Ferramentas para desenvolvimento de sistemas digitais reconfiguráveis [*]

Valery Sklyarov, Iouliia Skliarova

Resumo – **Este artigo descreve os métodos e as ferramentas recentes que são usados no processo educativo no Departamento de Electrónica e Telecomunicações da Universidade de Aveiro no âmbito das disciplinas relacionadas com o desenvolvimento de sistemas reconfiguráveis. São abordados os três tópicos seguintes: 1) as FPGAs das famílias Spartan-II/Spartan-IIE da Xilinx cujas vantagens principais são o baixo custo, a arquitectura bem definida de componentes primários, etc.; 2) as placas baseadas em FPGAs, nomeadamente a placa RC100 da Celoxica, a TE-XC2Se da Trenz Electronic e a XSA100 da XESS; 3) a ferramenta Xilinx ISE 5 que é um sistema de desenvolvimento assistido por computador. O artigo descreve alguns exemplos úteis que demonstram as capacidades de FPGAs e da Xilinx ISE, tais como o uso de linguagens de descrição de hardware (VHDL, em particular) e de bibliotecas de sistema ou criadas pelo utilizador, a implementação de componentes esquemáticos, a síntese de máquinas de estados finitos, a modelação, a interacção com os dispositivos periféricos (tais como o LCD – *liquid crystal display*, a RAM estática, o rato, etc.), a utilização de memória dedicada (*Block RAM*) e distribuída, a sincronização de circuitos com a ajuda de DLLs – *delay-locked loops*, os geradores de componentes parametrizáveis, o fluxo de projecto baseado em EDIF (*Electronic Design Interchange Format*) incluíndo a implementação de circuitos desenvolvidos com a ajuda de linguagens de especificação a nível de sistema (Handel-C, em particular), etc.**

Abstract – **The paper presents a tutorial, which covers new methods and tools that are used in educational process of Electronics and Telecommunications Department of Aveiro University for the disciplines devoted to the design of reconfigurable systems. Three following basic topics have been considered: 1) FPGAs of state-of-the-art Spartan-II/Spartan-IIE families that possess a number of advantages, such as low cost, well-defined architecture of primary components, etc.; 2) FPGA-based up to date prototyping boards, such as RC100 of Celoxica, TE-XC2Se of Trenz Electronic and XSA100 of XESS; 3) PC version of Xilinx ISE 5, which is modern CAD software integrating traditional tools for the design of FPGA-based digital systems. The paper describes a number of helpful examples that demonstrate capabilities of FPGAs and Xilinx ISE, such as the use of hardware description languages (VHDL in particular), system and user-defined libraries, implementation of schematic components, synthesis of finite state machines, modelling, interacting with external hardware (such as liquid crystal display - LCD, static RAM, mouse, etc), dedicated (Block RAM) and distributed memory, sinchronization circuits based on delay-looked loops, core generators, EDIF-based design flow including implementation of circuits designed from system-level specification languages (Handel-C in particular), etc.**

I. INTRODUCÃO

O progresso constante que se verifica hoje em dia na área de desenvolvimento de sistemas reconfiguráveis requer uma actualização periódica dos conteúdos das disciplinas relevantes que fazem parte da formação inicial e pós-graduada. O Departamento de Electrónica e Telecomunicações da Universidade de Aveiro lecciona três disciplinas dedicadas a esta área, nomeadamente "Computação reconfigurável", "Sistemas Digitais Avançados" e "Sistemas Reconfiguráveis Avançados". Durante alguns anos as aulas têm sido baseadas nas FPGAs da família XC4000 da Xilinx e nas ferramentas de desenvolvimento respectivas (*Xilinx Foundation Software*, versões 3.x e 4.x). A experiência ganha nesta área junto com a lista completa de referências às publicações relevantes preparadas pelos membros do Departamento estão apresentadas em [1]. A partir do final do ano 2002 foi efectuada uma mudança para o ambiente de software ISE 5 e as FPGAs das famílias Spartan-II/Spartan-IIE que possuem uma arquitectura semelhante à da família Virtex e são recomendadas para o desenvolvimento de sistemas digitais futuros. Estas ferramentas de hardware/software incluem muitos aperfeiçoamentos e consequentemente requerem uma revisão dos programas das disciplinas mencionadas acima. As secções seguintes descrevem as características e as capacidades mais importantes de FPGAs, placas de protótipo e sistemas de desenvolvimento assistido por computador, utilizados no processo educativo a partir do final do ano 2002. Estas demonstram como o software e hardware modernos podem ser utilizados no âmbito de fluxo de projecto tradicional para os sistemas reconfiguráveis. O objectivo principal que se pretende atingir com este artigo é auxiliar os estudantes a resolverem problemas prácticos com a ajuda das ferramentas contemporâneas.

[*] Trabalho financiado parcialmente com a bolsa da FCT-PRAXIS XXI/BD/21353/99

Uma área diferente que é extremamente importante hoje em dia, é o projecto a partir das linguagens de especificação a nível de sistemas, tais como SystemC [2], Handel-C [3] e muitas outras. Para a leccionação foi escolhida a linguagem Handel-C [4] e o ambiente DK1 desenvolvido por Celoxica. Esta escolha é fundamentada pelas razões seguintes:

- todas as ferramentas necessárias para o desenvolvimento imediato de sistemas digitais baseados em FPGA (tais como o ambiente DK1 [5], a placa RC100 [6], etc) podem ser facil e rapidamente recebidos da Celoxica juntamente com manuais e exemplos compreensivos;
- as ferramentas são directamente integráveis com ISE 5 da Xilinx; por exemplo, é possível construir no ambiente DK1 os módulos VHDL a partir da sua especificação em Handel-C e utilizá-los posteriormente como componentes de biblioteca em ISE 5;
- a Celoxica fornece alguns componentes complementares que são úteis para o desenvolvimento de sistemas digitais, tais como os controladores para dispositivos periféricos (monitor VGA, teclado, rato, memória, etc.), a biblioteca de macros Handel-C, as ferramentas de depuração e simulação, etc.;
- finalmente a nossa própria experiência indica que as ferramentas da Celoxica permitem resolver problemas de projecto bastante complicados.

SystemC é uma biblioteca de classes muito conhecida que possibilita a modelação de sistemas digitais recorrendo à tecnologia orientada por objectos. Para a parte lectiva foi preparada uma introdução aos tópicos relevantes baseada em [7].

O resto deste artigo está organizado da maneira seguinte. A secção II inclui uma revisão dos recursos modernos de hardware reconfigurável de que o Departamento dispõe actualmente. Na secção III apresenta-se a ferramenta ISE 5 junto com alguns exemplos de desenvolvimento de circuitos para as FPGAs da família Spartan-IIE e a placa TE-XC2Se. A seguir, na secção IV, descrevem-se exemplos de circuitos para as FPGAs da família Spartan-II e a placa RC100. Ambas as secções demonstram como se pode interagir com os dispositivos periféricos. Finalmente, as conclusões estão na secção V.

II. HARDWARE RECONFIGURÁVEL E PLACAS DE DESENVOLVIMENTO

Nesta secção são descritas as FPGAs e as placas de desenvolvimento utilizadas no processo educativo a partir do final do ano 2002.

A. FPGAs de famílias Spartan-II/Spartan-IIE

As FPGAs das famílias Spartan-II/Spartan-IIE da Xilinx são dispositivos reconfiguráveis baratos que possuem uma arquitectura semelhante à das FPGAs da família Virtex. A tabela 1 sumaria as características principais destas FPGAs. As 6 primeiras linhas correspondem à família Spartan-II. Os nomes dos dispositivos da família Spartan-IIE possuem o sufixo "E". As linhas marcadas com asterisco na tabela 1 contêm os tipos de FPGAs existentes em ambas as famílias.

FPGA	Array de CLBs	Número máximo de pinos de I/O	RAM distribuída (bits)	Blocos de RAM (Kbits)
XC2S15	8x12	86	6 144	16
XC2S30	12x18	132	13 824	24
XC2S50E*	16x24	182	24 576	32
XC2S100E*	20x30	202	38 400	40
XC2S150E*	24x36	265	55 296	48
XC2S200E*	28x42	289	75 264	56
XC2S300E	32x48	329	98 304	64
XC2S400E	40x60	410	153 600	160
XC2S600E	48x72	514	221 184	288

Tabela 1 – FPGAs das famílias Spartan-II/Spartan-IIE da Xilinx

A fig. 1 representa a arquitectura básica de FPGAs da família Spartan-IIE [8]. Existem 4 componentes primários na fig. 1: um array de blocos lógicos configuráveis (*Configurable Logic Blocks* - CLBs), blocos de RAM, DLLs (*Delay-Locked Loops*) e os blocos de entrada/saída (*Input/Output* - I/O).

O componente principal dum CLB é uma célula lógica (*LC – logic cell*) baseada em LUT (ver o canto inferior direito da fig. 1) que permite implementar qualquer função Booleana de 4 variáveis. A realização duma função particular é atingida ao carregar o *bitstream* na LUT relevante. A LC contém também um *flip-flop* e a lógica de *carry*. Cada CLB é composto por 4 LCs organizadas em duas *slices* semelhantes. Para além disso, um CLB inclui componentes lógicos adicionais que permitem ligar LCs de maneira a poderem realizar todas as funções Booleanas de 6 varáveis e algumas funções Booleanas que contêm no máximo 19 variáveis. As duas LUTs, que fazem parte dum *slice*, podem ser combinadas a fim de construir RAM/ROM com as seguintes configurações possíveis: 16x2 ou 32x1 *single-port*, ou 16x1 *dual-port*. Cada LUT permite também implementar um registo de deslocação (até 16 bits). Os *flip-flops* incluídos nas LCs podem ser configurados para funcionarem quer como *flip-flops* do tipo *D* activos na transição ascendente (*edge-triggered*) quer como *latches* activas ao nível (*level-sensitive*). Os componentes específicos (*carry logic*) existentes em LCs possibilitam a execução rápida de operações aritméticas. Cada LC permite a implementação de um somador completo de 1 bit. Os CLBs incluem também dois *drivers* do tipo *3-state* para ligações com os barramentos do *chip*.

Os blocos de I/O reprogramáveis (ver o canto inferior esquerdo da fig. 1) satisfazem uma ampla gama de normas de I/O existentes [8].

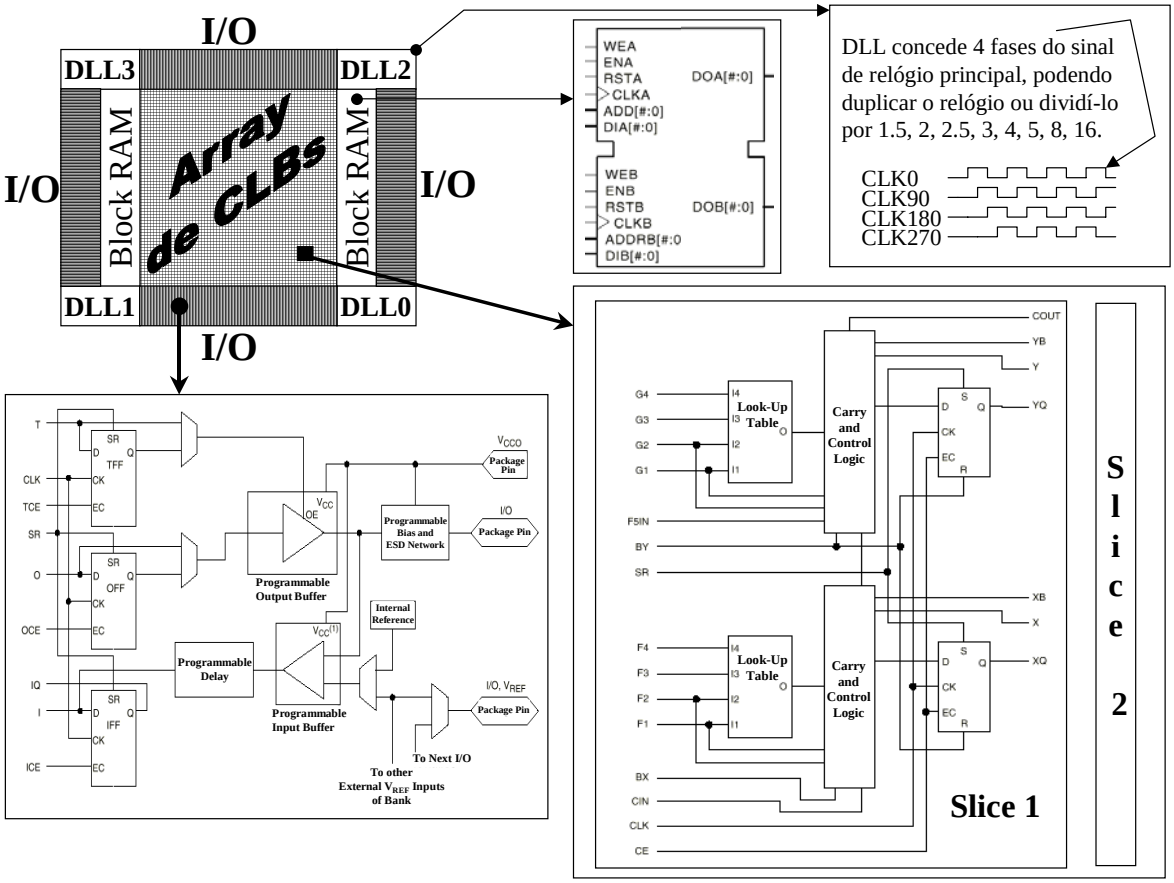

Fig. 1 – Arquitectura das FPGAs da família Spartan-IIE

Há três *flip-flops* de I/O que podem ser programados para funcionarem quer como *flip-flops* do tipo *D* activos na transição ascendente quer como *latches* activas ao nível. O *buffer* encaminha os sinais de entrada externos ou directamente para a lógica interna ou para um *flip-flop*. O caminho de saída contém um *buffer* do tipo *3-state* de saída. Os sinais de saída podem ser encaminhados quer directamente para este *buffer* quer passando primeiro através de um *flip-flop*.

Os DLLs (ver o canto superior direito da fig. 1) observam os sinais de relógio (os de entrada e os distribuídos) e ajustam automaticamente o elemento de atraso do relógio. Um atraso adicional é efectuado de tal maneira que as transições do relógio atingem os *flip-flops* internos passando exactamento um período de relógio depois destas aparecerem nas entradas. Isto permite eliminar o atraso de distribuição do relógio ao assegurar que as transições do relógio atingem os *flip-flops* internos de uma maneira síncrona com as transições do relógio nas entradas externas. Cada DLL implementa um número de funções úteis tais como a deslocação das fases do sinal de relógio de entrada, a duplicação ou a divisão do sinal de relógio, etc.

Cada bloco de memória (ver o topo da fig. 1) representa uma RAM *dual-port* de 4096 bits com os sinais de controlo independentes para ambas as portas. Cada bloco pode implementar uma RAM com as seguintes configurações: 4096x1, 2048x2, 1024x4, 512x8, 256x16, sendo a largura do barramento de dados seleccionada independentemente para cada porta.

Os recursos de encaminhamento da FPGA permitem efectuar as ligações arbitrárias entre as entradas/saídas internas dos blocos descritos acima através do carregamento do *bitstream* que especifica as funções dos diferentes blocos e todas as ligações necessárias. Os detalhes podem ser encontrados em [8].

B. A placa RC100 da Celoxica

Na fig. 2. são representados os componentes básicos da placa RC100 junto com os dispositivos externos que podem ser ligados a esta. A FPGA XC2S200-5-FG456 (ver tabela 1) da família Spartan-II é um componente reconfigurável (o 5 no nome da FPGA indica o seu desempenho, e FG456 especifica o tipo do pacote e o número de pinos). As fig. 3 e 4 mostram todas as ligações entre a FPGA e outros componentes da placa. O oscilador da placa gera sinais de relógio a 80 MHz (ver fig. 3). Os dados do utilizador podem ser visualizados em dois LEDs e dois *displays* de 8 segmentos. Os *displays* partilham os pinos da FPGA com o conector de expansão de 50 pinos. Isso significa que os *displays* e o conector de expansão são de utilização mutuamente exclusiva. Os *displays* são activados ao atribuir o valor ‘1’ aos pinos “*Enable*”.

