Desenvolvimento de um circuito em Handel-C para experiências com máquinas de estados finitos

Joel Arrais

***Resumo* - Este artigo descreve um circuito baseado numa FPGA (*Field Programmable Gate Array*), que permite a especificação e simulação de uma máquina de estados finitos de *Moore*. A interacção com o circuito é garantida através da interface gráfica (monitor VGA), do rato e do teclado. A especificação foi feita em Handel-C, que é uma linguagem de alto nível, desenvolvida pela *Celoxica*. Esta especificação em Handel-C foi verificada no ambiente *Celoxica* DK1 *Design Suite*, e compilada para um ficheiro EDIF, que por sua vez, foi convertido num "*bitstream*" no ambiente *Xilinx* ISE 5.2. O circuito especificado foi testado na placa RC100 da *Celoxica*, que tem como componente reconfigurável principal a FPGA Spartan-II XC2S200.**

***Abstract* – This paper describes an FPGA-based circuit, which permits to specify and to simulate finite state machines (Moore model). Interaction with the circuit is provided through a graphical interface designed for a VGA monitor with the relevant support for a mouse and a keyboard. Functionality of the circuit was described in Handel-C, which is a system-level specification language developed by *Celoxica*. The Handel-C description was verified in *Celoxica* DK1 environment and compiled to an EDIF file. The latter was used as an input for ISE 5.2 software, which generated a bitstream for an FPGA. The designed circuit was tested in the RC100 prototyping board of *Celoxica* that contains an FPGA XC2S200 of Spartan-II family of *Xilinx*.**

I . INTRODUÇÃO

O trabalho, aqui descrito, foi o projecto final da disciplina Computação Reconfigurável (CR), que forneceu as bases necessárias de linguagens de descrição de hardware, tanto de baixo como de alto nível: VHDL e Handel-C [1], respectivamente.

A. Características principais da placa RC100

A placa RC100 [2] tem como componente reconfigurável principal a FPGA da família *Spartan-II* [3] da *Xilinx*, com 200.000 portas de sistema.

Esta FPGA tem ligação directa com :

- Oscilador de cristal de 80 MHz ;
- Dois blocos de memória RAM síncrona estática (SSRAM) com 256K palavras de 36bits cada;
- *Flash* RAM de tamanho 64 Mbits;
- Vídeo DAC (VGA 24-Bits);
- Descodificador da entrada de vídeo;
- Porta PS/2;
- LEDs;
- 2 *displays* de 8 segmentos;
- Um barramento de expansão.

É ainda possível aceder à porta paralela através do CPLD.

B. Especificação do projecto

O objectivo deste projecto é o de, utilizando a linguagem Handel-C, desenvolver um circuito que permita a especificação e simulação de uma máquina de estados finitos (FSM – *Finite State Machine*) de Moore [4]. O utilizador interage com a aplicação, utilizando uma interface gráfica (monitor VGA), um rato e um teclado.

A aplicação possui uma janela que pode estar num dos dois seguintes estados: especificação ou simulação. A comutação entre estes é realizada através da opção colocada no canto superior direito da janela, que pode ser vista nas figs. 1a e 1b.

Na fig. 1a está representada a janela, quando se está no estado de simulação. Esta possui as seguintes interacções:

- STATE e OUT representam o estado actual e o respectivo valor de saída;
- IN representa o valor da entrada para a transição para o próximo estado. Este valor é introduzido utilizando o teclado sendo apenas aceites valores binários;
- Quando seleccionadas, as opções colocadas no canto inferior direito, permitem respectivamente avançar para o próximo estado e voltar ao estado inicial;
- As opções do rectângulo branco, situado no canto superior direito, permitem respectivamente criar uma nova FSM, abrir uma FSM que esteja guardada na memória *flash* e guardar a actual FSM na memória *flash*. Como a memória *flash* é um meio de armazenamento não volátil, a FSM pode ser reutilizada;
- É possível obter o valor de STATE, de OUT ou de IN no *display* de 8 segmentos, bastando para tal seleccionar, com o rato, respectivamente a opção desejada.

