

復旦大學



专用集成电路设计方法 课程 Project 文档

李琛 13307130163

赵春琪 13307130305

2016 年 6 月

演示视频: <https://youtu.be/dZ-EX1f50P0>

1 目录

1. 设计规划.....	3
1.1 设计要求.....	3
1.2 设计思路.....	3
1.2.1 相关参数.....	3
1.2.2 系统输入输出.....	3
1.2.3 系统功能.....	4
1.3 设计流程图.....	5
1.4 编程工具.....	8
1.4.1 硬件描述语言-Verilog HDL.....	8
1.4.2 软件开发平台.....	8
1.4.3 仿真工具——Modelsim.....	9
1.4.4 硬件开发平台——JFM4VSX55.....	9
2 设计实现.....	9
2.1 框图介绍.....	9
2.1.1 游戏流程.....	9
2.1.2 游戏界面.....	11
2.1.3 源文件与模块对照表.....	12
2.2 各模块设计与验证.....	14
2.2.1 main_module.v.....	14
2.2.2 div_clk.v.....	15
2.2.3 automaton.v.....	15
2.2.4 debouncer.v.....	19
2.2.5 loading_happen.v.....	21
2.2.6 square_gen.v.....	22
2.2.7 v_speed_debouncer.v.....	23
2.2.8 game_display.v.....	23
2.2.9 display_border.v.....	25
2.2.10 display_little_square.v.....	25
2.2.11 display_moving_square.v.....	26
2.2.12 display_next_square.v.....	27
2.3 综合与实现.....	28
2.3.1 管脚约束文件(ucf).....	28
2.3.2 综合结果.....	30
2.3.3 Warning 分析.....	30
2.3.4 综合报告.....	31
2.3.5 资源占用.....	32
2.3.6 时序分析.....	33
3 设计总结.....	34
3.1 注意事项与编程技巧.....	34
3.1.1 VGA 时序分析.....	34
3.1.2 图片的存储.....	37
3.1.3 Verilog 有限状态机三段式描述方法.....	39

3.2	体会心得.....	40
3.3	总结.....	41
4	附录：主要文件列表.....	42
4.1	Sources.....	42
4.2	Images.....	43
4.2.1	数字.....	43
4.2.2	文字.....	43
4.2.3	界面.....	43

1.设计规划

1.1 设计要求

使用 JFM4VSX55 芯片与硬件描述语言 (HDL) 设计一个俄罗斯方块游戏机。游戏界面通过 VGA 输出在显示屏上 通过 FPGA 开发板上的按钮控制方块位置与下落。

具体的显示分辨率与扩展游戏功能可自行添加。

1.2 设计思路

1.2.1 相关参数

本次课程作业设计思路大体与设计要求相符，具体参数如下：

显示器分辨率	640 * 480
硬件开发平台	JFM4VSX55
软件开发平台	Xilinx ISE 14.7 && Procise

1.2.2 系统输入输出

系统的输入输出如下：

1.2.2.1 输入：

时钟	Sys_clk
左移按键	Left
右移按键	right
下落按键	Speed
旋转按键	Rotate
复位信号	rst_n

1.2.2.2 输出：

VGA 时钟	VGA_CLK
VGA 水平同步信号	VGA_HSYNC
VGA 垂直同步信号	VGA_VSYNC
VGA 消隐信号	VGA_BLANK_N
输出 RGB 信号	VGA_R
	VGA_G
	VGA_B

1.2.3 系统功能

系统需要实现的功能如下：

1.2.3.1 按键的输入

- a. 需要能接收来自按键的输入
- b. 需要保证按键的稳定，要进行防抖动处理

1.2.3.2 游戏的显示

- a. 需要将输出的界面正确转化为 640*480 界面上的 RGB 信号。
- b. 通过 VGA 输出的 RGB 信号需要保证 VGA 时序正确。

1.2.3.3 图片的存储

- a. 需要将待输出的文件存放在开发板上。
- b. 存储的图像需要能正确输出，失真尽可能小。









1.2.3.4 游戏的流程

- a. 小方块的产生
- b. 当前方块的下落

- c. 堆积方块的显示
- d. 消行的判断
- e. 得分与游戏结束的判断

1.2.3.5 游戏规则

游戏规则设计如下：

-  游戏界面为一个用于摆放小型正方形的平面虚拟场地
-  一组由4个小型正方形组成的规则图形，英文称为 Tetromino，中文通称为方块共有7种，分别以S、Z、L、J、I、O、T这7个字母的形状来命名。
-  玩家可以做的操作有：以90度为单位旋转方块，以格子为单位左右移动方块，让方块加速落下。本游戏中，设置为当下落键按下时，方块以四倍速下落。
-  方块移到区域最下方或是着地到其他方块上无法移动时，就会固定在该处，而新的方块出现在区域上方开始落下。
-  当区域中某一列横向格子全部由方块填满，则该列会消失并成为玩家的得分。同时删除的列数越多，得分指数上升。
-  游戏会在左上角提示下一个要落下的方块。
-  为了提高游戏性，随着游戏的进行而加速提高难度。本游戏中，设置为一旦得分，则游戏进入SPEEDUP模式，速度变为原来两倍，游戏界面也会有相应变化。
-  当固定的方块堆到区域最上方而无法消除层数时，则游戏结束。

下面就具体的游戏实现进行说明。

1.3 设计流程图

本游戏设计遵循 TOP-DOWN 的设计模式。

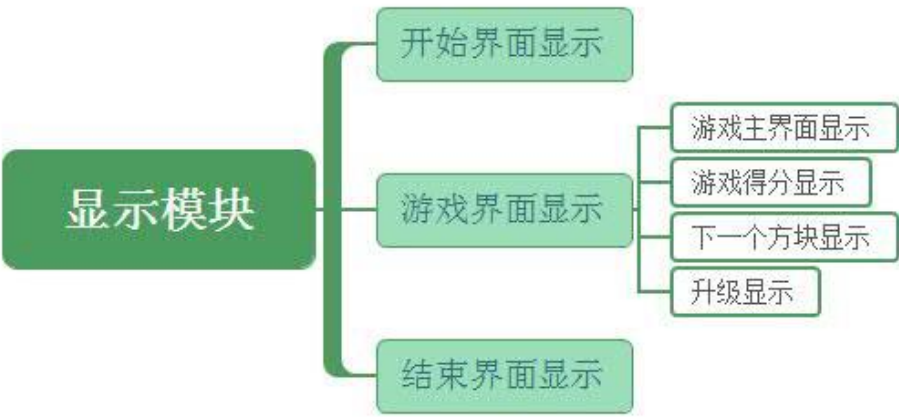
1. 先对进行系统级建模，确定顶层模块的输入输出。根据要求和上文的讨论，确

定输入信号为时钟、左移按键、右移按键、下落按键、旋转按键和复位信号，输出信号为 VGA 时钟、VGA 水平同步信号、VGA 垂直同步信号、VGA 消隐信号和输出 RGB 信号。

2. 划分功能块。根据设计的功能，我们可以将工程分为以下几个模块：

分频模块	对输入时钟进行分频
输入控制模块	读入按键信号并防抖动
显示模块	将待输出图像转换为 VGA 信号
游戏状态控制模块	确定当前是准备、游戏还是结束状态
游戏流程模块	核心模块，主游戏的算法模块
图像存储模块	存储待显示的图像

