Uma Introdução aos Processadores de Arquitectura Configurável

Orlando Moreira, Nuno Lau, António Ferrari

***Resumo* - Processadores de Arquitectura Configurável são sistemas de computação em que lógica programável é integrada na arquitectura de microprocessadores tradicionais, tentando combinar a versatilidade das implementações em *software* com o elevado desempenho das implementações em *hardware*. Este artigo dá uma visão geral deste tipo de sistemas, discutindo factores de implementação e mostrando exemplos de arquitecturas propostas.**

***Abstract* – Configurable Architecture Processors are computational systems where programmable logic is inserted within the architecture of a traditional microprocessor core in order to combine the versatility of software and the high performance of hardware. This paper gives an overview of this kind of systems, discussing implementation factors and showing examples of proposed architectures.**

I. INTRODUÇÃO

A. Dispositivos Lógicos Programáveis

Um Dispositivo Lógico Programável (DLP) pode ser definido como um circuito integrado passível de ser configurado pelo utilizador final para implementar diferentes circuitos digitais. Estes dispositivos são constituídos por lógica configurável e, na maior parte das vezes, *flip-flops* agregados por conexões programáveis. A informação relativa à configuração é frequentemente guardada em memórias incluídas no DLP, o que permite a alteração da configuração.

Existe uma grande variedade de arquitecturas possíveis para um DLP, bem como vários sistemas de classificação (ver a este respeito [1], [2]). De uma maneira geral, são reconhecidas diferenças fundamentais entre 4 tipos de dispositivos:

- SPLD (*Simple Programmable Logic Device*)- este nome genérico é dado a arquitecturas de lógica programável do tipo PAL e PLA. As PALs e PLAs implementam funções lógicas por meio de soma de termos-produto em módulo-2. Um problema das SPLDs é que não podem albergar uma grande quantidade de lógica, uma vez que o aumento desta acarreta um aumento dos atrasos de propagação no dispositivo. O primeiro fabricante a disponibilizar PLAs, nos anos 70, foi a Philips.
- CPLD (*Complex Programmable Logic Device*)- resulta da evolução das arquitecturas SPLD e foi desenvolvido de forma a permitir dispositivos com maior capacidade lógica. Um CPLD é constituído por uma matriz de encaminhamento (*switch-matrix*) programável que conecta blocos do tipo SPLD entre si e com as Entradas e Saídas. O número de blocos SPLD varia tipicamente entre 4 e 32. Esta estrutura é apresentada na Fig. 1. Cada uma das saídas dos blocos é terminada por um ou mais *flip-flops*, de forma a facilitar a implementação de lógica sequencial. Estas estruturas são apelidadas de macrocélulas. As macrocélulas devem ter conexões de retroalimentação de forma a permitir a implementação de lógica multinível. Actualmente, existem no mercado CPLDs com uma gama de capacidades atingindo as 960 macrocélulas (Xilinx XC3960 XPLA-II). Uma lista descritiva de alguns dispositivos deste género pode ser encontrada em [1] ou [3].

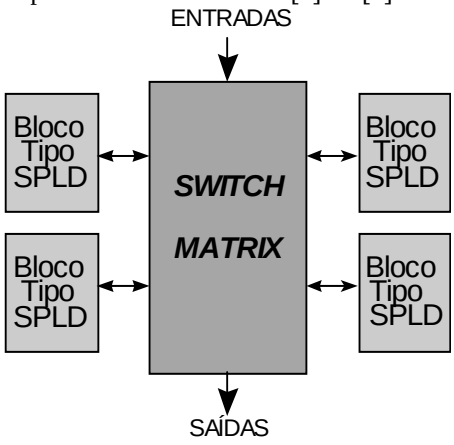

Fig. 1 A arquitectura de um CPLD.

- FPGA (*Field-Programmable Gate Array*)- estes circuitos oferecem normalmente maior capacidade lógica que os CPLDs, são mais flexíveis e implementam mais facilmente lógica multinível. São compostos de uma matriz de blocos funcionais de função programável e blocos E/S, com conexões programáveis entre si (Fig. 2). Uma FPGA típica contém entre 64 e vários milhares de blocos lógicos e um número ainda maior de *flip-flops* (cada bloco funcional costuma incluir um ou mais *flip-flops*). Visto que o número de conexões

entre blocos funcionais é limitado, as FPGAs dependem do *software* de desenvolvimento para instalar e encaminhar a lógica no dispositivo. É entre os DLPs normalmente classificados como FPGAs que se encontra a maior variedade de opções arquitecturais. Há duas características principais que variam de dispositivo para dispositivo: a quantidade de lógica implementada por cada bloco funcional, e a organização das interconexões programáveis. Entre os grandes fabricantes de FPGAs distinguem-se a Xilinx (que aliás foi a primeira a disponibilizar este tipo de circuitos) e a Altera. Em [1] são descritas várias arquitecturas de FPGAs existentes no mercado.

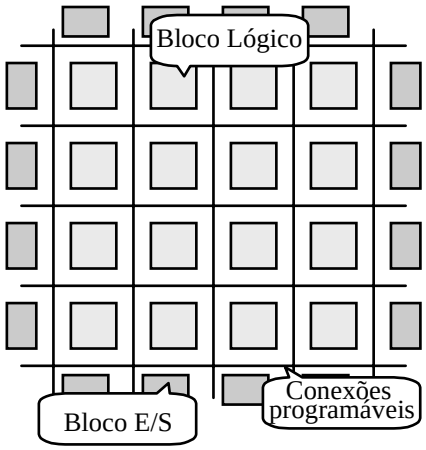

Fig. 2 A arquitectura de uma FPGA.

