Ambiente Integrado para Especificação, Projecto e Verificação de Unidades de Controlo em FPGAs

Andreia Melo, Valery Sklyarov

Resumo – **Este artigo descreve uma ferramenta de software que consiste num ambiente integrado utilizado para projectar, implementar e verificar unidades de controlo em FPGAs reconfiguráveis dinamicamente a partir da sua especificação formal. Esta é introduzida graficamente sob a forma de um algoritmo que será convertido numa descrição textual do circuito de modo a ser usado por uma ferramenta de síntese. Após a implementação é possível efectuar a depuração dos circuitos escolhendo em *run-time* o tipo de implementação que foi utilizada, assim como a FPGA escolhida, fechando assim o ciclo de desenvolvimento.**

Abstract - **This paper describes a software tool for design, implementation and testing of control circuits in dynamically reconfigurable FPGAs from their graphical specification. It consists of a graphical editor responsible for the control circuit specification, provides a link to the synthesis tool and allows the circuit debugging, for different types of implementations and FPGAs, thus closing the design cycle.**

I. INTRODUÇÃO

Um sistema digital é basicamente constituído por um circuito de controlo (*control path*) e pelo circuito que o implementa fisicamente (*data path*). O comportamento de circuitos de controlo pode ser descrito algoritmicamente utilizando Esquemas Gráficos (*GS – Graph-Schemes*) e suas variantes, como os Esquemas Gráficos Hierárquicos (*HGS – Hierarchical Graph-Schemes*) [1] e Paralelos Hierárquicos (*PHGS – Parallel Hierarchical Graph-Schemes*). O projecto de sistemas digitais é um processo que se divide em várias fases. A fase inicial corresponde à especificação do circuito, isto é, determinar qual o comportamento que se pretende do circuito utilizando uma notação formal específica, que neste caso se baseia em GSs. Seguidamente passa-se à síntese do circuito, onde a especificação formal do seu comportamento é convertida em código escrito numa linguagem de descrição de hardware (*HDL – Hardware Description Language*), como por exemplo, em VHDL. Neste artigo consideram-se apenas implementações dos circuitos em FPGAs reconfiguráveis dinamicamente, como por exemplo, as da família XC6200 da Xilinx. As fases finais de desenvolvimento são suportadas pelas ferramentas fornecidas pela Xilinx e dizem respeito à implementação dos circuitos nas FPGAs.

A ferramenta de software desenvolvida consiste num ambiente integrado que engloba todas estas fases de desenvolvimento [2]. É utilizada para construir circuitos de controlo de forma gráfica, seguindo as regras da especificação formal baseada em GSs. Manipula hierarquias de GSs, permitindo contudo, a reutilização de GSs em novos circuitos como entidades autónomas. É responsável por fornecer uma descrição do circuito à ferramenta de síntese de modo a obter-se uma descrição em VHDL do circuito. Após todas as fases necessárias para a implementação, permite ainda testar o funcionamento do circuito implementado para várias arquitecturas de FPGAs utilizando diferentes tipos de implementações.

Este artigo está dividido em seis secções. A primeira corresponde à presente introdução. Na segunda secção descreve-se como é efectuada a especificação dos circuitos de controlo e como são construídos os GSs e os HGSs. Descrevem-se também todos os tipos de nodos utilizados na especificação, assim como a sua funcionalidade. Na terceira secção é apresentada a ferramenta desenvolvida, onde são explicados alguns detalhes de implementação, a interface utilizada e as funcionalidades disponibilizadas. Na secção quatro é descrito o processo de síntese de GSs e qual o papel da ferramenta de software neste processo. A quinta secção consiste na descrição de todos os passos de desenvolvimento de um circuito de controlo, desde a sua construção à sua verificação, tomando como exemplo implementação de um circuito numa FPGA da família XC6200, utilizando codificação de estados *one-hot*. A secção seis contém as conclusões.

II. ESPECIFICAÇÃO FORMAL DOS CIRCUITOS DE CONTROLO

Um GS consiste numa especificação gráfica formal de um algoritmo de controlo, composto por diferentes tipos de nodos. Graficamente as suas funções distinguem-se pelas formas rectangulares e losangulares. Os nodos rectangulares, também chamados operacionais, correspondem aos vários estados do circuito e os losangulares, denominados também condicionais, determinam o fluxo do algoritmo mediante o valor binário de um sinal de entrada (0 ou 1).

Os vários tipos de nodos que constituem um Esquema Gráfico encontram-se descritos na tabela 1. Um esquema gráfico é constituído por nodos ligados entre si por intermédio de setas que estabelecem o fluxo de funcionamento do algoritmo. A sua construção é análoga à de um fluxograma. A existência de nodos contendo funções lógicas e macro operações permite a construção de hierarquias, isto é, circuitos de controlo hierárquicos.

Na Figura 1 mostra-se um exemplo de um HGS com dois níveis de hierarquia. À esquerda encontra-se o GS principal que é responsável por invocar os dois GSs da direita. O nível mais elevado da hierarquia corresponde ao GS principal. Quando um GS invoca outro, o nível actual de hierarquia é decrementado.

