffer* de armazenamento interno e um controlador de DMA (*Direct Memory Access*). O sistema é orientado às aplicações computacionalmente intensivas que operam sobre dados de tamanho de uma palavra tais como compressão de vídeo e codificação de dados.

C.2. Garp

A arquitectura Garp [35] pretende integrar numa mesma pastilha de silício um processador MIPS e um

coprocessador programável optimizado para acelerar ciclos em aplicações de software de uso geral (i.e. recorre à regra 90/10). O problema de largura de banda no acesso à memória (dado que os ciclos operam frequentemente sobre estruturas de dados residentes na memória) é resolvido ao dar à lógica reconfigurável acesso à hierarquia de memória do processador hospedeiro.

Para reduzir o tempo de reconfiguração, o coprocessador inclui uma *cache* que mantém configurações recentemente removidas. Depois de configurado, Garp pode executar de uma maneira independente podendo ser interrompido pelo processador a qualquer altura.

O compilador desenvolvido para Garp adapta muitas técnicas propostas previamente para a exploração do paralelismo ao nível de instruções em processadores VLIW (*Very Long Instruction Word*). O fluxo de compilação consiste na identificação de porções do código C que podem ser executadas pelo coprocessador reconfigurável, síntese do código de interface entre hardware/software e das configurações da lógica reconfigurável.

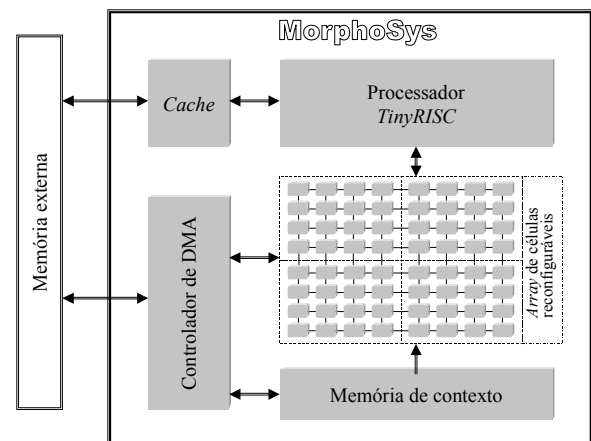


Fig. 15 – Componentes principais da arquitectura MorphoSys

C.3. RAW

O sistema RAW (*Reconfigurable Architecture Workstation*) [36] consiste em múltiplos blocos, sendo cada bloco composto por memória de instruções, memória de dados e um *datapath* com unidade reconfigurável e unidade de processamento de números reais. Todos os blocos estão interligados possibilitando comunicações entre as unidades de processamento vizinhas.

A abordagem adoptada por RAW para atingir uma grande largura de banda no acesso à memória é diferente da utilizada em Garp. Cada processador tem a ligação directa dedicada com a sua própria memória local resultando numa arquitectura do tipo MIMD (*Multiple Instruction streams, Multiple Data streams*). O sistema precisa de efectuar uma distribuição das instruções entre todos os blocos e tratar da comunicação entre estes.

C.4. OneChip

O sistema OneChip [27] é semelhante a PRISC pois utiliza PFU junto com as unidades funcionais fixas. Contudo, OneChip recorre a múltiplas PFUs cada uma das quais não precisa de ser combinatória pura podendo implementar circuitos sequenciais. Portanto, o atraso lógico em OneChip não é limitado a um ciclo de relógio. Para além disso, na arquitectura OneChip o processador fixo, a lógica reconfigurável e a memória são integrados numa única pastilha de silício. O processador e a lógica reconfigurável podem executar em paralelo. Para tal foi proposto um mecanismo que impede o processador de aceder a alguns blocos da memória que são utilizados pelo componente variável, preservando deste modo a consistência da memória. A comunicação com a memória é efectuada através de um controlador que arbitra acessos entre os dois componentes principais. Foi criado um protótipo do OneChip utilizando o sistema Transmogripher-2. Para avaliar a arquitectura OneChip foram escolhidas duas aplicações: a transformada bidimensional discreta do coseno e um filtro FIR (*Finite Impulse Response*).