- FPIC (*Field-Programmable InterConnect*)- é um dispositivo que, através de programação, conecta um dos seus pinos a outro, permitindo assim interconexões programáveis num sistema digital. Embora não sejam de facto DLP (apenas disponibilizam ligações, não têm lógica incorporada), são aqui referidos visto permitirem uma pequena capacidade de configuração a um sistema.

B. Projecto de circuitos digitais utilizando DLPs

O processo de projecto de circuitos digitais utilizando DLPs é geralmente assistido por um conjunto de ferramentas *software* e executa-se pelos seguintes passos:

Especificação funcional - o circuito é descrito pelo projectista através de uma linguagem de descrição de *hardware* (*eg:* VHDL, ABEL) ou captura esquemática. Esta descrição é traduzida pelas ferramentas de projecto numa descrição funcional do circuito, baseada em primitivas lógicas simples.

Simulação funcional - verificação da correcção lógica da especificação. Não toma em conta atrasos de propagação visto que estes dependem da implementação.

Partição lógica - o circuito é dividido em funções lógicas directamente implementáveis no dispositivo alvo.

*Posicionamento (*placement*)* - a cada bloco resultante da partição é atribuído um bloco funcional específico do DLP.

Encaminhamento (routing) - as conexões necessárias entre os blocos funcionais são estabelecidas, por programação das interconexões.

Simulação temporal e/ou testes finais sobre o hardware - o circuito é verificado, tendo agora em conta os tempos de propagação estabelecidos pela implementação.

C. Aplicações dos DLPs

Um processador tradicional é virtualmente capaz de implementar qualquer tarefa computacional possível. Isto é feito por decomposição dessa tarefa numa sequência específica de instruções do conjunto de instruções fixo que o processador disponibiliza. Este conjunto de instruções é, tipicamente, bastante genérico, incluindo instruções aritméticas e lógicas para palavras de comprimento fixo, controlo de fluxo da execução e acessos à memória. Embora esta configuração permita ao processador uma grande flexibilidade, limita o seu desempenho em termos de tempo de execução; frequentemente, uma implementação em *hardware* especificamente construído para efectuar uma determinada tarefa computacional (um ASIC -*Application Specific Integrated Circuit*) permite obter desempenhos muito melhores. O circuito obtido, contudo, apenas é útil para levar a cabo a tarefa em causa; após fabrico, a sua funcionalidade é imutável.

Embora com desempenhos inferiores a ASICs [1], os DLPs podem melhorar, em relação aos processadores tradicionais, o desempenho de certas aplicações visto que o comportamento do *hardware* pode ser adaptado para se ajustar às características individuais de uma determinada aplicação e re-utilizado para uma aplicação diferente após ser reprogramado.

Os DLPs surgem como uma solução intermédia entre um processador genérico e um ASIC, em termos de versatilidade/desempenho. Acrescente-se a isso que para casos de baixa produção uma implementação ASIC fica mais cara e demora mais tempo a produzir.

Tendo em conta estas características, os DLPs são interessantes para diversas áreas de aplicação:

- aplicações cuja baixa produção não justifica os custos de fabricação de um ASIC;
- aplicações com necessidade de adaptação "crónica" a novos padrões emergentes e em que as exigências de desempenho em tempo real não sejam cumpridas (a custos equivalentes) por processadores genéricos (eg criptografia, protocolos de rede);

- produtos que necessitem de chegar rapidamente ao mercado;
- prototipagem de circuitos digitais.

Mais, como veremos a seguir, os DLPs estão na base de um novo tipo de arquitectura de computadores. Este tema é discutido em [4].

D. Processadores de Arquitectura Configurável

Os sistemas completamente constituídos por PLDs são, regra geral, pouco eficientes na implementação da *totalidade* de aplicações computacionalmente complexas, que são normalmente atribuídas a processadores tradicionais. É sabido [5] que, tipicamente, uma aplicação deste género ocupa cerca de 90% do seu tempo de processamento em apenas 10% do código. Embora o impacto sobre o tempo de execução desses segmentos críticos justifique a implementação em *hardware*, outras partes do código há que raramente são executadas, e cujo mapeamento no DLP representa um desperdício de recursos[1]. Além disso, um processador genérico dispõe de unidades funcionais especializadas em computações comuns tais como aritmética inteira e lógica sobre palavras de comprimento fixo e aritmética de vírgula flutuante, entre outras. Estes circuitos dedicados garantem melhor desempenho que os DLP, nas operações que implementam directamente.

Tendo em conta estes factores, pode-se imaginar uma abordagem mista: um computador com uma estrutura dividida entre uma parte fixa (o processador tradicional) e outra parte adaptável, que se destinaria a gerar novos circuitos consoante as necessidades específicas de cada aplicação. Esta ideia tinha sido proposta em 1960 por Gerald Estrin [6], mas nessa altura ainda não existia a tecnologia necessária para implementar a parte adaptável da estrutura, e a ideia foi posta de lado. Com o surgimento dos DLPs na década de 80 o conceito foi retomado. Um *Processador de Arquitectura Configurável* (PAC) é a aplicação directa da ideia original de Estrin, tentando conciliar o melhor dos dois mundos: o *core* de um processador tradicional é acoplado a uma certa quantidade de recursos DLP. Desta forma, o processador fica responsável por operações em que é tipicamente eficiente, e pelo controlo do fluxo da execução, enquanto que partes do programa que possam beneficiar de forma significativa de uma implementação em *hardware* são adjudicadas aos recursos configuráveis. Embora esta ideia básica seja relativamente simples, a sua implementação é complexa. Entre os vários problemas implicados convém porventura realçar os seguintes: a integração do DLP no sistema de processamento e a técnica de compilação. Nenhum produto utilizando esta arquitectura atingiu ainda o mercado, mas muita investigação tem sido feita na área recentemente, e vários resultados interessantes foram dados a conhecer à comunidade científica.

