Circuitos Virtuais Baseados em Reprogramação e Reconfiguração Dinâmica

Valery Sklyarov, Andreia Melo, Arnaldo Oliveira, Nuno Lau, Ricardo Sal Monteiro

Resumo - **Este artigo aborda a reconfiguração dinâmica das FPGAs (*Field Programmable Gate Arrays*) e o seu uso em diferentes aplicações práticas. A técnica aqui considerada apresenta diferentes formas que podem ser usadas no projecto de circuitos lógicos virtuais que possuam as propriedades de flexibilidade, extensibilidade e reutilização. Estas podem ser encontradas nas técnicas de orientação por objectos, de onde podem surgir algumas ideias para o desenvolvimento de hardware: o encapsulamento (tornando possível a construção de objectos de hardware), herança (permitindo a reutilização de objectos já existentes) e a modificação da funcionalidade de um objecto de hardware durante o funcionamento normal do circuito. De forma a dotar os objectos de hardware com estas facilidades e eliminar algumas restrições é sugerida uma arquitectura da FPGA que combina as suas células lógicas com matrizes de estrutura regular que podem ser movidas dentro da FPGA.**

Abstract - **This paper discusses dynamically reconfigurable field programmable gate arrays (FPGA) and their use for different practical applications. The technique considered presents different ways that might be used in order to design virtual logic schemes, which have acquired such properties as flexibility, extensibility and reusability. Since object-oriented technology allows to design software systems with such facilities, we have been trying to apply some ideas of this technology to the design of hardware circuits based on reprogrammable and reconfigurable logic blocks, such as FPGA cells. These ideas are encapsulation, making it possible to construct hardware objects, inheritance, allowing to provide reuse of existing hardware objects, and run time support, which enables us to modify the functionality of hardware objects on the fly, i.e. dynamically during run-time. In order to provide hardware objects with the facilities considered above and eliminate some constrains, we suggest an architecture of FPGA combining its logic cells with regular matrix structures that are moveable within an FPGA. Finally the paper suggests some reasonable ways, aimed at developing FPGA-based circuits.**

I. INTRODUÇÃO

Os modelos de circuitos digitais podem ser visualizados recorrendo a diferentes níveis de abstracção [1]. Cada nível de abstracção descreve diferentes características e contém os detalhes que lhes estão associados. Neste artigo consideram-se dois: arquitectural e lógico. No primeiro nível um circuito é caracterizado por um conjunto de operações. No segundo nível, um circuito é definido por um conjunto de funções lógicas. A síntese arquitectural, também referida como síntese de alto nível ou síntese estrutural, permite-nos criar uma estrutura macroscópica para um circuito representando-o por um conjunto de blocos. Por outras palavras, o processo de síntese deve definir o *datapath* como um conjunto de recursos interligados (tais como registos, barramentos de dados e blocos funcionais), bem como um modelo lógico da unidade de controlo [1].

As tarefas principais da síntese arquitectural são *scheduling*, partilha de recursos, *binding* e criação do *datapath* e da unidade de controlo (estas tarefas são descritas em detalhe em [1]). A síntese de nível lógico é aplicada a circuitos combinatórios e sequenciais, permitindo-nos obter uma implementação do circuito na forma de uma *netlist* de portas ou de outros elementos tais como células lógicas, matriciais ou funcionais.

A combinação de um *datapath* e de uma unidade de controlo forma um dispositivo operacional que pode ser visto como um objecto de hardware. Baseado neste conceito um sistema digital pode ser modelado como um conjunto de objectos de hardware que comunicam entre si, em que a execução das funções de cada objecto é requerida através do envio de mensagens. Cada mensagem está relacionada com uma unidade de controlo específica e condiciona o seu comportamento de acordo com o que foi requerido (e, consequentemente, com o que se deseja). Neste tipo de abordagem o sistema deve ser decomposto de tal modo que os seus componentes sejam abstracções chave no domínio do problema. Esta decomposição, baseada numa colecção de objectos que comunicam entre si com o objectivo de cumprir uma determinada função, é designada por decomposição orientada por objectos (Object-Oriented Decomposition – OOD) [2].

De notar que o comportamento de um dado objecto de hardware é proporcionado pela sua unidade de controlo que é também responsável pela interface externa do objecto. Por outras palavras, a unidade de controlo responde às mensagens (externas) que vai recebendo, executando as acções por estas requisitadas. Em muitas aplicações práticas (em particular aplicações embebidas – *embedded applications* [3]), os sistemas digitais são heterogéneos e por isso as suas especificações podem

alterar-se continuamente. Sendo assim, necessitamos de os dotar de facilidades como a flexibilidade e extensibilidade que podem ser atingidas explorando a tecnologia *field-programmable*, isto é, circuitos integrados que podem ser facilmente alterados a partir do próprio sistema em que estão integrados [4]. Esta tecnologia torna possível a alteração da estrutura e da configuração de circuitos de hardware destinados a uma aplicação em particular, após fabrico dos mesmos. Os dispositivos *field-programmable* (PLDs) podem ser divididos nos seguintes grupos: PLDs simples, PLDs complexos (CPLDs) e FPGAs – *Field Programmable Gate Arrays* [4]. De notar que as FPGAs reconfiguráveis dinamicamente como a Xilinx XC6200 [5,6] podem ser integradas num sistema computacional e configuradas em *run-time* de forma a implementar uma função de software específica, com um elevado grau de desempenho. Estas FPGAs podem ainda ser reprogramadas parcialmente sem suspender a operação das restantes partes que não necessitam de reconfiguração. Isto é conseguido especificando as interligações entre os componentes lógicos usando SRAM. A reconfiguração dinâmica parcial é implementada definindo uma "janela" no mapa de memória numa posição específica X,Y que poderá ser carregada e alterada durante o funcionamento do circuito.

A utilização de FPLDs (*field programmable logic devices*) permite-nos o desenvolvimento de circuitos virtuais, o que torna possível a utilização do mesmo hardware para a implementação de múltiplas funcionalidades. A ideia principal é a seguinte: o circuito possui hardware, o qual realiza operações de acordo com os nossos requisitos. No entanto, o mesmo hardware pode também ser utilizado para implementar novas funções. Esta reutilização do hardware pode ser efectuada substituindo parte das funções anteriormente implementadas por novas funções. A substituição pode ser efectuada de um modo estático ou dinâmico.

No segundo caso temos que proporcionar suporte para modificação em *run–time*. Isto significa que mesmo que os recursos lógicos existentes num circuito sejam inferiores àqueles que actualmente necessitamos para resolver o nosso problema, o problema pode ser decomposto em blocos de menor dimensão, que podem ser executados de forma sequencial em diferentes instantes de tempo. O esquema virtual será assim modificado em *run-time*, de forma a que o mesmo hardware seja utilizado para a execução de diferentes sub-tarefas em diferentes instantes de tempo. Por exemplo, o modelo das máquinas de estados finitos hierárquicas (*Hierarchical Finite State Machine - HFSM*) consideradas em [8,9] possuem estas propriedades. A principal característica e vantagem das HFSMs é a possibilidade de configurar e executar todas as operações necessárias com macro-operações e funções lógicas virtuais. Uma macro-operação (ou função lógica) é chamada virtual se pode não ser associada de forma permanente durante o projecto de uma unidade de controlo. Para qualquer elemento virtual (EV), que pode ser uma macro-operação, ou uma função lógica, podemos aceitar diferentes implementações futuras. Um EV é descrito por um Grafo Hierárquico (*Hierarchical Graph-Schemes - HGS*) virtual [8,9]. Um HGS virtual pode ser encarado como uma parte variável na implementação de um algoritmo de controlo.

