

**浙江大学**

**计算机逻辑设计·课程报告**

## 基于数字系统的星空跳跃小游戏设计

汪珉凯，岳晟稼

2024年1月5日

目录

[基于数字系统的星空跳跃小游戏设计 1](#_Toc155381218)

[1. 摘要 3](#_Toc155381219)

[2. 游戏设计背景 3](#_Toc155381220)

[2.1 基于FPGA的游戏设计 3](#_Toc155381221)

[2.2 游戏设计 4](#_Toc155381222)

[2.3 设计阶段 5](#_Toc155381223)

[2.4 重难点分析 6](#_Toc155381224)

[3. 设计原理 6](#_Toc155381225)

[3.1 理论基础 6](#_Toc155381226)

[3.1.1 VGA显示原理 6](#_Toc155381227)

[3.1.2 PS/2键盘 9](#_Toc155381228)

[3.1.3 状态机的设计 10](#_Toc155381229)

[3.1.4 硬件描述语言 11](#_Toc155381230)

[3.1.5 可编程阵列逻辑 11](#_Toc155381231)

[3.2 实验器材 11](#_Toc155381232)

[4. 整体结构 11](#_Toc155381233)

[4.1 输入与输出 11](#_Toc155381234)

[4.2 模块工作流程 12](#_Toc155381235)

[4.2.1 VGA模块程序流程图 12](#_Toc155381236)

[4.2.2 宏观模块联系 12](#_Toc155381237)

[4.3 游戏过程与设计思路 13](#_Toc155381238)

[5. 模块介绍 13](#_Toc155381239)

[5.1 图片部分 14](#_Toc155381240)

[5.1.1 .coe文件生成 14](#_Toc155381241)

[5.1.2 IP核导入与VGA显示 14](#_Toc155381242)

[5.2 PS/2键盘部分 18](#_Toc155381243)

[5.3 七段数码管部分 18](#_Toc155381244)

[5.4 露比显示部分 19](#_Toc155381245)

[5.5 障碍物显示部分 19](#_Toc155381246)

[5.5.1 苦力怕地雷部分 19](#_Toc155381247)

[5.5.2 激光门部分 19](#_Toc155381248)

[5.6 top部分（包含碰撞检测与计分与运动状态显示） 19](#_Toc155381249)

[6. 模拟仿真 20](#_Toc155381250)

[7. 结果演示与分析 21](#_Toc155381251)

[8. 实验改进与不足 23](#_Toc155381252)

[9. 成员分工 24](#_Toc155381253)

[10. 参考资料与源代码 24](#_Toc155381254)

[10.1 bmp转coe部分 24](#_Toc155381255)

[10.2 TOP代码展示： 26](#_Toc155381256)

[10.3 ucf引脚文件 38](#_Toc155381257)

# 摘要

木星跳跃（Jupiter Jump）是一款趣味十足的休闲类游戏，其实跟普通游戏没多大区别，唯一感觉过瘾的是这极速般的跳跃方式，让游戏的上手程度也是十分爽。游戏讲述一个宇航员驾驶着他的宇宙飞船穿梭在浩瀚无际的宇宙中，忽然被不知道从哪里射出的激光击中，落到了木星上，然后宇航员就开始沿着木星表面高速跑起来了!控制小宇航员高高跃起，避开外星炸弹，使劲下踩，乘着间隙踩着绿块，一路蹦跳前进！木星跳跃 Jupiter jump游戏的计分也很有意思，穿过绿色激光门来获取跳跃能量分数。今天，我们将其带入“计算机逻辑设计”教室，并将其改名为星空跳跃，旨在使用FPGA板和verilog HDL语言重新创建这款简单而有趣的游戏。在实现这款游戏的过程中，我们主要提取其游戏内核的躲避玩法。

# 游戏设计背景

## 基于FPGA的游戏设计

FPGA（Field－Programmable Gate Array，即现场可编程门阵列）是在PAL、GAL、CPLD等可编程器件的基础上进一步发展产物。以硬件描述语 言（Verilog或VHDL）所完成的电路设计，可以经过简单的综合与布局， 快速的烧录至FPGA上进行测试。这些可编程元件可以被用来实现一些基本的逻辑门电路或者更复杂的一些的组合功能。Xilinx的SWORD实验板为可编程的实验板，而且实验板还可以和许多外部设备进行连接，例如PS2、鼠标或者显示器，从而可以在FPGA的基础上设计使用PS2、鼠标的调用， 或者显示器上显示图片。

本次设计所用的硬件编程软件为Xilinx公司设计的ISE14.7软件，所使用的语言为Verilog语言。该软件的使用和Verilog语言的学习都较为简单，和c语言有相似之处。同方式或混合方式对设计建模。这些方式包括：行为描述方式建模；数据流方式建模；结构化方式建模等。Verilog HDL 中有两类数据类型：线网数据类型和寄存器数据类型。线网类型表示构件间的物理连线，而寄存器类型表示抽象的数据存储元件。通过模块化的设计与相应的外部接口，可以实现较好的人机交互。

本次设计通过原理图和行为描述相结合的方式实现游戏的设计，显示器、PS2 外接设备优化游戏。

## 游戏设计

为增强游戏的灵活性，我们组在木星跳跃的基础上加入了手动改变方向的功能。游戏方主要通过键盘操纵游戏人物loopy的动作，通过躲避地雷并尽可能多的穿过激光门，来保持其生命。评价游戏表现得方式是通过存活时间和接触激光门门的次数来判断的。因此这款游戏能够考验玩家的注意力与敏锐程度。

游戏得分在本游戏中通过七段数码管记录。游戏总得分均会在开始游戏时置零，之后在游戏角色死亡时结束。因此，在游戏中的表现情况可以通过积分面板来评判。

为了增加游戏趣味性，我们借助上文所述的设定给游戏过程增加了一定其他操作。我们加入到方向改变键位，使得玩家不一定要凭借上下边界进行弹回反向操作，并且根据VGA显示的特性将地面的深渊设置了黑洞，在靠近边缘时加速有几率进入黑洞。这样游戏的可玩性上升了很多，玩家可以在漫天的苦力怕地雷中存活并自由穿梭。

## 设计阶段

本次课程设计是利用FPGA板，通过底层硬件控制的方法实现马里奥这款游戏，但是由于资源的限制，进行了一定程度上的简化。 在本组设计中，我们用外接的ps2键盘进行游戏的控制，采用两个按键（J、K键），分别控制角色进行上下的移动，来躲避障碍，还有L键进行角色移动方向的修改，最后使用任意键进行游戏状态重置。不同大小的障碍物会随机出现，和发射子弹。当障碍与游戏角色相撞时，本方游戏结束显示Game Over。游戏界面由显示器显示，板子上的七段数码管显示游戏得分。

设计主要分为以下阶段：

1. 基本规则设计：小组成员讨论游戏主题、游戏背景、游戏角色、游戏规则及改编等内容。
2. 图片处理：有关本实验所需的人物、地雷、激光门的图片处理与IP核的生成。
3. Top程序设计：针对实验所需规划输入、输出以及相关模块的设定。
4. 底层模块的编写：针对各项模块要求编写Verilog代码。
5. 代码改错与试运行：即Debug阶段。

## 重难点分析

本游戏设计难点在于：

1. 苦力怕和激光门的移动
2. 游戏人物的移动速度与方向
3. 游戏角色碰撞物体的判断
4. 碰撞得分的判断
5. 游戏结束的判定
6. 分数（坚持时间+碰撞门数量）在7段数码管上的展现

# 设计原理

## 理论基础

### VGA显示原理

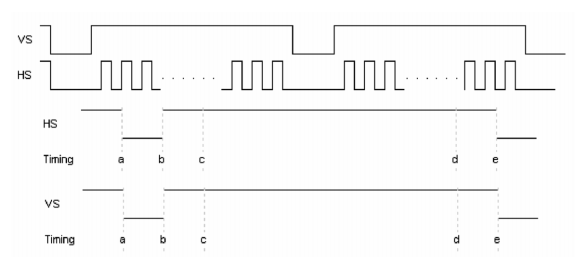
在本次作业中，我们利用了实验板中的 VGA 显示屏，通过对于 VGA 时序控制信号以及输出颜色的控制，实现了显示屏信号的分时输出。

VGA 是 IBM 在 1987 年随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。

一般所使用的 VGA 显示器均为标准五输入类型，如下表所示。通过对行列同步信号进行控制，遍历输出各个像素点的颜色信息，最终输出完整图像。



信号线中 HS 与 VS 用于控制显示器的显示频率，R、G 与 B 则用于控制显示器当前像素的颜色。



上图是 VGA 的标准时序。不难看出，列同步信号与行同步信号的时序都分

为 4 段。a-b 段称为同步期，此时同步信号为低电平；b-c 段称为消隐后肩，

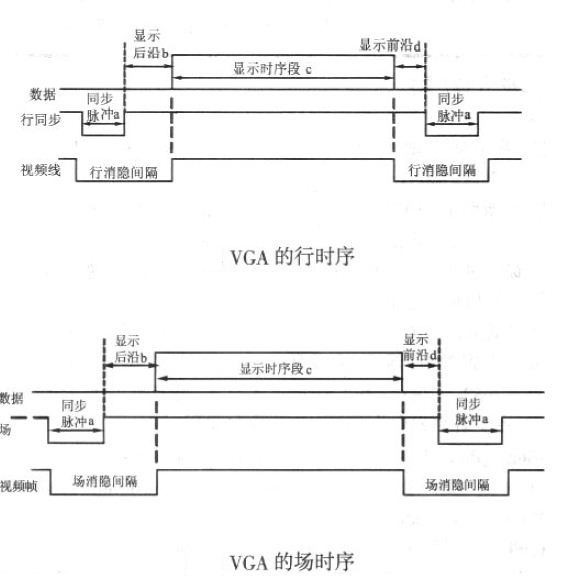
c-d 段称为显示期，是数据的有效区域，d-e 段称为消隐前肩，这三段的同步信

号为高电平。显示器就是根据列同步信号与行同步信号的周期，来决定显示的分

辨率、频率等参数。不同分辨率与频率对应的 VS-HS 数值，可以通过查表得到。

在实验中只需要在一个 VS-HS 完成一个周期的时间内，按从左上角到右下角的顺序输出各个像素点的颜色，就能在显示器上显示出对应的图案。需要注意的是，VGA 颜色的输出必须严格遵循时序，只能在 c-d 段输出非 0 数据，否则会对显示造成干扰。