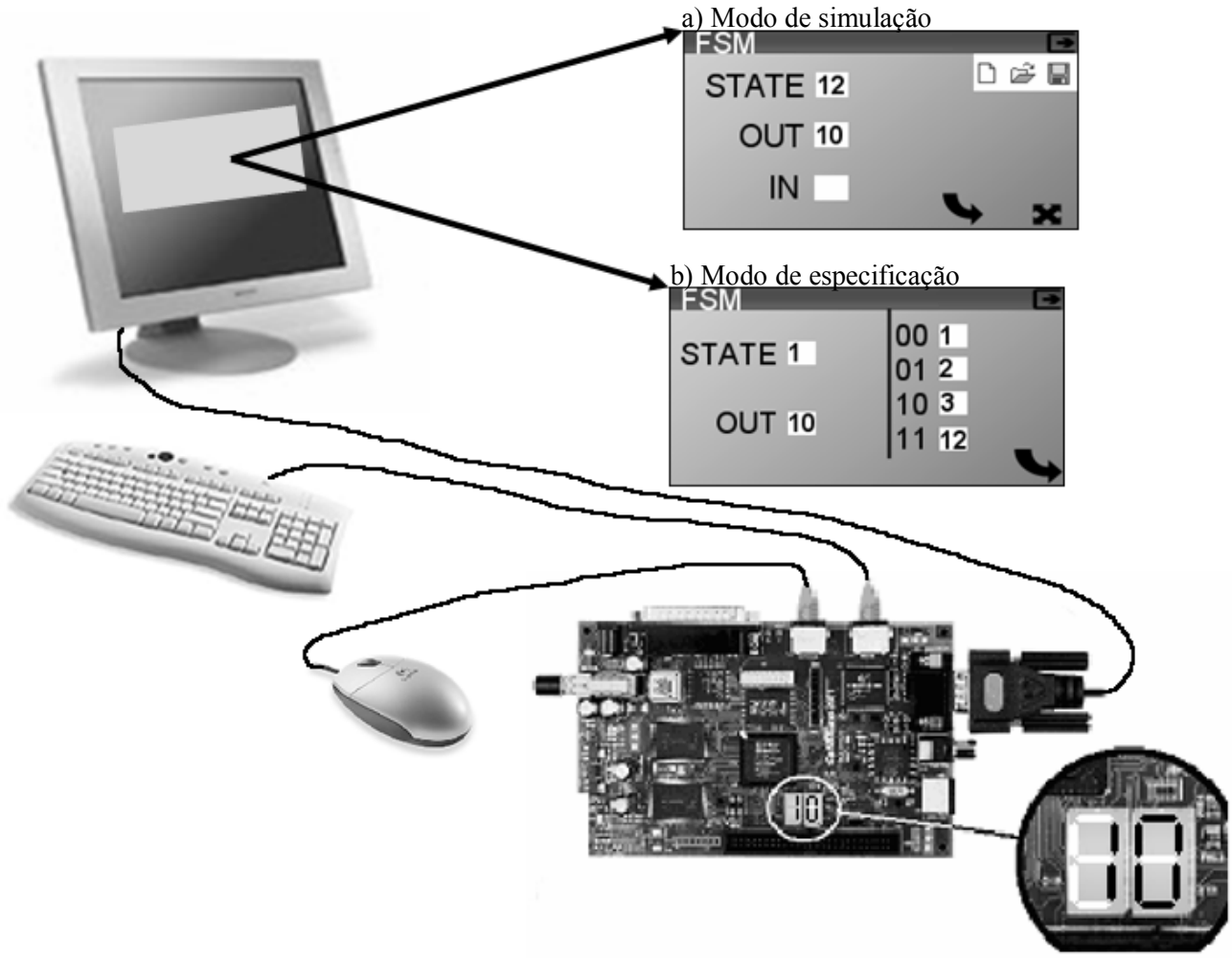

Fig. 1 – Esquema de ligação dos periféricos

Na fig. 1b está representada a janela, quando se está no estado de especificação. Esta permite introduzir todas as transições de saída para um determinado estado, possuindo as seguintes interacções:

● STATE e OUT representam o estado actual e o respectivo valor de saída. Estes valores são introduzidos utilizando o teclado sendo posteriormente validados;

● Utilizando o teclado também são especificados os próximos estados para todos os valores de entrada possíveis;

● A opção colocada no canto inferior direito permite validar e guardar a informação introduzida.

É, também possível especificar uma máquina de estados finitos, no computador, convertê-la para um formato adequado e transferi-la para a placa. Para realizar esta conversão, foi desenvolvida uma aplicação, em *Microsoft Visual Studio C++*. A fig. 2 mostra os passos necessários para a criação de uma máquina de estados no PC. No ponto 1, são especificados, num ficheiro, os estados e as transições entre estes. Nos pontos 2 e 3, é indicado, respectivamente, o nome do ficheiro de entrada e do de saída. No ponto 4, é realizada a conversão da informação para uma forma mais adequada, a ser escrita na memória *flash,* e, no ponto 5, é mostrado parte do ficheiro que contém essa informação.

