Especificação, Projecto e Implementação de Circuitos de Controlo Virtuais

Arnaldo Oliveira, Valery Sklyarov

***Resumo* – Este artigo descreve técnicas para especificação, projecto e implementação de circuitos de controlo virtuais. Tal como os circuitos de controlo ordinários, são normalmente implementados usando Máquinas de Estados Finitos. No entanto, através da sua reconfiguração, estes circuitos utilizam o mesmo *hardware* para executar diferentes partes de um algoritmo de controlo. O método de especificação adoptado baseia-se em Esquemas Gráficos Hierárquicos (*Hierarchical Graph-Schemes – HGSs*), com algumas extensões propostas pelos autores. Devido à arquitectura do dispositivo base utilizado, a técnica de codificação de estados *one-hot* é a mais indicada. Em algumas aplicações, a reconfiguração deve ser realizada sem interromper o funcionamento do circuito, o que implica a utilização de dispositivos reconfiguráveis dinamicamente. As duas implementações apresentadas utilizam uma FPGA reconfigurável dinamicamente da família XC6200 da Xilinx. É também proposta uma arquitectura optimizada, para implementar em VLSI unidades de controlo reconfiguráveis.**

***Abstract* – This paper describes specification, design and implementation techniques of virtual control circuits. Like the ordinary control circuits, they are usually implemented using Finite State Machines (FSMs). However, through reconfiguration, these circuits use the same hardware to implement different parts of a control algorithm. The adopted specification method is based on Hierarchical Graph-Schemes (HGSs) with some extensions proposed by the authors. Due to the architecture of the base device used, the one-hot state encoding technique is the most appropriate. In some applications, the reconfiguration should be done without stop the operation of the circuit, which implies the use of dynamically reconfigurable devices. The described implementations use a Xilinx XC6216 dynamically reconfigurable FPGA. An optimized architecture to implement reconfigurable control units in VLSI is proposed.**

I. INTRODUCÃO

Os sistemas digitais que efectuam processamento de dados são normalmente constituídos por duas componentes: unidade de execução (*datapath*) e unidade de controlo (*controlpath*). A unidade de execução contém geralmente registos para armazenamento de dados e unidades funcionais, tais como unidades aritméticas e lógicas e registos de deslocamento, que efectuam operações sobre esses dados. Por outro lado, a unidade de controlo garante a sequência correcta de operações na unidade de execução e é normalmente implementada usando máquinas de estados finitos (*Finite State Machines – FSM*). O projecto de unidades de execução é normalmente hierárquico, existindo bibliotecas de componentes já testados e optimizados, simplificando assim o processo de desenvolvimento. No entanto, tal já não acontece no caso das unidades de controlo, que normalmente são específicas de cada projecto. Contrariamente ao que se passa no caso das unidades de execução, são bastante irregulares não sendo possível estabelecer nenhum padrão. Devido a estes factos tem sido dada bastante atenção à síntese automática com base na sua descrição comportamental.

Tradicionalmente, a especificação de unidades de controlo é efectuada recorrendo a diagramas de estado, sendo neste caso o processo de desenvolvimento normalmente dividido nos seguintes passos:

- Escolha da implementação (Moore, Mealy, etc.);
- Construção do diagrama de estados;
- Conversão do diagrama numa tabela de transição de estados;
- Codificação de estados;
- Optimização da lógica combinatória;
- Desenho do circuito.

A técnica de especificação adoptada é baseada em Esquemas Gráficos Hierárquicos (*Hierarchical Graph-Schemes - HGSs*) [1]. Para além de proporcionarem uma representação natural dos algoritmos de controlo (notação semelhante a fluxogramas) têm relativamente aos diagramas de transição de estados as seguintes vantagens:

- Especificação independente da implementação do circuito;
- Suportam hierarquia, permitindo a decomposição *top-down* do algoritmo de controlo, sendo portanto uma notação conveniente para especificar o comportamento de máquinas de estado finitos hierárquicas (*Hierarchical Finite State Machines – HFSMs*).

Apesar do processo de desenvolvimento poder seguir, com algumas adaptações, os mesmos passos que o caso anterior, como veremos mais à frente, se for utilizada a técnica de codificação de estados *one-hot*, juntamente com uma arquitectura apropriada, a conversão de um HGS no respectivo circuito é directa, tornando desnecessária a maioria dos passos referidos. A utilização de máquinas de estados finitos hierárquicas permite um projecto mais estruturado de unidades de controlo do que as abordagens tradicionais. Nesta metodologia, um

algoritmo de controlo é dividido em partes relativamente independentes (sub-algoritmos). Durante a execução do algoritmo de controlo, os vários sub-algoritmos são invocados de acordo com o fluxo do mesmo. Esta técnica é muito semelhante ao desenvolvimento de *software*. Para além da diminuição da complexidade de desenvolvimento, já que cada sub-algoritmo pode ser individualmente projectado e testado, pode-se também em certas circunstâncias reutilizar sub-algoritmos noutros projectos. Esta abordagem é ainda mais poderosa se os sub-algoritmos puderem ser carregados em *hardware* somente quando necessários. Desta forma consegue-se uma economia de *hardware* e consequentemente uma diminuição do custo e do consumo do circuito. Para tal, torna-se necessária a utilização de dispositivos dinamicamente reconfiguráveis. As implementações realizadas utilizam como dispositivo base, uma FPGA dinamicamente reconfigurável XC6216 [2] da Xilinx. Em algumas aplicações é importante tornar a unidade de controlo flexível e extensível sendo por vezes necessário alterar o seu comportamento sem interromper a sua operação. A reconfiguração dinâmica permite alcançar estes objectivos, uma vez que a alteração do comportamento pode ser realizada através da actualização dos sub-algoritmos pretendidos, sem que seja necessário interromper o seu funcionamento.

Este artigo está dividido em oito secções. Na secção II é descrito sumariamente o método de especificação de HFSMs adoptado. Na secção III é apresentada uma implementação de uma HFSM em FPGA, que permite sem reconfigurar o circuito, realizar o *binding* dinâmico de sub-algoritmos. A implementação descrita na secção IV utiliza reconfiguração dinâmica para permitir a síntese independente de cada sub-algoritmo e o seu carregamento em hardware quando necessário. Na secção V é apresentada uma arquitectura (*VICON – Virtual Controller Architecture*), baseada na XC6200 [2] da Xilinx, para implementação de unidades de controlo reconfiguráveis. Na secção VI é descrito um método que permite a implementação de estados temporizados em HFSMs. Na secção VII é apresentada uma biblioteca, escrita em VHDL, de componentes parametrizáveis para a família XC6200. Finalmente, as conclusões encontram-se na secção VIII.

