

Desenvolvimento de circuitos reconfiguráveis que interagem com um monitor VGA

Iouliia Skliarova

Resumo – Este artigo descreve a implementação de circuitos reconfiguráveis que suportam interacção com monitores VGA. A visualização dos resultados e dos dados intermédios num monitor VGA facilita a depuração de circuitos realizados em FPGA e aumenta as potencialidades de interacção com o utilizador. O projecto desenvolvido foi utilizado no âmbito das disciplinas Computação Reconfigurável (4º ano de LECT) e Sistemas Digitais Reconfiguráveis (opção de 5º ano, LEET) e serviu de base para projectos individuais de alunos realizados no segundo semestre do ano lectivo 2004/2005.

Abstract – This paper describes implementation of reconfigurable circuits which support interaction with VGA monitors. Visualization of the results and intermediate data on a VGA monitor reduces the FPGA-based circuit design time and expands the potentialities of human-circuit interaction. The designed project was successfully employed within Reconfigurable Computing (4th year of Computer Engineering curriculum) and Reconfigurable Digital Systems (5th year, Electrical Engineering curriculum) disciplines and served as a base for individual student projects realized in the second semester of 2004/2005 academic year.

I. INTRODUÇÃO

Computação Reconfigurável (CR) é uma disciplina que é dada no 2º semestre do 4º ano aos alunos da Licenciatura em Engenharia de Computadores e Telemática (LECT) desde 2003. Os objectivos principais desta disciplina consistem em introduzir a tecnologia de desenvolvimento de sistemas digitais reprogramáveis estática e dinamicamente com base em FPGA dotando os alunos com conhecimentos necessários para poderem descrever, simular, sintetizar e implementar circuitos reconfiguráveis de complexidade média. Em particular, são abordados tópicos tais como arquitectura de CPLDs e FPGAs modernas, linguagens de descrição de hardware (com o exemplo de VHDL sintetizável), ferramentas de projecto assistido por computador (editores de VHDL, editores esquemáticos e bibliotecas de elementos, editores gráficos de máquinas de estados finitos, geradores de núcleos de propriedade intelectual parametrizáveis, simuladores, ferramentas de síntese e de implementação), protocolos de comunicação mais comuns, etc.

Para estimular o interesse dos alunos e aprofundar os seus conhecimentos são-lhes sugeridos projectos

individuais que são disponibilizados em meados do semestre e devem ser concluídos até ao final da época de exames. Estes projectos constituem um método alternativo de avaliação que substitui o exame teórico tradicional. Só os melhores alunos podem se candidatar aos projectos sendo estes seleccionados com um mini-teste voluntário realizado passadas 3-4 semanas lectivas. No ano académico 2004/2005 37 alunos estiveram inscritos na disciplina de CR, 27 alunos participaram no mini-teste, 11 alunos obtiveram a classificação necessária mínima e 9 alunos optaram por fazer projectos.

Todos os projectos dados foram concluídos com sucesso até a data limite estabelecida. Os autores dos melhores 6 projectos prepararam artigos para a revista do departamento (incluídos neste volume) [1-6] que descrevem o trabalho desenvolvido, dificuldades enfrentadas e experiência adquirida. A maioria dos alunos avaliou este método de avaliação como “bastante interessante” e “enriquecedor”.

No ano lectivo 2004/2005 todos os projectos sugeridos aos alunos consistiam em descrever em VHDL e implementar na placa TE-XC2Se [7] (utilizada nas aulas práticas) um circuito digital capaz de visualizar resultados num monitor VGA ligado à placa. A especificação em VHDL da interface com um monitor VGA constitui um dos tópicos leccionados em CR. Este artigo é um sumário da informação fornecida aos alunos e descreve implementação de um simples controlador VGA para a placa TE-XC2Se [7]. É com base neste controlador, disponibilizado parcialmente aos alunos, que eles desenvolveram os seus projectos. A implementação da interface com monitor VGA é também leccionada no âmbito da disciplina Sistemas Digitais Reconfiguráveis que é uma opção de 5º ano da Licenciatura em Engenharia Electrónica e Telecomunicações, os alunos da qual podem igualmente aproveitar informação recolhida aqui. Para auxiliar o processo de aprendizagem existe também uma série de *tutorials* [8,9] que explicam a utilização de construções sintetizáveis de VHDL, ilustram os princípios de interacção com ferramentas de projecto assistido por computador e mostram como implementar vários controladores em FPGA.

