Laboratório 7

Gráficos e Animação

O propósito deste laboratório é aprender a gerar imagens e realizar animação. Usaremos o sistema *DE2 Media Computer* e o conversor digital para analógico (DAC) do hardware gráfico VGA (*Video Graphics Array*).

Introdução

O sistema *DE2 Media Computer* usa um conjunto de circuitos, chamados de núcleos (*cores*), para controlar o circuito VGA DAC e gerar imagens na tela. Dois desses circuitos são o buffer de pixels VGA e o circuito controlador VGA, que são usados em conjunto com a memória SRAM e o controlador SRAM para permitir que programas executados pelo processador Nios II possam gerar imagens na tela. A parte do *DE2 Media Computer* usada neste laboratório é mostrada através da Figura 1.

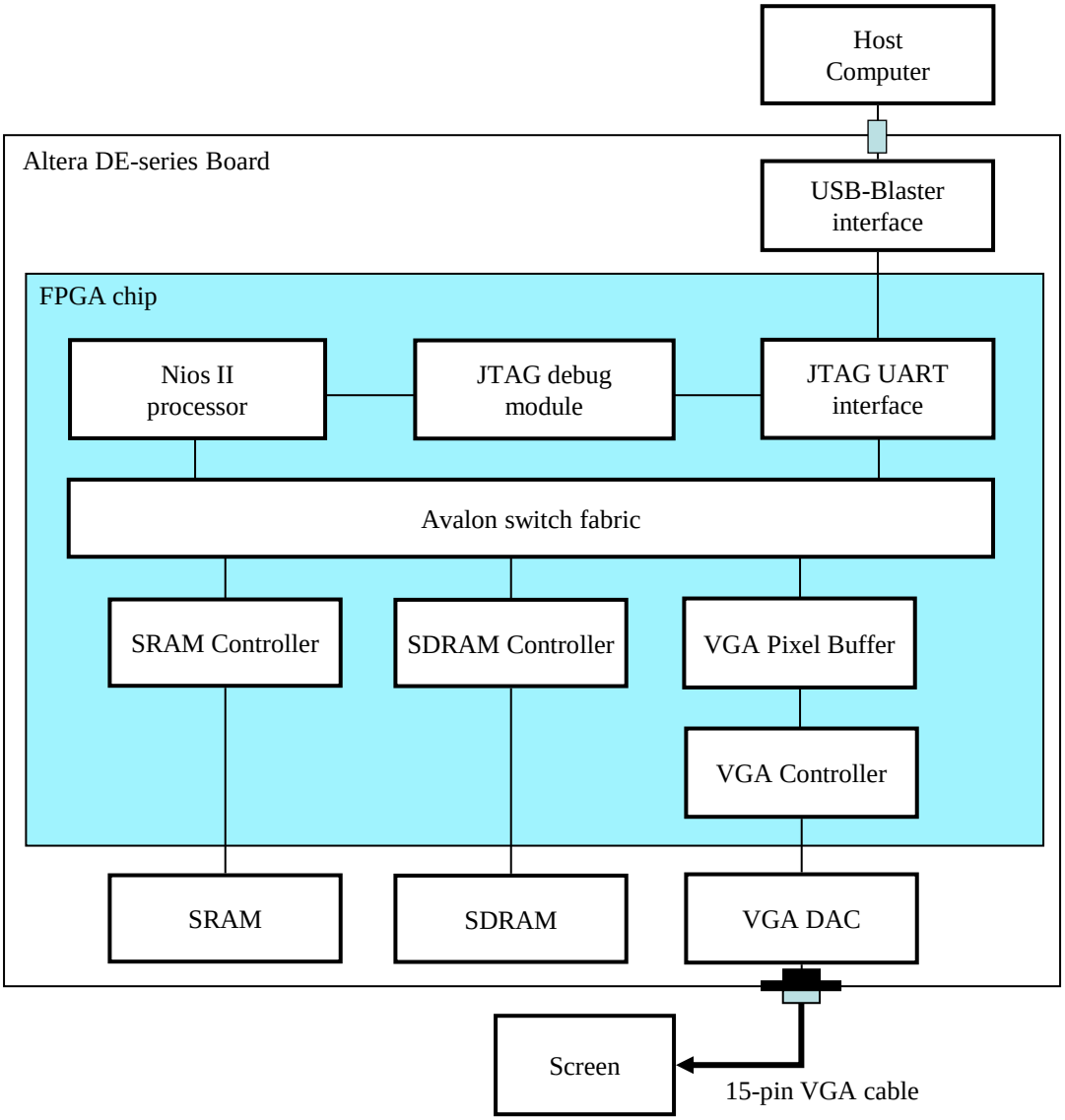


Figura 1. Parte do *DE2 Media Computer* usada neste laboratório.