[1] Uma forma de amenizar esse problema é recorrer à reconfiguração dinâmica (durante a execução) do dispositivo, mas tal processo é tipicamente lento, e implica, por vezes um custo demasiado grande em termos de desempenho.

II. FACTORES DE IMPLEMENTAÇÃO

Vários factores devem ser levados em conta na construção de um PAC. Nesta secção tentámos abordar os mais importantes.

A. Arquitectura do DLP

As duas arquitecturas de DLPs tipicamente utilizadas em PACs são as arquitecturas CPLD e FPGA, visto não terem as outras capacidade lógica suficiente para este tipo de aplicações. Uma análise comparativa dos dois tipos de dispositivos permite-nos tirar algumas conclusões:

- As FPGAs são mais adequadas para a implementação de lógica multinível, permitem maior versatilidade e a implementação de circuitos mais complexos.
- Estruturas mais simples e fortemente hierarquizadas como as dos CPLDs permitem modelos de temporização mais simples e determinísticos. Pela mesma razão, o desenvolvimento de *software* de projecto para CPLDs é menos complexo que para FPGAs e é frequentemente mais eficiente no aproveitamento dos recursos disponíveis.

Embora muitos PACs utilizem DLPs comerciais, outros empregam dispositivos desenhados especificamente para o sistema em causa. Entre as várias decisões de projecto a tomar pode-se salientar:

- a *granulosidade do dispositivo*, diz respeito à resolução da configurabilidade do dispositivo; se esta está ao nível dos *bits*, então o dispositivo diz-se *fine-grained*, se a configurabilidade só está disponível ao nível de palavras (conjuntos de *bits*), o dispositivo diz-se *coarse-grained*;
- em alguns casos, a *implementação de lógica puramente combinatória* é suficiente para o PAC pretendido; noutros casos, os registos são necessários para implementar circuitos sequenciais;

B. Integração dos recursos configuráveis

O acoplamento dos recursos configuráveis ao processador é um factor essencial na construção de um PAC. Duas aproximações são habitualmente usadas:

a) *Integração paralela (*loosely-coupled*)*- os recursos configuráveis são utilizados pelo

dispositivo na qualidade de coprocessador. A conexão com o processador hospedeiro é tipicamente feita através de um barramento e o processamento pode por vezes ser executado em concorrência, de forma assíncrona. Largos segmentos da aplicação devem ser mapeados no(s) coprocessador(es) de forma a rentabilizar o uso dos recursos configuráveis, tendo em conta a latência provocada pela comunicação com o processador, pelo que uma grande quantidade de recursos configuráveis deve estar disponível.

b) *Integração série* (tightly-coupled)- os recursos configuráveis são inseridos no *datapath* do processador. Isto pode ser feito utilizando um ou mais DLPs para implementar Unidades Funcionais Configuráveis (UFCs), de utilização análoga a outras unidades funcionais (eg a ALU). As UFCs são consideravelmente mais pequenas do que um coprocessador configurável (representado custos mais baixos). Este tipo de integração elimina os problemas de sincronização, mas exige em contrapartida que os tempos de propagação no DLP sejam simples de determinar e compatíveis com a *pipeline* do processador utilizado.

C. Política de carregamento de configurações

A configuração do(s) DLP(s) pode estar a cargo do processador principal ou de uma unidade de controlo de configuração independente. De qualquer maneira, o tempo de configuração é responsável por um decaimento considerável do desempenho do sistema, tornando-se importante descobrir formas de minorar o problema. Duas políticas diferentes de carregamento de configurações num DLP podem ser utilizadas:

a) *Pré-configuração (configuração estática)*- o processo de configuração é resolvido de forma estática: antes de uma aplicação começar a ser executada, a configuração correspondente do DLP é carregada uma única vez, mantendo-se constante durante toda a execução da aplicação. Note-se que cada aplicação apenas pode utilizar uma única configuração para cada DLP existente no sistema.

b) *Configuração dinâmica (run-time configuration)*- o DLP é configurado múltiplas vezes durante a execução de acordo com as necessidades da aplicação, sendo que cada aplicação pode utilizar várias configurações diferentes. A forma mais simples (e menos eficiente) de implementar este método é proceder à reconfiguração global do DLP. Assim, a execução é fraccionada em duas fases temporais distintas que se repetem ciclicamente: uma fase de reconfiguração e uma fase de processamento. Obviamente, o tempo de reconfiguração vai degradar significativamente o desempenho do DLP, e, em arquitecturas com integração série, implica frequentemente a paralisação da *pipeline* do processador. Alguns DLPs permitem que se proceda à sua *reconfiguração parcial dinâmica*, onde enquanto uma parte dos recursos do dispositivo está a ser utilizada pela aplicação, outra parte dos recursos está a ser reconfigurada para ser posteriormente usada. Este esquema de reconfiguração em paralelo pode também ser implementado em arquitecturas que usem *múltiplas unidades reconfiguráveis*. Técnicas semelhantes às utilizadas para fazer o carregamento da *cache* podem ser utilizadas para controlar o processo de reconfiguração. Neste caso, a gestão das reconfigurações pode ser feita em paralelo com a execução do programa.