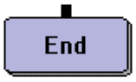
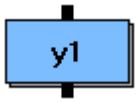
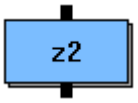
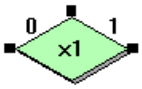
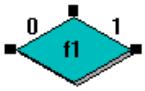
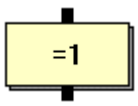
Begin 	• Obrigatoriamente presente num GS; • Representa o ponto de entrada do GS, indicando o início da execução do algoritmo de controlo por ele descrito; • Só é permitido apenas um nodo deste tipo num GS;
End 	• Obrigatoriamente presente num GS; • Representa um ponto de saída do GS, indicando o fim da execução do algoritmo de controlo por ele descrito; • Só é permitido apenas um nodo deste tipo num GS;
Rectangular  	• Contêm um conjunto de micro operações Y={y1, y2, …, yM} e/ou um conjunto de macro operações Z={z1, z2, …, zN} • As micro operações correspondem aos sinais de saída do circuito de controlo (cada sinal de saída possui apenas um bit) e as macro operações descrevem procedimentos descritos por outros GSs.
Losangular  	• Contêm apenas uma condição lógica ou uma função lógica, ou seja, apenas um elemento do conjunto C=X∪F, onde X={x1, x2,…, xi} é o conjunto das condições lógicas e F={f1, f2,…, fj} o conjunto das funções lógicas • As condições lógicas correspondem às entradas do circuito de controlo: uma condição lógica corresponde a um sinal de entrada de 1 bit • As funções lógicas descrevem procedimentos, descritos por outros GSs, os quais devolvem um valor (0 ou 1) que vai determinar o próximo estado do algoritmo.
Assign 	• São nodos rectangulares usados para devolver o valor de retorno de uma função lógica (0 ou 1).

Tabela 1. – Descrição dos vários tipos de nodos e a sua representação gráfica

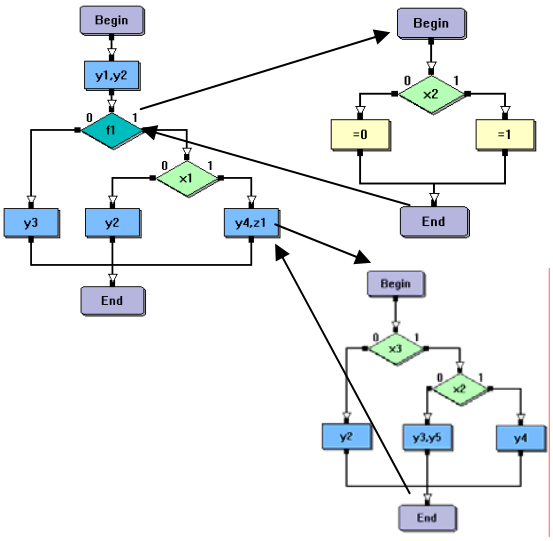


Fig.1 – Exemplo de um HGS com dois níveis de hierarquia

Quando o comportamento de uma unidade de controlo é descrito por um algoritmo extenso, pode ser vantajoso dividi-lo em sub-algoritmos, descritos por outros esquemas gráficos, que são invocados pelo esquema principal ou por outros sub-algoritmos. Quando o resultado da execução de um dado sub-algoritmo pode influenciar o fluxo de execução do algoritmo invocador, estamos perante a necessidade de implementação de uma função lógica, pois dependendo do valor que esta devolve se vai decidir qual o próximo estado do circuito. Por outro lado, se não for necessário devolver um valor, é da mesma forma útil separá-los em módulos de modo a evitar que a sua repetição torne a especificação extensa. Estes sub-algoritmos são descritos em macro operações.

Então, sob o ponto de vista da especificação formal do algoritmo, uma unidade de controlo é hierárquica quando o seu comportamento é descrito por um conjunto de GSs: um esquema gráfico principal (obrigatório), funções lógicas e/ou macro operações. Neste caso, o GS principal é responsável por invocar pelo menos uma função lógica ou macro operação. Quando um GS invoca outro, a sua execução é interrompida para que o novo GS possa ser executado, descendo assim um nível na hierarquia. A execução deste termina quando atinge o nodo End, o que provoca uma subida no nível da hierarquia passando o GS invocador a ser executado novamente a partir do local onde foi interrompido.

Quando pelo menos um nodo, do conjunto de GSs da hierarquia, possuir mais do que uma macro operação, este algoritmo é denominado Paralelo Hierárquico porque as macro operações são descritas por outros GSs que têm de ser executados no nível imediatamente inferior da hierarquia e simultaneamente. Assim, se existirem duas macro operações num nodo operacional, estas serão invocadas e executadas paralelamente e o GS que as invocou é interrompido nesse nodo operacional. Este GS só retoma a sua execução quando ambas as macro operações terminarem a sua execução.

A vantagem decorrente da utilização destes sub-algoritmos é a simplificação da síntese da unidade de controlo porque embora se obtenha um maior número de circuitos, estes serão mais simples de implementar. A descrição formal decorrente de um HGS é independente da implementação do circuito e revela-se bastante útil quando os sinais de entrada e de saída são constituídos por um bit.

A implementação de uma unidade de controlo deste tipo numa FPGA reconfigurável dinamicamente permite tratar cada algoritmo de forma modular. Assim, numa FPGA podem-se implementar vários algoritmos que durante a sua execução podem invocar outros. Se estes não estiverem presentes na FPGA quando forem invocados, é possível carregá-los quando for necessária a sua execução, tirando assim partido da reconfiguração dinâmica. No caso das FPGAs que permitem a sua reconfiguração parcial em pleno funcionamento, como as da família XC6200 da Xilinx, podem-se substituir blocos que não estejam a ser usados pelos algoritmos que foram invocados, reconfigurando apenas os blocos de hardware em causa. Por outro lado, as FPGAs da família XC4000, não permitem a reconfiguração parcial. Neste caso, quando for detectada uma invocação a um algoritmo que não esteja presente é necessário reconfigurar totalmente a FPGA.