O buffer de pixels VGA é uma interface entre programas executados no processador Nios II e o controlador VGA.

Ele contém o tamanho da tela e a posição na memória SRAM em que a imagem a ser mostrada será armazenada. Para gerar uma imagem na tela, o buffer de pixels VGA recupera as informações dos pixels da memória e os envia para o controlador VGA. Por sua vez, o controlador VGA usa o circuito VGA DAC para enviar os dados relativos à imagem através do cabo VGA até a tela do monitor.

Uma imagem consiste de vetores retangulares de elementos de figura, chamados *pixels*. Cada pixel aparece como um ponto na tela, e toda a tela consiste de uma matriz de 320 colunas por 240 linhas de pixels, como ilustrado pela Figura 2. Os pixels são arranjados em uma grade retangular, com a coordenada $(0, 0)$ no canto superior esquerdo da tela.

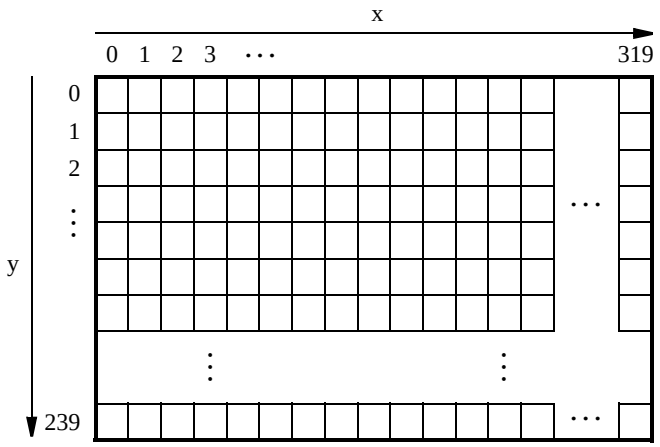
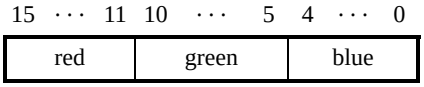
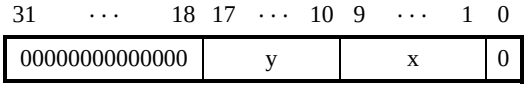


Figura 2. Vetores de pixels.

A cor de um pixel é a combinação de três cores primárias: vermelho, verde e azul. Ao variar a intensidade de cada cor primária, qualquer outra cor pode ser criada. Usaremos meia palavra de 16 bits para representar a cor de um pixel. Os cinco bits mais significativos e menos significativos nesta meia palavra representam a intensidade das componentes vermelho e azul, respectivamente, enquanto os seis bits restantes representam a intensidade da componente verde, como mostrado na Figura 3a. Por exemplo, a cor vermelha seria representada pelo valor $(F800)_{16}$, a cor roxa pelo valor $(F81F)_{16}$, branco por $(FFFF)_{16}$, e cinza por $(8410)_{16}$.



(a) Pixel color



(b) Pixel (x,y) offset

Figura 3. Cor e deslocamento do pixel.

A cor de cada pixel na imagem é armazenada no endereço correspondente em um buffer na memória SRAM. O endereço do pixel é uma combinação de um endereço base e um deslocamento (x, y). No *DE2 Media Computer*, o buffer é localizado no endereço $(08000000)_{16}$, que é o endereço inicial da memória SRAM. O deslocamento (x, y) é computado como a concatenação de 9 bits da coordenada x começando pelo primeiro bit, e 8 bits da coordenada y começando pelo décimo bit, como mostrado na Figura 3b. Em linguagem C, esta computação pode ser feita com o uso do operador de deslocamento para esquerda:

$$\text{offset} = (x << 1) + (y << 10)$$

Para determinar a localização de cada pixel na memória, adicionamos o deslocamento (x, y) ao endereço base. Usando este esquema, o pixel na posição $(0, 0)$ tem o endereço $(08000000)_{16}$, o pixel em $(1, 0)$ está no endereço base + $(00000002)_{16}$ = $(08000002)_{16}$, o pixel em $(0, 1)$ está no endereço base + $(00000400)_{16}$ = $(08000400)_{16}$, e o pixel na posição $(319, 239)$ está no endereço base + $(0003BE7E)_{16}$ = $(0803BE7E)_{16}$.

