



南京大學

《数字电路与数字系统实验》实验报告

实验九： VGA 接口控制器实现

姓名： 毛彦杰

学号： 191220081

班级： 数字电路与数字系统实验 2 班

院系： 计算机科学与技术系

邮箱： 1363818182@qq.com

实验时间： 2020/11/16

预习报告：

一、VGA 简介

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。关于模拟信号和数字信号的区别，请参考 (Analog Signal 及 Digital Signal)。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，现在仍有不少液晶显示器或投影仪还支持 VGA 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）。

预习任务

我们要先理解原理，才能开始实验，首先了解 VGA 的工作原理：我们使用的 VGA 的分辨率是 640×480 的分辨率的。然后要想屏幕看起来不闪烁需要刷新频率大于 24，从后面我们可以知道，我们

对 VGA 使用的频率是 25MHZ 的，这是通过一个分频模块，然后传入参数 2500 0000 用 50MHZ 的时钟实现的 25MHZ 的同步时钟，其分频模块如下： 传入参数 2500 0000 就得到了 25MHZ 的同步时钟，通过计算得知：在 25MHZ 的时钟周期下 总时长为 16.8 毫秒，所以对应大概约每秒 60 帧，大于 24 帧，人眼感受不到。

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

```

每一帧的图像都是从屏幕的左上方开始一行一行进行的，并且行同步信号是一个负脉冲， 当其有效的时候， RGB 端就会送出当前行显示的各像素点的 RGB 电压值，每一帧结束后都会由帧同步信号送出一个负脉冲，重新从屏幕的左上方开始进行扫描。

而且从 PDF 中我们了解到 RGB 端并不是所有时间都在传送像素信息，会因为电子束从一行尾到下一行头需要时间而出现行消隐时间，这时 RGB 送出的电压值为 0 (黑色)。并且行消隐 时间以像素为单位。

有效显示一行需要 800 个像素点的时间，其中每行显示 640 个像素点，其中行消隐时间为 160 个像素点。有效显示一帧图像需要 525 行时间，一帧的消隐时间为 40 行。

对于下面的 VGA 控制信号模块：

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,       //复位  
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)
```

主要作用：在顶层模块里面传入该模块需要的参数，然后利用其中的行和列的扫描坐标，在顶层文件里面修改传入的 vga_data，然后在该模块里面用改变的 vga_data 来改变相应的 vga_r、vga_g 以及 vga_b，从而实现对屏幕颜色的改变。

实验报告：

9.3.1 显示不同颜色的条纹:输出两种以上不同颜色条纹

9.3.2 图片显示：显示一张静态图片

一、实验目的

1. 本实验的目的是学习 VGA 接口原理.
2. VGA 接口控制器设计方法.
3. 能在代码出现 bug 的时候应用时序原理定位并修复 bug.

二、实验原理：显示控制原理

显示控制原理：

常见的彩色显示器一般由阴极射线管 (CRT) 构成, 彩色由 GRB (Green Red Blue) 基色组成。显示采用逐行扫描的方式解决, 阴极射线枪发出电子束打在涂有荧光粉的荧光屏上, 产生 GRB 基色, 合成一个彩色像素。扫描从屏幕的左上方开始, 从左到右, 从上到下, 逐行扫描, 每扫完二行, 电子束回到屏幕的左边下一行的起始位置。

在这期间, CRT 对电子束进行消隐, 每行结束时, 用行同步信号进行行同步; 扫描完所有行, 用场同步信号进行场同步, 并使扫描回到屏幕的左上方, 同时进行场消隐, 并预备进行下一次的扫描。



图 9-3: VGA 行扫描、场扫描时序示意图

由图 9-3 可知，在标准的 640×480 的 VGA 上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，然后每行显示 640 个像素点，最后行消隐前沿需要 16 个像素点的时间。所以一行中显示像素的时间为 640 个像素点时间，一行消隐时间为 160 个像素点时间。

