

实验九 VGA 接口控制器实现

2018 年秋季学期

主板上波光粼粼，显示阵列上的各色标志此起彼伏地闪动，人列计算机开始了漫长的计算。

— 《三体》，刘慈欣

VGA 接口是 IBM 制定的一种视频数据的传输标准，是电脑显示器最典型的接口。本实验的目的是学习 VGA 接口原理，学习 VGA 接口控制器的设计方法。

9.1 VGA 简介

9.1.1 VGA 接口的外观和引脚功能

VGA（Video Graphics Array）接口，即视频图形阵列。VGA 接口最初是用于连接 CRT 显示器的接口，CRT 显示器因为设计制造上的原因，只能接受模拟信号输入，这就需要显卡能输出模拟信号，VGA 接口就是显卡上输出模拟信号的接口，在传统的 CRT 显示器中，使用的都是 VGA 接口。VGA 接口是 15 针/孔的梯形插头，分成 3 排，每排 5 个，如图 9-1 所示：

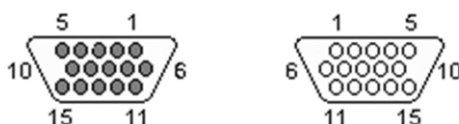


图 9-1: VGA 接口形状示意图

VGA 接口的接口信号主要有 5 个：R（Red）、G（Green）、B（Blue）、HS（Horizontal Synchronization）和 VS（Vertical Synchronization），即红、绿、蓝、水平同步和垂直同步（也称行同步和帧同步）。

9.1.2 VGA 的工作原理

图像的显示是以像素（点）为单位，显示器的分辨率是指屏幕每行有多少个像素及每帧有多少行，标准的 VGA 分辨率是 640×480，也有更高的分辨率，

如 1024×768 、 1280×1024 、 1920×1200 等。从人眼的视觉效果考虑，屏幕刷新的频率（每秒钟显示的帧数）应该大于 24，这样屏幕看起来才不会闪烁，VGA 显示器一般的刷新频率是 60HZ。

每一帧图像的显示都是从屏幕的左上角开始一行一行进行的，行同步信号是一个负脉冲，行同步信号有效后，由 RGB 端送出当前行显示的各像素点的 RGB 电压值，当一帧显示结束后，由帧同步信号送出一个负脉冲，重新开始从屏幕的左上端开始显示下一帧图像，如图 9-2 所示。

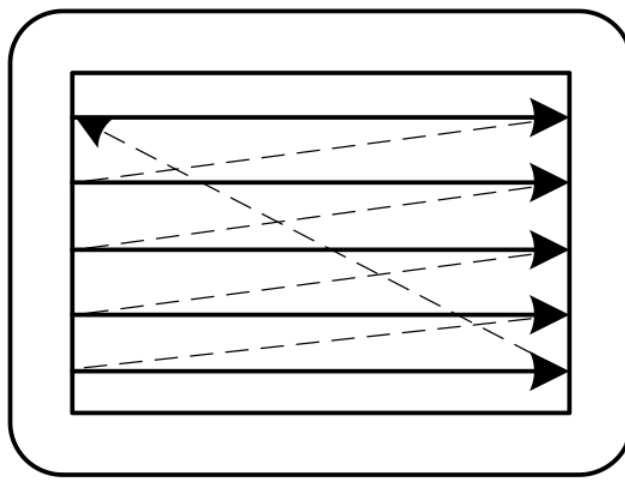


图 9-2: 显示器扫描示意图

RGB 端并不是所有时间都在传送像素信息，由于 CRT 的电子束从上一行的行尾到下一行的行头需要时间，从屏幕的右下角回到左上角开始下一帧也需要时间，这时 RGB 送的电压值为 0（黑色），这些时间称为电子束的行消隐时间和场消隐时间，行消隐时间以像素为单位，帧消隐时间以行为单位。VGA 行扫描、场扫描时序示意图如图 9-3 所示：

由图 9-3 可知，有效地显示一行信号需要 $96+48+640+16=800$ 个像素点的时间，其中行同步负脉冲宽度为 96 个像素点时间，行消隐后沿需要 48 个像素点时间，每行显示 640 个像素点，行消隐前沿需要 16 个像素点的时间，一行显示时间为 640 个像素点时间，一行消隐时间为 160 个像素点时间。

有效显示一帧图像需要 $2+33+480+10=525$ 行时间，其中场同步负脉冲宽度为 2 个行显示时间，场消隐后沿需要 33 个行显示时间，每场显示 480 行，场消隐前沿需要 10 个行显示时间，一帧显示时间为 525 行显示时间，一帧消隐时间为 45 行显示时间。

