Síntese e Implementação de Circuitos Digitais Reconfiguráveis Dinamicamente

Valery Sklyarov

Resumo – **Este artigo apresenta uma técnica de desenvolvimento de circuitos reconfiguráveis que foi proposta aos alunos da disciplina de computação reconfigurável da especialização LECT (Licenciatura em Engenharia de Computadores e Telemática). A técnica é baseada em *hardware templates* (HT) que são circuitos desenvolvidos para um grupo de aplicações semelhantes. A personalização do HT para uma aplicação particular é conseguida através da especificação da sequência de controlo apropriada. Alterações na sequência de controlo podem ser efectuadas sobre circuitos de controlo que permitem a modificação estática e dinâmica do seu comportamento. O comportamento é especificado com a ajuda de máquinas de estados finitos reprogramáveis (MEFR). O artigo descreve projectos propostos aos alunos com base na técnica mencionada acima. Cada projecto inclui uma parte de software desenvolvida em C++ e uma parte de hardware realizada numa FPGA XC4010XL da Xilinx ligada ao computador através da porta paralela. O circuito em FPGA é composto pelos quatro componentes seguintes: interface com o computador; um conjunto de unidades de controlo; uma unidade de execução e um componente que assegura modificações dinâmicas na funcionalidade do circuito. Os quatro artigos seguintes [1-4] apresentam os resultados concretos dos projectos e mostram como os problemas propostos foram resolvidos pelos alunos do quarto ano da LECT.**

Abstract – **The paper presents a technique for the design of dynamically reconfigurable circuits that has been proposed to students within the discipline on reconfigurable computing (computação reconfigurável) for the LECT specialty (Licenciatura em Engenharia de Computadores e Telemática). The technique is based on the use of *hardware templates* (HT), which are circuits that have been designed for a wide group of similar applications. Customizing the HT for a particular application is achieved by specification of the proper control sequence. Possible changes in control sequences are carried out by control circuits that allow static and dynamic modifications to their behavior. Such behavior has been provided with the aid of a reprogrammable finite state machine (RFSM). The paper describes the proposed projects based on the mentioned above technique. Each project includes a software part implemented in C++ programs running on a host computer and a hardware part realized in commercially available FPGA XC4010XL of Xilinx linked to the host computer through a parallel port. The FPGA circuit is composed of the following four primary components: interface with the host computer; a set of control units, an execution unit and a reconfiguration handler that provides the required dynamic modifications to the FPGA circuit functionality. The following 4 papers [1-4] present the concrete results of the projects and show how the proposed problems have been solved by the 4th year students of LECT.**

I. INTRODUÇÃO

Os sistemas digitais podem ser divididos em duas unidades que são a unidade de controlo (UC) e a unidade de execução (UE). Todos os detalhes sobre este tipo de decomposição estão apresentados em [5]. Neste artigo utilizamos a mesma terminologia e sinais convencionais descritos em [5]. Para a maioria das aplicações práticas a arquitectura de hardware dos sistemas digitais é orientada ao respectivo problema e esta arquitectura não pode ser alterada. Por exemplo, um processador implementa um conjunto de instruções que não podem ser modificadas de acordo com a tarefa a executar. Geralmente cada tarefa requer um subconjunto de operações específicas que permitem resolver o mais eficientemente possível do ponto de vista do tempo de execução ou de outras características. Por exemplo, problemas de processamento de imagens invocam várias operações sobre "bitmaps" (sobre fragmentos de memória), os sistemas de comunicação utilizam protocolos de troca de dados, os problemas de compressão de dados requerem operações sobre vectores booleanos e ternários, etc. Obviamente é difícil construir um sistema óptimo para todas as aplicações possíveis. Por outro lado, este problema não é importante. Geralmente é necessário construir um sistema orientado a uma área que requer modelos de computação específicos para resolver um subconjunto de tarefas bastante semelhantes. A área pode cobrir, por exemplo, problemas combinatórios. De facto existe grande número de modelos e métodos para resolver problemas combinatórios, mas normalmente queremos trabalhar com um subconjunto de modelos e métodos que preferimos, tais como foram considerados em [6,7]. É possível criar uma arquitectura adequada para implementar os modelos e métodos que foram escolhidos (ver, por exemplo, [8-10]). O respectivo circuito pode implementar todas as operações de que precisamos para os nossos modelos. Por outro lado, normalmente, cada tarefa particular só requer operações específicas e o número destas operações é bastante limitado, i.e. substancialmente menor que o

número de operações realizadas pelo circuito. Por isso, é desejável construir uma arquitectura reprogramável que pode ser alterada dependente do problema que tem de ser resolvido. Uma tarefa deste tipo foi proposta aos alunos do quarto ano da disciplina "Computação reconfigurável". Cada grupo composto por um ou dois alunos deve projectar e implementar um circuito que permite executar um conjunto W de operações sobre vectores booleanos. O número f de operações é fixo mas os tipos de operações podem ser arbitrários (dentro duns determinados limites). Assumimos que f=1. Neste caso uma única (f=1) operação pode ser: "verificar se um vector só contém um 1 e encontrar a posição deste 1", "contar o número de 1s" ou qualquer outra. De acordo com a especificação da tarefa cada vector V tem o tamanho h e $1 \le h \le 24$. Os alunos devem estar aptos a realizar as operações particulares (i.e. qualquer conjunto W com o número fixo f dos elementos) sem modificações físicas em hardware (i.e. é necessário desenvolver um circuito reconfigurável). Um requisito adicional é que o circuito deve ser reconfigurável estática e dinamicamente.

