ENG1448: Computação Digital - PUC-Rio, 2024.1, 2024.03.11

Laboratório 10: Controlador VGA e Desafio:

Professor: Felipe Calliari | Monitor: Cristiano Nascimento

Aluno: Pedro Gabriel Serodio Sales e Thiago Levis *Matrícula: 2211911 e 1812899*

1 Introdução

O objetivo deste relatório é apresentar as soluções para o laboratório 10 da disciplina ENG1448 assim como apresentações do laboratório em vídeo.

2 Resolução

2.1 : **Geração do Clock**

O primeiro passo foi criar um clock de 40 MHz, correspondente a tal resolução. Como sugerido no enunciado, usamos o IP Core fornecido no ambiente de desenvolvimento, Digital Clock Manager (DCM-SP), de modo que o clock de entrada fosse o de funcionamento normal da FPGA, 50 MHz, e o clock de saída de 40 MHz. Vale ressaltar que o manual deste componente indica que, apesar da saída STATUS apresentar 8 bits, apenas 3 deles estão definidos (bits de 0 a 2). Assim, apenas os 3 bits menos significativos são associados aos LEDs discretos da placa:

- STATUS[0]: indica um overflow do phase shift numerator
- STATUS[1]: indica a perda do clock de entrada, CLKIN
- STATUS[2]: indica que o clock de saída foi interrompido, CLKFX

Além disso, o manual define a saída LOCKED como um sinal indicador de que o clock gerado pelo DCM se encontra estável. No entanto, para simplificar decidimos deixar essa porta de saída aberta, considerando que uma possível desestabilização inicial do clock de saída não seria perceptível no monitor. Por fim, também é informado que CLKFB (feedback) é opcional, logo a saída CLK0 (clock de entrada com defasagem de 0ž) ligada a essa porta também é deixada aberta.

2.2 : **Controlador VGA**

A partir do código referente à entidade VGA-CTRL faz-se o controle de varredura da tela. Para tanto, a tela é interpretada como uma matriz de pixels, que devem ser acionados um a um, da esquerda para a direita, de cima para baixo. Dessa forma, a cada ciclo de clock (40MHz), um pixel do monitor é mostrado sob certa configuração de cores RBG (explicado mais adiante).

No entanto, o protocolo VGA também leva em consideração os conceitos de back-porch e front-porch dos antigos tubos de raios catódicos, além de pulsos que indicam a leitura de uma linha completa e do frame completo (Figura 1). Na prática, é preciso varrer mais pixels do que os que compõem a resolução de 800x600 para cobrir os tempos necessários do protocolo. Portanto, varre-se uma matriz de dimensão maior que emoldura a matriz original com a resolução especificada

De forma simplificada, o algoritmo usa dois contadores (h-counter, v-counter) para iterar entre os pixels da matriz moldura. A cada ciclo de clock um pixel de uma dada linha da matriz é varrido e incrementa-se h-counter. Quando todos os 800 pixels da linha são lidos, varre-se os pixels invisíveis (fora da região azul, Figura 2) que contabilizam os tempos de front-porch, sync pulse e back-porch horizontais. Daí, repete-se tal procedimento, incrementando o contador v-counter e mostrando todas as 600 linhas. Porém o algoritmo prossegue com as iterações sobre os pixels invisíveis restantes, a fim de contabilizar os tempos de front-porch, sync pulse e back-porch verticais.

2.3 : **Controlador de Pixel**

Assim como o módulo controlador do protocolo VGA, adaptamos o código de controle individual dos pixels, contido na entidade PIXEL-CTRL. Como o objetivo desta prática era apenas desenhar algum padrão visual diferente das 7 linhas verticais coloridas, alteramos o código para que fossem desenhados na tela 3 quadrados de cores vermelho, verde e azul.

Em termos de implementação, o componente recebe nas portas de entrada pixel-x e pixel-y as coordenadas do pixel recém lido na varredura matricial do componente VGA-CTRL. Com isso, é possível determinar se o pixel está inserido ou não nos intervalos que delimitam os quadrados coloridos. Em caso afirmativo, as portas de saída red, green e blue assumem os valores da codificação RGB correspondente; caso contrário, configura-se um pixel preto, atribuindo às 3 portas valor lógico 0. Nota-se que a configuração de cores do pixel só é considerada quando a porta de entrada display-on apresenta valor lógico 1, indicando que este é um dos pixels contido na área visível (região em azul, Figura 2). Em outros casos é atribuída ao pixel a configuração default de pixel preto.

2.4 : **Modulo Integrador**

Para unificar o funcionamento dos componentes VGA-CTRL e PIXEL-CTRL, criou-se um módulo integrador que instancia cada um destes e os interliga por meio de sinais auxiliares da arquitetura. Assim, foram conectadas as saídas display-on, pixel-x e pixel-y do VGA-CTRL às respectivas entradas do PIXEL-CTRL. Além disso, também é instanciado um Digital Clock Manager (DCM) que gera um clock de saída de 40 MHz e que serve como o clock de entrada dos demais componentes.

Por fim, o módulo integrador pode ser visto como um componente que possui as portas de entrada CLK e RST (sendo este clock o de 50 MHz da FPGA), e as portas de saída VGA-RED, VGA-GREEN, VGA-BLUE, VGA-HSYNC, VGA-VSYNC. Ressalta-se que os 3 primeiros sinais de saída equivalem às portas de saída red, green e blue da instância do PIXEL-CTRL, e as duas últimas às saídas ao h-sync e v-sync do VGA-CTRL.