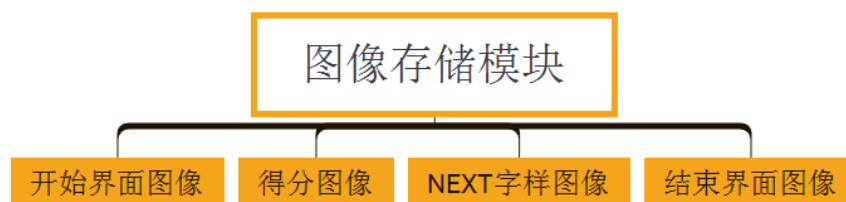
对于显示模块，针对不同的显示内容也需要划分不同的模块，细分如下：



游戏流程模块是核心模块，可以进一步细分：

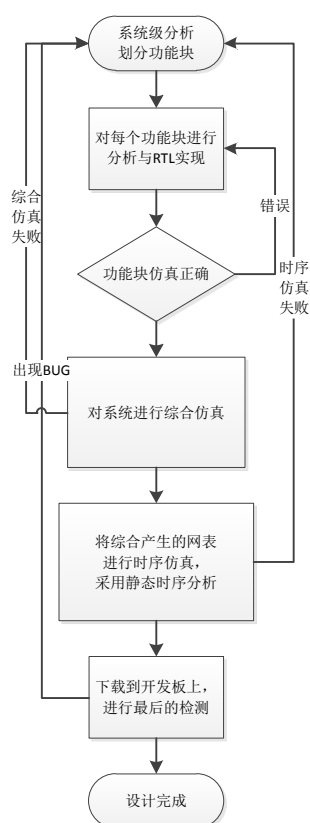


图像存储模块需要存储不同的模块，因此也需要进一步细分：



3. 之后再对每个功能块进行分析与 RTL 级实现。测试时先对每个功能块进行综合

与仿真，再对整个系统进行综合仿真。具体流程图如下：



1.4 编程工具

1.4.1 硬件描述语言-Verilog HDL

Verilog HDL 是一种硬件描述语言 (HDL: Hardware Description Language), 以文本形式来描述数字系统硬件的结构和行为的语言, 用它可以表示逻辑电路图、逻辑表达式, 还可以表示数字逻辑系统所完成的逻辑功能。 Verilog HDL 和 VHDL 是世界上最流行的两种硬件描述语言, 都是在 20 世纪 80 年代中期开发出来的。前者由 Gateway Design Automation 公司 (该公司于 1989 年被 Cadence 公司收购) 开发。两种 HDL 均为 IEEE 标准。

Verilog HDL 是一种硬件描述语言, 用于从算法级、门级到开关级的多种抽象设计层次的数字系统建模。被建模的数字系统对象的复杂性可以介于简单的门和完整的电子数字系统之间。数字系统能够按层次描述, 并可在相同描述中显式地进行时序建模。

1.4.2 软件开发平台

1.4.2.1 Xilinx ISE14.7

ISE 的全称为 Integrated Software Environment, 即

“集成软件环境”, 是 Xilinx 公司的硬件设计工具。

ISE 具有相对容易使用的、首屈一指的 PLD 设计环境,

将先进的技术与灵活性、易使用性的图形界面结合在一

起, 利于用户实现最佳的设计硬件设计。利用 Xilinx 公

司的 ISE 开发设计软件的工程设计流程, 具体分为五个步骤: 即输入 (Design Entry)、

综合 (Synthesis)、实现 (Implementation)、验证 (Verification)、下载 (Download)。

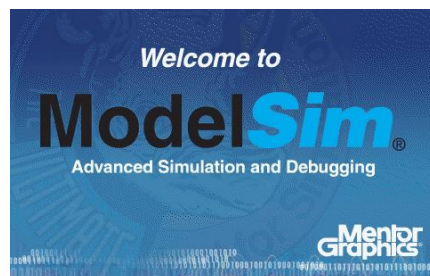
1.4.2.2 Procise



procise 为复旦微电子公司自主研发的 FPGA 布局布线工具，设计流程可覆盖地图、地点、路线、Bitgen、计划。procise 的设计目标：为与 Xilinx Virtex4 架构兼容的千万门级 FPGA 芯片提供的设计软件。软件支持的具体芯片型号包括 JFM4VSX55, JFM4VLX15, JFM4VSX35-RH, JFM4VSX55-RH。procise 的功能主要包括，完成电路的映射装箱、布局布线、时序分析与优化、位流生成、编程下载等。其存储空间不小于 1G；其主存容量不小于 4G；其外设的 USB 接口不少于两个。

1.4.3 仿真工具——Modelsim

Mentor 公司的 ModelSim 是业界最优秀的 HDL 语言仿真软件，它能提供友好的仿真环境，是业界唯一的单内核支持 VHDL 和 Verilog 混合仿真的仿真器。它采用直接优化的编译技术、Tcl/Tk 技术、和单



一内核仿真技术，编译仿真速度快，编译的代码与平台无关，便于保护 IP 核，个性化的图形界面和用户接口，为用户加快调错提供强有力的手段，是 FPGA/ASIC 设计的首选仿真软件。

1.4.4 硬件开发平台——JFM4VSX55

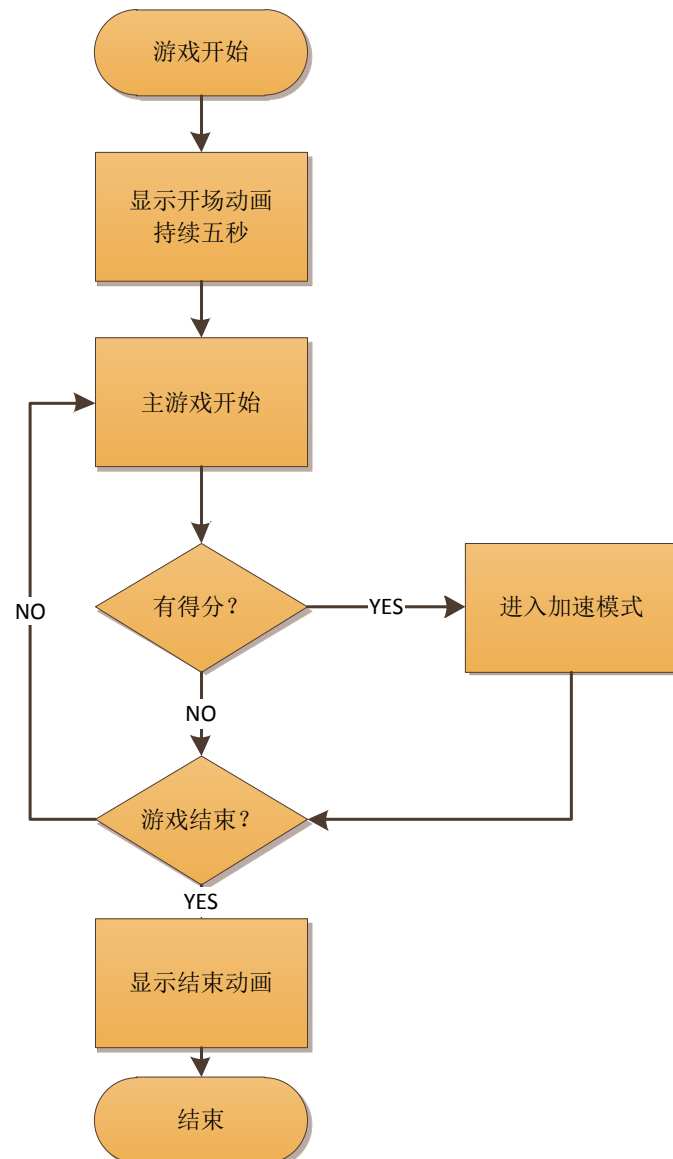


JFM4VSX55 是与 Xilinx 公司兼容的 FPGA 产品，包括宇航级 JFM4VSX55RT 和工业级 JFM4VSX55I，分别兼容于 Xilinx 公司的陶封产品 XQR4VSX55 和塑封产品 XC4VSX55。