O resto deste artigo está organizado de maneira seguinte. Na secção II são descritos os princípios de funcionamento de um monitor VGA. A secção III ilustra como o controlador pode ser especificado em VHDL e implementado na placa TE-XC2Se. A secção IV demonstra detalhadamente o desenvolvimento dum

simples controlador VGA que funciona no modo de texto. Finalmente, as conclusões estão na secção V.

II. PRINCÍPIOS DE FUNCIONAMENTO DE UM MONITOR VGA

A. Sinais de cor

Um monitor VGA recebe a informação de cor com a ajuda de 3 sinais: R (vermelho), G (verde) e B (azul). Cada um destes sinais controla a emissão de electrões que por sua vez iluminam um ponto na superfície do ecrã com uma cor primária. Os níveis analógicos destes sinais especificam a intensidade de cada uma das cores primárias podendo variar entre 0 V (completamente escuro) e 0.7 V (brilho máximo).

Na placa TE-XC2Se [7] as saídas de cor analógicas são controladas por um simples DAC (*digital to analog converter*) que possui 6 entradas digitais (ilustradas na Fig. 1). Sendo assim, torna-se possível especificar 4 níveis de cada cor (de 00 a 11) e representar na totalidade 64 cores diferentes (4^3).

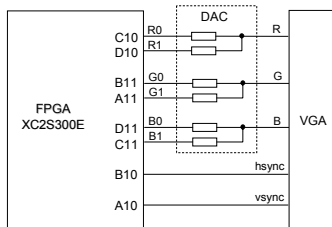


Fig. 1 – Ligação dos pinos da FPGA XC2S300E que fornecem a informação de cor e de sincronização com a saída VGA da placa TE-XC2Se

B. Características temporais de sinais VGA

Uma imagem no ecrã do monitor é composta por m linhas cada uma das quais contém n pixels. Para o nosso controlador usamos a resolução de $n \times m = 640 \times 480$ pixels. Para desenhar uma imagem o feixe de electrões percorre o ecrã de esquerda para a direita e de cima para baixo conforme ilustrado na Fig. 2. As linhas tracejadas representam os traços verticais e horizontais que acontecem quando o feixe de electrões atinge o final de uma linha (ou a última linha da imagem).

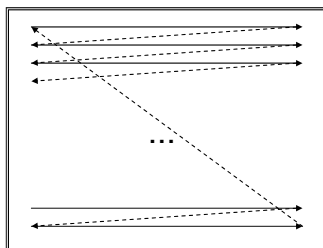


Fig. 2 – O feixe de electrões percorre o ecrã de esquerda para a direita e de cima para baixo

Os circuitos que controlam o movimento adequado de electrões precisam de dois sinais de sincronização ($hsync$ e $vsync$ na Fig. 1) que marcam o início e fim de cada linha. As características temporais dos sinais de sincronização são detalhadas na Fig. 3 [10].