VGA显示依赖于行扫描信号、场扫描信号和当前像素的RGB值。电子枪从左上角开始从左向右扫描，一行扫描完后，进行行消隐（使电子束不再发射，同时将其回归到下一行的最左边，此时行扫描信号为0），接着扫描下一行，当所有行扫描完后，进行场消隐（电子束回到第一行的最左边，此时场扫描信号为0），在每次指定的时钟的上升沿，输出场扫描信号、行扫描信号和对应的RGB值。



图表1 VGA的行、场时序

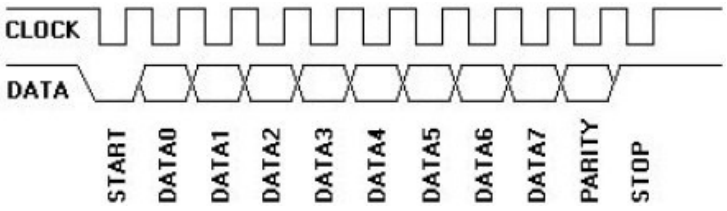
### PS/2键盘

PS/2 通信协议是一种双向同步串行通信协议。通信的两端通过 CLOCK(时钟脚)同步，并通过国 DATA(数据脚)交换数据。一般两台设备间传输数据的最大时钟频率是 33kHz，大多数 PS/2 设备工作在 10--20kHz。推荐值在 15kHz 左右，也就是说，CLOCK 高、低电平的持续时间都为 40us。每一数据帧包含 11—12 位，具体含义如下图所示。



图表 2 PS/2 通信数据帧格式

PS/2 到主机的通信时序如下图所示。数据在 PS/2 时钟的下降沿读取，PS/2 的时钟频率为 10—16.7kHz。对于 PS/2 设备，一般来说从时钟脉冲的上升沿到一个数据转变的时间至少要有 5us；数据变化到下降沿的时间至少要有 5us，并且不大于25us，这个时序非常重要应该严格遵循。主机可以在第 11 个时钟脉冲停止位之前把时钟线拉低，使设备放弃发送当前字节，当然这种情况比较少见。在停止位发送后设备在发送下个包前应该至少等待 50us，给主机时间做相应的处理。不过主机处理接收到的字节时一般会抑制发送(主机在收到每个包时通常自动做这个)。在主机释放抑制后，设备至少应该在发送任何数据前等 50us。



图表 3 设备到主机的通信

### 状态机的设计

有限状态机（Finite State Machine FSM）是时序电路设计中经常采用的一种方式，尤其适合设计数字系统的控制模块，在一些需要控制高速器件的场合，用状态机进行设计是一种很好的解决问题的方案，具有速度快、结构简单、可靠性高等优点。有限状态机非常适合用 FPGA 器件实现，用 Verilog HDL 的 case 语句能很好地描述基于状态机的设计，再过 EDA 工具软件的综合，一般可以生成性能极优的状态机电路，从而使其在执行时间、运行速度和占用资源等方面优于用 CPU 实现的方案。 有限状态机一般包括组合逻辑和寄存器逻辑两部分，寄存器逻辑用于存储状态，组合逻辑用于状态译码和产生输出信号。根据输出信号产生方法的不同，状态机可分为两类： 米里型(Mealy)和摩尔型(Moore)。摩尔型状态机的输出只是当前状态的函数。米里型状态机的输出是在输入变化后立即变化的，不依赖时钟信号的同步，摩尔型状态机的输入发生变化时还需要等待时钟的到来，必须在状态发生变化时才会导致输出的变化，因此比米里型状态机要多等待一个时钟周期。

### 硬件描述语言

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。 使用 Verilog 描述硬件的基本设计单元是模块（module）。构建复杂的电子电路，主要是通过模块的相互连接调用来实现的。模块被包含在关键字 module、endmodule 之内。实际的电路元件。Verilog 中的模块类似 C 语言中的函数，它能够提供输入、输出端口， 可以实例调用其他模块，也可以被其他模块实例调用。

### 可编程阵列逻辑

PAL 器件由可编程的与阵列、固定的或阵列和输出反馈单元组成。不同型号 PAL 器件有不同的可编程阵列逻辑输出和反馈结构，适用于各种组合逻辑电路和时序逻辑电路的设计，是一种可程式化的装置。PLA 具有一组可程式化的 AND 阶，AND 阶之后连接一组可程式化的 OR 阶，如此可以达到：只在合乎设定条件时才允许产生逻辑讯号输出。 PLA 如此的逻辑闸布局能用来规划大量的逻辑函式，这些逻辑函式必须先以积项（有时是多个积项）的原始形式进行齐一化。在 PLA 的应用中，有一种是用来控制资料路径， 在指令集内事先定义好逻辑状态，并用此来产生下一个逻辑状态（透过条件分支）。 举例来说，如果目前机器（指整个逻辑系统）处于二号状态，如果接下来的执行指令中含有一个立即值（侦测到立即值的栏位）时，机器就从第二状态转成四号状态，并且也可以进一步定义进入第四状态后的接续动作。

## 实验器材

本实验主要采用ISE平台进行实验。ISE 是使用 XILINX 的 FPGA 的必备的设计工具。它可以完成 FPGA 开发的全部流程，包括设计输入、仿真、综合、布局布线、生成 BIT 文件、配置以及在线调试等，功能非常强大。ISE 除了功能完整，使用方便外，它的设计性能也非常好，以集成的时序收敛流程整合了增强性物理综合优化，提供最佳的时钟布局、更好的封装和时序收敛映射，从而获得更高的设计性能。

除此之外，SWORD板是进行验证与视频录制的载体，我们需要通过其来验证自己程序的有效性。此外，在处理图片部分我们使用了PS软件与C++语言的程序编译器；这些均需要在个人计算机上完成。

# 整体结构

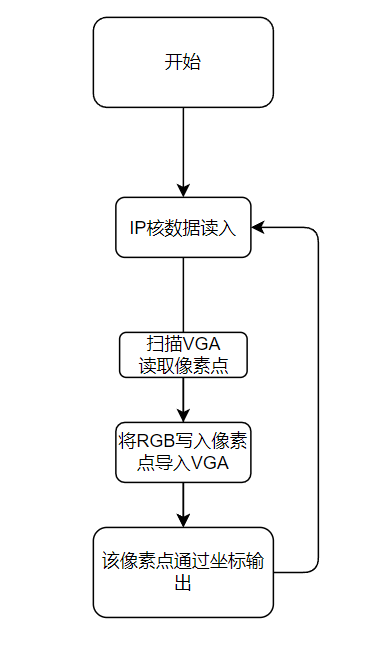
## 输入与输出

本游戏使用键盘进行输入—用键盘JKL键分别进行加速减速和方向转换的操作。

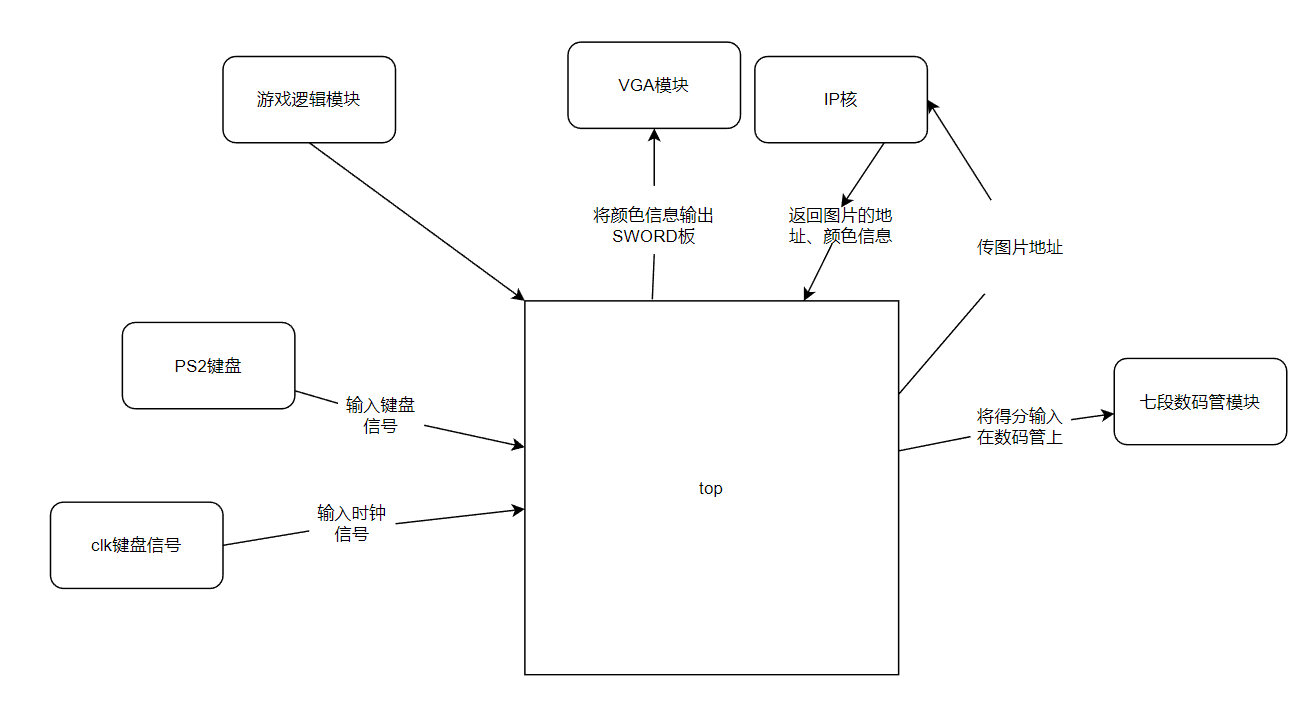
输出分为两部分一部分为VGA模块的显示器游戏界面输出，另一部分为SWORD板上的七段数码管显示的玩家总体得分。

