Projecto e Implementação de um subconjunto da arquitectura MIPS16 com base em FPGA XC4010XL

Iouliia Skliarova, António B.Ferrari

***Resumo* – Este artigo descreve o projecto de um processador com a arquitectura MIPS com base em FPGA XC4010XL e apresenta as ferramentas de software desenvolvidas que permitem analizar os circuitos digitais construídos em FPGA, implementar o conjunto de instruções desejado para o processador e trabalhar com este num modo interactivo. O projecto é baseado em ferramentas e bibliotecas do Xilinx Foundation Series 1.5 Software. Os circuitos finais foram implementados na FPGA XC4010XL instalada na placa XS40 da XESS. Todas as experiências foram efectuadas com as placas XS40 e XStend ligadas ao computador através da porta paralela. Pretende-se que o núcleo de processador desenvolvido possa vir a ser usado eficientemente em aplicações *embedded*. O software desenvolvido inclui um micro assembler para o processador com um conjunto de instruções facilmente modificável, ferramentas gráficas que permitem analizar vários circuitos do processador e uma interface amigável ao utilizador que suporta o modo interactivo.**

***Abstract* – The paper considers the design of a proccessor with MIPS16 architecture on the base of FPGA XC4010XL and presents the developed software tools that allow to analyse the constructed digital circuits in FPGA, to implement the desired set of instructions for the processor, and to work with the processor in interactive mode. The design flow is based on tools and libraries of the commercially available Xilinx Foundation Series 1.5 Software. The final circuits have been implemented in FPGA XC4010XL to be installed on the XS40 board of XESS. The experiments have been performed with XS40 and XStend boards connected to a PC computer via the parallel port. We expect that the designed processor core migth be efficiently used in embedded applications. The developed software includes a micro assembler for the processor with a flexibly modifiable set of instructions, graphical tools enabling to analyse different circuits of the processor and a user-friendly interface supporting interactive mode.**

I. INTRODUCÃO

O desenvolvimento da tecnologia de componentes programáveis, em particular das FPGAs (Field Programmable Gate Arrays), levou ao aparecimento de uma nova área designada *computação reconfigurável*, que investiga a interligação das FPGAs com a tecnologia tradicionalmente utilizada na concepção dos sistemas computacionais. Neste sentido já foram obtidos uns resultados bastante promissores. A vantagem principal desta tecnologia está na possibilidade de construção de dispositivos computacionais e de controlo, optimizados para aplicações específicas, sem a necessidade de desenvolvimento de novos componentes físicos. A alteração das funções de hardware é efectuada por via de reprogramação das funções de células lógicas básicas das FPGAs e das ligações entre estas. Tais dispositivos computacionais e de controlo podem ser utilizados com sucesso em sistemas *embedded*.

Tradicionalmente os sistemas com processadores incorporados, sejam eles "microcontroladores" ou "*embedded microprocessors*", utilizam arquitecturas de 4 ou 8 bits, estas ainda actualmente dominantes em termos de número de unidades. O aparecimento de novos mercados e produtos com requisitos de processamento mais exigentes, leva a que as arquitecturas de 16 bits tenham conquistado uma significativa fatia de mercado e que as de 32 bits sejam aquelas cuja utilização em sistemas *embedded* se encontra actualmente em mais rápida expansão [1,2].

Neste artigo é considerado um processador com a arquitectura MIPS [3,4], que pertence ao grupo das arquitecturas RISC (*Reduced Instruction Set Computers*). Este grupo, ao contrário do grupo CISC (*Complex Instruction Set Computers*), distingue-se pela relativa simplicidade do design e pelo desempenho elevado, mas, em geral, exige um número de instruções maior para a execução de um certo trabalho (programa). Repare-se que algumas das instruções simples, em princípio, poderiam ser representados por um número menor de bits e, consequentemente, uma certa quantidade da memória do programa é usada ineficientemente. Esta desvantagem destaca-se particularmente em sistemas *embedded* que possuem pequenos *on-chip instructions caches*. A extensão da arquitectura MIPS – a MIPS16 resolve o problema codificando instruções em 16 bits [1,2]. MIPS16 é totalmente compatível com as arquitecturas existentes de 32 bits (MIPS - I/II) e de 64 bits (MIPS - III) [1].

Como dispositivo base para implementar o processador foi utilizada a FPGA XC4010XL da Xilinx, que pertence à família XC4000XL. As FPGAs desta família incluem dois tipos de elementos principais: CLBs (Configurable Logic Blocks) e IOBs (Input/Output Blocks). Os CLBs servem para implementar a lógica do utilizador e os IOBs

asseguram a interface entre os pinos da FPGA e os seus sinais internos. Todos os CLBs são interligados entre si com a ajuda das PSMs (Programmable Switch Matrices) [5,6]. Para guardar todas as ligações utilizam-se células da memória estática interna. A FPGA XC4010XL possui 400 CLBs, organizados num array de 20*20. A estrutura básica de um CLB está apresentada na fig. 1 [5]. Este tem 2 geradores de funções de 4 entradas (*F* e *G* na fig. 1) e um de 3 entradas (*H* na fig. 1). O *H* recebe sinais das saídas do *F* e/ou *G* ou do exterior do CLB. Portanto um CLB é capaz de executar funções que têm até 9 variáveis [5]. Os geradores de funções são implementados como LUTs (look-up tables), utilizando RAM. Cada CLB contém também 2 flipflops que podem guardar as saídas dos *F*, *G* e *H* ou ser utilizados independentemente. É possível configurar *F* e *G* como RAMs de 16x2, 32x1 ou 16x1 bits, dependendo do modo (*level-sensitive*, *edge-triggered* ou *dual-port edge-triggered*). O conteúdo destas RAMs carrega-se durante a fase de configuração da FPGA e não é afectado pelo *reset* global. É de notar que a FPGA XC4010XL é reconfigurável estaticamente (à distinção das FPGAs reconfiguráveis dinamicamente da família XC6200 [4]). Para a programação do conteúdo das LUTs e das ligações entre os CLBs foram utilizados ficheiros de configuração gerados pelo Xilinx Foundation Series 1.5 Software.