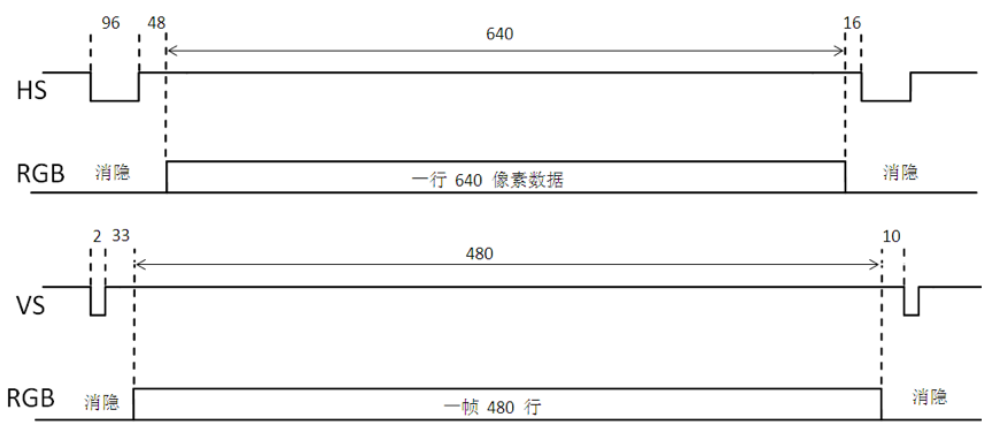


图 9-3: VGA 行扫描、场扫描时序示意图

9.2 VGA 显示的实现

9.2.1 DE10-Standard 开发板上的 VGA 接口

DE10-Standard 开发板上使用了一块 VGA DAC ADV7123 芯片来实现 VGA 功能。该芯片完成 FPGA 数字信号到 VGA 模拟信号的转换，具体连接方式如图 9-4 所示。

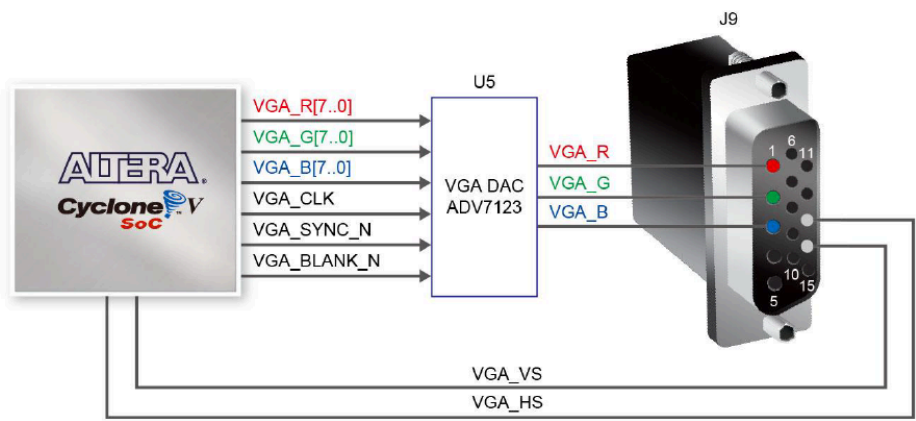


图 9-4: DE10-Standard 的 VGA 连接示意图

开发板和 ADV7123 芯片之间的接口引脚包括 3 组 8bit 的颜色信号 VGA_R[7:0], VGA_G[7:0], VGA_B[7:0], 行同步信号 VGA_HS, 帧同步信号 VGA_VS, VGA 时钟信号 VGA_CLK, VGA 同步 (低有效) VGA_SYNC_N,