D. Técnicas de compilação

O problema da compilação está intimamente ligado com a forma de integração dos recursos reconfiguráveis e com a política de carregamento de configurações de um determinado dispositivo. Independentemente disso, um compilador desenhado para um PAC deve englobar as seguintes funcionalidades:

a) Detecção de segmentos de código adequados para implementação nos recursos configuráveis (blocos candidatos). A natureza destes blocos candidatos está dependente da forma de integração dos recursos configuráveis: para integração paralela, um bloco candidato é tipicamente um procedimento que possa ser executado em paralelo com o resto do programa; para integração série, um bloco candidato equivale normalmente a uma sequência de algumas linhas de código *assembly*.

b) Selecção, entre os blocos candidatos, daqueles que devem ser implementados em *hardware*, tendo em conta a sua importância relativa no tempo de processamento da aplicação. Para levar a cabo este passo é necessário analisar o comportamento em tempo de execução da aplicação, pelo que é comum utilizarem-se ferramentas de *profiling*. Este é porventura o passo cuja automatização é mais complexa.

c) Produção de informação de configuração do DLP, relativa aos blocos seleccionados. Este processo é frequentemente efectuado por uma ferramenta que traduz os blocos seleccionados para uma HDL.

d) Produção de código executável para o processador, que aceda aos recursos reconfiguráveis. O código da aplicação é alterado, substituindo-se os blocos sintetizados por acessos aos recursos DLP.

Em alternativa ou em complemento a este processo de Detecção/Selecção/Síntese de Blocos Candidatos, alguns sistemas disponibilizam Bibliotecas de Configurações acessíveis aos programas do utilizador.

Dependendo do tamanho da função que se deseja sintetizar em *hardware*, a detecção/selecção de candidatos é posicionada em pontos diferentes da cadeia de compilação. Para PACs que mapeiam procedimentos completos em *hardware* (normalmente sistemas com integração paralela dos recursos DLP), a detecção/selecção é feita sobre o código fonte em linguagem de alto nível. Para PACs que utilizam os seus recursos configuráveis somente para optimização ao nível da manipulação de *bits*, a detecção/selecção é frequentemente feita sobre código *assembly* ou código máquina produzido para o processador hospedeiro.

Na construção do compilador existe um conjunto de princípios que devem ser, tanto quanto possível, tomados em conta:

- *O compilador deve esconder os pormenores da implementação do programador*. Assim, a escolha das secções do código a ser mapeadas para implementação em DLP e a produção de ficheiros de configuração relativos a estas secções para a implementação em lógica configurável deve ser feita automaticamente pelo compilador a partir de um programa numa linguagem de alto nível como, por exemplo, a linguagem C.
- *Quanto mais próximo um PAC estiver de uma arquitectura* standard *de processador mais simples será a construção do compilador*, visto que maior será o número de funcionalidades de compiladores já existentes que podem ser utilizadas. As plataformas com integração série são aquelas que se coadunam melhor com este princípio.
- *Se a quantidade de blocos candidatos ultrapassar a capacidade dos recursos configuráveis, devem ser sintetizados os blocos candidatos que implicarem maiores ganhos de desempenho*. Este princípio, embora óbvio, tem um corolário importante: as escolhas feitas pelo compilador devem tomar em conta o comportamento dinâmico do algoritmo.

E. Tipos comuns de Processadoresde Arquitectura Configurável

Podemos identificar dois tipos principais de PACs, que representam abordagens diferentes, com vantagens e desvantagens próprias:

- *Sistemas de Coprocessador Configurável:* sistemas, por definição, de integração paralela, utilizam grande quantidade de recursos configuráveis e mapeiam procedimentos completos da aplicação em *hardware*. Este tipo de sistemas exige uma forma de sincronização entre processador hospedeiro e DLP, e só se torna eficiente quando existem largos segmentos de código repetidos frequentemente e que, de preferência, possam ser processados em paralelo com outras partes do código. As aplicações que mais beneficiam desta abordagem são aplicações em tempo real, que impliquem manipulação de *bits* e que utilizem processamento concorrente. Como exemplo, podem-se citar aplicações de rede e multimédia.

- *Sistemas de Unidade Funcional Configurável:* sistemas de integração série, representam um investimento modesto em recursos reconfiguráveis, tendo tanto custos como desempenhos inferiores aos sistemas de coprocessador. Sendo a UFC integrada no *datapath* do processador, em paralelo com outras unidades funcionais, é necessário um modelo temporal determinístico que encaixe na *pipeline* do processador hospedeiro. Devido a isto, as UFCs têm normalmente arquitecturas simples, que implementam circuitos com um número de níveis lógicos limitado, e são muitas vezes dispositivos puramente combinacionais. As aplicações que mais beneficiam desta abordagem são aquelas que exigem manipulação de *bits* em tempo real, em sistemas com mudança de contexto frequente, como, por exemplo, aplicações encorporadas de criptografia.

Em ambos os casos o processo de compilação é um problema interessante e de todo não trivial. A boa escolha de sectores de código a implementar em *hardware* passa pelo estudo do comportamento em tempo de execução do código, está dependente do *scheduling* de instruções, dos tempos de reconfiguração do DLP (em casos de configuração dinâmica) e da capacidade do DLP. Em sistemas de coprocessador há também o problema da sincronização a ser levado em conta.

III. EXEMPLOS DE PROCESSADORES DE ARQUITECTURA CONFIGURÁVEL

A. PRISM (Processor Reconfiguration through Instruction Set Metamorphosis)

O sistema PRISM [7] foi desenvolvido no Instituto Politécnico e Universidade do Estado da Virgínia. Trata-se de um PAC com integração em paralelo: os recursos configuráveis estão acoplados a um processador de propósito geral por intermédio de um barramento externo. O protótipo construído consiste numa placa de processador Motorolla 68010 a 10 MHz ligado por intermédio de um barramento de 16 *bits* a uma segunda placa contendo quatro FPGAs 3090 da Xilinx. Foi

desenvolvido um compilador que, a partir de código em linguagem C produz dois tipos de informação, a imagem *software* e a imagem *hardware* de cada uma das funções do código fonte (ver Fig. 3).