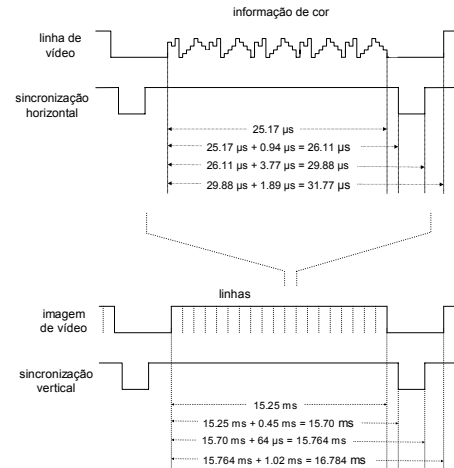


Fig. 3 – Características temporais dos sinais de sincronização

O sinal de sincronização horizontal ($hsync$) marca o início e fim de cada linha com pulsos curtos negativos e garante que o monitor visualiza os pixels entre as extremidades esquerda e direita da área visível do ecrã. Os pixels são enviados para o monitor (com os sinais R, G e B) durante os 25.17 μs . A seguir há um intervalo de 0.94 μs que medeia o fim de recepção da informação de cor e o início do pulso negativo do sinal $hsync$ que por sua vez tem a duração de 3.77 μs . Finalmente o intervalo de 1.89 μs medeia o fim do pulso negativo do $hsync$ e o início de recepção da informação de cor para a linha seguinte. Sendo assim, uma linha de pixels ocupa 25.17 μs dos 31.77 μs ($25.17 + 0.94 + 3.77 + 1.89$), nos restantes 6.6 μs ($31.77 - 25.17$) os sinais de cores são anulados (*blanked*).

De modo semelhante, os pulsos negativos no sinal de sincronização vertical ($vsync$) marcam o início e fim de cada imagem (composta por m linhas) e asseguram que o monitor visualize as linhas entre o topo e a base da área visível do ecrã. As linhas são visualizadas durante os 15.25 ms ($31.77 \mu s \times 480$ linhas). A seguir, há um intervalo de 0.45 ms que medeia o fim de visualização das linhas e o início do pulso negativo do sinal $vsync$ que por sua vez dura 64 μs e é separado por 1.02 ms do início da imagem seguinte. Sendo assim, uma imagem de pixels ocupa 15.25 ms do intervalo de 16.784 ms (15.25 ms + 0.45 ms + $64 \mu s + 1.02$ ms), nos restantes 1.534 ms os sinais de cor são anulados (*blanked*).

C. Geração de sinais VGA

Tendo descrito todos os sinais que permitem controlar um monitor VGA a partir da placa TE-XC2Se, é possível especificar o algoritmo que visualize no ecrã uma imagem. O pseudocódigo deste algoritmo está ilustrado na Fig. 4. O pseudocódigo inclui dois ciclos principais o

primeiro dos quais visualiza L linhas de pixels visíveis e o segundo insere $BL-L$ linhas nulas e o pulso de sincronização vertical. O primeiro ciclo por sua vez inclui dois subciclos: um que envia P pixels de cada linha de vídeo para o monitor e outro que insere $BP-P$ pixels nulos e um pulso de sincronização horizontal.

```

while (1)
{
  //envia L linhas de vídeo para o monitor
  for line 1 to L
  {
    //envia P pixels visíveis para cada linha
    for pixel 1 to P
    {
      //lê o valor do pixel da memória de vídeo
      pixel_value = VRAM(line, pixel)
      //converte o valor lido em informação de cor
      (R,G,B) = COLORMAP(pixel_value)
    }

    //anula saídas R, G, B para BP-P pixels
    for pixel P to BP
      (R,G,B) = 0

    // pulso de sincronização horizontal
    if ( HB ≤ pixel < HE)
      hsync = 0
    else hsync = 1
  }

  //anula saídas R, G, B para BL-L linhas
  for line L to BL
  {
    (R,G,B) = 0

    // pulso de sincronização vertical
    if ( VB ≤ line < VE)
      vsync = 0
    else vsync = 1
  }
}

```

Fig. 4 – Pseudocódigo do algoritmo para controlar um monitor VGA

III. IMPLEMENTAÇÃO DO CONTROLADOR VGA PARA A PLACA TE-XC2SE

Para o projecto foi usado o gerador de relógio de 25 MHz disponível na placa TE-XC2Se [7]. Este relógio estabelece a frequência máxima com a qual pixels podem ser enviados para o monitor. Dado que o intervalo para a transmissão de pixels visíveis em cada linha é de $25.17 \mu\text{s}$ (ver Fig. 3), então se emitirmos uma nova actualização de vídeo para as saídas R, G, B a cada ciclo de relógio, no máximo poderemos visualizar $25.17 \mu\text{s} \times 25 \text{ MHz} = 629$ pixels (valor do parâmetro P em Fig. 4) em cada linha. De modo semelhante é possível calcular os valores dos parâmetros restantes do algoritmo da Fig. 4. Assim, são necessárias $6.6 \mu\text{s} \times 25 \text{ MHz} = 165$ pixels nulos ($BP-P$), o pulso negativo de sincronização horizontal deve começar contados $26.11 \mu\text{s} \times 25 \text{ MHz} = 653$ pixels (HB) do início da transmissão de uma linha, e deverá ser desactivado contados $29.88 \mu\text{s} \times 25 \text{ MHz} = 747$ pixels (HE).