A fig. 1 mostra a especificação geral do problema que inclui as duas partes seguintes :

1. Desenvolvimento e implementação do hardware com base em FPGA XC4010XL que faz parte da placa XS40. A placa é ligada a um computador PC através da porta paralela e pode receber 8 bits do PC e mandar 4 bits para o PC. O hardware permite resolver as três seguintes tarefas:

1.1. Comunicar com o PC através da porta paralela;

1.2. Alterar o comportamento da unidade de controlo reprogramável (UCR) de acordo com a informação recebida do PC. Isto permite modificar os algoritmos de controlo e implementar operações diferentes sobre os vectores booleanos;

1.3. Receber e gravar o vector booleano a partir do PC, efectuar operações sobre vectores de acordo com o algoritmo especificado para a UCR e gravar o resultado no registo que pode ser lido pelo PC.

2. Desenvolvimento do software com base na linguagem C++. O software permite comunicar com a FPGA através da porta paralela e resolver as duas seguintes tarefas:

2.1. Alteração do comportamento da UCR;

2.2. Comunicação com o hardware (com a FPGA) que permite: a) especificar um vector booleano com qualquer tamanho h de 1 a 24 bits; iniciar uma operação sobre o vector em hardware (em FPGA); receber e visualizar o resultado da operação.

A especificação básica (ver fig. 1) teve várias modificações para cada grupo de alunos (ver secção II). Por exemplo, um conjunto de operações concretas foi proposto. Para resolver o problema é permitido utilizar todas as ferramentas e bibliotecas disponíveis na "Xilinx Foundation Software". Os artigos [1-4] descrevem cada um dos quatro projectos desenvolvidos em 2002 pelos alunos do quarto ano da LECT.

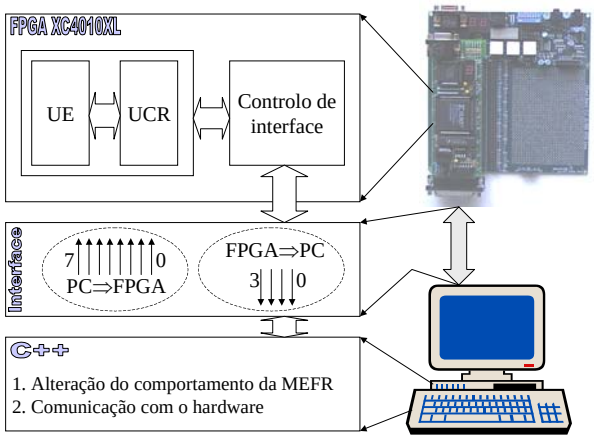

Figure 1. Especificação do problema

II. REQUISITOS BÁSICOS AO HARDWARE

Para implementar a parte de hardware os alunos utilizaram a placa XS40 da XESS [11] que inclui uma FPGA XC4010XL da Xilinx [12]. Obviamente, outros tipos de FPGAs (por exemplo, [13]) podem ser usados praticamente sem modificações na especificação do projecto.

O hardware inclui duas partes principais que são o circuito que permite implementar várias operações sobre os vectores (ver fig. 2) e a interface com o computador (ver fig. 3).

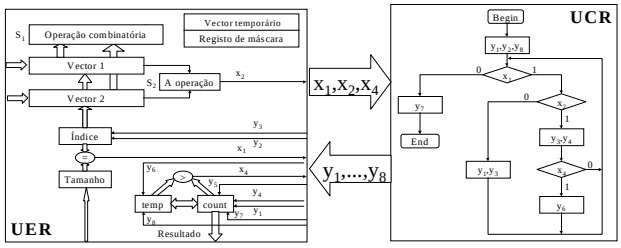

Figure 2. O circuito que permite implementar várias operações sobre os vectores

O circuito na fig. 2 permite implementar operações quer sobre um quer sobre dois vectores (todos os detalhes foram considerados em [5,14]). Os vectores podem ser quer binários, quer ternários. Os circuitos S_1 e S_2 realizam várias operações ou sobre os vectores (S_1) ou sobre bits dos vectores que têm o mesmo índice (S_2). Estes circuitos podem ser implementados com base em RAM [8,9] que permite alterar as operações estática ou dinamicamente. Então a UE torna-se reprogramável (UER). Para simplificação da tarefa os alunos consideraram um só vector que pode ter qualquer tamanho de 1 a 24 bits. É permitido desenvolver a UE com a ajuda de qualquer ferramenta da "Xilinx Foundation Software" tal como:

- a linguagem VHDL e as respectivas ferramentas para modelar, sintetizar e verificar o circuito;

- o editor esquemático e a biblioteca de circuitos digitais;
- a ferramenta "LogiBLOX module generator";
- a ferramenta "Core generator", etc.