III. SOFTWARE PARA ESPECIFICAÇÃO

Foi desenvolvida uma aplicação para plataformas Windows para construir e depurar GSs hierárquicos e paralelos graficamente. O seu nome é Graph Scheme Builder e permite interagir com mais do que uma hierarquia de GSs utilizando uma interface amigável. Esta aplicação foi desenvolvida em Microsoft Visual C++ 6.0, utiliza a arquitectura Document/View e suporta a interacção simultânea de mais do que um documento (MDI – Multiple Document Interface). Um documento corresponde a uma hierarquia de GSs e é gravado com a extensão *.hgs. O ambiente é mostrado na Figura 2 onde se podem ver dois tipos de Views utilizadas nesta aplicação sobre o mesmo documento. Estão separadas por um *splitter* vertical de modo que se possa aumentar ou diminuir o seu tamanho mediante as necessidades. A View da esquerda contém controlos para gestão de toda a hierarquia e uma listagem das descrições dos sinais de entrada (condições lógicas) e de saída (micro operações) da unidade de controlo relativa ao GS que estiver a ser mostrado à direita. Aqui se incluem também as descrições do funcionamento das funções lógicas e das macro operações. A View da direita consiste numa janela que nos mostra graficamente o GS seleccionado da hierarquia.

A. Implementação de nodos, esquemas gráficos e hierarquias

Tal como já foi referido atrás, um esquema gráfico é constituído por vários tipos de nodos que em Visual C++ são descritos segundo uma hierarquia de classes, mostrada

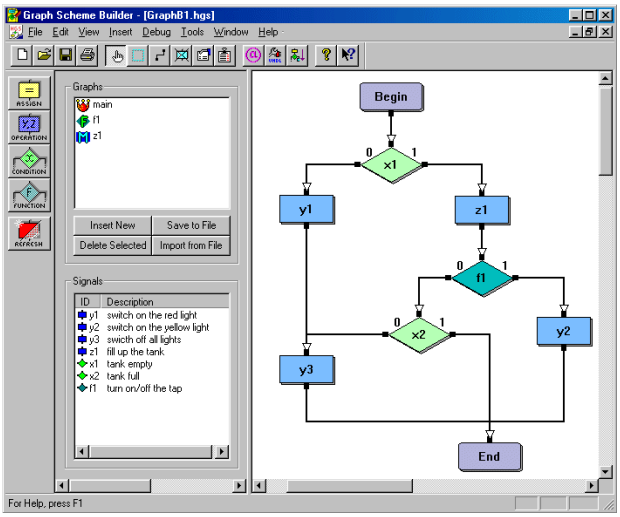


Fig.2 – O ambiente da aplicação Graph Scheme Builder

na Figura 3. Existe uma classe base, CNode, que descreve um nodo e possui atributos e funções comuns a todos os tipos de nodos. Desta classe derivam todas as outras. No entanto, os nodos são primeiramente classificados em nodos rectangulares e losangulares existindo para tal as classes CRectNode e CRhombNode para o efeito, pois estes dois tipos de nodos possuem características específicas. Da classe CRectNode derivam as classes dos nodos rectangulares, que são a CBeginNode, CEndNode, CAssignNode e COpNode que descrevem os nodos Begin. End, Assign e Operacionais, respectivamente. Da classe CRhombNode derivam as classes CLogCondNode e CLgFuncNode que descrevem os nodos que contêm condições lógicas e funções lógicas.

Além das classes dos nodos existem mais duas classes: CGraphScheme e CHierGraph que implementam, respectivamente, um esquema gráfico e uma hierarquia. A primeira, contém nodos e é responsável pela sua gestão, como por exemplo, inserir e apagar nodos, editar as suas propriedades específicas, etc. A segunda contém GSs, instâncias da classe CGraphScheme, permite adicionar GSs à hierarquia, apagá-los, gravá-los separadamente num ficheiro com extensão *.gs e importá-los de novo para uma hierarquia.

Os GSs são encarados como entidades autónomas, podendo ser reutilizados em diferentes unidades de controlo. Assim, foi introduzida a possibilidade de gravação em ficheiros de forma a facilitar a especificação de GSs previamente construídos.

Todas as classes que foram até aqui descritas encontram-se armazenadas numa DLL (Dynamic Link Library) que é carregada em *load-time*. A aplicação GS Builder necessita desta DLL no directório C:\Windows\System.

B. Criação e gestão de uma hierarquia

Um documento armazena uma hierarquia de GSs, contendo cada uma delas pelo menos um GS. A criação de

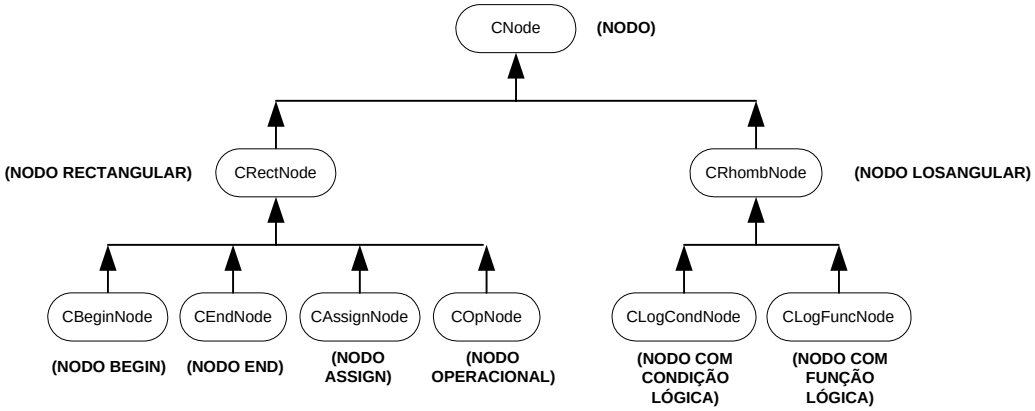


Fig.3 – Hierarquia de classes que representam os nodos

uma nova hierarquia é automática e está associada à criação de um novo documento. O GS principal inserido automaticamente contém apenas dois nodos, Begin e End, pois estes são obrigatórios e únicos em qualquer GS.