Dado que a visualização de uma imagem no ecrã demora 16.784 ms e cada linha de pixels juntamente com a sincronização horizontal ocupa $31.77 \mu\text{s}$, pode-se concluir que são enviadas 528 linhas para o monitor (das quais $L = 480$ são linhas de pixels visíveis e $BL-L = 48$ são linhas nulas necessárias para o retraço vertical). O pulso negativo de sincronização vertical deve começar contados

$15.7 \text{ ms} \times 25 \text{ MHz} / BP = 494$ linhas (VB) do início da transmissão da imagem, e deverá ser desactivado contados $15.764 \text{ ms} \times 25 \text{ MHz} / BP = 496$ linhas (VE).

A imagem no ecrã é actualizada com a frequência de $1000 / 16.784 \text{ ms} = 59.6 \approx 60 \text{ Hz}$.

Os processos de sincronização podem ser descritos trivialmente em VHDL com a ajuda de dois contadores que vão determinar que linha (*line*) e que pixel (*pixel*) dentro desta linha está a ser varrido em cada momento conforme mostra o seguinte excerto de código:

```

signal pixel, line : natural;
signal h_blank, v_blank, blank: std_logic;

--constantes para sincronização horizontal
constant P: natural:= 628; -- blanking start
constant HB: natural:= 652; -- h-sync start
constant HE: natural:= 746; -- h-sync end
constant BP: natural:= 793; -- blanking end

--constantes para sincronização vertical
constant L: natural:= 479; -- v blanking start
constant VB: natural:= 493; -- v-sync start
constant VE: natural:= 495; -- v-sync end
constant BL: natural:= 527; -- v blanking end

horizontal_sync: process(clk25, rst)
begin
  if rst = '1' then -- reset activo
    pixel <= 0;
  elsif rising_edge(clk25) then
    if pixel = BP then
      pixel <= 0;
    else
      pixel <= pixel + 1;
    end if;
  end if;
end process horizontal_sync;

horiz_blank: h_blank <= '1' when pixel > P else
  '0'; -- blanking

horiz_sync:
hsync <= '0' when (pixel >= HB) and (pixel < HE)
  else '1'; --pulso de sincron. horizontal

vertical_sync: process(hsync, rst)
begin
  if rst = '1' then -- reset activo
    line <= 0;
  elsif rising_edge(hsync) then
    if line = BL then
      line <= 0;
    else
      line <= line + 1;
    end if;
  end if;
end process vertical_sync;

vert_blank: v_blank <= '1' when line > L else
  '0'; -- blanking

vert_sync:
vsync <= '0' when (line >= VB) and (line < VE)
  else '1'; --pulso de sincron. vertical

blank <= h_blank or v_blank;

pixel_data <= VRAM (line, pixel);

r <= "00" when blank = '1' else
  COLORMAP(pixel_data);

g <= "00" when blank = '1' else
  COLORMAP(pixel_data);

b <= "00" when blank = '1' else
  COLORMAP(pixel_data);

```

O primeiro processo implementa o contador horizontal de pixels. O contador é inicializado com 0 de uma maneira assíncrona quando a entrada de reset (*rst*) está activa. O contador incrementa o seu valor em cada transição ascendente do sinal de relógio *clk25*. A gama de valores do contador de pixels é [0, 793]. Quando o contador atinge o valor 793 ele volta novamente a 0 no ciclo seguinte do relógio. Sendo assim, o período do contador horizontal é de $793 / 25 \text{ MHz} = 31.76 \mu\text{s}$.

A linha marcada com a etiqueta *horiz_sync* no código acima gera o pulso de sincronização horizontal (sinal *hsync*). O sinal *hsync* recebe o valor '0' (activo) quando o contador de pixels horizontal se encontra no intervalo [652, 745] e o valor '1' em todos os casos restantes. Como resultado é produzido um pulso negativo com a duração de 94 ciclos de relógio, o que equivale a $94 / 25 \text{ MHz} = 3.76 \mu\text{s}$.