II. ESPECIFICAÇÃO DE HFSMS

A especificação de HFSMs pode ser realizada ao nível comportamental usando Esquemas Gráficos Hierárquicos (*Hierarchical Graph-Schemes – HGSs*) [1, 3]. Os HGSs são bastante úteis no projecto de unidades de controlo porque fornecem uma descrição formal do comportamento que é independente da implementação do circuito. Esta notação é particularmente útil para descrever algoritmos de controlo, nos quais quer as entradas quer as saídas são constituídas por sinais de um bit. Um HGS deve possuir pelo menos um grafo, denominado grafo principal, que implementa o núcleo principal do algoritmo de controlo. Opcionalmente, pode possuir outros grafos que implementam sub-algoritmos. Estes sub-algoritmos podem ser divididos em dois tipos: macro-operações e funções lógicas. As macro-operações não retornam qualquer resultado ao grafo invocador, sendo portanto análogas a procedimentos. Por outro lado, as funções lógicas retornam um valor ao grafo invocador. Cada grafo é constituído por dois tipos de nodos: rectangulares e losangulares. A execução é iniciada num nodo rectangular (*Begin*) e termina outro nodo rectangular (*End*). Os restantes nodos rectangulares, também chamados nodos operacionais, contêm um subconjunto de micro-operações do conjunto $Y=\{y_1, y_2, \ldots, y_M\}$ e/ou um sub-conjunto de macro-operações do conjunto $Z=\{z_1, z_2, \ldots, z_N\}$. Uma micro-operação é um sinal de saída da unidade de controlo. Uma macro-operação é descrita por um HGS de nível mais baixo. No caso de um nodo conter mais do que um elemento do conjunto Z estamos na presença de macro-operações paralelas. A presente discussão assume a existência de somente uma macro-operação em cada nodo, ou seja, estamos a considerar apenas processos sequenciais (não paralelos). Os nodos losangulares, também chamados condicionais possuem apenas um elemento do conjunto $C=X \cup F$, onde $X=\{x_1, x_2, \ldots, x_i\}$ é o conjunto das condições lógicas e $F=\{f_1, f_2, \ldots, f_j\}$ é o conjunto das funções lógicas. Uma condição lógica é um sinal de entrada da unidade de controlo. Uma função lógica é calculada, executando alguns passos

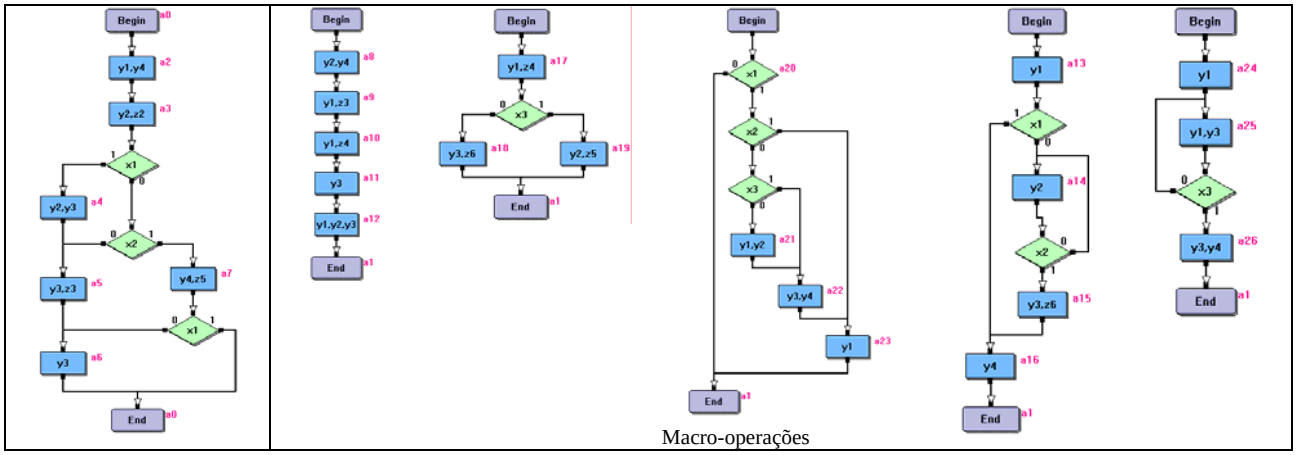

Figura 1 - Exemplo de um HGS.

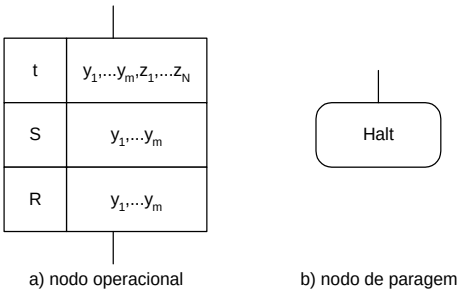

Figura 2 - Extensões propostas aos HGSs

sequenciais pré-definidos, descritos por um GS de nível mais baixo. A figura 1 mostra um exemplo de um HGS, desenhado com um editor gráfico desenvolvido no grupo [4]. Este exemplo é constituído por um grafo principal e cinco macro-operações. O algoritmo de controlo é iniciado no grafo principal, sendo os restantes grafos invocados ao longo da sua execução.

A experiência obtida no desenho de unidades de controlo usando HGSs, leva-nos a propor algumas extensões por forma a aumentar as potencialidades desta notação. Estas extensões estão representadas na figura 2. Os nodos operacionais passam a ser constituídos por três campos:

- O primeiro é análogo à representação tradicional, mas suporta a especificação da duração das micro-operações. O caracter *t* deve ser substituído pela duração em ciclos de relógio. Na ausência de valor assume-se a duração de um ciclo de relógio.
- O segundo e terceiro campos permitem especificar as micro-operações que se pretendem activar e desactivar respectivamente. O seu valor permanece inalterado até surgir outro nodo que force outros valores.

Qualquer um destes campos é opcional, devendo no entanto existir sempre pelo menos um.

A segunda extensão proposta é a adição de um novo nodo rectangular (*Halt*), que representa uma paragem da máquina de estados. De notar que o nodo *End* significa o fim de uma macro-operação, ou do algoritmo, não implicando uma paragem da FSM.