A imagem *software* trata-se de código executável para o processador hospedeiro, ao passo que a imagem *hardware* é um ficheiro de configuração para os recursos DLP. O comportamento em execução do código é analisado e em seguida o compilador identifica no código fonte as funções adequadas para implementação em *hardware*. O sistema não é completamente automático visto que o programador deve escolher entre as funções propostas pelo compilador aquelas que efectivamente serão mapeadas no DLP.

Os resultados apresentados de *benchmarking* apontam para uma aceleração do processamento desde 100% a mais de 5000%, consoante a aplicação, comparativamente ao processador hóspede sem DLP.

B. PRISC (Programmable Instruction Set Computer)

O projecto PRISC [8] foi levado a cabo na Universidade de Harvard. O protótipo PRISC-I é baseado no *core* de um processador MIPS R2000 aumentado de uma UFC (embora o processo possa facilmente extender-se a múltiplas UFCs). A UFC, desenhada especificamente para o projecto (ver Fig. 4), é constituída por camadas alternadas de matrizes de interconexão e de blocos lógicos, blocos esses que implementam por meio de *Look-Up Tables* (LUTs) programadas, uma função lógica das suas entradas. A UFC representa um modesto investimento em *hardware*, equivalente a menos de 1KB de memória SRAM.

A ideia é adicionar os recursos configuráveis mantendo

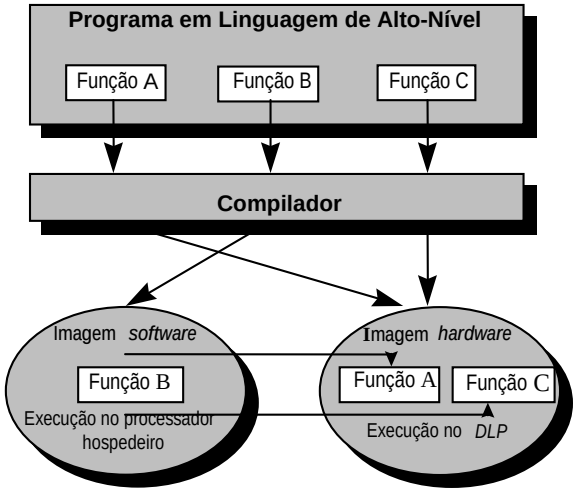

Fig. 3 A abordagem PRISM. As funções A, B e C duma aplicação representada em linguagem de alto nível são analisadas pelo Compilador de Configuração, o qual considera A e C mais adequadas para mapeamento em *hardware*, e B em *software*. As imagens *software* e *hardware* da aplicação são geradas. A imagem *software* inclui chamadas às funções implementadas em *hardware*.

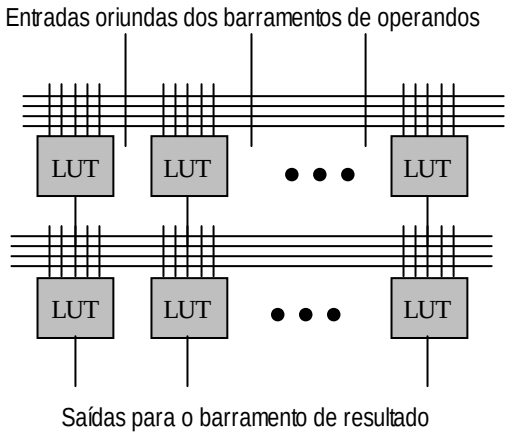

Fig. 4 A UFC do PRISC-I tem uma estrutura simétrica e por camadas.

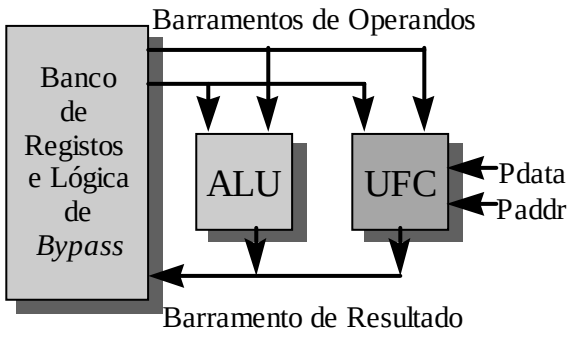

Fig. 5 O *datapath* do PRISC-I. A UFC é integrada na *pipeline* de um processador MIPS, de forma similar e em paralelo a outras unidades funcionais, como a ALU.

os benefícios das técnicas RISC, como seja o formato fixo das instruções e o *pipelining*, pelo que a UFC, acoplada em paralelo à ALU, (ver Fig. 5) tem o número de níveis de lógica limitado a 3, assegurando um modelo temporal determinístico que permite velocidades de relógio até 200 MHz.

Para aceder aos recursos configuráveis foi adicionada uma nova instrução ao conjunto de instruções do processador MIPS. A instrução *expfu* é uma instrução tipo R (ver Fig.6) que, além dos dois registos de entrada (*rs*, *rt*) e do registo de saída (*rd*), tem como argumento o número da configuração requerida (*Lpnum*).

6	5	5	5	11
expfu	Rs	rt	rd	Lpnum

Fig. 6 O formato da instrução *expfu*.