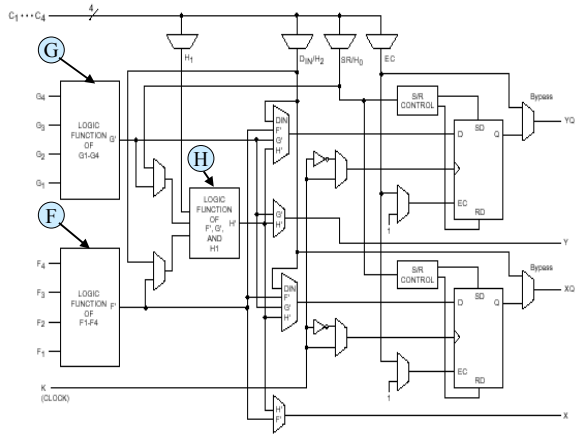

Fig. 1 – Estrutura básica do CLB

Nos dias de hoje a complexidade das FPGAs está a crescer constantaneamente. P. ex., a FPGA XC40250XV tem 20 vezes mais CLBs do que a XC4010XL, e possui também melhores recursos de encaminhamento de ligações [5]. Isto permite implementar numa FPGA vários processadores que interagem entre si e com os outros objectos possíveis. Sendo assim, pode-se modificar cada processador de maneira a optimizá-lo (orientá-lo para a resolução de um determinado grupo de tarefas). Como vários processadores trabalham dentro de um dispositivo (FPGA), é possível optimizar a velocidade da sua interacção, os custos de projecto, simplificar a modificação, depuração, etc.

II. SISTEMA DE DESENVOLVIMENTO XILINX

Para o projecto lógico de circuitos que se pretende implementar em FPGAs, pode-se usar o sistema Xilinx Foundation Series 1.5 [6]. Este sistema aceita como entradas três tipos possíveis de especificações (fig. 2), que podem ser utilizados separadamente ou em qualquer combinação.

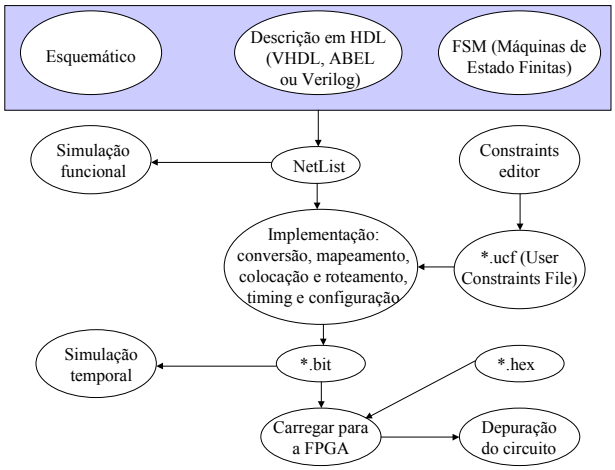

Fig. 2 - Fases principais de desenvolvimento de um circuito digital no Xilinx Foundation Series 1.5 Software

O resultado final dos editores de esquemático, de HDL (Hardware Description Language) e de FSM (Finite State Machine) é uma *NetList* (lista das portas lógicas e das interligações entre estas). O sistema possui um simulador funcional que permite verificar a lógica do circuito. A seguir, o software compila a *NetList* e gera um *bitstream* utilizado para a programação da FPGA. Em geral, nesta fase são programados CLBs e PSMs arbitrários de maneira a minimizar quer os atrasos de sinais quer a área ocupada pelo circuito. O utilizador pode com a ajuda de um ficheiro *.ucf (User Constraints File) obrigar a utilização de determinados CLBs e também limitar os atrasos máximos entre áreas do seu circuito. Nesta fase já são consideradas as particularidades de uma FPGA concreta portanto o sistema é capaz de determinar todos os atrasos das portas lógicas e das ligações entre estas o que permite efectuar uma simulação temporal. Para carregar o *bitstream* na FPGA pode ser utilizado o programa XSLOAD [7], que recebe como parâmetros um ficheiro *.bit (bitstream) e, caso seja necessário, um ficheiro *.hex utilizado para a programação da SRAM externa. A última fase (fig. 2) é a depuração do circuito em FPGA.

Para todas as experiências utilizou-se a placa XStend à qual foi ligada a placa XS40 [7]. Esta contém uma FPGA XC4010XL, 32Kx8 de SRAM, um LED de 7 segmentos e os necessários circuitos de sincronização, da interface com a porta paralela do computador e de fonte de alimentação. A placa XStend contém 2 botões de uso geral, 8 DIP switches, 10 LED indicadores e 2 LEDs de 7 segmentos [7]. Os dados de entrada para o circuito digital

implementado são fornecidos através quer da porta paralela do computador quer dos botões montados na placa. A resposta do circuito pode ser observada nos LEDs ou lida da porta paralela do computador.

III. PROJECTO DO DATAPATH

No nível superior do projecto (fig. 3) representa-se o processador considerado em forma de uma combinação tradicional do datapath e do circuito de controlo. O datapath contém 8 registos de 32 bits que guardam dados de entrada/saída e os resultados intermédios, uma ALU e circuitos auxiliares. A unidade de controlo recebe os códigos das instruções MIPS16 e gera uma sequência de sinais elementares para a execução destas instruções no datapath.

Cada um dos blocos (fig. 3) foi descrito em linguagem VHDL [8], especificado em forma de uma FSM ou construído dos elementos da biblioteca XC4000 da Xilinx (descrição esquemática). Consideremos em mais pormenor como se pode especificar um circuito digital utilizando cada um destes métodos.

A.1. Definição de circuitos em VHDL

Para ilustrar este método consideremos a especificação de um descodificador para um display de 7 segmentos (bloco *BIT2LED* na fig. 3). Este bloco recebe como entrada um sinal de 4 bits e transforma-o num número hexadecimal que pode ser visualizado num LED de 7 segmentos, i.e. este bloco é utilizado para os fins de depuração do circuito. Como as placas Xstend e XS40 em conjunto contêm 3 LEDs de 7 segmentos o bloco pode ser utilizado mais de uma vez.