Como se pode ver na Figura 2, a View da esquerda contém todos os controlos necessários à gestão de uma hierarquia. Por baixo da lista de nomes dos GSs contidos na hierarquia existem botões que permitem inserir e apagar GSs na hierarquia, gravá-los e importá-los. Para modificar o respectivo nome basta editá-lo na lista, tendo o cuidado de o seleccionar previamente. Por defeito é mostrado o GS principal na janela pertencente à View da direita mas efectuando um *double-click* sobre o nome de uma macro operação ou de uma função lógica, o respectivo esquema é imediatamente mostrado.

C. Interacção e visualização gráfica

A interface da aplicação GS Builder é constituída por um menu principal, onde se pode aceder a todas as funcionalidades disponibilizadas pelo programa, *toolbars* específicas para cada tipo de operação e duas janelas, uma para controlo e outra para visualização gráfica de um dado GS. Esta última permite alguma interacção com o rato incluindo a activação de menus popup, específicos para cada objecto escolhido.

Existem três *toolbars* entre as quais apenas duas são inicialmente visíveis. A principal situa-se por baixo do menu principal e é utilizada para efectuar operações comuns sobre ficheiros, como por exemplo, criar um documento/hierarquia, abrir e gravar uma hierarquia, imprimir o esquema e comutar entre os diferentes modos de operação: selecção, por defeito, inserção de ligações entre nodos, apagar nodos seleccionados e editar as suas propriedades. À esquerda situa-se a segunda *toolbar*, utilizada para interagir com a janela gráfica, que se destina a inserir diferentes tipos de nodos no GS que está a ser mostrado no momento. Os nodos Begin e End não estão presentes nesta *toolbar* pois são inseridos automaticamente quando o GS principal é criado ou uma nova função lógica ou macro operação é inserida na hierarquia. Como cada GS tem obrigatoriamente apenas um par de nodos Begin e End, a inserção automática restringe o acesso a estes nodos por parte do utilizador de modo a evitar que se cometam erros. A terceira *toolbar* é activada/desactivada quando se inicia/termina a depuração de um GS. As opções disponibilizadas serão descritas na secção IV.

Os nodos são representados graficamente por figuras geométricas que possuem um ponto (conector) de entrada e um/dois ponto(s) (conector(es)) de saída para os nodos rectangulares e losangulares respectivamente (ver tabela 1). As excepções a esta regra são os nodos Begin, com apenas um conector de saída e End, com apenas um conector de entrada. Cada tipo de nodo tem uma cor associada e no seu interior encontram-se as propriedades relevantes para cada tipo, como se podem ver na tabela 2. Todos os nodos possuem dimensões fixas à excepção dos operacionais, que possuem um comprimento mínimo fixo mas que pode aumentar em função do tamanho da lista de micro e macro operações que possui no seu interior.

Tipo de Nodo	Propriedade visualizada graficamente
Begin	Palavra “Begin”
End	Palavra “End”
Assign	Valor lógico: “=0” ou “=1”
Operacional	Lista de micro e macro operações. Ex: “y1, y3, z2”
Condição Lógica	Nº da condição lógica. Ex: “x1”
Função Lógica	Nº da função lógica. Ex: “f4”

Tabela 2. – Propriedades de cada tipo de nodo mostradas graficamente

Esta tabela permite constatar que um sinal (entrada = condição lógica; saída = micro operação) é representado por uma letra seguida de um número. Perante esta notação simplificada, para uma fácil representação gráfica, torna-se útil associar uma descrição ao sinal para que seja mais compreensível durante a análise do algoritmo. São estas

descrições que se encontram na listagem da janela da esquerda, que já foi referida anteriormente.

As ligações estabelecidas entre nodos são representadas por setas que começam num conector de saída de um nodo e terminam num conector de entrada de outro nodo. Não é pois permitido ligar a saída de um nodo à sua própria entrada. O desenho das ligações é efectuado de forma automática, carregando com o botão do rato sobre um conector de saída e arrastando-o até ao conector de entrada do próximo nodo. De um conector de saída só pode sair uma ligação enquanto a um conector de entrada podem chegar uma ou mais ligações. Cada ligação é constituída por segmentos de recta horizontais e verticais. Assim, durante o movimento do rato, as coordenadas dos vértices da ligação são calculadas tendo em conta as coordenadas do conector de saída e as coordenadas actuais do rato. Quando se move um nodo previamente ligado, dado que as suas coordenadas se vão alterar, as ligações são actualizadas imediatamente durante o deslocamento.

Ao efectuar um *double-click* sobre um nodo, é determinado o seu tipo em *run-time* e é aberta uma janela com duas pastas, específicas desse nodo, que permitem alterar as suas propriedades. A primeira pasta contém as características do nodo (que se podem editar) e a segunda permite alterar as suas características gráficas, isto é, as posições dos pontos de conexão.

Utilizando o botão direito do rato sobre um nodo activa-se um menu popup. Existem três tipos de menus, com itens diferentes, que são activados consoante o local escolhido pelo rato. Ao carregar sobre o rectângulo/losângulo de um nodo, podem-se escolher as opções:

- **Properties** – abre uma janela que permite visualizar/alterar as propriedades do nodo; é equivalente a um *double-click* sobre o nodo
- **Delete** – apaga o nodo
- **Flip Vertically** – o conector de cima troca de posição com o de baixo; não tem qualquer efeito sobre os nodos Begin e End porque estes devem manter-se fixos
- **Flip Horizontally** - o conector da esquerda troca de posição com o da direita

Ao carregar sobre um ponto de conexão activa-se o segundo menu popup com apenas uma opção Disconnect, que permite desfazer todas as ligações estabelecidas nesse ponto. O terceiro é activado carregando sobre o fundo da janela. Este contém também apenas uma opção Refresh, que permite redesenhar o conteúdo da janela.

