Desenvolvimento de um projecto em VHDL para ordenação de vectores ternários

Filipe Cunha

Resumo **– Este artigo descreve os passos percorridos no desenvolvimento de um circuito desenvolvido numa FPGA que ordenasse uma matriz ternária 16x16 e que a imprimisse num monitor VGA.**

Ao iniciar o programa, a matriz original irá ser preenchida com 0, 1 e '-' (don't care) e em seguida será ordenada através do algoritmo de ordenação *bubblesort* sendo depois ambas imprimidas no ecrã.

Abstract – This paper describes the steps to be executed for the design of an FPGA-based circuit that sorts ternary vectors of a matrix (with the size 16x16) and visualizes the result on a VGA monitor screen. The initial matrix is arbitrary filled with values 0, 1 and '-' (don't care) and then it is sorted using the *bubblesort* algorithm. Finally the initial and the sorted matrices will be displayed on the monitor screen.

I. INTRODUÇÃO

Este trabalho aqui descrito, foi o projecto de avaliação da cadeira Computação Reconfigurável, leccionada pelo Professor Valery Sklyarov. Na realização deste projecto recorreu-se aos conhecimentos e material [1] fornecidos pelo docente da cadeira. A especificação do projecto foi realizada em VHDL e, para a sua implementação, fez-se uso do ambiente Xilinx ISE 5.2 [2].

A.. Descrição do projecto

O objectivo deste projecto é o de desenvolver um circuito, utilizando a linguagem VHDL, que ordene matrizes de tamanho 16x16 em função do número de elementos (0, 1 ou '-') de cada linha. Por exemplo, se o elemento de ordenação fosse o 0 (zero), depois da matriz ser ordenada, na primeira linha fica a linha que tenha menos zeros. Por outro lado na última linha ficaria a linha que tenha o maior número de zeros. Antes da matriz ser ordenada é preenchida dinamicamente por um algoritmo aleatório, sendo depois copiada para outra matriz de igual tamanho. Esta cópia é que vai ser ordenada dando a possibilidade de no fim da execução do programa tanto a matriz original como a matriz ordenada serem imprimidas no monitor VGA, tal como o elemento de ordenação.

II ORGANIZAÇÃO ESTRUTURAL DO PROJECTO

Na figura 1 podem ser vistos todos os blocos constituintes deste projecto.

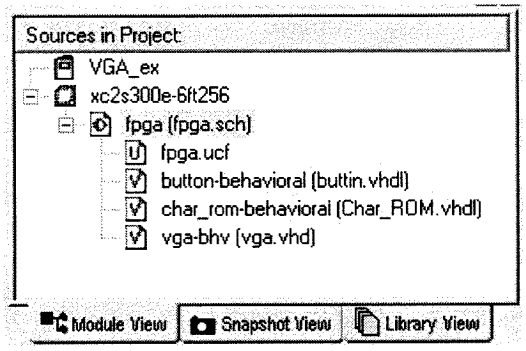

Figura 1 – Blocos constituintes do projecto.

Como se vê na figura acima, o projecto é formado por três blocos em linguagem VHDL e um ficheiro com extensão UCF. UCF significa *User Constraint File* e indica quais os pinos atribuídos e definições do sinal de relógio. Neste projecto usou-se um relógio de 25 MHz

No bloco "button" é controlado o uso dos botões usados (botão que modifica o elemento de ordenação). No bloco "char_rom" está especificada uma ROM que contém os três caracteres 16x16 usados na impressão das matrizes. O bloco "vga" é o bloco principal do projecto. É lá que estão as operações relacionadas com as matrizes e com a impressão destas no monitor VGA.

Na figura 2 seguinte são mostradas as ligações existentes entre os três blocos.

III IMPLEMENTAÇÃO

Para a realização deste projecto foi utilizado o projecto *VGA_ex* disponibilizado pelo Professor Valery Sklyarov. No projecto *VGA_ex* era implementado um contador de 0 a 9 accionado por um botão específico da placa Trenz Spartan IIE [3], contador que era imprimido no monitor VGA.