Observacao: a porta de saída LEDS de 8 bits tem valor variável apenas nos 3 menos significativos, sendo estes os valores correspondentes a codificação de status de funcionamento do DCM.

2.5 : **Arquivo de Constraints**

```
# VGA Constraints

NET "VGA_RED" LOC = "H14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_GREEN" LOC = "H15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_BLUE" LOC = "G15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_HSYNC" LOC = "F15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_VSYNC" LOC = "F14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;

# Discrete LED's Constraints

NET "LEDS<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LEDS<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

# Clock Constraints

NET "CLK" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "CLK" PERIOD = 20.0ns HIGH 50%;

NET "RST" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN ;
```

(1)

Figura 1: Mapeamento das portas na placa e sinais usados pela FPGA

2.6 : **Desafio do Laboratorio**

Tendo feito essa versão inicial decidimos implementar uma versão com apenas um quadrado que se move na tela, tendo seu movimento limitado pelos limites da tela, verticais e horizontais. Para tanto, define-se as fronteiras horizontais e verticais em que desenha-se na tela pixels coloridos que constituem o quadrado como a hitbox do objeto. Assim, nosso objetivo passou a ser alterar os valores que delimitam a hitbox do quadrado com incrementos ou decrementos de posição no plano (x,y) definido pela tela.

Para tanto, criou-se um novo processo síncrono que incrementa um contador até 1000000. Ao atingir esse limiar, resetamos o contador zerando-o e atualiza-se as coordenadas horizontais e verticais referentes a hitbox, como descrito nas imagens abaixo:

```
if (hit_box_xi <= 0) then
   hit_box_xi <= hit_box_xi + vel_x;
   img_vel_x  <= vel_x;
   idx_color <= idx_color + 1;
   if idx_color + 1 = colors'length then
      idx_color <= 0;
   end if;
elsif (hit_box_xf >= h_pixels - 1) then
   hit_box_xi <= hit_box_xi - vel_x;
   img_vel_x  <= -vel_x;
   idx_color <= idx_color + 1;
   if idx_color + 1 = colors'length then
      idx_color <= 0;
   end if;
else
   hit_box_xi <= hit_box_xi + img_vel_x;
end if;
```

(2)

Figura 2: Atualizacao das coordenadas horizontais

```
if (hit_box_yi <= 0) then
   hit_box_yi <= hit_box_yi + vel_y;
   img_vel_y  <= vel_y;
   idx_color <= idx_color + 1;
   if idx_color + 1 = colors'length then
      idx_color <= 0;
   end if;
elsif (hit_box_yf >= v_pixels - 1) then
   hit_box_yi <= hit_box_yi - vel_y;
   img_vel_y  <= -vel_y;
   idx_color <= idx_color + 1;
   if idx_color + 1 = colors'length then
      idx_color <= 0;
   end if;
else
   hit_box_yi <= hit_box_yi + img_vel_y;
end if;
```

(3)

Figura 3:Atualizacao das coordenadas verticais

Ressalta-se que nos códigos acima também consideramos os casos em que o quadrado se choca com os limites horizontais e verticais da tela. Nesses casos, invertemos o sentido de sua velocidade no eixo de colisão, x ou y, e troca-se sua cor para a próxima cor no vetor colors. Daí tem-se uma versão do processo síncrono que desenha os pixels coloridos que constituem o quadrado, considerando dinamicamente os limites da hitbox do mesmo e sua cor atual sendo a indexação do vetor de cores.

```
process(pixel_clock)
begin
   if rising_edge(pixel_clock) then

--       -- Código inicial que desenha 3 quadrados na tela,
--       --    cada um de uma cor: vermelho, verde e azul
--       --
--       if (pixel_x >= 350 and pixel_x <= 450) then
--          if (pixel_y >= 250 - 100 and pixel_y <= 350 - 100) then
--             RGB <= "001"; -- Blue
--          end if;
--       elsif (pixel_x >= 350 - 150 and pixel_x <= 450 - 150) then
--          if (pixel_y >= 250 + 100 and pixel_y <= 350 + 100) then
--             RGB <= "100"; -- Red
--          end if;
--       elsif (pixel_x >= 350 + 150 and pixel_x <= 450 + 150) then
--          if (pixel_y >= 250 + 100 and pixel_y <= 350 + 100) then
--             RGB <= "010"; -- Green
--          end if;
--       else
--          RGB <= "000"; -- Black
--       end if;

         if (pixel_x >= hit_box_xi and pixel_x <= hit_box_xf) and
            (pixel_y >= hit_box_yi and pixel_y <= hit_box_yf) then
            RGB <= colors(idx_color);
         else
            RGB <= "000"; -- Black
         end if;

   end if;
end process;
```

(4)

Figura 4: Pixel Clock

Como resultado dessa implementação do desafio do laboratório, gravamos um breve vídeo mostrando o quadrado se movendo ao redor da tela. Ao colidir com as fronteiras verticais e horizontais da mesma ele muda de cor, variando entre vermelho, azul, verde e amarelo.

3 Vídeo de Apresentação

O vídeo de apresentação segue ao lado: **Lab 10: Controlador VGA + Desafio**

ou acessar pelo url:**`https://youtu.be/1IOO9ujYo5g`**