IV. O PROCESSO DE SÍNTESE DE ESQUEMAS GRÁFICOS

Do projecto de uma unidade de controlo à sua implementação na FPGA percorrem-se várias etapas. A fase inicial consiste na especificação formal do algoritmo, isto é, na construção do GS por intermédio do GS Builder. Aqui o GS é gravado em formato binário (*.hgs para uma hierarquia e *.gs para um esquema) e opcionalmente em formato de texto (*.txt). Este último é utilizado por uma ferramenta de síntese que converte o GS em código VHDL, como se pode ver na figura 4.

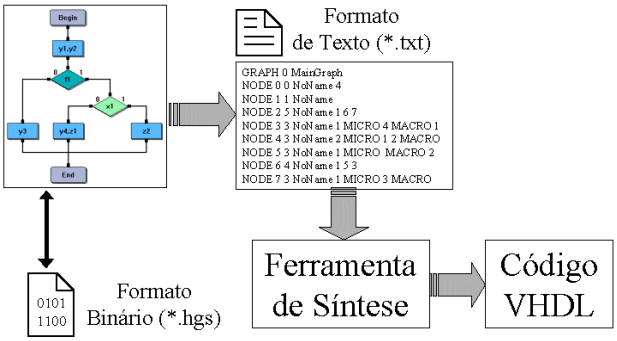


Fig. 4 – A ferramenta de síntese utiliza o formato de texto produzido pelo GS Builder para converter o GS em código VHDL

Antes de executar o programa de síntese a partir do GS Builder, o GS, ou a hierarquia de GSs, é analisada(o) para verificar se todos os nodos estão correctamente ligados e se existe algum nodo ao qual não foi atribuída nenhuma condição ou função lógica, micro ou macro operação. Toda a informação estatística acerca de uma hierarquia se pode observar numa janela que contém os seguintes itens:

- Nº total de nodos;
- Nº de entradas não ligadas;
- Nº de saídas não ligadas;
- Nº de nodos operacionais vazios (sem micro e macro operações);
- Nº de nodos com condições lógicas vazios (aos quais não foi atribuída uma condição lógica);
- Nº de nodos com funções lógicas vazios (aos quais não foi atribuída uma função lógica);
- Tabela com o nº de condições e funções lógicas, micro e macro operações para cada GS pertencente à hierarquia.

Se for encontrado algum nodo nestas circunstâncias o utilizador é notificado para corrigir o esquema e só depois de corrigido é que lhe é permitido sintetizar o circuito.

Assim, a ligação entre o GS Builder e o programa de síntese é estabelecida por este ficheiro de texto que descreve toda a hierarquia. Diferentes formatos de texto podem ser convertidos no formato utilizado pelo GS Builder utilizando uma aplicação simples de software que actue como conversor. Desta forma, resultados fornecidos por outras ferramentas de síntese, que utilizem outro tipo de especificação, podem ser comparados com os resultados obtidos pela ferramenta de síntese desenvolvida. Esta gera dois tipos de implementações, codificação de estados *one-hot* e codificação binária de micro operações, e fornece uma *test-bench* permitindo observar os resultados para efeitos de teste. Na figura 5

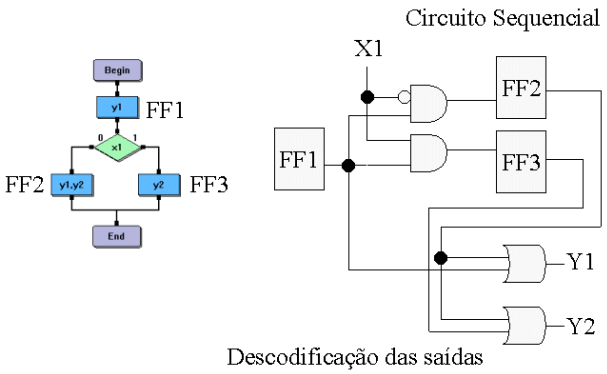


Fig. 5 – Exemplo de síntese de um GS simples utilizando codificação one-hot

mostra-se um exemplo simples da síntese de um GS utilizando a codificação *one-hot*.

Esta ferramenta não necessita de efectuar uma marcação de estados do GS para implementar uma máquina de estados finita de Moore ou de Mealy. No entanto, é possível efectuar a marcação dos estados para uma máquina de Moore utilizando o GS Builder. Opcionalmente, esta marcação pode ser gravada no ficheiro de texto que poderá servir de entrada a outras ferramentas de síntese. A marcação dos estados obedece às regras específicas de cada máquina de estados, que não se pretendem detalhar neste artigo [3]. Contudo, na figura 6 mostra-se o resultado de uma marcação de estados num HGS para uma máquina de Moore, utilizando o GS Builder.

V. CONSTRUÇÃO E DEPURAÇÃO DE UM ESQUEMA GRÁFICO HIERÁRQUICO

Após a especificação, síntese e execução do GS na FPGA, o ciclo de desenvolvimento fecha-se a partir do momento em que é possível efectuar a sua depuração no GS Builder. Nesta secção descreve-se o ciclo de desenvolvimento completo no caso da implementação da unidade de controlo numa FPGA reconfigurável dinamicamente XC6200 da Xilinx.

O GS é construído utilizando o GS Builder e gravado em formato de texto, escolhendo a opção Save As do menu File e colocando explicitamente a extensão *.txt. A ferramenta de síntese converte a descrição textual em código VHDL que será compilado pelo *VELAB elaborator* [4] uma ferramenta disponibilizada pela Xilinx para produzir uma Netlist EDIF. O *XACT6000* [5] utiliza esta Netlist para efectuar o mapeamento, colocação e roteamento do circuito na FPGA. Produz um ficheiro *.cal, que é responsável pela configuração da FPGA e um ficheiro *.sym, o qual será utilizado pelo GS Builder para proceder à depuração do circuito.