A alteração da operação que pode ser executada pela UE é efectuada através de modificação do respectivo algoritmo implementado na UCR (ver fig. 2). Todos os detalhes serão abordados na secção III.

A fig. 3 mostra a especificação básica do circuito que implementa a interface com o PC. A placa XS40 permita receber 8 bits do PC e enviar 4 bits para o computador PC. Estes bits têm de ser utilizados para:

- fazer a sincronização entre a placa e o computador PC;
- receber o vector para executar a respectiva operação;
- receber o tamanho do vector;
- reprogramar o comportamento da máquina de estados finitos reprogramável (MEFR) que é considerada como o modelo da UCR. Isto permite alterar o algoritmo implementado na UCR e finalmente modificar a respectiva operação a executar sobre o vector. Todos os detalhes de reprogramação vão ser abordados na secção III;
- reprogramar alguns componentes da UER o que permite alterar a funcionalidade da UER.
- receber o resultado da operação.

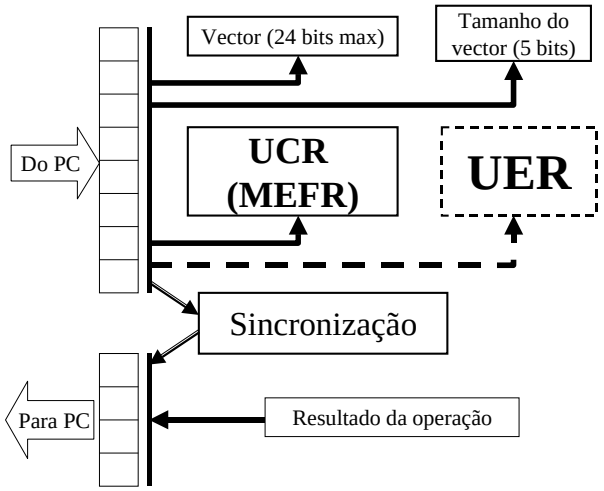

Figure 3. A estrutura de interface entre o computador e a placa

Finalmente os blocos UCR e UER têm circuitos permanentes (i.e. não alteráveis do ponto de vista do hardware), mas a funcionalidade destes circuitos pode ser modificada através de programação (ou configuração) a partir do computador (ou qualquer outro sistema externo). Tais blocos chamam-se "hardware templates" [5,14].

Para realizar o projecto considerado acima os alunos têm que ser capazes de desenvolver todos os componentes de interface. Em 2002 (2º semestre) só um componente que é responsável pela alteração da funcionalidade da UER, não foi implementado.

III. UNIDADE DE CONTROLO REPROGRAMÁVEL

Existem vários métodos que permitem desenvolver unidades de controlo reprogramáveis [14]. Os alunos utilizaram o método apresentado em [5]. O modelo da UCR é uma MEFR que é composta por um registo para guardar o estado e um conjunto de blocos RAM considerados em [5]. De acordo com [5] a alteração do comportamento da MEFR pode ser feita através das duas possibilidades seguintes :

- troca de segmentos dos blocos RAM;
- reprogramação dos blocos RAM.

Os artigos [5,14] ilustram a primeira possibilidade em detalhe. A segunda possibilidade é demostrada em circuito desenvolvido em Xilinx Foundation Software (ver fig. 4). A MEFR inclui dois níveis [5] sendo cada um deles composto por blocos STRAM (*state transition* RAM), que implementam as transições entre os estados da MEFR, e PM (*programmable multiplexer*) - REP_MUL que é um multiplexador reprogramável. O bloco OR_MOORE permite gerar as saídas da MEFR para o modelo de Moore, i.e. este bloco gera os sinais de saída $y_1,...,y_N$ para cada estado da MEFR. Então, para alterar o comportamento da MEFR (i.e. para implementar um novo algoritmo de controlo da UCR) é necessário reprogramar a memória para os blocos OR_MOORE (H31), STRAM1 (H33), STRAM2 (H34), REP_MUL (U3) e REP_MUL (U32). A memória de cada bloco é construída como "*dual port RAM*". A primeira porta da memória é utilizada para o funcionamento normal, i.e. para efectuar as transições e gerar saídas. A segunda porta serve para reprogramar a memória, i.e. para modificar os dados escritos nos endereços respectivos. A reprogramação pode ser feita com a ajuda do circuito da fig. 3 que executa os seguintes passos:

- recebe a informação a partir do PC para reprogramação, i.e. recebe o nome do bloco RAM considerado acima, o endereço para este bloco e os dados que devem ser escritos no endereço;
- escreve os dados novos no respectivo bloco;
- repete o primeiro e o segundo passos até que todos os dados de que precisamos sejam recebidos.