Um algoritmo hierárquico de uma unidade de controlo pode ser descrito por um conjunto de HGSs. Cada HGS tem associados dados e deve ser executado de acordo com uma sequência de controlo. A combinação de registos, unidades funcionais e lógica de encaminhamento torna possível o processamento dos dados de forma a obtermos todos os cálculos necessários à produção de resultados da execução de um algoritmo hierárquico.

Se queremos projectar um objecto de hardware virtual para que a sua funcionalidade possa ser alterada após a sua implementação física (mesmo dinamicamente em *run-time*), temos de construir um circuito de controlo virtual de forma a ser possível substituir ou modificar os blocos *self-contained* que constituem o algoritmo de controlo, que serão HGSs relativamente autónomos. Além disto, devemos construir uma biblioteca de componentes reutilizáveis para *datapath*. De forma a simplificar a substituição de componentes, devemos estabelecer uma correspondência directa entre os HGSs e os elementos reprogramáveis (reconfiguráveis) do esquema lógico futuro. Neste caso, se pretendemos modificar ou substituir o HGS, temos apenas de reprogramar (reconfigurar) o respectivo elemento.

As noções de reprogramação e reconfiguração têm sido usadas no seguinte sentido. Um circuito reprogramável baseia-se em elementos de funcionalidade modificável, como esquemas baseados em matrizes [18]. Um esquema reconfigurável contém elementos cujas interligações podem ser alteradas numa área reservada especialmente para *routing*.

A complexidade das recentes FPGAs permite a implementação de um sistema completo, incluindo memória, *datapath*, interface e unidades de controlo. Assim, a existência de áreas diferentes da FPGA dedicadas a certos tipos de circuitos poderá fazer sentido [10]. Uma introdução aos diferentes métodos de projecto de circuitos em FPGAs pode ser encontrado em [11].

II. HARDWARE RECONFIGURÁVEL DINAMICAMENTE

A. Introdução às FPGAs

A partir de meados dos anos 80 uma nova tecnologia começou a ser utilizada na implementação de circuitos digitais, as FPGAs. As FPGAs são circuitos digitais programáveis pelo utilizador final constituídos por várias unidades funcionais que se podem ligar entre si de uma forma programada (Fig. 1) [12]. Em cada unidade funcional a função lógica que determina o valor das saídas a partir dos valores das entradas pode também ser programada. A implementação um dado circuito lógico

numa FPGA é realizada através da partição do circuito em unidades funcionais realizáveis na FPGA, da sua interligação de forma conveniente e da programação da FPGA de acordo com o pretendido.

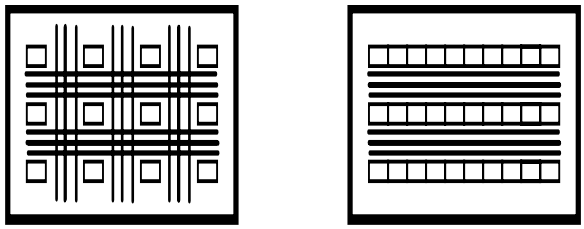

Fig. 1 - Arquitecturas possíveis para FPGAs : baseadas em matriz ou em linhas de unidades funcionais.

As possibilidades lógicas das unidades funcionais, bem como as possibilidades de programação das suas interligações devem permitir a implementação de uma variedade de circuitos lógicos tão grande quanto possível. Como exemplos de arquitecturas de unidades funcionais temos aquelas que são baseadas em multiplexador e aquelas que são baseadas em *look-up tables* (Fig. 2). Os recursos de interligação entre unidades funcionais tanto podem privilegiar a ligação local entre blocos próximos como a ligação entre blocos distantes. Existem FPGAs disponíveis com cerca de 16000 unidades funcionais para um total de cerca de 100000 portas lógicas.

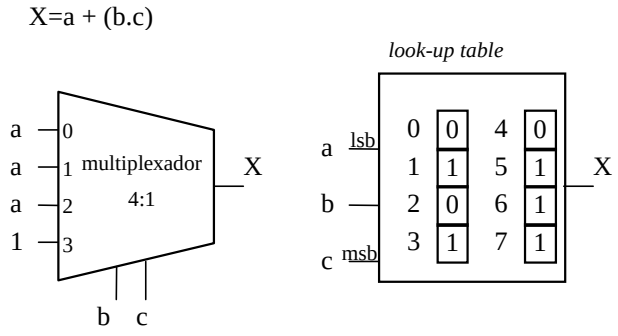

Fig. 2 - Exemplo de implementação de um dado circuito lógico utilizando um multiplexador ou uma *look-up table*. No caso de estas unidades serem unidades funcionais de uma FPGA então as ligações das entradas do multiplexador e a tabela interna da *look-up table* seriam programáveis.

As tecnologias de programação de FPGAs são as seguintes:

- SRAMs: a FPGA contém uma memória RAM interna cujo conteúdo define a estrutura das interligações e das unidades funcionais. Este tipo de FPGAs pode ser programado várias vezes num espaço de tempo curto, mas a programação da FPGA deve ser recarregada de cada vez que o circuito for ligado;
- Anti-fusíveis: Estes dispositivos permitem a interligação programada de 2 condutores, funcionando de uma forma inversa à dos fusíveis correntes, no estado normal não existe ligação, se sujeitos a uma tensão elevada (durante a programação) garantem a interligação dos condutores. A programação destes dispositivos é realizada uma única vez e permanece activa para sempre;
- EPROM e EEPROMs: o conteúdo da EPROM ou EEPROM define a configuração da FPGA. Tal como seria de esperar este tipo de FPGAs permitem a sua reprogramação, e não necessitam de alimentação para manter a configuração. A sua reprogramação é mais lenta do que a tecnologia SRAM e exige um dispositivo de programação específico. No caso das EEPROMs é possível integrar o dispositivo de programação no próprio circuito.

As FPGAs têm sido utilizadas principalmente como dispositivos de:

- Substituição de outros tipos de circuitos lógicos – uma FPGA, dada a sua arquitectura multi-nível e alta capacidade lógica pode substituir algumas dezenas de PALs, ou mesmo ser mais rentável na implementação de um ASIC, se a produção for de pequenas quantidades, do que outras tecnologias disponíveis tais como os circuitos programáveis por máscaras (MPGAs);
- A prototipagem de circuitos – as FPGAs tornaram possível o teste do projecto de um circuito digital complexo em hardware, com todas as implicações que daí podem advir em termos de velocidade e de eficácia dos testes realizados.

B. Circuitos reconfiguráveis

Com o aumento de capacidade das FPGAs e o melhoramento das ferramentas de desenvolvimento o uso de FPGAs baseadas em SRAMs, como elementos de processamento reconfiguráveis na própria placa do sistema, tem merecido redobrada atenção. Esta propriedade de reconfiguração dos elementos de processamento permite implementar sistemas electrónicos com uma característica inovadora: a possibilidade de alterar facilmente as funções de partes do hardware durante o funcionamento normal do sistema.

Apesar dos sucessivos aumentos da capacidade das FPGAs esta é ainda menor que a dos ASICs programáveis por máscaras. Esta diferença, superior a uma ordem de grandeza, advém da redundância inerente à arquitectura das FPGAs, que se deve essencialmente aos elementos necessários para permitir a sua reconfiguração.

As limitações de capacidade das FPGAs também têm servido de motivação para explorar as suas capacidades de reconfiguração. A partilha temporal de uma mesma FPGA para implementar sequencialmente circuitos diferentes ao longo do tempo, permite obter uma maior eficiência no uso das *gates* que a constituem, permitindo reduzir o número de FPGAs necessárias para construir um dado sistema.

O conceito de sistema reconfigurável pode ser usado tanto em sistemas de controlo embebidos como em sistemas de coprocessamento associados a processadores de uso geral.