2 设计实现

2.1 框图介绍

2.1.1 游戏流程



2.1.2 游戏界面

2.1.2.1 游戏开始界面

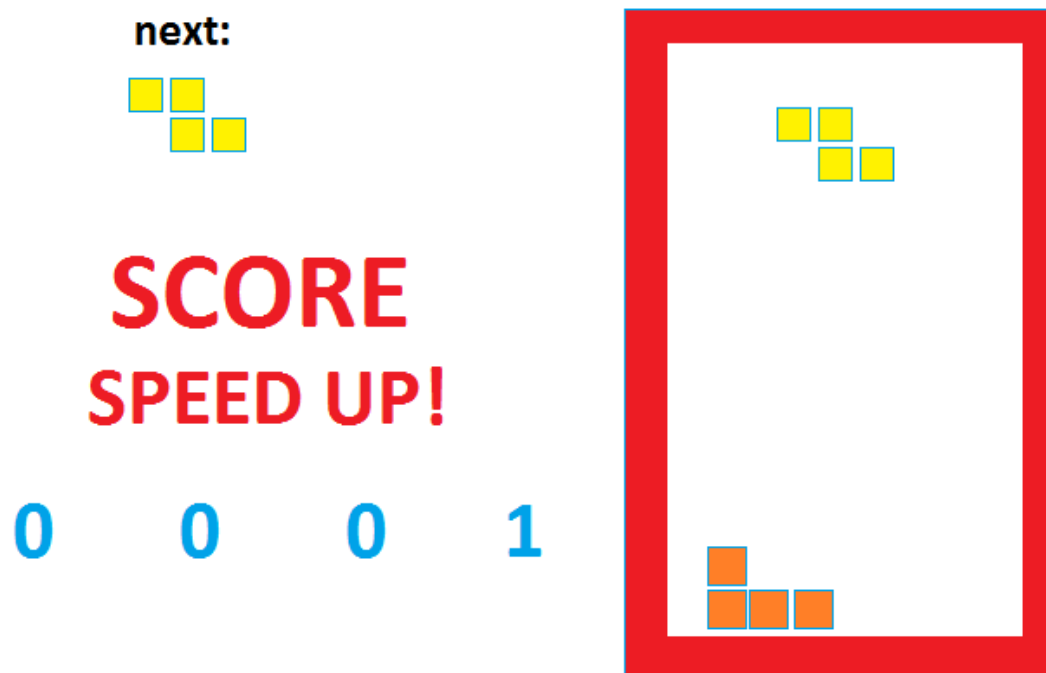
游戏开始时，这张图片从中间像两边展开，持续五秒。

TETRIS

俄罗斯方块

李琛 赵春琪

2.1.2.2 游戏界面



游戏界面分为左右两部分，左上角显示NEXT和下一个小方块，左侧中央为SCORE和SPEED UP!字样。默认情况下SPEED UP!被隐藏，如果进入加速模式则显示出来。左下角有四个数字，用四位十进制数表示当前得分。右边为俄罗斯方块界面，正在下落的方块与已固定的方块颜色是不同的。

2.1.2.3 游戏结束界面

游戏结束时，这张图片从上到下移入画面，覆盖原本的游戏界面。

小组成员：

李琛 赵春琪

THANK YOU!

謝謝觀看！

2.1.3 源文件与模块对照表

 <div>transform.m</div>	将图片转换为 COE 的脚本
 <div>main_module.v</div>	系统主模块
 <div>automaton.v</div>	系统状态机控制模块
 <div>debouncer.v</div>	防抖动模块
 <div>display_border.v</div>	显示游戏边界模块

 display_little_square.v	显示静止方块模块
 display_moving_square.v	显示下落方块模块
 display_next_square.v	显示下一个方块模块
 display_score.v	显示得分模块
 div_clk.v	系统分频模块
 game_display.v	游戏界面显示模块
 game_sync_module.v	游戏界面输出模块
 loading_happen.v	俄罗斯方块主算法模块
 over_sync_module.v	结束动画产生模块
 over_vga_control_module.v	结束动画输出模块
 show_next.v	NEXT 字样显示模块
 show_score.v	SCORE 字样显示模块
 square_gen.v	小方块产生模块
 start_sync_module.v	开始动画显示模块

 start_vga_control_module.v	开始动画输出模块
 v_speed_debouncer.v	下落按键防抖动模块
 vga_select_module.v	VGA 输出选择模块

2.2 各模块设计与验证

2.2.1 main_module.v

系统主模块，在这里，我们定义了整个模块的输入和输出引脚，并且将其他模块创建实例后连接起来形成整个系统。

下表为用到的输入输出口介绍：

input sys_clk	系统输入时钟
input rst_n	系统 reset 信号输入
input right	方块右移操作输入
input left	方块左移操作输入
input rotateR	方块顺时针旋转操作
input verticalspeed	方块快速下落的操作输入
output[7:0] VGA_G	VGA 中 R G B 信号的输出，决定了屏幕上颜色的显示
output[7:0] VGA_R	
output[7:0] VGA_B	
output VGA_CLK;	VGA 时钟输出
output VGA_VSYNC	VGA 横、纵座标的输出
output VGA_HSYNC	

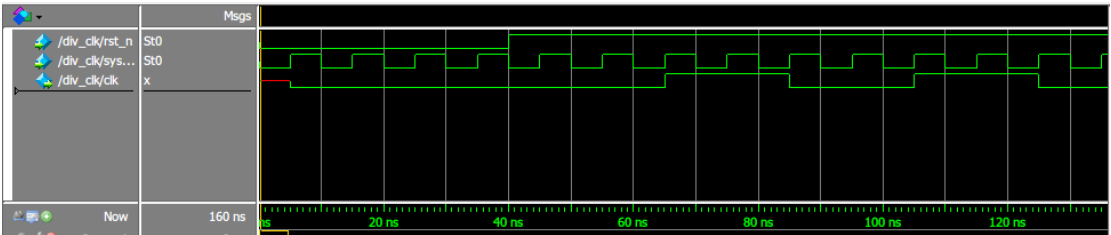
output VGA_PSVAE_N	VGA 显示中的使能型号
output VGA_SYNC_N	
output VGA_BLANK_N	

2.2.2 div_clk.v

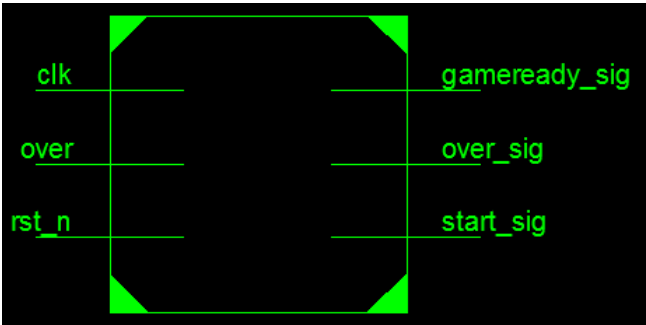
时钟分频模块，用来将 100MHz 的输入时钟分四分频为 25MHz 的时钟频率输出。

input sys_clk	输入系统时钟 100MHz
input rst_n	系统 reset 信号输入
output clk;	输出分频 25MHz

这个模块的实现，主要是靠定义一个寄存器 h，让其值在 0, 1, 2, 3 之间循环。h 为 0 和 1 时，输出低电平；h 为 1 和 2 时，输出高电平。以下为仿真波形，显示正常工作。



2.2.3 automaton.v



input clk	输入时钟
input rst_n	系统 reset 信号输入

input over	游戏结束信号输入
output start_sig	游戏开始信号输出
output gameready_sig	游戏准备信号输出
output over_sig	游戏结束信号输出

控制模块的状态机，对整个游戏的状态（准备游戏，进行游戏，游戏结束）三个状态进行处理判断。

```
        end
    else if(count_T5S == T5S)
        begin
            count_T5S <= 0;
            start <= 1;
        end
    else
        begin
            count_T5S <= 0;
            start <= 0;
        end
    end
end
end

/*****/

reg[2:0] game_current_process;
reg[2:0] game_next_process;
reg start_sig;
reg gameready_sig;
reg over_sig;

always @ ( posedge clk or negedge rst_n )
begin
    if( !rst_n )
        game_current_process <= ready;
    else
        game_current_process <= game_next_process;
    end

always @ ( game_current_process or start or over )
begin
    case( game_current_process )
    ready:
    begin
        ready_out;
        if( start )
            game_next_process = game;
        else
            game_next_process = ready;
    end
    game:
    begin
        game_out;
```

```
        if( over )
            game_next_process = game_over;
        else
            game_next_process = game;
        end
    game_over:
    begin
        over_out;
        game_next_process = game_over;
    end
    default:
    begin
        ready_out;
        game_next_process = ready;
    end
endcase
end

task ready_out;
{ gameready_sig, start_sig, over_sig } = 3'b100;
endtask

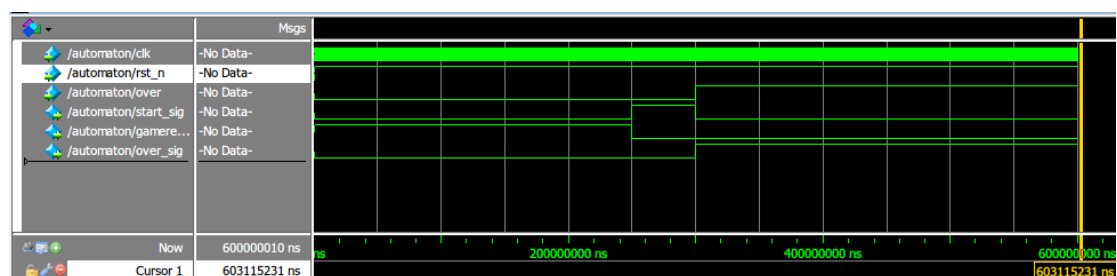
task game_out;
{ gameready_sig, start_sig, over_sig } = 3'b010;
endtask

task over_out;
{ gameready_sig, start_sig, over_sig } = 3'b001;
endtask

/***** /

endmodule
```