O descodificador DECODE na fig. 4 recebe o código SELECT que indica o nome do bloco e selecciona este bloco (ver linhas D_OUT). Depois a informação nas linhas DATA_REP escreve-se no bloco seleccionado de acordo com o endereço nas linhas ADR_REP (ver fig. 4). Para a maioria dos algoritmos de controlo práticos, os dois níveis considerados para a MEFR na fig. 4 não são suficientes. Normalmente o circuito deve incluir três ou quatro níveis, i.e. temos de reprogramar 7 (para três níveis) ou 9 (para quatro níveis) blocos de memória. Por exemplo, para o algoritmo na fig. 2 (ver fig. 5) devemos

utilizar os três níveis. As etiquetas a_0,...,a_5 são associadas com os respectivos estados da MEFR. De notar que a MEFR pode ser implementada em circuito com só dois níveis mas neste caso é necessário incluir uma etiqueta adicional (ver a etiqueta a_6 na fig. 5). Todos os detalhes foram apresentados em [5,14]. Para simplificação vamos assumir que o código $K(a_m)$ de cada estado a_m é igual ao índice m deste estado, i.e. $K(a_0) = 000$, $K(a_1) = 001$, ... Neste caso o bloco OR_MOORE deve ser programado da seguinte forma:

$000 \Rightarrow 00000000$; $001 \Rightarrow 10000011$; $010 \Rightarrow 00100000$; $011 \Rightarrow 00000101$; $100 \Rightarrow 00001100$; $101 \Rightarrow 01000000$

(ver [5] para detalhes). Vamos assumir que no estado a_4 as saídas y_3, y_4 devem ser alteradas para y_2, y_5, y_6, y_8. Neste caso os dados 00001100 escritos no endereço 100 devem ser alterados para 10110010. O bit do lado esquerdo mostra o valor da saída y_8 e o bit do lado direito mostra o valor da sadia y_1.

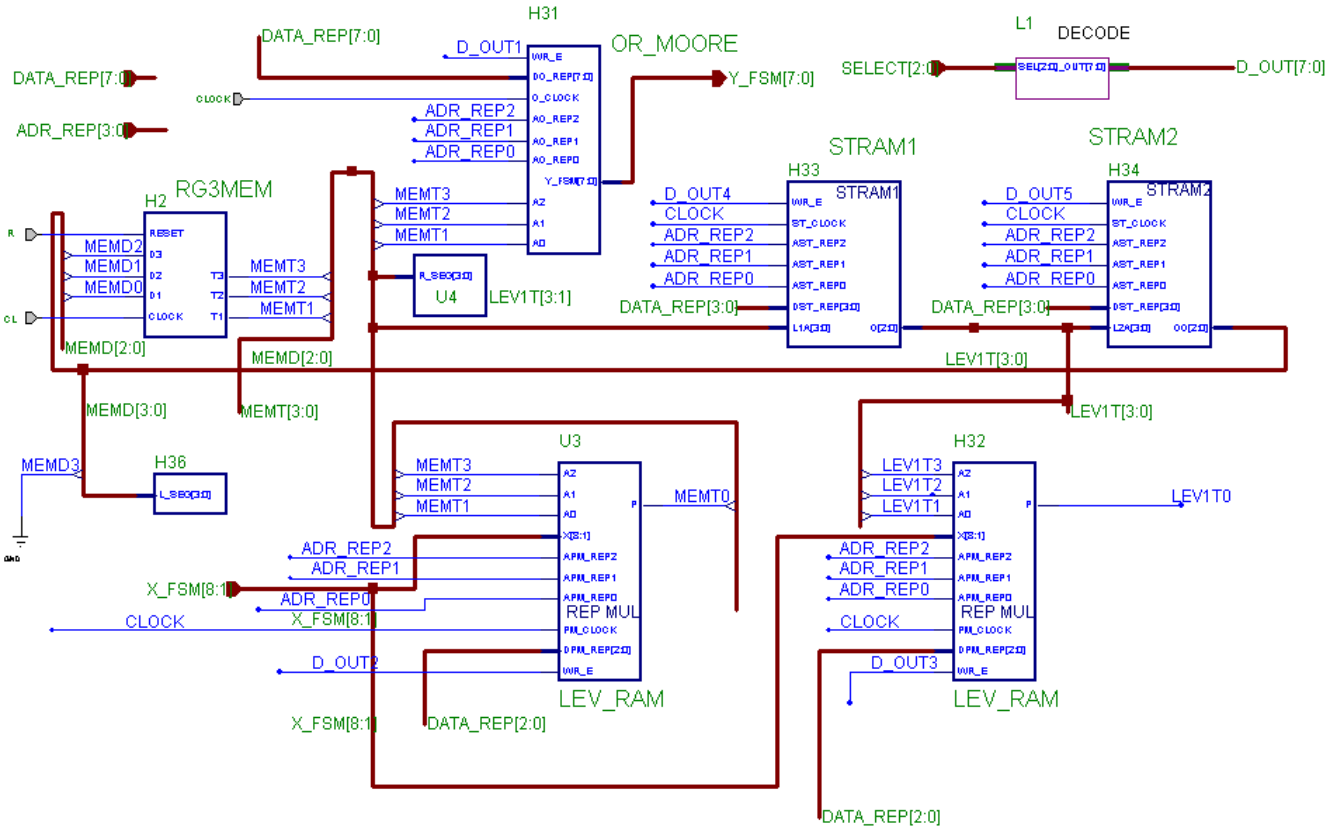

Figure 4. Uma máquina de estados finitos reprogramável através de recarregação dos blocos RAM (os blocos U4 e H36 são auxiliares que são utilizados para visualizar o estado actual e o estado próximo da MEFR)