A política de configuração é dinâmica: a UFC tem um registo de 11 *bits* chamado *Pnum* que contém a identificação da função lógica correntemente programada. Sempre que a UFC deve executar uma função, o conteúdo de *Pnum* é comparado com o valor de *Lpnum* na instância da instrução *expfu* a executar. Caso a instrução apropriada esteja carregada, a execução prossegue. Em caso contrário, uma excepção ocorre e a rotina de atendimento à interrupção carrega a configuração desejada (o

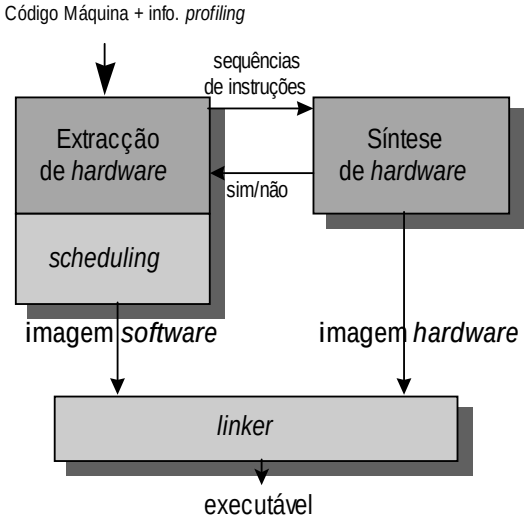

Fig. 7 Passos fundamentais do sistema de compilação do PRISC-I. O módulo de extracção de *hardware* envia sequências de instruções ao sistema de síntese de *hardware*, que gera informação de configuração da UFC. O sistema de síntese pode indicar ao módulo de extracção que a sequência de instruções não cabe na UFC.

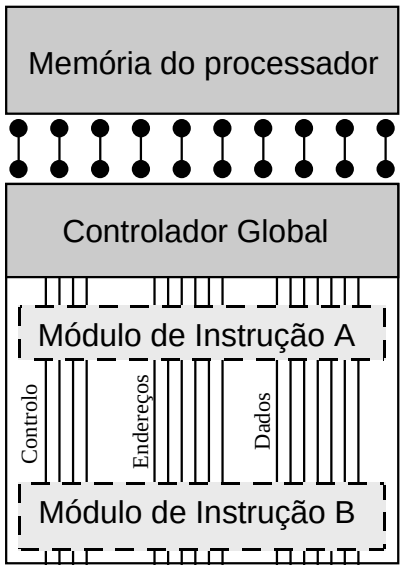

Fig. 8 A organização do *hardware* no DISC: os Módulos de Instrução são instalados horizontalmente ao longo da FPGA e atravessados por barramentos de Controlo, Endereços e Dados que permitem o contacto com os outros Módulos de Instrução e o Controlador Global.

carregamento de uma nova configuração demora cerca de 500 ciclos).

O compilador é responsável pela detecção e selecção de sequências de código que beneficiam de implementação em *hardware*, tendo em conta informação de *profiling* e os tempos de reconfiguração da UFC. A Fig. 7 mostra os passos essenciais da cadeia de compilação.

Os blocos candidatos a implementação na UFC são sobretudo sequências de instruções aritméticas/lógicas, mas também são consideradas certas instruções de controlo de fluxo, como sejam alguns casos de estruturas *if-then-else*. Os autores mostram [9] que o desempenho do processador para *short-sect vectors*, *hash-tables* e máquinas de estado finitas pode ter grandes melhorias, através da utilização de técnicas de aceleração de *hardware* que façam uso da UFC. Resultados apresentados indicam um *speed-up* médio de 22% para o conjunto de *benchmarks* SPECint92, em relação a um processador MIPS R2000 tradicional, mas não tomam em conta efeitos relacionados com o sistema de memória. Também se verifica um descongestionamento do banco interno de registos do processador, visto que deixa de ser necessário armazenar os vários resultados intermédios no interior de cada bloco sintetizado.

C. DISC (Dynamic Instruction Set Computer)

Implementado utilizando uma FPGA comercial, a National Semiconductor CLAy, que tem capacidade de reconfiguração dinâmica e parcial, o DISC [10] trata todas as instruções como módulos substituíveis de acordo com os requisitos do programa em execução. A cada instrução está associado um determinado circuito digital independente, dito um *módulo de instrução*, ao qual corresponde um mapa de configuração parcial e realocável da FPGA. Um processador actuando como controlador global gere a memória e o processo de configuração do dispositivo, sequencia a execução de instruções e armazena o estado do sistema.

A superfície da FPGA está particionada horizontalmente em várias áreas onde podem ser carregados os módulos de instruções, como se pode ver na Fig. 8. Quando uma determinada instrução é necessária, o controlador global verifica se a instrução em causa corresponde à configuração de um dos módulos carregados e, em caso afirmativo, procede à execução desse módulo. Caso contrário, o controlador global pára a execução e carrega a configuração desejada num espaço livre na superfície da FPGA. Se não houver espaço suficiente para o módulo requerido, um módulo carregado é seleccionado por uma política simples de *Least Recently Used* e descartado para libertar espaço. Uma rede interna de comunicação permite o acesso aos recursos globais para todos os módulos de instruções, bem como a comunicação entre módulos.

O processador utilizado para implementar o controlador global é extremamente simples, tratando-se de um processador de 8 *bits* com um conjunto de instruções mínimo necessário ao suporte da linguagem C.