Para gerar imagens a partir de um programa executando no *DE2 Media Computer*, o módulo de buffer de pixels VGA contém registradores mapeados em memória que são usados para acessar a informação do módulo e controlar sua operação. Estes registradores, localizados no endereço inicial $(10003020)_{16}$, estão listados na Figura 4.

Address	31 . . . 24	23 . . . 16	15 . . . 8	7 . . . 4	3	2	1	0	
0x10003020	front buffer address								Buffer register
0x10003024	back buffer address								Backbuffer register
0x10003028	Y		X						Resolution register
0x1000302C	m	n	Unused	B	Unused	A	S		Status register

Figura 4. Registradores mapeados em memória do buffer de pixels VGA.

Os registradores *Buffer* e *Backbuffer* armazenam a localização na memória na qual os dois buffers da imagem estão localizados. O primeiro buffer, chamado de *front buffer*, é a memória onde a imagem atualmente visível na tela é armazenada. O segundo buffer, chamado de *back buffer*, é usado para desenhar a próxima imagem a ser mostrada. Inicialmente, ambos registradores armazenam o valor $(08000000)_{16}$.

O registrador de resolução (*Resolution*) armazena a largura e altura da tela em termos de pixels. Os 16 bits mais significativos indicam a resolução vertical, enquanto os 16 bits menos significativos indicam a resolução horizontal da tela. O registrador *Status* armazena informação sobre o buffer de pixels VGA. Discutiremos o uso desses registradores conforme necessário durante este laboratório.

Parte I

Implemente uma rotina para limpar a tela com uma determinada cor (passada como parâmetro) e outra para desenhar um pixel na coordenada (x, y). Os protótipos das funções são dados abaixo.

```
void clear_screen();
void draw_pixel(int x, int y, short int color);
```

Você vai precisar dessas rotinas nas demais partes desse laboratório. Para testá-las, crie um novo projeto no Altera Monitor e escreva um programa que use as sub-rotinas para limpar a tela e depois desenhar pixels em algumas posições (teste com várias cores). Conecte o cabo VGA de 15 pinos na placa DE2 em um monitor.

Parte II

Neste parte você aprenderá a implementar um algoritmo simples para desenhar linhas.

Desenhar uma linha na tela requer colorir pixels entre dois pontos, (x_1, y_1) e (x_2, y_2), de tal forma que eles assemelhem-se ao máximo a uma linha. Considere o exemplo na Figura 5.

Queremos desenhar uma linha entre os pontos $(1, 1)$ e $(12, 5)$. Os quadrados representam pixels que podem ser coloridos. Para desenhar uma linha usando os pixels, temos que seguir a linha e, para cada coluna, colorir o pixel mais perto da linha. Para formar uma linha entre os pontos $(1, 1)$ and $(12, 5)$, nós colorimos os pixels cinzentos na figura.

Podemos usar álgebra para determinar quais pixels colorir. Isto é feito usando os pontos extremos e a inclinação (coeficiente angular) da linha. A inclinação da curva é $inclinacao = (y_2 - y_1)/(x_2 - x_1) = 4/11$. Começando

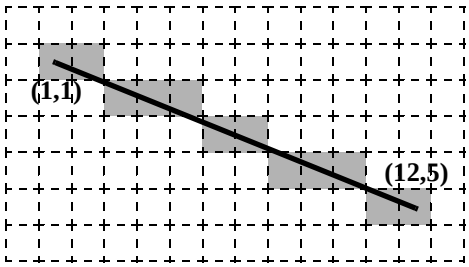


Figura 5. Um exemplo de desenho de uma linha entre os pontos $(1, 1)$ e $(12, 5)$.

do ponto $(1, 1)$ podemos mover ao longo do eixo x e computar a coordenada y para a linha como a seguir:

$$y = inclinacao \times (x - x_1) + y_1$$

Logo, para a coluna $x = 2$, a posição y do pixel é $\frac{4}{11} + 1 = 1\frac{4}{11}$. Dado que as posições do pixel são definidas por valores inteiros, temos de arredondar o valor da coordenada y para o inteiro mais próximo e determinar que para a coluna $x = 2$ temos de colorir o pixel na posição $y = 1$. Realizamos esta computação para cada coluna entre x_1 e x_2.