Para reprogramar os outros blocos de memória podemos utilizar métodos considerados em [5] sem quaisquer modificações. As variáveis x_1, x_2, x_4 na fig. 5 apresentam os valores das entradas da UCR (MEFR) e as variáveis y_1,...,y_8 apresentam as saídas da UCR. As implementações particulares da MEFR são consideradas em [1-4].

IV. REQUISITOS BÁSICOS AO SOFTWARE

Para implementar a parte de software os alunos utilizaram a linguagem de programação Visual C++. Fig. 6 mostra os objectivos e a organização de software. Comunicação com a FPGA é realizada através das funções *DlPortWritePortUchar* e *DlPortReadPortUchar* que fazem parte da biblioteca API respectiva. A primeira função envia oito bits (i.e. um byte) com os dados para a placa XS40 (i.e. para a FPGA). A segunda função recebe quatro bits da placa XS40 (i.e. da FPGA). Os códigos das funções são apresentados na fig. 6. As linhas *value &= 0x78; value >>= 3;* permitem escolher os quatro bits apropriados. De notar que os endereços da porta respectiva *0x378* e *0x379* (ver as linhas *ULONG port = 0x0378;* e *ULONG port = 0x0379;*) são dependentes da selecção (LPT1, LPT2) e instalação das portas e por isso podem ter outros valores.

O software deve estar apto a gerar os dados necessários para o hardware implementado com base em FPGA. A informação inicial (tal como o vector do tamanho respectivo, o algoritmo de controlo para executar a operação de que precisamos, etc.) pode ser preparada pelo utilizador e quer especificada a partir do teclado quer lida

de um ficheiro gravado anteriormente. O software deve converter esta informação numa forma percebível por hardware e transferir os dados respectivos para a placa. Depois da operação tiver sido executada o software deve estar apto a receber o resultado da operação e visualizá-lo no ecrã.

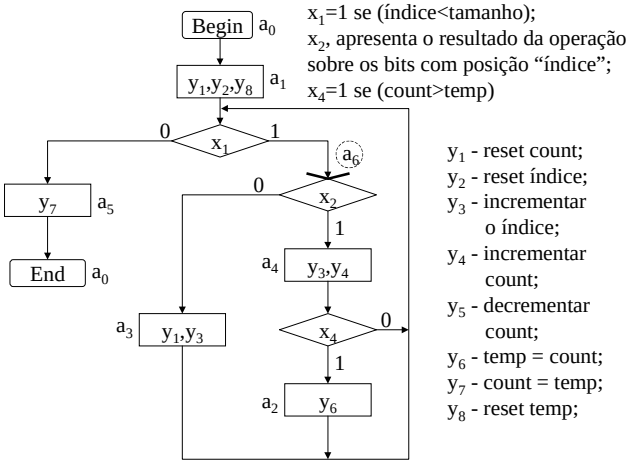

Figure 5. O algoritmo que permite encontrar o número máximo de uns sucessívos num vector binário

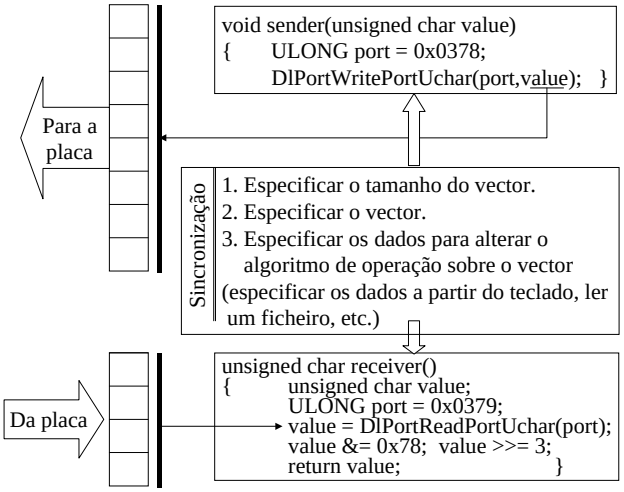

Figure 6. Os objectivos e a organização de software

Para implementar os passos considerados acima é necessário assegurar a sincronização própria entre a placa e o computador. Por outros palavras a placa deve saber:

- quando os dados do computador estão disponíveis para leitura nas oito linhas da porta paralela (ver fig. 6);
- quando os dados disponibilizados para o computador nas quatro linhas da porta paralela (ver fig. 6) podem ser modificados.