Os sistemas de controlo embebidos reconfiguráveis podem ser constituídos por uma ou mais FPGAs ligadas a uma memória externa contendo as configurações necessárias para implementar as diversas funções a desempenhar por cada FPGA podendo ainda conter lógica implementada em dispositivos não reconfiguráveis (Fig. 3). Nestes casos o próprio sistema pode detectar a necessidade de alterar a sua própria funcionalidade e carregar a configuração necessária a partir da memória externa.

Os sistemas de coprocessamento baseados em circuitos reconfiguráveis têm como principal objectivo acelerar o processamento de tarefas computacionalmente mais intensivas realizadas em computadores de uso geral. Neste tipo de sistemas além do tipo mais puro de coprocessamento, em que o coprocessador baseado em FPGAs desempenha tarefas de processamento digital de dados a pedido do processador de uso geral (Fig. 4-a), podemos também incluir controladores de elevado desempenho baseados em FPGAs que cooperam com o processador de uso geral em tarefas mais específicas, como por exemplo visualização e manipulação 3D, preparação e controlo de impressões, codificação e descodificação de FAX, etc (Fig. 4-b).

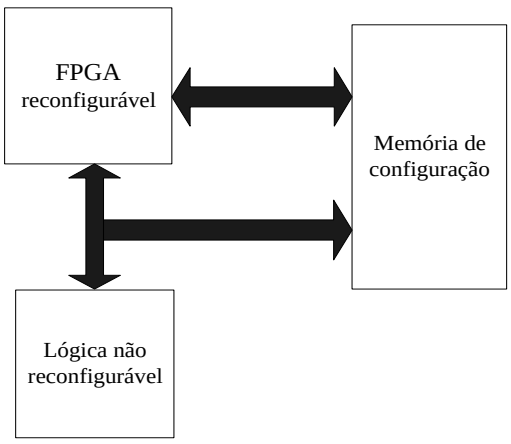

Fig. 3 - Exemplo de um sistema de controlo embebido reconfigurável.

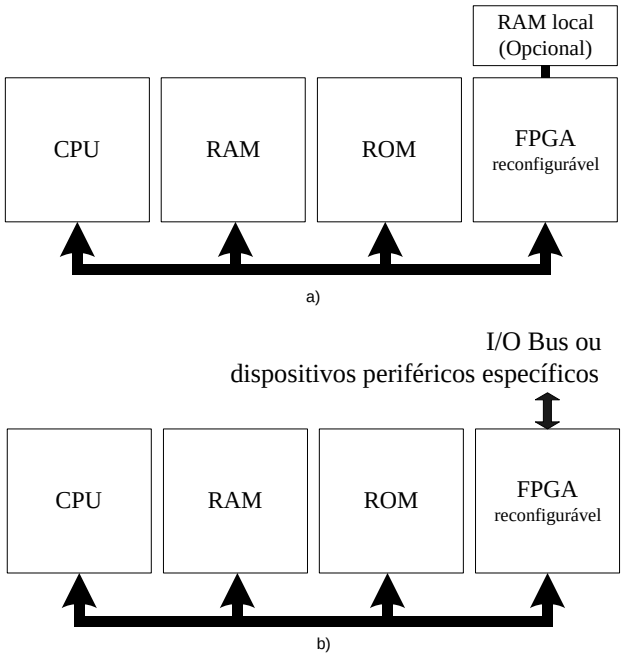

Fig. 4 - Exemplo de um sistema computacional com coprocessador reconfigurável (a) e de um com controlador reconfigurável (b).

A utilização destes sistemas de coprocessamento reconfiguráveis tem despertado o interesse daqueles que necessitam de resolver tarefas de processamento digital de dados mais exigentes em termos de tempo de processamento, por constituir uma alternativa viável às implementações tradicionais a um preço razoável [13]. De facto estes sistemas, também referidos como processadores reconfiguráveis, permitem combinar a versatilidade de utilização de um processador de uso geral com o melhor desempenho de hardware específico.

Desde o final dos anos 80 início dos anos 90, os processadores reconfiguráveis baseados em FPGAs e a sua utilização, também referida como computação reconfigurável (*Reconfigurable Computing*), têm obtido o suporte da comunidade científica e académica, assim como o interesse das companhias comerciais, tendo sido alvo de dezenas de projectos de investigação [14-17] e integrado alguns produtos comerciais.

A maioria destes sistemas reconfiguráveis tem sido implementado com as FPGAs reconfiguráveis mais comuns disponíveis no mercado, como por exemplo, a família XC3000 ou XC4000 da Xilinx. Contudo este tipo de FPGAs apresenta algumas limitações que têm um grande impacto na construção de sistemas reconfiguráveis:

- Os tempos de reconfiguração são relativamente elevados, na ordem das dezenas ou centenas de milisegundos;
- Não é possível a alteração parcial do circuito implementado na FPGA, se se pretende alterar a

configuração de apenas uma parte da FPGA, toda a FPGA deve ser reprogramada do ínicio, ou seja o tempo de reconfiguração não depende da zona a alterar mas do tamanho total da FPGA;

- Durante a reconfiguração o funcionamento dos circuitos implementados na FPGA é suspenso, por exemplo colocando os pinos de função programável num estado de alta impedância;
- O interface com o bus do processador de uso geral tem de ser construído usando recursos lógicos da própria FPGA, o que diminui a quantidade de lógica disponível;
- Todos os dados registados no interior da FPGA, que definem o estado do sistema antes da reconfiguração são apagados, não existindo nenhuma forma de reter a informação dentro da FPGA entre reconfigurações. Toda a informação que passa de uma reconfiguração para a seguinte deve ser acessível através dos pinos de saída da FPGA.

C. Circuitos reconfiguráveis dinamicamente

Um circuito reconfigurável dinamicamente deve permitir a alteração da sua configuração (estrutura das unidades funcionais e interligação entre elas) em *runtime*. Ou seja, deve permitir a alteração da configuração de uma parte do circuito enquanto que o resto do circuito se mantém em funcionamento normal. No fundo o circuito reconfigurável dinamicamente deve eliminar as restrições que foram apontadas para os circuitos reconfiguráveis.

Este tipo de circuito abre perspectivas para um novo tipo de utilizações das FPGAs em que estas passariam a funcionar como hardware virtual, numa analogia com a chamada memória virtual, em que ao longo do tempo diferentes partes de um circuito, ou mesmo diferentes circuitos seriam carregados e descarregados da FPGA, à medida que fossem necessários.

A utilização de mecanismos de reprogramação e reconfiguração dinâmica no projecto de sistemas digitais pode ter dois grandes campos de aplicação:

- Implementação de circuitos de grande dimensão e complexidade com uma economia de hardware que pode ser bastante significativa. Este tipo de abordagem pode ser utilizada, por exemplo, no projecto de sistemas que implementem algoritmos de controlo complexos. Neste caso, desde que seja possível a sua decomposição em blocos de menor complexidade e de execução disjunta no tempo, podemos pensar numa solução em que somente os blocos activos se encontram implementados na FPGA. Os restantes, ainda não executados ou temporariamente suspensos, estão residentes em memória externa à FPGA.
- Coprocessadores destinados ao cálculo de algoritmos computacionamente intensivos que podem ser transpostos para hardware, contribuindo assim para um aumento do desempenho do sistema. Dependendo da complexidade dos algoritmos, o mesmo coprocessador (virtual) pode ser partilhado por várias aplicações e algoritmos sem que estes interfiram entre si.