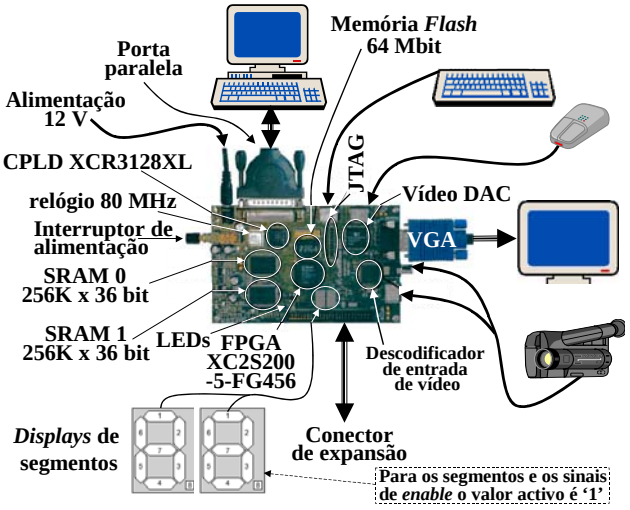

Fig. 2 – Placa RC100 da Celoxica

A placa pode receber os dados do rato [9-12] e do teclado [9,13,14], e por sua vez visualizar os dados num monitor VGA [15] (ver fig. 3). A informação adicional sobre o funcionamento de vários dispositivos periféricos está disponível em [16-18].

A programação da FPGA faz-se através da porta paralela do computador (ver fig. 4). Para além disso a porta paralela é controlável com a ajuda do CPLD XCR3128XL [19]. Portanto a placa pode enviar dados arbitrários para o computador e receber os dados arbitrários do computador através da mesma porta paralela (neste caso o CPLD controla directamente a porta). Os interruptores permitem activar um dos 64 segmentos da memória *Flash* de 128KByte cada um como segmento de arranque [20].

As linhas FP_RW, FP_COM[2:0] e FP_PARPORT_MASTER (ver fig. 4) controlam o CPLD e a porta paralela. Para assumir o controlo sobre a porta paralela, o sinal FP_PARPORT_MASTER deve receber o valor '0'. Isto permite à FPGA aceder à porta paralela escrevendo e lendo o barramento *Data(7:0)* na fig. 4, e controlar a porta directamente com a ajuda dos sinais FP_COM[2:0] mapeados nas linhas *Select* (AA12), *Busy* (Y12) e *Ack* (V12) da porta paralela (ver [18] onde o controlo da porta paralela se considera em detalhe). O sinal FP_WR especifica a direcção de fluxo dos dados ('1' - da FPGA para o computador, '0' – do computador para a FPGA). Quando o sinal FP_PARPORT_MASTER recebe o valor '1', isto permite à FPGA escrever os comandos de controlo de 3 bits para o CPLD a fim de: pôr o CPLD no modo *idle* (111), reconfigurar a FPGA a partir da memória *Flash* (101), ou reconfigurar o sector de arranque da memória *Flash* (011). Todos os detalhes são considerados nas páginas 8-9 de [6 – *hardware manual*].

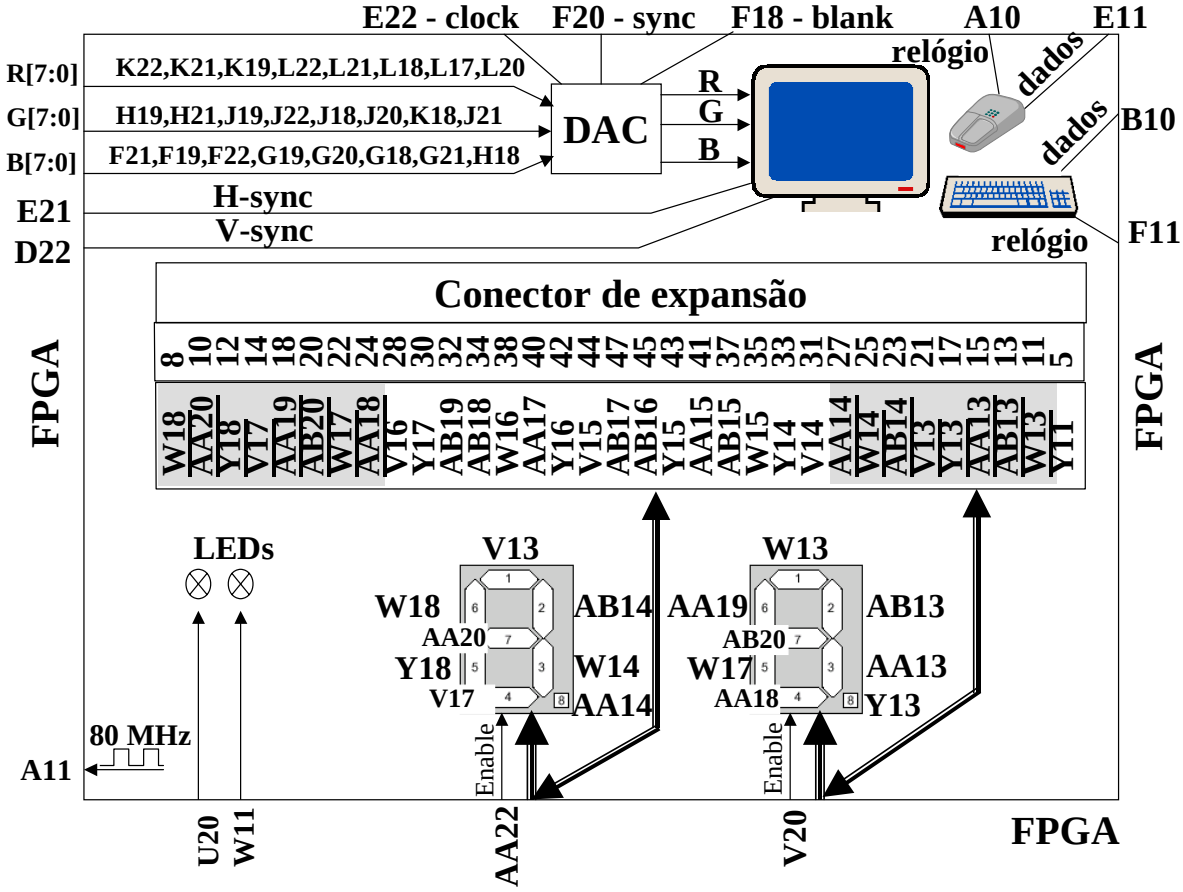

Fig. 3 – Ligações entre a FPGA e alguns componentes da placa RC100 e os dispositivos externos que podem ser ligados

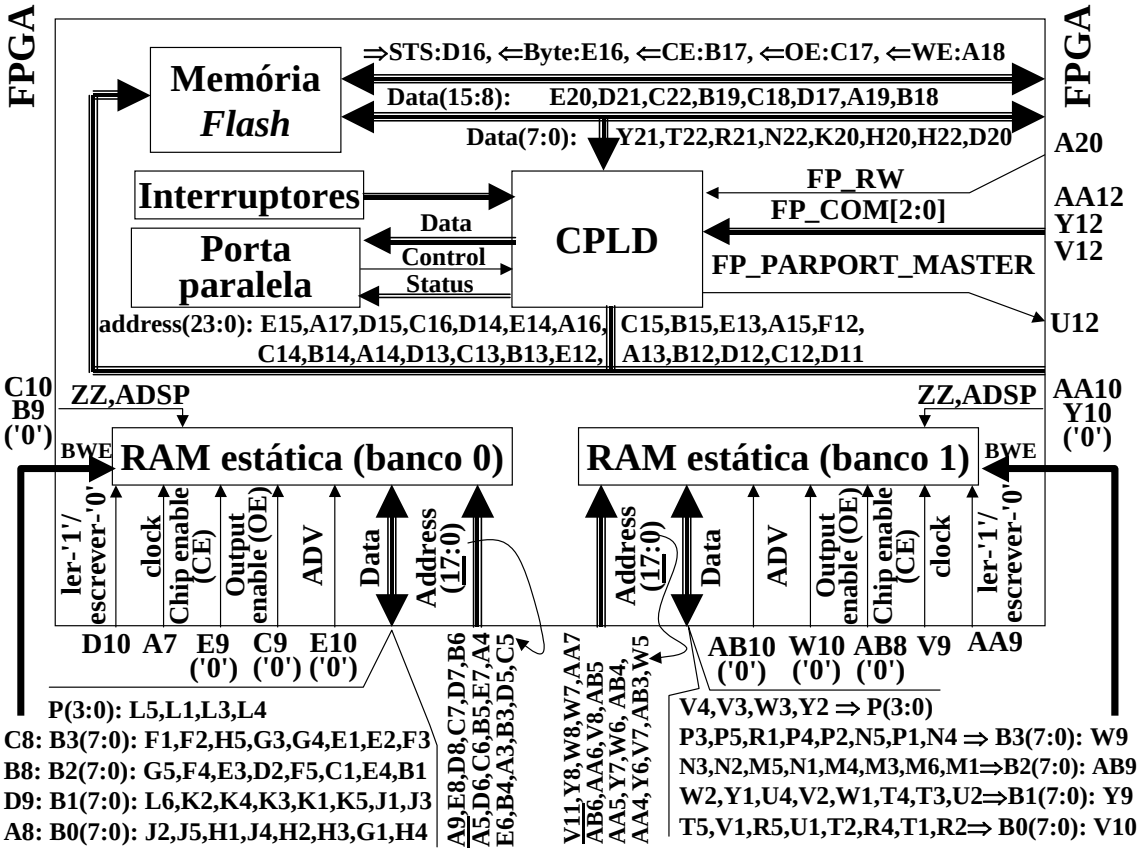

Fig. 4 – Ligações entre a FPGA e os componentes seguintes da placa RC100: CPLD, RAM estática, memória *Flash*, interruptores e a porta paralela.

A placa RC100 inclui também dois blocos de memória estática de 256Kx36-bit cada (ver os bancos 0 e 1 na fig. 4) [21]. Na secção IV.C será apresentado um exemplo de uso da memória estática. A informação sobre o DAC de vídeo e o descodificador de vídeo NTSC/PAL pode ser encontrada em [22,23].

O *bitstream* pode ser carregado para a FPGA com a ajuda da ferramenta *File Transfer Utility* disponibilizada pela Celoxica.

C. A placa TE-XC2Se da Trenz Electronic

A fig. 5 mostra os components principais da placa TE-XC2Se assim como os dispositivos externos que podem ser ligados à placa. A FPGA XC2S300E-6-FT256 (ver tabela 1) da família Spartan-IIE é o componente reconfigurável (o 6 no nome da FPGA indica o seu desempenho, e FT256 especifica o tipo do pacote e o número de pinos). As ligações básicas entre a FPGA e os componentes restantes da placa estão representadas na fig. 6. A placa possui dois conectores de expansão. O primeiro tem 26 pinos alguns dos quais são partilhados com o *display* LCD de 2x16 caracteres (os detalhes podem ser encontrados nas páginas 9-10 em [24]). O segundo conector é do tipo VG96 e tem 96 pinos descritos nas páginas 12-13 em [24]. A placa inclui dois osciladores que geram sinais de relógio de 48 MHz (este é necessário para a porta USB - *Universal Serial Bus* e para além disso pode servir de relógio para o utilizador) e 25 MHz (ver fig. 5,6). O segundo oscilador é substituível.

Os dados do utilizador podem ser visualizados num *display* LCD de 2x16 caracteres (ver [25,26] para os detalhes) e em 5 LEDs individuais. Os LEDs L_1,...,L_4 da fig. 6 são ligados com o CPLD [27], enquanto o LED L_5 é controlável directamente através do pino C15 da FPGA (ver fig. 6). O CPLD permite também receber os dados dos quatro botões B_1,...,B_4 e dos oito interruptores S_1,...,S_8 (ver fig. 6). Os componentes L_1,..,L_4, B_1,...,B_4, S_1,...,S_8 devem ser controlados através de escrita (L_1,..,L_4 – endereço 2) dos dados para o CPLD ou de leitura (B_1,...,B_4 – endereço 0, S_1,...,S_8 - endereço 1) dos dados do CPLD. Os números de componentes individuais tais como L_1, L_2,... correspondem aos bits 0,1,... dos dados a escrever no endereço em causa. Por exemplo, para ligar o LED L_4 e desligar os L_1, L_2 e L_3 é necessário especificar o valor "----1000" (onde '-' significa *don't care*) nos pinos P16,L16,L13,J14,G15,F14,E15,B16 da FPGA e escrever este valor no endereço 2 (i.e. "10") definido nos pinos M10,T10 da FPGA (ver o canto inferior direito na fig. 6).

A placa pode controlar a porta série RS232 (ver [16] para os detalhes) e visualizar os dados num monitor VGA (ver as fig. 5 e 6) [17,28,29]. Para programar a FPGA é preciso carregar a configuração na memória *Flash* indestrutível [30] através da porta USB (ver fig. 5). A porta USB fornece também a alimentação para a placa quando o interruptor respectivo está ligado, sendo o estado do interruptor reflectido no LED de alimentação.

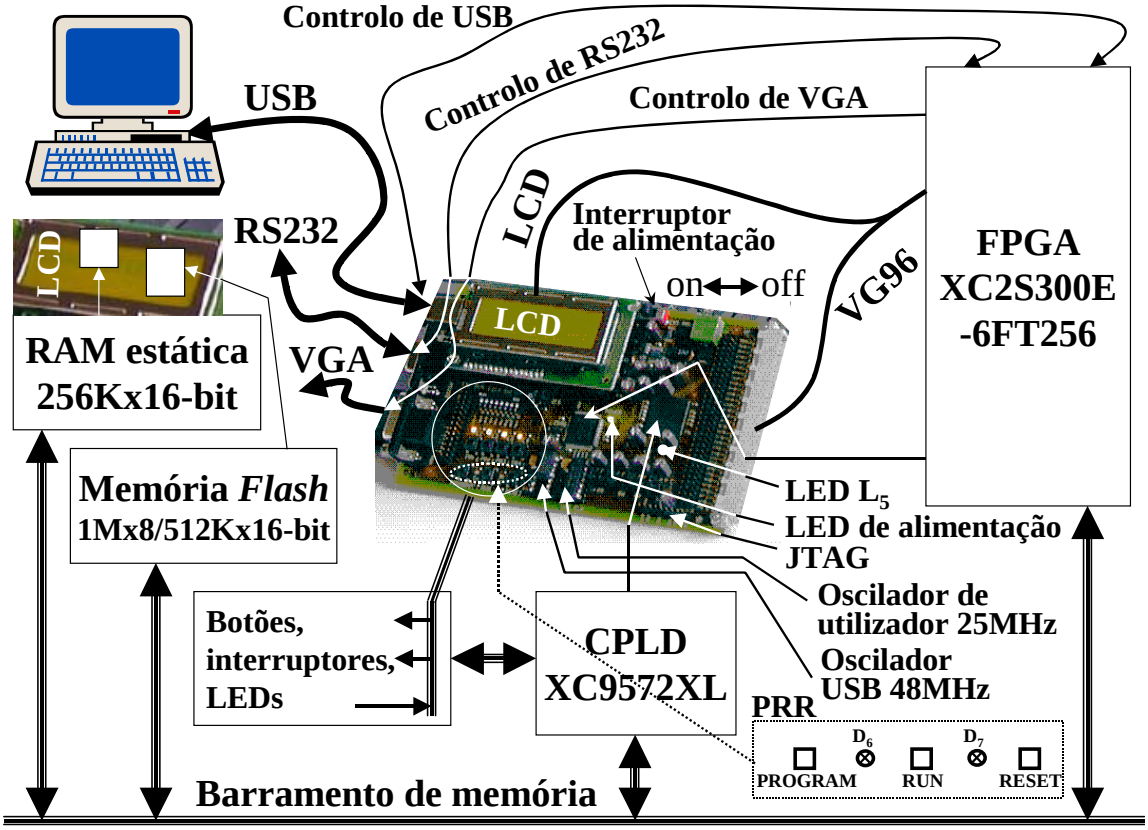

Fig. 5 – Components principais da placa TE-XC2Se e os dispositivos externos que podem ser ligados a esta