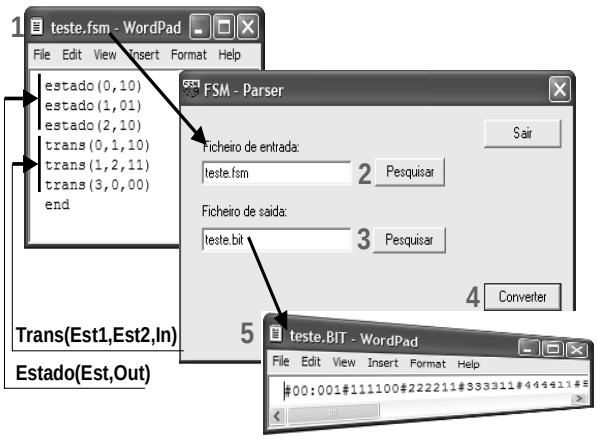

Fig. 2 – Especificação da máquina de estados no PC

II . ESTRUTURA DO PROGRAMA EM HANDEL-C

Ao contrário dos processadores convencionais, em que, em cada instante de tempo, apenas uma instrução pode ser executada, numa FPGA, várias operações podem ser executadas em paralelo. Deste modo, e com o objectivo de optimizar o desempenho, nesta aplicação, todos os blocos principais são executados em paralelo, tal como a fig. 3 ilustra.

O bloco "Movimento da janela e selecções do rato" é responsável por desencadear todos os eventos que as acções do rato podem provocar, sejam elas o movimento das janelas, ou a selecção sobre determinadas áreas.

O bloco "Leitura e validação de dados do teclado" tem como função receber dados do teclado, validá-los e armazená-los.

O bloco "Controlo do *display* VGA" é responsável por, para cada *pixel* do monitor, determinar qual é o correspondente valor do sinal a enviar, na forma RGB (8 *bits Red*, 8 *bits Green*, 8 *bits Blue*).

O bloco "Actualiza a *display* VGA RAM" tem como função actualizar o valor das variáveis presentes no *display* VGA. Este bloco só é, verdadeiramente, executado, se algum dos valores tiver sido alterado.

O bloco "Controlo dos *displays* de oito segmentos" tem como função enviar o valor de uma das seguintes variáveis, para os *displays*: STATE, OUT ou IN.

O bloco "Protecção de ecrã" detecta longos períodos de não utilização da aplicação, e activa a protecção de ecrã.

O bloco "Sincronização" é responsável por determinar a sequência de execução dos diferentes blocos, de modo a evitar problemas de funcionamento, resultante de interferências.