Obviamente que a reprogramação dinâmica da FPGA exige que exista um circuito de controlo semelhante aqueles que eram utilizados para reprogramar as FPGA reconfiguráveis, mas agora com a possibilidade de uma reconfiguração parcial e não total. O circuito de controlo deve possuir capacidades de reserva e libertação dinâmica de recursos. Sempre que um dado bloco/algoritmo tiver de ser executado mas ainda não estiver residente na FPGA, deve ser feita a transferência da memória externa para a memória de configuração da FPGA do *bitmap* correspondente. É de notar que dada a arquitectura simétrica deste tipo de FPGAs o mesmo bloco poderá funcionar da mesma forma em diferentes zonas da FPGA. Por outro lado, sempre que for necessário executar um novo bloco não mapeado na FPGA e nesta não existir nenhuma zona disponível para o fazer, deve-se seleccionar, entre os blocos que estão mapeados na FPGA, um que já não seja necessário, ou que esteja temporariamente suspenso e configurar a zona ocupada pelo bloco inactivo com o novo bloco (Fig. 5).

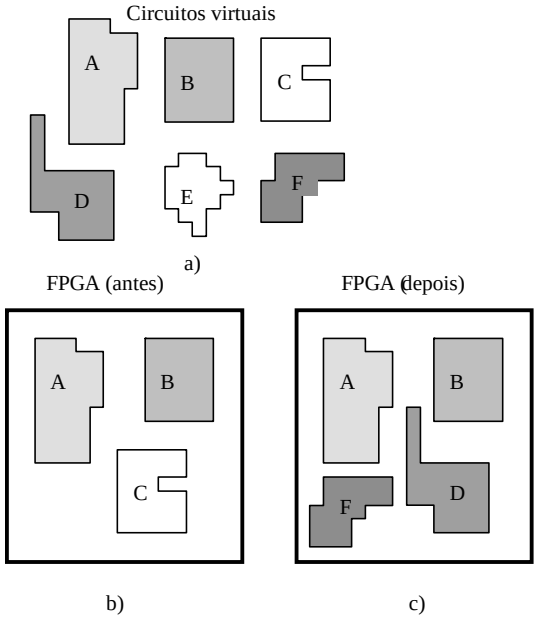

Fig. 5 - Exemplo de um conjunto de configurações (a). No estado anterior (b), os blocos A e B estão activos e é necessário mapear os blocos D e F e depois da reconfiguração parcial A, B, D e F estão activos (c).

D. A família XC6200

Existem actualmente duas famílias de circuitos reconfiguráveis dinamicamente a Xilinx XC6200 [5] e a Atmel AT6000. Este artigo irá descrever com mais pormenor o funcionamento da família XC6200.

A arquitectura da família XC6200 foi concebida de modo a que estes dispositivos pudessem ser facilmente usados como coprocessadores reconfiguráveis de um processador principal, embora nada impeça que estes possam ser usados noutros tipos de aplicações, quer utilizem ou não a reconfiguração dinâmica.

As principais características que diferenciam esta família de FPGAs das anteriores são as seguintes:

- Interface FastMAP[tm] : estas FPGAs incluem uma interface desenhada para ser ligada directamente ao barramento do processador principal, sendo o acesso à FPGA realizado da mesma forma que o acesso a memória. Esta interface pode ser configurada para uma largura de dados de 8, 16 ou 32 bits. Qualquer bit de qualquer registo interno de uma unidade funcional da FPGA pode ser mapeado na memória do processador principal;
- Reconfiguração dinâmica da FPGA: usando a interface FastMAP[tm] é possível reconfigurar apenas uma parte da FPGA enquanto o resto da FPGA se mantém activo. Esta interface permite reconfigurações muito mais rápidas do que as reconfigurações normais (segundo o fabricante até 1000 vezes mais rápida);
- Unidade funcional simples: a FPGA é constituída por uma matriz de unidades funcionais baseadas em multiplexadores que podem implementar qualquer função lógica de 2 entradas, qualquer mutiplexer de 2 para 1, valor constante (0 ou 1), ou qualquer função de apenas uma entrada, ou ainda qualquer uma das anteriores mais um flipflop do tipo D (Fig. 6). As interligações entre as unidades funcionais estão organizadas de forma hierárquica constituindo grupos de 4*4 e 16*16 células. Dentro de cada grupo existem determinadas capacidades de interligação e cada grupo terá recursos de interligação específicos que lhe permitem a interligação com os grupos adjacentes;

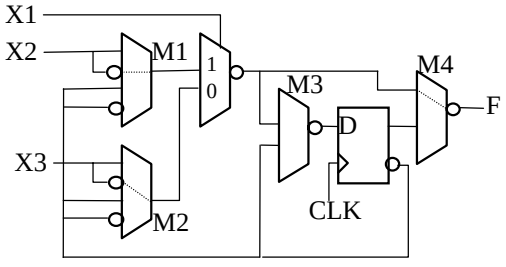

Fig. 6- Unidade funcional da família XC6200, os multiplexadores M1 a M4 são programáveis através da SRAM. Supondo que X1 e X3 estão ligadas a uma entrada lógica A e X2 está ligada a uma entrada lógica B então as linhas a tracejado indicam a programação dos multiplexadores para se obter um NAND lógico de A e B em F.

- Memória distribuída: Cada unidade funcional pode funcionar como função lógica, como memória ou como lógica com memória. O flipflop existente em cada célula pode ser aproveitado para funcionar como registo acessível do exterior através da interface FastMAP[tm]. O desenhador terá uma grande facilidade em adaptar as necessidades de memória do algoritmo à arquitectura da FPGA;
- Arquitectura simétrica: quer quanto à organização das unidades funcionais, quer quanto à distribuição dos recursos de interligação a arquitectura é simétrica. Esta simetria permite a separação entre a configuração que desempenha determinada função e a posição dentro da FPGA em que essa configuração vai ser mapeada. Esta separação tem grande importância para a reconfiguração dinâmica.

Em termos de conectividade com outros componentes a FPGA pode funcionar segundo os níveis lógicos TTL ou CMOS. Existem várias dispositivos pertencentes a esta família que variam na sua capacidade lógica, a tabela 1 apresenta os valores do número de portas lógicas, número de unidades funcionais, número de blocos de entrada/saída (IOBs) e o tempo de reconfiguração total para alguns dos dispositivos desta família.

	XC6209	XC6216	XC6236	XC6264
nº portas	9k-13k	16k-24k	36k-55k	64k-100k
nº U. F.	2304	4096	9216	16384
tempo reconfig.	120µs	200µs	450µs	800µs
nº IOBs	192	256	384	512

Tab. 1 - A família XC6200

III. Implementação de circuitos dinamicamente reconfiguráveis baseados em elementos matriciais

Existem vários tipos de circuitos lógicos para os quais uma implementação usando elementos matriciais tem vantagens. Uma vez que os esquemas matriciais têm uma estrutura bastante regular estão vocacionados para permitir a modificação da sua funcionalidade.

Os elementos de lógica matricial permitem-nos implementar funções lógicas universais, tais como NAND ou NOR. As matrizes lógicas são usadas como blocos funcionais básicos em muitos dispositivos práticos, tais como ROM, RAM, PLA, PAL, etc.

Se considerarmos uma matriz AND (Fig. 7) em que as variáveis de entrada possam ser negadas podemos implementar q produtos $b_1,\ldots,b_q$ de n variáveis $x_1,\ldots,x_n$, sendo $q \leq Q$ e $n \leq N$.

Na figura 7, por exemplo, estão programados os seguintes produtos:

$$b_1 = x_1\overline{x}_2; \qquad b_2 = \overline{x}_1 x_2;$$

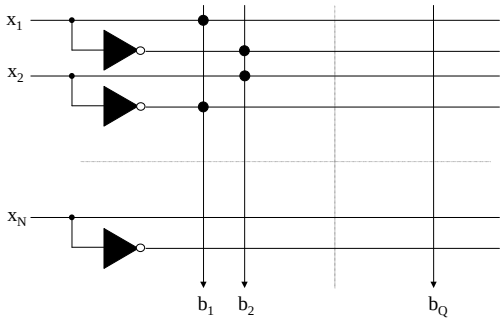

Fig. 7 - Matriz NAND

Na figura 8 encontra-se representada uma matriz OR ou NOR. Esta matriz permite implementar *m* somas booleanas $y_1,\ldots,y_m$ de *q* variáveis $b_1,\ldots,b_q$, sendo $q \le Q$ e $m \le M$.