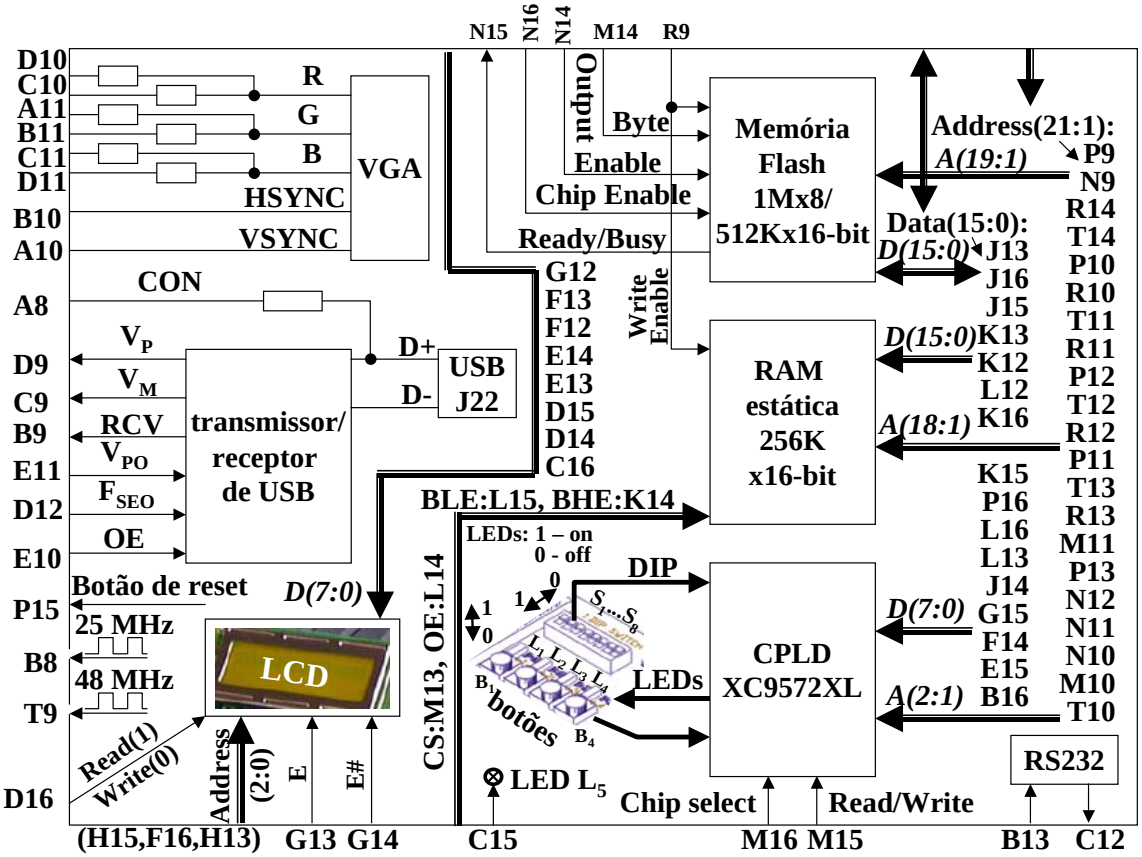

Fig. 6 – Ligações básicas entre a FPGA e os componentes restantes da placa TE-XC2Se

Existem dois tipos de configurações: a configuração de fábrica e a do utilizador. Ao ligar a alimentação é automaticamente carregada para a FPGA a configuração do utilizador. Ao pressionar o botão PROGRAM (ver fig. 5), carrega-se a configuração de fábrica o que permite reprogramar a memória *Flash* através de USB. Para reconfigurar a FPGA a partir da memória *Flash* é necessário pressionar o botão RUN (ver fig. 5). Ao pressionar o botão RESET (ver fig. 5) o valor '0' é enviado para o pino P15 da FPGA (ver fig. 6). O estado da placa é indicado em dois LEDs D_6 (configuração de fábrica) e D_7 (configuração do utilizador) mostrados na fig. 5. O transmissor/receptor de USB [31] está ligado à FPGA. A informação adicional sobre interface USB pode ser encontrada em [32]. É permitido personalizar a funcionalidade da porta USB quando a FPGA executa a configuração do utilizador. A placa possui também memória estática [33] ligada à FPGA como se pode ver nas fig. 5 e 6.

O *bitstream* pode ser carregado para a memória *Flash* da placa com a ajuda da ferramenta *TEprog* disponível da Trenz electronic. Para tal é necessário realizar os passos seguintes: ligar a placa à porta USB do computador, ligá-la à alimentação (ver o "Interruptor de alimentação" na fig. 5), pressionar o botão PROGRAM (ver fig. 5), executar *TEprog.exe*:

TEprog name_of_bitstream_file.bit

Como resultado, o *bitstream* será transferido para o espaço de configuração de utilizador da memória *Flash* (no endereço 0x40000) [24].

D. A placa XSA100 da XESS

Na fig. 7 estão representados os components principais da placa XSA100. Esta inclui uma FPGA XC2S100-TQ144 (ver tabela 1) da família Spartan-II (a abreviatura TQ144 especifica o tipo de pacote e o número de pinos).

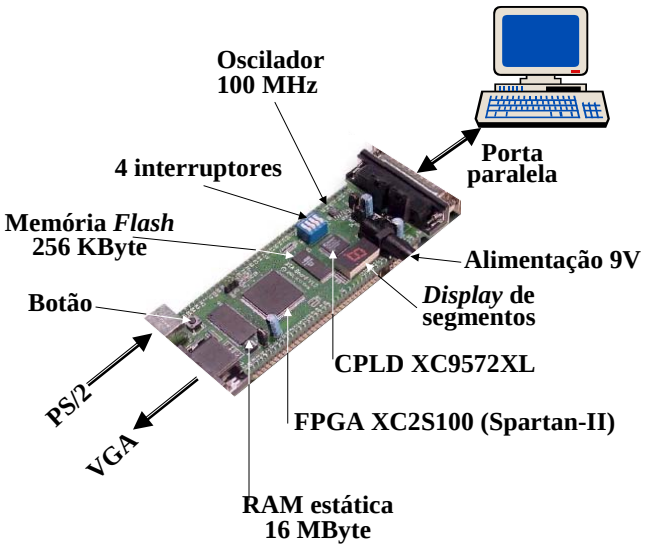

Fig. 7 – Placa XSA100 da XESS

A placa XSA100 é a mais simples comparando-a com as consideradas acima, portanto serão descritas apenas as suas características básicas. O CPLD XC9572XL implementa a interface entre a porta paralela do computador e os outros componentes da placa. A memória *Flash* possibilita o armazenamento persistente dos dados e dos *bitstreams* de configuração. A porta PS/2 permite comunicar com dispositivos tais como teclado e rato. O carregamento dos *bitstreams* e a comunicação com o computador efectuam-se através da porta paralela. Todos os detalhes podem ser encontrados em [34]. O teste e a programação da placa realizam-se com a ajuda de ferramentas da XESS [34]. Os resultados de desenvolvimento de vários dispositivos úteis com base em placas da XESS são apresentados em [35].

III. FERRAMENTA ISE 5. EXEMPLOS DE DESENVOLVIMENTO DE CIRCUITOS PARA AS FPGAS DA FAMÍLIA SPARTAN-IIE

Nesta secção são apresentadas as características básicas da ferramenta de software ISE 5 da Xilinx e descritos todos os passos necessários para o desenvolvimento e a implementação de circuitos digitais com base em FPGA da família Spartan-IIE. Na qualidade de plataforma de hardware será utilizada a placa TE-XC2Se considerada na subsecção *C* da secção II.

A. Ferramenta ISE e os passos básicos de desenvolvimento

Nas secções III e IV é demonstrado como se pode utilizar as ferramentas seguintes, que fazem parte da ISE 5:

- projecto baseado em VHDL incluíndo os tópicos seguintes:
  - criação do módulo VHDL no nível superior da hierarquia;
  - geração do ficheiro de restrições do utilizador (*User Constraints File* - UCF);
  - implementação de módulos VHDL sintetizáveis descritos a nível comportamental;
  - implementação de módulos VHDL sintetizáveis descritos a nível estrutural;
  - criação de módulos de biblioteca para possibilitar o desenvolvimento hierárquico;
  - ligação com as bibliotecas VHDL;
- projecto esquemático incluíndo os tópicos seguintes:
  - criação de módulos no editor esquemático;
  - geração de módulos de biblioteca para possibilitar o desenvolvimento hierárquico;
  - utilização de bibliotecas esquemáticas;
- simulação em *ModelSim* incluíndo os pontos seguintes:
  - criação de ficheiros com entradas de teste (*testbenches*);

  - fluxo de simulação funcional;
- desenvolvimento de máquinas de estados finitos (MEF);
- utilização da ferramenta *Core Generator*;
- implementação do projecto.

Serão considerados os passos de desenvolvimento seguintes:

- descrição duma entidade de projecto utilizando qualquer combinação de ferramentas assim como o VHDL, o editor esquemático, o editor de MEF, o *Core Generator*, e as bibliotecas existentes na ISE 5 e as definidas pelo utilizador;
- simulação da entidade de projecto com a ajuda de *ModelSim*;
- especificação de restrições;
- síntese e implementação do projecto, i.e. geração do *bitstream* a ser carregado na FPGA a fim de construir o dispositivo físico que poderá ser testado em hardware.

B. Desenvolvimento de um circuito trivial

O primeiro exemplo demonstra os passos necessários para construir um circuito trivial que permite fazer com que o LED L_5 da fig. 5. fique intermitente. A fig. 8 representa a sequência das acções que devem ser executadas para criar um projecto novo em ISE 5:

1) seleccionar o menu *File*;
2) escolher a opção *New Project*;
3) aparecerá a janela de diálogo *New Project*;
4) dar o nome ao projecto, por exemplo *trenz_trivial1*;
5) especificar as características do projecto, nomeadamente a família, o nome, o tipo do pacote e o desempenho da FPGA na qual vamos implementar o circuito, e o tipo do projecto (*Spartan-IIE*, *XC2S300e*, *FT256*, *6*, *XST VHDL*);
6) carregar no botão *OK*;
7) o projecto *trenz_trivial1* será criado.

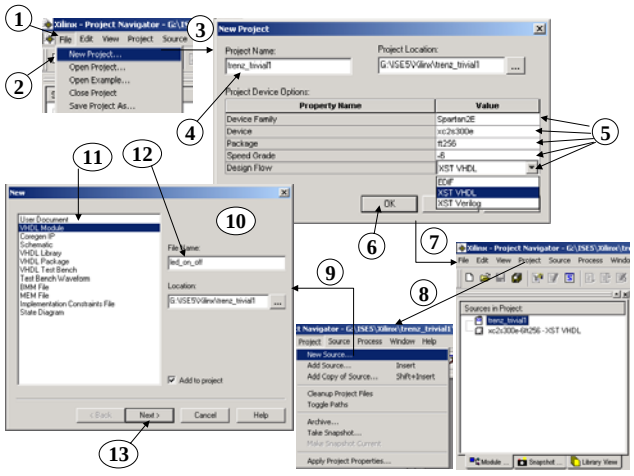

Fig. 8 – Sequência dos passos para criar um projecto novo em ISE 5

Há três opções possíveis para o tipo do projecto. Neste artigo consideram-se duas destas: *XST VHDL* e *EDIF*.

O projecto irá incluir apenas um ficheiro simples descrito em VHDL. As fig. 8 e 9 representam as sequências de passos necessários para criar este ficheiro:

8) seleccionar o menu *Project*;
9) escolher a opção *New Source*;
10) aparecerá a janela de diálogo *New*;
11) seleccionar a opção *VHDL Module*;
12) escrever o nome do ficheiro, por exemplo, *led_on_off*;
13) pressionar o botão *Next*;
14) aparecerá a janela de diálogo *Define VHDL Source* (ver fig. 9);
15) especificar as portas de entrada e de saída; são precisas duas entradas (*clk*, *reset*) e uma saída (*led*);
16) pressionar o botão *Next*;
17) pressionar o botão *Finish*;
18) o módulo *led_on_off* será incluído no projecto.

Em todos os exemplos seguintes recorreremos ao mesmo método ilustrado para especificar a sequência dos passos a realizar. Portanto é de notar que as descrições textuais serão se possível omitidas.

As opções *MSB* (*Most Significant Bit*) e *LSB* (*Less Significant Bit*) existentes na janela de diálogo *Define VHDL Sources*, permitem definir os barramentos. Um exemplo é dado na parte inferior da fig. 9.

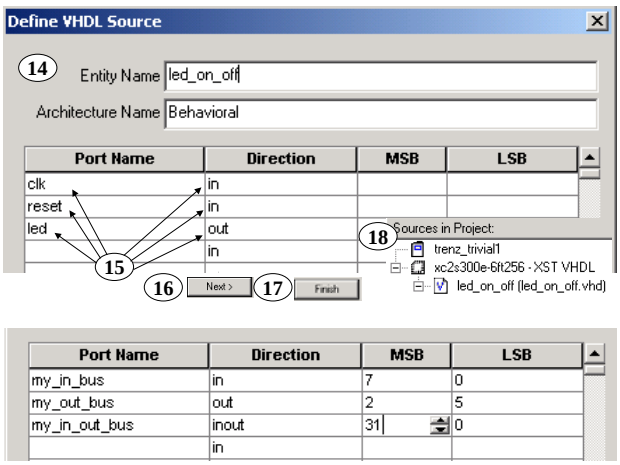

Fig. 9 – Sequência dos passos para criar um ficheiro em VHDL

Se deslocar o cursor sobre o nome *led_on_off* (ver ponto 18 na fig. 9) e carregar duas vezes no botão esquerdo do rato, aparecerá o seguinte código em VHDL que ainda está incompleto:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity led_on_off is
   Port ( clk : in std_logic;
```

```
   reset : in std_logic;
   led : out std_logic);
end led_on_off;

architecture Behavioral of led_on_off is
begin
end Behavioral;
```

Para completar o módulo VHDL é necessário modificar a sua arquitectura de modo seguinte:

```
architecture Behavioral of led_on_off is
   signal internal_clock : std_logic_vector (25
   downto 0);
begin

process(clk,reset)
begin
   if reset = '0' then
      internal_clock <= (others=>'0');
   elsif clk'event and clk = '1' then
      internal_clock <= internal_clock+1;
   end if;
end process;

led <= internal_clock(internal_clock'left);

end Behavioral;
```

Este código põe todos os bits do sinal *internal_clock* a zero caso o sinal *reset* seja igual a '0'. Caso contrário, o sinal de relógio externo *clk* (que é de 48MHz para o exemplo) é dividido por 2^{25}. Isto permite fazer com que o LED fique periodicamente ligado/desligado com um intervalo de ~2/3 de segundo.

Agora pode-se sintetizar o circuito ao carregar duas vezes com o botão esquerdo do rato sobre a opção *Synthesize* (ver ponto 19 na fig. 10). A fig. 10 representa o resultado desta operação.

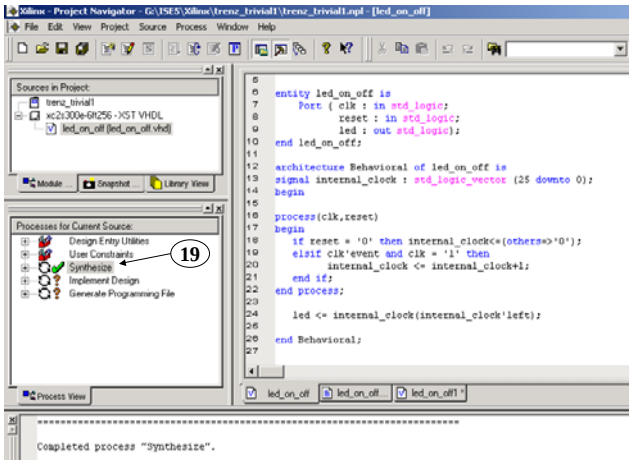

Fig. 10 – Síntese do módulo VHDL

A fim de implementar o circuito é preciso especificar os pinos da FPGA que vão corresponder às entradas e saídas externas, tais como *clk*, *reset* e *led*. As fig. 11 e 12 ilustram todos os passos (ver os pontos 20-40) necessários para a geração do ficheiro de restrições do utilizador *led_on_off.ucf* e para a descrição dos parâmetros do relógio.

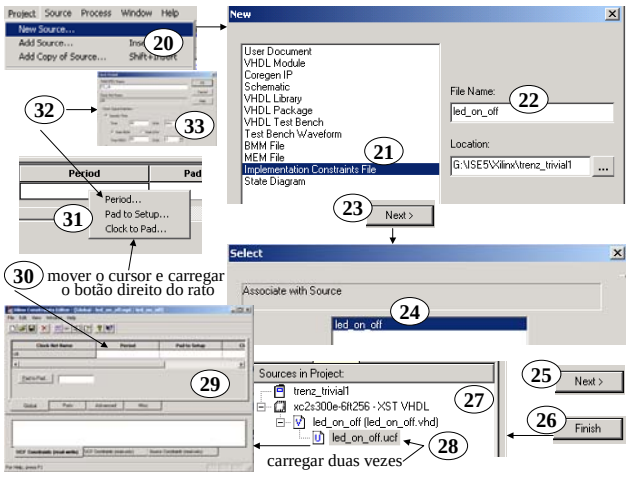

Fig. 11 – Criação do ficheiro de restrições do utilizador

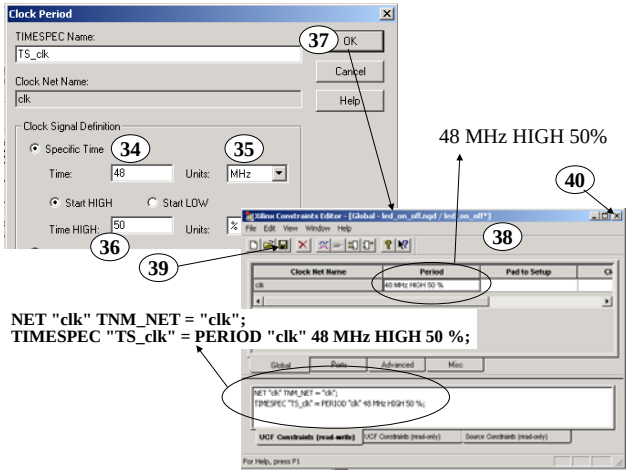

Fig. 12 – Especificação dos parâmetros do relógio

A fig. 13 mostra a sequência dos passos (ver os pontos 41-53) a realizar a fim de estabelecer a correspondência entre as entradas/saídas do circuito e os pinos da FPGA (ver também a fig. 6). Na fig. 13 são utilizadas as abreviaturas seguintes: *rdc* – o botão direito do rato deve ser carregado duas vezes; *ldc* - o botão esquerdo do rato é clicado duas vezes; *rc* - o botão direito do rato deve ser carregado uma vez; *lc* - o botão esquerdo do rato é pressionado uma vez. Os pinos podem ser atribuídos quer escrevendo o número respectivo do pino (ver o ponto 44 na fig. 13) quer seleccionando o sinal (ver o ponto 45) e movendo-o (*drag and drop*) para o local desejado (ver a sequência dos passos 46-49). O ponto 50 conduz ao resultado final (ver também a fig. 6). Agora o ficheiro *led_on_off.ucf* tem o conteúdo seguinte:

```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 48 MHz HIGH 50%;
#PACE: Start of Constraints extracted by …
NET "reset" LOC = "P15";
NET "led" LOC = "C15";
```

```
NET "clk" LOC = "T9";
```

Este ficheiro pode ser aberto ao executar o ponto 53 da fig. 13. O símbolo # permite escrever os comentários, possibilitando deste modo remover temporariamente qualquer pino.

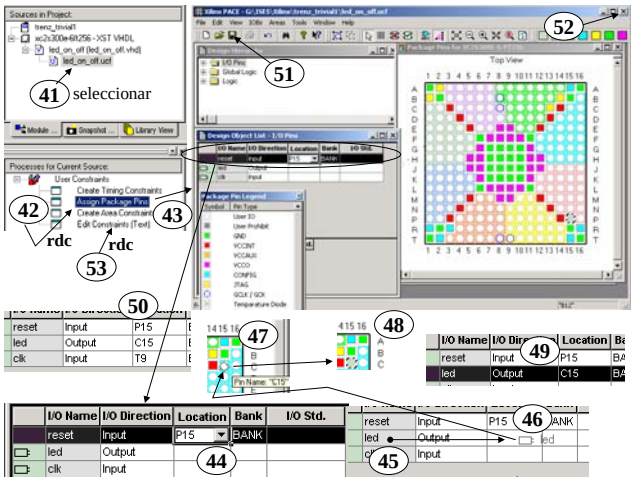

Fig. 13 – Atribuição dos pinos da FPGA

A fig. 14 demonstra os passos restantes (ver os pontos 54-61) necessários para implementar e testar o dispositivo físico respectivo. O LED L_5 está ligado se o sinal respectivo tiver o valor '1' (ver fig. 6). O ponto 58 na fig. 14 ilustra como executar o programa da Trenz electronic a fim de carregar o *bitstream* para a FPGA. A seguir é necessário pressionar a botão RUN (ver o ponto 59).

O exemplo apresentado é muito simples. Contudo este permite começar a trabalhar rapidamente com ISE 5 e perceber o fluxo de desenvolvimento respectivo. A descrição detalhada de ISE 5 pode ser encontrada em [36-38]. Os exemplos restantes, considerados nas secções seguintes, irão apresentar algumas novas capacidades do ISE 5.

C. Interacção com os botões, interruptores e LEDs através do CPLD

Para todos os exemplos apresentados nesta secção vamos tentar dar às variáveis e aos sinais os mesmos nomes que são utilizados pela Trenz electronic [24,29].

O segundo exemplo demonstra a interacção com os botões B_1,...,B_4, os interruptores S_1,...,S_8 e os LEDs L_1,...,L_4 (ver fig. 6). O projecto inclui um ficheiro VHDL *led_but_sw.vhd* e um ficheiro de restrições do utilizador *led_but_sw.ucf*. O ficheiro VHDL tem o código seguinte:

```
entity led_but_sw is
   Port (clk : in  std_logic; -- relógio 48 MHz
   rst : in  std_logic; -- reset
   cpld_rw : inout std_logic;
   -- cpld read/write (fig. 6)
   cpld_cs : out std_logic;  -- chip select
   a : out std_logic_vector(2 downto 1);
   d : inout   std_logic_vector(7 downto 0)  );
end led_but_sw;
-- a – endereço, d – dados (ver fig. 6)

architecture Behavioral of led_but_sw is
   signal state: std_logic_vector(3 downto 0);
   signal lled: std_logic_vector(7 downto 0);
   signal lpb   : std_logic_vector(7 downto 0);
   signal dipswitch: std_logic_vector(7 downto
   0);
begin

process(clk, rst) – o processo que descreve
begin      -- a sequência dos estados
   if rst= '0' then
      state<=(others=> '0'); -- state = "0000"
   elsif rising_edge(clk) then
      state<= state + 1; -- incrementar o estado
   end if;
end process;

process(clk, rst) – o processo que descreve
begin     -- a interacção com o CPLD (ver fig. 6)
   if rst= '0' then
      cpld_cs <= '1';
   elsif (clk='0' and clk'event) then
      case state is
         -- endereço 00 – ler o estado dos botões
         when "0000"=> a<= "00";
            cpld_cs <= '1'; --passivo
            cpld_rw <= '1'; --ler
         when "0001"=> cpld_cs <= '0'; --activo
         when "0010"=> lpb <= d;
         when "0011"=> cpld_cs <= '1'; --passivo
         -- endereço 10 – escrever os LEDs
         when "0100"=> a <= "10";
            cpld_rw <= '0';    --escrever
         when "0101"=> cpld_cs <= '0'; --activo
         when "0110"=> cpld_cs <= '1'; --passivo
         -- endereço 01 – ler o estado dos
         -- interruptores
         when "0111"=> a <= "01";
            cpld_cs <= '1'; --passivo
            cpld_rw <= '1'; --ler
         when "1000"=> cpld_cs <= '0'; --activo
         when "1001"=> dipswitch <= d;
         when "1010"=> cpld_cs <= '1'; --passivo
         when others => cpld_cs <= '1'; --passivo
      end case;
      lled <= dipswitch; -- copiar o estado dos
      -- interruptores para os LEDs
      if (dipswitch(7)='1') then
         lled(3 downto 0)<= not lpb(3 downto 0);
      else null;
   end if;
   end if; -- se S7=0 então o estado dos
   -- interruptores é copiado para os LEDs
end process;   -- se S7=1 então o estado dos
-- botões é copiado para os LEDs

process (lled,cpld_rw)  -- copiar o sinal lled
```

```
begin                    -- para o CPLD
   if (cpld_rw='0') then – se o sinal write é
   -- activo
      d <= lled;
   else
      d <= "ZZZZZZZZ"; -- alta impedância
   end if;
end process;

end Behavioral;
```

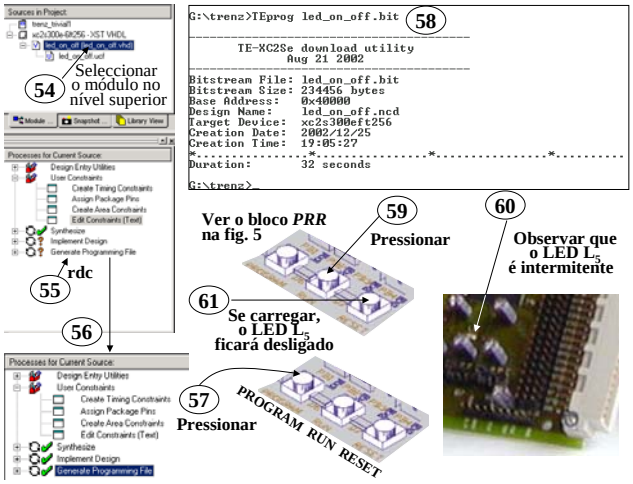

Fig. 14 – Implementação e teste do circuito

Os sinais internos *lled*, *lpb* e *dipswitch* retêm a informação sobre os LEDs (----$L_4L_3L_2L_1$), os botões (----$B_4B_3B_2B_1$) e os interruptores ($S_8S_7S_6S_5S_4S_3S_2S_1$) respectivamente (ver a subsecção *C* da secção II e a fig. 6 para os detalhes). O ficheiro de restrições do utilizador *led_but_sw.ucf* tem o código seguinte:

```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 48 MHz HIGH 50%;
NET "rst" LOC = "P15";
NET "d<7>" LOC = "P16";  NET "d<6>" LOC = "L16";
NET "d<5>" LOC = "L13";  NET "d<4>" LOC = "J14";
NET "d<3>" LOC = "G15";  NET "d<2>" LOC = "F14";
NET "d<1>" LOC = "E15";  NET "d<0>" LOC = "B16";
NET "cpld_rw" LOC = "M15";
NET "cpld_cs" LOC = "M16";
NET "clk" LOC = "T9";
NET "a<2>" LOC = "M10"; NET "a<1>" LOC = "T10";
```

Para testar o circuito é necessário realizar as mesmas acções que são mostradas na fig. 14. Agora se o interruptor S_7 for igual a '0', então o carregamento de qualquer botão vai ligar o LED respectivo, e se S_7=1, então os interruptores S_1,...,S_4 podem ligar/desligar os LEDs correspondentes.

D. Interacção com o LCD

O circuito a projectar está representado na fig. 15. Este executa uma operação de adição simples A+B=C, onde A=0,1,…,9, B=0,1,…,9 e C= 0,1,…,18. Os valores A e B podem ser incrementados ao pressionar os botões B_1 e B_2 respectivamente. A linha A+B=C com os valores de A, B e o resultado da operação C será mostrada no *display* LCD (ver a parte esquerda da fig. 15). O valor binário C é convertido num valor BCD com a ajuda duma ROM construída com base em memória distribuída da FPGA (i.e. com base em LUTs dos CLBs). O bloco ROM foi criado com a ferramenta *Core Generator*.

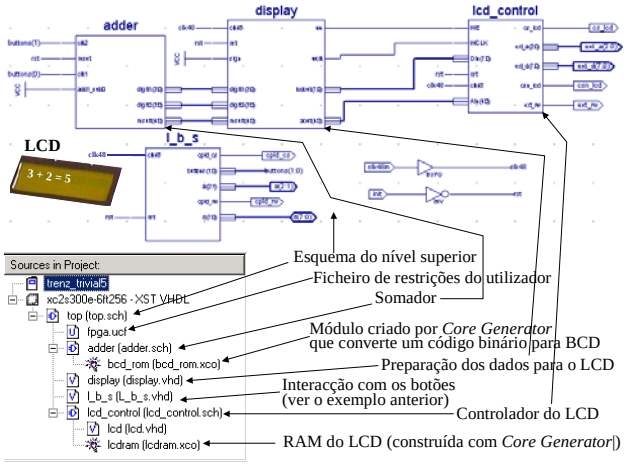

Fig. 15 – Circuito que soma dois valores e visualiza o resultado no LCD

As fig. 16-18 demonstram a sequência dos passos (os pontos 1-46) necessários para criar o conversor de código binário para BCD com a ajuda de *Core Generator*. Como resultado, o módulo *bcd_rom.xco* será incluído no projecto. Se carregar no nome do ficheiro com o botão esquerdo do rato e expandir a hierarquia para *coregen* na janela inferior esquerda, poderá ver o código VHDL respectivo construído por *Core Generator* (seleccione a opção *View VHDL functional model*).

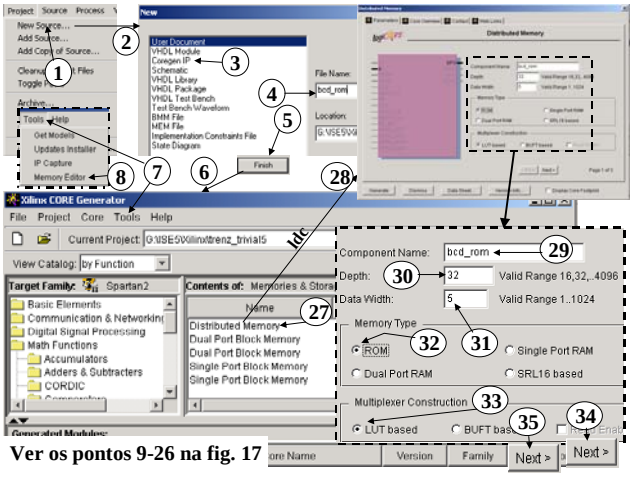

Fig. 16 – Criação do módulo do conversor BCD com a ajuda de *Core Generator* (passos 1-8 e 27-35)

A mesma sequência dos passos é executada para criar a RAM para o LCD. Para a RAM pode-se escolher o tipo "*Dual Port Block Memory*" (ver a terceira linha no ponto 27 na fig. 16). Para este bloco atribuem-se os parâmetros seguintes: largura da porta A – 8 bits, largura da porta B –

8 bits, profundidade para ambas as portas – 32, a porta A – só de escrita, a porta B – só de leitura, o sinal de relógio é activo nas transições ascendentes, o sinal de *write enable* – é activo quando a '1'. O módulo com as características especificadas requer apenas um bloco de RAM da FPGA (ver tabela 1). Como resultado, o ficheiro *lcdram.xco* será incluído no projecto.

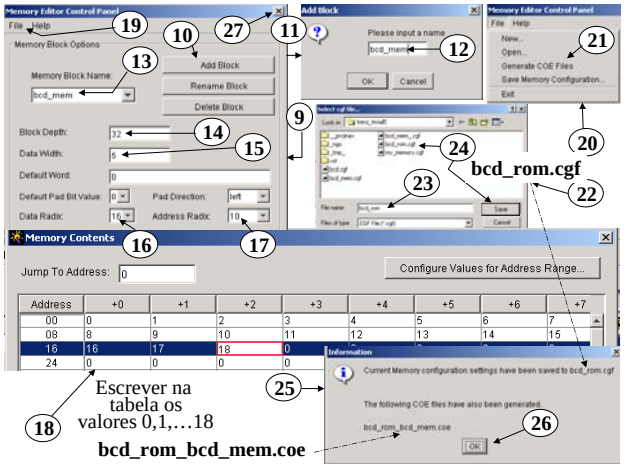

Fig. 17 – Criação do módulo do conversor BCD com a ajuda de *Core Generator* (passos 9-26)

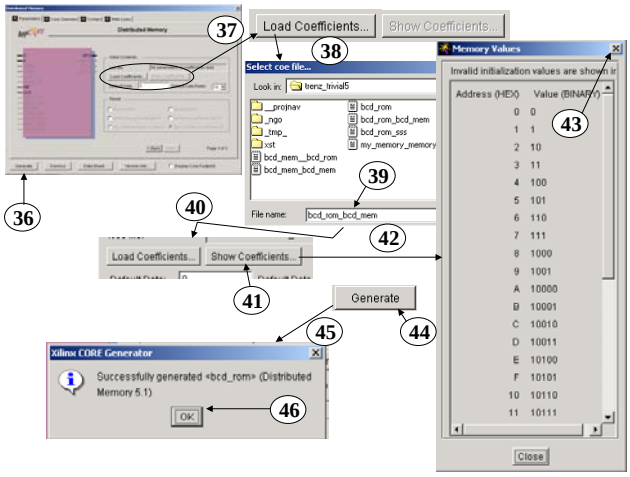

Fig. 18 – Criação do módulo do conversor BCD com a ajuda de *Core Generator* (passos 36-46)

A seguir cria-se o ficheiro esquemático para o somador (ver fig. 15). Para tal é necessário repetir os pontos 1,2 da fig. 16, seleccionar a opção *Schematic* no ponto 3 e especificar o nome *adder* no ponto 4. Como resultado, aparecerá a janela do editor esquemático. A fig. 19 demonstra como se pode encontrar o módulo do conversor BCD e modificar o seu aspecto gráfico se for necessário. O esquema final do somador está representado na fig. 20. Na mesma figura são explicados alguns comandos do editor esquemático. A fig. 21 demonstra uma série de opções disponíveis no editor. Vários pontos desta figura ilustram como se podem realizar as operações seguintes (todos os detalhes são apresentados em [38]):

- encontrar um elemento (por exemplo, um *D flip-flop*) na biblioteca do sistema (ver os pontos 1-6);
- desenhar um fio ou um barramento (ver os pontos 7-9);
- especificar o nome dum barramento (ver os pontos 10-13);
- adicionar as derivações (*taps*) a um barramento (ver os pontos 14-20);
- seleccionar e apagar os elementos esquemáticos (ver os pontos 21-24).

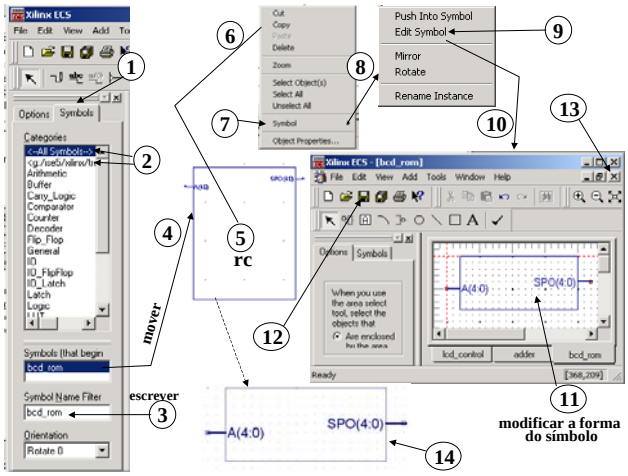

Fig. 19 – Modificação de aparência dum símbolo

Para descrever o componente *lcd_control.sch* são aplicados passos semelhantes (ver fig. 15 e 22). O bloco *lcd* que controla o LCD [25,26] foi adoptado dum exemplo de demonstração da Trenz electronic [29]. O código VHDL deste bloco está disponível na WebCT [39].

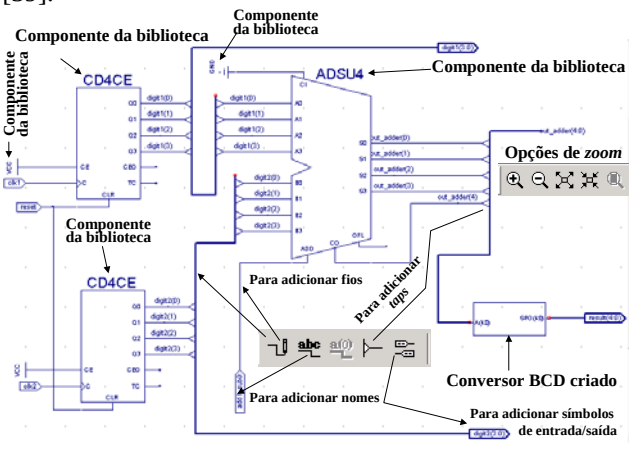

Fig. 20 – Esquema do somador e os comandos mais utilizados do editor esquemático

O código do bloco *l_b_s* é semelhante ao do bloco *led_on_off* considerado na subsecção III.B. As únicas alterações são que o ficheiro *l_b_s.vhd* inclui a linha adicional na declaração das portas:

```
buttons  : out std_logic_vector(1 downto 0);
```

e contém o commando seguinte logo depois da linha *when "0010" => lpb <= d;*

```
buttons(1 downto 0) <= not d(1 downto 0);
```

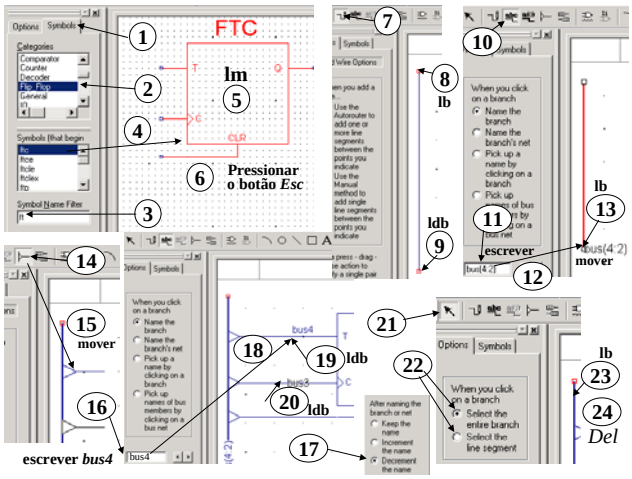

Fig. 21 – Opções do editor esquemático

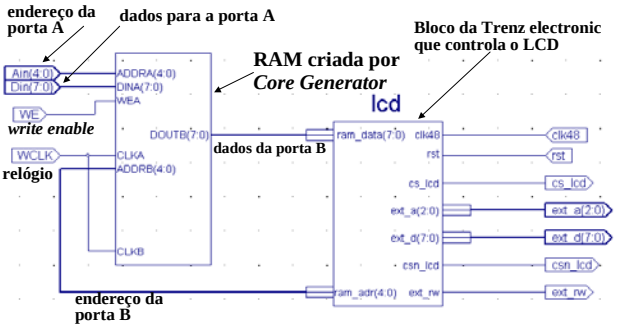

Fig. 22 – Esquema do componente *lcd_control*

O bloco *display* da fig.15 tem o código VHDL seguinte:

```
entity display is
   Port (clk48 : in  std_logic; -- relógio 48 MHz
   rst : in  std_logic;  -- reset
   we : out std_logic; -- write enable para RAM
   wclk : out std_logic; -- relógio para RAM
   lcdout : out std_logic_vector(7 downto 0);
   -- dados de RAM
   aout : out std_logic_vector(4 downto 0);
   -- endereço de RAM
   sign : in  std_logic;
   -- sinal da operação (+ ou -)
   digit1 : in std_logic_vector(3 downto 0);
   -- operando 1
   digit2 : in std_logic_vector(3 downto 0);
   -- operando 2
   result : in std_logic_vector(4 downto 0)
   -- resultado
);
end display;

architecture Behavioral of display is
   signal state : std_logic_vector(4 downto 0);
   signal line : string(1 to 3);
   -- string para o sinal (" + " ou " - ")
   constant line1: string(1 to 3) := " = ";
   -- string para " = "
begin

process(clk48, rst) -- sequência dos estados
begin
   if rst= '1' then
      state <= (others=> '0');
   elsif rising_edge(clk48) then
      state <= state + 1;
   end if;
end process;

we <= '1';
wclk <= not clk48;
-- relógio para as portas A,B da RAM
aout <= state; -- o estado representa o endereço
-- da porta A da RAM

process(clk48, rst) -- escreve dados na
-- porta A da RAM
begin
   if sign = '1' then line <= " + ";
   -- o sinal é +
   else line <= " - "; -- o sinal é - (não é
   -- utilizado neste exemplo)
   end if;

   if rst= '1' then
      null; -- a operação não é executada
   elsif rising_edge(clk48) then -- se o sinal de
   -- reset for inactivo
      case state is -- escreve 10 caracteres para
      -- a primeira linha do LCD
         when "11111"=>
            lcdout <= "0011" & digit1;
         when "00000"=>
            lcdout<=std_logic_vector(to_unsigned(
            character'pos(line(1)), 8));
         when "00001"=>
            lcdout<=std_logic_vector(to_unsigned(
            character'pos(line(2)), 8));
         when "00010"=>
            lcdout<=std_logic_vector(to_unsigned(
            character'pos(line(3)), 8));
         when "00011"=>
            lcdout <= "0011" & digit2;
         when "00100"=>
            lcdout<=std_logic_vector(to_unsigned(
            character'pos(line1(1)), 8));
         when "00101"=>
            lcdout<=std_logic_vector(to_unsigned(
            character'pos(line1(2)), 8));
         when "00110"=>
            lcdout<=std_logic_vector(to_unsigned(
            character'pos(line1(3)), 8));
         when "00111"=>
```

```
            lcdout<="0011000" & result(4);
         when "01000"=>
            lcdout<="0011" & result(3 downto 0);
         when others  =>
            lcdout <= x"20";-- escreve os espaços
      end case;
   end if;
end process;

end Behavioral;
```

As linhas do género *lcdout <= "0011" & digit1;* escrevem um byte (neste caso, o byte *"0011" & digit1*) na RAM. Os 4 bits mais significativos do byte são *0011*. O código ASCII do dígito "0" é *00110000*. Se concatenar o código binário *0011* e os códigos dos algarismos respectivos (de 0 a 9), i.e. 0 (que tem o código binário *0000*), 1 (que tem o código binário *0001*), 2 (que tem o código binário *0010*), etc., serão recebidos os códigos ASCII dos dígitos 0,1,...,9 que são precisos para o LCD. As posições da *string line*: *line(1)*, *line(2)* e *line(3)* contêm "espaço", "sinal" e "espaço" respectivamente. O resultado é composto por dois dígitos que devem ser escritos separadamente. Finalmente, "20"="00100000" é o código hexadecimal (indicado pelo especificador *x*) do espaço. Este código deve preencher todas as posições do LCD que não são utilizadas (i.e. 32-10 posições). Os endereços da RAM são formados com base em estados. Os dados são escritos na porta *A* do bloco de RAM *lcdram* criado com *Core Generator*. O bloco *lcd* lê os dados da porta *B* da RAM e envia-os para o LCD.

Agora o projecto é idêntico ao representado na fig. 15. Portanto pode-se gerar o *bitstream* e implementar o circuito realizando para tal os mesmos passos que foram considerados nas subsecções anteriores. O ficheiro de restrições do utilizador para este exemplo tem o código seguinte:

```
NET "clk48in" TNM_NET = "clk48in";
TIMESPEC "TS_clk48in"=PERIOD "clk48in" 48 MHz HIGH 50 %;
NET "clk48in" LOC = "t9";
NET "init" LOC = "p15";       # o botão RESET
NET "ext_a<2>" LOC = "H15";   # endereço para
NET "ext_a<1>" LOC = "F16";   # o LCD
NET "ext_a<0>" LOC = "H13";
NET "csn_lcd" LOC = "G14";    # controlo do LCD
NET "cs_lcd" LOC = "G13";
NET "ext_rw" LOC = "D16";     # LCD read/write
NET "ext_d<7>" LOC = "G12";   # dados para o LCD
NET "ext_d<6>" LOC = "F13";
NET "ext_d<5>" LOC = "F12";
NET "ext_d<4>" LOC = "E14";
NET "ext_d<3>" LOC = "E13";
NET "ext_d<2>" LOC = "D15";
NET "ext_d<1>" LOC = "D14";
NET "ext_d<0>" LOC = "C16";
NET "cpld_cs" LOC = "m16";    # CPLD chip select
NET "cpld_rw" LOC = "m15";    # CPLD read/write
NET "d<7>" LOC = "P16";       # dados para CPLD
NET "d<6>" LOC = "L16";
NET "d<5>" LOC = "L13";  NET "d<4>" LOC = "J14";
NET "d<3>" LOC = "G15";  NET "d<2>" LOC = "f14";
NET "d<1>" LOC = "e15";  NET "d<0>" LOC = "b16";
NET "a<1>" LOC = "t10";    # endereço para o
NET "a<2>" LOC = "m10";    # CPLD
```

E. Projecto de MEFs e simulação

A fig. 23 representa o circuito que se pretende implementar. Este contém o bloco *l_b_s* considerado acima que interage com os interruptores, botões, LEDs e um bloco novo *lcd_struc* que controla os LEDs de maneira mostrada na parte inferior da fig. 23.

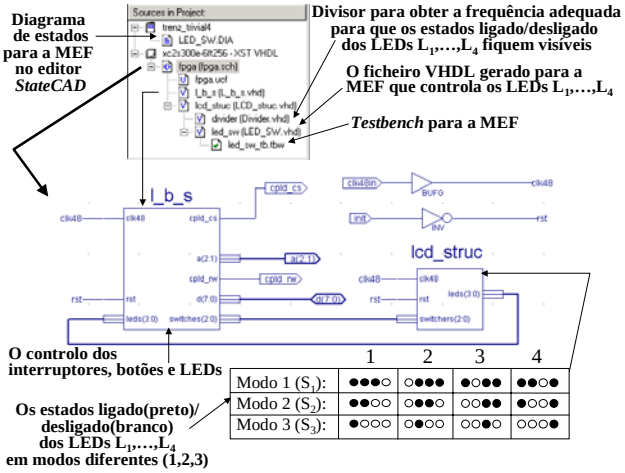

Fig. 23 – Circuito que controla os LEDs da placa

O bloco *lcd_struc* é descrito em VHDL estrutural, sendo o código VHDL respectivo apresentado na fig. 24 a qual além disso mostra todos os componentes gráficos (ver os blocos *Divider* e *LED_SW*) e todas as ligações entre estes.

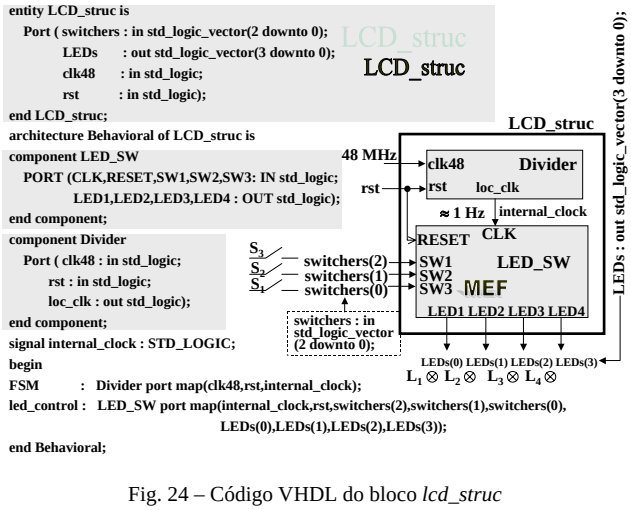

Fig. 24 – Código VHDL do bloco *lcd_struc*

O bloco *LED_SW* é uma MEF. O diagrama de estados para a MEF criado com a ajuda do editor *StateCAD* da Xilinx está apresentado na fig. 25. Para arrancar o editor *StateCAD* é necessário adicionar uma fonte nova ao projecto, i.e. repetir os passos 1-5 da fig. 16 com as

modificações seguintes: ponto 3 – seleccionar a opção *State Diagram*; ponto 4 – especificar o nome *LED_SW*. A seguir, cria-se o diagrama de estados mostrado na fig. 25, grava-se no disco e compila-se para o código VHDL que é incluído no projecto (ver o ficheiro *LED_SW.vhd* na fig. 23). A fig. 25 explica também as opções mais utilizadas do editor de diagramas de estados.

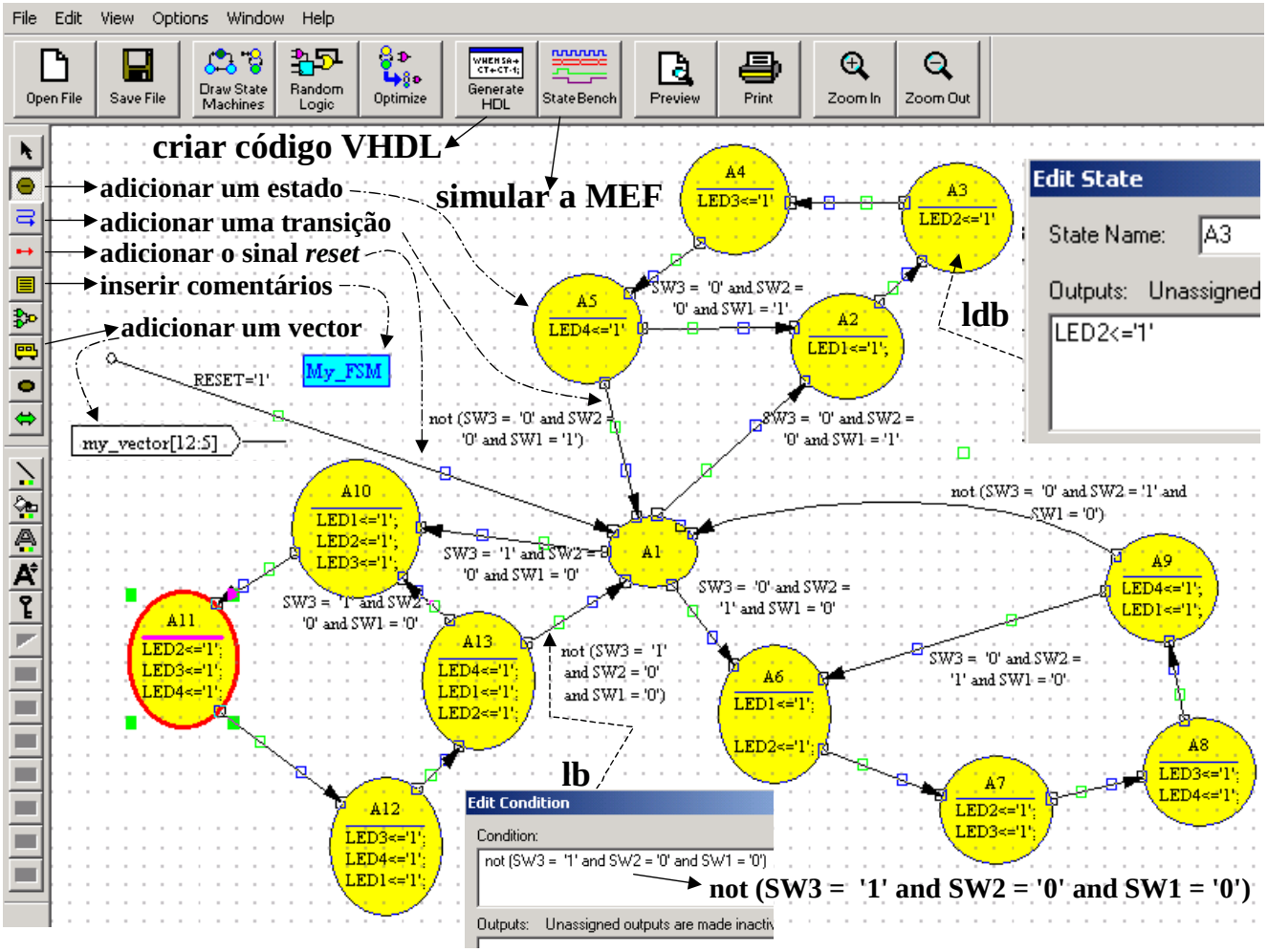

Fig. 25 – Diagrama de estados da MEF criado no editor *StateCAD*

O passo seguinte é criar um *testbench* para efectuar a simulação comportamental em *ModelSim*. Para criar um *testbench* é preciso adicionar uma fonte nova ao projecto, i.e. repetir os passos 1-5 da fig. 16 com as modificações seguintes: ponto 3 – seleccionar a opção *Test Bench Waveform*; ponto 4 – especificar o nome *led_sw_tb*. A seguir o *testbench* deve ser associado com o componente *led_sw* e finalmente selecciona-se o botão *Finish* (ver o ponto 5 da fig. 16).

As fig. 26, 27 mostram todos os passos seguintes necessários (ver os pontos 1-8 para gerar os resultados da simulação esperados, e os pontos 9,10 para simular o modelo VHDL comportamental). A fig. 27 explica também algumas opções úteis de *ModelSim*. É possível adicionar os pontos de paragem (*breakpoints*) aos sinais (ver os passos 11-14) e realizar a simulação entre estes pontos (ver os passos 15-17). Pode-se apagar alguns sinais (ver os passos 18, 19) e inseri-los com a ajuda do rato (ver o ponto 20). Finalmente, é possível efectuar a simulação do modelo VHDL que já foi colocado e encaminhado (ver o ponto 21 na fig. 26). A descrição detalhada destas opções pode ser encontrada em [40].

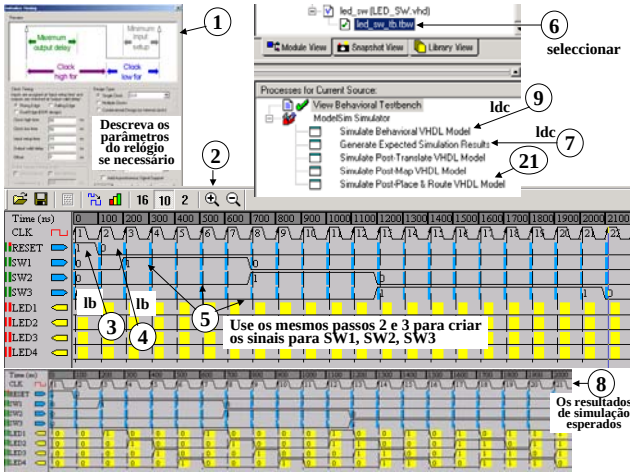

Fig. 26 – Resultados da simulação

É possível também efectuar a simulação em *StateCAD* [41]. A fig. 28 ilustra algumas operações que podem ser executadas. Os sinais na janela inferior podem ser alterados ao apontar o sinal e carregar duas vezes com o botão esquerdo do rato (ver os pontos 1-5 que demonstram também a simulação durante um ciclo de relógio). Os pontos 6-8 explicam como se efectua a simulação durante um período de tempo específico. O ponto 9 mostra o resultado final da simulação durante 1500 ns.

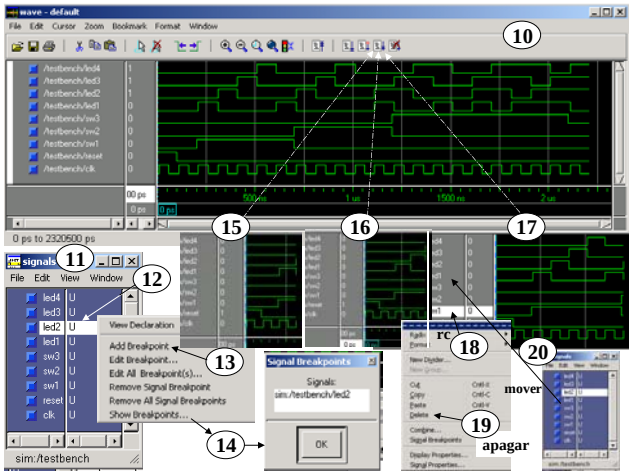

Fig. 27 – Simulação comportamental em *ModelSim*

O código para o divisor é igual ao considerado na subsecção III.B. Este código pode ser copiado para o projecto (para tal usam-se as opções *Project – Add Copy of Source*) e modificado ligeiramente (em particular são alterados os nomes das entradas *clk48* e *rst* e da saída *loc_clk*). O ficheiro de restrições do utilizador é construído com base em ficheiros considerados nas subsecções anteriores. Agora pode-se gerar o *bitstream* e implementar o circuito.

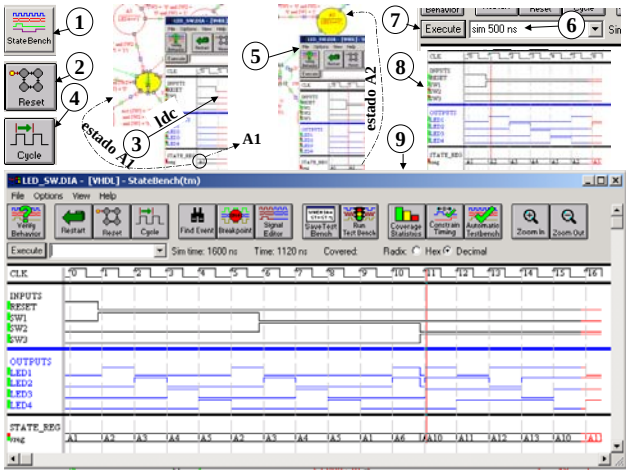

Fig. 28 – Simulação em *StateCAD*

F. Uso de DLLs

À medida que a densidade de FPGAs aumenta, a qualidade da distribuição do sinal de relógio no *chip* torna-se muito importante [42]. DLL (*Delay-Locked Loop*) permite resolver este problema e oferece mais algumas funcionalidades tais como a síntese da frequência (multiplicação/divisão do relógio) e o condicionamento do relógio (correcção do ciclo e deslocação de fases). Uma das funções do DLL é inserir um atraso entre o relógio de entrada – *CLKIN* e o relógio de realimentação – *CLKFB* (ver fig. 29) até que as duas transições coincidam fazendo com que os dois relógios fiquem deslocados de 360º (i.e. pondo-os em fase).

A fig. 29 representa o elemento da biblioteca *CLKDLL* que corresponde ao DLL. O sinal do relógio de entrada (*CLKIN*) pode proceder quer dum dos componentes *BUFG* ou *IBUFG*, quer do pino *IO_LVDS_DLL* (este é adjacente ao pino do relógio global). A entrada *CLKFB* precisa do sinal de realimentação para assegurar que a saída seja corrigida. A porta *CLKFB* é ligada a uma das saídas *CLK0*, *CLK2X* ou é controlada por componentes e pinos mencionados acima para o caso da entrada *CLKIN*. O pino *RST* (activo quando a '1') pode ser quer ligado a um sinal dinâmico quer posto a '0'. A saída *CLKDV* (divisão do relógio) é controlada pelo atributo *CLKDV_DIVIDE* que especifica o factor de divisão (ver fig. 1). A saída *LOCKED* indica se as saídas do DLL são válidas. Até que este sinal fique activo (i.e. a '1') os relógios de saída do DLL não devem ser utilizados pois podem exibir picos, *glitches*, etc. São precisos vários milhares dos ciclos de relógio para que o DLL atinja a estabilidade.

Os diagramas temporais para as saídas restantes do DLL estão apresentados na fig. 29. A fig. 1 ilustra as localizações dos DLLs (0, 1, 2 e 3). O macro *BUFGDLL* (ver o circuito cercado por um rectângulo traçejado na fig. 29) pertence à biblioteca da Xilinx, possui uma entrada e uma saída, e constitui a maneira mais simples de distribuir o sinal do relógio proveniente de uma entrada externa. Os

outros macros que servem para os mesmos fins são *CLKDLL* (ver fig. 29), *CLKDLLE* e *CLKDLLHF*.

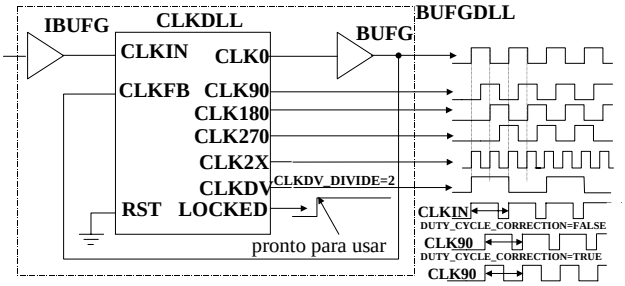

Fig. 29 – Estrutura e funcionamento de DLL

A fig. 30 representa a sequência dos passos necessários para modificar as propriedades de DLL. O circuito *d_ff_4* mostrado na parte inferior da fig. 30, ilustra uma possibilidade de utilização dos sinais do DLL para transferir dados entre vários componentes de memória.

A fig. 31 demonstra um circuito que efectua várias manipulações com os sinais de relógio. O circuito inclui cinco divisores (ver o código VHDL na subsecção III.B) cujas saídas estão ligadas com os LEDs para visualizar as alterações na frequência do relógio.

A fig. 32 representa o componente *f4x48m* que multiplica a frequência externa por 2 e por 4. Isto permite gerar sinais de relógio internos de 96MHz e 192MHz respectivamente. O ficheiro de restrições do utilizador é construído com a ajuda da fig. 6. Informação adicional pode ser encontrada em [42]. Para além disso, vários exemplos de uso de DLLs (descritos em VHDL/Verilog) estão disponíveis em [43].

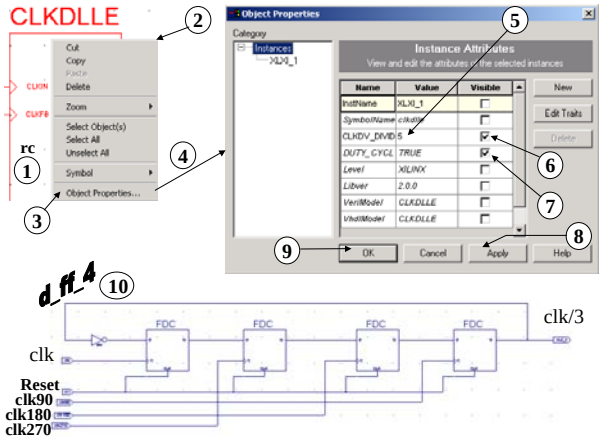

Fig. 30 – Modificação das propriedades de DLL

Qualquer módulo esquemático ou VHDL, pode ser incluído na biblioteca do utilizador. Para adicionar um módulo à biblioteca é necessário seleccionar o elemento respectivo e carregar duas vezes com o botão esquerdo do rato sobre a opção "*Create Schematic Symbol*" disponível na lista "*Design Entry Utilities*". Isto permite utilizar o módulo nos níveis superiores da hierarquia do projecto.

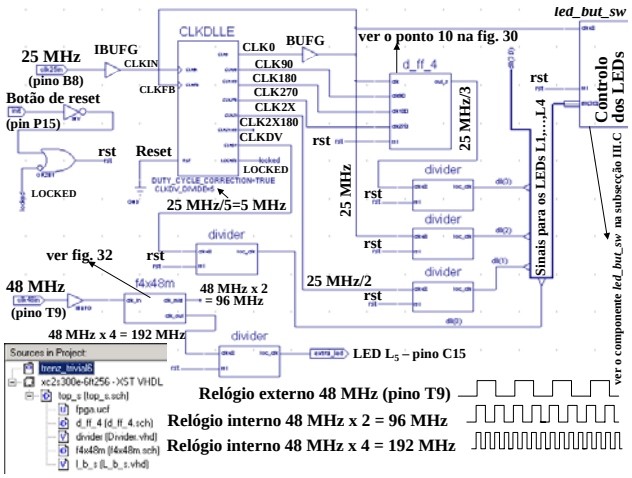

Fig. 31 – Uso de DLL num circuito

IV. EXEMPLOS DE DESENVOLVIMENTO DE CIRCUITOS PARA AS FPGAS DA FAMÍLIA SPARTAN-II

Nesta secção descrevem-se todos os passos necessários para o desenvolvimento e a implementação de circuitos digitais com base em placa RC100 considerada em detalhe na subsecção II.B.

A. Projecto de um conversor de códigos

O primeiro exemplo demonstra o projecto dum circuito que converte os códigos hexadecimais em números a serem visualizados num *display* de 8 segmentos. Para desenvolver o circuito é necessário realizar os passos considerados na secção anterior. Primeiro, deve-se criar um projecto e especificar a família *Spartan 2*, o dispositivo *xc2s200*, o pacote *fg456* e o desempenho *5*. A seguir, inclui-se uma nova fonte no projecto (ver o ponto 1 na fig. 33).

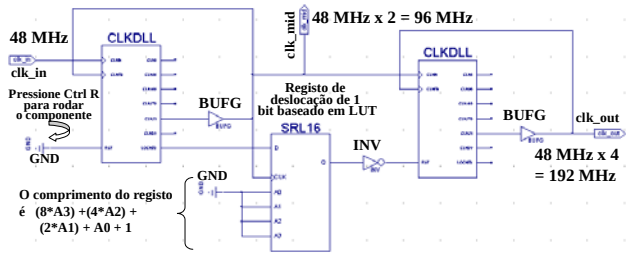

Fig. 32 – A multiplicação do relógio (componente *f4x48m*)

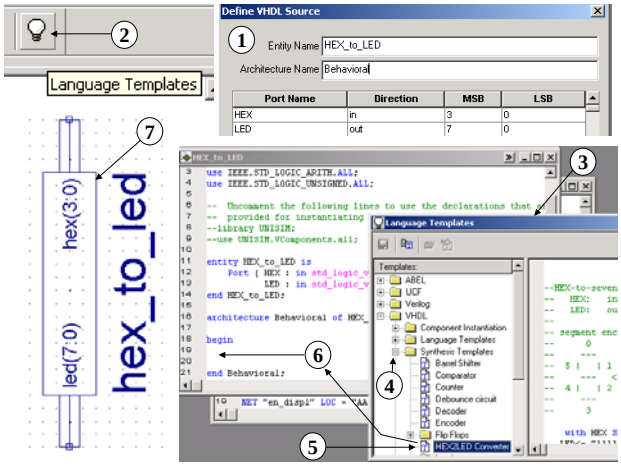

Fig. 33 – Projecto dum conversor de códigos

Existem vários modelos (*templates*) em Xilinx os quais podemos aproveitar incluíndo-os em ficheiros VHDL ou UCF. Estes modelos representam as construções mais utilizadas da linguagem, pedaços de código VHDL para circuitos diferentes e exemplos típicos de ficheiros de restrições do utilizador. Portanto, o código para o conversor pode ser copiado do modelo de síntese respectivo (ver os pontos 2-6 na fig. 33). No ponto 6 o nome do modelo move-se com o rato para a janela com o ficheiro VHDL. O código resultante deve ser modificado ligeiramente para que todos os 8 segmentos sejam utilizados (em vez dos 7 segmentos existentes no modelo). Para além disso, os valores activos devem ser '1's (em vez dos '0's especificados no modelo). O código final é o seguinte:

```
entity HEX_to_LED is
   Port ( HEX : in std_logic_vector(3 downto 0);
   LED : out std_logic_vector(7 downto 0));
end HEX_to_LED;

architecture Behavioral of HEX_to_LED is
begin
-- conversão de HEX para LED
   with HEX SELect
      LED<= "00000110" when "0001",    --1
      "01011011" when "0010",    --2
      "01001111" when "0011",    --3
      "01100110" when "0100",    --4
      "01101101" when "0101",    --5
      "01111101" when "0110",    --6
      "00000111" when "0111",    --7
      "01111111" when "1000",    --8
      "01101111" when "1001",    --9
      "01110111" when "1010",    --A
      "01111100" when "1011",    --B
      "00111001" when "1100",    --C
      "01011110" when "1101",    --D
      "01111001" when "1110",    --E
      "01110001" when "1111",    --F
      "00111111" when others;    --0
end Behavioral;
```

O símbolo criado, que pode ser utilizado para o projecto hierárquico (ver o final da secção III), está apresentado no ponto 7 da fig. 33.

B. Interacção com o rato

Para projectar um circuito que recebe dados dum rato ligado à placa RC100 será utilizado o conversor descrito na subsecção IV.A. A interface com o rato é organizada via duas linhas (ver os pinos A10 – relógio e E11 – dados na fig. 3). O rato envia os dados para a placa em série. Se mover o rato ou carregar num dos seus botões, serão enviados para a placa três pacotes através da linha de dados (pino E11). Cada pacote inclui um bit inicial (que está sempre a '0'), 8 bits de dados, 1 bit de paridade e 1 bit final (que está sempre a '1'). Cada bit de dados só é válido quando o sinal do relógio proveniente do rato (pino A10) muda de '1' para '0' (transição descendente). Sendo assim, o rato envia 33 bits e para cada 11 bits é necessário fazer o seguinte: esperar pela transição descendente do relógio; esperar até que o bit inicial seja igual a '0'; receber 8 bits de dados; receber o bit de paridade e verificar se não ocorreu nenhum erro; esperar até que o bit final seja igual a '1'. Os dois últimos bytes de dados (pacotes 2 e 3) contêm o número de pulsos detectados nas direcções X (movimento horizontal) e Y (movimento vertical) desde que os 33 bits precedentes foram enviados. Os bits do primeiro byte possuem o significado seguinte (contando do bit menos significativo até ao mais significativo): o estado do botão esquerdo do rato, o estado do botão direito ('0' – inactivo, '1' – pressionado para ambos os casos), '0', '1', a direcção X, a direcção Y, *overflow* na direcção X, *overflow* na direcção Y. Os detalhes podem ser encontrados em [9-12]. A seguir está apresentado o código VHDL simplificado para receber os dados do rato (o código completo encontra-se em [39]).

```
entity get_mouse_data is
   Port ( m_clk : in std_logic;
   m_data : in std_logic; -- dados do rato
   --ver subsecção IV.A
   HEX_L : out std_logic_vector(3 downto 0);
   HEX_R : out std_logic_vector(3 downto 0);
   l_m_LED : out std_logic;
   -- para o LED esquerdo da RC100
   r_m_LED : out std_logic);
   -- para o LED direito da RC100
end get_mouse_data;

architecture Behavioral of get_mouse_data is
   signal m_data_s : std_logic; -- dados na
   -- transição descendente do relógio
begin

process (m_clk) -- relógio proveniente do rato
begin
   if m_clk'event and m_clk='0' then -- transição
   -- descendente do sinal de relógio
      m_data_s<=m_data;
```

```
      -- guardar os dados do rato
   end if;
end process;

process (m_clk)
   variable count33 : integer range 0 to 33 := 0;
   -- contador
begin
   if m_clk'event and m_clk='1' then
      count33 := count33+1;
      if count33 = 1 then
         if m_data_s = '0' then null;
         -- esperar pelo bit inicial
         else  count33 := 0;
         end if;
      -- mostrar os estados dos botões nos LEDs
      -- pinos U20,W11 na fig.3
      elsif count33 = 2 then
         r_m_LED <= m_data_s;
      elsif count33 = 3 then
         l_m_LED <= m_data_s;
      elsif count33 = 11 then
         if m_data_s = '1' then null;
         -- esperar pelo bit final
         else count33 := 10;
         end if;
      elsif count33 = 12 then
         if m_data_s = '0' then null;
         -- esperar pelo bit inicial
         else count33 := 11;
         end if;
      -- visualizar os dados no display esquerdo
      --(ver conversor na secção IV.A)
      elsif count33 = 13 then
         HEX_L(0) <= m_data_s;
      elsif count33 = 14 then
         HEX_L(1) <= m_data_s;
      elsif count33 = 15 then
         HEX_L(2) <= m_data_s;
      elsif count33 = 16 then
         HEX_L(3) <= m_data_s;
      elsif count33 = 22 then
         if m_data_s = '1' then null;
         -- esperar pelo bit final
         else  count33 := 21;
         end if;
      elsif count33 = 23 then
         if m_data_s = '0' then null;
         -- esperar pelo bit inicial
         else  count33 := 22;
         end if;
      -- visualizar os dados no display direito
      elsif count33 = 24 then
         HEX_R(0) <= m_data_s;
      elsif count33 = 25 then
         HEX_R(1) <= m_data_s;
      elsif count33 = 26 then
         HEX_R(2) <= m_data_s;
      elsif count33 = 27 then
         HEX_R(3) <= m_data_s;
      elsif count33 = 33 then
         if m_data_s = '1' then count33 := 0;
         -- esperar pelo bit final
         else   count33 := 32;
         end if;
      end if;
   end if;
end process;

end Behavioral;
```

O ficheiro de restrições do utilizador tem o código seguinte:

```
NET "clk_m" LOC = "A10";      # rato
NET "mouse_data" LOC = "E11";
NET "l_led" LOC = "U20";      # LEDs individuais
NET "r_led" LOC = "W11";
NET "data_l<0>" LOC = "V13";  # display esquerdo
NET "data_l<1>" LOC = "AB14";
NET "data_l<2>" LOC = "W14";
NET "data_l<3>" LOC = "V17";
NET "data_l<4>" LOC = "Y18";
NET "data_l<5>" LOC = "W18";
NET "data_l<6>" LOC = "AA20";
NET "data_l<7>" LOC = "AA14";
NET "en_disp1" LOC = "AA22";  # sinais de enable
NET "en_disp2" LOC = "V20";
NET "data_r<0>" LOC = "W13";  # display direito
NET "data_r<1>" LOC = "AB13";
NET "data_r<2>" LOC = "AA13";
NET "data_r<3>" LOC = "AA18";
NET "data_r<4>" LOC = "W17";
NET "data_r<5>" LOC = "AA19";
NET "data_r<6>" LOC = "AB20";
NET "data_r<7>" LOC = "Y13";
```

Agora é necessário construir o ficheiro *.bit respectivo que poderá ser carregado para a FPGA com a ajuda de ferramenta *File Transfer Utility*. Para tal primeiro selecciona-se a placa (RC100) e a seguir escreve-se o nome do ficheiro de *bitstream*.

C. Interacção com a RAM estática

Este exemplo demonstra como se pode ler e escrever dados para a RAM estática [21] disponível na placa RC100. A fig. 34 contém os diagramas temporais [21] correspondentes a estas operações e representa todas as ligações entre a RAM estática (banco 0) e a FPGA (ver também a fig. 4 que inclui as ligações para ambos os bancos 0 e 1).

Uma particularidade a realçar é que a leitura/escrita dos dados deve ser deslocada um ciclo de relógio relativamente aos endereços respectivos. Por exemplo, os dados que aparecerem no ciclo de relógio corrente serão escritos para o endereço que foi indicado no ciclo de relógio anterior, enquanto o endereço corrente será

utilizado no ciclo de relógio seguinte. Todos os detalhes podem ser encontrados em [21].

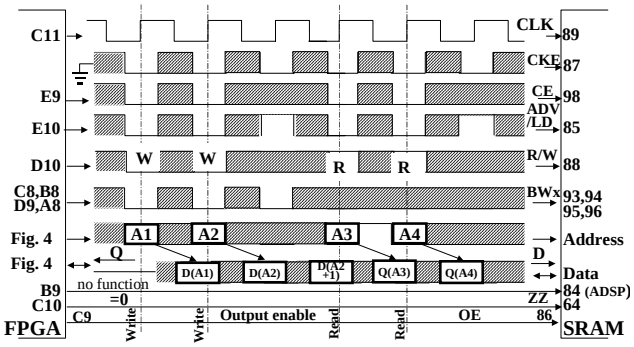

Fig. 34 – Os diagramas temporais e as ligações para a RAM estática (banco 0); os sinais tracejados correspondem a *don't care*

A fig. 35 contém o esquema do nível superior (*top_ram*) do circuito respectivo.

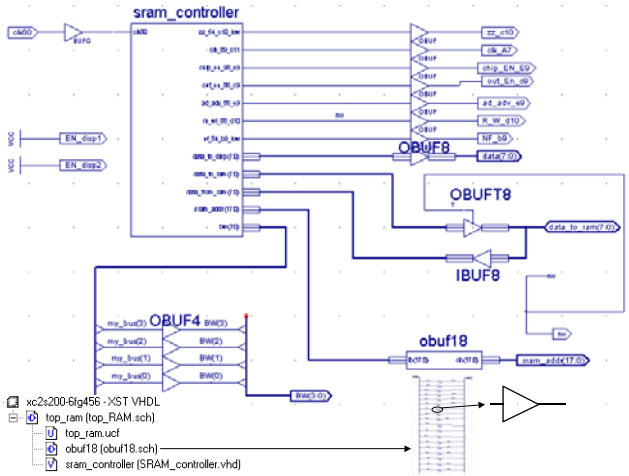

Fig. 35 – Esquema do circuito que efectua leitura/escrita para a RAM estática da RC100

O bloco principal do circuito da fig. 35 é o controlador *sram_controller* que pode ser descrito em VHDL da seguinte maneira:

```
entity SRAM_controller is
   Port ( data_to_RAM :out std_logic_vector(7
   downto 0);
   data_from_RAM : in std_logic_vector(7 downto
   0);
   data_to_disp : out std_logic_vector(7 downto
   0);
   clk80 : in std_logic;
   zz_64_C10_low : out std_logic;
   BW : out std_logic_vector(3 downto 0);
   clk_89_C11 : inout std_logic;
   Chip_En_98_E9 : out std_logic;
   Out_En_86_C9 : out std_logic;
   Ad_Adv_85_E9 : out std_logic;
   Rh_Wl_88_D10 : inout std_logic;
   NF_84_B9_low : out std_logic;
   SRAM_addr : out std_logic_vector(17 downto 0)
   );
end SRAM_controller;

architecture Behavioral of SRAM_controller is
   signal state : std_logic_vector(4 downto 0);
   signal div : unsigned (21 downto 0);
   signal loc_clk : std_logic;
begin

process(clk80)
-- relógio interno de frequência baixa
begin
   if rising_edge(clk80) then  div<=div + 1;
   end if;
end process;

loc_clk <=  not div(div'left);

process(loc_clk) -- sequência dos estados
begin
   if rising_edge(loc_clk) then
      state <= state + 1;
   end if;
end process;

process(loc_clk)
-- leitura/escrita da RAM estática
begin
   if rising_edge(loc_clk) then
      case state is
      -----------------ESCRITA------------
         when "00000" => Chip_En_98_E9<= '1';
            clk_89_C11 <= '0';
         when "00001" => Chip_En_98_E9<= '0';
            NF_84_B9_low<= '0';
            zz_64_C10_low<= '0';
            Ad_Adv_85_E9<= '0';
            Rh_Wl_88_D10<= '0';
            Out_En_86_C9<= '0';
            BW<= "1110";
            SRAM_addr<= "000000000111111111";
         when "00010" => clk_89_C11<= '1';
         when "00011" => clk_89_C11<= '0';
            SRAM_addr<= "000000000111111110";
         when  "00100" =>
            data_to_RAM<= "00111111"; -- "0"
         when  "00101" => clk_89_C11<= '1';
         when  "00110" => clk_89_C11<= '0';
            SRAM_addr <= "000000000111111101";
         when  "00111" =>
            data_to_RAM<= "00000110"; -- "1"
         when "01000" => clk_89_C11<= '1';
         when "01001" => clk_89_C11<= '0';
            SRAM_addr<= "000000000111111100";
         when "01010" =>
            data_to_RAM<= "11011011";  -- "2"
         when "01011" => clk_89_C11<= '1';
         when "01100" => clk_89_C11<= '0';
            SRAM_addr<= "000000000111111011";
         when "01101" =>
```