A abordagem de percorrer o eixo x possui uma desvantagem quando a linha é íngreme. Uma linha íngreme possui mais linhas do que colunas, logo se o algoritmo percorrer o eixo x para computar a coordenada y para cada coluna haverá lacunas na linha. Por exemplo, uma linha vertical possui todos os pontos em uma única coluna, e o algoritmo falharia ao desenhá-la corretamente. Para remediar esse problema, vamos alterar o algoritmo para percorrer o eixo y quando a linha for muito íngreme. Com esta mudança, podemos implementar o algoritmo conhecido como algoritmo de Bresenham. O pseudo-código para este algoritmo é apresentado na Figura 6.

```
1  draw_line(x0, x1, y0, y1)
2      boolean is_steep = abs(y1 - y0) > abs(x1 - x0)
3      if is_steep then
4          swap(x0, y0)
5          swap(x1, y1)
6      if x0 > x1 then
7          swap(x0, x1)
8          swap(y0, y1)
9      int deltax = x1 - x0
10     int deltay = abs(y1 - y0)
11     float error = 0
12     float slope = deltay / deltax
13     int y_step
14     int y = y0
15     if y0 < y1 then y_step = 1 else y_step = -1
16     for x from x0 to x1
17         if is_steep then draw_pixel(y,x) else draw_pixel(x,y)
18         error = error + slope
19         if error >= 0.5 then
20             y = y + y_step
21             error = error - 1.0
```

Figura 6. Pseudo-código do algoritmo de desenho de linha.

Este algoritmo usa operações em ponto flutuante para computar a localização de cada pixel na linha. Dado que operações em ponto flutuante são geralmente muito mais lentas do que operações inteiras, a maioria das implementações deste algoritmo é alterada para usar somente operações inteiras.

Escreva uma programa em linguagem C para desenhar algumas linhas na tela usando este algoritmo. Faça o seguinte:

1. Escreva um programa em linguagem C que implemente o algoritmo de linha.
2. Crie um novo projeto para o *DE2 Media Computer* usando o programa Altera Monitor.
3. Carregue o *DE2 Media Computer* na placa FPGA.
4. Compile e execute seu programa.

Em seu relatório, comente porque a memória SRAM foi escolhida para armazenar os pixels.

Part III

Animação é um aspecto importante em computação gráfica. Mover um objeto na tela é uma ilusão criada ao mostrar o mesmo objeto em diferentes posições na tela. Para mover um objeto na tela devemos primeiramente mostrá-lo em uma posição, e logo a seguir em uma posição diferente. Um modo simples de fazê-lo é desenhar um objeto em uma posição, e então apagá-lo e desenhá-lo em outra posição.

A chave para a animação é a temporização, já que para desempenhar a animação é necessário mover objetos em intervalos de tempo regulares. Os intervalos de tempo dependem do controlador gráfico. O controlador VGA no sistema *DE2 Media Computer* redesenha a tela a cada $1/60$ avos de segundo. Já que a imagem na tela não pode mudar mais rapidamente que isso, esta será a unidade de tempo.

Para garantir que desenhemos na tela uma vez a cada $1/60$ avos de segundo, usamos o pixel buffer VGA para sincronizar o programa executado no *DE2 Media Computer* com o ciclo de redesenho do controlador VGA. Isso é feito ao se escrever o valor 1 no registrador *Buffer* e esperar até que o bit 0 do registrador *Status* no pixel buffer VGA se torne 0. Isso significa que $1/60$ avos de segundo se passaram desde a última vez que a imagem foi desenhada na tela.

Escreva um programa em linguagem C para mover uma linha horizontal verticalmente através da tela. Ao colidir com as partes superiores e inferiores da tela, a linha deve mudar sua direção. O programa deve primeiramente limpar a tela, configurando todos os pixels para a cor preta, e então repetidamente desenhar e apagar (desenhe a mesma linha usando a cor preta) a linha durante todo ciclo de redesenho. Quando a linha atingir a parte de cima ou de baixo da tela, ela deve começar a se mover na direção oposta.

Copyright ©2011 Altera Corporation.
Tradução para o português de Alexandro Baldassin.