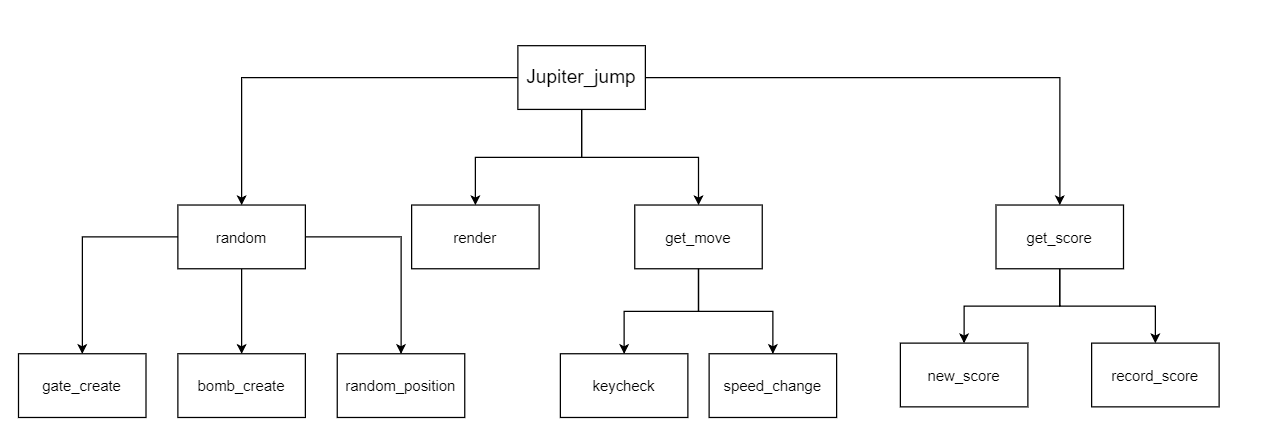
## 模块工作流程

### VGA模块程序流程图



### 宏观模块联系





## 游戏过程与设计思路

程序上板后，会直接开始游戏，在游戏的初始状态开始，露比位于屏幕最左边，随时间流逝会出现需要躲避的苦力怕地雷和可以用来得分的激光门，露比需要躲避出现的障碍物并尽量碰到激光门，延长自己的生存时间。