在标准的 640×480 的 VGA 上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，然后每场显示 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。

因此，在 640×480 的 VGA 上的一幅图像需要 $525 \times 800=420k$ 个像素点的时间。而每秒扫描 60 帧共需要约 25M 个像素点的时间。

三、实验设备环境

硬件器材：FPGA 开发板、显示器

软件平台：Quartus 开发平台

四、实验步骤

9.3.1 显示不同颜色的条纹

设计思路：

通过上面的预习实验我已经基本了解了 VGA 的工作原理。

此次实验我们可以分多个模块来实现：

首先我们得设计一个顶层实体，该实体里面的参数是 VGA 需要的参数。然后用 PDF 上的 `clkgen` 模块，实例化出来一个 25MHZ 的同步时钟。

再用这个 25MHZ 的同步时钟去实例化 PDF 中的另一个模块，并且我们需要根据 PDF 中的 VGA 参考代码模块实例化出来的提供给上层用的像素点的坐标来进行对整个屏幕的颜色控制。

我们只需要用该像素点的扫描列的坐标，通过取模来分割色条。
可以得出如下设计代码：

设计代码：

顶层 exp_nine_vga.v:

```
exp_nine_vga.v
1 module exp_nine_vga(
2   clk, reset, vga_r, vga_g, vga_b, vga_sync_n,
3   hsync, vsync, valid, rst, clken, pclk
4 );
5   input clk; //时钟
6   input reset; //复位
7   output wire [7:0] vga_r;
8   output wire [7:0] vga_g;
9   output wire [7:0] vga_b;
10  output wire vga_sync_n;
11  output wire hsync; //行同步和列同步
12  output wire vsync;
13  output wire valid; //消隐信号
14  input rst;
15  input clken;
16  output wire pclk;
17  reg [23:0] vga_data; //上层模块提供的VGA颜色数据
18  wire [9:0] h_addr; //提供给上层模块的当前扫描像素点坐标
19  wire [9:0] v_addr;
20
21  always begin
22    if(h_addr % 120 < 14) vga_data = 24'hFF0000;
23    else if(h_addr % 120 < 27) vga_data = 24'h0000FF;
24    else vga_data = 24'h00FF00;
25  end
26  assign vga_sync_n = 0;
27  vga_ctrl vga_s(
28    .pclk(pclk), //25MHz时钟
29    .reset(1'b0), //复位
30    .vga_data(vga_data), //上层模块提供
31    .h_addr(h_addr), //提供给上层模块的当前扫描像素点坐标
32    .v_addr(v_addr),
33    .hsync(hsync), //行同步和列同步信号
34    .vsync(vsync),
35    .valid(valid), //消隐信号
36    .vga_r(vga_r), //红绿蓝颜色信号
37    .vga_g(vga_g),
38    .vga_b(vga_b)
39  );
40
41  clkgen #(25000000) clk_s(
42    .clk_in(clk),
43    .rst(1'b0),
44    .clken(clken),
45    .clk_out(pclk)
46  );
47
48  endmodule
49
50
```

正如我们在设计思路中所述的，用扫描的行坐标来进行对整个屏幕的划分，因为整个屏幕可以分为 480 行，所以我们前 160 行传给 vga_data \leftarrow 0xFF0000 显示红色，中间 160 行传给 vga_data \leftarrow 0x00FF00，显示绿色，最后 160 行传给 vga_data \leftarrow 0x0000FF，显示蓝色。

clkgen.v:

```
1 module clkgen(  
2     input clk_in,  
3     input rst,  
4     input clk_en,  
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 reg[31:0] clkcount;  
10 always @(posedge clk_in)  
11     if(rst)  
12     begin  
13         clkcount=0;  
14         clkout=1'b0;  
15     end  
16     else  
17     begin  
18         if(clk_en)  
19         begin  
20             clkcount=clkcount+1;  
21             if(clkcount>=countlimit)  
22             begin  
23                 clkcount=32'd0;  
24                 clkout=~clkout;  
25             end  
26             else  
27                 clkout=clkout;  
28             end  
29         else  
30         begin  
31             clkcount=clkcount;  
32             clkout=clkout;  
33         end  
34     end  
35 endmodule  
36  
37
```

