Desenvolvimento de uma calculadora booleana em VHDL

Filipe David Borba Manana

***Resumo* – Este artigo descreve a implementação de uma calculadora booleana numa FPGA da família Spartan-IIE utilizando a linguagem VHDL. Esta calculadora tem como operandos dois vectores de 20 bits cada, e como resultado também um vector de 20 bits. A interface da calculadora consiste num monitor VGA e nos botões da placa que inclui a FPGA.**

***Abstract* – This paper describes the implementation of a boolean calculator on the basis of a Spartan-IIE FPGA with the aid of VHDL language. The calculator operates over two 20-bit boolean vectors and produces a 20-bit result. The user interface consists of a VGA monitor and push buttons available on the employed FPGA based prototyping board.**

I. INTRODUÇÃO

A calculadora aqui descrita foi projecto na cadeira de Computação Reconfigurável [1], leccionada pelos professores Valery Sklyarov e Iouliia Skliarova no segundo semestre do ano lectivo de 2004/2005. A placa utilizada para o projecto foi o modelo TE-XC2Se da Trenz Electronic [2] e que inclui uma FPGA da família Spartan-IIE. Entre outras características, esta placa tem uma saída para um monitor VGA e quatro botões, como mostra a figura 1.

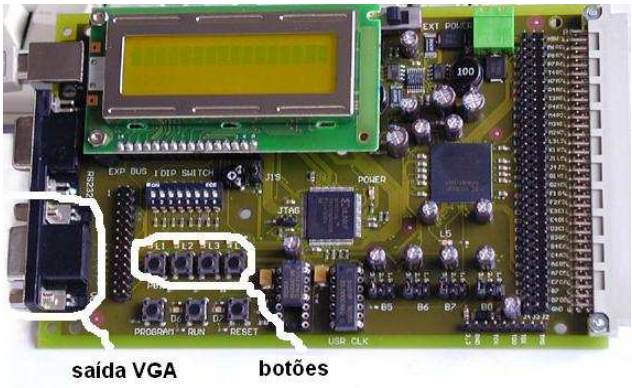

Fig. 1 – placa TE-XC2Se

A interface da calculadora apresentada no monitor VGA (figura 2) tem uma caixa que lista os vários operadores disponíveis, uma caixa para cada operando e uma caixa para o resultado da operação. Cada bit dos operandos pode assumir os valores '0', '1' ou '-' (don't care).

Fig. 2 – interface da calculadora

Os operadores AND, OR e XOR têm como resultado as operações lógicas de "e", "ou" e "ou exclusivo" efectuadas bit a bit nos respectivos operandos. O operador ORT (de ortogonalidade) tem como resultado "00..01" se existem pelo menos dois bits de igual índice com valores '0' e '1' ou '1' e '0', caso contrário o resultado corresponde a "00..00". O operador INT devolve "00..01" se os operandos se intersectam ou "00..00" caso contrário. Os operadores "<" e ">" correspondem aos operadores normais de menor e maior, devolvendo "00...01" ou "00..00" consoante se verifique ou não a respectiva desigualdade.

Os botões da placa (figura 3) têm as seguintes funções:

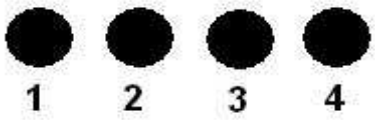

Fig. 3 – botões da placa

1 – seleccionar o bit seguinte de um operando
2 – seleccionar o bit anterior de um operando
3 – mudar o valor do bit seleccionado
4 – mudar de operação

II. ESTRATÉGIA TOMADA

Este trabalho baseou-se no projecto VGA disponibilizado nas aulas práticas de Computação Reconfigurável [1]. Este projecto demonstra a interacção da FPGA com um monitor VGA. A implementação da

calculadora foi efectuada de forma iterativa e de acordo com a seguinte ordem:

1) visualização no monitor VGA de toda informação estática, isto é, caixa e lista de operadores, caixas dos operandos e caixa do resultado;

2) visualização no monitor VGA dos operandos e resultado;

3) representação gráfica de dois cursores: um para assinalar o operador escolhido, e outro para assinalar o bit escolhido de um dos operandos;

4) implementação das operações da calculadora e das funcionalidades dos botões da placa.

III. IMPLEMENTAÇÃO

Para implementação do trabalho foi utilizado o ambiente ISE v6.2 da Xilinx [3].