Outros exemplos de sistemas reconfiguráveis [42] incluem Nano Processor [43], RENCO (*REconfigurable Network COmputer*) [22], SPYDER (*REconfigurable Processor Development SYstem*) [22], HARP [44], MATRIX [45], NAPA (*National Adaptive Processing Architecture*) [46, 47], KressArray [48], etc.

VI. APLICAÇÕES DA COMPUTAÇÃO RECONFIGURÁVEL

Para algumas classes de tarefas, sistemas reconfiguráveis foram capazes de atingir um desempenho muito bom em comparação com os computadores de uso geral. Outras tarefas foram mapeadas para hardware reconfigurável pois este abre oportunidades inovadoras a explorar. Todas as aplicações potenciais podem ser divididas em três categorias de acordo com o objectivo principal que se pretende atingir. A seguir, descrevemos brevemente estas categorias ilustrando cada área de aplicação com exemplos.

A. Aceleração de tarefas computacionalmente intensivas

Todas as implementações pertencentes a esta categoria têm como objectivo principal acelerar tarefas computacionalmente intensivas. Uma característica comum é que estas aplicações servem bem para implementações paralelas aproveitando das capacidades básicas da computação reconfigurável.

Processamento de sinal e imagem é um domínio de aplicações bastante popular na área de computação reconfigurável. Este domínio caracteriza-se pelo elevado grau de paralelismo inerente, grande quantidade dos dados a processar e é adequado para *pipelining* [49]. Estes factores permitem atingir um bom desempenho num sistema reconfigurável comparando com implementações num computador de uso geral. Por exemplo, PAM foi

utilizado para implementar um algoritmo de visão estéreo conseguindo-se uma aceleração de 30 vezes em comparação com uma realização em hardware dedicado baseado em 4 DSPs (*Digital Signal Processors*) [29]. Splash-2 também foi utilizado para processamento de imagens. Os dados de vídeo entram numa extremidade do *array* linear e eram processados ao passar por cada elemento de processamento. Descrições de outros sistemas reconfiguráveis destinados a processar imagens e sinais encontram-se na literatura [35, 37, 50-53].

Criptografia e aritmética de inteiros longos também são áreas bastante populares na computação reconfigurável. Isto explica-se pelo facto de as operações sobre inteiros longos não serem directamente suportadas pelos computadores convencionais. Assim, PAM foi utilizado para calcular o resultado da operação $A \times B + C$, onde A pode ter até 2Kbits, e B e C são parâmetros de tamanhos arbitrários [29]. O sistema produziu resultados à taxa de 66Mb/s. A criptografia de RSA, por sua vez, envolve operações que podem ser decompostas numa sequência de multiplicações de inteiros longos. Por exemplo, PAM para as chaves de 512 bits atingiu a velocidade de descodificação de 300Kb/s [29].

Foram propostas várias arquitecturas de sistemas reconfiguráveis na área de comunicações. Estes sistemas são otimizados para tarefas tais como implementação de protocolos de acesso, armazenamento e envio de dados, controlo de tráfego e suporte para a qualidade de serviço [54-56].

Tradicionalmente, a maioria de sistemas implementados em hardware reconfigurável realizavam computações regulares sobre grandes volumes de dados. Recentemente, foram efectuadas muitas tentativas de acelerar aplicações que envolvem algoritmos de controlo bastante complexos. Uma atenção especial neste contexto foi dada aos problemas na área de optimização combinatória [57]. Muitos dos problemas que surgem em optimização combinatória são de elevada complexidade, não sendo resolúveis em tempo polinomial. Prestam-se no entanto à exploração de algoritmos de solução com um considerável grau de paralelismo. Deste modo torna-se viável a construção de coprocessadores, ou aceleradores, reconfiguráveis para problemas combinatórios. Este potencial foi intensivamente explorado o que resultou no desenvolvimento de uma série de arquitecturas quer orientadas para um problema combinatório específico quer destinadas a resolver um conjunto de problemas da área de optimização combinatória. As acelerações atingidas com estes sistemas reconfiguráveis em comparação com as implementações em software vão até às ordens de grandeza.