和 VGA 消隐信号（低有效）VGA_BLANK_N。如图 9-5 所示。

<i>Signal Name</i>	<i>FPGA Pin No.</i>	<i>Description</i>	<i>I/O Standard</i>
VGA_R[0]	PIN_AK29	VGA Red[0]	3.3V
VGA_R[1]	PIN_AK28	VGA Red[1]	3.3V
VGA_R[2]	PIN_AK27	VGA Red[2]	3.3V
VGA_R[3]	PIN_AJ27	VGA Red[3]	3.3V
VGA_R[4]	PIN_AH27	VGA Red[4]	3.3V
VGA_R[5]	PIN_AF26	VGA Red[5]	3.3V
VGA_R[6]	PIN_AG26	VGA Red[6]	3.3V
VGA_R[7]	PIN_AJ26	VGA Red[7]	3.3V
VGA_G[0]	PIN_AK26	VGA Green[0]	3.3V
VGA_G[1]	PIN_AJ25	VGA Green[1]	3.3V
VGA_G[2]	PIN_AH25	VGA Green[2]	3.3V
VGA_G[3]	PIN_AK24	VGA Green[3]	3.3V
VGA_G[4]	PIN_AJ24	VGA Green[4]	3.3V
VGA_G[5]	PIN_AH24	VGA Green[5]	3.3V
VGA_G[6]	PIN_AK23	VGA Green[6]	3.3V
VGA_G[7]	PIN_AH23	VGA Green[7]	3.3V
VGA_B[0]	PIN_AJ21	VGA Blue[0]	3.3V
VGA_B[1]	PIN_AJ20	VGA Blue[1]	3.3V
VGA_B[2]	PIN_AH20	VGA Blue[2]	3.3V
VGA_B[3]	PIN_AJ19	VGA Blue[3]	3.3V
VGA_B[4]	PIN_AH19	VGA Blue[4]	3.3V
VGA_B[5]	PIN_AJ17	VGA Blue[5]	3.3V
VGA_B[6]	PIN_AJ16	VGA Blue[6]	3.3V
VGA_B[7]	PIN_AK16	VGA Blue[7]	3.3V
VGA_CLK	PIN_AK21	VGA Clock	3.3V
VGA_BLANK_N	PIN_AK22	VGA BLANK	3.3V
VGA_HS	PIN_AK19	VGA H_SYNC	3.3V
VGA_VS	PIN_AK18	VGA V_SYNC	3.3V
VGA_SYNC_N	PIN_AJ22	VGA SYNC	3.3V

图 9-5: DE10 Standard 的 VGA 引脚

VGA 信号首先需要有一个时钟驱动，我们这里使用 25MHz 的时钟来驱动 VGA_CLK。每个时钟周期扫过一个像素点，因此在 640×480 的分辨率下，我们需要 $800 \times 525 = 420,000$ 个时钟周期才能扫描完一帧（此处考虑了消隐的时间）。在 25MHz 的时钟周期下总时长为 16.8 毫秒，对应约每秒约 60 帧。

我们使用一个简单的分频器来从 50MHz 的时钟来产生所需的 VGA_CLK。

表 9-1: 通用时钟生成代码

```

1 module clkgen(
2     input clkkin,
3     input rst,
4     input clken,
5     output reg clkout
6 );
7     parameter clk_freq=1000;
8     parameter countlimit=50000000/2/clk_freq; // 自动计算计数次数
9
10    reg[31:0] clkcount;
11    always @ (posedge clkkin)
12        if(rst)
13            begin
14                clkcount=0;
15                clkout=1'b0;
16            end
17        else
18            begin
19                if(clken)
20                    begin
21                        clkcount=clkcount+1;
22                        if(clkcount>=countlimit)
23                            begin
24                                clkcount=32'd0;
25                                clkout=~clkout;
26                            end
27                        else
28                            clkout=clkout;
29                    end
30                else
31                    begin
32                        clkcount=clkcount;
33                        clkout=clkout;
34                    end
35            end
36 endmodule

```

该生成器可以按照调用时的参数来生成不同频率的时钟：

```

1 clkgen #(25000000) my_vgaclk(CLOCK_50,SW[0],1'b1,VGA_CLK);

```

在该时钟的驱动下我们需要生成各类驱动信号。其中 VGA 同步信号 VGA_SYNC_N 可以长期置零。其他信号可以参考表 9-2 来实现。

表 9-2: VGA 参考代码

```

1 module vga_ctrl(
2     input                pclk,        //25MHz 时钟
3     input                reset,       //置位
4     input  [23:0]        vga_data,    //上层模块提供的VGA颜色数据
5     output [9:0]         h_addr,      //提供给上层模块的当前扫描像素点坐标
6     output [9:0]         v_addr,
7     output               hsync,      //行同步和列同步信号
8     output               vsync,
9     output               valid,      //消隐信号
10    output [7:0]         vga_r,      //红绿蓝颜色信号
11    output [7:0]         vga_g,
12    output [7:0]         vga_b
13 );
14
15 //640x480分辨率下的VGA参数设置
16 parameter    h_frontporch = 96;
17 parameter    h_active = 144;
18 parameter    h_backporch = 784;
19 parameter    h_total = 800;
20
21 parameter    v_frontporch = 2;
22 parameter    v_active = 35;
23 parameter    v_backporch = 515;
24 parameter    v_total = 525;
25
26 //像素计数值
27 reg [9:0]     x_cnt;
28 reg [9:0]     y_cnt;
29 wire         h_valid;
30 wire         v_valid;
31
32 always @(posedge reset or posedge pclk) //行像素计数
33     if (reset == 1'b1)