O segundo processo implementa o contador de linhas vertical. Este contador é inicializado com valor 0 assincronamente quando a entrada de reset está activa. O contador vertical incrementa o seu valor sempre que se atinge o final de uma linha. A gama de valores do contador vertical de linhas é [0, 527]. Quando o contador atinge o valor 527, ele volta novamente a 0 no ciclo seguinte, i.e. o período do contador é igual a 528 linhas, o que equivale a $528 \times 31.76 \mu\text{s} = 16.77 \text{ ms}$.

A linha marcada com a etiqueta *vert_sync* no código acima gera o pulso de sincronização vertical (sinal *vsync*). O sinal *vsync* recebe o valor '0' (activo) quando o contador de linhas vertical se encontra no intervalo [493, 494] e o valor '1' em todos os casos restantes. Como resultado é produzido um pulso negativo com a duração de 2 linhas, o que equivale a $2 \times 31.76 \mu\text{s} = 63.52 \mu\text{s}$. As linhas marcadas com as etiquetas *horiz_blank* e *vert_blank* no código acima geram os sinais de anulação (*blanking*) do sinal de vídeo, *hblank* e *vblank*, respectivamente. O sinal *hblank* é activado sempre que sejam visualizados 629 pixels numa linha enquanto o sinal *vblank* é activado sempre que sejam transmitidas 480 linhas. Os sinais *hblank* e *vblank* são somados logicamente para produzir o sinal de anulação global *blank* que sempre que activo coloca as saídas R, G, B a zero. Finalmente *VRAM* é uma memória de vídeo que guarda a imagem a visualizar no ecrã da qual podemos extrair o valor do pixel a transmitir (sinal *pixel_data*). A tabela *COLORMAP* define o mapeamento do valor do pixel em cores R, G, B. Se o sinal de vídeo é anulado, as cores são forçadas a valor 0.

IV. IMPLEMENTAÇÃO DO CONTROLADOR VGA QUE FUNCIONA NO MODO DE TEXTO

No modo de texto, podemos representar cada caracter como um *array* bidimensional de $a \times b$ pixels. Se escolhermos $a = b = 16$, uma imagem será composta por $480 / 16 = 30$ linhas de texto cada uma das quais irá incluir $624 / 16 = 39$ caracteres (dos 629 pixels que se pode apresentar só serão usados 624, os 5 restantes serão

anulados pois de qualquer modo não permitem representar um caracter inteiro).

Neste exemplo simples o texto branco é visualizado sobre o fundo azul. Para trabalhar com apenas duas cores precisaremos de somente um bit para representar o valor de cada pixel. O mapa de cores *COLORMAP* terá o conteúdo ilustrado na tabela 1.

Tabela1. Conteúdo da tabela *COLORMAP* para controlar monitor VGA no modo de texto simples

<i>pixel_data</i>	R	G	B	cor
0	00	00	11	azul
1	11	11	11	branca
<i>blank='1'</i> (activo)	00	00	00	preta

A Fig. 5 ilustra uma implementação possível do sistema. Neste caso são necessários dois blocos de memória. O bloco *S_ROM* é um gerador de caracteres que define a representação gráfica de todos os símbolos que é necessário visualizar com o formato de 16×16 pixels. Para tal é estabelecido um mapeamento entre o código ASCII de cada símbolo e um *array* bidimensional de 16×16 pixels. Por exemplo, o caracter 'A' pode ser representado de maneira seguinte (as colunas e linhas que só contêm zeros servem para implementar o espaçamento entre caracteres adjacentes e linhas consecutivas):

```
65 => ("0000011000000000", --A
      "0000111100000000",
      "0001111110000000",
      "0011100111000000",
      "0011100111000000",
      "0011100111000000",
      "0011100111000000",
      "0111111111000000",
      "0111111111000000",
      "0111000001110000",
      "1110000000111000",
      "1110000000111000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000")
```

