实验报告

自选实验：

VGA 卡牌小游戏

王 谱越 516030910462

2018-12-29

目录

一 实验目的：

1. 理解 VGA 接口的原理，掌握利用 FPGA 进行 VGA 显示的流程。
2. 深刻理解 FPGA 并行化编程原理，编写一个简易小游戏

二 实验内容：

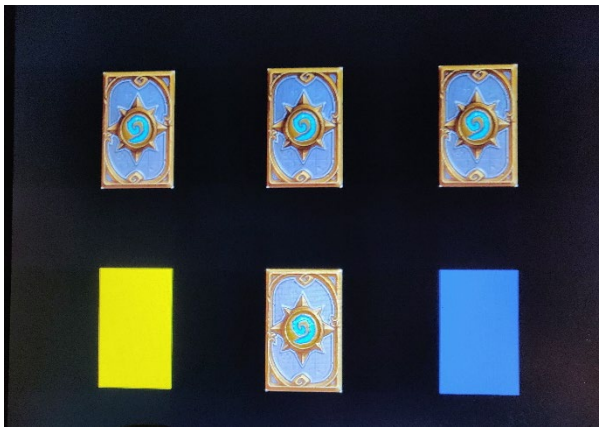
1. 采用 Verilog 在 quartus II 中实现基本的 VGA 显示功能。
2. 利用 ROM 完成静态图像在 VGA 上的显示功能。
3. 利用并行控制流，控制图片显示，完成简易翻牌小游戏。

三 实验器材：

- 硬件： DE1-SoC 实验板
- 软件： Altera Quartus II 13.1、 Altera ModelSim 10.1d

四 实验简介：

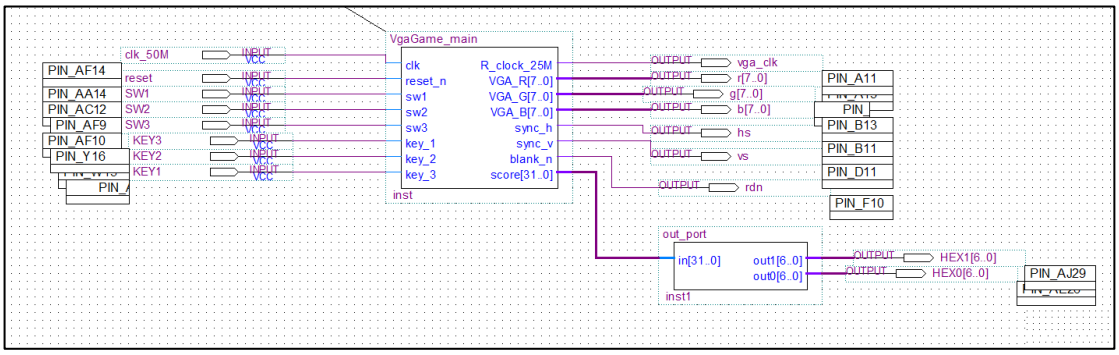
本次实验我利用 VGA 完成了一个简单的翻牌小游戏，一共有 6 张牌，通过按键翻转其中的 2 张，如果 2 张颜色相同则配对成功，每次翻牌步数将会加 1，需要玩家在最少的步数里完成所有配对。



五 实验过程：

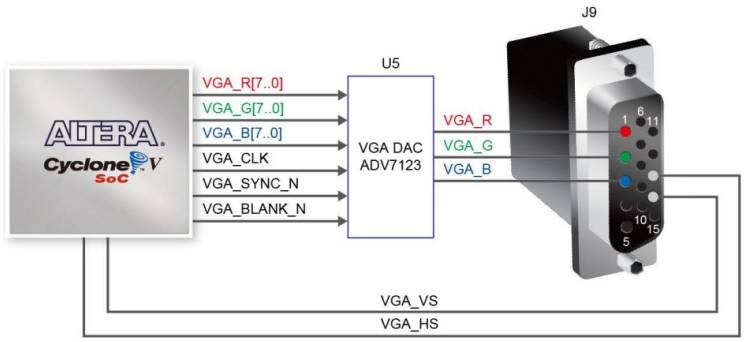
1. **模块设计：**

1.1 顶层模块

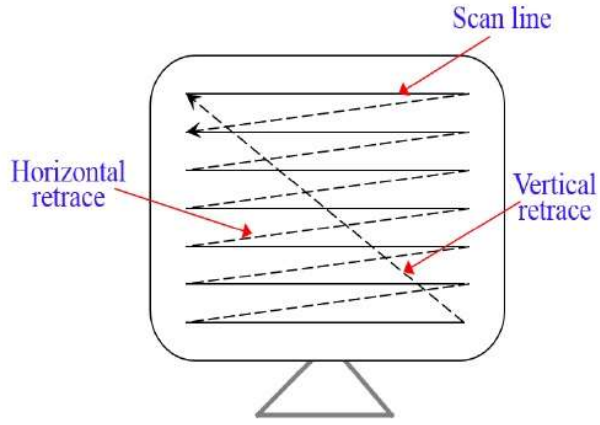


1.2 VGA 显示模块

VGA 与 FPGA 的连接需要提供以下信号：VGAR/G/B[7..0],VGA_CLK,VGA_SYNC_N,VGA_BLANK_N,VGA_VS,VGA_HS. 如下图：