```

```
34         x_cnt <= 1;
35     else
36     begin
37         if (x_cnt == h_total)
38             x_cnt <= 1;
39         else
40             x_cnt <= x_cnt + 10'd1;
41     end
42
43     always @(posedge pclk) // 列像素计数
44         if (reset == 1'b1)
45             y_cnt <= 1;
46         else
47         begin
48             if (y_cnt == v_total & x_cnt == h_total)
49                 y_cnt <= 1;
50             else if (x_cnt == h_total)
51                 y_cnt <= y_cnt + 10'd1;
52         end
53     // 生成同步信号
54     assign hsync = (x_cnt > h_frontporch);
55     assign vsync = (y_cnt > v_frontporch);
56     // 生成消隐信号
57     assign h_valid = (x_cnt > h_active) & (x_cnt <= h_backporch);
58     assign v_valid = (y_cnt > v_active) & (y_cnt <= v_backporch);
59     assign valid = h_valid & v_valid;
60     // 计算当前有效像素坐标
61     assign h_addr = h_valid ? (x_cnt - 10'd144) : {10{1'b0}};
62     assign v_addr = v_valid ? (y_cnt - 10'd35) : {10{1'b0}};
63     // 设置输出的颜色值
64     assign vga_r = vga_data[23:16];
65     assign vga_g = vga_data[15:8];
66     assign vga_b = vga_data[7:0];
67 endmodule
```

此代码对外提供了 VGA 控制信号，代码输出的红 R、绿 G、蓝 B 三种颜色分别是以 vga_r,vga_g,vga_b 三个 8 位的二进制信号表示的，这三组 8 位数字

信号将被传送到开发板上的数模转换器，转换成模拟信号，经 VGA 接口送入显示器中。

该控制器的特点是可以方便地实现上层系统对显示内容的控制。例如，如果在模块调用时设置 `vga_data=24'hFF0000`，就可以直接显示全屏红色。上层系统也可以根据当前扫描的像素坐标，选择合适的颜色给不同的像素设置不同的 `vga_data`。更重要的是，上层系统可以分配一块显示存储，利用 `v_addr`, `h_addr` 来索引该显存，每次扫描到特定像素点时，按照显存的值来设置 `vga_data`。这样，其他应用就可以直接对显存进行操作，显存改变自动反应到 VGA 的显示上，而不用关心 VGA 扫描的具体过程了。

非常不幸的是，如果每个像素点用 3 个 8bit 数来表示，一个像素点需要 24bit，640×480 的像素点需要 7.372M bit 的 RAM。我们的 FPGA 只有 5.57M bit 片内内存，不够实现 24bit 颜色的 VGA 显存。可能的解决方案包括，i). 调用片外的 64M SDRAM（此方案过于复杂，不建议使用），ii). 只给 256×256 的像素范围分配显存，iii). 降低颜色分辨率至 12bit，即 RGB 各用 4bit 来表示，颜色数量变少（**建议方案**）。

9.3 实验内容

9.3.1 显示不同颜色条纹

在上述 VGA 控制器中，根据扫描的行或列数据，输出两种以上的不同颜色条纹（横条或竖条均可以）。

9.3.2 图片显示

利用上述控制器，在显示器上显示一张静态图片。

我们建议使用低比特的颜色显示的方式来绕过 RAM 不足的问题。当然有兴趣的同学可以通过其他方式来实现高分辨率的图像显示。


▣ 低比特颜色显示方案

- 显存分配大小为 640×512 word, 每个 word 为 12bit。用 `h_addr` 的全部 10 位和 `v_addr` 的低 9 位合成 19 位地址来索引显存。为方便寻址，我们给行 `v_addr` 分配了 512 行的空间。这样，可以不用对地址进行复杂的转换。此处只需要分配 327680 个连续的存储单元，不需要考虑 `h_addr` 大于 640 的情况。

- assign 红、绿、蓝颜色的时候，根据 12bit 显存数据中对应颜色的 4bit 值，设置输出 8bit 数据的高 4 位，低 4 位置零。
- 对显存用 .mif 文件初始化。可以自己用常用的语言生成 .mif 文件，我们也提供了一张 640×512 的 12bit 图片的 my_picture.mif 文件，其中每像素按 RGB 各 4 比特，地址按列排列，开头是第一列像素 512 个点，其中超过 480 行的像素置为白色。然后顺序排列 640 列像素。



需要注意的是，显存占用空间较大，实现时需要用时钟沿驱动的显存，这样系统可以用 BLOCK RAM(M10K) 来实现。当资源不够时，Quartus 可能会无法综合，耗费大量时间编译。

 **最低要求：**本实验有一定难度，最低完成要求是能够显示条纹（9.3.1节），仅满足最低要求无法获得全部分数。