Ao começar este projecto foi implementada a matriz como um array de *std_logic_vector*. Posteriormente deparei-me com um problema de comparações pois a linguagem VHDL aceitava como verdade a comparação '-'='0' ou '-'='1'. Neste caso sempre que um dos valores da matriz fosse *don't care* esse valor seria contabilizado

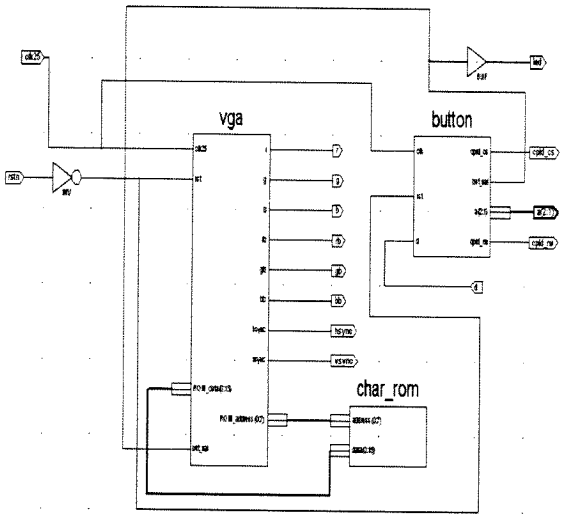

Figura 2 – Ligações entre os blocos

como 0 ou 1 consoante o valor de VAL (valor que vai definir a ordenação da matriz), não permitindo assim a ocorrência de *don't cares*.

Foi alterada então a matriz para um array de array de inteiros de 0 a 2, onde o 2 correspondia ao *don't care*, e assim já não existia o problema de comparações tornando o projecto viável.

Antes de ser construído o código para que a matriz fosse imprimida no monitor VGA, foi decidido implementá-lo de modo a imprimir uma pequena matriz (4x4) no LCD da placa Trenz Spartan IIE, para que testasse o funcionamento do processo de ordenação e da função contar.

Depois de verificado o bom funcionamento do programa começou-se a implementar a parte de impressão da matriz no monitor VGA.

Implementei então um processo de ordenação baseado no algoritmo *bubblesort*, mas com a particularidade de em vez de comparar o valor de um array, compara o número de 0, 1 ou – em cada linha da matriz e troca linhas adjacentes caso seja necessário. O número de 0,1 ou – em cada linha é devolvido pela função **contar**.

```
function contar ( linha : tipo_linha) return
std_logic_vector is
variable contador: std_logic_vector(0 to 4);
 begin
       contador:="00000";
       for i in 0 to N-1 loop
         if linha(i)=VAL then
           contador:=contador+1;
         end if;
       end loop;
 return contador;
 end contar;
```

O processo de ordenação começa com a verificação se o sinal de reset (rst) foi activado ou se está no início da execução do programa. Se sim então as variáveis necessárias ao funcionamento do projecto são inicializadas.

```
if rst= '1' then
       -- inicialização de valores
       matrix_completa:=false;
       i:=0;
            TROCAS:=false;
            FIM:=false;
            VAL<=0;
              col:=0;
              row:=0;
```

Caso contrário começaria o preenchimento da matriz.

Para que a matriz original não seja sempre a mesma (não seja estática mas sim dinâmica) foi criado um algoritmo pseudo-aleatório (com recurso a um signal std_logic_vector que era incrementado em cada ciclo de relógio e uma operação XOR) que preenche a matriz.

```
elsif rising_edge(clk25) then
   -- incremento do vector que vai ser usado
   -- no algoritmo de inserção de valores na
matriz
   if vector="11111111" then
     vector:="00000000";
     else vector:=vector+1;
   end if;

   -- preenchimento da matriz pelo algoritmo
aleatório
   if matrix_completa=false then
        temporario:=((vector(7) & vector(0) &
vector(3)) xor (vector(1) & vector(6) &
vector(4)));
        if temporario="000" or temporario="010"
or temporario="101" then MATRIX(row)(col)<=0;
MATRIX_ORD(row)(col)<=0;
        elsif temporario="001" or
temporario="100" or temporario="110" then
MATRIX(row)(col)<=1; MATRIX_ORD(row)(col)<=1;
        else MATRIX(row)(col)<=2;
MATRIX_ORD(row)(col)<=2;
        end if;

        if((row=N-1) and (col=N-1)) then
          -- activação do sinal para que se possa
iniciar a ordenação
          matrix_completa:=true;
        elsif(col=N-1) then
          col:=0;
          row:=row+1;
        else col:=col+1;
        end if;
```

Quando a matriz estiver completa a variável matriz_completa passa a ter o valor true o que permite o início da ordenação da matriz.