Por exemplo, na figura 8 encontram-se programadas as seguintes somas:

$$y_1 = b_1 + b_2; \qquad y_2 = \overline{b_1};$$

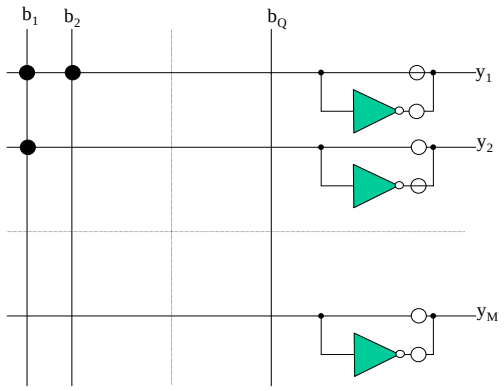

Fig. 8 - Matriz OR/NOR.

Se ligarmos as saídas da matriz AND às entradas da matriz OR/NOR obtemos um dispositivo lógico conhecido como PLA (Fig. 9a).

Na figura 9 estão representadas diferentes combinações de elementos matriciais, eventualmente combinados com outros circuitos lógicos pré-definidos. Estas combinações correspondem a dispositivos usados frequentemente em sistemas digitais, como por exemplo, ROM (Fig. 9b), RAM (Fig. 9c), PAL (Fig. 9d), CPLD (Fig. 9e), sequenciador programável ou PLA com memória (Fig. 9f).

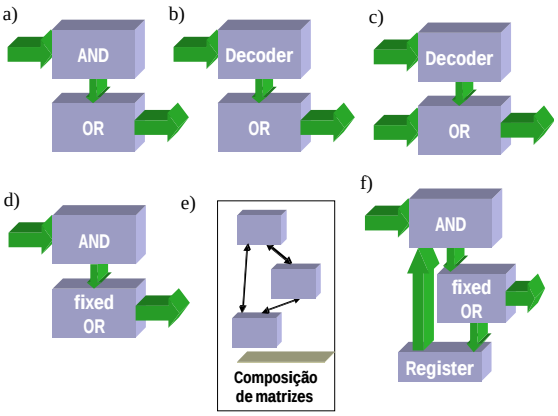

Fig. 9 - Alguns dispositivos baseados em elementos matriciais.

Dado que este tipo de dispositivos são bastante eficientes e úteis no projecto de diferentes sistemas digitais [18-20], pode ser interessante explorar arquitecturas de dispositivos reconfiguráveis dinamicamente (por exemplo, FPGAs), que permitam implementar eficientemente elementos matriciais configuráveis. Nos casos apresentados em [21,22] são propostas arquitecturas optimizadas para circuitos com características específicas: *datapath* [21] e memórias [22].

A. Considerações sobre uma possível arquitectura de FPGA

Apesar da sua extrema flexibilidade, as unidades funcionais simples (do tipo daquela utilizada na família XC6200) não se adaptam bem à implementação de expressões lógicas em que o número de entradas em cada termo é grande. A implementação de expressões deste tipo implica a utilização de muitas unidades funcionais, visto que cada unidade apenas consegue cumprir uma pequena parte da expressão a implementar e implica uma arquitectura de implementação com vários níveis o que aumenta o atraso do circuito.

A existência de unidades funcionais do tipo matricial foi já considerada por alguns fabricantes [12], mas também este tipo de FPGAs, se é verdade que se adaptam muito bem se as expressões lógicas a implementar contiverem um número de termos compatível com o número de entradas das unidades funcionais do tipo matricial, em termos gerais perdem alguma flexibilidade pois neste caso a implementação de uma expressão lógica simples numa unidade funcional com mais entradas do que as necessárias implica um desperdício de recursos.

Uma outra alternativa será a concepção de FPGAs em que coexistam unidades funcionais de diferentes tipos: umas baseadas em *gates* outras baseadas em componentes matriciais. Um problema que se poderá apontar a esta solução é que a capacidade de mobilidade dentro da FPGA de uma configuração que utilize diferentes tipos de unidades funcionais poderá ser substancialmente reduzida, a não ser que a disposição dos diferentes tipos de

unidades mantenha uma estrutura simétrica. Uma configuração que cumpre estas características seria a inclusão entre as colunas de unidades funcionais simples de uma FPGA de uma coluna que pudesse ser configurada como um AND lógico de todas as saídas da coluna anterior (Fig. 10). Usando manipulação lógica estas colunas poderiam também funcionar como portas OR. Esta arquitectura incluiria um número reduzido de unidades matriciais, mas cada unidade funcional manteria a flexibilidade de funcionar como entrada para a unidade matricial ou de ter um funcionamento completamente independente desta.

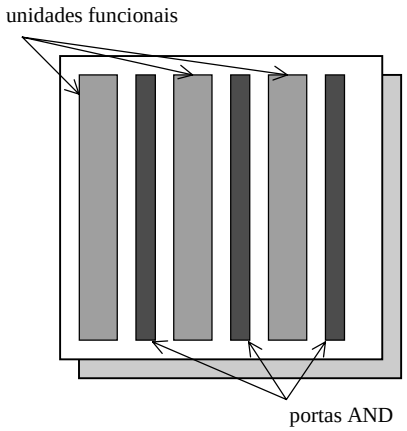

Fig. 10 - Disposição das unidades funcionais simples e das colunas AND que permitirão implementar elementos matriciais.

A ligação entre as saídas das unidades funcionais e a porta lógica que implementaria a unidade matricial poderia ser realizada através de um circuito do tipo do representado na figura 11 em que os ORs poderiam ser internos à unidade funcional.

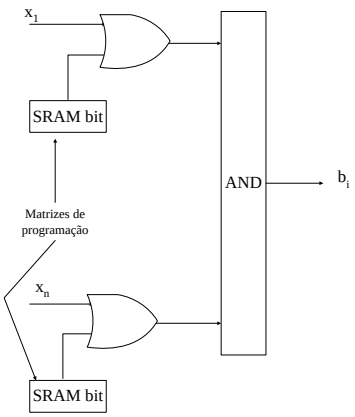

Fig. 11 - Ligação entre as unidades funcionais e a coluna AND.

IV. OBJECTOS DE HARDWARE VIRTUAIS

À medida que a complexidade e os requisitos para um reduzido tempo de desenvolvimento de sistemas e aplicações aumentam, surge também uma necessidade crescente de métodos e ferramentas de projecto avançados. Tal como a Engenharia de Software já provou, a melhor maneira de lidar com a complexidade, bem como aumentar a reutilização e interoperação, é a aplicação de princípios de orientação por objectos na análise, projecto e implementação de uma aplicação.

A reconfiguração dinâmica devido a todos os seus requisitos e potencialidades é, sem dúvida, uma área bastante complexa. Será pois legítimo aplicar princípios de orientação por objectos no desenvolvimento de hardware. A questão que se coloca é se a aplicação dos mesmos deve ser feita tal como no desenvolvimento de software ou, por outro lado, a abordagem deverá ser adaptada. Nos próximos parágrafos, irá ser justificada a aplicação destes conceitos, assim como as restrições que daí advêm.

Como já foi dito atrás, a utilização de mecanismos de reprogramação e reconfiguração dinâmica no projecto de sistemas digitais pode ter dois grandes campos de aplicação:

- Implementação de circuitos de grande dimensão e complexidade, desde que seja possível a sua decomposição em blocos de menor complexidade e de execução disjunta no tempo, podemos pensar numa solução em que somente os blocos activos se encontram implementados na FPGA;
- Coprocessadores destinados ao cálculo de algoritmos intensivos de software. Cada algoritmo deve ser descrito por um bloco a implementar na FPGA.