玩家可以通过按下按键（JKL）来控制露比的移动，使其躲过苦力怕；

一旦碰撞到苦力怕，则玩家游戏结束，计分板上玩家停止计分，屏幕显示GAMEOVER界面。

此时任意键都可以作为游戏快速重启的方式。

因此，本实验主要分为以下几个模块：图片的导入（IP核），人物的运动速度与方向，碰撞检测与反馈，以及计分模块。下面我们将以代码块的模式来讲解上述模块。

# 模块介绍

## 图片部分

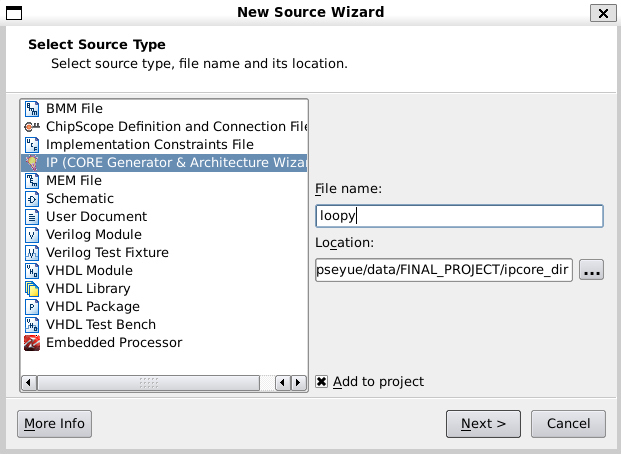
### .coe文件生成

1）用微软系统截图得到的32位bmp图片

2）使用c++语言库函数，将32位bmp图片生成为coe文件，代码请见源代码10.1节。

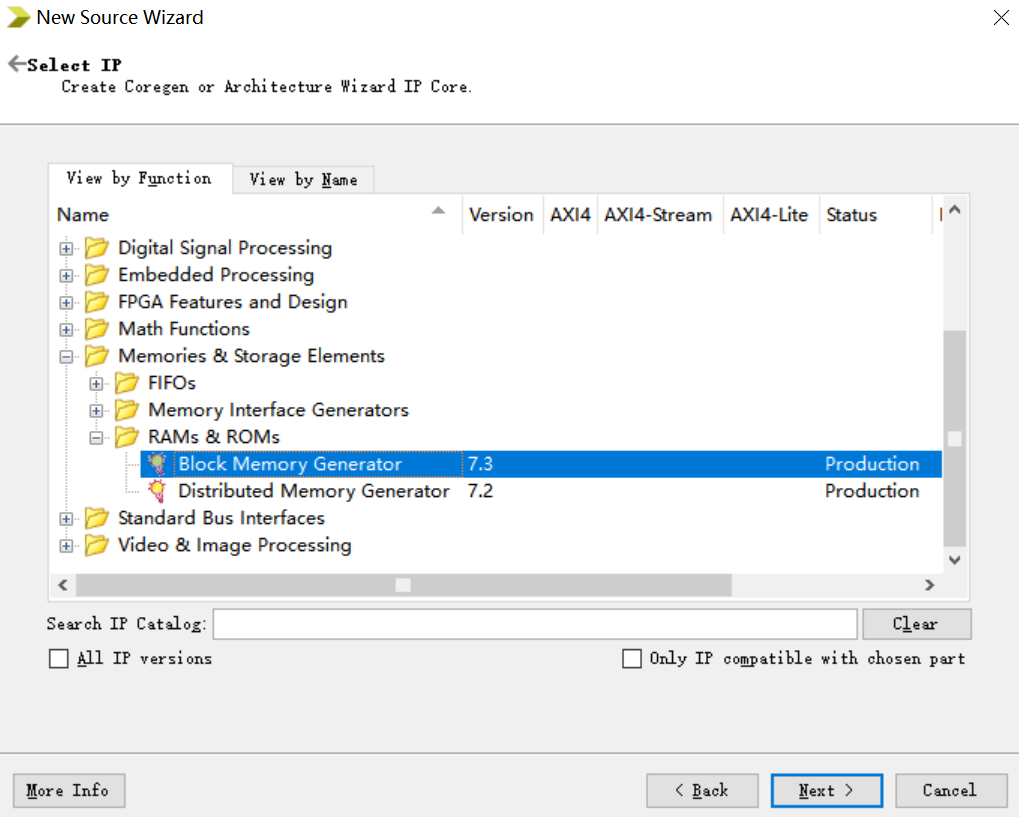
### IP核导入与VGA显示

1. **以loopy为例，add new source，添加新IP文件**



1. **选择类型**

选择 RAM & ROM 中的 Block Memory Generator 7.3

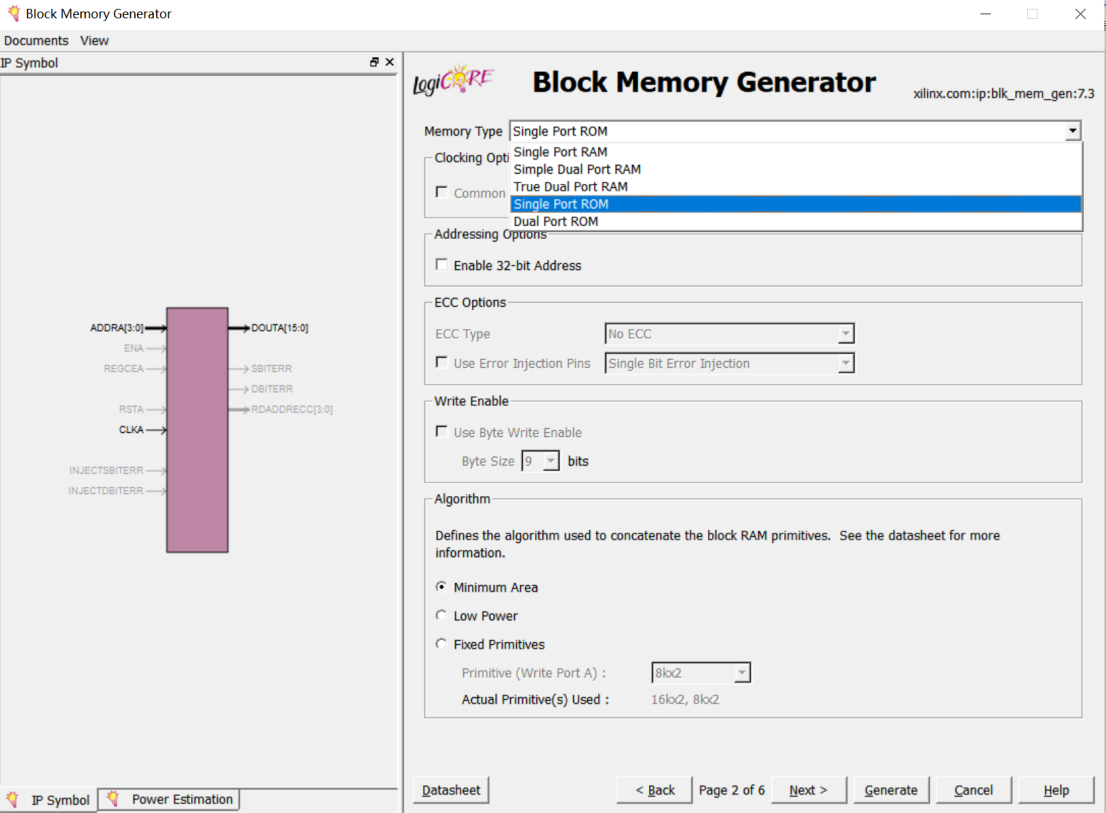


其中.xco文件生成较慢：

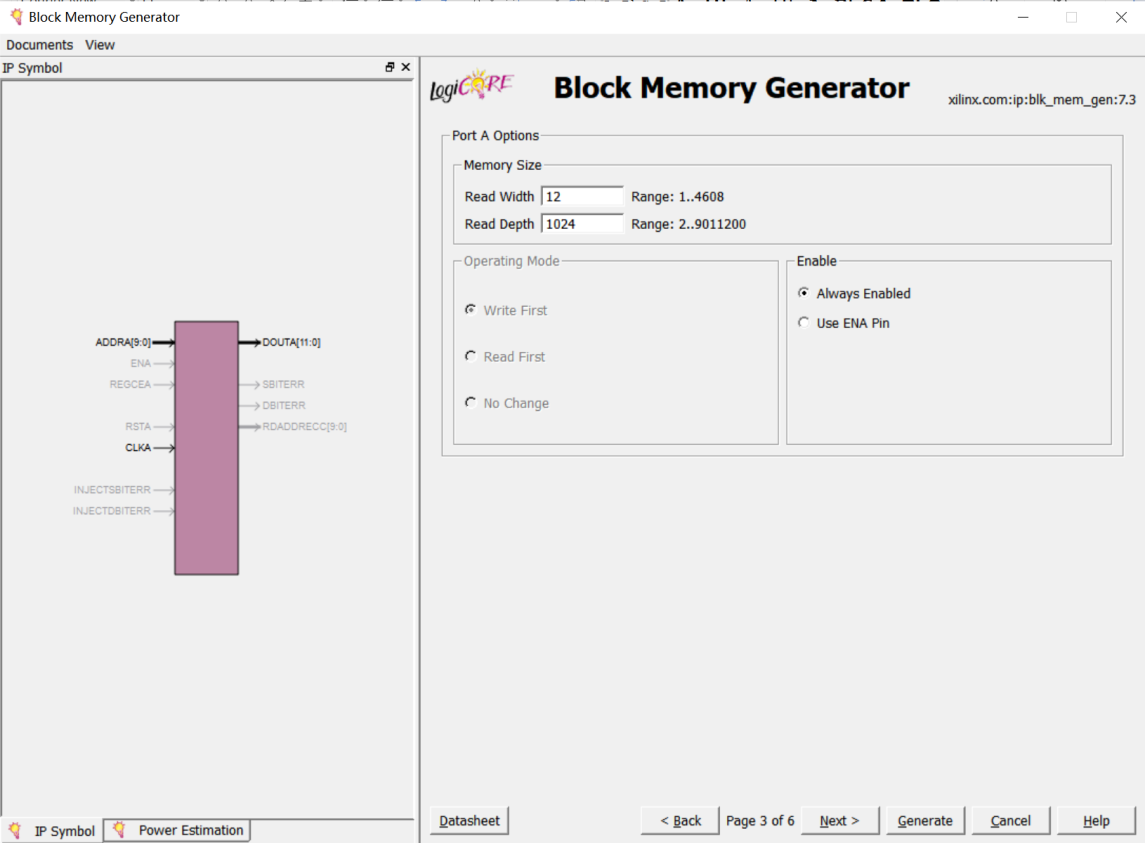


1. **配置IP核**

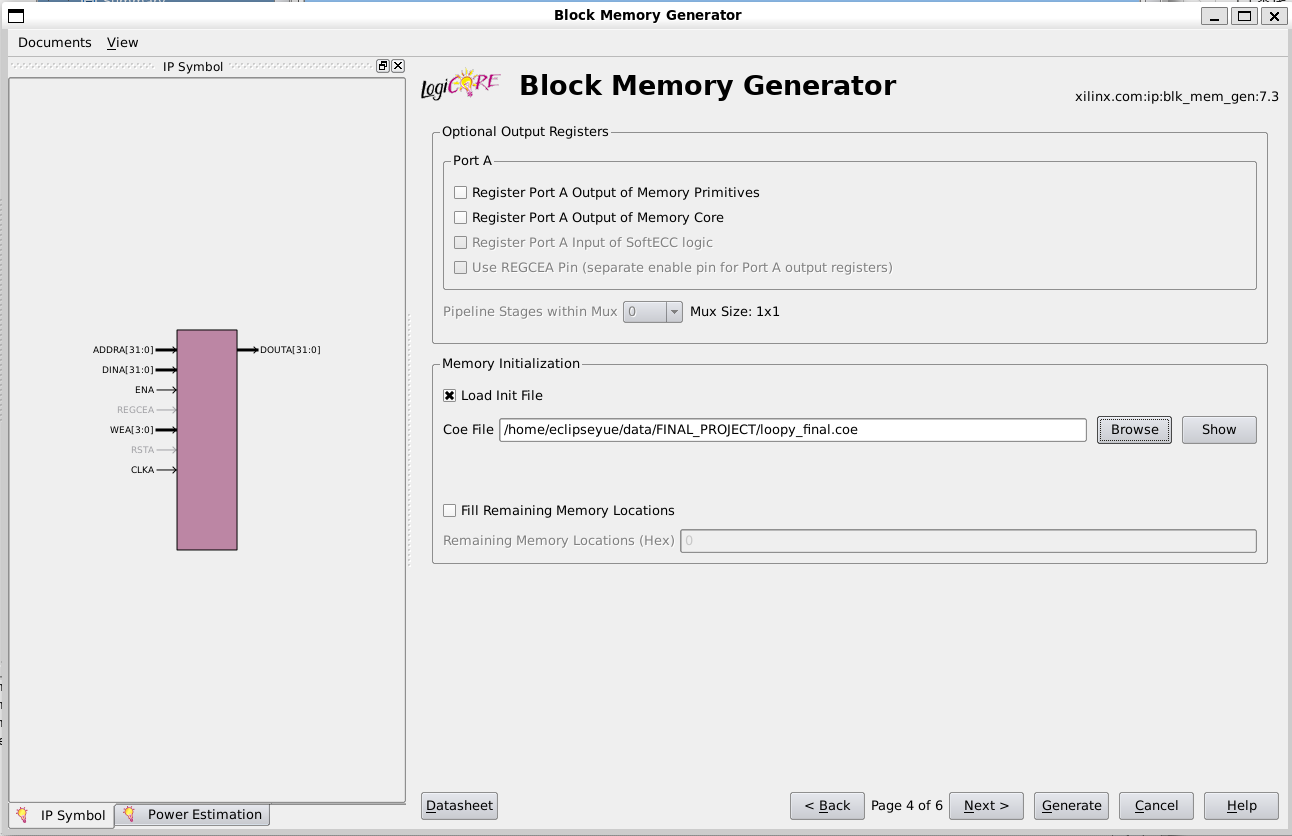
选择single port ROM



width是12位VGA输出，Depth是图片的大小30\*30=900（以loopy为例）



添加.coe文件



选择完毕后，点击generate即可生成ip核，ip核会根据coe文件的大小来加载，所以图片越大，加载的时间越长，画面未响应的现象是非常正常的。

1. **生成IP核，生成成功后会有图标提示**



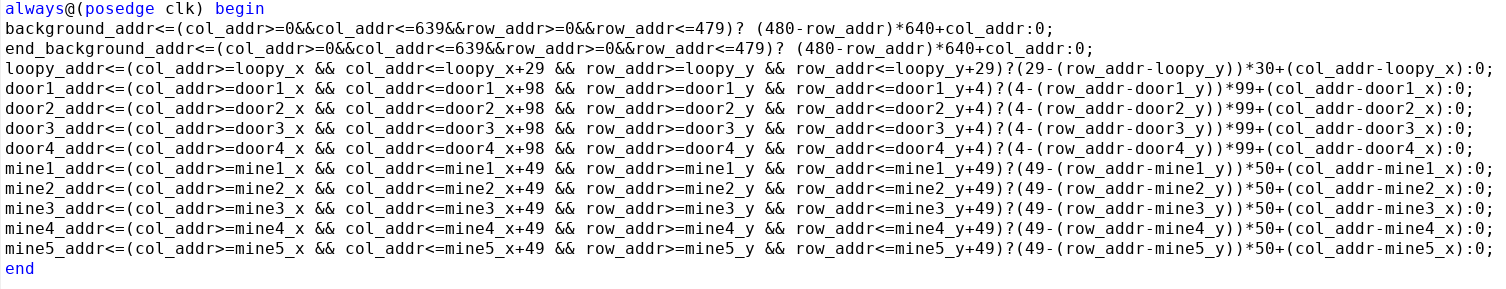
1. **VGA的使用和IP核的调用**

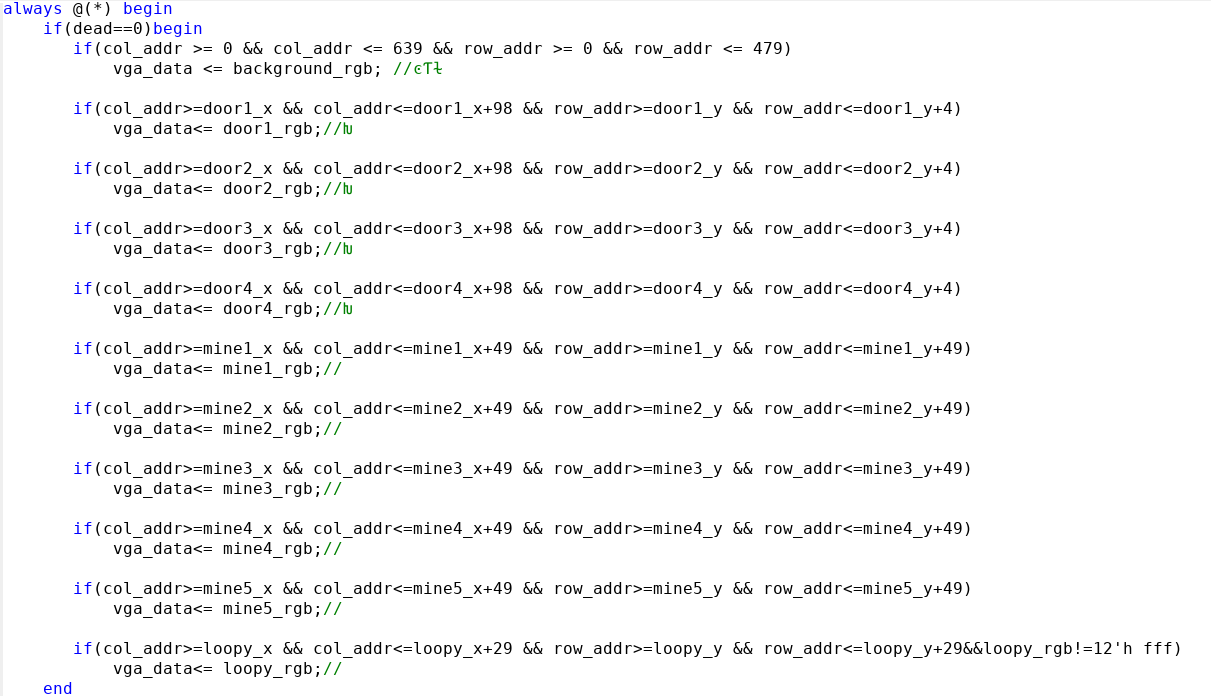
在调用vgac模块之后，我们首先要调用自己生成的ip核的模块。