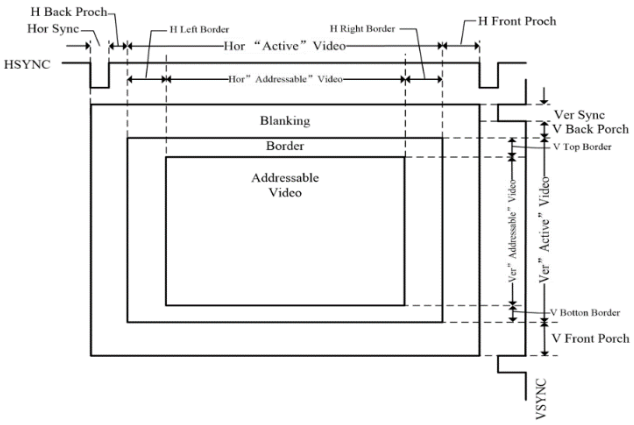
VGA 显示器扫描方式从屏幕左上角一点开始，从左向右逐点扫描，每扫描完一行,电子束回到屏幕的左边下一行的起始位置，其扫描示意图如下图所示：



Raster scanning terminology for a single output frame.

VGA 在从上一行尾到下一行头的期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，同时进行场消隐,开始下一帧。完成

一行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂直扫描时间，其倒数称为场频率。VGA 行扫描、场扫描时序如图：



对于不同的 VGA 分辨率，由于扫描行数和列数的不同，需要的时钟频率也不同，对于 640*480 的显示器来说，需要的时钟频率为 25MHZ 而 FPGA 的时钟频率为 50MHZ，因此需要对时钟进行 2 分频：

```
always @(posedge clk or negedge reset_n)begin
    if(!reset_n)
        R_clock_25M    <=  1'b0         ;
    else
        R_clock_25M    <=  ~R_clock_25M  ;
end
```

而 VGA 模块则负责产生行同步信号，列同步信号以及有效区域信号。同时将当前扫描到的行和列输出，用于其他模块协调，为相应位置提供正确 RGB 值。

```
module vga640x480 (
    input vga_clk,
    input reset,
    output vga_h_sync,
    output vga_v_sync,
    output inDisplayArea,
    output  reg[9:0] col_i,
    output  reg[9:0] row_i
);
```

1.3 图片读取模块

为了让图片能够让 FPGA 成功读取，需要将图片转化为 mif,这个工作通过 Python 脚本完成,脚本需要将图片从上到下按行将每个像素值填写到 MIF 文件中。

得到 MIF 文件后，为了能够在特定位置显示正确的图片，需要对显示器坐标和图片相对坐标进行转化：

```
module region_detect (pixel_r, pixel_c, target_r, target_c, in_region, rel_r, rel_c);
    input [9:0] pixel_r, pixel_c, target_r, target_c;

    output       in_region;
    output [7:0] rel_r, rel_c;
    // 表示当前像素是否是该图片需要显示区域。 `core_h 图片水平宽度 `core_v 图片纵向宽度
    assign in_region =(pixel_r>target_r)&&(pixel_r<target_r+`core_v)&&(pixel_c>target_c)&&(pixel_c<target_c+`core_h);
    // 相对于图片坐标
    assign rel_r = pixel_r - target_r;
    assign rel_c = pixel_c - target_c;
endmodule
```

之后将相应的内存地址传递到 ROM 模块即可读取到像素数据：

```
assign index = rel_r*`core_h+rel_c;
imgrom_back  back_imgshow(index,clk,{back_r, back_g, back_b});
```

1.4 游戏控制模块

游戏控制模块，记录了几个变量用于整个游戏的控制：

```
//game control
reg [31:0]score;
reg [2:0] cur_card;
reg [2:0] last_card,saved_card1,saved_card2,saved_card3,saved_card4;
wire[2:0] last_color,cur_color;
```