B. Emulação de hardware

É bastante difícil saber se algum circuito projectado é correcto e estimar o seu desempenho antes da sua implementação física. Tradicionalmente utilizam-se duas abordagens para a verificação de projectos complexos: prototipagem e simulação em software. A construção de

um protótipo é uma tarefa dispendiosa e morosa. As técnicas de simulação são muito flexíveis, mas muito lentas.

É de salientar que as FPGAs servem muito bem para emular dispositivos de hardware, pois são plataformas extremamente flexíveis que em geral não exigem conhecimento prévio sobre as características das aplicações que podem ser mapeadas [58, 59]. A emulação é muito mais rápida que a simulação possibilitando deste modo a verificação de um projecto com grandes quantidades de dados reais. Para além disso, a emulação reflecte as características da implementação final mais fielmente do que a simulação, e portanto resulta em avaliação do desempenho e verificação do projecto mais correctas.

FPGAs recentes tais como Virtex-II da Xilinx e Stratix da Altera incorporam grandes quantidades de lógica, unidades aritméticas, blocos de memória embutidos, o que as torna uma plataforma muito adequada para a emulação. Uma revisão de plataformas de emulação e de prototipagem baseadas em FPGAs disponíveis comercialmente e desenvolvidas no âmbito de investigação académica, é feita em [60].

C. Hardware evolutivo

Uma área de investigação bastante recente é o *hardware evolutivo*. Este termo descreve abordagens diferentes utilizadas para desenvolver circuitos electrónicos com a ajuda de técnicas evolutivas o que permite uma exploração mais alargada do espaço de projecto, conforme representado na fig. 16 [61, 62].

Normalmente estas abordagens dividem-se em dois grupos de acordo com a área de aplicação. O primeiro grupo abrange a evolução directa em componentes existentes (geralmente, em FPGAs), enquanto o segundo grupo inclui as tentativas de simular a funcionalidade desejada a fim de a implementar posteriormente em componentes reais.

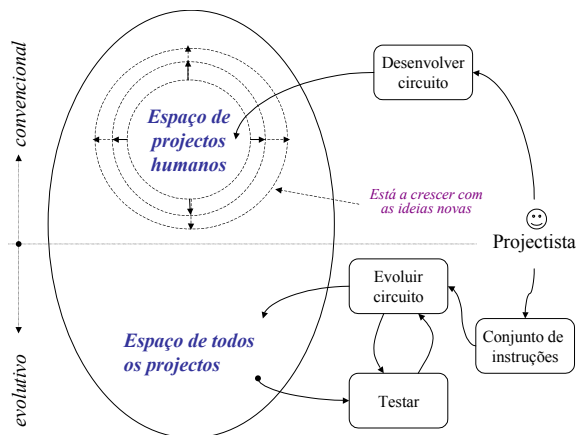


Fig. 16 – Projecto convencional vs. projecto evolutivo

As potencialidades das técnicas de hardware evolutivo são bastante extensas pois teoricamente possibilitam a

construção autónoma (i.e. quase sem intervenção dos peritos humanos) dos circuitos com as características necessárias cuja estrutura é previamente desconhecida. Isto explica-se pelo facto de os circuitos evolutivos possuírem capacidades de auto-optimização. Por exemplo, Miller *et al.* evoluíram um multiplicador de 3 bits 20% mais eficiente (em termos do número de portas lógicas utilizadas) do que qualquer projecto convencional conhecido [63].

Uma outra aplicação possível destas técnicas é a auto-reparação de circuitos autónomos cuja reparação por peritos humanos é por algum motivo impossível (por exemplo, nos sistemas utilizados no espaço) [64]. Tudo isto torna o hardware evolutivo uma área de investigação bastante promissora [65-67]. É de notar todavia que ainda não foi construído nenhum sistema suficientemente complexo, capaz de efectuar auto-optimização rápida e eficiente. Algumas razões que explicam este facto podem ser encontradas em [68].

VII. CONCLUSÕES

A lógica programável é um dos segmentos de desenvolvimento mais intensivo de todo o mercado de semicondutores [69]. As diferenças de custos entre a produção de PLDs e ASICs em volumes grandes estão a diminuir [70]. Prevê-se que no futuro todos os sistemas, à excepção daqueles que exigem desempenho ou volumes de produção extremamente elevados, serão baseados em PLDs [70].