接着，我们需要给ip核的输入进行赋值。我们采取用图片左上角的坐标来表示图片的位置。我们判定如果col\_addr和row\_addr在该显示的区域内，我们就给输入赋值成(col\_addr-x)\*图片高度+(row\_addr-y),其中x，y即为图片左上角的坐标，col\_addr和row\_addr为vga的扫描横坐标和纵坐标。

在给输入赋值完毕后，我们就可以选择合适的位置让图片显示。要想让图片显示，只需将ip核的地址赋给vga\_data即可。



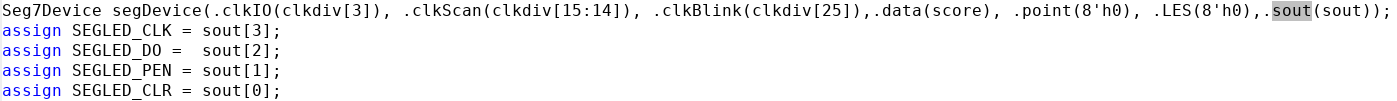


## PS/2键盘部分

PS2键盘传输的数据会在data\_out中输出，其中键盘的JKL三个键分别控制loopy的加速减速转换运动方向。具体代码请见10.2节。

## 七段数码管部分

将得分更新到score中然后在七段数码管中显示，具体代码请见10.3节。



## 露比显示部分

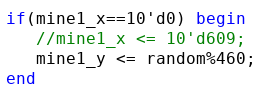
在top里直接调用loopy的ip模块，详见top模块。

## 障碍物显示部分

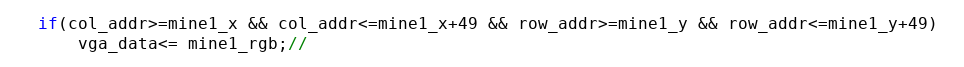
### 苦力怕地雷部分

本实验中空中障碍物主要有苦力怕地雷，主要用以区分图片左上角位置的差别，碰撞逻辑等环节是类似的。下面进行说明：

输入与输出部分：







逻辑判断：

扫描输出图像的方法已经在实验原理中叙述，在此不再赘述。完整源代码请见10.4.1节。

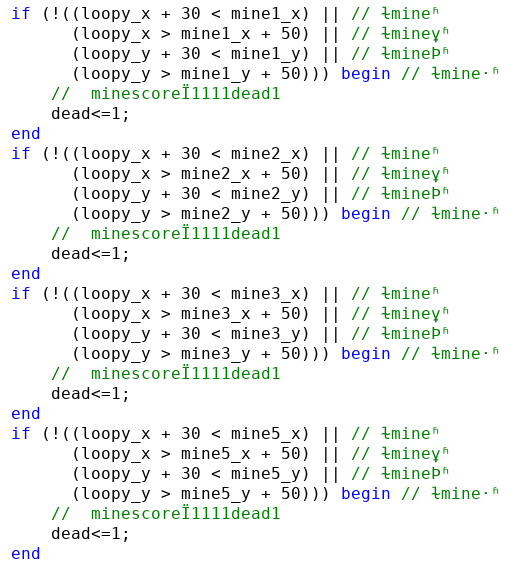
### 激光门部分

与苦力怕地雷的实现方法类似，从最右侧刷新然后随机指定纵坐标。具体源代码请见10.4.2节。

## top部分（包含碰撞检测与计分与运动状态显示）

top模块主要包含以下三个小模块：碰撞检测，物体的运动与人物的移动。

碰撞检测部分的核心代码如下。在扫描到角色所在的固定区域时，以下代码触发。



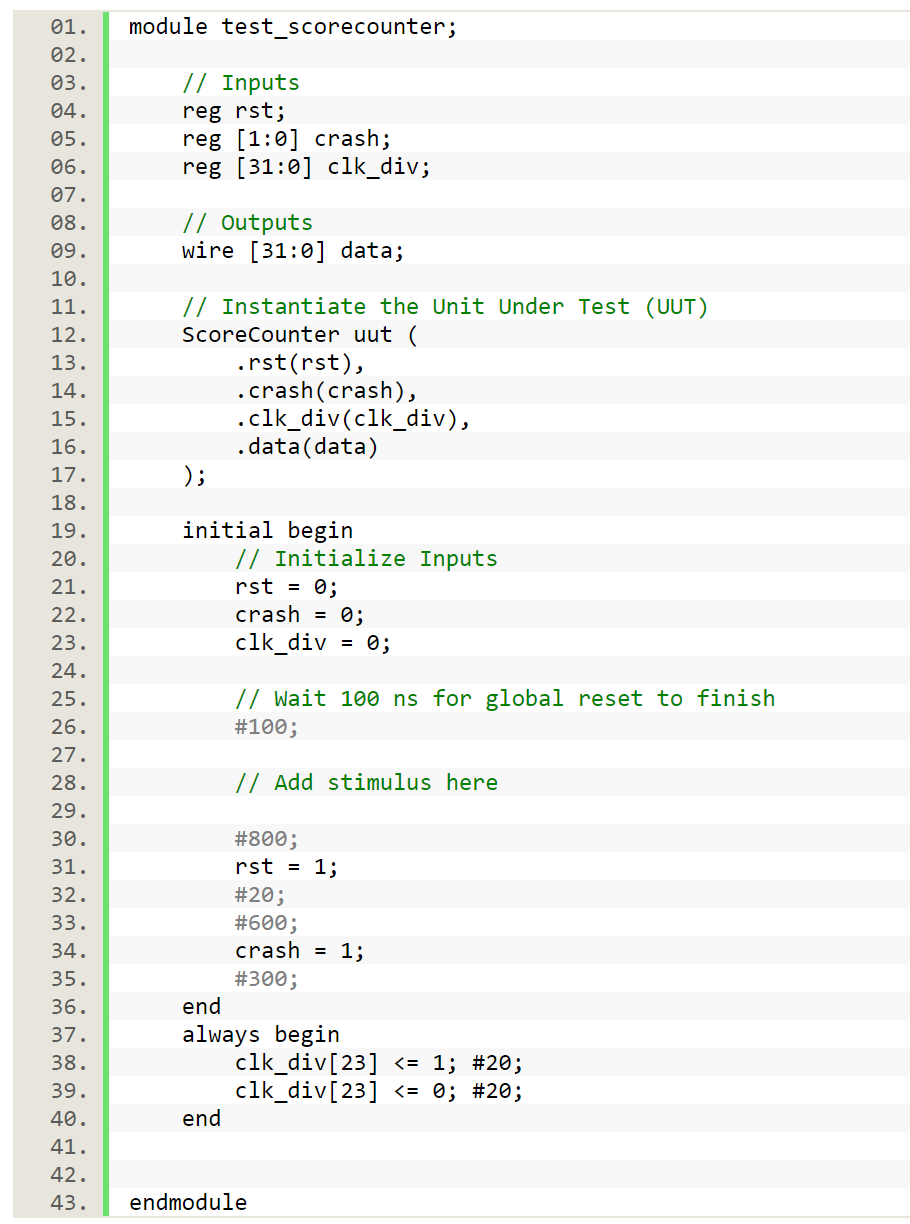
露比的移动部分主要由下方代码实现。

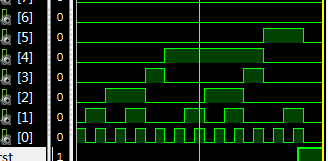
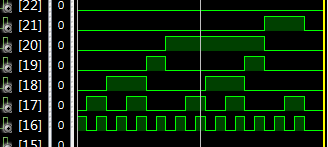
ucf引脚部分