下面是仿真时序



2.2.4 debouncer.v

input clk	输入时钟
input rst_n	系统 reset 信号输入
input key_in	按键信号输入
output key_out	按键信号输出

用来对各个操作按键进行防抖动处理。抖动是由于按键的内置机械结构导致的不可避免问题。

```
`timescale 1ns / 1ps

module debouncer
  (clk, rst_n, key_in, key_out
  );
  input clk;
  input rst_n;
  input key_in;
  output key_out;

  /*****/

  reg key_in_dly1;
  reg key_in_dly2;

  always @ ( posedge clk or negedge rst_n )
    begin
      if( !rst_n )
        begin
          key_in_dly1 <= 1'b0;
          key_in_dly2 <= 1'b0;
        end
      else
        begin
          key_in_dly1 <= key_in;
          key_in_dly2 <= key_in_dly1;
        end
      end
    end

  /*****/

  reg[17:0] count_debouncer;
```

```
reg key_out_tmp;

always @ ( posedge clk or negedge rst_n )
begin
    if( !rst_n )
    begin
        count_debouncer <= 18'd260_000;
        key_out_tmp <= 1'b0;
    end
    else if(key_in_dly2)
    begin
        count_debouncer <= 18'd0;
        key_out_tmp <= 1'b0;
    end
    else if( count_debouncer == 18'd250_000 )
    begin
        key_out_tmp <= 1'b1;
    end
    else if( count_debouncer == 18'd260_000 )
    begin
        key_out_tmp <= 1'b0;
    end
    else
        count_debouncer <= count_debouncer + 1'b1;
    end

    /*****/

reg key_out_tmp_dly1;

always @ ( posedge clk or negedge rst_n )
begin
    if( !rst_n )
        key_out_tmp_dly1 <= 1'b0;
    else
        key_out_tmp_dly1 <= key_out_tmp;
    end

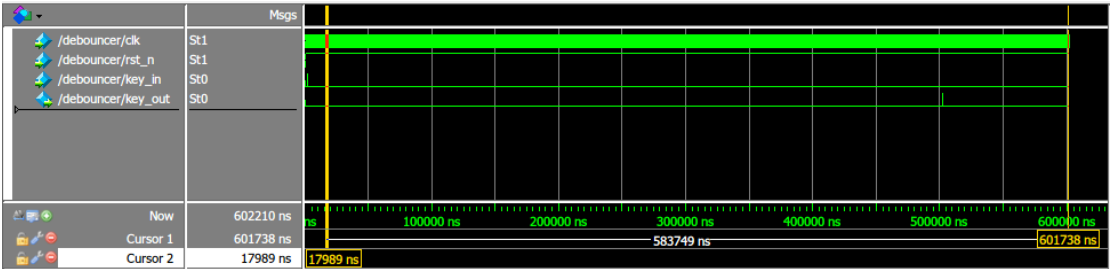
    /*****/

assign key_out = key_out_tmp & ( ~key_out_tmp_dly1 );

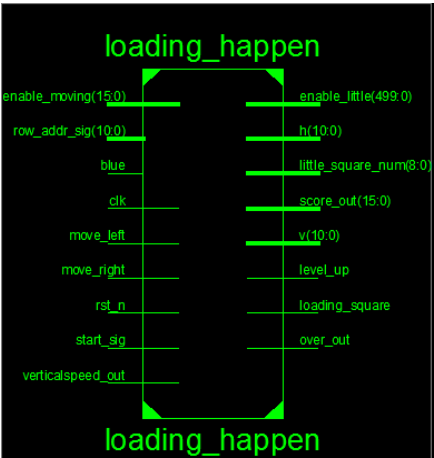
    /*****/
```

```
endmodule
```

仿真结果如下



2.2.5 loading_happen.v

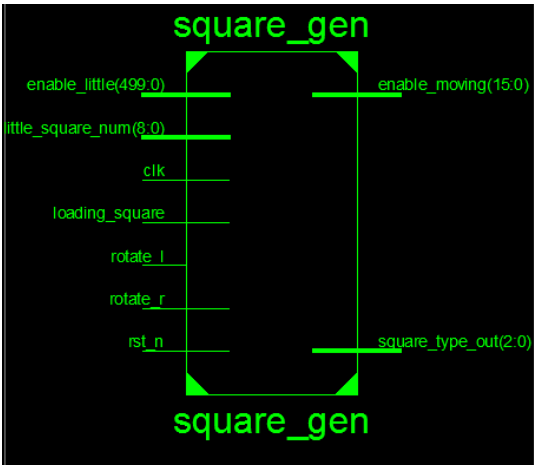


input clk	输入时钟
input rst_n	系统 reset 信号输入
input move_right	下落方块向右移动输入
input move_left	下落方块向左移动输入
input start_sig	游戏开始输入信号
input[15:0] enable_moving	正在下落方块形状
input verticalspeed_out	垂直下落加速输入
output[10:0] h	下落小方块水平像素位置
output[10:0] v	下落小方块垂直像素位置

output[499:0] enable_little	游戏界面存储寄存器
output loading_square	请求生成下一个小方块
output[8:0] little_square_num	正在下落方块座标
output over_out	输出游戏结束信号
output[15:0] score_out	计分结果输出
output level_up	输出提高难度级别信号

在这里我们实现一系列系统最终功能 ,如俄罗斯方块的基本消除功能 ,下落方块的下落计时 ,下落方块的位置提供 ,计分功能等等。

2.2.6 square_gen.v



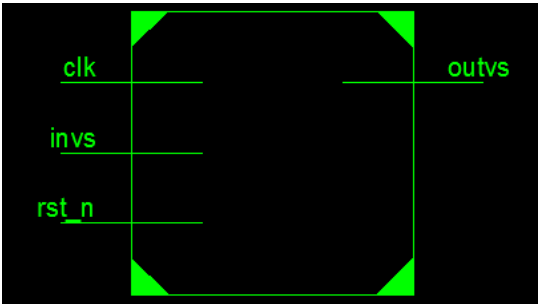
input clk	输入时钟
input rst_n	系统 reset 信号输入
input rotate_r	向右旋转按钮输入
input loading_square	加载下一方块按键输入
input[499:0] enable_little	游戏界面寄存器
input[8:0] little_square_num	正在下落的方块座标