A depuração de um GS depende, obviamente, do tipo de implementação escolhido. Para que a depuração não se restrinja a um único tipo de implementação, são utilizadas DLLs cuja função é encapsular o tipo de implementação utilizado, assim como a arquitectura da própria FPGA, como por exemplo a implementação que utiliza a codificação *one-hot* nas XC6200.

Estas DLLs são responsáveis pelo carregamento da configuração do circuito na FPGA. No caso da implementação de um circuito que foi sintetizado utilizando a codificação one-hot na família XC6200, o primeiro passo é o carregamento do ficheiro *.cal, a programação da memória e inicialização.

As DLLs possuem uma interface gráfica própria (janela de Setup), que é utilizada pelo GS Builder, onde são introduzidos os parâmetros específicos dessa implementação. Desta forma, o GS Builder tem a possibilidade de apresentar interfaces distintas para cada tipo de implementação e acede indirectamente ao circuito na FPGA por intermédio da DLL, que lhe esconde os detalhes, como por exemplo, o número de ciclos de

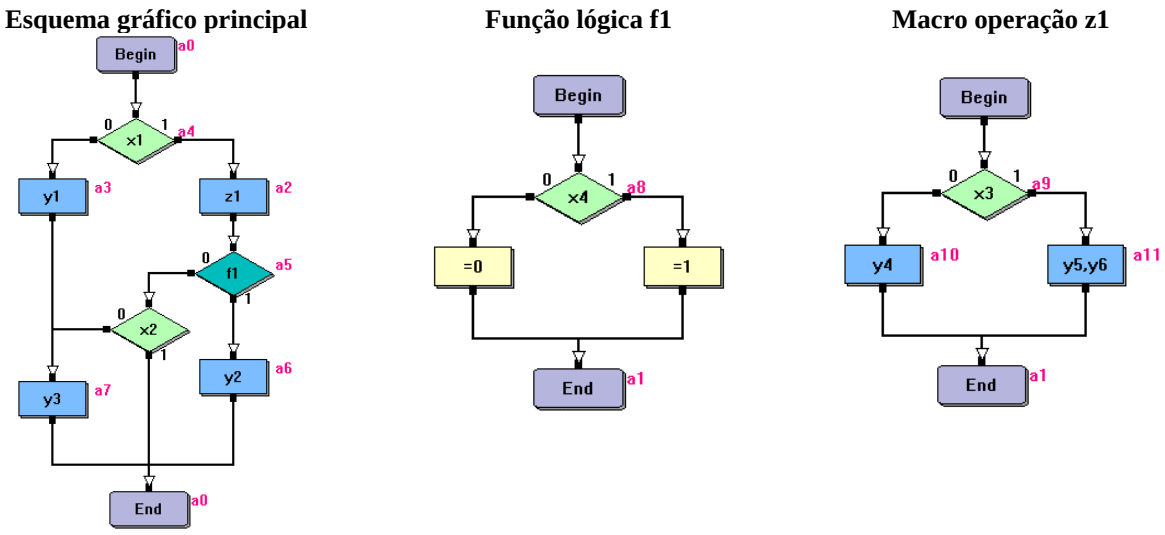


Fig.6 – Marcação de estados num HGS para uma máquina de estados finita de Moore

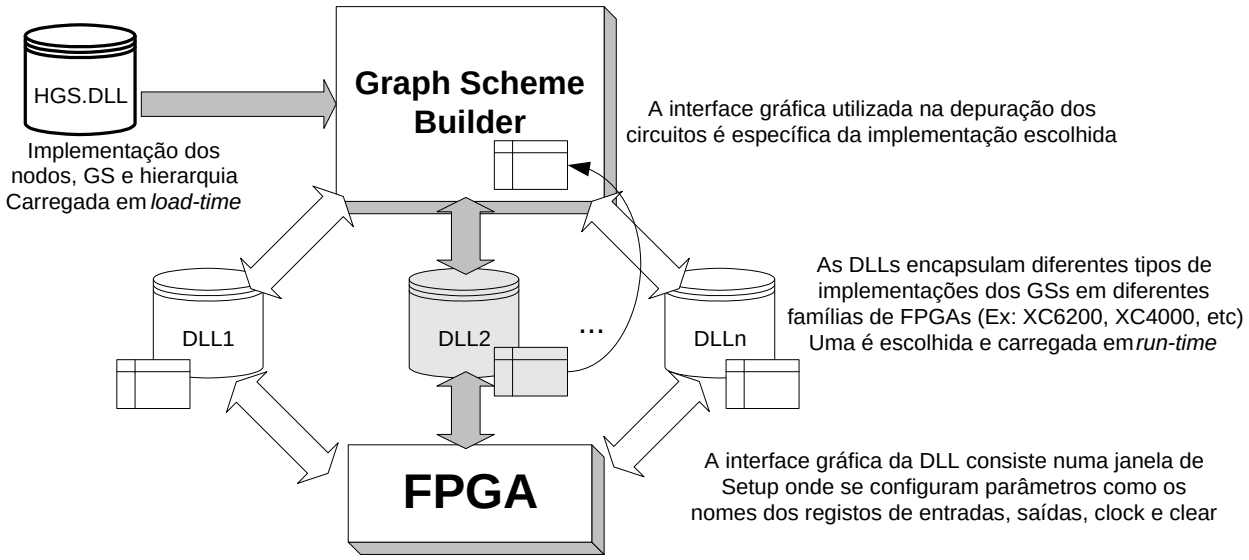


Fig.7 – Arquitectura utilizada na depuração de GSs implementados numa FPGA

relógio que são necessários para efectuar uma transição de estado no algoritmo. A figura 7 mostra a arquitectura utilizada na depuração de GSs.