O computador deve saber:

- quando os dados da placa estão disponíveis para a leitura nas quatro linhas da porta paralela (ver fig. 6);
- quando os dados gerados para a placa nas oito linhas da porta paralela (ver fig. 6) podem ser alterados.

O problema da sincronização pode ser resolvido com a ajuda dos métodos seguintes:

separação dos bits da porta em duas partes que são usadas para a sincronização e para fornecer a informação principal que deve ser transmitida, respectivamente;

implementação dum protocolo específico de comunicação que inclui os dados com alguns comandos e os dados com a informação principal.

Cada projecto pode utilizar qualquer método e a realização particular está apresentada em [1-4].

V. FERRAMENTAS (XILINX FOUNDATION SOFTWARE)

Xilinx Foundation Software inclui vários componentes que permitem especificar, desenvolver, simular, verificar e implementar circuitos digitais com base em FPGA de Xilinx. Nem todos os componentes têm de ser utilizados para o projecto considerado acima. Por outro lado um conjunto de componentes foi seleccionado e proposto aos alunos para uso obrigatório. Estes componentes são os seguintes:

- Editor esquemático;
- Linguagem VHDL;
- Sintetizador de circuitos com base em VHDL;
- Geradores de módulos, tais como "*LogiBLOX module generator*" ou "*Core generator*";
- Editor dos diagramas de máquinas de estados finitos (MEF);
- Simulador;
- Bibliotecas de Xilinx e as construídas pelo utilizador;
- Sintetizador de circuitos hierárquicos compostos por todos os meios de especificação considerados acima, i.e. blocos esquemáticos, componentes de várias bibliotecas, blocos descritos em VHDL, diagramas de MEF e módulos gerados automaticamente.

Todos os outros componentes de Xilinx Foundation Software são de uso opcional.

Os alunos devem estar aptos a utilizar os métodos seguintes :

- desenvolvimento de circuitos hierárquicos com base em abordagem "*top-down*" (i.e. de cima para baixo), "*bottom-up*" (i.e. de baixo para cima) e numa mistura destas;
- simulação do circuito e de todos os componentes do circuito necessários em software utilizando o simulador que faz parte da Xilinx Foundation Software;
- implementação do circuito físico com base em FPGA e a sua verificação em hardware com a ajuda de programas respectivos (tais como *GXSTEST*, *GXSLOAD* e *GXSPORT*) e do software desenvolvido pelos alunos e implementado num computador de uso geral (tal como o computador PC);
- reconfiguração estática e dinâmica do circuito utilizando os métodos considerados acima;

- desenvolvimento e utilização de alguns circuitos adicionais para depurar e testar todos os componentes do projecto em hardware.

VI. PERSONALIZAÇÃO DO PROJECTO

A especificação do projecto é comum para todos os grupos dos alunos. Por outro lado cada projecto tem que ser individual e por isso assume uma personalização que foi considerada para as partes seguintes:

- software desenvolvido para o computador PC;
- interface entre o computador e a placa;
- implementação de hardware com base em FPGA;
- métodos e arquitecturas de circuitos para reconfiguração.

O software deve estar apto a resolver o problema geral considerado na secção IV. Não existem quaisquer outros requisitos específicos excepto que o software de um grupo não pode ser semelhante ao software de outro grupo. Esta última regra é comum para todos os pontos apresentados acima. Cada projecto utiliza os seguintes componentes únicos de software que devem ser discutidos com o professor:

- sincronização com o hardware;
- codificação dos dados, tais como a informação para especificar o algoritmo de operação que deve ser executada pela MEFR;
- interface externa com o utilizador, i.e. como especificar os dados de entrada (vector, o seu tamanho e algoritmo) e como visualizar o resultado da operação sobre o vector;
- componentes adicionais que podem ser úteis para testar o circuito implementado em FPGA.

Interface entre o computador e a placa requer a especificação do protocolo de troca de dados e a identificação de cada linha para interligar o computador e a placa (ver fig. 3) , i.e. é necessário definir como utilizar os sinais de cada linha em hardware e como estes sinais podem ser alterados dependente das sequências de palavras entre o computador e a placa.

O hardware pode ser desenvolvido com base em vários métodos, tais como:

- especificação de qualquer parte pode ser feita com base em VHDL, editor esquemático, etc.;
- um bloco de hardware pode ser descrito completamente em VHDL ou pode ser decomposto em sub-blocos tais como sub-blocos descritos a nível das transições entre registros - RTL (*register-transfer level*) e uma MEF (ou um conjunto de MEFs). Mais uma vez, para cada sub-bloco podemos utilizar vários métodos de especificação, etc.;
- uma MEF pode ser implementada com base em vários métodos (ver [14] para detalhes);
- método de reconfiguração pode ser diferente (ver [14]) e o hardware deve ser orientado ao método seleccionado;
- a UE que efectua as operações sobre o vector pode ter várias arquitecturas. Esta unidade pode ser quer reconfigurável quer não, pode realizar operações sobre um vector ou vários vectores, o vector pode ser binário, ternário ou de outro tipo, etc.