Nenhum sistema de compilação foi desenvolvido para o DISC. As várias configurações de módulos funcionais são escritas pelo programador em HDL. O código executável

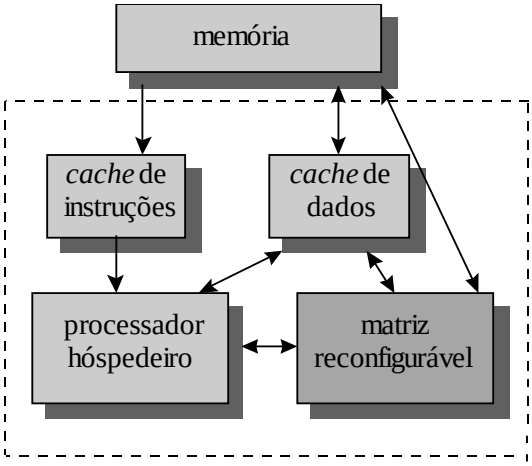

Fig. 9 Diagrama de blocos do Garp. A matriz reconfigurável tem acesso directo aos vários níveis de memória do sistema.

para o controlador global pode ser escrito em linguagem C.

D. Garp

O Garp [11] foi desenvolvido na Universidade da Califórnia, em Berkeley. Como no DISC, os módulos de *hardware* são colocados em secções horizontais da lógica configurável, reconfiguradas em tempo real e individualmente (reconfiguração dinâmica e parcial). O carregamento e execução de configurações é controlado pelo processador hospedeiro.

O *core* de processador utilizado é o de um MIPS-II (com o conjunto de instruções extendido de algumas instruções específicas para permitir a comunicação com os recursos configuráveis), pelo que no Garp o processador hospedeiro é utilizado para computação e não apenas para efeitos de controlo. Os recursos reconfiguráveis são implementados por meio de uma FPGA projectada especialmente para este sistema, à qual é dado acesso directo à memória externa do sistema (ver Fig. 9).

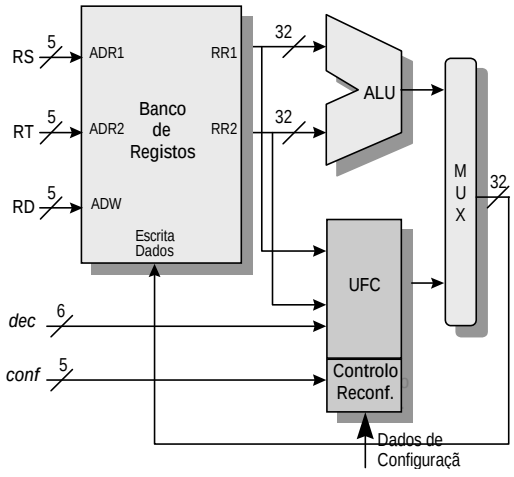

Fig. 10 O *datapath* do ConCISe.

O sincronismo entre o processador e o DLP é implementado por meio de um contador decrescente de ciclos de relógio. Este contador é inicializado com um certo valor cada vez que o DLP é acedido, e a próxima instrução só será executada quando este contador chegar a zero, indicando que terminou o processamento no DLP.

O problema da compilação é resolvido fazendo uso de bibliotecas pré-definidas de configurações, acedidas através de directivas *#include* no código fonte em C.

Os resultados de simulação obtidos para um Garp com relógio de 133 MHz, indicam computações entre 2 e 24 vezes mais rápidas que um UltraSparc a 166 MHz, verificando-se o melhor resultado para o algoritmo de criptografia DES.

E. ConCISe

O ConCISe [12] foi desenvolvido nos laboratórios da Philips Research, em Eindhoven. Como o PRISC, trata-se de uma arquitectura híbrida com integração série dos recursos configuráveis, sob a forma de uma UFC (ver Fig 10). O processador hospedeiro é igualmente um RISC. Contudo, o ConCISe difere do PRISC em alguns pontos importantes. Em primeiro lugar, a UFC é baseada no *core* de uma CPLD comercial de arquitectura XPLA-II (tecnologia recentemente vendida pela Philips à Xilinx). Esta arquitectura tem como características específicas um modelo temporal extremamente determinístico (mesmo para uma CPLD) e uma alta capacidade de encaminhamento de projectos com pinos pré-determinados, características estas que facilitam imenso a sua inserção na *pipeline* de um processador.

Além disso, cada configuração da UFC codifica várias funções multiplexadas. A UFC aceita, por isso, além de duas entradas de 32 *bits* cada, para os operandos, uma

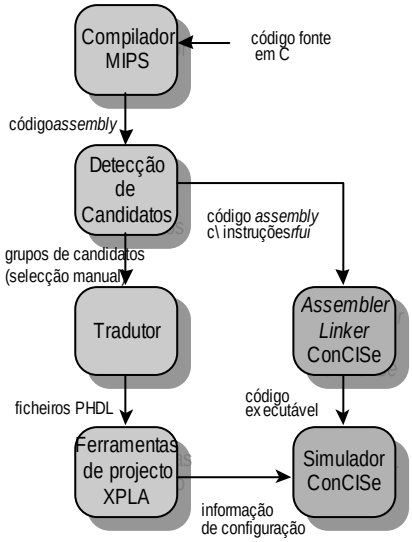

Fig. 12 Cadeia de compilação do simulador da arquitectura ConCISe

Sistema	DLP	Integração	Configuração	Compilação
PRISM	FPGA *Xilinx 3090*	Paralela	Estática	Detecção de candidatos automática. Selecção de candidatos manual.
PRISC	FPGA* (lógica puramente combinatória)	Série	Dinâmica Global	Detecção e selecção de candidatos automática.
DISC	FPGA *National CLAy*	Série.	Dinâmica Parcial.	Toda a aplicação é implementada no DLP. As configurações dos módulos funcionais são escritas directamente pelo programador.
Garp	FPGA*	Paralela.	Dinâmica Parcial.	Biblioteca de Configurações.
ConCISe	CPLD *Xilinx XPLA2* (versão simplificada, lógica puramente combinatória)	Série.	Estática (ou Dinâmica Global).	Detecção de Candidatos Automática. Selecção de Candidatos (ainda) manual.