Para a definição e a compilação do código HLD serve um editor especial (opção do menu *Tools / HDL Editor*). Ao entrar neste editor é necessário escolher uma determinada HDL. A seguir, pode-se escrever manualmente o código ou aproveitar as facilidades fornecidas pelo *HDL Design Wizard*. Este permite definir todas as portas de entrada e saída (fig. 4) e constroi automaticamente a estrutura básica do programa (fig. 5). Na fig. 5 a primeira parte do código inclui todas as bibliotecas standard necessárias, a segunda parte define a interface com o exterior, i.e. especifica as portas de entrada/saida do circuito, e a terceira descreve o seu comportamento ou a sua constituição.

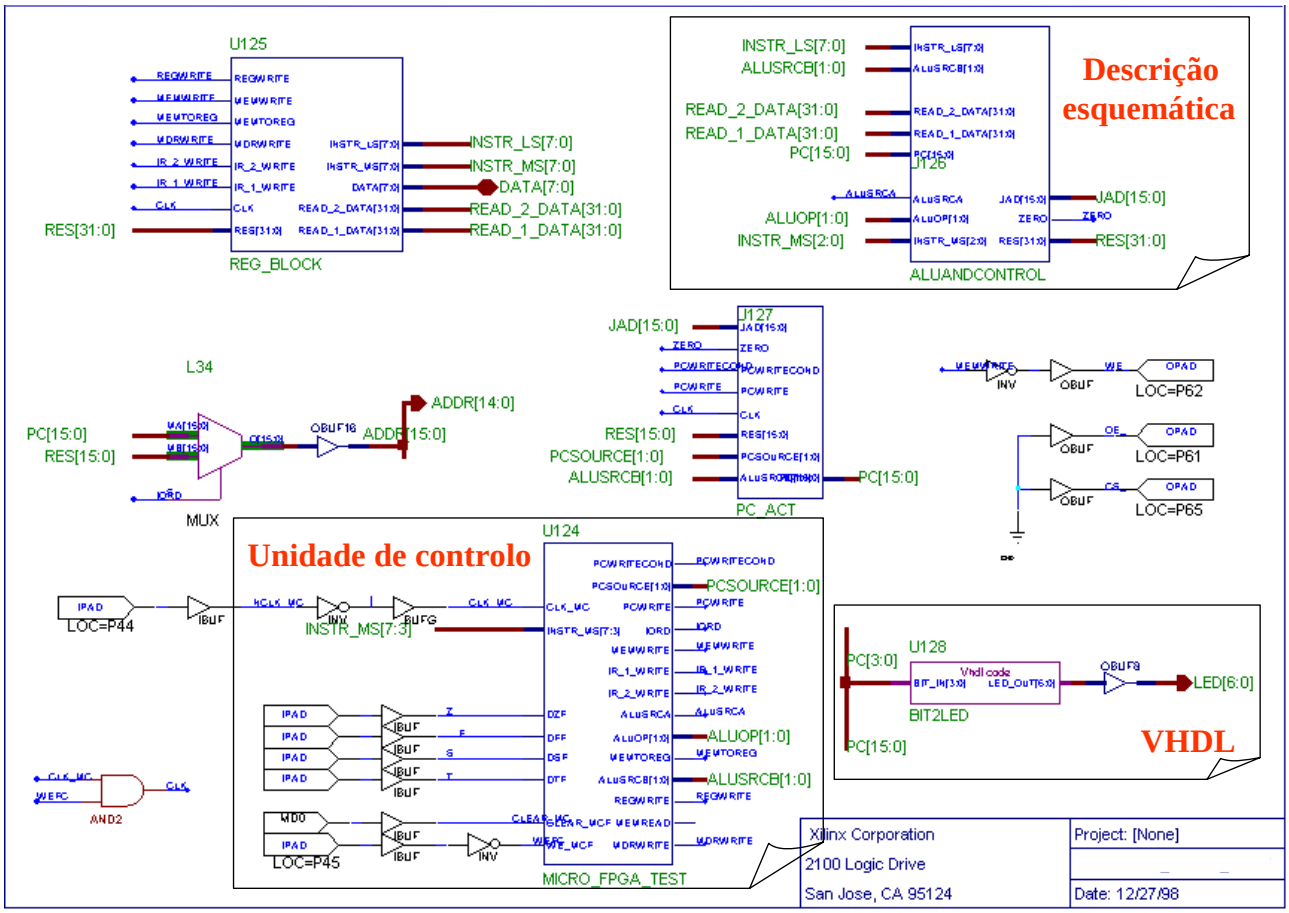

Fig. 3 – Nível superior do projecto

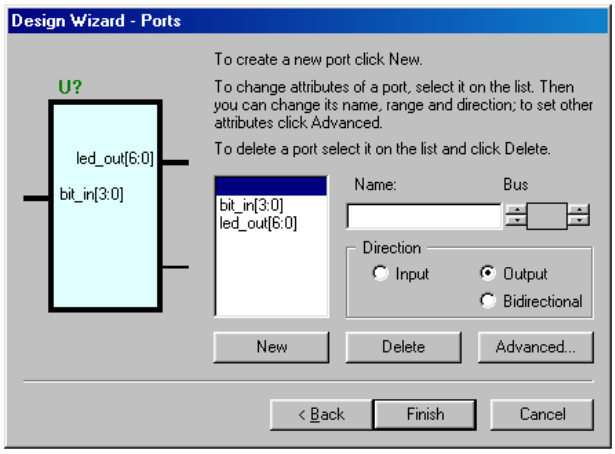

Fig. 4 - HDL Design Wizard

```
1  library IEEE;
   use IEEE.std_logic_1164.all;

2  entity BIT2LED is
   port ( bit_in: in STD_LOGIC_VECTOR (3 downto 0);
        led_out: out STD_LOGIC_VECTOR (6 downto 0) );
   end BIT2LED ;

3  architecture BIT2LED_arch of BIT2LED is
   begin
     -- <<enter your statements here>>
   end BIT2LED_arch;
```

Fig. 5 – Estrutura básica de um programa VHDL

Ao obter a estrutura básica do programa pode-se desenvolvê-la manualmente ou usar o *Language Assistant* (opção do menu *Tools / Language Assistant*) que contém as construções principais da HDL escolhida o que permite diminuir a probabilidade de erros sintácticos e aumentar a produtividade. P.ex., para o nosso caso podemos seleccionar o comando VHDL *select* (fig. 6).