III. IMPLEMENTAÇÃO DE HFSMS EM FPGAS

A estrutura inicial da HFSM utilizada para implementar

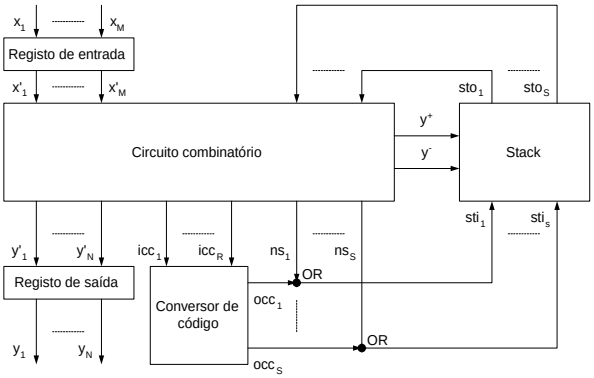

Figura 3 – Estrutura base da HFSM.

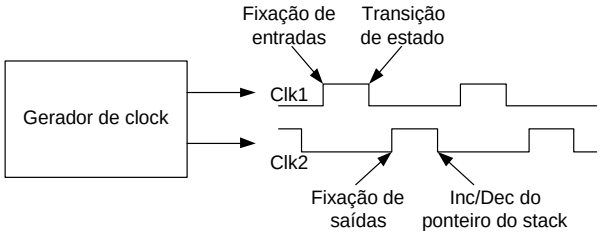

Figura 4 - Sinais de sincronização.

um HGS está representada na figura 3. Neste esquema, o *stack* armazena os estados da HFSM. Para que se possa variar o número de estados e/ou níveis hierárquicos o stack deve ser parameterizável. O conversor de código transforma os nomes virtuais dos sub-algoritmos no estado dos seus nodos iniciais. Assim, o estado seguinte é fornecido pelo circuito combinatório ou pelo conversor de código nos casos de transições ordinárias ou hierárquicas respectivamente. O registo de saída assegura a duração correcta dos sinais de saída. O registo de entrada é utilizado para amostrar as entradas no instante desejado.

Uma desvantagem das HFSMs é a necessidade de uma sincronização mais complexa do que as FSMs ordinárias. A figura 4 mostra um possível conjunto de sinais de sincronização e os seus eventos associados:

- Fixação de entradas;
- Transição de estados;
- Fixação de saídas;
- Incremento / decremento do *stack pointer* (no caso de transições hierárquicas).

A HFSM especificada pelo HGS da figura 1, foi implementada numa FPGA XC6216 da Xilinx. Para o efeito, foi utilizada uma placa de desenvolvimento distribuída pela *Annapolis Micro Systems (Firefly Development System)* Esta placa possui interface PCI destinando-se a ligar a um PC. Os seus componentes principais são (figura 5):

- Uma FPGA dinamicamente reconfigurável XC6216 da Xilinx;
- Uma FPGA XC4013 para interface com o bus PCI;
- 512 Kbytes de SRAM local;
- Compnentes para geração de *clock* e controlo de consumo.

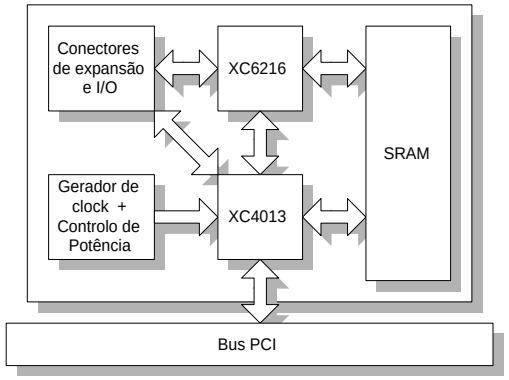

Figura 5 – Componentes do sistema de desenvolvimento FireFly.

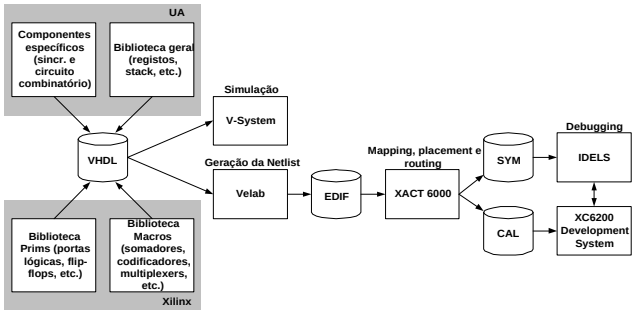

Figura 6 – Sequência de tarefas realizadas na implementação da HFSM.

Para optimizar a utilização de recursos da FPGA, o esquema base da figura 3 foi adaptado ao sistema de desenvolvimento utilizado. Assim, o circuito combinatório, a sincronização e os registos de entrada/saída foram implementados na FPGA, enquanto que o stack e o conversor de código foram implementados na SRAM local.

Os passos executados para implementar a HFSM estão representados na figura 6. Tendo por base as bibliotecas VHDL, PRIMS e MACROS fornecidas pela Xilinx, foram desenvolvidos alguns componentes específicos (circuitos de sincronização e combinatório) e também uma biblioteca com componentes gerais (registos, contadores, etc.). Estes componentes foram testados separadamente recorrendo ao compilador/simulador de VHDL VSystem da Model Technology. Após o desenvolvimento de todos os componentes necessários à implementação, as suas descrições VHDL foram convertidas numa netlist EDIF com o VELAB (VHDL Elaborator) [5]. Esta netlist é utilizada pelo XACT6000 [6] para realizar o mapping, placement e routing do circuito na FPGA. Esta ferramenta pode produzir dois tipos de ficheiros: *.CAL contém a configuração da FPGA; *.SYM contém a localização e informação sobre os componentes do circuito. Estes ficheiros também podem ser utlizados por programas de debugging, tais como *Graph Builder* [4] ou IDELS [7]. A disposição dos componentes na FPGA está representada na figura 7.

Esta implementação permite alterar dinamicamente as ligações entre os GSs através da modificação do conteúdo do conversor de código. No entanto, todos os sub-algoritmos devem estar previamente implementados em *hardware*. As técnicas de marcação e codificação de estados utilizadas tornam a reconfiguração dinâmica do circuito completamente impossível. Para além disso, a multiplexagem no acesso à memória entre o stack e o conversor de código, é responsável por um baixo desempenho do circuito.