vga_ctrl.v:

```
1 module vga_ctrl(  
2     input pclk, //25MHz 时钟  
3     input reset, //复位  
4     input [11:0] 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     end  
40 end
```

```

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[11:8], 1'b0, 1'b0, 1'b0, 1'b0};
65     assign vga_g = {vga_data[7:4], 1'b0, 1'b0, 1'b0, 1'b0};
66     assign vga_b = {vga_data[3:0], 1'b0, 1'b0, 1'b0, 1'b0};
67 endmodule

```

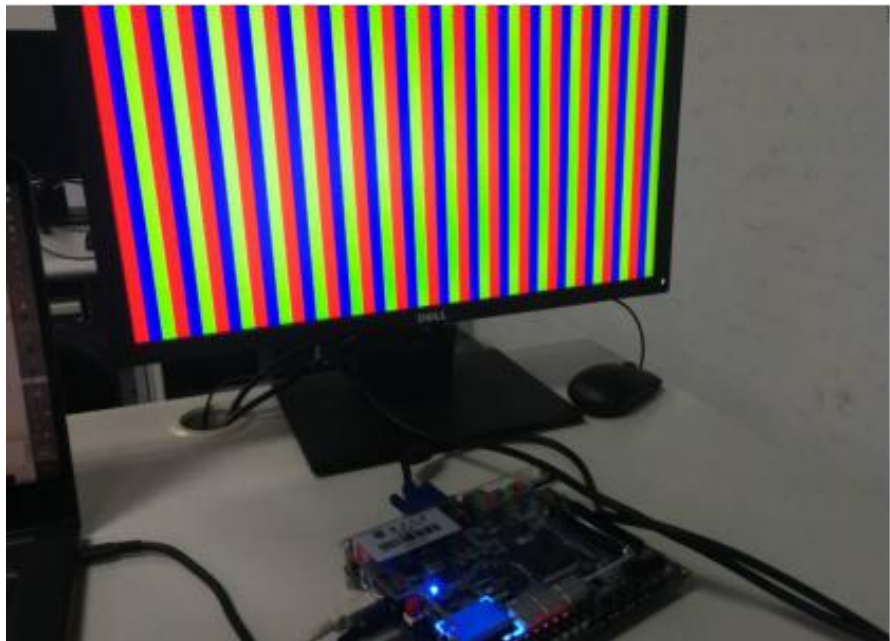
激励代码：将使能端全部置为有效即可测试分频模块

引脚分配：

Node Name	Direction	Location	I/O Bank	VREF Group
in clk	Input	PIN_AF14	3B	B3B_N0
in clken	Input	PIN_AB28	5B	B5B_N0
out hsync	Output	PIN_AK19	4A	B4A_N0
out pclk	Output	PIN_AK21	4A	B4A_N0
in reset	Input	PIN_AC29	5B	B5B_N0
in rst	Input	PIN_AB30	5B	B5B_N0
out valid	Output	PIN_AK22	4A	B4A_N0
out vga_b[7]	Output	PIN_AJ21	4A	B4A_N0
out vga_b[6]	Output	PIN_AJ20	4A	B4A_N0
out vga_b[5]	Output	PIN_AH20	4A	B4A_N0
out vga_b[4]	Output	PIN_AJ19	4A	B4A_N0
out vga_b[3]	Output	PIN_AH19	4A	B4A_N0
out vga_b[2]	Output	PIN_AJ17	4A	B4A_N0
out vga_b[1]	Output	PIN_AJ16	4A	B4A_N0
out vga_b[0]	Output	PIN_AK16	4A	B4A_N0
out vga_g[7]	Output	PIN_AK26	4A	B4A_N0
out vga_g[6]	Output	PIN_AJ25	4A	B4A_N0
out vga_g[5]	Output	PIN_AH25	4A	B4A_N0
out vga_g[4]	Output	PIN_AK24	4A	B4A_N0
out vga_g[3]	Output	PIN_AJ24	4A	B4A_N0
out vga_g[2]	Output	PIN_AH24	4A	B4A_N0
out vga_g[1]	Output	PIN_AK23	4A	B4A_N0
out vga_g[0]	Output	PIN_AH23	4A	B4A_N0
out vga_r[7]	Output	PIN_AK29	4A	B4A_N0
out vga_r[6]	Output	PIN_AK28	4A	B4A_N0
out vga_r[5]	Output	PIN_AK27	4A	B4A_N0