Os pontos considerados acima podem ser detalhados para cada grupo de alunos e de facto existe um número infinito das variantes possíveis.

Do ponte de vista de reconfiguração existem também várias possibilidades. Estas foram consideradas parcialmente na secção III e nos artigos [5,14] em detalhe.

VII. CONCLUSÕES

Este artigo apresenta uma metodologia que foi utilizada no âmbito da disciplina "Computação Reconfigurável" para os alunos do quarto ano lectivo da especialização LECT (Licenciatura em Engenharia de Computadores e Telemática). Esta metodologia permite aprender como especificar, desenvolver, verificar e implementar circuitos com base em novos métodos, potencialidades e arquitecturas de sistemas digitais, tais como reconfiguração estática e dinâmica, interligação entre os sistemas implementados em hardware e software, utilização de "*hardware templates*", etc. Os alunos ganharam a experiência com as FPGAs fabricadas na indústria e sistemas assistidos por computador (*CAD - computer-aided design systems*), tais como *Xilinx Foundation Software*. Todos os projectos foram implementados pelos alunos e apresentados nos quatro artigos seguintes [1-4]. Estes mostram as realizações particulares com a respectiva personalização de cada projecto.

Este artigo continua uma série de artigos publicados na revista "Electrónica e Telecomunicações" e dedicados à área de desenvolvimento de sistemas digitais incluíndo alguns tópicos avançados. A seguir está apresentada uma lista completa de artigos publicados em português que podem ser úteis para os alunos das especializações LECT, LEET (Licenciatura em Engenharia Electrónica e Telecomunicações), LEEC (Licenciatura em Engenharia Electrotécnica e de Computadores), etc.

- os artigos [1-4] apresentam os resultados concretos dos projectos especificados acima;
- o artigo [5] apresenta em detalhe como utilizar os *hardware templates* para desenvolver circuitos reprogramáveis;
- o artigo [15] demostra como utilizar computação reconfigurável para implementar os algoritmos evolucionais;
- os artigos [16,17] consideram grafos de algoritmos hierárquicos e um modelo de máquina de estados finitos hierárquica.
- os artigos [18,19] apresentam uma linguagem gráfica para descrição de vários algoritmos de controlo em sistemas digitais.
- o artigo [20] descreve um modelo de interligação de hardware (FPGA) e software. O software representa uma interface visual que permite mostrar todos os movimentos dum plotter. O hardware controla os

movimentos e efectua as respectivas operações através da porta paralela. O artigo [21] (em inglês) descreve outras aplicações do mesmo tipo.

- os artigos [22,23,24] descrevem vários métodos de modelação e desenvolvimento de circuitos digitais com base nas FPGAs reconfiguráveis dinamicamente.
- o artigo [25] modifica a linguagem C++ para permitir simular algumas arquitecturas específicas tais como as utilizadas para os controladores CAN.
- o artigo [6] descreve vários problemas da área de optimização combinatória que podem ser formulados sobre matrizes binárias e ternárias.
- os artigos [26,27] apresentam os resultados da implementação do processador MIPS com base na FPGA XC4010XL de Xilinx.
- os artigos [7,28] descrevem um problema combinatório muito importante que se chama "Satisfação booleana". De notar que o artigo [7] contém uma lista bastante extensa da bibliografia nesta área, incluíndo a utilização de vários recursos de computação reconfigurável.
- A informação adicional para os alunos do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro, pode ser encontrada na WebCT [29-32].

AGRADECIMENTOS

O autor agradece ao Professor António de Brito Ferrari pela ajuda prestada na elaboração deste artigo.

REFERÊNCIAS

[1] J. Santos, N. Duarte, Síntese e Implementação de Circuitos Digitais Recinfiguráveis Dinamicamente (Projecto 1). Electrónica e Telecomunicações (E&T), Vol. 3, Nº 8, Jan. 2003 (nesta revista).

[2] D.Gomes, N.Carvalho, Síntese e Implementação de Circuitos Digitais Recinfiguráveis Dinamicamente (Projecto 2). E&T, Vol. 3, Nº 8, Jan. 2003 (nesta revista).

[3] C.Teixeira, J.Girão, Síntese e Implementação de Circuitos Digitais Recinfiguráveis Dinamicamente (Projecto 3). E&T, Vol. 3, Nº 8, Jan. 2003 (nesta revista).

[4] J.P.Barraca, N.J.Sénica, Síntese e Implementação de Circuitos Digitais Recinfiguráveis Dinamicamente (Projecto 4). E&T, Vol. 3, Nº 8, Jan. 2003 (nesta revista).

[5] V. Sklyarov, Modelação em C++, Síntese e Implementação de Circuitos Digitais com base em FPGA. E&T, Vol. 3, Nº 8, Jan. 2003 (nesta revista).

[6] I.Skliarova, A.Ferrari, Modelos matemáticos e problemas de optimização combinatória. E&T, Jan., Vol. 3, Nº 3, 2001, pp. 202-208.