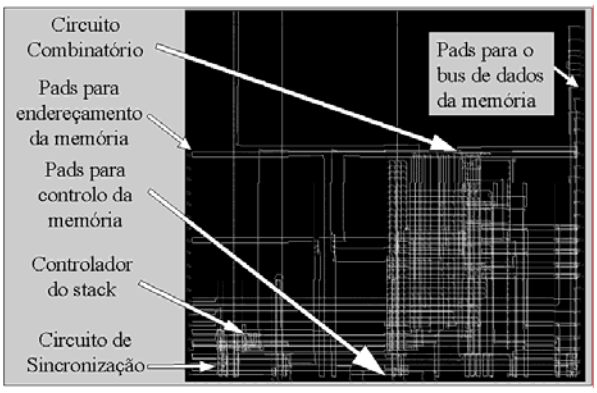

Figura 7 - Layout do circuito.

IV. IMPLEMENTAÇÃO RECONFIGURÁVEL DINAMICAMENTE

Para ultrapassar as limitações da implementação anterior, estamos a explorar a utilização da reconfiguração dinâmica no projecto de unidades de controlo. Esta abordagem permite a construção de circuitos de controlo virtuais, que permitem multiplexar no tempo a utilização do mesmo *hardware* para executar diferentes partes de um algoritmo de controlo (sub-algoritmos). A figura 8 ilustra a estrutura de alto nível da implementação realizada. Os seus componentes principais são:

- Blocos reconfiguráveis – cada bloco suporta a configuração de um sub-algoritmo de controlo (o número de blocos depende da aplicação e das dimensões do dispositivo utilizado);
- Lógica de mapeamento – realiza o mapeamento do estado do grafo invocador no código do próximo grafo a ser executado, verificando se este se encontra carregado num dos blocos;
- Stack – armazena o código dos grafos interrompidos e dos respectivos estados.
- Sincronização – assegura a sequência correcta dos sinais de controlo.

Todos os blocos reconfiguráveis têm acesso às entradas e saídas da unidade de controlo, no entanto, somente o bloco activo lê as entradas e controla as saídas. Durante transições de estado pertencentes ao mesmo sub-algoritmo, todas as linhas de entrada da lógica de mapeamento estão desactivadas. Quando ocorre a invocação de um sub-algoritmo é activada uma dessas linhas, correspondendo ao estado interrompido do respectivo sub-algoritmo. Por sua vez, a lógica de mapeamento converte este estado, no código do sub-algoritmo invocado verificando também se este se encontra disponível. Em caso afirmativo, é feita a salvaguarda do contexto no stack (identificação do sub-algoritmo + estado interrompido). Seguidamente é

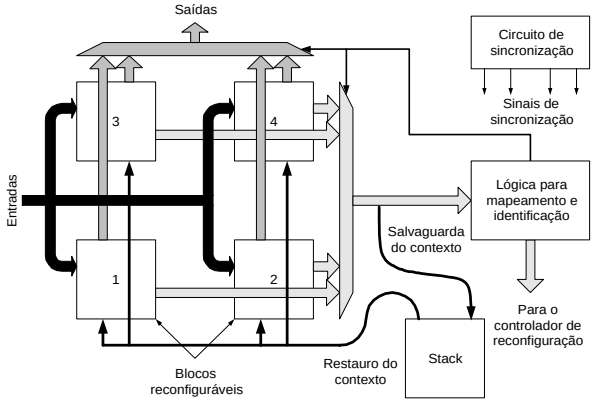

Figura 8 – Estrutura da implementação reconfigurável dinamicamente.

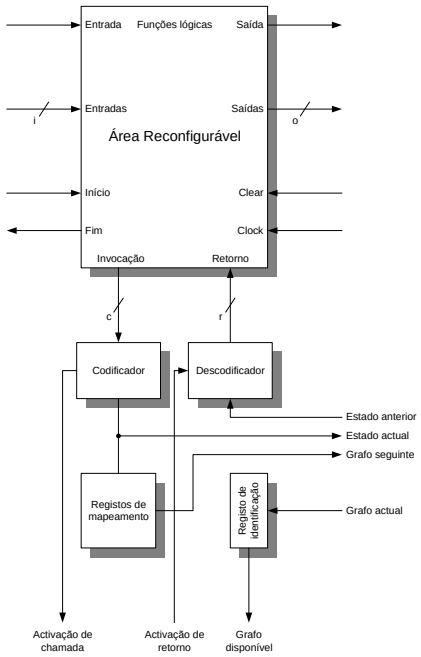

Figura 9 – Bloco base da HFSM reconfigurável dinamicamente.

executado o novo sub-algoritmo e por fim restaurado o contexto. Caso contrário, é feito um pedido ao controlador de reconfiguração (no presente caso o PC) para programar o sub-algoritmo invocado. A profundidade de invocação só é limitada pelo número de níveis do *stack*, podendo existir recursividade. Uma importante característica desta implementação é que somente os estados onde ocorram transições hierárquicas necessitam de ser guardados no stack (os restantes estados são um atributo interno de cada sub-algoritmo). Este facto contribui para um melhor desempenho da unidade de controlo do que a implementação anterior. A flexibilidade deste esquema permite a um sub-algoritmo, iniciar e recomeçar a sua execução após ter sido interrompido, em diferentes blocos físicos. A estrutura proposta é um *template* para construir

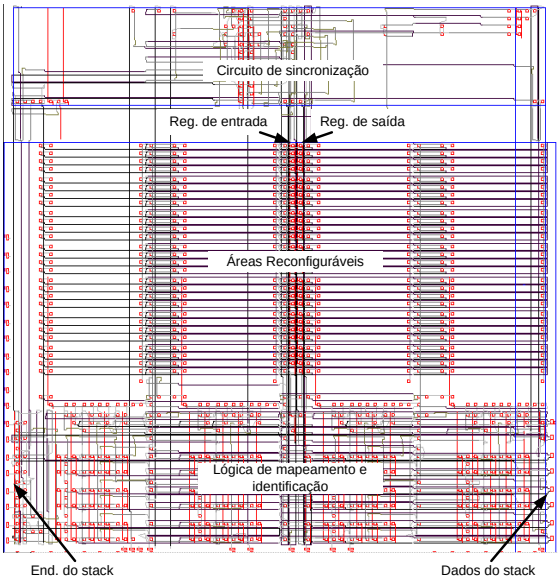

Figura 10 – Layout da HFSM reconfigurável dinamicamente.