out	vga_r[4]	Output	PIN_AJ27	4A	B4A_N0
out	vga_r[3]	Output	PIN_AH27	4A	B4A_N0
out	vga_r[2]	Output	PIN_AF26	4A	B4A_N0
out	vga_r[1]	Output	PIN_AG26	4A	B4A_N0
out	vga_r[0]	Output	PIN_AJ26	4A	B4A_N0
out	vga_sync_n	Output	PIN_AJ22	4A	B4A_N0
out	vsync	Output	PIN_AK18	4A	B4A_N0

硬件实现截图



9.3.2 图片显示

设计思路：

按照 PDF 上面给出的提示我们首先需要用实现一个存储器，用来存储这个静态图片的信息，然后用 PDF 上面给出的.mif 文件来进行初始化，然后我们只是将核心部分的工作做了一下转换，以前是只要扫描行在对应的颜色区里面我们就设置相应的 vga_data，来显示相应的颜色。

现在也基本是类似的，不同的是我们首先需要用用一个变量来对扫描的行和列进行转换，也即将扫描到的位置映射到我们存储器相应的位置上去，然后在每个时钟上升沿到来时，根据存储器里面的内容对相应的 vga_data 进行改变，也即只改变高四位，低四位清零即可。就这样我们就能实现静态图片的显示了：

设计代码：

顶层 exp_nine_picture.v:

```
1 module exp_nine_picture(c1k,reset,VGA_R,VGA_G,VGA_B,VGA_HS,VGA_VS,VGA_CLK,VGA_SYNC_N,VGA_BLANK_N);
2     input c1k;
3     input reset;
4     output [7:0] VGA_R;
5     output [7:0] VGA_G;
6     output [7:0] VGA_B;
7     output VGA_HS,VGA_VS;
8     output VGA_CLK;
9     output VGA_BLANK_N;
10    output VGA_SYNC_N;
11    assign VGA_SYNC_N = 0;
12    c1kgen #(25000000) vgaclk(c1k,reset,1'b1,VGA_CLK);
13
14    wire [9:0] h_addr;
15    wire [9:0] v_addr;
16    wire [3:0] r,g,b;
17
18    ram ram(
19        .addr({h_addr,v_addr[8:0]}),
20        .c1k(VGA_CLK),
21        .r(r),
22        .g(g),
23        .b(b));
24
25    vga_ctrl VGA(
26        .pclk(VGA_CLK), //25MHz 时钟
27        .reset(reset) //复位
28        .vga_data({r,4'b0000,g,4'b0000,b,4'b0000}), //上层模块提供的VGA颜色数据
29        .h_addr(h_addr), //提供给上层模块的当前扫描像素点坐标
30        .v_addr(v_addr),
31        .hsync(VGA_HS), //行同步和列同步信号
32        .vsync(VGA_VS),
33        .valld(VGA_BLANK_N), //消隐信号
34        .vga_r(VGA_R), //红绿蓝颜色信号
35        .vga_g(VGA_G),
36        .vga_b(VGA_B)
37    );
38
39
```