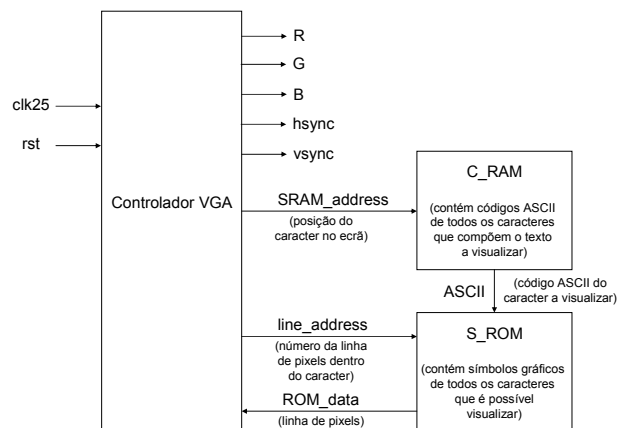


Fig. 5 – Estrutura de um simples controlador VGA no modo de texto

O segundo bloco de memória (*C_RAM*) contém códigos ASCII dos caracteres que compõem o texto a visualizar no monitor VGA. Na totalidade podemos mostrar no ecrã $30 \times 39 = 1170$ caracteres. Esta memória pode ser construída com base em blocos *BLOCK_RAM* embutidos na FPGA. Cada *BLOCK_RAM* das FPGAs da família Spartan-IIe é capaz de guardar até 4096 bits (com modos de endereçamento parametrizáveis). Se reservarmos 8 bits para representar códigos ASCII conseguiremos guardar $4096/8 = 512$ códigos (de caracteres) num *BLOCK_RAM*. Para representar 1170 caracteres de uma imagem são necessários três *BLOCK_RAM*.

O controlador VGA, para gerar endereços de dois blocos de memória e receber o valor do pixel, necessita conhecer que caracter e que linha de pixels que compõem o caracter estão a ser iluminados em cada momento. Esta informação pode ser calculada com a ajuda de 4 contadores controlados com os dois processos seguintes:

```

constant MAX_ROWS : natural := 30;
constant MAX_SYMBOLS : natural := 39;

--contadores de caracteres
signal text_row : natural range 0 to MAX_ROWS;
signal text_col : natural range 0 to MAX_SYMBOLS;

--contadores de linha/coluna dentro do caracter
signal char_row, char_col: natural range 0 to 15;

---
horizontal_text: process(clk25, rst)
begin
  if rst = '1' then --reset activo
    char_col <= 0;
    text_col <= 0;
  elsif rising_edge(clk25) then
    if pixel > P then --fim de pixels visíveis
      char_col <= 0;
      text_col <= 0;
    else
      if char_col = 15 -- caracter seguinte
        char_col <= 0;
        text_col <= text_col + 1;
      else char_col <= char_col + 1;
      end if;
    end if;
  end if;
end process horizontal_text;

vertical_text: process(hsync, rst)
begin
  if rst = '1' then --reset activo
    char_row <= 0;
    text_row <= 0;
  elsif rising_edge(hsync) then
    if line > L then --fim de linhas visíveis
      char_row <= 0;
      text_row <= 0;
    else
      if char_row = 15 then --linha seguinte
        char_row <= 0;
        text_row <= text_row + 1;
      else char_row <= char_row + 1;
      end if;
    end if;
  end if;
end process vertical_text;

```

O primeiro processo determina que coluna de caracteres do texto (sinal *text_col*) e que coluna de pixels dentro dum caracter (sinal *char_col*) estão a ser iluminadas em cada instante. Quando o sinal de inicialização *rst* está activo

ambos os contadores (*text_col* e *char_col*) são colocados a 0. O contador *char_col* é actualizado em cada transição ascendente do sinal de relógio *clk25* contando desse modo de 0 a 15 e voltando novamente a 0. Consequentemente, torna-se possível discriminar a posição horizontal dos pixels que compõem um caracter. Quando o contador *char_col* atinge o valor 15, no próximo ciclo de relógio, devemos já passar a enviar o caracter seguinte. Para isso é incrementado o contador de caracteres *text_col* de 16 em 16 pixels. Caso estejamos fora da área visível do ecrã, ambos os contadores são colocados a 0.