output[15:0] enable_moving	正在下落的方块形状
----------------------------	-----------

output[2:0] square_type_out 下一个落下方块的形状

这个模块实现了两个功能，一是能够旋转方块并且对方块位置进行更新，二是能够通过生成伪随机数实现随机生成下一个需要的方块形状。

2.2.7 v_speed_debouncer.v



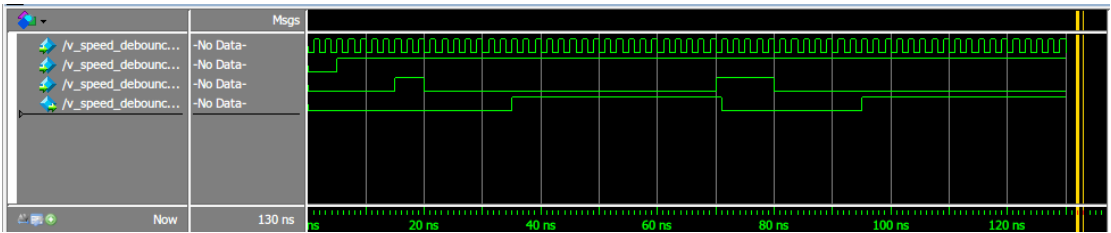
input clk	输入时钟
-----------	------

input rst_n 系统 reset 信号输入

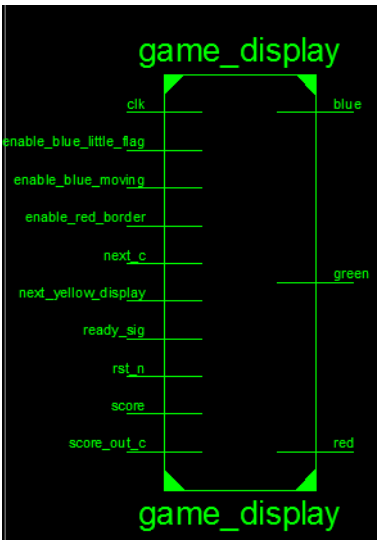
input invs	输入垂直方向下落加速信号
------------	--------------

output outvs 输出垂直方向下落加速信号

和之前防抖动类似的，这里对下落功能按键进行防抖动处理，仿真如下：



2.2.8 game_display.v



input clk	输入时钟
input rst_n	系统 reset 信号输入
input ready_sig	RGB 输出使能信号的输入
input enable_red_border	显示边界的使能信号
input enable_blue_moving	正在下落的小方块显示使能信号
input enable_blue_little_flag	静止小方块显示使能信号
input next_yellow_display	下一个小方块的显示使能信号
input next_c	NEXT 字符显示使能信号
input score_out_c	SCORE 字符显示使能信号
input score	分数显示使能信号
output red	R 信号输出
output green	G 信号输出
output blue	B 信号输出

这个模块实际上是根据内部的各模块状态决定输出 RGB 的模块。

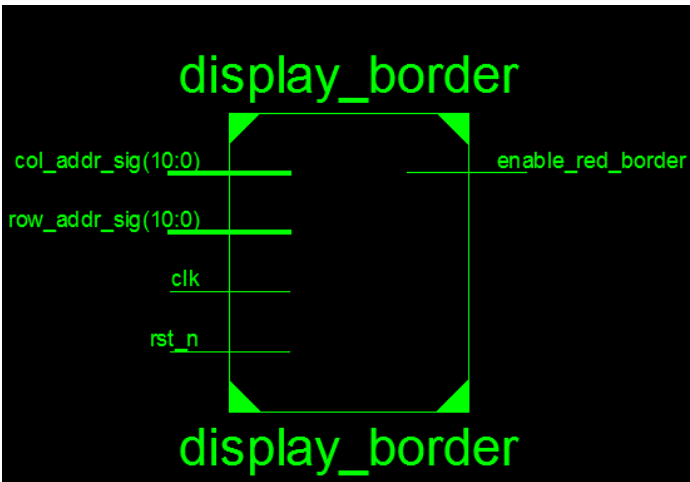
R、G、B 分别由几个信号“或”运算得到结果。

R：下一方块的显示、NEXT 四个字符、SCORE5 个字符。

G：分数、边界显示信号、下一方块的显示、静止小方块的显示信号、NEXT 四个字符的显示信号。

B：分数、静止小方块的显示信号、运动小方块的显示信号、NEXT 四个字符的显示信号。

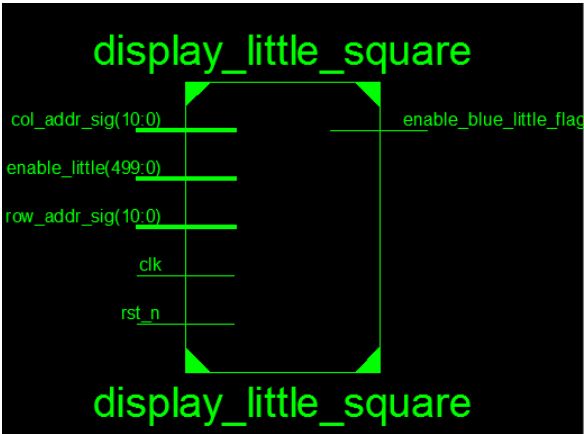
2.2.9 display_border.v



input clk	输入时钟
input rst_n	系统 reset 信号输入
input[10:0] col_addr_sig	输入的列地址信号
input[10:0] row_addr_sig	输入的行地址信号
output enable_red_border	是否作为边框显示

这个模块对屏幕进行扫描，在边框设定范围内的像素点输出为 1，和上面的 game_display 协同工作。

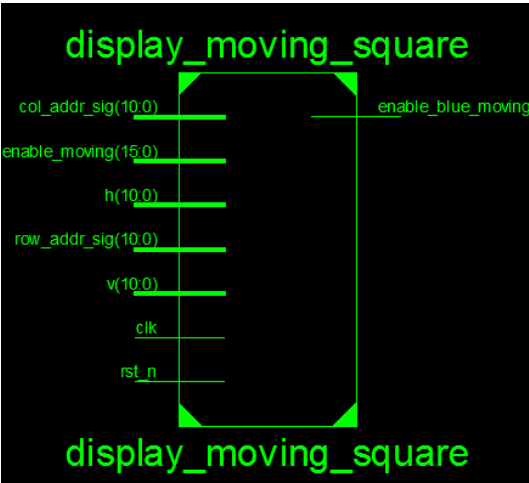
2.2.10 display_little_square.v



input clk	输入时钟
input rst_n	系统 reset 信号输入
input[10:0] col_addr_sig	输入的列地址信号
input[10:0] row_addr_sig	输入的行地址信号
input[499:0] enable_little	游戏区域的状态输入
output enable_blue_little_flag	输出静止小方块显示与否

这里和上面类似，输入整个游戏区域的小方块堆叠状态，并且进行扫描，如果存在小方块则输出 1，显示；否则不显示，输出 0。

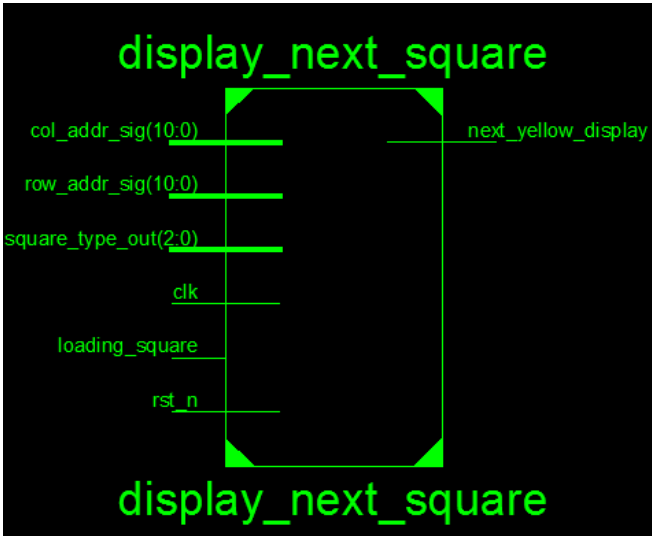
2.2.11 display_moving_square.v



input clk	输入时钟
input rst_n	系统 reset 信号输入
input[10:0] col_addr_sig	输入的列地址信号
input[10:0] row_addr_sig	输入的行地址信号
input[15:0] enable_moving	正在下落的小方块形状信号
input[10:0] h	输入的行像素坐标
input[10:0] v	输入的列像素坐标
output enable_blue_little_flag	输出静止小方块显示与否

同样类似上面，不同的是加入了 h、v 来协助判断。同样需要与 game_display 协同进行显示判断。

2.2.12 display_next_square.v



input clk	输入时钟
input rst_n	系统 reset 信号输入
input[10:0] col_addr_sig	输入的列地址信号
input[10:0] row_addr_sig	输入的行地址信号

input loading_square	是否载入下一个小方块的信号
input[2:0] square_type_out	输入随机生成的小方块的形状
output next_yellow_display	显示下一个小方块与否