从存储器中取出相应的存储内容，然后改变 vga_data，并且将 vga_data 的低四位清零，然后将改变的 vga_data 传入 PDF 提供的 VGA 参考代码里面就可以相应的改变传给 VGA 的 vga_r、vga_g 以及 vga_b 了。

clkgen.v:

```

1 module clkgen(
2     input clk_in,
3     input rst,
4     input clk_en,
5     output reg clk_out
6 );
7     parameter clk_freq=1000;
8     parameter countlimit=50000000/2/clk_freq; // 自动计算计数次数
9     reg[31:0] clkcount;
10    always @ (posedge clk_in)
11    if(rst)
12    begin
13        clkcount=0;
14        clk_out=1'b0;
15    end
16    else
17    begin
18        if(clk_en)
19        begin
20            clkcount=clkcount+1;
21            if(clkcount>=countlimit)
22            begin
23                clkcount=32'd0;
24                clk_out=~clk_out;
25            end
26            else
27                clk_out=clk_out;
28            end
29        else
30        begin
31            clkcount=clkcount;
32            clk_out=clk_out;
33        end
34    end
35 endmodule
36
37

```

vga_ctrl.v:

```

1 module vga_ctrl(
2     input pclk, //25MHz 时钟
3     input reset, //复位
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     end
40
41

```

```

39         else
40             x_cnt <= x_cnt + 10'd1;
41         end
42     end
43     always @(posedge clk) // 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'd145) : {10{1'b0}};
62         assign v_addr = v_valid ? (y_cnt - 10'd36) : {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
68
69
70
71
72

```
































ram. v:

```

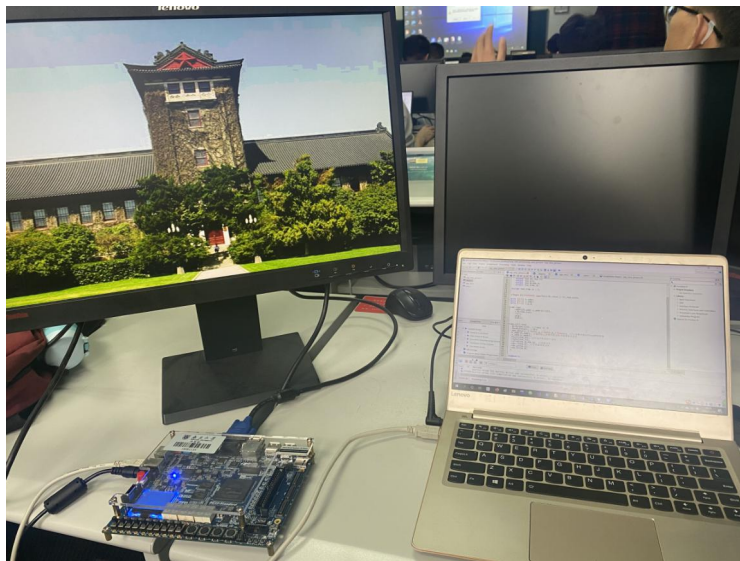
1  module ram(addr, clk,r,g,b);
2      input [18:0] addr;
3      input clk;
4      output [3:0] r,g,b;
5
6      (* ram_init_file = "pic.mif" *) reg [11:0] ram [327679:0];
7      reg [11:0] tmp;
8
9
10     always @(posedge clk)
11     begin
12         tmp <= ram[addr];
13     end
14
15     assign r = tmp[11:8];
16     assign g = tmp[7:4];
17     assign b = tmp[3:0];
18 endmodule
19