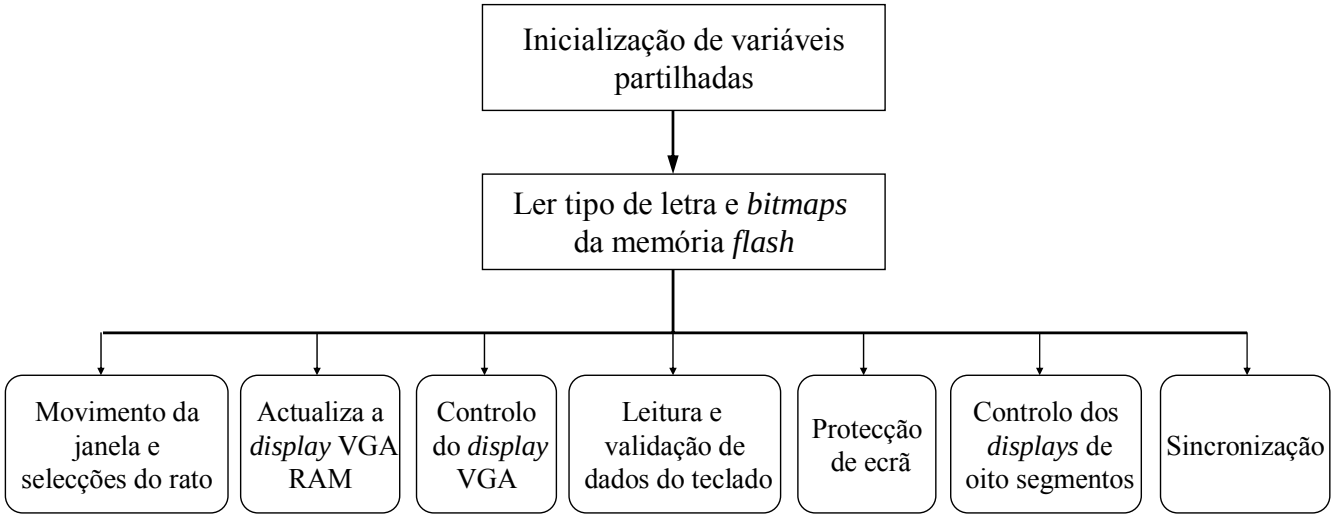

Fig. 3 – Principais blocos do programa

III . CONTROLO DOS DISPOSITIVOS

A. Controlo do display VGA

A janela tem dimensões de 256*128 *pixels* e está mapeada em RAM, pelo que, qualquer alteração, à janela, terá que ser realizada na RAM. Deste modo, o bloco que controla o *display* VGA ([5], pp. 28-34), apenas, tem que ler, da RAM ([5], pp. 14-15), a informação correspondente ao *pixel* que pretende, tal como o seguinte código ilustra:

```
par
{
 RC100ReadSSRAM1(Y<-10 @ X<-8, Pixel);
 VideoPtr->Output = Pixel;
}
```

A utilização deste método evita que se tenha que determinar, para cada ciclo de relógio, o valor de saída do *display* VGA.

B. Construção e utilização do tipo de letra

Foi utilizado um tipo de letra que se encontra mapeado num ficheiro do tipo "*.raw*". Este ficheiro foi copiado, para a memória *flash*, e, em tempo de execução, o programa copia-o, para a RAM, onde vai aceder, cada vez que pretender escrever um caracter. As letras encontram-se sobrepostas, verticalmente, e cada caracter corresponde a uma matriz binária, com 16 linhas e 8 colunas, em que o valor '1' corresponde à letra, e o '0' corresponde ao fundo. A fig. 4 dá um exemplo desta representação.

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 **1** 0 0 0 0
0 0 **1** **1** **1** 0 0 0
0 **1** **1** 0 **1** **1** 0 0
1 **1** 0 0 0 **1** **1** 0
1 **1** 0 0 0 **1** **1** 0
1 **1** **1** **1** **1** **1** **1** 0
1 **1** 0 0 0 **1** **1** 0
1 **1** 0 0 0 **1** **1** 0
1 **1** 0 0 0 **1** **1** 0
1 **1** 0 0 0 **1** **1** 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Fig. 4 – Representação do caracter 'A' na fonte utilizada

Com o objectivo de diminuir acessos à RAM, é lida uma linha (8 *bits*) da letra, por acesso. No exemplo seguinte, o `ILetra` é o código ASCII [6] da letra pretendida, o `ILinha` a linha pretendida da letra e `Data` é a variável de saída, que vai guardar os 8 bits correspondentes.

```
RC100ReadSSRAM1(0 @ (ILetra<-8) @ (ILinha<-4)
                                        ,Data);
```

Após a linha ter sido lida, ela é escrita outra vez, na RAM, mas, desta vez, na zona pretendida da janela. Com o objectivo de facilitar este processo, foi criada a função `WriteChar`, que escreve o caracter correspondente ao código ASCII [6] `AsciiCode`, na posição `x,y` da janela, em que `x,y` são as coordenadas, em *pixels*, relativas à janela.

```
void WriteChar(unsigned 10 x,unsigned 10 y
                         ,unsigned 36 AsciiCode)
```

C. Movimento da janela

Existem duas variáveis (`Jx` e `Jy`), que armazenam a posição, do *pixel* do canto superior esquerdo da janela. Cada vez que o botão esquerdo do rato é pressionado, sobre a barra superior, o seguinte código é executado:

```
par
{
 difx = px-Jx;
 dify = py-Jy;
}
 do
 {
  par
  {
   if (difx < px) Jx = px-difx; else Jx = 1;
   if (dify < py) Jy = py-dify; else Jy = 1;
  }
}while (bl);
```

Do código anterior, `px` e `py` possuem o valor em *pixels* da posição actual do rato, e `bl` é uma *flag*, que se encontra activa sempre que o botão esquerdo do rato está pressionado.