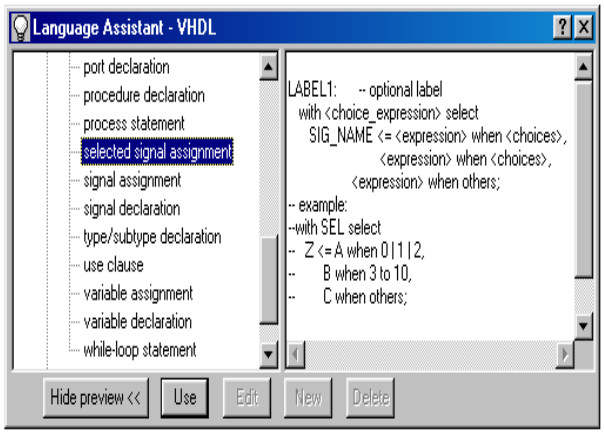

Fig. 6 – Uso do *Language Assistant*

A seguir, a cada valor do vector de entrada de 4 bits é necessário arranjar um vector correspondente de saída de 7 bits (fig. 7). Cada "1" no vector de saída significa o estado activo do segmento LED respectivo. Cada um dos segmentos LED é ligado a um dos pinos da FPGA. A correspondência entre os sinais do circuito e os pinos da FPGA pode ser definida quer no editor esquemático pelo atributo *LOC=PNºpino*, quer num ficheiro **.ucf* com o comando seguinte: *NET "Nome_do_sinal<número_do_ bit>" LOC = "Pnúmero_do_pino";*.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity BIT2LED is
    port (   bit_in: in STD_LOGIC_VECTOR (3 downto 0);
             led_out: out STD_LOGIC_VECTOR (6 downto 0) );
end BIT2LED;
architecture BIT2LED_arch of BIT2LED is
begin
   with bit_in SELect
      led_out<=
           "0010010" when "0001", --1
           "1011101" when "0010", --2
           "1011011" when "0011", --3
           "0111010" when "0100", --4
           "1101011" when "0101", --5
           "1101111" when "0110", --6
           "1010010" when "0111", --7
           "1111111" when "1000", --8
           "1111011" when "1001", --9
           "1110111" when "0000", --0
           "1111110" when "1010", --A
           "0101111" when "1011", --B
           "1100101" when "1100", --C
           "0011111" when "1101", --D
           "1101101" when "1110", --E
           "1101100" when "1111", --F
           "1000000" when others; --0
end BIT2LED arch;
```

Fig. 7 – Código final do descodificador

Ao preparar o código VHDL é necessário sintetizar o circuito correspondente (opção do menu *Synthesis / Synthesize*) e, como o nosso descodificador faz parte de um circuito maior, criar uma macro-definição (opção do menu *Project / Create Macro*). Caso o compilador não encontre erro nenhum gerar-se-á a *NetList* respectiva (fig. 2). A macro criada pode ser agora utilizada no editor esquemático e a *NetList* será aproveitada na fase da implementação do circuito.

A.2. Definição de circuitos usando o editor esquemático

Analizemos a construção da unidade aritmética e lógica (ALU) que faz parte do bloco *ALUANDCONTROL* (fig. 3), que contém também circuitos de controlo auxiliares. Neste bloco entram os operandos da ALU e os sinais de controlo que escolhem os operandos necessários e indicam o código da operação a executar. Na saída de *ALUANDCONTROL* aparece o resultado da operação. Como se implementou apenas um subconjunto das instruções MIPS, o número das operações executáveis pela ALU é limitado.

Para a construção da ALU pode-se aproveitar as facilidades fornecidas pelo *Symbol Wizard* (opção do menu *Hierarchy / New Symbol Wizard…*), que permite especificar todas as portas de entrada/saída da maneira descrita no A.1 (fig. 4) e visualizá-los no esquema. Pretende-se implementar as operações de adição, subtração, comparação, "E" lógico e "OU" lógico sobre

vectores de 32 bits. Comecemos pela simples operação de "E" lógico que pode ser realizada utilizando um dos 3 métodos. O primeiro consiste na descrição da operação pretendida numa HDL (ver ponto A.1). Na biblioteca standard XC4000 da Xilinx há elemento (*AND*) que executa "E" lógico sobre dois sinais de um bit, portanto o segundo método está na construção de um esquema com 32 desses elementos. O terceiro método é o mais simples. Consiste na definição de uma macro LogiBlox (opção do menu *Options / Logiblox*). LogiBlox é uma ferramenta de software fornecida juntamente com o Xilinx Foundation Series 1.5 Software e que, possuindo uma interface amigável com o utilizador, permite construir blocos funcionais comuns (descodificadores, contadores, somadores, portas lógicas, etc.) de uma dimensão arbitrária. Por exemplo, para o nosso caso é necessário definir o bloco pretendido de maneira apresentada na fig. 8.