Em qualquer dos casos estamos na presença de um sistema que deve possuir capacidades de reserva e libertação dinâmica de recursos. Sempre que um dado bloco/algoritmo tiver de ser executado mas ainda não estiver residente na FPGA, deve ser feita a sua transferência da memória externa para a memória da FPGA (Fig. 12).

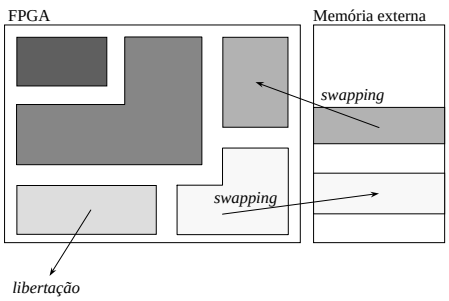

Fig. 12 - Transferência de blocos executáveis entre a FPGA e a memória externa

Por outro lado, tal como já foi mencionado atrás, sempre que for necessário executar um novo bloco e a FPGA não possuir os recursos necessários para o fazer, deve-se, se possível, seleccionar um determinado bloco que já não

seja necessário, ou que esteja temporariamente suspenso e efectuar o seu *swapping* para a memória externa, conseguindo assim obter os recursos para executar o novo bloco.

A. Aplicação de programação por objectos

Por forma a simplificar e sistematizar o desenvolvimento de aplicações, os mecanismos atrás descritos, bem como a forma de descrever os blocos individualmente, devem ser devidamente estudados, com vista a obter uma representação formal. Assim, para atingir estes objectivos e dada a natureza do problema, a abordagem que melhor se aplica deve-se basear numa orientação por objectos.

Os vários blocos destinados a implementar um dado circuito/algoritmo correspondem aos vários objectos que comunicam entre si, trocando informação que, do ponto de vista de objectos corresponde ao envio de mensagens (Fig. 13). Do ponto de vista físico, estas mensagens correspondem a sinais existentes nas linhas de controlo e de dados que ligam os vários objectos. No caso de um objecto de hardware ser uma implementação de um algoritmo de software (com objectivo de acelerar o processamento), a troca de mensagens também pode ser efectuada entre as componentes de software e hardware da aplicação.

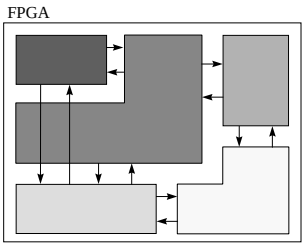

Fig. 13 - Troca de mensagens entre os objectos

Sempre que for necessário fazer o *swapping* de um dado bloco, deve-se transferir para/da memória os dados contidos nos registos (de notar que a configuração já se deve encontrar na memória externa, não sendo portanto necessário efectuar a sua salvaguarda).

Dependendo da natureza da memória externa, poderá também estar implementada a persistência de um dado objecto.

O processo de construção de um objecto consiste no carregamento de uma *stream* de bits na memória da FPGA. Por outro lado, a destruição consiste em libertar o espaço ocupado por esse objecto na FPGA, tendo o cuidado de suspender as ligações entre este e outros objectos com os quais este comunica (Fig. 14). Para este efeito, além da lógica necessária à implementação de um dado objecto, deve também ser implementada alguma lógica adicional cuja função é isolar os objectos que interagem com aquele que vai ser substituído. Este procedimento é importante porque durante a reconfiguração de um bloco da FPGA o seu

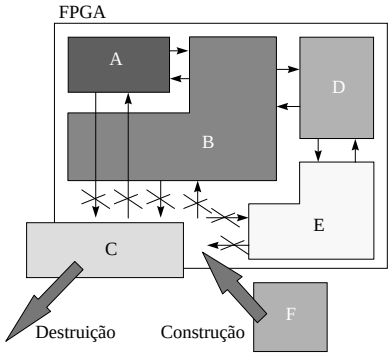

Fig. 14 - Construção/Destruição de um objecto de hardware.

comportamento pode ser considerado aleatório.

Um bloco que implementa um dado circuito será designado como um objecto de hardware virtual (OHV). Este é definido como uma implementação de um algoritmo que pode ser reconfigurado dinamicamente. Por isso, estes objectos podem ser utilizados em sistemas reconfiguráveis (FPGAs).

Um aspecto importante na definição de um OHV prende-se com a complexidade do mesmo. A utilização desta metodologia de desenvolvimento tem como um dos objectivos a manipulação de circuitos com alguma complexidade. Não é nossa intenção, pelo menos neste momento, aplicar estes conceitos na descrição de circuitos simples, para os quais já há técnicas de projecto, optimização e síntese bem conhecidas.

Uma possibilidade é considerar um OHV como sendo constituído por duas grandes componentes, normalmente conhecidas por *datapath* e *control path*. Ou seja, um OHV deve possuir capacidade de processamento interno, servindo as mensagens para trocar informação com outros objectos.

Vamos agora apresentar em mais pormenor a constituição interna de um OHV.

Este contém um *datapath* constituído por:

- Componentes usados no armazenamento de valores: registos, contadores, etc;
- Blocos funcionais: ALU, somador, multiplicador, etc;
- Elementos de ligação: bus, multiplexador, etc.

Deve possuir também uma unidade de controlo – *control path* – que determina as suas funcionalidades e o seu comportamento. Esta deve ser descrita por um conjunto de grafos hierárquicos (HGSs) [8] que descrevem o algoritmo de funcionamento do circuito (sequência de operações) e possibilitam a utilização de hierarquia e paralelismo.

A implementação de um objecto deste tipo consiste na interligação de todos estes componentes destinados a executar um dado algoritmo.

O objecto representado na figura 15 é composto por um dado circuito (*datapath*) onde são executados 2

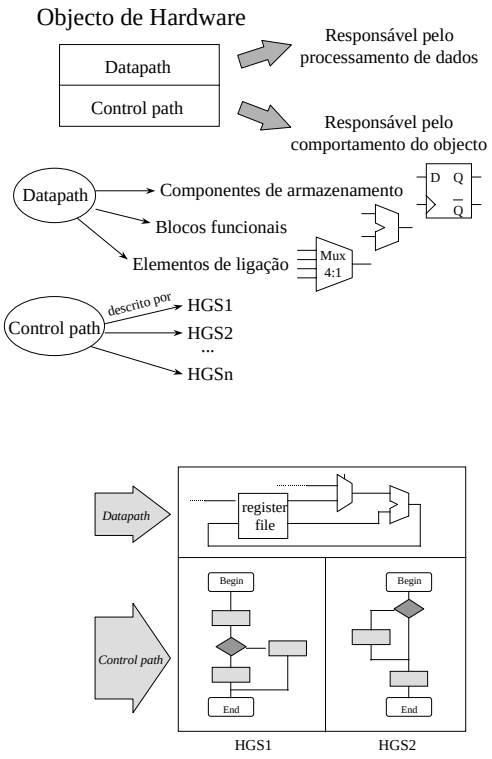

Fig. 15 - Constituição interna de um OHV

algoritmos descritos pelos HGS1 e HGS2 que determinam o seu comportamento (*control path*).

B. Flexibilidade, Extensibilidade e Reutilização de Objectos

Da utilização da metodologia de desenvolvimento orientada por objectos podem resultar várias vantagens, das quais se destacam:

- Flexibilidade
- Extensibilidade
- Reutilização

A flexibilidade e a extensibilidade de um objecto traduzem-se na possibilidade de alteração quer do *datapath* quer do *control path*.

B.I Modificação do Control Path

A alteração do *control path* pode ser determinada pela alteração, adição ou eliminação de pelo menos um HGS. Se não se pretender alterar a estrutura das mensagens entre objectos, as alterações no *control path* não devem alterar o interface físico do mesmo, isto é, devem apenas afectar a sua implementação interna, bem como o conteúdo da informação contida nas mensagens. As ligações físicas estabelecidas com outros objectos podem assim permanecer inalteradas.