Neste contexto a computação reconfigurável tornou-se uma área de investigação importante. Ao atribuir porções computacionalmente intensivas de uma aplicação ao hardware reconfigurável, pode-se acelerar significativamente a execução desta aplicação. Isto deve-se ao facto de sistemas reconfiguráveis combinarem vantagens das implementações baseadas em ASIC e das baseadas em software. Os circuitos resultantes são flexíveis podendo ser modificados em qualquer momento mesmo durante a execução da aplicação. Para além disso, estes circuitos são capazes de atingir melhor desempenho que o de software ao eliminar o *overhead* inerente a qualquer processador de uso geral.

Para atingir melhores resultados, os sistemas reconfiguráveis são normalmente compostos pela lógica reconfigurável interligada com um processador de uso geral. O processador executa operações que não podem ser realizadas eficientemente em lógica reconfigurável, enquanto as partes computacionalmente intensivas da aplicação são mapeadas em hardware. Em sistemas reconfiguráveis o nível de interligação dos componentes principais pode variar desde unidades funcionais configuráveis integradas no processador até FPGAs autónomas.

Configurações diferentes podem ser utilizadas em várias fases de execução de uma aplicação personalizando o hardware a cada uma das fases. A reconfiguração em *run-time* permite implementar configurações maiores que os recursos de hardware disponíveis, dividindo os circuitos

respectivos em circuitos mais pequenos que são utilizados sucessivamente. Devido aos atrasos associados à configuração, a reconfiguração deve ser realizada de uma maneira muito eficiente.

A implementação de uma aplicação num sistema reconfigurável não é uma tarefa simples. Esta envolve a programação a nível de software e o projecto da parte de hardware. A fim de atingir bons resultados no mapeamento de uma aplicação na FPGA é necessário explorar o paralelismo e operações específicas e optimizadas. Para além disso, a compilação de configurações de hardware deve ser efectuada rapidamente.

Em [69] afirma-se que um dos grandes sucessos da indústria de software foi o facto de sistemas computacionais tradicionais serem baseados em RAM. O código é carregado na RAM para ser executado, podendo ser modificado em qualquer altura, resultando assim numa extrema flexibilidade. As FPGAs reconfiguráveis no sistema são também baseadas em RAM. Neste caso o código estrutural (o código de configuração) é carregado na RAM da FPGA. Contudo, as arquitecturas de FPGAs correntes não são escaláveis ao contrário do que acontece nos computadores convencionais [69]. A consequência disso é que é necessário recompilar e redepurar todos os circuitos para cada novo tipo de FPGA.

É de notar também que embora já tenham sido propostas muitas arquitecturas eficientes que revelaram vantagens significativas da computação reconfigurável, ainda há muitas questões por responder. Em particular, os sistemas reconfiguráveis introduziram uma nova dimensão a nível de programação. Portanto, são necessárias tecnologias e ferramentas inovadoras que sejam capazes de cobrir todos os aspectos da programação e do uso eficiente de sistemas reconfiguráveis.