收到 loading_square 的使能后读入下一个随机生成的小方块形状,并扫描相关像素点给出对应的使能信号,使得能够显示。

2.3 综合与实现

2.3.1 管脚约束文件(ucf)

```
//clock
NET sys_clk LOC = AN20;
NET sys_clk IOSTANDARD = LVCMOS33;

//reset
Net rst_n PULLUP;
Net rst_n TIG;
Net rst_n LOC = AJ22;
NET rst_n IOSTANDARD= LVCMOS33;

//left
Net left PULLUP;
Net left TIG;
Net left LOC = AL18;
NET left IOSTANDARD= LVCMOS33;

//right
Net right PULLUP;
Net right TIG;
Net right LOC = AL16;
NET right IOSTANDARD= LVCMOS33;

//rotate
Net rotateR PULLUP;
Net rotateR TIG;
Net rotateR LOC = AP20;
NET rotateR IOSTANDARD= LVCMOS33;
```

```
//drop down
Net verticalspeed PULLUP;
Net verticalspeed TIG;
Net verticalspeed LOC = AM18;
NET verticalspeed IOSTANDARD= LVCMOS33;

//VGA parts
NET VGA_PSVAE_N LOC=A31;
Net VGA_PSVAE_N IOSTANDARD = LVCMOS33;
NET VGA_BLANK_N LOC=F28;
Net VGA_BLANK_N IOSTANDARD = LVCMOS33;
NET VGA_SYNC_N LOC=C29;
Net VGA_SYNC_N IOSTANDARD = LVCMOS33;

Net VGA_CLK LOC = C30;
Net VGA_CLK SLEW = FAST;
Net VGA_CLK IOSTANDARD = LVCMOS33;
Net VGA_CLK DRIVE = 8;

Net VGA_VSYNC LOC = E26;
Net VGA_VSYNC SLEW = FAST;
Net VGA_VSYNC IOSTANDARD = LVCMOS33;
Net VGA_VSYNC DRIVE = 8;

Net VGA_HSYNC LOC = C25;
Net VGA_HSYNC SLEW = FAST;
Net VGA_HSYNC IOSTANDARD = LVCMOS33;
Net VGA_HSYNC DRIVE = 8;

NET VGA_R<7> LOC=F26;
NET VGA_R<6> LOC=C23;
NET VGA_R<5> LOC=E24;
NET VGA_R<4> LOC=D25;
NET VGA_R<3> LOC=E23;
NET VGA_R<2> LOC=C24;
NET VGA_R<1> LOC=B33;
NET VGA_R<0> LOC=B32;
NET VGA_R<*> SLEW = FAST;
NET VGA_R<*> DRIVE = 8;
NET VGA_R<*> IOSTANDARD = LVCMOS33;

NET VGA_B<7> LOC=F29;
NET VGA_B<6> LOC=D29;
NET VGA_B<5> LOC=E29;
```

```
NET VGA_B<4> LOC=F30;
NET VGA_B<3> LOC=D30;
NET VGA_B<2> LOC=E31;
NET VGA_B<1> LOC=D31;
NET VGA_B<0> LOC=F31;
NET VGA_B<*> SLEW = FAST;
NET VGA_B<*> DRIVE = 8;
NET VGA_B<*> IOSTANDARD = LVCMOS33;

NET VGA_G<7> LOC=C28;
NET VGA_G<6> LOC=E28;
NET VGA_G<5> LOC=C27;
NET VGA_G<4> LOC=D27;
NET VGA_G<3> LOC=E27;
NET VGA_G<2> LOC=F25;
NET VGA_G<1> LOC=D26;
NET VGA_G<0> LOC=F24;
NET VGA_G<*> SLEW = FAST;
NET VGA_G<*> DRIVE = 8;
NET VGA_G<*> IOSTANDARD = LVCMOS33;
```

2.3.2 综合结果

Total REAL time to Xst completion: 549.00 secs

Total CPU time to Xst completion: 548.64 secs

-->

Total memory usage is 718432 kilobytes

Number of errors : 0 (0 filtered)

Number of warnings : 154 (0 filtered)

Number of infos : 3 (0 filtered)

可以看到 warning 有 154 个，下面就这些 warning 进行一下分析。

2.3.3 Warning 分析

2.3.3.1 始终为常数 warning

```
WARNING:Xst:1895 - Due to other FF/Latch trimming, FF/Latch
<enable_next_display_v_12> (without init value) has a constant
```

```
value of 0 in block <display_next_square>. This FF/Latch will be trimmed during the optimization process.
```

这里涉及到了综合时的优化问题，ISE在综合时发现它始终为常数，因此发出了一个warning，并把它优化掉。

2.3.3.2 未使用 warning

```
WARNING:Xst:646 - Signal <enable_blue_little_v<499:395>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <enable_blue_little_v<384:375>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <enable_blue_little_v<364:355>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
WARNING:Xst:646 - Signal <enable_blue_little_v<344:335>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.
```

我们并不是对于画面中每个像素都要处理的，这些像素不处理就默认它是黑的。

2.3.3.3 IP 核综合 warning

```
WARNING:Xst:2211 - "ipcore_dir/score.v" line 81: Instantiating black box module <score>.
WARNING:Xst:2211 - "ipcore_dir/spscore.v" line 87: Instantiating black box module <spscore>.
```

插入图片需要使用 IP 核，但是因为 IP 核不是采用 HDL 语言描述的，所以在综合的时候，IP 核会被当做黑盒子，从而出现警告，可以忽略。

2.3.4 综合报告

综合后，摘取.map 文件如下：

```
Design Summary
-----

Design Summary:
Number of errors:      0
Number of warnings:   4
Logic Utilization:
  Number of Slice Flip Flops:      1,704 out of 49,152   3%
  Number of 4 input LUTs:         22,734 out of 49,152  46%
```


Logic Distribution:

Number of occupied Slices: 12,168 out of 24,576 49%
Number of Slices containing only related logic: 12,168 out of 12,168 100%

Number of Slices containing unrelated logic: 0 out of 12,168 0%

*See NOTES below for an explanation of the effects of unrelated logic.

Total Number of 4 input LUTs: 22,969 out of 49,152 46%
Number used as logic: 22,734
Number used as a route-thru: 235

The Slice Logic Distribution report is not meaningful if the design is

over-mapped for a non-slice resource or if Placement fails.

Number of bonded IOBs: 39 out of 640 6%

Number of BUFG/BUFGCTRLs: 2 out of 32 6%

Number used as BUFGs: 2

Number of FIFO16/RAMB16s: 129 out of 320 40%

Number used as RAMB16s: 129

Number of DSP48s: 3 out of 512 1%

Average Fanout of Non-Clock Nets: 4.93

Peak Memory Usage: 625 MB

Total REAL time to MAP completion: 41 secs

Total CPU time to MAP completion: 37 secs

2.3.5 资源占用

Device utilization summary:

Selected Device : 4vsx55ff1148-12

Number of Slices: 11686 out of 24576 47%
Number of Slice Flip Flops: 1704 out of 49152 3%
Number of 4 input LUTs: 22885 out of 49152 46%
Number of IOs: 38
Number of bonded IOBs: 36 out of 640 5%
Number of FIFO16/RAMB16s: 129 out of 320 40%
Number used as RAMB16s: 129
Number of GCLKs: 2 out of 32 6%

Number of DSP48s:	3 out of 512	0%
-------------------	--------------	----

2.3.6 时序分析

综合后，摘取.syr 文件如下：

Timing Summary:

Speed Grade: -12

Minimum period: 11.247ns (Maximum Frequency: 88.915MHz)

Minimum input arrival time before clock: 4.236ns

Maximum output required time after clock: 4.547ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

可以得出最高频率是 88.915MHz.