Como disse anteriormente implementei um processo de ordenação baseado no algoritmo *bubblesort,* com as necessárias alterações:

```
-- ordenação da matriz
else
if FIM=false then
  if
contar(MATRIX_ORD(i))>contar(MATRIX_ORD(i+1))
then
       -- troca de linhas
            MATRIX_ORD(i)<=MATRIX_ORD(i+1);
            MATRIX_ORD(i+1)<=MATRIX_ORD(i);
            TROCAS:=true;
        end if;

        i:=i+1;
        -- se não houve trocas no fim de um
ciclo de iterações
        -- a matriz está ordenada, caso
contrário volta à posição inicial
        if i=N-1 then
            if TROCAS=false then
                FIM:=true;
            else
                TROCAS:=false;
                i:=0;
            end if;
        end if;
      end if;
end if;
```

De notar que na linguagem VHDL (ao contrário das outras linguagens, como C, C++, entre outras) não é necessária uma variável *"temp"* para que a troca de linhas se processe.

Caso fosse necessário testar a ordenação da matriz com os 3 elementos (0, 1 e -) não seria necessário alterar no código e voltar a compilar o projecto, pois foi criado um sinal que muda dinamicamente quando se carrega num determinado botão da placa Trenz. Ao carregar neste botão o sinal VAL é alterado para um dos elementos de ordenação da matriz na seguinte ordem: 0, 1, -.

```
-- alteração do valor de VAL
elsif but_val='1' then
        if matrix_completa=true then
            if VAL=2 then VAL<=0;
                else VAL<=VAL+1;
                end if;
                FIM:=false;
                i:=0;
                TROCAS:=false;
                MATRIX_ORD<=MATRIX;
            end if;
```

De cada vez que o valor da ordenação é alterado e para que o processo de ordenação decorra normalmente é necessário inicializar algumas variáveis é também é preciso reescrever na matriz que vai ser ordenada com os valores da matriz original para que não se percam valores.

Se for necessário testar novas matrizes apenas é necessário carregar no botão de reset (rst) e então todas as variáveis são inicializadas e a matriz é preenchida então por esse algoritmo aleatório sendo sempre diferente.

Foi desenvolvido outro processo que imprime no monitor VGA. No canto superior esquerdo é imprimida a matriz original, nas coordenadas (1,17) é imprimido o valor em que a matriz é ordenada (0,1 ou -) e no canto superior direito é imprimida a matriz ordenada consoante o valor de ordenação.

```
--impressão da MATRIX (matrix original)
if (text_row<N and text_col<N) then
  if
MATRIX(to_integer(text_row))(to_integer(text_col)
)=1 then
      ROM_address      <=      "0001"      &
std_logic_vector(char_row);
      pixel                                <=
ROM_data(conv_integer(std_logic_vector(char_col))
);
  elsif
MATRIX(to_integer(text_row))(to_integer(text_col)
)=0 then
      ROM_address      <=      "0000"      &
std_logic_vector(char_row);
      pixel<=ROM_data(conv_integer(std_logic_ve
ctor(char_col)));
  else
      ROM_address      <=      "0010"      &
std_logic_vector(char_row);
      pixel<=ROM_data(conv_integer(std_logic_ve
ctor(char_col)));
  end if;
```

O código indicado acima é o código necessário para a impressão da matriz original no canto esquerdo do monitor VGA.

Para se imprimir um carácter no monitor é preciso "desenhá-lo" primeiro, isto é, como o monitor VGA só apresenta pixéis é necessário transformar o carácter num conjunto de pixéis através de uma matriz (neste caso também de tamanho 16x16). O bloco Char_ROM recebe um endereço e devolve os dados correspondentes a esse endereço. No código indicado em baixo, mostra-se a maneira de imprimir um 0 (zero) da maneira acima indicada:

```
my_ROM(0)  <= "0111111111100000";
my_ROM(1)  <= "1111111111110000";
my_ROM(2)  <= "1100000000110000";
my_ROM(3)  <= "1100000000110000";
my_ROM(4)  <= "1100000000110000";
my_ROM(5)  <= "1100000000110000";
my_ROM(6)  <= "1100000000110000";
my_ROM(7)  <= "1100000000110000";
my_ROM(8)  <= "1100000000110000";
my_ROM(9)  <= "1100000000110000";
my_ROM(10) <= "1111111111110000";
my_ROM(11) <= "0111111111100000";
my_ROM(12) <= "0000000000000000";
my_ROM(13) <= "0000000000000000";
my_ROM(14) <= "0000000000000000";
my_ROM(15) <= "0000000000000000";
```