此部分详见源代码10.6节。

# 模拟仿真

本次实验中，主要在设计开始阶段对七段数码管得分部分进行了模拟。具体代码与结果如下。





如上图显示，两位玩家的计分器数据部分均像本次课程中Lab12的数字时钟一样，可以达成加法与进位的任务。

由于本次的设计中，代码的逻辑部分相对比较复杂，而且涉及到 IP 核内存的读取，采用仿真波形图的方式不太直观。所以所有的调试都通过生成 bit 文件烧录之后，通过 VGA 输出，具体实践来调试。

# 结果演示与分析

此部分主要通过视频呈现。在报告中，我们将着重于一些亮点来进行直观地展示。

1. 碰撞检测

碰到苦力怕变为Gameover界面：

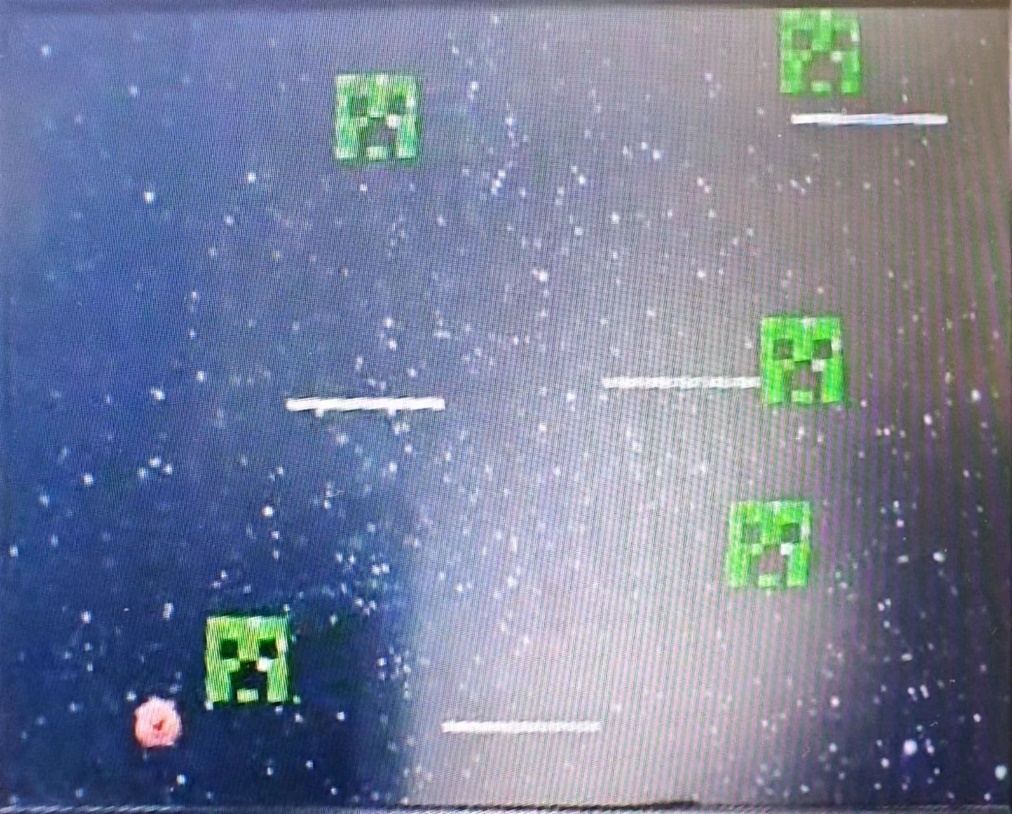


图7-1碰撞前



图7-2碰撞后

1. 计分效果

本游戏计分基于存活时间长度和接触激光门得分，原理与Lab12数字时钟类似。



图7-3 计分画面

1. 键盘控制

本实验中，键盘的JKL可以控制小人的跳跃。结束后任意键可以起到重新开始/清楚一切状态的效果。连续动作演示图片省略，于附视频中共同演示。

# 实验改进与不足

报告的最后我们想要梳理一下我们在实验中遇到的困难以及解决方法，大家引以为戒：

1. coe文件制作：

由于我们一开始第一张图片为32位bmp图片，所以我们使用了32位bmp转换为coe的cpp程序，然后后续的24位bmp图片并不能在该程序进行处理，而我们紫萼的24位程序又出现了问题，所以我们决定将所有选中图片都转换为32位bmp进行处理，然而windows自带的画图软件只能进行处理另存为24位bmp图，经过一番折腾，最后我们选择了一个折中的方法，用选中的图片进行windows截图，用PS进行大小裁剪，解决了所有问题

1. VGA显示

VGA显示就是老生常谈的图片方向问题，详见代码

1. 边界判断问题

人物图像触碰到到上下边界有几率进入屏幕外部转一个循环再回来，这个怀疑是每次步长（速度speed）加速为2时时序判断和系统时序判断冲突，修改了之后还是偶尔会有问题，不好解决。

1. 开局初始化问题：

写了游戏开局就会将dead置为1，然后显示gameover，未查明原因但通过放到不出问题的位置解决。

尽管我们有很多收获与成果，但也同样留下了一些遗憾。

1. 游戏逻辑设计部分

在设计时，我们存在一些考虑不周到的地方，例如进入黑洞是利用了系统的特性，不好控制与量化，然后物体的横向移动在屏幕左侧消失时散发的像素点不知道造成原因，但不影响游戏进行，时间较短、准备不充足，还请老师谅解。

1. 游戏效果部分

在图片显示的效果上，我们的遗憾还包括没有增加开始的图、游戏音效、多种游戏人物、多种游戏难度等等，如果时间更充足一些的话，我们应当能够做到这些。

1. 实验感悟

完成这个实验的过程中，我们解决问题的历程也是十分值得回味的。还记得最开始导入露比图片的时候，总会有图片显示错位的问题，导致露比脸部产生错切，然后我们通过研究解决之后，后面切换游戏人物时又出现了新的错误，以及当时写物体循环移动边缘判断初始化时与系统边缘判断的冲突，我们选择了接受系统的边缘判断循环。就像这个“系统边缘判断”的问题一样，我们遇到了不少很难寻找原因，很难修改的问题，但是我们解决了很大一部分，通过一些方法规避了一部分，最后完整实现游戏效果时还是非常欣喜的，这也许就是硬件课不同于软件课的感受所在。

# 成员分工

汪珉凯：负责游戏逻辑部分的VGA和逻辑代码实现与报告中相关部分的撰写（50%）

岳晟稼：负责图片处理部分以及游戏移动逻辑与报告中相关部分的撰写（50%）

# 参考资料与源代码

## bmp转coe部分

1. #define \_CRT\_SECURE\_NO\_DEPRECATE
2. #include <windows.h>
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <string.h>
6. FILE\* outfp;
7. int carry2;
8. int BMPtoCOE(char \*BMPfilename, int isfinished)
9. {
10. FILE \*fp;
11. if ((fp = fopen(BMPfilename, "rb")) == NULL)
12. {
13. printf("地址错误，无法打开\n");
14. return 1;
15. }
16. char bm[2];
17. fseek(fp, sizeof(BITMAPFILEHEADER), 0);
18. BITMAPINFOHEADER head;
19. fread(&head, sizeof(BITMAPINFOHEADER), 1, fp);
20. int picwidth = head.biWidth, picheight = head.biHeight;
21. short ind;
22. fseek(fp, 0L, 0);
23. fread(bm, 2, 1, fp);
24. fseek(fp, 0x1CL, 0);
25. fread(&ind, 2, 1, fp);
26. if (bm[0] != 'B' || bm[1] != 'M' || ind != 32)
27. {
28. printf("图片格式不正确\n");
29. return 1;
30. }
31. if (picheight < 0) picheight = -picheight;
32. fseek(fp, 0x36L, 0);
33. printf("图片宽\*高：%d\*%d\n", picwidth, picheight);
34. int buf = (picwidth \* 3 % 4) ? 4 - (picwidth \* 3) % 4 : 0;
35. char \*tmp = (char\*)malloc(sizeof(char)\*buf);
36. int i, j;
37. unsigned char r, g, b, t;
38. for (i = 0; i < picheight; i++)
39. for (j = 0; j < picwidth; j++)
40. {
41. fread(&b, 1, 1, fp);
42. fread(&g, 1, 1, fp);
43. fread(&r, 1, 1, fp);
44. fprintf(outfp, "%3x%x%x", (int)(b / 256.0 \* 16), (int)(g / 256.0 \* 16), (int)(r / 256.0 \* 16));
45. if (i == picheight - 1 && j == picwidth - 1 && isfinished)
46. fprintf(outfp, ";");
47. else
48. fprintf(outfp, ",");
49. carry2++;
50. if (carry2 == 16)
51. fprintf(outfp, "\n"), carry2 = 0;
52. fread(&t, 1, 1, fp);
53. }
54. free(tmp);
55. fclose(fp);
56. return 0;
57. }
58. int main()
59. {
60. outfp = fopen("C:/Users/l/Desktop/steve.coe", "w");
61. int i;
62. char s[40] = "C:/Users/l/Desktop/steve.bmp";
63. //   fprintf(outfp, "memory\_initialization\_radix=16;\nmemory\_initialization\_vector =\n");
64. fprintf(outfp, "memory\_initialization\_radix=16;\nmemory\_initialization\_vector =\n");
65. BMPtoCOE(s, 1);
66. system("pause");
67. return 0;
68. }