A seguir o LogiBlox cria a *NetList* correspondente e a macro-definição respectiva que poderá ser utilizada no esquema. De maneira análoga especificam-se as operações de adição, subtração e "OU" lógico. A comparação pode ser composta pela subtração e comparação sucessiva do resultado com zero. Tendo todos ob blocos necessários convém interligá-los entre si e com as portas de entrada/saída de modo apresentado na fig. 9.

O bloco construído pode agora ser inserido em qualquer lugar nos níveis superiores da hierarquia esquemática ligando às suas portas de entrada/saída os sinais respectivos. É possível modificar a disposição inicial das portas e a representação do bloco com a ajuda do *Symbol Editor* (opção do menu *Options / Symbol Editor*).

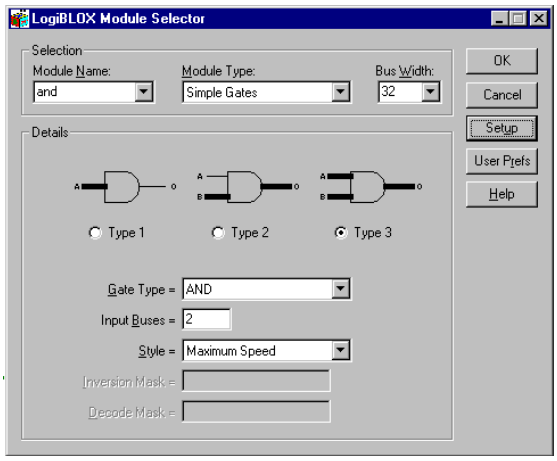

Fig. 8- Especificação de um elemento LogiBlox

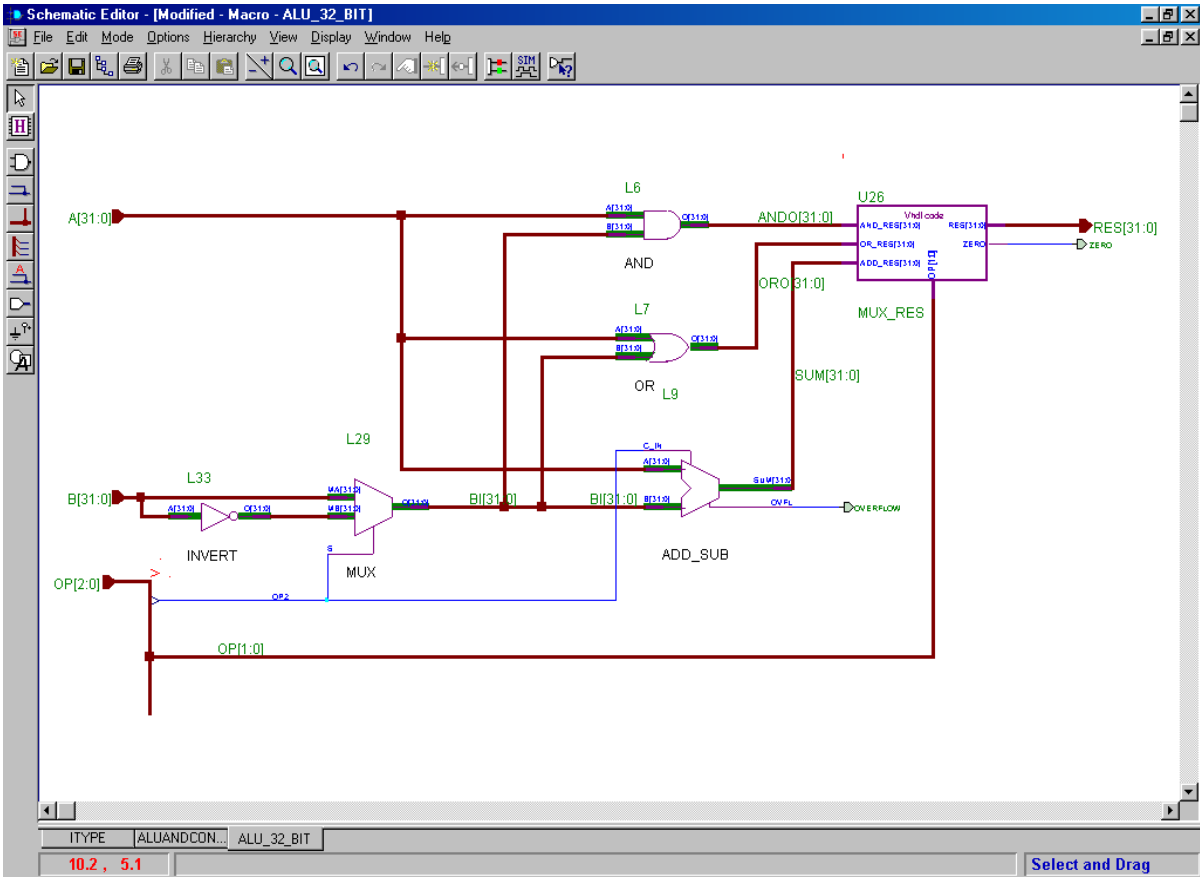

Fig. 9 – Alu de 32 bits

IV. PROJECTO DA UNIDADE DE CONTROLO

Foram considerados 2 métodos diferentes de especificação da unidade de controlo. O primeiro consiste na utilização do editor de estados, o segundo - na construção do circuito com o editor de esquemáticos.

O editor de estados permite descrever as máquinas de Moore e Mealy de uma forma gráfica [6]. Analizemos a implementação de uma das instruções MIPS, por exemplo da instrução de adição. A estrutura básica de uma máquina de estados finita está apresentada na fig. 10. No nosso caso a entrada são os 5 bits do código da operação e na saída aparece a sequência de sinais de controlo necessários para a execução desta operação no *datapath*.

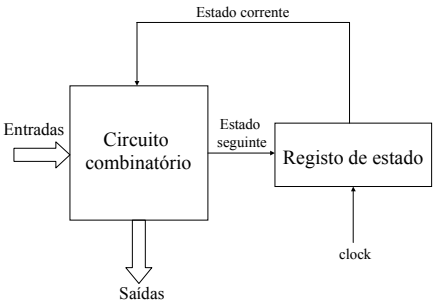

Fig. 10 - Estrutura básica de uma máquina de estados finita

Para a especificação das portas de entrada/saída pode-se usar novamente o *HDL Design Wizard* (fig. 4). O *Wizard* permite definir o número pretendido das máquinas de estado finitas (FSM) e escolher a HDL em que estas serão descritas. No nosso caso precisamos de uma FSM. Se for necessário pode-se definir várias FSMs, inclusivamente FSMs que interagem entre si e funcionam em paralelo.

Utilizou-se a implementação comum da instrução de adição composta pelas seguintes 4 fases: extracção da instrução da memória, a sua descodificação, a execução da operação de adição e a escrita do resultado para um dos registos. No MIPS16 as instruções ocupam 16 bits. No entanto a linha de dados da RAM externa que foi utilizada para o código do programa tem 8 bits. Portanto a fase da extracção da instrução da memória ocupa 2 ciclos de relógio. Consequentemente a máquina de estados finita precisa de ter 5 estados que podem ser definidos com a ajuda da opção do menu *FSM / State* (fig. 11). A seguir deve-se especificar todas as transições entre os estados (opção do menu *FSM / Transition*) e as condições sob as quais as várias transições têm lugar (opção do menu *FSM / Condition*). A última fase da descrição da FSM é a especificação das acções que devem ser efectuadas em cada estado, i.e. a definição de todos os sinais de controlo que devem ser activados (opção do menu *FSM / Action / State*) (fig. 11). A seguir convém gerar o código HDL correspondente (opção do menu *Synthesis / HDL Code Generation*) e sintetizar o esquema respectivo (opção do menu *Synthesis / Synthesize*). Agora a macro-definição obtida pode ser utilizada no editor esquemático.