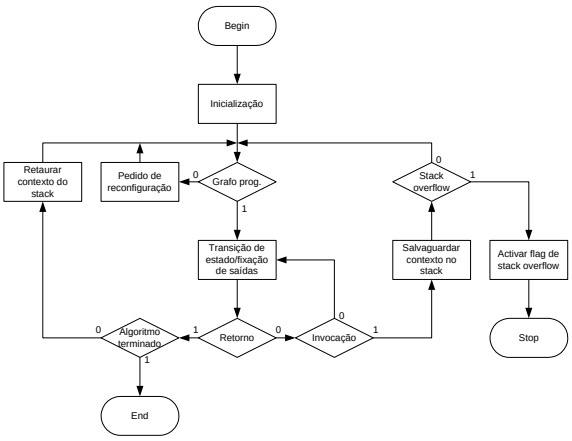

Figura 11 – Especificação simplificada do circuito de sincronização

circuitos de controlo virtuais. Para simplificar a reconfiguração, os pontos de entrada e saída de cada bloco foram fixados. Tal como no caso anterior, foi utilizada a placa Firefly para implementar este circuito. Foram implementados quatro blocos reconfiguráveis. A estrutura de cada bloco é mostrada na figura 9. Na sua interligação, as saídas foram multiplexadas tal como indicado na figura 8. As entradas do circuito foram distribuídas por todos os blocos. A figura 10 mostra o *layout* do circuito na FPGA. A especificação do circuito de sincronização (figura 11) é mais complexa do que no caso anterior devido às capacidades adicionais para detecção do sub-algoritmo pretendido e de *stack overflow*. O objectivo final desta implementação é a síntese automática de um algoritmo, descrito por um HGS, e o carregamento em *hardware* dos seus sub-algoritmos à medida que forem sendo necessários. A ferramenta de síntese utiliza codificação de estados *one-hot*, porque tal como já foi demonstrado em [8], é a mais apropriada para ser utilizada na família XC6200 da Xilinx devido à qualidade de *routing*, menor consumo do circuito e facilidade de conversão de um HGS no respectivo circuito.

V. VICON – UMA ARQUITECTURA VLSI PARA CIRCUITOS DE CONTROLO VIRTUAIS

Tal como já foi referido na secção anterior, a técnica de codificação *one-hot* é bastante apropriada para ser utilizada na implementação de unidades de controlo em FPGAs da família XC6200 da Xilinx. Uma das grandes vantagens deste método é a facilidade de conversão de um algoritmo de controlo descrito por um HGS, no circuito que o implementa. Para além disso, dado que numa transição só dois flip-flops mudam de estado, é possível diminuir consideravelmente o consumo do circuito. Neste método, todas as optimizações são efectuadas ao nível algorítmico (HGS). Após a optimização do algoritmo, a sua conversão num circuito é efectuada directamente. A figura 12 ilustra um exemplo de conversão de uma parte de um GS, no respectivo circuito.

Relativamente às abordagens tradicionais de projecto de unidades de controlo, este método tem a desvantagem de

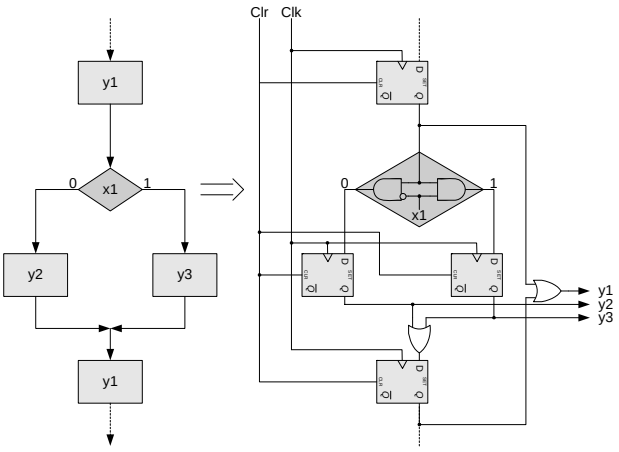

Figura 12 - Conversão de um HGS no respectivo circuito

utilizar um maior número de flip-flops. No entanto, devido às possibilidades de integração existentes actualmente, esta desvantagem pode ser minimizada ou até mesmo anulada se for utilizada uma arquitectura apropriada. No caso da família XC6200, cada célula contém um *flip-flop*, logo o custo de um registo de estados maior, é negligivel.

Combinando as ideias da implementação apresentada na secção anterior com algumas características da família XC6200 da Xilinx, vamos agora apresentar uma arquitectura (*VICON - Virtual Controller Architecture*) para implementação em VLSI de unidades de controlo reconfiguráveis (estatica/dinamicamente). O principal alvo desta arquitectura é a sua integração em circuitos nos quais se pretenda que a unidade de controlo seja flexível e extensível. Estes objectivos são conseguidos através da decomposição do algoritmo em componentes relativamente independentes e também através da reconfiguração da unidade de controlo.

A figura 13 ilustra a estrutura geral da arquitectura. Os seus componentes podem ser divididos em três grupos:

- Área reconfigurável;
- Circuito de sincronização;
- *Template* parametrizável.

Vamos agora descrever detalhadamente cada um destes grupos.

A. Área reconfigurável

A área reconfigurável destina-se a implementar o algoritmo de controlo propriamente dito. A sua estrutura está optimizada para implementar qualquer algoritmo de controlo descrito por um HGS e que utilize na sua implementção codificação *one-hot*. Para uma dada aplicação, a dimensão deste bloco deve ser suficiente para implementar o maior sub-algoritmo existente. Para além das linhas de entrada/saída convencionais, existe também mais uma linha de cada tipo para suportar a implementação de funções lógicas. A diferença entre os dois tipos de saídas é o facto do valor retornado por uma função lógica ser preservado durante o retorno de um sub-algoritmo. A linha “Inicio” é activada pelo circuito de sincronização para que um sub-algoritmo começe a sua execução. Por outro lado, a linha “Fim” é activada pelo sub-algoritmo no final da sua execução para que o circuito de sincronização seja notificado. As linhas “Clock” e “Clear” são ligadas às respectivas entradas de todos os *flip-flops* existentes na área reconfigurável. As linhas “Invocação”/“Retorno” fornecem/recebem o estado *one-hot* actual/interrompido, durante a invocação/retorno de um sub-algoritmo. O número destas linhas corresponde ao número máximo de invocações que um algoritmo pode efectuar.