D. Controlo das acções do rato

Na janela, os botões de selecção correspondem a certas áreas. O facto do rato estar sobre, ou de ser pressionado um dos botões deste pode desencadear determinados eventos. A área correspondente aos eventos está mapeada, como o seguinte de código ilustra:

```
rom unsigned 36 Area1 [1] =
{//SetArea(XInicial,YInicial,XFinal,YFinal)
```

```
  // Botão OK
  SetArea(222, 103, 254, 126),

  // Botão MUDAR DE JANELA
  SetArea(177   ,103,227 ,117),
  …
}with { block = 1 };
```

O seguinte código é versão simplificada do bloco que trata dos eventos desencadeados pelas acções do rato:

```
do
{/*Verifica se a actual posição do
  *rato corresponde a algum botão*/

  if InArea [ Area1 [Index] ]
  {
    if(index==0)/*Acção para botão OK*/
    if(index==1)/*Acção para mudar de janela*/
  }
  else
  {
    Index++;
  }
}while (1)
```

E. Leitura e validação de valores do teclado

A leitura de valores do teclado faz-se da seguinte forma ([5], pp. 16-27):

```
DataPort ? Input;
```

`DataPort` é um canal de saída de dados (neste caso do teclado), e `Input` é uma variável, que vai conter o valor do código ASCII [6] da tecla pressionada. Tendo em conta o contexto do programa, o valor de `Input` é validado e armazenado.

F. Processamento dos dados

Tanto no modo de simulação, como no de especificação, os dados de entrada são processados, quando, respectivamente, é seleccionado o botão de "próximo estado", ou o de "validar".

No primeiro caso, é determinado o próximo estado, tendo em consideração o estado actual e o valor de entrada. Se o valor de entrada for incompleto, nenhuma acção é realizada.

No segundo caso, os dados introduzidos são armazenados e substituem os anteriores, referentes ao estado em questão. De salientar que a informação apenas é armazenada, após a validação de todos os campos.

G. Sincronismo entre os diferentes processos

Como existem recursos da placa que só podem ser acedidos uma vez por ciclo de relógio, e blocos que partilham estes mesmos recursos, é necessário um bloco que coordene a atribuição destes recursos.

Um exemplo dum destes recursos é a memória RAM, em que a janela se encontra mapeada, pois esta tanto pode ser acedida pelo bloco que controla o monitor, como por outras funções (`MudarJanela` e `WriteChar`), que escrevem dados nesta. O problema foi resolvido, dando prioridade aos processos que pretendem escrever relativamente aos que lêem, ou seja, através da atribuição de um valor de saída fixo ao *display* VGA, sempre que um outro processo pretenda realizar operações de escrita.

H. Controlo dos displays de oito segmentos da placa

Para facilitar o controlo dos *displays* de segmentos ([4], pp. 56-58), foram criadas 2 ROMs, que têm como função converter o código hexadécimal para código especial, da seguinte forma:

```
rom unsigned HexDisplayEncode1 [16] =
 {0x3f,0x6,0x5B,0x4F,0x66,0x6D,0x7D,0x7,0x7F,0x6
              F,0x77,0x7C,0x39,0x5E,0x79,0x71};
rom unsigned HexDisplayEncode2 [16] =
 {0x3f,0x6,0x5B,0x4F,0x66,0x6D,0x7D,0x7,0x7F,0x6
              F,0x77,0x7C,0x39,0x5E,0x79,0x71};
```

São necessárias duas ROMs, porque existem dois *displays*, que têm de ser controlados, simultaneamente, e cada ROM apenas permite um acesso por ciclo de relógio.

Existe uma *flag* (`Modo7Seg`) que determina qual variável deve ser mostrada no *display*. Tal como referido anteriormente o valor desta *flag* é determinado, através de selecções do rato, podendo estar num dos seguintes estados:

- Desligado;
- Valor de STATE;
- Valor de OUT;
- Valor de IN.

O bloco que controla os *displays* apenas tem que ter em conta o valor da *flag* `Modo7Seg`, e determinar o valor de saída, que o seguinte código ilustra, apenas para um dos *displays* ([7], pp. 3-14).

```
par
{
  RC100Set7Seg1(0 @ DisplayOutput1);

  switch (Modo7Seg)
  {
```

```
   case 0: DisplayOutput1 = 0x00<-7;break;

   case 1: DisplayOutput1 = HexDisplayEncode1
               [(EstActual / 10)<-4]<-7 ;break;

   case 2: DisplayOutput1 = (HexDisplayEncode1
         [(Estados [EstActual].Out [0]==0)?0:1 ]
                                    <-7);break;

   case 3: DisplayOutput1= InMenu0[0]==espaco)?
      0x00:(HexDisplayEncode1 [(InMenu0 [0]==0)?
                                0:1 ]<-7);break;

   default:DisplayOutput1 = 0x00<-7;break;
  }
}
```

I. Configuração da máquina de estados finitos

Como referido, existem duas possibilidades de especificar uma máquina de estados finitos. A primeira é através da sua especificação, em tempo de execução, e a segunda é através de um ficheiro colocado na memória *flash*.