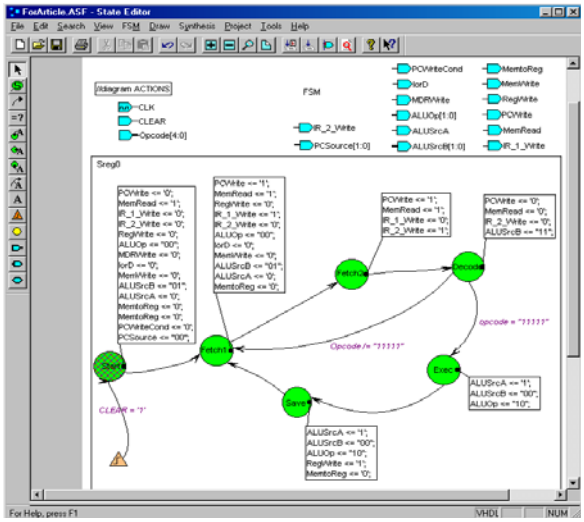

Fig. 11 - Especificação de uma FSM

Consideremos um método diferente que embora ocupe mais recursos da FPGA é muito mais eficiente porque permite programar a unidade de controlo do exterior, i.e. modificar as instruções existentes e definir novas instruções, sem necessidade de alteração do circuito e, consequentemente, sem a repetição da compilação do projecto e do carregamento do ficheiro da configuração para a FPGA.

Esta unidade de controlo tem a estrutura apresentada na fig. 12. Para a RAM carrega-se o microprograma necessário através da porta paralela do computador. O bloco *Cálculo do endereço seguinte da RAM* determina o endereço de leitura com base no código da operação e na microinstrução corrente.

A RAM pretendida pode ser implementada em forma de um conjunto de registos definidos como macros LogiBlox. No entanto este método, embora simples, não é eficiente porque a RAM é sintetizada recorrendo aos D-flipflops (cada CLB tem 2 D-flipflops [5]). Por exemplo, para implementar uma RAM16x16 são necessários 128 CLBs (16*16 bits / 2 D-flipflops em cada CLB), i.e. mais de 1/3 dos recursos da FPGA XC4010XL.

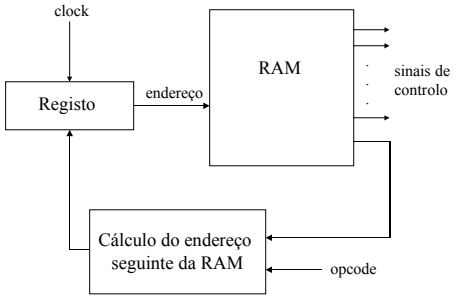

Fig. 12 - Estrutura da unidade de controlo

Existe um método diferente que consiste na construção do esquema pretendido a partir de elementos da biblioteca XC4000 da Xilinx (p. ex., o elemento RAM16x4). Neste caso a RAM é implementada utilizando as LUTs. Cada CLB tem 2 LUTs de 4 entradas que podem ser utilizadas para estes fins, i.e. cada CLB é capaz de guardar 32 bits [5]. Sendo assim para a construção da RAM16x16 serão necessários apenas 8 CLBs (16*16 bits / 32 bits em cada CLB).

V. SISTEMA DE DESENVOLVIMENTO PARA CIRCUITOS IMPLEMENTADOS EM FPGA

Para a interacção com o processador implementado em FPGA foram desenvolvidas em Visual C++ 6.0 várias ferramentas. As facilidades principais fornecidas pelas ferramentas são representadas na fig. 13.

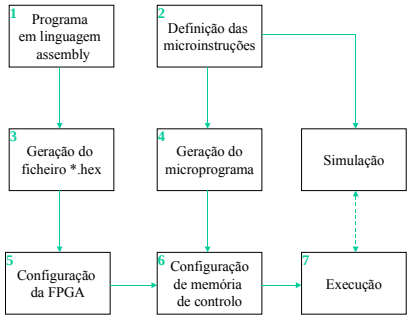

Fig. 13 - Possibilidades das ferramentas de software desenvolvidas

Analizemos cada uma delas em pormenor. Em primeiro lugar, existe um editor de texto (n. 1 na fig. 13) que permite ao utilizador escrever programas em linguagem assembly MIPS. O assembler desenvolvido traduz o programa criado e gera o código máquina correspondente (n. 3 na fig. 13).

Desenvolveram-se também ferramentas que permitem a extensão do reportorio de instruções implementado, quando a unidade de controlo é microprogramada. Para especificar uma instrução nova é preciso indicar o seu nome (que deve ser único), o código (que deve ser único também), o formato e desenvolver o microprograma correspondente (n. 2 na fig. 13 e fig. 14). O software une automaticamente as microinstruções recém-definidas com o microprograma existente (n. 4 na fig. 13). É de notar que as possibilidades de extensão do conjunto das instruções são limitadas, primeiro, pelo tamanho da RAM de controlo (especificado no Xilinx Foundation Series 1.5 Software) e, segundo, pela quantidade de condições lógicas descritas no esquema. De uma maneira semelhante pode-se modificar as instruções existentes (fig. 14).