## 10.2 TOP代码展示：

由于我们将代码展现在TOP代码中，我们决定在top中进行各个模块注释以及分块的写作：PS/2键盘部分，七段数码管部分，苦力怕地雷部分，激光门部分，露比部分

module top(

    input clk,        // 输入时钟信号

    input rstn,       // 输入复位信号（负逻辑）

    input ps2\_clk,    // PS2时钟信号

    input ps2\_data,   // PS2数据信号

    output hs,        // 水平同步信号

    output vs,        // 垂直同步信号

    output [3:0] r,    // 红色输出信号

    output [3:0] g,    // 绿色输出信号

    output [3:0] b,    // 蓝色输出信号

    output SEGLED\_CLK, // 7段LED时钟信号

    output SEGLED\_CLR, // 7段LED清零信号

    output SEGLED\_DO,  // 7段LED数据输出信号

    output SEGLED\_PEN  // 7段LED使能信号

);

reg dead = 0; // 游戏状态寄存器

reg [31:0] clkdiv; // 时钟分频寄存器

reg [11:0] vga\_data; // VGA数据寄存器

wire [9:0] col\_addr; // VGA列地址信号

wire [8:0] row\_addr; // VGA行地址信号

wire key\_down; // 按键按下信号

wire [7:0] keyboard; // 键盘输入数据信号

reg [31:0] score = 32'b0; // 得分寄存器

wire [3:0] sout; // 得分输出信号

reg[8:0] random = 9'b101011010; // 随机数寄存器

always @(posedge clk) begin

    clkdiv <= clkdiv + 1'b1; // 每个上升沿时钟时，时钟分频器加1

end // 每个时钟周期时，时钟分频器加1

reg [8:0] loopy\_x = 10'd80; // 初始小人的X坐标

reg [9:0] loopy\_y = 10'd240; // 初始小人的Y坐标

reg [8:0] door1\_x = 10'd200; // 门1的X坐标

reg [9:0] door1\_y = 10'd300; // 门1的Y坐标

// ... 以此类推，定义了门、地雷等的坐标

reg [8:0] door2\_x = 10'd400;

reg [9:0] door2\_y = 10'd200;

reg [8:0] door3\_x = 10'd300;

reg [9:0] door3\_y = 10'd400;

reg [8:0] door4\_x = 10'd520;

reg [9:0] door4\_y = 10'd100;

reg [8:0] mine1\_x = 10'd150;

reg [9:0] mine1\_y = 10'd400;

reg [8:0] mine2\_x = 10'd1;

reg [9:0] mine2\_y = 10'd1;

reg [8:0] mine3\_x = 10'd500;

reg [9:0] mine3\_y = 10'd200;

reg [8:0] mine4\_x = 10'd230;

reg [9:0] mine4\_y = 10'd100;

reg [8:0] mine5\_x = 10'd480;

reg [9:0] mine5\_y = 10'd300;

reg [31:0] cnt = 0; // 计数器

reg [31:0] cnt\_score = 0; // 计分计数器

reg wasReady = 0; // 按键准备信号

reg [7:0] last\_key = 0; // 上一次按下的键值

//----------------------------------------------------------

// VGA控制模块

vgac v0 (

    .vga\_clk(clkdiv[1]), // VGA时钟信号，取时钟分频器的第2位

    .clrn(1'b1),          // VGA清零信号，保持高电平

    .d\_in(vga\_data),      // VGA数据输入信号

    .row\_addr(row\_addr),  // VGA行地址信号

    .col\_addr(col\_addr),  // VGA列地址信号

    .r(r),                // VGA红色输出信号

    .g(g),                // VGA绿色输出信号

    .b(b),                // VGA蓝色输出信号

    .hs(hs),              // VGA水平同步信号

    .vs(vs)               // VGA垂直同步信号

); // 实例化VGA模块

// PS2控制模块

PS2 p0(

    .clk(clk),

    .rst(1'b0),

    .ps2\_clk(ps2\_clk),

    .ps2\_data(ps2\_data),

    .data\_out(keyboard),

    .ready(key\_down)

);

//7段数码管显示实例化

Seg7Device segDevice(

    .clkIO(clkdiv[3]),

    .clkScan(clkdiv[15:14]),

    .clkBlink(clkdiv[25]),

    .data(score),

    .point(8'h0),

    .LES(8'h0),

    .sout(sout)

);

// 将sout中的特定位分配给SEGLED信号

assign SEGLED\_CLK = sout[3];

assign SEGLED\_DO = sout[2];

assign SEGLED\_PEN = sout[1];

assign SEGLED\_CLR = sout[0];

//---------------------------------------------------------------

// 游戏逻辑寄存器

reg [2:0] speed = 1;

reg flag = 1; // 1 up, 0 down

// Counters for game timing

always @(posedge clk) begin

    if (cnt < 100\_000\_000) begin

        cnt <= cnt + 1;

    end

    else begin

        cnt <= 0;

    end

    if (cnt\_score < 1000\_000\_000) begin

        cnt\_score <= cnt\_score + 1;

    end

    else begin

        cnt\_score <= 0;

    end

    // 更新游戏随机性的随机数生成器

    if (dead == 0) begin

        random <= {random[7:0], random[7] ^ random[5] ^ random[4] ^ random[3]};

        // Check keyboard input for game control

        if (wasReady == 0 && key\_down == 1 && keyboard != last\_key) begin

            case (keyboard)

                8'h3b: begin

                    speed <= 3'b10; // Set speed based on keyboard input J

                end

                8'h42: begin

                    speed <= 1; // Set speed based on keyboard input K

                end

                8'h4b: begin

                    if (flag == 0)

                        flag <= 1;

                    else

                        flag <= 0;

                end

                default: ;

            endcase

            last\_key <= keyboard;

        end

        else if (wasReady == 0 && key\_down == 1)

            last\_key <= 8'b0;

        wasReady <= key\_down;

        // Collision detection with mines

        if (!((loopy\_x + 30 < mine1\_x) ||

              (loopy\_x > mine1\_x + 50) ||

              (loopy\_y + 30 < mine1\_y) ||

              (loopy\_y > mine1\_y + 50))) begin

            dead <= 1; // Collision with mine1, set dead signal

        end

        if (!((loopy\_x + 30 < mine2\_x) ||

              (loopy\_x > mine2\_x + 50) ||

              (loopy\_y + 30 < mine2\_y) ||

              (loopy\_y > mine2\_y + 50))) begin

            dead <= 1; // Collision with mine2, set dead signal

        end

        if (!((loopy\_x + 30 < mine3\_x) ||

              (loopy\_x > mine3\_x + 50) ||

              (loopy\_y + 30 < mine3\_y) ||

              (loopy\_y > mine3\_y + 50))) begin

            dead <= 1; // Collision with mine3, set dead signal

        end

        if (!((loopy\_x + 30 < mine4\_x) || // If loopy is to the left of mine4

                (loopy\_x > mine4\_x + 50) || // If loopy is to the right of mine4

                (loopy\_y + 30 < mine4\_y) || // If loopy is above mine4

                (loopy\_y > mine4\_y + 50))) begin // If loopy is below mine4

                // Set dead signal and reset score to 1111 when collision with mine4 occurs

                dead <= 1;

        end

        // Initialize mine and door positions based on random values

        if(mine1\_x==10'd0) begin

            mine1\_y <= random%460;

        end

        if(mine2\_x==10'd0) begin

            mine2\_y <= random%460;

        end

        if(mine3\_x==10'd0) begin

            mine3\_y <= random%460;

        end

        if(mine4\_x==10'd0) begin

            mine4\_y <= random%460;

        end

        if(mine5\_x==10'd0) begin

            mine5\_y <= random%460;

        end

        if(door1\_x==10'd0) begin

            door1\_y <= random%460;

        end

        if(door2\_x==10'd0) begin

            door2\_y <= random%460;

        end

        if(door3\_x==10'd0) begin

            door3\_y <= random%460;

        end

        if(door4\_x==10'd0) begin

            door4\_y <= random%460;

        end

        // Handle loopy's movement at screen boundaries

        if(loopy\_y==10'd0) begin

            flag <= 0; // Move upward when reaching the top boundary

        end

        else if(loopy\_y==10'd480) begin

            flag <= 1; // Move downward when reaching the bottom boundary

        end

        // Update positions of mines, doors, and loopy periodically

        if(cnt%2\_000\_000==0) begin

            mine1\_x <= mine1\_x - 2;

            mine2\_x <= mine2\_x - 2;

            mine3\_x <= mine3\_x - 2;

            mine4\_x <= mine4\_x - 2;

            mine5\_x <= mine5\_x - 2;

            door1\_x <= door1\_x - 2;

            door2\_x <= door2\_x - 2;

            door3\_x <= door3\_x - 2;

            door4\_x <= door4\_x - 2;