REFERÊNCIAS

- [1] R. Hartenstein, "A decade of Reconfigurable Computer: A Visionary Retrospective", Proc. Design, Automation and Test in Europe Conf. (DATE), pp. 642-649, 2001.
- [2] Xilinx. [Online]: <http://www.xilinx.com/>.
- [3] HandelC. [Online]: <http://www.celoxica.com/>.
- [4] SystemC. [Online]: <http://www.systemc.org/>.
- [5] CoCentric. [Online]: http://www.synopsys.com/products/cocentric_systemC/cocentric_systemC_ds.pdf.
- [6] Tutorials. [Online]: <http://webct.ua.pt>, "2º semestre", disciplina "Computação Reconfigurável".
- [7] V. Sklyarov, I. Skliarova, "Ferramentas para desenvolvimento de sistemas digitais reconfiguráveis", *Electrónica e Telecomunicações*, v. 3, Nº 8, Janeiro 2003, pp. 743-764.
- [8] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing Surveys*, vol. 34, nº 2, pp. 171-210, Jun. 2002.
- [9] J. Babb, R. Tessier, A. Agarwal, "Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators", *Proceedings IEEE Workshop on FPGA-based Custom Computing Machines*, pp. 142-151, Apr. 1993.
- [10] B. Przybus, "New ChipScope Pro Integrated Bus Analyzer", *Xcell Journal*, Winter 2002, n. 44, pp. 24-25.
- [11] A.M. Hernandez, "Deep Memory Yields Effective In-System Debugging", *Xcell Journal*, Winter 2002, n. 44, pp. 26-28.
- [12] A.K. Sharma, "Programmable Logic Handbook. PLDs, CPLDs, and FPGAs", McGraw-Hill, 1998.
- [13] Actel. [Online]: <http://www.actel.com/products/devices.html>.
- [14] Altera. [Online]: <http://www.altera.com/products/devices/dev-index.jsp>.
- [15] Atmel. [Online]: <http://www.atmel.com/products/FPGA/>.
- [16] Lattice. [Online]: <http://www.vantis.com/products/index.cfm>.
- [17] QuickLogic. [Online]: <http://www.quicklogic.com/>.
- [18] G. Estrin, "Organization of Computer Systems—The Fixed Plus Variable Structure Computer," *Proc. Western Joint Computer Conf.*, New York, 1960, pp. 33-40.
- [19] G. Estrin, "Reconfigurable Computer Origins: The UCLA Fixed-Plus-Variable (F+V) Structure Computer", *IEEE Annals of the History of Computing*, Oct.-Dec., 2002, pp. 3-9.
- [20] S.A. Guccione, "Reconfigurable Computing at Xilinx", apresentado em *Euromicro Symposium on Digital System Design – DSD'2001*, Varsóvia, Polónia, Setembro, 2001.
- [21] K. Compton, S. Hauck, "An Introduction to Reconfigurable Computing", *IEEE Computer*, Apr. 2000.
- [22] E. Sanchez, M. Sipper, J.O. Haenni, J.L. Beuchat, A. Stauffer, A. Perez-Urbe, "Static and Dynamic Configurable Systems", *IEEE transactions on Computers*, vol. 48, No. 6, June 1999, pp. 556-564.
- [23] I. Skliarova, A.B. Ferrari, "Design and Implementation of Reconfigurable Processor for Problems of Combinatorial Computations", *Proceedings of the Euromicro Symposium on Digital System Design – DSD'2001*, Varsóvia, Polónia, Setembro, 2001, pp. 112-119.
- [24] P. Schaumont, I. Verbauwhede, K. Keutzer, M. Sarrafzadeh, "A quick safari through the reconfiguration jungle", *Proc. 38th Design Automation Conf. (DAC'01)*, pp. 172-177, 2001.
- [25] B.L. Hutching, M.J. Wirthlin, "Implementation Approaches for Reconfigurable Logic Applications", *Proceedings of FPL, Lecture Notes in Computer Science*, vol. 975, pp. 419-428, Springer, 1995.
- [26] I. Skliarova, A.B. Ferrari, "The Design and Implementation of a Reconfigurable Processor for Problems of Combinatorial Computation", a aparecer em *Journal of Systems Architecture, Special Issue on Reconfigurable Systems*, 2003.
- [27] J.A. Jacob, "Memory Interfacing for the OneChip Reconfigurable Processor", *Dissertação de mestrado*, Univ. de Toronto, 1998.
- [28] B. Radunovic, V. Milutinovic, "A Survey of Reconfigurable Computing Architectures", *Proceedings of FPL, Lecture Notes in Computer Science*, vol. 1482, pp. 376-385, Springer, 1998.
- [29] J.E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H.H. Touati, P. Boucard, "Programmable Active Memories: Reconfigurable Systems Come of Age", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, nº 1, pp. 56-69, Mar. 1996.
- [30] D. A. Buell, J. M. Arnold, W. J. Kleinfelder, "Splash 2: FPGAs in a Custom Computing Machine", *IEEE Computer Society Press*, Los Alamitos, CA, 1996.
- [31] P.M. Athanas, H.F. Silverman, "Processor Reconfiguration through Instruction Set Metamorphosis", *Computer*, vol. 26, nº 3, pp. 11-18, Mar. 1993.
- [32] R. Razdan, M.D. Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units", *Proceedings of the 27th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 172-180, Nov. 1994.
- [33] V. Gustin, "An FPGA extension to ALU functions", *Microprocessors and Microsystems*, 22, pp. 501-508, 1999.
- [34] S. Hauck, T.W. Fry, M.M. Hosler, J.P. Kao, "The Chimaera Reconfigurable Functional Unit", *IEEE Symp. On Field-Programmable Custom Computing Machines*, pp. 87-96, 1997.