O GS Builder necessita de conhecer os nomes das DLLs para as poder eventualmente utilizar. Assim, estas devem ser colocadas num sub-directório específico (\Libraries) do directório onde está colocado o ficheiro executável GraphBuilder.exe. Para que seja possível utilizar diferentes DLLs para depurar diferentes implementações, estas devem possuir a mesma interface, isto é, os mesmos protótipos das funções que exportam, para que o GS Builder as possa invocar de igual forma. A diferença entre elas reside no corpo das funções, que é específico de cada uma e transparente ao GS Builder.

Antes de proceder à depuração do algoritmo, este é mais uma vez verificado para ter a certeza de que foi correctamente construído. O próximo passo é escolher a implementação que foi utilizada. O GS Builder verifica quais as DLLs que estão disponíveis no sub-directório Libraries e apresenta uma lista com os seus nomes e uma breve descrição da implementação. Aqui o utilizador deve efectuar a sua escolha, que conduz ao carregamento da respectiva DLL. Seguidamente surge uma janela de Setup, fornecida pela DLL, com uma listagem dos registos que serão utilizados. Por exemplo, registos de entradas, saídas, clock e clear. Consultando o ficheiro *.sym, preenchem-se os campos desta janela.

Na figura 8 mostra-se o ambiente utilizado para a depuração de um circuito. Aqui se podem ver as janelas destinadas à visualização dos valores dos sinais de entrada e de saída e uma janela de *log*. Inicialmente as duas primeiras contêm os valores de inicialização (zeros em todos os sinais) e a última contém a mensagem "Reset circuit" que corresponde à primeira operação efectuada sobre o circuito. Estas janelas podem ser activadas a partir de uma *toolbar*, que disponibiliza algumas funções de suporte. Da esquerda para a direita, temos:

	Visualizar a janela de Setup onde se podem editar os nomes de todos os registos relevantes (entradas, saídas, clock e clear)
	Recomeçar a depuração efectuando um *reset* ao circuito (*restart*)
	Terminar a depuração regressando ao ambiente anterior (*stop*)
	Transitar para o próximo estado (*step*)
	Executar o algoritmo de controlo até que algo interrompa a sua execução (*run*); aqui são efectuadas transições de estados até que o utilizador interrompa a execução do algoritmo de controlo pressionando o próximo botão desta lista, ou quando for encontrado um *breakpoint*
	Interrupção da execução do algoritmo de controlo (*break*)
	Inserir e remover *breakpoints*; para inserir um *breakpoint* basta seleccionar o nodo onde se pretende interromper a execução do algoritmo de controlo e pressionar este botão. O breakpoint é indicado pela presença de um círculo ao lado direito do nodo. Se quando este botão for pressionado o nodo seleccionado já possuir um *breakpoint*, então este será retirado
	Actuar nos valores binários dos sinais de entrada seleccionando o sinal que se pretende na lista dos sinais de entrada e pressionando de seguida este botão. Como se trata de sinais binários existem apenas dois valores possíveis (0 e 1). Assim, ao carregar sobre este botão o valor que se obtém corresponde à negação do valor anterior.
	Visualizar os sinais de entrada numa lista na janela *watch inputs*
	Visualizar os sinais de saída numa lista na janela *watch output*

	consultar uma janela de *log* para obter informações acerca dos passos efectuados até ao momento. Se ocorrer um erro durante o carregamento do ficheiro *.cal ou na abertura do ficheiro *.sym, esse erro será identificado. Assim como se for efectuado o *reset*, por exemplo, por outra aplicação que consiga aceder à FPGA, esta situação é detectada e é enviada uma mensagem para a janela de *log*.
	apagar o conteúdo da janela de *log*

A depuração é interrompida no caso do GS ser alterado graficamente, pois neste caso o novo algoritmo não corresponde ao circuito implementado e a depuração actual deixa de fazer sentido.

O GS Builder efectua simultaneamente dois tipos de depuração: por software e por hardware. Os nodos activos durante a execução do circuito da unidade de controlo são marcados de forma diferente para ambos os casos. Graficamente, um nodo é identificado como activo quando é envolvido por um rectângulo, cuja cor é específica para cada tipo de depuração: rosa corresponde à depuração por software e preto a hardware.

A depuração por software segue a execução do algoritmo baseada nos valores dos sinais de entrada (contidos no registo de entradas), activando em cada transição de estado, o nodo que espera estar realmente activo, sem verificar os valores das saídas. Esta verificação é efectuada pela depuração por hardware, que com base nos valores das saídas (contidos no registo de saídas), identifica quais os nodos que contêm apenas as micro operações que estão activas no momento e activa-os colocando um rectângulo preto em todos os nodos que satisfaçam essa condição.

O nodo que está activo por software deve ser o mesmo que está activo por hardware, caso contrário é identificado um erro, pois o nodo com o rectângulo preto (o que está activo fisicamente) deve coincidir com o nodo activo previsto por software, marcado com o rectângulo rosa. Graficamente, o erro corresponde à separação dos dois rectângulos.

Todo este procedimento permite fechar o ciclo de desenvolvimento de uma unidade de controlo, porque após todos os passos da síntese do circuito existe *feedback* para a especificação inicial. Isto introduz a possibilidade de correcção do GS ou da hierarquia inicial e se for necessário sintetiza-se de novo o circuito e testa-se na FPGA com a ajuda do GS Builder.