De maneira semelhante funcionam os dois contadores verticais *char_row* e *text_row* que são actualizados no segundo processo e servem para determinar que linha de texto (*text_row*) e que linha de pixels (*char_row*) dentro dum caracter está a ser varrida. O uso dos 4 contadores está ilustrado na Fig. 6.

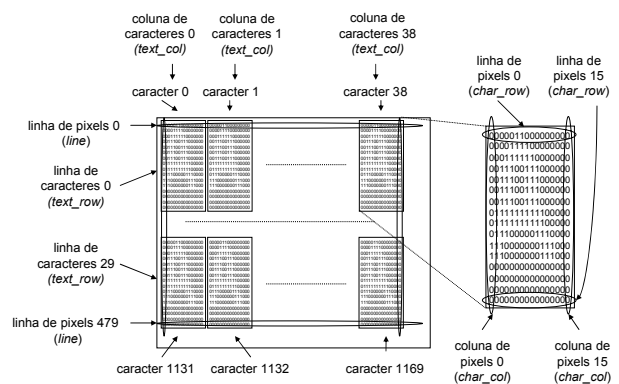


Fig. 6 – Endereçamento de vários elementos no ecrã

Finalmente, pode-se gerar os sinais de endereço para ambos os blocos de memória da Fig. 5. Para formar o endereço da *C_RAM* é preciso calcular qual é a posição relativa no ecrã (de 0 até 1169) do caracter que estará a ser iluminado a seguir. Estes cálculos podem ser feitos trivialmente com o código seguinte:

```

SRAM_address: out std_logic_vector(10 downto 0);
-- ...
--temos que ler o caracter seguinte em relação
--àquele que está a ser visualizado
SRAM_address <= (others => '0') when
(text_row = 0) and (text_col = 0) and (pixel > P)
else conv_std_logic_vector(text_row * MAX_SYMBOLS
+ text_col + 1, SRAM_address'length);

```

No código acima é declarada uma porta de saída de 11 bits (número de bits necessários para endereçar 1170 caracteres). Todos os caracteres no ecrã são numerados sequencialmente de 0 a 1169 (ver Fig. 6), tendo o primeiro caracter da primeira linha o número 0, o primeiro caracter da linha *k* o número do último caracter da linha *k-1* acrescentado de 1, e o último caracter da última linha o número 1169. Os dois bits mais significativos do sinal *SRAM_address* (que indica a posição do caracter no ecrã) vão distinguir entre os três *BLOCK_RAMs* que guardam o texto a visualizar no monitor. O código ASCII do caracter em causa é de seguida enviado para o bloco *S_ROM*.

De maneira semelhante calcula-se o endereço do bloco S_ROM da Fig. 5 que indica o número da linha de pixels dentro do caracter:

```
line_address : out std_logic_vector(3 downto 0);
---
--número da linha de pixels dentro do caracter
line_address <= conv_std_logic_vector(char_row,
                                     line_address'length);
```

O bloco S_ROM com base no código ASCII do caracter e no número da linha de pixels deste caracter que está a ser iluminada, gera os valores dos 16 pixels (no sinal ROM_data de 16 bits da Fig. 5), que são transformados em saídas R, G, B e enviados para o monitor. Os dados provenientes da S_ROM são carregados no registo pixel_data no início de transmissão de uma linha de pixels dum caracter e nos 15 ciclos de relógio seguintes são deslocados um bit à esquerda. O deslocamento faz com que o bit mais significativo do registo pixel_data contenha sempre o valor do pixel que está a ser iluminado naquele momento. O processo seguinte descreve o registo que guarda dados lidos da memória S_ROM:

```
signal pixel_data : std_logic_vector(0 to 15);
---
process (clk25, rst)
begin
  if (rst = '1') then --reset activo
    pixel_data <= (others => '0');
  elsif rising_edge(clk25) then
    if (char_col = 0) then
      pixel_data <= ROM_data;
    else --deslocamento à esquerda
      pixel_data <= pixel_data(1 to 15) & '0';
    end if;
  end if;
end process;
```