- [35] T.J. Callahan, J.R. Hauser, J. Wawrzynek, "The Garp Architecture and C Compiler", *Computer*, pp. 62-69, Apr. 2000.
- [36] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, A. Agarwal. "Baring it all to Software: RAW machines", *IEEE Computer*, vol. 30, n. 9, Sept. 1997, pp. 86-93.
- [37] H. Singh, M.H. Lee, G. Lu, F.J. Kurdahi, N. Bagherzadeh, E.M.C. Filho, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications", *IEEE Trans. on Computers*, vol. 49, n° 5, pp. 465-481, May 2000.
- [38] M. Havener, N. Hartl, "500+ Xilinx FPGAs Search for Elusive Higgs Boson at 1.5 Terabytes per Second", *Xcell Journal*, Winter 2002, n. 44, pp. 68-73.
- [39] J. de Sousa, J.P. Marques-Silva, M. Abramovici, "A configure/software approach to SAT solving", 9th IEEE Proc. Int. Symp. on Field-Programmable Custom Computing Machines, FCCM, Maio 2001.
- [40] A. DeHon, "The Density Advantage of Configurable Computing", *Computer*, vol. 33, n° 4, pp. 41- 49, Apr. 2000.
- [41] D. Abramson, P. Logothetis, A. Postula, M. Randall, "FPGA Based Custom Computing Machines for Irregular Problems", Proc. of the 4th Int. Symp on High-Performance Computer Architecture – HPCA'98, Fevereiro de 1998, Las Vegas, Nevada.
- [42] RC Bibliography. [Online]: <http://www.ife.ee.ethz.ch/~enzler/rc/bib.html>.
- [43] M.J. Wirthlin, B.L. Hutching, K.L. Gilson, "The Nano Processor: a Low Resource Reconfigurable Processor", *Proceedings IEEE Symp. on FPGAs for Custom Computing Machines*, pp. 23-30, Apr. 1994.
- [44] I. Page, "Reconfigurable Processor Architectures", "Microprocessors and Micrisystems", vol. 20, n° 3, pp. 185-196, 1996.
- [45] E. Mirsky, A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources", *Proceedings of FCCM'96*, pp. 157-166, Apr. 1996.
- [46] C.R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J.M. Arnold, M. Gokhale, "The NAPA Adaptive Processing Architecture", *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 28-37, Apr. 1998.
- [47] M. B. Gokhale, J. M. Stone, "NAPA C: Compiling for a Hybrid RISC/FPGA Architecture", *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 126-135, Apr. 1998.
- [48] R. Hartenstein, "Reconfigurable Computing: A New Business Model – and its Impact on SoC Design", *Proceedings of the Euromicro Symposium on Digital System Design – DSD'2001*, Varsóvia, Polónia, Setembro, 2001, pp. 103-110.
- [49] W.H. Mangione-Smith, B.L. Hutchings, "Configurable Computing: The Road Ahead", *Proc. Reconfigurable Architectures Workshop - RAW*, pp. 81-96, 1997.
- [50] A. Bouridane, D. Crookes, P. Donachy, K. Alotaibi, K. Benkrid, "A high level FPGA-based abstract machine for image processing", *Journal of Systems Architecture*, 45, pp. 809-824, 1999.
- [51] C. Dick, F. Harris, "Virtual signal processors", *Microprocessors and Microsystems*, 22, pp. 135-148, 1999.
- [52] S.D. Haynes, J. Stone, W. Luk, "Video Image Processing with the Sonic Architecture", *Computer*, pp. 50-57, Apr. 2000.
- [53] R. Tessier, W. Bursleson, "Reconfigurable Computing for Digital Signal Processing: A Survey", *Journal of VLSI Signal Processing*, vol. 28, n° 1/2, pp. 7-27, May 2001.