B. Circuito de sincronização

O circuito de sincronização estabelece a sequência de passos a ser executados pela unidade de controlo (ex. fixação de saídas, salvaguarda e restauro do contexto, controlo de condições de excepção – *stack overflow*, etc.). Ele próprio é uma máquina de estados, mas que se pretende que seja a mesma para todas as aplicações.

C. Template parametrizável

Este grupo engloba os seguintes componentes: registos de entrada/saída, codificador, descodificador, registos de mapeamento, registos de identificação, registo de algoritmo e *stack*. A sua estrutura é suficientemente geral para acomodar uma grande gama de algoritmos de controlo. Assim, as ligações entre os seus componentes são fixas, no entanto, o tamanho dos componentes deve

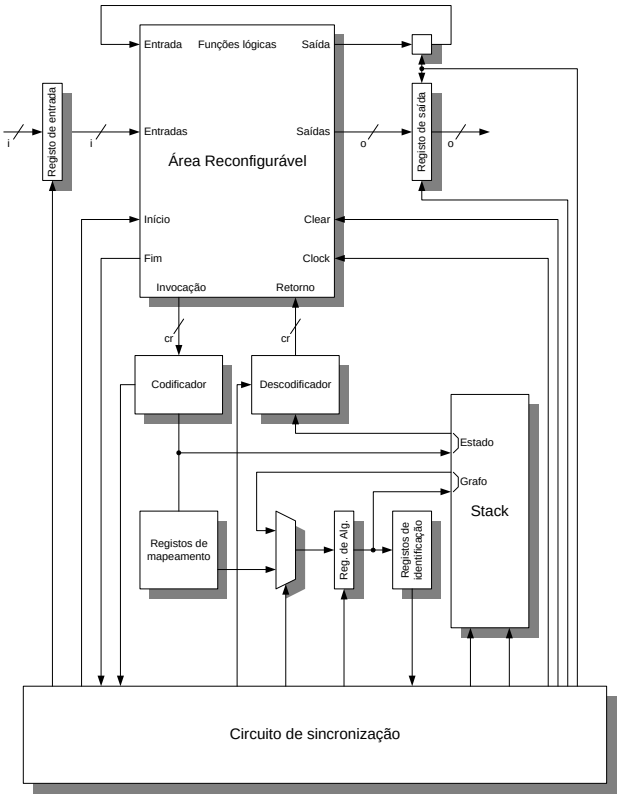

Figura 13 – Estrutura geral da arquitectura VICON.

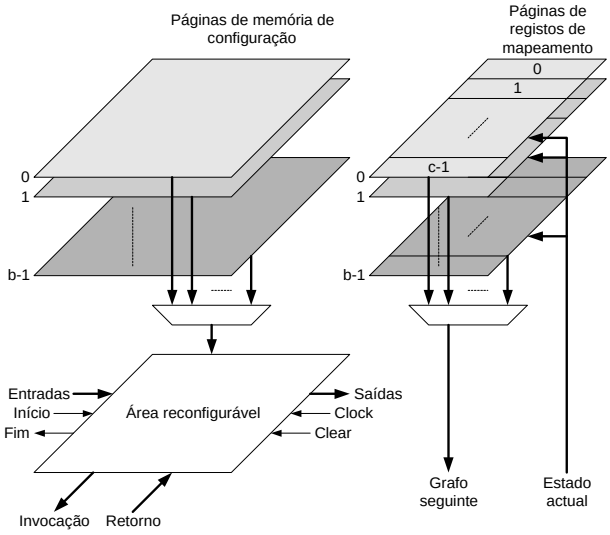

Figura 14 - Comutação de sub-algoritmos na arquitectura VICON.

ser variável em função da complexidade do algoritmo a implementar (número máximo de sub-algoritmos, número máximo de estados por sub-algoritmo e número máximo de invocações por sub-algoritmo). Uma vez que o sub-algoritmo implementado na área reconfigurável utiliza codificação *one-hot*, o codificador permite usar um *stack* com uma palavra de menor dimensão. O codificador só é activado durante a invocação de um sub-algoritmo. Por outro lado, o descodificador executa a função inversa e só é activado durante o retorno de um sub-algoritmo. Como no *stack* é guardado o código do sub-algoritmo interrompido e respectivo estado, é possível efectuar a síntese de cada sub-algoritmo independentemente dos restantes. A ligação entre os diversos sub-algoritmos é efectuada pelos Registos de Mapeamento, que convertem o código binário do estado do sub-algoritmo invocador, no código do próximo sub-algoritmo a ser executado. O Registo de Algoritmo armazena o código do sub-algoritmo a ser executado. As suas entradas podem ser controladas pelos registos de mapeamento ou pelo *stack*, nos casos de invocação ou retorno de sub-algoritmos respectivamente. Os Registos de Identificação possuem os códigos dos sub-algoritmos carregados em hardware.

Esta arquitectura admite pelo menos duas implementações distintas. A primeira é análoga à apresentada na secção anterior, na qual se usam várias áreas reconfiguráveis e multiplexers para efectuar a comutação entre elas. Esta implementação tem a desvantagem de sub-aproveitar o hardware, já que num dado momento só uma área se encontra activa. A outra implementação proposta possui apenas uma área reconfigurável. A sua configuração é controlada por um conjunto de páginas de memória de configuração (figura 14). Em cada instante só uma página se encontra activa, podendo ser efectuada a sua comutação. O número de bancos de registos de mapeamento deve ser igual ao número de páginas de configuração, sendo comutados em sincronismo. A selecção das páginas activas é controlada pelo circuito de sincronização, com o auxílio dos registos de identificação que possuem o código do sub-algoritmo implementado em cada página.

D. Recursos de routing

A área reconfigurável é constituída por matrizes de 4x4 células ligadas em cascata até se atingir o tamanho pretendido (figura 15). Cada matriz possui três tipos de recursos de *routing*:

- **Local** – estabelece a ligação entre células adjacentes em qualquer um dos quadrantes N, S, E, W. Este tipo de *routing* estabelece também a ligação entre células que se encontrem nos extremos de matrizes adjacentes;
- **L4** – em redor de cada matriz de 4x4 células existem 16 multiplexers (4 por lado) que controlam linhas de *routing* comuns a um conjunto de 4 células alinhadas segundo os eixos N, S, E, W. As entradas destes multiplexers podem ser as células que se encontrem nos extremos das matrizes, a linha de *routing* idêntica proveniente da matriz adjacente ou ainda valores lógicos fixos. Nos extremos da área reconfigurável, devido à ausência de matrizes adjacentes, algumas das entradas destes multiplexers podem ser utilizadas como pontos de entrada da área reconfigurável.