模块中会记录当前翻开的卡牌和上一次翻开的卡牌，这些信号在卡牌展示模块中转化为 6 张卡牌的翻开与否的状态。

```
module card_show(first_card,second_card,saved_card1,saved_card2,saved_card3,saved_card4,showlist);
   input [2:0]first_card;
   input [2:0]second_card,saved_card1,saved_card2,saved_card3,saved_card4;
   output [5:0]showlist;
   reg [5:0] showlist_1,showlist_2,showlist_3,showlist_4,showlist_5,showlist_6,showlist;
   always @ * begin

      case(first_card)
         3'd0:showlist_1<= 6'd0;
         3'd1:showlist_1<=6'b000001;
         3'd2:showlist_1<=6'b000010;
         3'd3:showlist_1<=6'b000100;
         3'd4:showlist_1<=6'b001000;
         3'd5:showlist_1<=6'b010000;
         3'd6:showlist_1<=6'b100000;
      endcase
```

模块记录的当前翻开的卡牌的颜色以及上一次翻开的卡牌的颜色，根据颜色是否相同来判断是否一致，卡牌的颜色利用随机数生成，并记录在一个卡牌信息模块中。

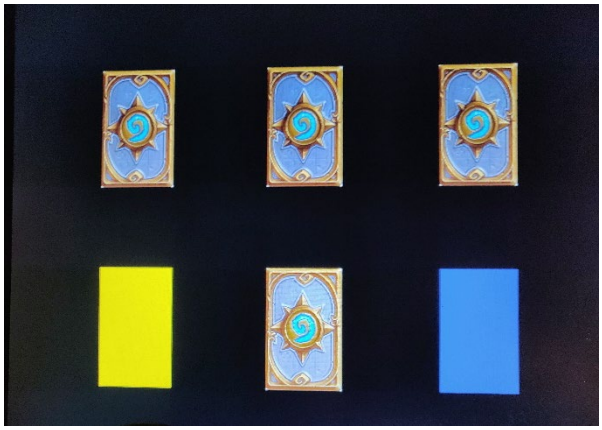
```
module card_info(first_card,rand_k,card_color);
   input [2:0]first_card;
   input [7:0] rand_k;
   output [2:0] card_color;
   reg [2:0] card_color;
   always @ * begin
      case(first_card)
         3'd0:card_color<= 3'd3;
         3'd1:card_color<=(1+rand_k)%6%3;
         3'd2:card_color<=(5+rand_k)%6%3;
         3'd3:card_color<=(2+rand_k)%6%3;
         3'd4:card_color<=(4+rand_k)%6%3;
         3'd5:card_color<=(3+rand_k)%6%3;
         3'd6:card_color<=(6+rand_k)%6%3;
      endcase
   end
endmodule
```

每次按键翻开一张牌，当翻开两张牌后，若两张牌相同，则将这两张牌储存到 saved_card 中表示已经翻开并配对。同时每次翻牌将会将当前的操作步数加 1。所有牌都配对翻开后，游戏结束。

六 实验结果与感想：

1. 实验结果

经过仿真验证以及实测，由于 FPGA 开发板的内存限制，只能加载两张图片，无法满足游戏需要，因此只能改为简单的纯色卡牌。以下为实验效果图：



2. 遇到的问题

- **随机数生成器：**

  由于游戏需要随机数来生成不同的游戏场景，因此我需要设计一个随机数生成器，经过了解，我选用了 LFSR 的方法来进行随机数生成，LFSR 利用移位寄存器来生成每个时钟周期都不同的随机数。

```
module RanGen(
    input               rst_n,    /*rst_n is necessary to prevet locking up*/
    input               clk,      /*clock signal*/
    output reg [7:0]    rand_num  /*random number output*/
);


always@(posedge clk or negedge rst_n)
begin
    if(!rst_n)
        rand_num    <=8'd3;
    else
        begin
            rand_num[0] <= rand_num[7];
            rand_num[1] <= rand_num[0];
            rand_num[2] <= rand_num[1];
            rand_num[3] <= rand_num[2];
            rand_num[4] <= rand_num[3]^rand_num[7];
            rand_num[5] <= rand_num[4]^rand_num[7];
            rand_num[6] <= rand_num[5]^rand_num[7];
            rand_num[7] <= rand_num[6];
        end

end
endmodule
```

3. 感想与反思

在本次实验中，我成功利用 FPGA 在 VGA 上显示出了预期图像，完成了简单的翻牌游戏，但比较可惜的是由于板子的内存限制，无法加载超过 2 张的图片，多次调整编程方式后还是没法加载更多的图片，因此很遗憾没能完成最终想要完成的效果，这也是让我第一次体会到了硬件限制对编程的影响，对我的开发思维有了很大的提醒和扩展。