```

引脚分配：

Node Name	Direction	Location	I/O Bank	VREF Group
 VGA_B[7]	Output	PIN_AK16	4A	B4A_NO
 VGA_B[6]	Output	PIN_AJ16	4A	B4A_NO
 VGA_B[5]	Output	PIN_AJ17	4A	B4A_NO
 VGA_B[4]	Output	PIN_AH19	4A	B4A_NO
 VGA_B[3]	Output	PIN_AJ19	4A	B4A_NO
 VGA_B[2]	Output	PIN_AH20	4A	B4A_NO
 VGA_B[1]	Output	PIN_AJ20	4A	B4A_NO
 VGA_B[0]	Output	PIN_AJ21	4A	B4A_NO
 VGA_BLANK_N	Output	PIN_AK22	4A	B4A_NO
 VGA_CLK	Output	PIN_AK21	4A	B4A_NO
 VGA_G[7]	Output	PIN_AH23	4A	B4A_NO
 VGA_G[6]	Output	PIN_AK23	4A	B4A_NO
 VGA_G[5]	Output	PIN_AH24	4A	B4A_NO
 VGA_G[4]	Output	PIN_AJ24	4A	B4A_NO
 VGA_G[3]	Output	PIN_AK24	4A	B4A_NO
 VGA_G[2]	Output	PIN_AH25	4A	B4A_NO
 VGA_G[1]	Output	PIN_AJ25	4A	B4A_NO
 VGA_G[0]	Output	PIN_AK26	4A	B4A_NO
 VGA_HS	Output	PIN_AK19	4A	B4A_NO
 VGA_R[7]	Output	PIN_AJ26	4A	B4A_NO
 VGA_R[6]	Output	PIN_AG26	4A	B4A_NO
 VGA_R[5]	Output	PIN_AF26	4A	B4A_NO
 VGA_R[4]	Output	PIN_AH27	4A	B4A_NO
 VGA_R[3]	Output	PIN_AJ27	4A	B4A_NO
 VGA_R[2]	Output	PIN_AK27	4A	B4A_NO
 VGA_R[1]	Output	PIN_AK28	4A	B4A_NO
 VGA_R[0]	Output	PIN_AK29	4A	B4A_NO
 VGA_SYNC_N	Output	PIN_AJ22	4A	B4A_NO
 VGA_VS	Output	PIN_AK18	4A	B4A_NO
 clk	Input	PIN_AF14	3B	B3B_NO
 reset	Input	PIN_AB30	5B	B5B_NO

硬件实现截图



五：实验过程中遇到的问题及解决

1. 开始条纹显示很暗，并且有些色块没有显示出来。后来发现应该是 pdf 代码的复制错误，报了很多警告但是可以过，后来用 Adobe 再复制一下就 ok 了。

2. 忘记将 `vga_data` 的低四位清零

3. 引脚分配时分配错了，误将 `reset` 按钮分配到了 KEY 中，后来重新检查引脚，将 `reset` 按钮分配到 SW 开关中就好了。看来引脚分配时还需要我更加细心。

4. 本次实验本设想实现多张图片的显示，用更多的开关去实现不同图片的转换，从而达到类似放映 PPT 的效果，在编译时卡住，百思不得其解，仔细阅读手册后发现了以下文字：

注意：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。

于是减少到一张图片去实现。

六、实验得到的启示

这次实验的难度又比之前的实验上升了许多，更具有挑战性了。

面对更大规模的项目，更需要我们坚持先规划好各个模块的大框架，再专门实现每一个小功能。自顶向下，逐步求精。并且在实现具体的小功能的时候考虑好使用的变量类型/宽度/个数，语句类型/函数/任务等等问题，用恰当的语句来实现适合的功能，构建数字电路建模的 first principle 思考体系。

机器总是对的。对于 Quartus 而言它自身不具备特别好的调试工具，所以应该巧妙运用更多自己的调试方法来对一个项目进行分模块的调试。

七、意见与建议

实验手册中的实例对学习 verilog 语句使用与 quartus 相关操作有巨大帮助,希望能在学习新知识的同时接触更多相关例子来加深对新知的理解与应用能力。

希望自己能加强对网上知识的搜索自学能力,这样可以迅速掌握新的知识。
非常感谢老师和主教哥哥们的指点!! 谢谢!