[7] I.Skliarova, A.B.Ferrari, Satisfação booleana: algoritmos, aplicações, implementações, E&T, Jan., Vol. 3, Nº 5, 2002, pp. 400-408.

[8] I.Skliarova, A.B.Ferrari, Design and Implementation of Reconfigurable Processor for Problems of Combinatorial Computations, Proceedings of the Euromicro Symposium DSD'2001, Warsaw, Poland, September 2001, pp.112-119.

[9] I.Skliarova, A.Ferrari, Exploiting FPGA-Based Architectures and Design Tools for Problems of Combinatorial Computations. Proceeding of SBCCI'2000, Manaus, Brazil, 2000, pp. 347-352.

[10] I.Skliarova, A.B.Ferrari, A SAT Solver Using Software and Reconfigurable Hardware. Proc. of the Design, Automation and Test in Europe Conference (DATE), March, 2002, Paris, France.

[11] XStend board V1.2 Manual., XESS Corporation, 1998, 68 pages.

[12] Xilinx, The programmable logic data book, Xilinx, San Jose, 2000.

[13] Virtex™-E 1.8 V Extended Memory Field Programmable Gate Arrays, 2000, www.xilinx.com.

[14] V. Sklyarov, Reconfigurable models of finite state machines and their implementation in FPGAs. Journal of Systems Architecture, 47, 2002, pp. 1043-1064.

[15] I.Skliarova, A.Ferrari, Utilização do hardware reconfigurável para acelerar algoritmos evolutivos: o caso do problema do caixeiro viajante. E&T, vol. 3, no. 7, pp. 613-620, 2002.

[16] V.Sklyarov, António Adrego da Rocha, Sintese de Unidades de Controlo Descritas por Grafos dum Esquema Hierárquicos, Electrónica e Telecomunicalções, vol. 1, no. 6, pp. 577-588, 1996.

[17] António Adrego da Rocha, V.Sklyarov, Simulação em VHDL de Máquinas de Estados Finitas Hierárquicas. E&T, 1997, Vol. 2, N 1, pp. 83-94.

[18] A.Melo, V.Sklyarov, A.Ferrari, HiParaGraphs, uma Linguagem de Especificação de Algoritmos de Controlo Paralelos e Hierárquicos. E&T, Jan., Vol. 3, Nº 3, 2001, pp. 214-221.

[19] A.Melo, V.Sklyarov, A.Ferrari, Especificação e Simulação Interactiva de Algoritmos de Controlo Paralelos e Hierárquicos. E&T,, Sep., Vol. 3, Nº 4, 2001, pp. 332-344.

[20] V.Silva, F.Santos, V.Sklyarov, A interligação do Visual C++ com as FPGAs da XILINX. E&T, Jan., Vol. 2, Nº7, 2000, pp. 874-883.

[21] V.Sklyarov, Hardware/Software Modeling of FPGA-based Systems. Parallel Algorithms and Applications, ISSN 1063-7192, Volume 17, Number 1 (2002), pp. 19-39.

[22] V.Sklyarov, A.Melo, A.Oliveira, N.Lau, Ricardo Sal Monteiro, Circuitos Virtuais Baseados em Reprogramação e Reconfiguração Dinâmica. E&T, 1998, Vol. 2, N 2, pp. 248-260.

[23] A.Melo, V.Sklyarov, Ambiente Integrado para Especificação, Projecto e Verificação de Unidades de Controlo em FPGAs, E&T, Jan., Vol. 2, Nº4, 1999, pp. 477-485.

[24] A.Oliveira, V.Sklyarov, Especificação, Projecto e Implementação de Circuitos de Controlo Virtuais, E&T, Jan., Vol. 2, Nº4, 1999, pp. 487-495.

[25] A.Oliveira, V.Sklyarov, A.Ferrari, EaSys - Uma Linguagem Orientada por Objectos para Descrição de Sistemas Digitais. E&T Sep., Vol. 3, Nº 4, 2001, pp. 311-331.

[26] I.Skliarova, A.B.Ferrari, Implementação e Simulação do Processador MIPS com a ALU reconfigurável dinamicamente, E&T, v.2, Nº3, January 1999, pp. 497-504.

[27] I.Skliarova, A.B.Ferrari, Projecto e Implementação de um subconjunto da arquitectura MIPS16 com base em FPGA XC4010XL, E&T, v.2, Nº6, September 1999, pp. 724-732.

[28] I.Skliarova, A.B.Ferrari, Utilização de hardware reconfigurável para acelerar a satisfação booleana, E&T, September 2001, pp. 304-310.

[29] Página *http://webct.ua.pt*, "1º Semestre", a disciplina "Programação por Objectos".

[30] Página *http://webct.ua.pt*, "1º Semestre" a disciplina "Sistemas Digitais Avançados".

[31] Página *http://webct.ua.pt*, "1º Semestre", a disciplina "Paradigmas de Programação *I*".

[32] Página *http://webct.ua.pt*, "2º Semestre", a disciplina "Computação Reconfigurável".