- [54] M. Iliopoulos, T. Antonakopoulos, "Reconfigurable Network Processors Based on Field Programmable System Level Integrated Circuits", *Proceedings of FPL, LN in Computer Science*, vol. 1896, pp. 39-47, Springer, 2000.
- [55] X. Tang, M. Aalsma, R. Jou, "A Compiler Directed Approach to Hiding Configuration Latency in Chameleon Processors", *Proceedings of FPL, LN in Computer Science*, vol. 1896, pp. 27-38, Springer, 2000.
- [56] J.M. Rabaey, "Silicon Platforms for the Next Generation Wireless Systems – What Role Does Reconfigurable Hardware Play?", *Proceedings of FPL, LN in Computer Science*, vol. 1896, pp. 277-285, Springer, 2000.
- [57] I.Skljarova, A.B.Ferrari, "Reconfigurable Hardware SAT Solvers: A Survey of Systems", *Proc. Int. Conference on Field-Programmable Logic – FPL'2003*, Lisboa, Portugal, Setembro, 2003.
- [58] M. Dubois, J. Jeong, Y.H. Song, A. Moga, "Rapid hardware prototyping on RPM-2", *Design & Test of Computers*, IEEE, vol. 15, n° 3, pp. 112 –118, Jul.-Sep., 1998.
- [59] M. Gschwind, V. Salapura, D. Maurer, "FPGA prototyping of a RISC processor core for embedded applications", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 9, n° 2, pp. 241-250, Apr. 2001.
- [60] H. Krupnova, G. Saucier, "FPGA-Based Emulation: Industrial and Custom Prototyping Solutions", *Proceedings of FPL, Lecture Notes in Computer Science*, vol. 1896, pp. 68-77, Springer, 2000.
- [61] J. F. Miller, P. Thomson, T. Fogarty, "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study", "Genetic Algorithms and Evolution Strategies in Engineering and Computer Science", D. Quagliarella, J. Périaux, C. Poloni, G. Winter, eds., pp. 105-131, 1998.
- [62] A. Thompson, P. Layzell, R.S. Zebulum, "Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution", *IEEE Trans. on Evolutionary Computation*, vol. 3, no. 3, Set. 1999, pp. 167 – 196.
- [63] J.F. Miller, D. Job, V.K. Vassilev, "Principles in the Evolutionary Design of Digital Circuits – Part I". *Genetic Programming and Evolvable Machines 1*, pp. 7-35, 2000.
- [64] S. Vigander, "Evolutionary Fault Repair of Electronics in Space Applications", *Universidade de Sussex*, [Online]: http://www.cogs.susx.ac.uk/users/adrianth/SVERRE_VIGANDER/sverre.html, 2001.
- [65] E. Sanchez, M. Tomassini (ed.), "Towards Evolvable Hardware. The Evolutionary Engineering Approach", *Lecture Notes in Computer Science*, vol. 1062, 1996.
- [66] M.Sipper, D.Mange, A.Pérez-Urbe (ed.), "Evolvable Systems: From Biology to Hardware", *Lecture Notes in Computer Science*, vol. 1478, 1998.
- [67] J.R. Koza, F.H. Bennet III, D. Andre, M.A. Keane, "Genetic Programming III", Morgan Kaufmann Publishers, 1999.
- [68] J. Torresen, "Possibilities and Limitations of Applying Evolvable Hardware to Real-World Applications", *Proceedings of FPL, LN in Computer Science*, vol. 1896, pp. 230-239, Springer, 2000.
- [69] R. Hartenstein, "Configure/Software Co-Design: be prepared for the next revolution!", *keynote talk*, *Proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop – DDECS*, Brno, Rep. Checa, Abril, 2002, pp. 19-34.
- [70] N. Tredennick, B. Shimamoto, "The Rise of Reconfigurable Systems", *keynote talk*, *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms – ERS'A'2003*, Las Vegas, EUA, Junho, 2003, pp. 3-9.