*dispositivo especificamente desenhado para o sistema em causa

Tab. 1 Quadro resumo das características dos sistemas apresentados

entrada extra para seleccionar qual das funções contidas na actual configuração da UFC é que deve ser executada.

O conjunto de instruções do processador é aumentado de uma nova instrução, *rfui*, para aceder à nova unidade funcional. Esta instrução tem o formato apresentado na Fig. 11.

6	5	5	5	5	6
Rfui	rs	rt	rd	conf	Dec

Fig. 11A instrução *rfui.*

em que o campo *conf* corresponde ao número da configuração pretendida[2] e *dec* à instrução pretendida dentro dessa configuração.

A cadeia de compilação (que pode ser vista na Fig. 12) detecta blocos candidatos que são blocos de instruções constituídos apenas por instruções aritméticas/lógicas, com uma estrutura de dependências que implique apenas dois registos como entrada e um registo como saída. De acordo com um estudo do comportamento do programa em execução, são escolhidos para síntese em *hardware* os blocos candidatos que representarem maiores ganhos (este processo deverá vir a ser automático, embora, até à data, todos os resultados obtidos se baseiem em selecção manual). Em seguida, o compilador produz ficheiros de configuração para a UFC e código executável, substituindo os blocos seleccionados por chamadas à uma instrução *rfui*.

Embora a possibilidade de reconfiguração dinâmica tenha sido estudada, a capacidade de multiplexação de várias instruções numa única configuração torna, na maior parte dos casos, esta característica desnecessária. Por meio de simulação (utilizando um MIPS R3000 como processador hospedeiro, e tendo em conta o efeito da hierarquia de memória) obtiveram-se [13], para algoritmos de criptografia, *speed-ups* desde 8% a 62%, sendo que o melhor resultado obtido foi para o DES.

[2]O campo *conf* foi incluído prevendo a possibilidade de utilizar o sistema com reconfiguração dinâmica, o que não foi feito nos testes realizados, como veremos mais adiante.

V. CONCLUSÃO

Neste artigo foi dada uma visão geral dos Processadores de Arquitectura Configurável, incluindo factores de implementação e exemplos.Na Tab. 1 apresenta-se um quadro resumo dos sistemas referidos. Alguns sistemas interessantes ficaram por referir, como o *OneChip* [14] ou o *Chimaera* [15], entre outros. Não foram abordados também outros tipos de arquitecturas de computador baseadas em DLPs, como sejam as arquitecturas multi-FPGA [4].

Não existem ainda no mercado processadores com arquitecturas (re)configuráveis. Tendo em conta os interessantes resultados obtidos com protótipos tal não deve, de qualquer forma, tardar a acontecer.

AGRADECIMENTOS:

O primeiro autor deste artigo deseja agradecer a Bernardo Kastrup e a todo o *Compiler Technology Cluster* do Nat.Lab, Laboratório de Investigação da *Philips Research*, em Eindhoven nos Países-Baixos, onde fez um estágio de 6 meses durante o qual ganhou grande parte dos conhecimentos que tem nesta área, e que possibilitaram a escrita deste artigo.

REFERÊNCIAS:

[1] S. Brown e J. Rose, "Architecture of FPGAs and CPLDs: A Tutorial", *IEEE Design and Test of Computers*, Vol. 13, Nº 2, pp 42-57, 1996

[2] S. Brown *et al*, *Field-Programmable Gate Arrays*, Kluwer, ISBN 0-7923-9248-5, Junho, 1992

[3] D. Bursky, "Advanced CPLD Architectures Challenge FPGAs, *Electronic Design Online*, Vol 46, Nº 22, Outubro 1998, URL:devl.pentom.com/ed/Pages/magpages/oct0198/digdes/1001del.htm

[4] S. Hauck, "The Roles of FPGAs in Reprogrammable Systems", *Proceedings of the IEEE*, Vol. 86, Nº 4, pp 615-639, Abril, 1998

[5] J. Hennessy e D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kauffmann Publishers, 1990

[6] G. Estrin, "Organization of Computer Systems: The Fixed-Plus Variable Structure Computer", *Proc. Western Joint Computer Conf.*, pp 33-40, Am. Inst. Electrical Engineers, Nova Iorque, 1960

[7] P. M. Athanas e H.F. Silverman, "Processor Reconfiguration Through Instruction Set Metamorphosis", *Computer*, 26, 3, pp 11-18, Março ,1993

[8] R. Razdan e M. D. Smith, "A High Performance Microarchitecture with Hardware-Programmable Functional Units", *Proceedings of the 27th Annual IEEE/ACM Intl. Symposium On Microarchitecture*, pp 172-180, Novembro, 1998

[9] R. Razdan e M. D. Smith, "PRISC Software Acceleration Techniques", *Proceedings of the IEEE Intl. Conference On Computer Design*, pp 145-149, Outubro, 1994

[10] M. J. Wirthlin e B. L. Hutchings, "A Dynamic Instruction Set Computer", *IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, Abril, 1995

[11] J.R. Hauser e J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor", *Proccedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp 24-33, 1997

[12] B. Kastrup *et al*, "ConCISe: A Compiler-Driven CPLD-Based Instruction Set Accelerator", *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Abril, 1999

[13] O. Moreira, "Benchmarking ConCISe: Simulation-Based analysis of a Hybrid Reconfigurable Architecture", Philips Nat.Lab. Report NL-UR 821/99, Agosto, 1999

[14] R. D. Wittig e P. Chow, "OneChip: An FPGA Processor with Reconfigurable Logic", *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp 126-135, Los Alamitos, CA, Abril, 1996

[15] S. Hauck *et al*, "The Chimaera Reconfigurable Functional Unit", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1997