3 设计总结

3.1 注意事项与编程技巧

3.1.1 VGA 时序分析

3.1.1.1 VGA 简介

VGA (Video Graphics Array)是 IBM 在 1987 年随 PS / 2 机一起推出的一种视频传输标准,具有分辨率高、显示速率快、颜色丰富等优点,在彩色显示器领域得到了广泛的应用。

一个完整的 VGA 图形显示系统由三部分组成:图形主机、显示卡和显示器。主机所发出的图象数据由显示卡负责接收和储存,并对该数据进行处理和转换,生成一定的时序信号传送给显示器;显示器按照显示卡所发送的信号进行屏幕显示。

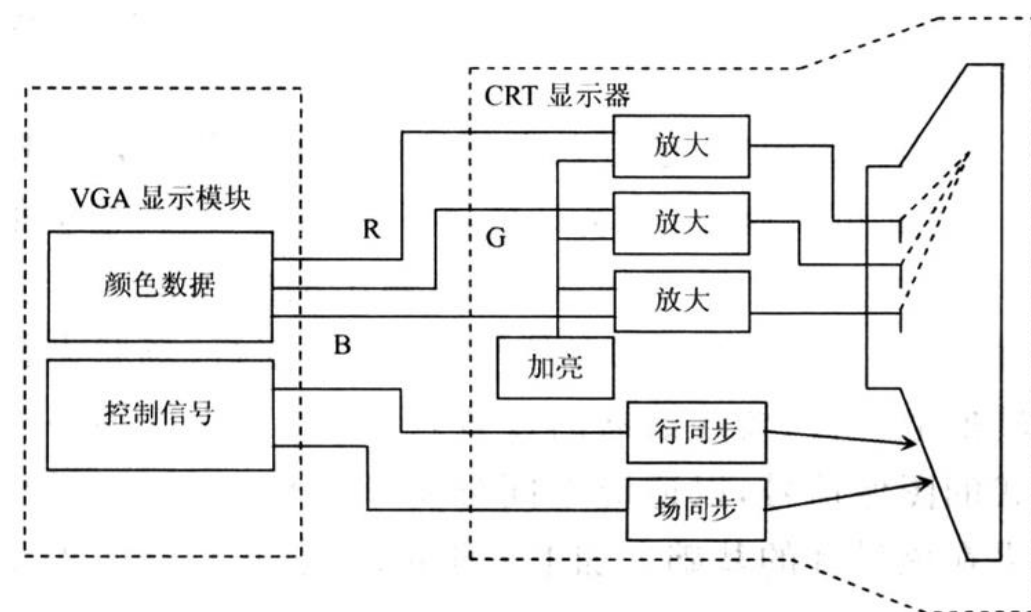


Figure 1.VGA 显示控制器控制 CRT 显示器图

显示器扫描方式分为逐行扫描和隔行扫描。我们采用逐行扫描,是扫描从屏幕左上角一点开始,从左像右逐点扫描,每扫描完一行,电子束回到屏幕的左边下一行的起始位置,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。隔行扫描是指电子束扫描时每隔一行扫一线，完成一屏后在返回来扫描剩下的线，隔行扫描的显示器闪烁的厉害，会让使用者的眼睛疲劳。

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，即刷新一屏的频率，常见的有 60Hz, 75Hz 等等。

标准的 VGA 显示的场频 60Hz, 行频 31.5KHz。

行场消隐信号:是针对老式显像管的成像扫描电路而言的。电子枪所发出的电子束从屏幕的左上角开始向右扫描，一行扫完需将电子束从右边移回到左边以便扫描第二行。在移动期间就必须有一个信号加到电路上，使得电子束不能发出。不然这个回扫线会破坏屏幕图像的。

这个阻止回扫线产生的信号就叫作消隐信号,场信号的消隐也是一个道理。

3.1.1.2 VGA 时序图

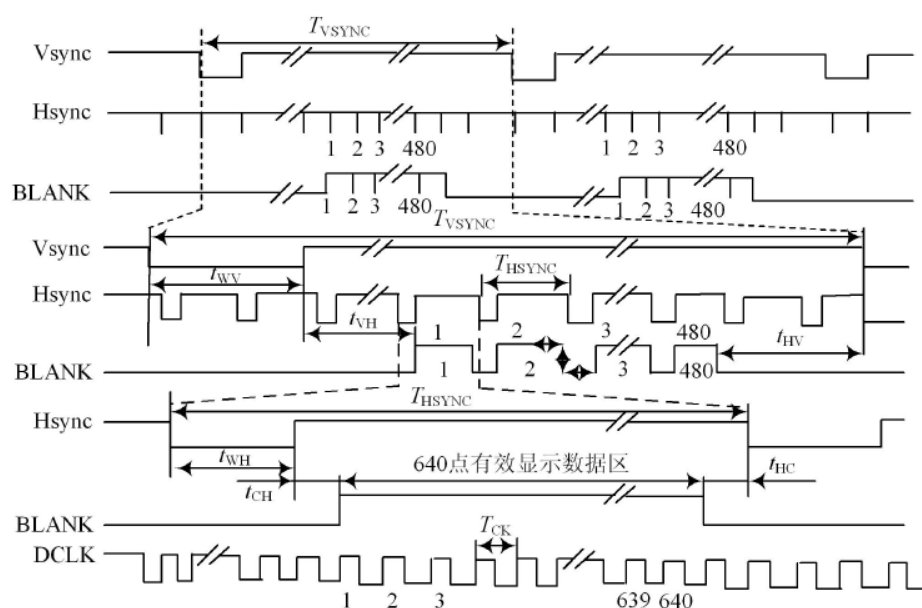


Figure 2.VGA(640X480,60Hz)图像格式信号时序图

VESA 中定义行时序和场时序都需要同步脉冲（Sync a）、显示后沿（Back porch b）、显示时序段（Display interval c）和显示前沿（Front porch d）四部分。VGA 工业标准显示模式要求：行同步，场同步都为负极性，即同步脉冲要求是负脉冲。

3.1.1.3 VGA 接口定义

本次实验采用了 640 * 480 的分辨率，刷新频率为 60Hz。根据 VGA 的时序表，我们需要约 25MHz 的时钟，具体计算如下：

每场对应 525 个行周期($525=10+2+480+33$),其中 480 为显示行。每场有场同步信号,该脉冲宽度为 2 个行周期的负脉冲，每显示行包括 800 点时钟,其中 640 点为有效显示区,每一行有一个行同步信号，该脉冲宽度为 96 个点时钟。由此可知：行频为 $525*59.94=31469\text{Hz}$,需要点时钟频率： $525*800*59.94$ 约 25MHz.

Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

在编写 Verilog 代码时，需要注意显示的合法区域：

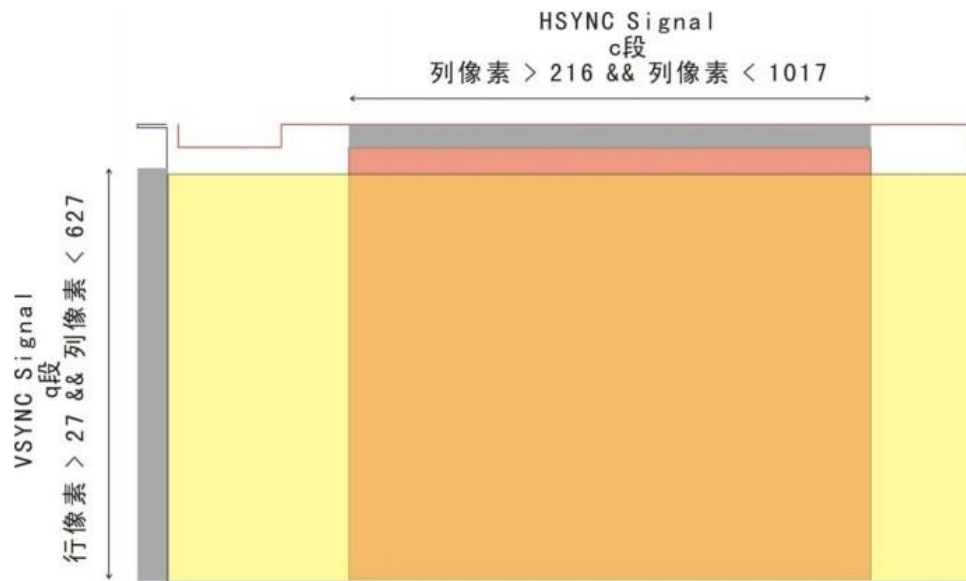


Figure 3.800 * 600 的显示区（橙色）

需要输出的 VGA 信号应该都落在橙色的显示区内，因此对于每个点(cnt_h, cnt_v)，我们需要判断它是否在显示区内，再赋给它对应的显示信号。

对应的代码如下：

```
always @ ( posedge clk or negedge rst_n )
begin
    if( !rst_n )
        isready <= 1'b0;
    else if( cnt_h >= 11'd143 && cnt_h < 11'd783 && cnt_v >= 11'd32
    && cnt_v < 11'd512 )
        isready <= 1'b1;
    else isready <= 1'b0;
end

/*****/

assign hsync = ( cnt_h <= 11'd95 ) ? 1'b0 : 1'b1;
assign vsync = ( cnt_v <= 11'd1 ) ? 1'b0 : 1'b1;
assign col_addr_sig = isready ? ( cnt_h - 11'd143 ) : 11'd0;
assign row_addr_sig = isready ? ( cnt_v - 11'd32 ) : 11'd0;
```