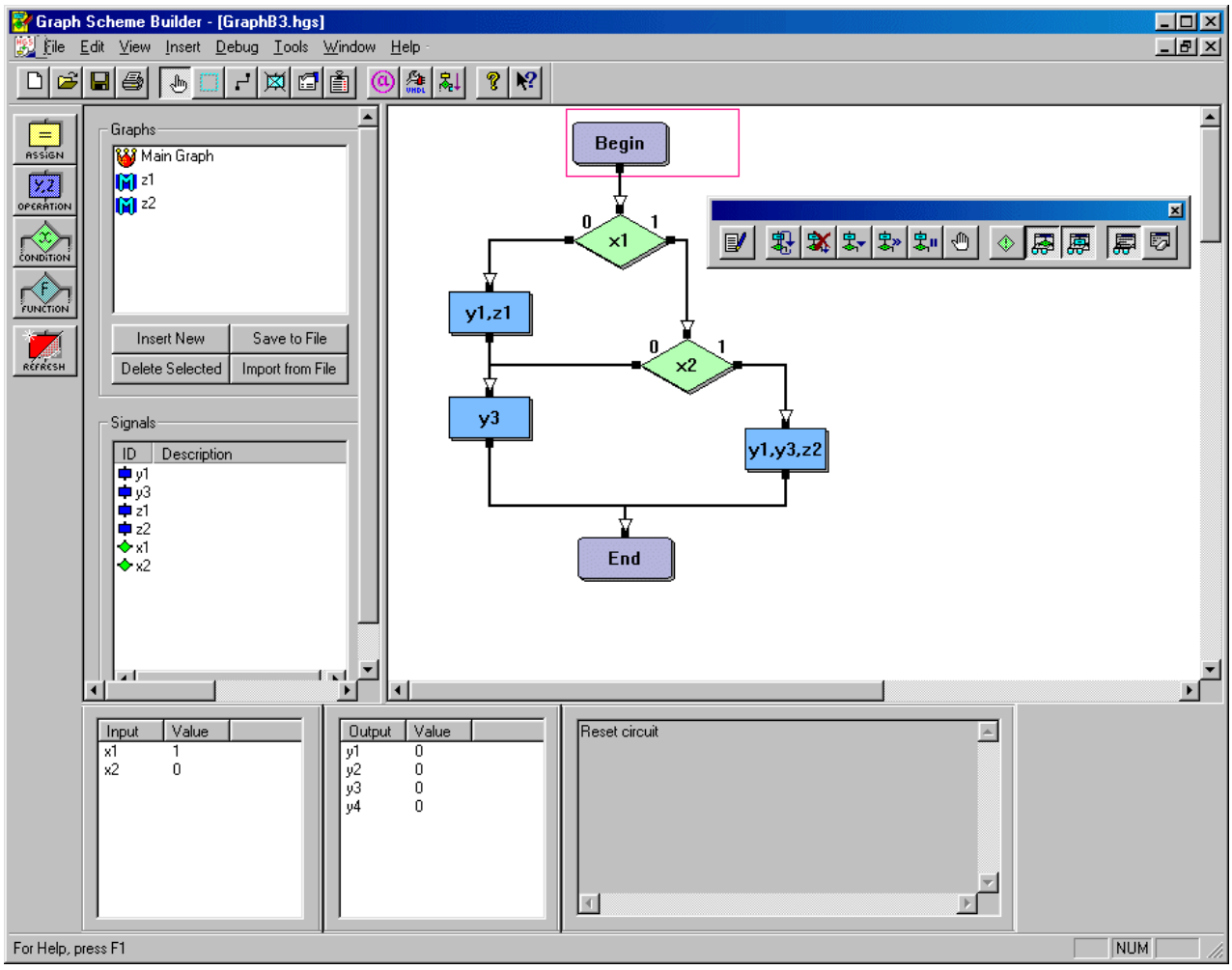


Fig. 8 – Inicialização do ambiente para depuração de um HGS

VI. CONCLUSÕES

A especificação dos algoritmos de controlo através de HGSs suporta a hierarquia, fornecendo assim o suporte para uma decomposição *top-down*. Desta forma, revela-se uma notação conveniente para especificar o comportamento de máquinas de estado finitas.

Os componentes de uma hierarquia, isto é, os vários esquemas gráficos que constituem a hierarquia podem ser encarados como entidades autónomas e independentes, podendo ser reutilizados noutras unidades de controlo, e por isso em diferentes produtos.

O tipo de especificação apresentada pode ser facilmente modificada e expandida. Isto traduz-se em alterações que se podem introduzir aos tipos de nodos disponibilizados, assim como a criação de novos tipos [6], [7].

O software desenvolvido fornece um ambiente integrado que permite construir e alterar algoritmos de controlo de forma gráfica e pode ser utilizado para sintetizar o circuito, implementá-lo na FPGA e testá-lo. Fecha o ciclo de desenvolvimento, fornecendo *feedback* para a especificação inicial, permitindo a alteração dessa especificação e a repetição da síntese.

O tipo de arquitectura utilizada no teste ou depuração dos algoritmos, baseada em bibliotecas (DLLs), é extremamente flexível, possibilitando a depuração de unidades de controlo independentemente da FPGA onde foram implementadas e do tipo de implementação que se utilizou. Existe uma DLL para cada caso, isto é, para cada tipo de implementação em cada arquitectura, que o utilizador escolhe em *run-time*, antes de iniciar a depuração do circuito.

AGRADECIMENTOS

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

REFERÊNCIAS

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

[2] Melo, A., Sklyarov, V., "Software Tools for Design and Implementation of Control Circuits from their Graphical Specification", actas de 8th BELSIGN workshop, Madrid, Outubro 1998.

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

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

[5] Xilinx, “Series 6000 User Guide”, 1997.

[6] Sklyarov, V., Monteiro, R. S., Lau, N., Melo, A., Oliveira, A., Kondratjuk, K., “Integrated Development for Logic Synthesis Based on Dynamically Reconfigurable FPGAs”, *Field-Programmable Logic and Applications*, 8th International Workshop, FPL ’98, Tallinn, Estonia, Agosto 1998.

[7] Sklyarov, V., Lau, N., Oliveira, A., Melo, A., Kondratjuk, K., Ferrari, A. B., Monteiro, R. S., Sklyarova, I., “Synthesis tools and design environment for dynamically reconfigurable FPGAs”, *SBCCI98 XI Brazilian Symposium on Integrated Circuit Design*, Búzios, Rio de Janeiro, Brazil, Setembro 1998.