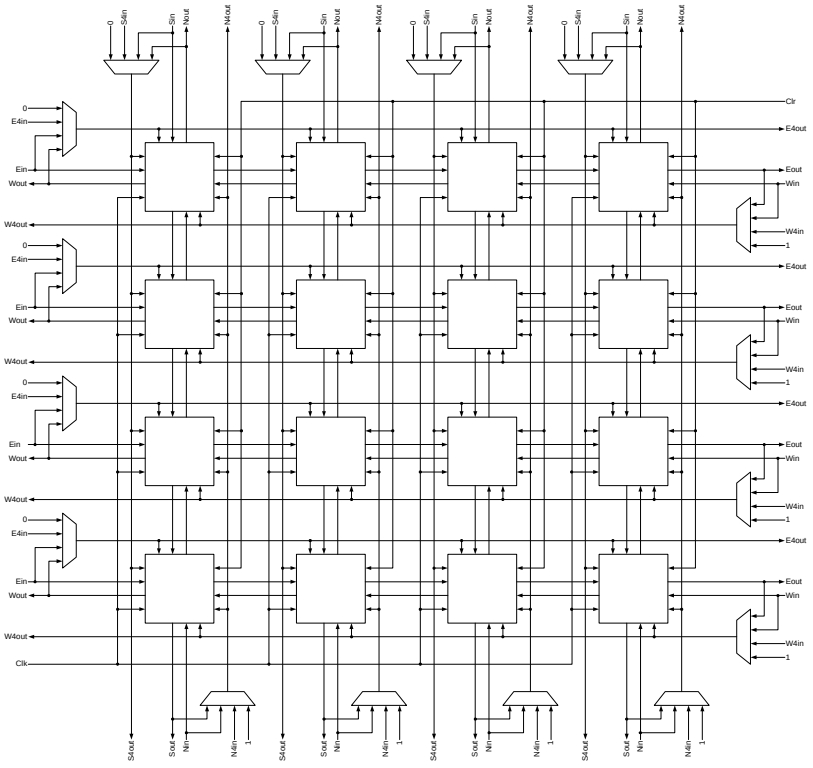

Figura 15 - Bloco de 4x4 células, routing e respectivos multiplexers na arquitectura VICON.

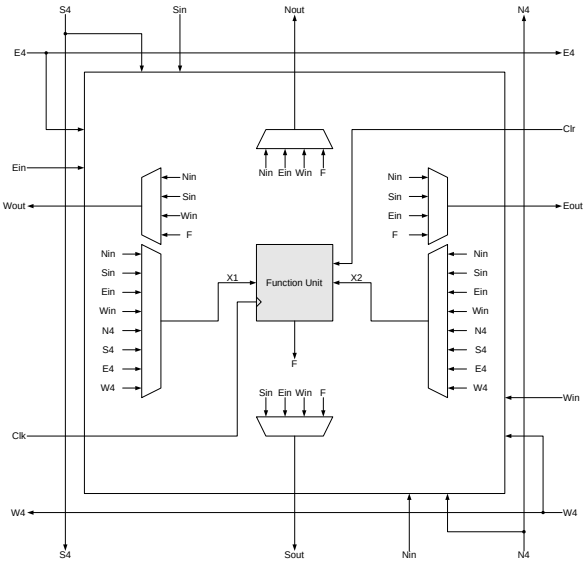

Figura 16 - Routing local da arquitectura VICON.

- **Controlo** – inclui as linhas de *clock* e *clear* que são distribuidas por todas as células sem passar por qualquer multiplexer.

A figura 16 ilustra o *routing* local da arquitectura. As duas entradas da unidade funcional podem ser controladas por qualquer uma das linhas N_{in}, S_{in}, E_{in}, W_{in}, N_4, S_4, E_4, W_4. Os multiplexers que controlam o *routing* local possuem quatro entradas, sendo uma delas a saída da unidade funcional e as restantes outros sinais locais. Cada célula pode ser utilizada em simultâneo para implementar funções lógicas e para efectuar o *routing* de sinais locais.

E. Unidade funcional

Para implementar um algoritmo de controlo especificado por um HGS e utilizando técnicas de codificação *one-hot*, é necessário um reduzido número de primitivas. A tabela 1 ilustra as primitivas necessárias, bem como a sua aplicação.

	Implementação de nodos condicionais
	Cálculo das saídas da unidade de controlo e convergência de nodos
	Implementação de nodos operacionais

Tabela 1 – Primitivas necessárias para implementar unidades de controlo

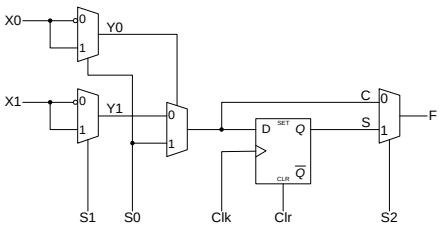

Figura 17 – Célula da arquitectura VICON.

utilizando codificação *one-hot*.

A unidade funcional está optimizada para este pequeno conjunto de primitivas. O seu diagrama lógico está representado na figura 17. Tal como acontece na familia XC6200, é também possível integrar na mesma unidade funcional duas primitivas, desde que uma seja um *flip-flop* e a outra uma porta lógica. Na tabela 2 são apresentadas as configurações possíveis da unidade funcional com os respectivos valores das variáveis de selecção dos multiplexers.

Configuração	Mux Sel		
Nome	S0	S1	S2
0	0	0	0
1	1	0	0
BUF/+FF	1	1	0/1
AND2/+FF	0	1	0/1
AND2B1/+FF	0	0	0/1
OR2/+FF	1	1	0/1
OR2B1/+FF	1	0	0/1

Tabela 2 – Primitivas e respectivos valores das variáveis de controlo dos multiplexers.

VI. ESTADOS TEMPORIZADOS

Para que a arquitectura proposta suporte a especificação de estados de duração variável, para além dos sinais de entrada, saída, invocação e retorno, cada bloco deverá possuir sinais de saída adicionais que serão utilizados pelo circuito de sincronização para controlar o número de

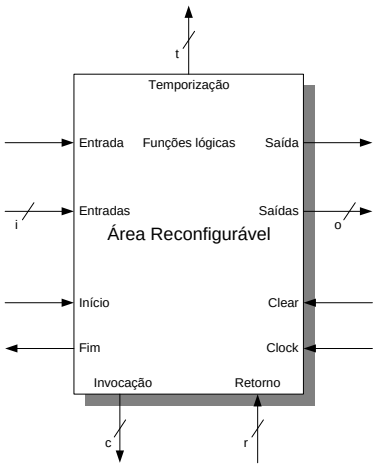

Figura 18 – Área reconfigurável com saídas para controlo de temporização.