Para a análise do funcionamento do processador foi desenvolvido um simulador especial (fig. 15) que permite analisar as fases principais da execução de uma instrução ao nível do conteúdo dos blocos básicos estruturais, dos sinais de controlo e dos sinais gerados no *datapath*. O simulador tem 4 áreas: a área de controlo, o *datapath*, a memória e a interface. A área de controlo que fica na parte esquerda da fig. 15, gera todos os sinais de controlo necessários, actualiza o *program counter*, e apresenta também informações úteis para o utilizador, por exemplo, visualiza a instrução que está a ser executada em assembly e em formato binário, mostra o número de ciclos de relógio passados desde o início da execução da instrução, etc.

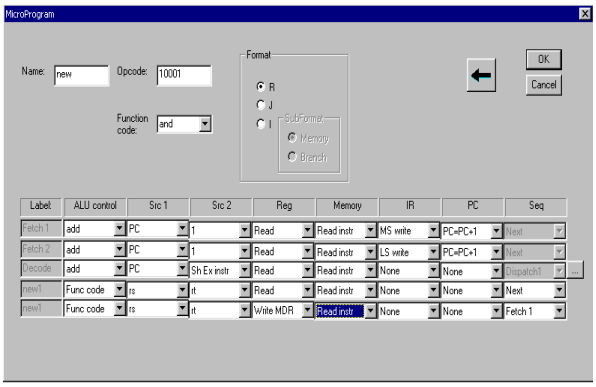

Fig. 14 - Definição / modificação das instruções

O bloco *datapath* (fig. 15) visualiza o conteúdo dos registos de uso geral, a ALU e os blocos funcionais auxiliares necessários. O bloco *memory* apresenta a memória que foi utilizada para o código e os dados, e o bloco *interface* efectua a interacção com a memória.

Todos os sinais e registos que mudaram de valor no ciclo corrente, destacam-se com uma cor mais brilhante. O simulador possui 3 modos de funcionamento, permitindo executar de uma só vez todo o programa em linguagem assembly introduzido pelo utilizador e mostrando os estados e valores finais de todos os blocos e sinais, ou executando o programa instrução a instrução, ou microinstrução a microinstrução (ciclo a ciclo). Nos dois últimos casos o simulador visualiza os valores de todos os sinais e os estados dos blocos estruturais em cada passo.

Sendo assim, se o utilizador cometeu um erro na definição de uma instrução (fig. 14), ao analisar a sua execução passo a passo no simulador, ele poderá detectá-la facilmente e corrigi-la. Crêmos que estas facilidades tornem a ferramenta num instrumento de grande utilidade no ensino de microprogramação.

Depois de verificar o programa com o auxílio do simulador, o utilizador pode executá-lo na FPGA. As ferramentas desenvolvidas carregam para a FPGA o *bitstream* gerado pela Xilinx Foundation Series 1.5. Software e inicializam a 32Kx8 SRAM montada na placa XS40 com a ajuda de um ficheiro do formato *.hex que contém os dados e o código do programa (n. 5 na fig. 13). A seguir através da porta paralela do computador configura-se a unidade de controlo de maneira a esta executar o microprograma especificado pelo utilizador (n. 6 na fig. 13). Isso faz-se escrevendo cada microinstrução na respectiva célula da RAM de controlo e programando o bloco que calcula os endereços da RAM (fig. 12).

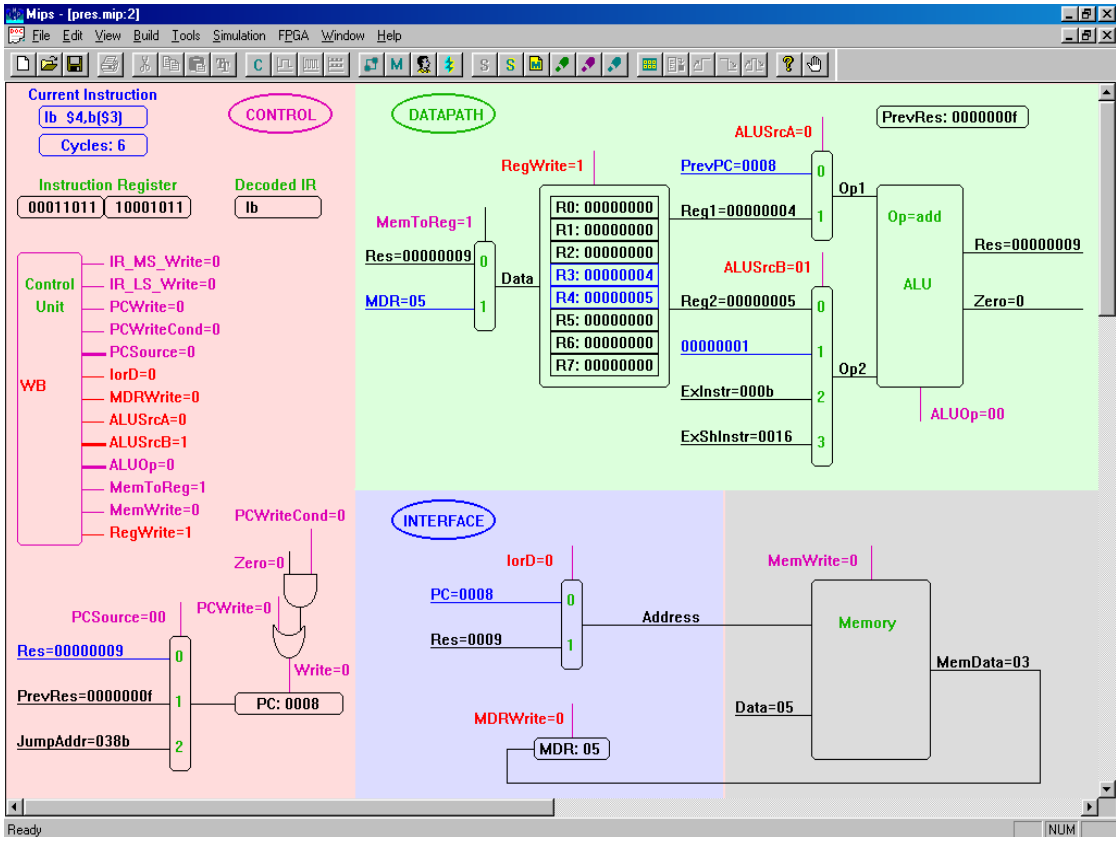

Fig. 15 – Simulador

Agora pode-se executar o programa enviando através da porta paralela do computador os sinais de sincronização correspondentes (n. 7 na fig. 13). Na ausência do simulador seria preciso verificar o circuito implementado na FPGA com o simulador da Xilinx Foundation Series 1.5 Software, o que é praticamente impossível devido à necessidade de inicializar manualmente muitos blocos estruturais.

VI. ESTATÍSTICAS DA IMPLEMENTAÇÃO

O processador implementado é capaz de executar apenas um subconjunto das instruções MIPS que inclui as principais instruções aritméticas e lógicas, os comandos básicos para a interacção com a memória e as instruções de salto condicional e incondicional. O processador é composto pelos seguintes blocos: 8 registos de 32 bits (*REGBLOCK*), uma ALU de 32 bits (*ALU*), a unidade de controlo (*CONTROL*) e bloco que actualiza o *program counter* (*PC_ACT*). Vários blocos utilizam os CLBs de modos diferentes. Nuns (*ALU*, *PC_ACT*) os CLBs são programados de maneira a implementarem funções lógicas, noutros (*REGBLOCK*, *CONTROL*) os CLBs são configurados como RAMs rápidas. Todos os blocos foram construídos recorrendo principalmente a elementos da biblioteca XC4000 da Xilinx, embora alguns destes fossem descritos em VHDL. Na tabela 1 são representados os recursos da FPGA utilizados pelos diferentes blocos .