O sinal de anulação de saídas vídeo (*blank*) deve ser escalonado (*pipelined*) para que o possamos usar no ciclo seguinte de relógio, quando já será calculado o valor do pixel. Para tal o sinal *blank* pode ser guardado num registo conforme ilustrado no excerto de código seguinte:

```
signal blank, pblank : std_logic;
---
process (clk25, rst)
begin
  if (rst = '1') then --reset activo
    pblank <= '0';
  elsif rising_edge(clk25) then
    pblank <= blank;
  end if;
end process;
```

Finalmente, o valor do pixel corrente é mapeado nos 6 bits que representam as cores R, G, B de acordo com a tabela 1:

```
r <= "00" when pblank= '1' else
    (pixel_data(0), pixel_data(0));
g <= "00" when pblank= '1' else
    (pixel_data(0), pixel_data(0));
b <= "00" when pblank= '1' else "11";
```

Resta apenas definir a interface externa do controlador VGA que é especificada na declaração seguinte:

```
entity vga is
port (
```

```
clk25 : in std_logic; -- relógio de 25 MHz
rst   : in std_logic; -- reset

--interface com blocos de memória
ROM_data : in std_logic_vector(0 to 15);
--número da linha de pixels do caracter
line_address : out std_logic_vector
              (3 downto 0);

--posição do caracter no ecrã (de 0 a 1169)
SRAM_address : out std_logic_vector
              (10 downto 0);

--sinais para o monitor VGA
hsync : inout std_logic; --sincr. horizontal
vsync : out std_logic; --sincr. vertical
--intensidades das três cores primárias
r      : out std_logic_vector(1 downto 0);
g      : out std_logic_vector(1 downto 0);
b      : out std_logic_vector(1 downto 0);
end vga;
```

V. CONCLUSÕES

Neste artigo descrevemos como circuitos reconfiguráveis baseados em FPGA podem interagir com monitores VGA. Em particular foram revistas as características temporais de monitores VGA e foi ilustrado detalhadamente o código VHDL que implemente um controlador VGA para a placa TE-XC2Se [7]. Estas informações são leccionadas no âmbito das disciplinas Computação Reconfigurável e Sistemas Digitais Reconfiguráveis e servem de base para os alunos criarem circuitos simples semelhantes nos laboratórios e fazerem projectos mais complexos fora das horas lectivas. Seis projectos individuais de alunos desenvolvidos no 2º semestre do ano lectivo 2004/2005 que suportam a interacção com um monitor VGA, encontram-se descritos em artigos que seguem [1-6].

REFERÊNCIAS

- [1] F. Manana, "Desenvolvimento de uma calculadora booleana em VHDL", *Electrónica e Telecomunicações, Set. 2005* (nesta revista).
- [2] A. Cantanhede, "Implementação de um controlador de semáforos numa FPGA com visualização num monitor VGA", *Electrónica e Telecomunicações, Set. 2005* (nesta revista).
- [3] S. Veiga, "Desenvolvimento de um "Counter" & "Shifter" em VHDL", *Electrónica e Telecomunicações, Set. 2005* (nesta revista).
- [4] R. Santos, "Implementação de um projecto em VHDL para a execução de várias operações sobre um vector booleano", *Electrónica e Telecomunicações, Set. 2005* (nesta revista).
- [5] C. Pires, "Desenvolvimento de uma calculadora baseada numa FPGA e num VGA", *Electrónica e Telecomunicações, Set. 2005* (nesta revista).
- [6] L. Gomes, "Desenvolvimento de um analisador lógico simples", *Electrónica e Telecomunicações, Set. 2005* (nesta revista).
- [7] Trenz Electronic GmbH, Products, [Online]: <http://www.trenz-electronic.de/home/indexen.htm>.
- [8] Tutorials. [Online]: <http://www.ieeta.pt/~iouliaa/Courses/SDR/tutorials/>.
- [9] V. Sklyarov, I. Skliarova, "Teaching Reconfigurable Systems: Methods, Tools, Tutorials and Projects", *IEEE Transactions on Education*, vol. 48, no. 2, Maio 2005, pp. 290-300.
- [10] D. Vanden Bout, XESS Corporation, "VGA Generator for the XSA Boards", Application Note, Out. 2004.