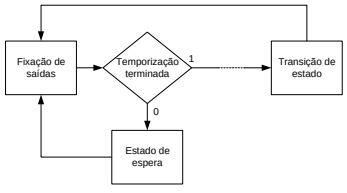

Figura 19- Alteração da especificação do circuito de sincronização para suportar temporização de estados.

ciclos de cada estado da HFSM (figura 18). O número mínimo destes sinais deverá ser $t=\log_2(c)$, em que c é o número máximo de ciclos de relógio que um estado pode durar. A especificação do circuito de sincronização deverá ser alterada por forma a que, entre a fixação das saídas e a transição para o estado seguinte seja introduzido o número de ciclos de espera pretendidos (figura 19).

VII. BIBLIOTECA DE COMPONENTES PARAMETRIZÁVEIS

A biblioteca PRIMS, disponibilizada pela Xilinx, contém todos os componentes que se podem implementar numa célula da família XC6200. Estas primitivas restringem-se a portas lógicas de 2 entradas, buffers, inversores, multiplexers de 2 para 1 e *flip-flops*. Possui também algumas primitivas para simplificar a configuração dos blocos de entrada/saída (*IOBs*) que estabelecem o interface entre as células e os *pads* do circuito. No entanto, é bastante útil possuir portas lógicas de várias entradas. Em particular, no caso das unidades de controlo sintetizadas usando codificação *one-hot* é conveniente possuir portas lógicas OR de N entradas. Para este efeito, foi implementada uma biblioteca de componentes parametrizáveis em VHDL. Esta biblioteca é baseada na PRIMS e entre outros componentes possui: registos, latches, portas lógicas, *arrays* de componentes, contadores, etc. Os registos permitem parametrizar o número de bits, o seu espaçamento e em casos especiais o seu valor inicial. As portas lógicas podem ser de qualquer tipo e permitem a parametrização do número de entradas. A figura 20 ilustra a descrição em VHDL de uma porta OR de N entradas (N≥2). De notar que a interligação das portas OR de 2 entradas que a constituem é efectuada de forma a minimizar os atrasos do circuito (figura 21).

```
-- Porta lógica OR de N entradas
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library PRIMS;
use PRIMS.XC6000_COMPONENTS.ALL;
entity ORN is
  generic(N: integer := 3);
  port(I: in  STD_LOGIC_VECTOR((N-1) downto 0);
       O: out STD_LOGIC);
    attribute FLATTEN of ORN : entity is "";
end ORN;
architecture STRUCT of ORN is
  signal T: STD_LOGIC_VECTOR((2*N)-2 downto 0);
begin
  T((N-1) downto 0) <= I((N-1) downto 0);
    G   : for J in N downto 2 generate
  begin
    B   : OR2  port map(I0 => T(2*(N-J)),
                        I1 => T((2*(N-J))+1),
                         O  => T((2*(N-J))+J));
  end generate;
  O <= T(2*N)-2);
end STRUCT;
```

Figura 20 – Exemplo de uma porta lógica OR de N entradas construida com N-1 portas de 2 entradas.

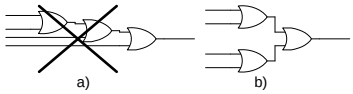

Figura 21 - Porta lógica OR de 4 entrada construída com portas de 2 entradas – a) configuração incorrecta, b) configuração correcta.

VIII. CONCLUSÃO

Os circuitos de controlo são os componentes mais irregulares dos sistemas digitais. Assim, é bastante difícil prever a estrutura de um circuito que garanta bons resultados para a generalidade das aplicações. O problema torna-se ainda mais complicado no caso de circuitos de controlo virtuais, nos quais se pretende alterar o comportamento sem interromper a execução. A abordagem proposta para solucionar este problema baseia-se na especificação hierárquica dos algoritmos de controlo, na síntese separada de cada sub-algoritmo e finalmente na definição de uma arquitectura de circuitos de controlo com as restrições adequadas, por forma a garantir a obtenção de resultados razoáveis desde a fase de síntese até à fase de *routing* do circuito. A arquitectura proposta é constituída por partes fixas, parametrizáveis e sintetizáveis e foi validada numa FPGA da família XC6200 da Xilinx.

AGRADECIMENTOS

Gostaríamos de agradecer ao Professor António Ferrari pelos seus comentários e sugestões durante a realização deste artigo.

REFERÊNCIAS

[1] V. Sklyarov, “Hierarchical Graph-Schemes”. Latvian Academy of Science, Automatics and Computers, Riga, 1984, N 2, pp 82-87.

[2] Xilinx, “XC6200 Field Programmable Gate Arrays, Product Description”, (http://www.xilinx.com/partinfo/6200.pdf), Abril 1997.

[3] V. Sklyarov, A. Rocha, A. Ferrari, “Synthesis of Reconfigurable Control Devices Based on Object-Oriented Specifications”, *Advanced Techniques for Embedded Systems Design and Test*, Kluwer Academic Publishers, 1998.

[4] A. Melo, V. Sklyarov, “Software tools for Design and Implementation of Control Circuits from their Graphical Specification”, 8th HCM BELSIGN Workshop, Outubro 1998.

[5] Xilinx, “Velab: VHDL Elaborator for XC6200 (V0.52)”, http://www.xilinx.com/apps/velabrel.htm.

[6] Xilinx, *Series 6000 User Guide*, 1997.

[7] V. Sklyarov, R. Monteiro, N. Lau, A. Melo, A. Oliveira, K. Kondratjuk, “Integrated Development for Logic Synthesis Based on Dynamically Reconfigurable FPGAs”, *8th International Workshop, FPL’98*, Setembro 1998.

[8] V. Sklyarov, N. Lau, A. Oliveira, A. Melo, K. Kondratjuk, A. Ferrari, R. Monteiro, I. Skliarova, “Synthesis Tools and Design Environment for Dinamically Reconfigurable FPGAs”, Actas do *XI Brazilian Symposium on Integrated Circuit Design*, Setembro 1998.