A – Informação estática

Toda a informação estática é armazenada na Block RAM da própria FPGA. O projecto VGA organiza o monitor VGA em 30 linhas, cada uma com 40 colunas e a cada caracter é associado um valor numérico entre 0 e 90, codificado em 1 byte. São assim necessários 1200 (40x30) bytes de memória para armazenar todos os caracteres a enviar para a saída VGA. Cada bloco de RAM da FPGA contém 4096 bits, ou seja 512 bytes, pelo que foi necessário utilizar 3 blocos. Estes 3 blocos são instanciados em código VHDL e no seu conteúdo foram colocados os valores numéricos dos caracteres correspondentes a toda a informação estática. O primeiro byte do primeiro bloco de RAM corresponde ao caracter da linha 1 e coluna 1, o primeiro byte do segundo bloco de RAM corresponde ao caracter da linha 12 e coluna 33, etc. A memória é endereçada de forma linear por 11 bits (número minímo de bits necessário para codificar valores entre 0 e 1199). Os 9 bits menos significativos do endereço são utilizados para endereçar em simultâneo os 3 blocos, enquanto que os 2 bits mais significativos são utilizados para seleccionar a saída do bloco correspondente ao endereço linear. O seguinte excerto de código mostra como é efectuado este processo:

```
data <= data1 when address(10 downto 9) = "00" else
        data2 when address(10 downto 9) = "01" else
        data3 when address(10 downto 9) = "10" else
        "ZZZZZZZZ"; -- alta impedância
```

Os sinais **data1**, **data2** e **data3** correspondem às saídas do primeiro, segundo e terceiro blocos respectivamente.

De notar que a representação gráfica dos caracteres encontra-se definida num array de matrizes de dimensão 16x16. Cada elemento destas matrizes corresponde a um pixel do respectivo caracter. O array é indexado pelo valor numérico que representa o caracter pretendido. O projecto VGA só continha a definição dos caracteres ‘A’ a ‘Z’, ‘0’ a ‘9’ e espaço, pelo que foi necessário definir os caracteres que são utilizados para desenhar as caixas de texto (‘ ┌’, ‘─’ , ‘ ┐’, ‘ │’, ‘ └’ e ‘ ┘’) e para o valor “don’t care” (‘-‘).

B – Visualização dos operandos e resultado

Para representar os operandos e o resultado foi definido um novo tipo de dados:

```
type mybit is ('0', '1', '-');
type mybit_vector is array(natural range <>) of mybit;
```

Na arquitectura que implementa a interface com o monitor VGA foram definidas constantes que definem a posição inicial de cada caixa de texto, e sinais que representam os operandos e o resultado:

```
-- início da caixa de selecção do operador
constant opbox_start_row : natural := 1;
constant opbox_start_col : natural := 1;

-- início da caixa do primeiro operando
constant op1box_start_row : natural := 5;
constant op1box_start_col : natural := 1;

....

-- tamanho em bits de cada operando e resultado
constant vector_length : positive := 20;

 -- operandos e resultado
signal op1, op2, result : mybit_vector(1 to vector_length);
```

O código VHDL desta arquitectura lê constantemente da memória Block RAM o caracter (byte) correspondente à posição actual no monitor VGA que está a ser varrida. Sempre que a posição actual corresponder à posição de um bit de um dos operandos ou do resultado, então o caracter a enviar para a saída VGA corresponde ao valor desse bit e não ao valor que foi lido da Block RAM.

C – Implementação dos cursores

Para implementar os cursores foram definidos os seguintes sinais na arquitectura do projecto VGA:

```
signal current_op : natural range 0 to 6;
signal highlight : boolean;
signal position : positive range 1 to (2 * vector_length);
```

O sinal **current_op** indica qual é a operação em vigor (0 – and, 1 – or, 2 – xor, etc), **highlight** assume o valor *true* quando a posição no monitor VGA que está a ser percorrida faz parte da região de um dos cursores, e **position** que assume valores entre 1 e 40 serve para indicar qual o operando seleccionado e qual o bit desse operando está seleccionado. Se **position** estiver entre 1 e 20, o bit número **position** do primeiro operando está seleccionado, enquanto que se estiver entre 21 e 40 encontra-se seleccionado o bit número (**position – 20)** do segundo operando. Por exemplo, o pedaço de código que activa o cursor no operador AND quando este está seleccionado é o seguinte:

```
if text_row = opbox_start_row then

case current_op is
  when 0 =>
   if text_col >= (opbox_start_col + 1) and
   text_col <= (opbox_start_col + 3)   then
        highlight <= true;
   else   highlight <= false;
   end if;
  …..
end case;
```

E o pedaço de código que activa o cursor num bit seleccionado do primeiro operando é o seguinte:

```
elsif text_row = op1box_start_row then
   if position >= 1    and
      position <= vector_length   and
      text_col = (op1box_start_col + position)
   then
         highlight <= true;
   else  highlight <= false;
   end if;
```

O valor do sinal **highlight** afecta directamente o valor das componentes do sistema de cores RGB que são enviadas para o monitor VGA, como mostra o seguinte excerto de código colocado na arquitectura do módulo responsável pela interacção com o monitor VGA**:**

```
R  <= '0' when blank = '1' else pixel;
G <= '0' when blank = '1' else
       pixel when highlight = false
       else '0';
B  <= '0' when blank = '1' else
       '1' when highlight = false
       else '0';
```

O sinal **pixel** faz parte da arquitectura mencionada e corresponde a um pixel do caracter que está a ser enviado para a saída VGA num dado momento. A posição na matriz de pixeis do caracter é dada por dois sinais auxiliares, que indexam a matriz que representa o caracter.