Num caso mais geral e sempre que necessário ou desejável, a alteração da unidade de controlo pode também provocar alterações no interface de um objecto.

B.II Modificação do Datapath

Tal como foi dito atrás, o *datapath* é normalmente constituído pelos seguintes 3 tipos de componentes – armazenamento, funcionais e ligação/encaminhamento.

A reconfiguração do datapath pode ser efectuada por alteração de qualquer um dos componentes referidos. De notar que a alteração do datapath conduz geralmente a alterações da unidade de controlo.

- Modificação da estrutura - Componentes de armazenamento

Sempre que necessário podemos alterar o número de componentes de armazenamento. Esta alteração pode ser provocada por uma diferente implementação de um dado objecto ou por reutilização do mesmo. Ex: alteração do número de registos num *datapath*.

- Modificação do conjunto de funções - Componentes funcionais

A forma como determinadas funções são implementadas internamente pode ser alterada, bem como o conjunto das mesmas. Isto pode ser provocado por modificação na interpretação da informação contida nas mensagens, por questões de optimização de processamento ou ainda por alteração do número de funções executadas por um dado elemento (reutilização). Estes factos traduzem-se na reprogramação dos blocos funcionais. Ex: tipo de operações executadas por uma ALU.

- Modificação da configuração - Elementos de ligação/encaminhamento

Se por um lado o número de componentes de armazenamento pode ser alterado e por outro a forma como as funções estão implementadas também o pode, é possível que seja também necessário alterar a configuração dos elementos que interligam todos os blocos internos do OHV. Ex: estrutura do bus que interliga os vários componentes do objecto.

De um modo geral, é possível modificar a estrutura interna de um objecto com ou sem alteração do seu interface. A ideia principal é reutilizar, bem como expandir as funcionalidades incorporadas num dado objecto.

C. Reutilização de Objectos

Da aplicação destas técnicas resulta também a possibilidade de reutilização de OHVs. Esta pode ser feita de duas formas:

- Interligação de objectos já existentes e devidamente testados por forma a tornar mais rápido o projecto de um novo sistema. A interligação de vários OHVs pode resultar num objecto com maior funcionalidade

que depois de devidamente testado pode também ele ser reutilizado. Este procedimento corresponde à inclusão de vários objectos noutro. Ex: registos, ALU e unidade de controlo são exemplos de objectos contidos num processador;

- Adição, eliminação e redefinição de novas funcionalidades a OHVs já existentes, por forma a obter um objecto com as funcionalidades pretendidas. A adição e redefinição estão directamente relacionadas com a derivação de classes de objectos a partir de classes já existentes. Por exemplo, se o objecto P (Fig. 16) for um processador e o objecto PD for um processador digital de sinal, estes terão funcionalidades em comum. Possuindo o objecto P funcionalidades mais gerais, a teoria da orientação por objectos sugere que PD seja derivado de P, pois PD é também um processador. Este possuirá as funcionalidades de P e de PD.

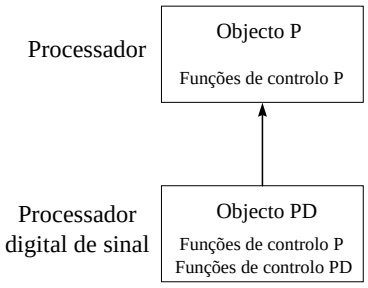

Fig. 16 - Especialização de objectos

PD é um objecto especializado, com operações adicionais que pode ser criado a partir de P.

Assim, pode-se verificar que os objectos de hardware possuem uma parte de controlo extensível (PD adiciona funcionalidades a P), reutilizável (PD usufrui das funcionalidades de P) e flexível devido à reconfiguração dinâmica, isto é, à possibilidade de alteração da parte de controlo, como já foi descrito atrás.

Torna-se pois extremamente útil a construção de uma biblioteca de OHVs predefinidos, de forma a que no futuro estes blocos possam ser utilizados na construção de novos sistemas de forma mais rápida, económica e fiável.

A expansão desta biblioteca é também bastante útil, podendo armazenar novos OHVs, bem como OHVs resultantes da expansão / modificação das funcionalidades já existentes.

D. Implementação Prática de OHVs

Na prática, a aplicação destes conceitos pode defrontar-se com algumas dificuldades. A principal prende-se exactamente com a inexistência de uma linguagem de especificação e simulação de hardware orientada por objectos suficientemente divulgada. A generalidade dos fabricantes baseia as suas ferramentas de desenvolvimento em dois grandes princípios:

- Captura de esquema
- Linguagem de descrição de hardware (VHDL/Verilog)

Como estas linguagens não são orientadas por objectos, esta metodologia só pode ser aplicada nas fases de análise e projecto de um sistema. Na implementação deve-se converter as especificações dos objectos para um formato suportado na ferramenta a utilizar. Apesar de todos estes problemas, a linguagem VHDL permite que esta conversão se faça com alguma facilidade. Características como a agregação e o encapsulamento já estão incorporadas na linguagem. A característica de herança existente em linguagens orientadas por objectos não é suportada. Devido a estes factos, pode-se considerar o VHDL uma linguagem baseada em objectos em vez de orientada por objectos.

Só para ilustrar a possibilidade de adaptar a característica de herança em VHDL, apresentamos a seguir um exemplo muito simples (Fig. 17). Inicialmente é construído um flip-flop tipo D. O passo seguinte foi acrescentar a este flip-flop uma entrada de Clear. De notar que o exemplo apresentado contraria a definição de OHV apresentada atrás (um OHV deve ser constituído por *datapath* e *control path*). No entanto, optamos por este exemplo por uma questão de simplicidade e clareza.

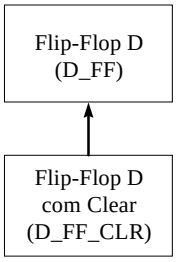

Fig. 17 - Diagrama de classes das entidades representadas

A listagem de VHDL utilizada foi a seguinte:

```
-- ENTIDADE FLIP FLOP TIPO D
entity D_FF is
  port(D, CLK : in bit;
       Q : out bit);
end D_FF;

architecture behav of D_FF is
begin
  process(CLK)
  begin
    if CLK='1' then
      Q <= D;
    end if;
  end process;
end behav;
```

```
-- ENTIDADE FLIP-FLOP TIPO D COM CLEAR
entity D_FF_CLR is
  port(D_2, CLK_2, CLR : in bit;
       Q_2 : out bit);
end D_FF_CLR;

architecture behav of D_FF_CLR is
  signal TEMP : bit;
begin
  FF_BASE: entity work.D_FF(behav)
    port map(D=>D_2, CLK=>CLK_2, Q=>TEMP);

  process(CLK_2, CLR, TEMP)
  begin
    if CLR='1' then
      Q_2 <= '0';
    else
      Q_2 <= TEMP;
    end if;
  end process;
end behav;
```

A título meramente ilustrativo a figura 18 mostra o resultado da simulação da entidade D_FF_CLR.

O processo utilizado não é completamente isento de erros. Quem implementar uma dada descrição tem de fazer algum esforço adicional devido à falta de suporte da programação por objectos em VHDL. Com o objectivo de solucionar este problema, existe já um grupo de trabalho responsável por criar um conjunto de extensões à linguagem VHDL, bem como a sua standardização, destinadas a tornar esta linguagem compatível com as metodologias de implementação por objectos [23]. Existe também outra linguagem de descrição de hardware intitulada "LOLA" possuindo já suporte para a descrição de objectos. No entanto, as ferramentas de optimização e síntese de circuitos digitais ainda não possuem suporte para estas linguagens, podendo demorar algum tempo até isso se verificar.