No segundo caso, é necessário especificar os estados, segundo um formato perceptível ao utilizador e, posteriormente, convertê-lo, para um formato mais adequado à memória *flash* da placa. Para realizar esta conversão, foi desenvolvida uma aplicação em *Microsoft Visual Studio C++*.

Após a informação, relativa aos estados, ser copiada, para a memória *flash,* na posição `0x10` um estado pode ser lido, desta, do seguinte modo:

```
do
{
 Data = RC100FlashReadData(Address);
 Address++;

 if (Data != 0x23) return;

 Data = RC100FlashReadData(Address);
 Estados [i].In [0] [0] = (Data-0x30)<-4;
 Address++;

 Data = RC100FlashReadData(Address);
 Estados [i].In [0] [1] = (Data-0x30)<-4;
 Address++;

 Data = RC100FlashReadData(Address);
 Estados [i].In [1] [0] = (Data-0x30)<-4;
 Address++;

 Data = RC100FlashReadData(Address);
 Estados [i].In [1] [1] = (Data-0x30)<-4;
 Address++;

 Data = RC100FlashReadData(Address);
 Estados [i].Out [0] = (Data-0x30)<-1;
 Address++;

 Data = RC100FlashReadData(Address);
 Estados [i].Out [1] = (Data-0x30)<-1;
 Address++;

 i++;
}while (i < 15);
```

A especificação de cada estado é antecedida pelo caracter de controlo '#' (`0x23`), com vista a diminuir as hipóteses de ler uma máquina de estados não válida.

Deste modo a organização da informação, na memória *flash,* é a seguinte:

```
#Next00 Next01 Next10 Next11 Out0 Out1
```

O estado em questão é determinado pela ordem de leitura, e `NextXX` indica o estado para qual transitar, se a entrada for `XX`. `OutX` representa o índice `X` do valor de saída do estado em questão.

J. Execução

O código relativo ao circuito implementado está disponível na WebCT [8]. Para o testar, na placa RC100, basta utilizar a aplicação FTU (*File Transfer Utility*) da *Celoxica* e executar o *script* de configuração: "Load FSM.txt". Após estes passos terem sido realizados, tanto o circuito, como os ficheiros adicionais estão na memória *flash*, e a aplicação é executada sempre que a placa for ligada. De salientar que os DIP *switchers* da placa devem possuir a configuração ilustrada na fig. 5.

Fig. 5 – Posição dos DIP *switchers* para auto arranque

IV. CONCLUSÕES

Este artigo demonstra as potencialidades do Handel-C, como linguagem de especificação de hardware de alto nível. O Handel-C tem a vantagem de permitir a implementação de circuitos numa FPGA, de um modo mais eficiente e simples, em termos de tempo de desenvolvimento, do que seria possível, se fossem utilizadas outras linguagens de baixo nível, como por exemplo o VHDL.

Um outro aspecto importante para o sucesso do nosso trabalho foi a placa RC100, tanto pelos vários periféricos que suporta, como pela facilidade com que estes podem ser utilizados.

Um ponto menos positivo pode ser dado à capacidade da FPGA, que apesar de ser elevada, se torna limitada, quando se pretende tirar o máximo proveito de todas as capacidades da placa.

AGRADECIMENTOS

O autor agradece ao Professor Valery Sklyarov e à Iouliia Skliarova pela ajuda prestada na elaboração deste artigo.

REFERÊNCIAS

[1] "Handel-C Language Reference Manual", Celoxica, 2002

[2] "RC100 Hardware Manual", Celoxica, 2001.

[3] "Spartan™-II Data Sheet", http://www.xilinx.com/partinfo/ds001.htm, em 22 de Julho de 2003.

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

[5] "RC100 Function Library Reference" Celoxica, 2001.

[6] "ASCII Table", http://www.asciitable.com/, em 22 de Julho de 2003.

[7] "RC100 Tutorials", Celoxica 2001.

[8] Página http://webct.ua.pt, "2º Semestre", a disciplina "Computação Reconfigurável" com login alunoreconf2 e password reconf2.