	REG_ BLOCK	*ALU*	*CONT- ROL*	*PC_ ACT*	*Total*
CLBs	58 (14%)	94 (23%)	51 (12%)	17 (4%)	207 (51 %)
Flipflops	24	0	20	32	75
4 i/p LUTs	99	164	54	34	288
3 i/p LUTs	12	37	19	8	68
Número equivalenet e de portas lógicas	4450	1943	2115	1442	8824

Tabela 1 - Os recursos da FPGA XC4010XL utilizados pelo projecto

Todo o projecto ocupou aproximadamente 1/2 dos recursos da FPGA XC4010XL, no entanto alguns dos blocos ainda podem ser optimizados. É de notar que a soma dos recursos utilizados por cada bloco separadamente excede os recursos necessários para todo o projecto porque muitos dos CLBs executam funções comuns.

Na tabela 2 para os fins de comparação é apresentado um sumário de recursos ocupados pelos blocos *REG_BLOCK* e *CONTROL* implementados de um modo alternativo. O bloco *REG_BLOCK** foi construído com base nos registos gerados pela ferramenta LogiBlox, e o bloco *CONTROL** foi descrito no editor de estados de maneira explicada no ponto IV (fig. 11).

	REG_ BLOCK	***REG_ BLOCK****	*CONT-ROL*	***CONT-ROL****
CLBs	58 (14%)	**157 (39%)**	51 (12%)	**48 (12%)**
Flipflops	24	**280**	20	**12**
4 i/p LUTs	99	**59**	54	**82**
3 i/p LUTs	12	**108**	19	**11**
Número equivalente de portas lógicas	4450	**3570**	2115	**613**

Tabela 2 - Os recursos da FPGA XC4010XL utilizados pelo blocos *REG_BLOCK** e *CONTROL**

VII. CONCLUSÕES

É conhecido que a complexidade e as possibilidades funcionais das FPGAs contemporâneas estão a crescer rapidamente e que hoje em dia as FPGAs podem ser utilizadas não só para a implementação de circuitos lógicos irregulares simples ("*glue logic*"), mas também na construção de sistemas computacionais e, em particular, de sistemas *embedded* bastante complexos. O artigo apresenta uma experiência da realização de um processador com a arquitectura MIPS16 de 32 bits em FPGA XC4010XL. Para estudar este processador e trabalhar com ele foram desenvolvidas ferramentas de software especiais. Estas possuem facilidades tais como a modificação do conjunto de instruções do processador, a análise da execução de vários comandos, observação de todos os passos intermédios necessários para a execução das instruções pelo processador, etc. Tais ferramentas, do nosso ponto de vista, são úteis tanto para os fins de investigação quanto para o processo educativo. Para a implementação física do processador foram utilizados os meios comerciais de software (Xilinx Foundation Series 1.5 Software) e de hardware (a FPGA XC4010XL, as placas XS40 e XStend). A descrição inicial dos componentes do processador foi efectuada ao nível de esquemáticos e com a ajuda de um subconjunto da linguagem VHDL (designado VHDL-sintetizável).

REFERÊNCIAS

[1] Kevin D.Kissel, "MIPS16: High-density MIPS for the Embedded Market".

[2] "TinyRISC™ Microprocessor with 16/32 - Bit Code Compression", http://www.lsilogic.com/products/unit5_5th.html.

[3] Patterson D.A., Hennessy J.L., "Computer Organization and Design. The Hardware/Software Interface", Morgan Kaufmann Publishers, Inc., 1998.

[4] Skliarova I., Ferrari A.B., "Implementação e Simulação do Processador MIPS com a ALU Reconfigurável Dinamicamente", Electrónica e Telecomunicações, vol. 2, Nº 4, pp. 497-504, 1999.

[5] 1999 Xilinx Data Book, 1999.

[6] Dave Van den Bout, "The practical Xilinx Designer Lab Book", Prentice Hall, Inc., 1998.

[7] http://www.xess.com/FPGA.

[8] Peter J.Ashenden, "The Designer's Guide to VHDL", Morgan Kaufmann Publishers, Inc., 1996.