```
            data_to_RAM<= "11001111"; -- "3"
         when "01110" => clk_89_C11<= '1';
         when "01111" => clk_89_C11<= '0';
         -----------------LEITURA------------
         when "10000" => Rh_Wl_88_D10<= '1';
            SRAM_addr<= "0000000000111111111";
         when "10001" => clk_89_C11<= '1';
         when "10010" => clk_89_C11<= '0';
            SRAM_addr<= "0000000000111111110";
         when"10011" => clk_89_C11<= '1';
            data_to_disp <= data_from_RAM; -- "0"
         when "10100" => clk_89_C11<= '0';
            SRAM_addr<= "0000000000111111101";
         when "10101" => clk_89_C11<= '1';
            data_to_disp <= data_from_RAM; -- "1"
         when "10110" => clk_89_C11<= '0';
            SRAM_addr<= "0000000000111111100";
         when "10111" => clk_89_C11<= '1';
            data_to_disp <= data_from_RAM; -- "2"
         when "11000" => clk_89_C11<= '0';
            SRAM_addr<= "0000000000111111011";
         when "11001" => clk_89_C11<= '1';
            data_to_disp <= data_from_RAM; -- "3"
         when "11010" => clk_89_C11 <= '0';
         when others => Chip_En_98_E9<= '1';
      end case;
   end if;
end process;

end Behavioral;
```

O ficheiro de restrições do utilizador é bastante grande, portanto está apresentado em formato duma linha contínua:

```
NET "clk80" TNM_NET = "clk80"; TIMESPEC "TS_clk80" = PERIOD "clk80" 80 MHz HIGH 50 %; NET "ad_adv_e9" LOC = "E10"; NET "bw<0>" LOC = "A8"; NET "bw<1>" LOC = "D9"; NET "bw<2>" LOC = "B8"; NET "bw<3>" LOC = "C8"; NET "chip_en_e9" LOC = "E9"; NET "clk_a7" LOC = "A7"; NET "clk80" LOC = "A11"; NET "data<0>" LOC = "W13"; NET "data<1>" LOC = "AB13"; NET "data<2>" LOC = "AA13"; NET "data<3>" LOC = "AA18"; NET "data<4>" LOC = "W17"; NET "data<5>" LOC = "AA19"; NET "data<6>" LOC = "AB20"; NET "data<7>" LOC = "Y13"; NET "data_to_ram<0>" LOC = "H4"; NET "data_to_ram<1>" LOC = "G1"; NET "data_to_ram<2>" LOC = "H3"; NET "data_to_ram<3>" LOC = "H2"; NET "data_to_ram<4>" LOC = "J4"; NET "data_to_ram<5>" LOC = "H1"; NET "data_to_ram<6>" LOC = "J5"; NET "data_to_ram<7>" LOC = "J2"; NET "en_disp1" LOC = "AA22"; NET "en_disp2" LOC = "V20"; NET "nf_b9" LOC = "B9"; NET "out_en_c9" LOC = "C9"; NET "rw" LOC = "AA20"; NET "r_w_d10" LOC = "D10"; NET "sram_addr<0>" LOC = "C5"; NET "sram_addr<1>" LOC = "D5"; NET "sram_addr<2>" LOC = "B3"; NET "sram_addr<3>" LOC = "A3"; NET "sram_addr<4>" LOC = "B4"; NET "sram_addr<5>" LOC = "E6"; NET "sram_addr<6>" LOC = "A4"; NET "sram_addr<7>" LOC = "E7"; NET "sram_addr<8>" LOC = "B5"; NET "sram_addr<9>" LOC = "C6"; NET "sram_addr<10>" LOC = "D6"; NET "sram_addr<11>" LOC = "A5"; NET "sram_addr<12>" LOC = "B6"; NET "sram_addr<13>" LOC = "D7"; NET "sram_addr<14>" LOC = "C7"; NET "sram_addr<15>" LOC = "D8"; NET "sram_addr<16>" LOC = "E8"; NET "sram_addr<17>" LOC = "A9"; NET "zz_c10" LOC = "c10";
```

O exemplo considerado escreve sequencialmente na RAM estática os códigos de quatro dígitos (0, 1, 2 e 3) que correspondem à sua representação num *display* de 8 segmentos. A escrita é efectuada para os endereços 000000000111111111, 0…01…110, 0…01…101 e 0…01…100 respectivamente. Posteriormente estes valores serão lidos e visualizados no *display* direito (ver fig. 3). O bit que distingue a leitura da escrita é mostrado no segmento central do *display* esquerdo.

Toda a estrutura o projecto está apresentada no canto inferior esquerdo da fig. 35.

D. Exemplo de projecto baseado em EDIF

A placa RC100 foi especialmente orientada às aplicações desenvolvidas em Handel-C. Handel-C é uma linguagem de especificação a nível de sistemas desenvolvida com base em standard C. Um dos projectos finais de curso do ano lectivo corrente é dedicado à primeira experiência nesta área. Uma aplicação potencial baseada em Handel-C incluíndo a implementação do circuito respectivo em hardware, é descrita em [44]. Handel-C permite que as especificações do sistema baseadas em C sejam transformadas no código VHDL sintetizável ou sejam sintetizadas no circuito do sistema e representadas no formato intermédio EDIF. O código seguinte representa uma especificação trivial em Handel-C:

```
set clock = external "A11"; //pino do relógio
set part = "2S200FG456-5"; //tipo da FPGA
rom unsigned HexDisplayEncode[16]={0x3f,0x6,0x5B,
0x4F,0x66,0x6D,0x7D,0x7,0x7F,0x6F,0x77,0x7C,0x39,
0x5E,0x79,0x71};

void main(void)    // programa principal
{ unsigned 7 d;  // inteiro positivo de 7 bits
  unsigned 32 Delay;   // atraso de 32 bits
  unsigned 4 i; // inteiro positivo de 4 bits
  interface bus_out() en(unsigned 2 enable = 0x3)
  with {data={"AA22","V20"}}; //activa os
                              //displays
  interface bus_out() seg(unsigned 7 dis = d)
  //ver os pinos na fig. 3
  with {data={"AB20", "AA19", "W17", "AA18",
  "AA13", "AB13", "W13"}};  // a variável d está
  // relacionada com os pinos do display
  interface bus_in(unsigned 1 To) in_clock()
  //ver os pinos na fig. 3
  with { data={"A11"}}; //o pino do relógio é A11
  while (1)// ciclo infinito
  {   for(i=0; i<10; i++) // atraso de ~1 segundo
      {   d = HexDisplayEncode[i];
          Delay = 80000000;
          // frequência da RC100 é de 80 MHz
          do { Delay--; } while(Delay!=0); }
          // o ciclo de atraso
  }                          }
```

Esta especificação foi transformada num ficheiro EDIF no ambiente DK1 da Celoxica. Todos os passos

subsequentes para ISE 5 estão apresentados na fig. 36 (ver os pontos 1-9). Primeiro, é necessário criar um projecto novo do tipo EDIF (ver o ponto 1). A seguir, o ficheiro EDIF produzido com as ferramentas da Celoxica deve ser adicionado ao projecto (ver os pontos 2-3). Finalmente, pode ser gerado e carregado para a FPGA o ficheiro de *bitstream*. O ponto 9 na fig. 36 mostra os resultados. Os dígitos do *display* direito da RC100 serão alternados de 0 a 9 estando cada dígito iluminado durante aproximadamente 1 segundo.

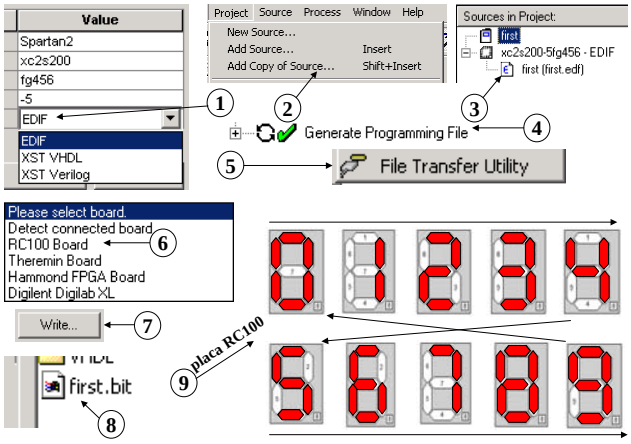

Fig. 36 – A implementação dum circuito a partir da sua especificação em Handel-C

O ambiente DK1 permite também construir módulos VHDL sintetizáveis. Para aproveitar estes módulos em ISE é necessário criar uma biblioteca VHDL denominada *HandelC* e incluir nela os ficheiros *HandelC.vhd* e *xilroc.vhd* fornecidos pela Celoxica. A sequência respectiva dos passos a serem realizados em ISE 5 é a seguinte: *Project -> New Source -> VHDL Library -> HandelC -> Next -> Finish*. Agora pode-se seleccionar *Library View* na janela *Sources in Project* (i.e. janela superior esquerda) e adicionar os ficheiros necessários à biblioteca *HandelC* utilizando os passos seguines: *Select HandelC library -> Project -> Add Source -> directório \HandelC*. Em seguida, pode-se voltar a *Module View* na janela superior esquerda.

V. CONCLUSÕES

Este artigo foi preparado na forma dum manual que descreve as ferramentas de software e hardware recentes utilizadas no processo educativo no Departamento de Electrónica e Telecomunicações da Universidade de Aveiro. Os tópicos abordados incluem as FPGAs das famílias Spartan-II/Spartan-IIE, placas de protótipo e ambientes de desenvolvimento assistido por computador. Os conteúdos das secções II-IV permitem perceber os passos principais de projecto de sistemas baseados em FPGA e esclarecem as características e possibilidades de software ISE 5 da Xilinx. Os exemplos apresentados podem ser implementados e testados em hardware.

Infelizmente, o tamanho limitado do artigo impede considerar todos os detalhes desejados. A fim de compensar esta desvantagem, foram preparados materiais adicionais que estão disponíveis na WebCT [39]. Estes são apresentados na forma dum capítulo que possui o mesmo nome que o artigo. O capítulo está dividido em secções cujos nomes correspondem aos nomes das secções do artigo. As secções incluem todas as figuras (em formato *PowerPoint*), apresentações animadas, projectos de ISE 5 e outros materiais úteis. A cada secção associa-se um ficheiro *ReadMe* destinado a guiar o processo de consulta. Este trabalho foi feito para o curso especial de Microelectrónica (no âmbito do programa *"Advanced Microelectronics Engineering"*) que foi parcialmente leccionado pela Universidade de Aveiro. É de notar que a secção "Referências" disponível na WebCT contém *links* para a maioria das fontes adicionais listadas no fim deste artigo. Caso, depois do artigo tiver sido publicado, ocorram algumas modificações ou correcções, estas serão descritas na secção especial intitulada "História de revisões".

Note-se também que em [1] são apresentadas muitas referências úteis. A informação sobre as bibliotecas da Xilinx pode ser encontrada em [45]. Os documentos [46] contêm todos os dados sobre as ferramentas da Xilinx para ISE 5. Vários projectos que descrevem diferentes tipos de interfaces com FPGA incluíndo os códigos VHDL, estão disponíveis em [17]. A informação adicional sobre as placas de protótipo encontra-se em [47].

AGRADECIMENTOS

Os autores agradecem ao Professor José Luis Oliveira pela ajuda prestada na elaboração deste artigo.

REFERÊNCIAS

[1] V.Sklyarov, "Síntese e Implementação de Circuitos Digitais Reconfiguráveis Dinamicamente", E&T, Vol. 3, Nº 8, Jan. 2003.
[2] SystemC: http://www.systemc.org/
[3] HandelC: http://www.celoxica.com/
[4] Handel-C Language Reference Manual, Celoxica, 2002
[5] DK1 Design Suite User Manual, Celoxica, 2002.
[6] RC100: User Manual, Function Library Manual, Tutorial manual, Hardware manual. Celoxica, 2001.
[7] SystemC™ Version 2.0 User's Guide, 2002.
[8] Spartan-IIE 1.8V FPGA Family, 2002: http://www.xilinx.com/
[9] PS/2 protocols: http://www.networktechinc. com/ps2-prots.html
[10] http://www.howstuffworks.com/ mouse3.html
[11] Mouse information: http://www.hut.fi/~then/mytexts/mouse. html
[12] PS/2 Mouse/Keyboard Protocol: http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/PS2/ps2.htm
[13] Keyboard interfacing: http://www.howell1964.freeserve.co.uk/logic/burched/b5_kbd.htm
[14] The AT-PS/2 Keyboard Interface: http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/ keyboard/atkeyboard.html
[15] VGA Signal Generation with the XS Board: CD of XESS corporation, 2002: http://www.xess.com/
[16] Synaptics TouchPad Interfacing Guide. Synaptics, Inc., 1998.

[17] XSV Board Projects: http://www.itee.uq.edu.au/ #peter/xsvboard/
[18] Interfacing the Extended Capabilities Parallel Port: http://www.beyondlogic.org/ecp.htm
[19] Xilinx CoolRunner® XPLA3 CPLD. http://www.xilinx.com/
[20] Intel StrataFlash Datasheet, Intel, Inc.: http://developer.intel.com/ design/flcomp/datashts /290667.htm
[21] Micron ZBT SRAM: http://www.micronsemi.com Part no. MT55L256L36F.
[22] Bt121 Monolithic CMOS Triple 8-bit VIDEODAC. Rockwell Semiconductor System. L121001, Ref. G.
[23] SAA7111A Enhanced Video Input Processor. Philips Semiconductors, 1998.
[24] Spartan-IIe Development Platform: www.trenz-electronic.de
[25] Liquid crystal display of type PC-160203. Polytronix, Inc 2001.
[26] HD44780U Dot Matrix Liquid Crystal Display Controller/Driver. Hitachi, 1998.
[27] CPLD XC9572XL High Performance CPLD, Xilinx, 2000: www.xilinx.com.
[28] VGA Signal Generation with the XS Board. XESS CD.
[29] Application demo example including VHDL code for interfacing with VGA monitor provided by Trenz Electronic available on Trenz CD.
[30] Am29LV800B 8 Megabit (1 M x 8-Bit/512 K x 16-Bit) Flash Memory. AMD, Publication# 21490, 2000.
[31] PDIUSBP11A Universal Serial Bus Transceiver. Product data. Philips Semiconductors, 2001.
[32] Full-Speed USB 1.1 Function Controller. Application: SoC with Xilinx Virtex. Trenz electronic.
[33] IDT71V416S SMOS Static RAM 4 Meg (256K x 16-Bit). Integrated Device Technology, Inc, 2000.
[34] XSA Board V1.1, V1.2 User Manual, 2002: http://www.xess.com/
[35] XESS CD, 2002.
[36] ISE Quick Start Tutorial, Xilinx, 2002: www.xilinc.com.
[37] ISE 5 In-Depth Tutorial, Xilinx, 2002: www.xilinc.com.
[38] Development System Reference Guide, Xilinx, 2002.
[39] Página *http://webct.ua.pt*, "2º Semestre", a disciplina “Computação Reconfigurável”.
[40] http://www.model.com/support/documentation.asp
[41] StateCAD User’s Guide, Xilinx, 2002.
[42] Using the Virtex Delay-Locked Loop. XAPP 132, 2002: www.xilinx.com.
[43] ftp:/ftp.xilinx.com/pub/applications/xapp/xapp132.zip
[44] P.Almeida, M.Almeida, “Desenvolvimento de um circuito aritmético a partir da sua especificação em Handel-C”, E&T, Vol. 3, Nº 8, Jan. 2003.
[45] Libraries guide, Xilinx, 2002.
[46] http://www.xilinx.com/support/sw_manuals/xilinx5/download/
[47] http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=protoboards_protoboards_page