3.1.2 图片的存储

为了提高游戏的美观性，我们添加了开场动画与结束动画，核心内容是模仿 PPT 里的幻灯

片切入效果，将储存在开发板 RAM 里的图片用展开动画的形式展现出来。

TETRIS

俄罗斯方块

李琛 赵春琪

Figure 4.开始界面

小组成员：

李琛 赵春琪

THANK YOU!

謝謝觀看！

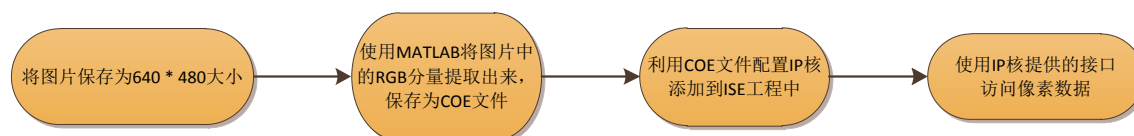
Figure 5.结束动画

同时，为了显示得分与“next”、“score”、“speed up!”等字样，我们也需要把这些字样作为图片保存在 RAM 里。由于这些字样不需要全屏显示，所占空间较小，我们可以认为一共需要大约 3 张 640 * 480 大小的图片。

初始想法是加入的图片都是 256RGB 全彩色图像，但是经过计算发现这样是不行的。如果采用 256 级存储的话，需要 $640 \times 480 \times 24 \times 3 = 22118400 \text{bit} \approx 22 \text{M}$ 的储存空间，而根据 JFM4VSX55 的器件手册，它的 RAM 只有 5760K，即约 5M。如果需要存 256 级图片的话，就要访问 ROM 空间，较为繁琐。这样我们只能采用压缩 RGB 位的方法，选择将 256 级改为 1 级，即 RGB 每一位都只有 0/1 两个状态。这样存储三张图片的空间就节省到了

$640 * 480 * 3 * 3 = 2764800\text{bit} \approx 2.7\text{M}$ ，可以存放在 RAM 内。但是这样带来的就是可选颜色的减少，因此我们最后退而求其次，选择使用黑白图片。

将图片添加进工程的流程如下：



其中 MATLAB 的脚本如下：

```
im = imread('start.bmp');
[a,b,c] = size(im);
rgbs = im;
imshow(rgbs);
fid = fopen('start.coe', 'w+');
fprintf(fid, 'memory_initialization_radix=2;\n');
fprintf(fid, 'memory_initialization_vector =\n');
for i = 1 : a
    for j = 1 : b
        fprintf(fid, '%d',rgbs(i, j, 1) > 0);
        fprintf(fid, '%d',rgbs(i, j, 2) > 0);
        fprintf(fid, '%d,\n',rgbs(i, j, 3) > 0);
    end
end
fprintf(fid, ';');
fclose(fid);
```

3.1.3 Verilog 有限状态机三段式描述方法

状态机描述时关键是要描述清楚几个状态机的要素，即如何进行状态转移，每个状态的输出是什么，状态转移的条件等。具体描述时方法各种各样，最常见的有三种描述方式：

(1) 一段式：整个状态机写到一个 always 模块里面，在该模块中既描述状态转移，又描述状态的输入和输出；

(2) 二段式：用两个 always 模块来描述状态机，其中一个 always 模块采用同步时序描述

状态转移；另一个模块采用组合逻辑判断状态转移条件，描述状态转移规律以及输出；

(3) 三段式：在两个 always 模块描述方法基础上，使用三个 always 模块，一个 always 模块采用同步时序描述状态转移，一个 always 采用组合逻辑判断状态转移条件，描述状态转移规律，另一个 always 模块描述状态输出(可以用组合电路输出，也可以时序电路输出)。

一般而言,推荐的 FSM 描述方法是后两种。这是因为：FSM 和其他设计一样，最好使用同步时序方式设计，以提高设计的稳定性，消除毛刺。状态机实现后，一般来说，状态转移部分是同步时序电路而状态的转移条件的判断是组合逻辑。

在本游戏中，状态机较为复杂，因此采用二段式的设计方法，在判断状态转移时也产生输出，这样较为直观，但是可能会有毛刺的产生。因此，最好在后面将其改写为三段式的状态机。

3.2 体会心得

这次 Project 作为 专用集成电路设计方法 这门课的课程大作业，很大程度上对课上讲的完整 IC 项目的设计流程实现了补充。具体到用 Verilog 完成项目，我们有以下体会。

Verilog 进行 Code 时，尽量坚持一个 V 文件一个 module 的原则。按照功能进行划分模块，并且分别用一个文件进行存放，利于项目管理，并且文件的结构也和整个系统的结构对应。在团队合作进行 code 时，这样也能够更好地实现版本控制。

硬件的测试相比软件要来的困难而不直观。当我们完成一个版本的 Verilog 时，想要进行测试的话，用 Modelsim 进行仿真是一种手段，但是出来的结果是时序图，并不是那么具有感受性；但是具体烧到板子上进行测试，又很难查验出具体哪里出了问题。所以一般而言的测试流程是最经济最有效的，即先在 Modelsim 里进行仿真，再烧到板子上实际进行测试。

同时，这次的课程作业是我首次处理需要使用 VGA 进行输出的情况。之前有用 C 写过俄罗斯方块，也用 Verilog 做过实验，但这次需要将写好的游戏展示在显示屏上，就是一种全新

的挑战了。在完成算法的同时，我们也需要学习 VGA 的时序，了解 HSYNC/VSYNC 等参数的实际意义，才能正确的将自己的算法显示出来。可以说，VGA 显示是最有挑战性的地方。在熟悉了 VGA 的时序之后，我们也对游戏进行了进一步的改进，例如在开始与结束界面处我们添加了切入与切出动画，这些都是在理解了时序基础上补充的。

团队合作中，代码的注释就显得格外重要。我们自己写东西时常常因为熟悉自己的思路，就省了注释这一步。但在 Group Work 里没了注释确实花费了我们很多时间来对代码进行理解。

3.3 总结

这是真正意义上的纯硬件实现一个 Project，也是真正意义上的走了一遍 ASIC 设计中自顶向下的设计流程。在这个项目中，我们也熟悉了 FPGA 的编程、烧录、调试步骤，也对 VGA 显示的工作原理有了了解，同时也丰富了小组合作的经验。我们从中都有一些收获。

4 附录：主要文件列表

4.1 Sources



automaton.v



over_sync_module.v



debouncer.v



over_vga_control_module.v



display_border.v



show_next.v



display_little_square.v



show_score.v



display_moving_square.v



square_gen.v



display_next_square.v



start_sync_module.v



display_score.v



start_vga_control_module.v



div_clk.v



transform.m



game_display.v



v_speed_debouncer.v



game_sync_module.v



vga_select_module.v



loading_happen.v



main_module.v

4.2 Images

4.2.1 数字

4 3 2 1 0

9 8 7 6 5

4.2.2 文字

SCORE
SPEED UP!

4.2.3 界面

TETRIS

俄罗斯方块

李琛 赵春琪

小组成员：
李琛 赵春琪

THANK YOU!

謝謝觀看！