Quando se pretende testar OHVs incluídos numa aplicação com uma componente de hardware e outra de software, a componente de hardware pode ser encapsulada numa *test bench* e testada num simulador de VHDL. Escrever uma *test bench* pode ser uma tarefa que leve algum tempo a executar (uma vez que deve simular o funcionamento normal do sistema). Com o objectivo de ultrapassar este problema, a Model Technology introduziu no seu simulador de VHDL – V-System uma característica inovadora – FLI (*Foreign Language Interface*). Esta facilidade permite o acesso a uma simulação através de um programa escrito em C/C++. Isto permite que os testes da componente de hardware sejam efectuados através de um programa que servirá de base à componente de software da aplicação em causa. É assim possível testar completamente uma aplicação em que a componente de hardware é constituída por OHVs e a componente de software descrita por uma linguagem orientada por objectos (C++) [24].

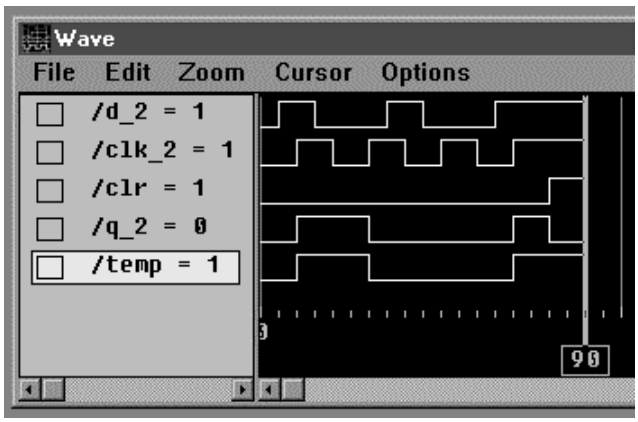

Fig. 18 - Simulação da entidade D_FF_CLR

V. Conclusões

Os OHVs descritos neste artigo são uma representação eficiente da implementação em hardware de blocos funcionais/algoritmos. As principais vantagens que decorrem da sua utilização são:

- Manipulação mais simples e eficiente da complexidade de um sistema;
- Aceleração de algoritmos normalmente executados em software;
- Reserva / Libertação dinâmica de blocos pertencentes a um circuito complexo.

As possibilidades de aplicação destes conceitos são inúmeras. Áreas tão diversas como as telecomunicações, processamento de sinal e a visualização científica têm muito a ganhar com a utilização desta tecnologia.

Para além de um melhor aproveitamento dos recursos do sistema, estamos também na presença de uma solução que diminui drasticamente o tempo e os custos de desenvolvimento de uma dada aplicação, comparativamente ao que se passa com o desenvolvimento de um circuito integrado específico (ASIC) para a aplicação em causa.

Outra vantagem é a possibilidade de resposta rápida a alterações de mercado impostas pelos clientes ou pela evolução dos standards.

Do ponto de vista económico, as vantagens tornam-se evidentes se a aplicação tirar o máximo partido da reconfiguração dinâmica. Além disso, estamos na presença de uma tecnologia relativamente recente, mas em franco crescimento. Tudo leva a crer que a sua utilização se irá generalizar, logo, tornando-a competitiva com as tecnologias já existentes.

VI. Referências

[1] Giovanni De Micheli, "*Synthesis and Optimization of Digital Circuits*", McGraw-Hill, Inc., 1994.

[2] Grady Booch, "*Object-Oriented Analysis and Design*", Second Edition, The Benjamin/Cummings Publishing Company, Inc., 1994.

[3] Stephen Edwards, Luciano Lavagno, Edward A.Lee, Alberto Sangiovanny-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis", *Proceeding of the IEEE*, vol. 85, no. 3, pp. 366-390, March, 1997.

[4] Maxfield, Clive "Max", "Field Programmable Devices," *EDN*, pp. 201-206, October, 1996.

[5] Xilinx, "XC6200 Field Programmable Gate Arrays", Product Description, (http://www.xilinx.com/partinfo/6200.pdf), April, 1997.

[6] Patric Lysaght, Jon Stockwood, "A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays", *IEEE Trans. On VLSI Syst*., vol. 4, no. 3, pp. 381-390, September, 1996.

[7] John H.Mayer, "Reconfigurable Computing Redefines Design Flexibility", *Computer Design*, pp. 49-52, February, 1997.

[8] Valery Sklyarov, António Adrego da Rocha. "Sintese de Unidades de Controlo Descritas por Grafos dum Esquema Hierarquicos", *Electrónica e Telecomunicações*, vol. 1, no. 6, 577-588, 1996.

[9] António Adrego da Rocha, Valery Sklyarov, "Simulação em VHDL de Máquinas de Estados Finitas Hierárquicas", *Electrónica e Telecomunicações*, Vol. 2, N 1, 83-94, 1997.

[10] G.Bostock, "*FPGAs and Programmable LSI. A designer's handbook*". Butterworth-Heinemann, 1996.

[11] S.Brown, "FPGA Architectural Research: A Survey", *IEEE Design & Test of Computers*, Vol. 13, No 4, 9-15, 1996.

[12] Stephen Brown, Robert Francis, J. Rose and Z.Vranesic, "*Field-programmable Gate Arrays*", Kluwer Academic Publishers, 1992.

[13] Bradly Fawcett, J. Watson, "Reconfigurable Processing with Field Programmable Gate Arrays", In *Proc. International Conference on Application-Specific System, Architectures and Processors*, Chicago, 293-302, 1996.

[14] P. Bertin, et al, "Programmable Active Memories: a Performance Assessment", *First International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1992.

[15] M. Gokhale, et al, "Building and Using a Highly Parallel Programmable Logic Array", *IEEE Computer*, Vol. 24, N 1, 81-89, 1991.

[16] J. Arnold, et al, "SPLASH 2", *Proceedings of the 4th Annual ACM Synposium on Parallel Algorithms and Architectures*, 1992.

[17] Jean Vuillemin, et al, "Programmable Active Memories: Reconfigurable Systems Come of Age", *IEEE Trans. on VLSI Systems*, Vol 4, N 1, 56-82, 1996.

[18] Valery Sklyarov, "Applying Finite State Machine Theory and Object-Oriented Programming to the Logical Synthesis of Control Devices". *Electrónica e Telecomunicações*, Vol. 1, N 6, 515-529, 1996.

[19] Antonio Adrego da Rocha, Valery Sklyarov, Antonio Ferrari, "Hierarchical Description and Design of Control Circuits Based on Reconfigurable and Reprogrammable Elements", *Proceeding of the International Workshop on Logic and Architectural Synthesis IWLAS'97*, Grenoble, 1997.

[20] Valery Sklyarov, Antonio Adrego da Rocha, Antonio de Brito Ferrari, "Applying Procedural and Object-Oriented Decomposition to the Logical Synthesis of Digital Devices", *Proceeding of the second International Conference on Computer-Aided Design of Discrete Devices - CAD DD'97*, Minsk, 1997.

[21] D.Chrepacha, D.Lewis, "DP-FPGA: An FPGA Architecture Optimised for Datapath", *VLSI Design*, Vol. 4, No 4, 329-343, 1996.

[22] S.Wilton, J.Rose, Z.Vranesic, "Architecture of Centralized Field-Configurable Memory", *Proc. Third ACM Int'l Symp. On Field Programmable Gate Arrays*, Assoc. For Computing Machinery, New York, 97-103, 1995.

[23] P. J. Ashenden and P. A. Wilsey, "Considerations on Object-Oriented Extensions to VHDL", *Proc. VIUF Spring'97 Conference*, April 1997.

[24] Xilinx Corp., "Co-simulation of Hardware and Software", XApp. 087, (http://www.xilinx.com/xapp/xapp087.pdf), July 1997.