D – Implementação das operações e funcionalidades dos botões da placa

Todas as operações da calculadora foram implementadas num pacote próprio. As operações ORT e INT foram implementadas como funções normais, enquanto que para as restantes foi utilizada a redefinição de operadores, uma vez que estes operadores já existem na linguagem VHDL. Por exemplo, o código para a operação “>” é o seguinte:

```
function ">" (signal a, b : in mybit_vector) return
mybit_vector
is
        variable result : mybit_vector(a'range);
begin
        result := (others => '0');

        for i in result'range loop
                if a(i) = '0' and b(i) = '1' then
                        exit;
                elsif a(i) = '1' and b(i) = '0' then
                        result(result'right) := '1';
                        exit;
                elsif a(i) = '-' or b(i) = '-' then
                        result := (others => '-');
                        exit;
                        end if;
                end loop;
return result;
end ">";
```

A invocação das operações é efectuada na arquitectura do módulo que faz interface com o monitor VGA, com a seguinte instrução:

```
result <= op1 and op2 when current_op = 0
          else op1 or op2 when current_op = 1
          else op1 xor op2 when current_op = 2
          else ort(op1,op2) when current_op =3
          else int(op1,op2) when current_op = 4
          else op1 < op2 when current_op = 5
          else op1 > op2 when current_op = 6;
```

O controlo dos botões da placa é efectuado através do CPLD desta. A interacção com o CPLD é efectuada num módulo próprio, sendo este baseado no exemplo de interacção com o CPLD fornecido nas aulas de Computação Reconfigurável [1]. Este módulo tem um sinal de saída de 4 bits que indica o estado dos botões da placa. Este sinal de saída é recebido no sinal **buttons(3 downto 0)** da entidade do módulo que faz interface com o monitor VGA. Quando pressionado o botão 1, é incrementado por uma unidade o valor do sinal **position** se este for menor que 40. De modo simétrico, quando o botão 2 é pressionado é decrementado de uma unidade o valor do sinal **position** se este for maior que 1. A funcionalidade do botão 3 (mudar o valor do bit seleccionado) é implementada pelo seguinte processo:

```
operands_change : process(rst, iclk)
begin
  if rst = '1' then
     op1<=(others=>'0');
     op2<=(others=> '0');
  elsif rising_edge(iclk) then
      if buttons(2) = '1' then
         if position >= 1 and position <= vector_length
         then
            case op1(position) is
              when '0' => op1(position) <= '1';
              when '1' => op1(position) <= '-';
              when '-' => op1(position) <= '0';
            end case;
         elsif position >= (vector_length+1) and
              position <= (2*vector_length)  then
           case op2(position - vector_length) is
             when '0' =>
                      op2(position - vector_length)  <= '1';
             when '1' =>
                       op2(position – vector_length) <= '-';
             when '-' =>
                      op2(position – vector_length)  <= '0';
           end case;
         end if;
      end if;
   end if;
end process operands_change;
```

Para a mudança de operação, sempre que pressionado o botão 4, o valor do sinal **current_op** é incrementado de uma unidade se for menor que 6 ou colocado a zero se for igual a 6.

IV. CONCLUSÕES

O objectivo global deste trabalho consistia na pedagogia dos sistemas digitais reconfiguráveis, nomeadamente as FPGAs e ferramentas para esta finalidade. Outros objectivos inerentes à natureza do trabalho foram a aprendizagem da interacção entre a FPGA e componentes (CPLD, saída VGA) da placa utilizada, tal como o uso do ambiente ISE e da linguagem VHDL para estes fins. Todos estes objectivos foram atingidos na sua plenitude.

AGRADECIMENTOS

Agradeço aos professores Valery Sklyarov e Iouliia Skliarova pela ajuda, orientação e conselhos fornecidos tanto para a elaboração do projecto como para a elaboração deste artigo.

REFERÊNCIAS

[1] http://www.ieeta.pt/~iouliia/courses/CR - website da cadeira de Computação Reconfigurável

[2] http://www.trenz-electronic.com - website da Trenz Electronic

[3] http://www.xilinx.com - website da Xilinx