Neste projecto apenas foi necessário "desenhar" 3 caracteres (o 0, o 1 e o '-'), mas o array está definido para se poderem usar até 16 caracteres.

Há também de neste projecto se poder escolher a cor do fundo e dos caracteres. Há seis tipos de cores (R, RB, G, GB, B, BB), e pode-se escolher cores isoladas, ou combinações de cores. Neste projecto escolhi o preto como cor de fundo e verde como cor dos caracteres através do seguinte código:

```
-----------------------------------
-- output signals
blank  <= h_blank or v_blank;

-- fundo preto e letras a verde
r   <= '0' when blank= '1' else '0';
rb  <= '0' when blank= '1' else '0';
g   <= '0' when blank= '1' else pixel;
gb  <= '0' when blank= '1' else pixel;
b   <= '0' when blank= '1' else '0';
bb  <= '0' when blank= '1' else '0';
```

IV EXECUÇÃO DO PROJECTO

Depois de o projecto ser compilado, o ficheiro fpga.bit é criado. Através do programa TEProg.exe este ficheiro é enviado para a FPGA da placa Trenz, podendo aí ser executado.

Como se pode ver na figura 3, naquela execução o elemento de ordenação é o 0 (zero). Podemos ver a matriz original (a da esquerda) e à direita a matriz ordenada, onde ser verifica que a ordenação está correcta, pois a primeira linha corresponde à linha com menor número de zeros (0 zeros) e a última linha corresponde à linha com maior número de zeros (16 zeros).

V CONCLUSÃO

O tamanho da matriz usada neste projecto faz com que sejam ocupadas mais de 100% das slices da FPGA, o que pode provocar situações (embora muito raras) onde existam problemas de sincronismo. Caso a matriz seja mais pequena, por exemplo 6x6 ou 10x10, este problema

```
00-00111-10110-0                1111111111111111
--0110-11----010                ----------------
1-0-000--010-00-        0       --0110-11----010
0-01000-000-1010                -0-1-11-0-10-01-
1-0-01-100-10-10                -000--1-1101-1-0
-000--1-1101-1-0                00-01100---1111-
1111111111111111                1-01001-1111--00
11-00000010110-0                1-0-01-100-10-10
00-01100---1111-                1-11-1-0000-100-
1-01001-1111--00                010-1-00101-1-10
1-11-1-0000-100-                00-00111-10110-0
0000000000000000                1-0-000--010-00-
-0-1-11-0-10-01-                11-00000010110-0
010-1-00101-1-10                -----0000-0000-0
----------------                0-01000-000-1010
+----0000-0000-0                0000000000000000
```

Fig3–Possível resultado da execução do projecto.

não acontece.

Apesar destes problemas, todos os objectivos propostos pelo professor foram cumpridos sendo introduzidos outros objectivos para que a execução do programa fosse mais atractiva: a matriz é preenchida dinamicamente, é ordenada correctamente, são mostradas no ecrã as matrizes original e a ordenada e é possível alteração o valor da ordenação visualizando a matriz ordenada segundo esse valor.

Mesmo assim há sempre lugar para futuros melhoramentos e optimizações, nomeadamente para tentar diminuir o número de slices ocupadas e para diminuir também o tempo de compilação.

Este projecto serviu para melhorar os conhecimentos da cadeira Computação Reconfigurável e interacção com a placa Spartan IIE da Trenz.

AGRADECIMENTOS

Agradeço ao Professor Valery Sklyarov e aos meus colegas de Computação Reconfigurável pela ajuda fornecida.

REFERENCES

[1] V. Sklyarov, material da disciplina Computação Reconfigurável, 2º semestre, WebCT.

[2] http://www.xilinx.com, ISE 5.2, Xilinx series FPGA.

[3] http://www.trenz-electronic.com