            // Move loopy vertically based on the flag (upward or downward)

            if(flag == 1) begin

                loopy\_y <= loopy\_y - speed;

            end

            else if(flag == 0) begin

                loopy\_y <= loopy\_y + speed;

            end

            // Collision detection with door1 and increment score

            if (!((loopy\_x + 30 < door1\_x) ||

                (loopy\_x > door1\_x + 99) ||

                (loopy\_y + 30 < door1\_y) ||

                (loopy\_y > door1\_y + 5))) begin

                // Increment score when collision with door1 occurs

                score <= score + 32'b1;

            end

           if (!((loopy\_x + 30 < door2\_x) ||

                (loopy\_x > door2\_x + 99) ||

                (loopy\_y + 30 < door2\_y) ||

                (loopy\_y > door2\_y + 5)))

            begin

                // Increment score if there is a collision with door2

                score <= score + 32'b1;

            end

            // Check collision with door3

            if (!((loopy\_x + 30 < door3\_x) ||

                (loopy\_x > door3\_x + 99) ||

                (loopy\_y + 30 < door3\_y) ||

                (loopy\_y > door3\_y + 5)))

            begin

                // Increment score if there is a collision with door3

                score <= score + 32'b1;

            end

            // Check collision with door4

            if (!((loopy\_x + 30 < door4\_x) ||

                (loopy\_x > door4\_x + 99) ||

                (loopy\_y + 30 < door4\_y) ||

                (loopy\_y > door4\_y + 5)))

            begin

                // Increment score if there is a collision with door4

                score <= score + 32'b1;

            end

        end

            // Check if score is a multiple of 200,000,000

            if (cnt\_score % 200\_000\_000 == 0)begin

                // Increment score by 1 in this case

                score <= score + 32'b1;

            end

        end

            else

            begin

                // Reset the game state if a key is pressed (possibly a restart)

                if (key\_down)begin

                    dead <= 0;

                    // Reset the positions of doors, mines, and character

                    // Set other variables to initial values

                    loopy\_x <= 10'd80;

                    loopy\_y <= 10'd240;

                    door1\_x <= 10'd200;

                    door1\_y <= 10'd300;

                    // ... (similar resets for other doors, mines, and variables)

                    door2\_x <= 10'd400;

                    door2\_y <= 10'd200;

                    door3\_x <= 10'd300;

                    door3\_y <= 10'd400;

                    door4\_x <= 10'd520;

                    door4\_y <= 10'd100;

                    mine1\_x <= 10'd150;

                    mine1\_y <= 10'd400;

                    mine2\_x <= 10'd1;

                    mine2\_y <= 10'd1;

                    mine3\_x <= 10'd500;

                    mine3\_y <= 10'd200;

                    mine4\_x <= 10'd230;

                    mine4\_y <= 10'd100;

                    mine5\_x <= 10'd480;

                    mine5\_y <= 10'd300;

                    score <= 32'b0;  // Reset the score

                    cnt <= 32'b0;    // Reset the counter

                    cnt\_score <= 32'b0;  // Reset the score counter

                    speed <= 1;      // Reset speed

                    last\_key <= 0;   // Reset last key pressed

                    score <= 0;      // Reset score again (possible duplicate, might want to remove)

                end

            end

//--------------------------------------------------------------------------------

             // Memory addresses for different elements on the screen

            reg [18:0] background\_addr;

            wire [11:0] background\_rgb;

            reg [18:0] end\_background\_addr;

            wire [11:0] end\_background\_rgb;

            reg [9:0] loopy\_addr;

            wire [11:0] loopy\_rgb;

            reg [8:0] door1\_addr;

            wire [11:0] door1\_rgb;

            reg [8:0] door2\_addr;

            wire [11:0] door2\_rgb;

            reg [8:0] door3\_addr;

            wire [11:0] door3\_rgb;

            reg [8:0] door4\_addr;

            wire [11:0] door4\_rgb;

            reg [11:0] mine1\_addr;

            wire [11:0] mine1\_rgb;

            reg [11:0] mine2\_addr;

            wire [11:0] mine2\_rgb;

            reg [11:0] mine3\_addr;

            wire [11:0] mine3\_rgb;

            reg [11:0] mine4\_addr;

            wire [11:0] mine4\_rgb;

            reg [11:0] mine5\_addr;

            wire [11:0] mine5\_rgb;

            //---------------------------------------------

            //---------------------------------------

            // Image address calculation logic

            always @(posedge clk) begin

                // Calculate background address based on column and row address within valid screen range

                background\_addr <= (col\_addr >= 0 && col\_addr <= 639 && row\_addr >= 0 && row\_addr <= 479) ? (480 - row\_addr) \* 640 + col\_addr : 0;

                // Calculate end\_background\_addr with similar logic

                end\_background\_addr <= (col\_addr >= 0 && col\_addr <= 639 && row\_addr >= 0 && row\_addr <= 479) ? (480 - row\_addr) \* 640 + col\_addr : 0;

                // Calculate loopy\_addr based on the character's position

                loopy\_addr <= (col\_addr >= loopy\_x && col\_addr <= loopy\_x + 29 && row\_addr >= loopy\_y && row\_addr <= loopy\_y + 29) ? (29 - (row\_addr - loopy\_y)) \* 30 + (col\_addr - loopy\_x) : 0;

                // Calculate door1\_addr, door2\_addr, door3\_addr, door4\_addr based on respective door positions

                door1\_addr <= (col\_addr >= door1\_x && col\_addr <= door1\_x + 98 && row\_addr >= door1\_y && row\_addr <= door1\_y + 4) ? (4 - (row\_addr - door1\_y)) \* 99 + (col\_addr - door1\_x) : 0;

                door2\_addr <= (col\_addr >= door2\_x && col\_addr <= door2\_x + 98 && row\_addr >= door2\_y && row\_addr <= door2\_y + 4) ? (4 - (row\_addr - door2\_y)) \* 99 + (col\_addr - door2\_x) : 0;

                door3\_addr <= (col\_addr >= door3\_x && col\_addr <= door3\_x + 98 && row\_addr >= door3\_y && row\_addr <= door3\_y + 4) ? (4 - (row\_addr - door3\_y)) \* 99 + (col\_addr - door3\_x) : 0;

                door4\_addr <= (col\_addr >= door4\_x && col\_addr <= door4\_x + 98 && row\_addr >= door4\_y && row\_addr <= door4\_y + 4) ? (4 - (row\_addr - door4\_y)) \* 99 + (col\_addr - door4\_x) : 0;

                // Calculate mine1\_addr, mine2\_addr, mine3\_addr, mine4\_addr, mine5\_addr based on respective mine positions

                mine1\_addr<=(col\_addr>=mine1\_x && col\_addr<=mine1\_x+49 && row\_addr>=mine1\_y && row\_addr<=mine1\_y+49)?(49-(row\_addr-mine1\_y))\*50+(col\_addr-mine1\_x):0;

                mine2\_addr<=(col\_addr>=mine2\_x && col\_addr<=mine2\_x+49 && row\_addr>=mine2\_y && row\_addr<=mine2\_y+49)?(49-(row\_addr-mine2\_y))\*50+(col\_addr-mine2\_x):0;

                mine3\_addr<=(col\_addr>=mine3\_x && col\_addr<=mine3\_x+49 && row\_addr>=mine3\_y && row\_addr<=mine3\_y+49)?(49-(row\_addr-mine3\_y))\*50+(col\_addr-mine3\_x):0;

                mine4\_addr<=(col\_addr>=mine4\_x && col\_addr<=mine4\_x+49 && row\_addr>=mine4\_y && row\_addr<=mine4\_y+49)?(49-(row\_addr-mine4\_y))\*50+(col\_addr-mine4\_x):0;

                mine5\_addr<=(col\_addr>=mine5\_x && col\_addr<=mine5\_x+49 && row\_addr>=mine5\_y && row\_addr<=mine5\_y+49)?(49-(row\_addr-mine5\_y))\*50+(col\_addr-mine5\_x):0;

                end

        //------------------------------------------

        //VGA调用

            background a1(.clka(clkdiv[1]),.addra(background\_addr),.douta(background\_rgb));

            end\_background a2(.clka(clkdiv[1]),.addra(end\_background\_addr),.douta(end\_background\_rgb));

            loopy a3(.clka(clkdiv[1]),.addra(loopy\_addr),.douta(loopy\_rgb));

            door a4(.clka(clkdiv[1]),.addra(door1\_addr),.douta(door1\_rgb));

            door a5(.clka(clkdiv[1]),.addra(door2\_addr),.douta(door2\_rgb));

            mine a6(.clka(clkdiv[1]),.addra(mine1\_addr),.douta(mine1\_rgb));

            mine a7(.clka(clkdiv[1]),.addra(mine2\_addr),.douta(mine2\_rgb));

            mine a8(.clka(clkdiv[1]),.addra(mine3\_addr),.douta(mine3\_rgb));

            mine a9(.clka(clkdiv[1]),.addra(mine4\_addr),.douta(mine4\_rgb));

            mine a10(.clka(clkdiv[1]),.addra(mine5\_addr),.douta(mine5\_rgb));

            door a11(.clka(clkdiv[1]),.addra(door3\_addr),.douta(door3\_rgb));

            door a12(.clka(clkdiv[1]),.addra(door4\_addr),.douta(door4\_rgb));

        //-------------------------------------------

        // VGA显示逻辑

        always @(\*) begin

            if (dead == 0) begin // 如果状态为非结束状态

                if (col\_addr >= 0 && col\_addr <= 639 && row\_addr >= 0 && row\_addr <= 479)

                    vga\_data <= background\_rgb; // 显示背景色

                // 门的显示逻辑

                if (col\_addr >= door1\_x && col\_addr <= door1\_x + 98 && row\_addr >= door1\_y && row\_addr <= door1\_y + 4)

                    vga\_data <= door1\_rgb; // 显示门1颜色

                if (col\_addr >= door2\_x && col\_addr <= door2\_x + 98 && row\_addr >= door2\_y && row\_addr <= door2\_y + 4)

                    vga\_data <= door2\_rgb; // 显示门2颜色

                if (col\_addr >= door3\_x && col\_addr <= door3\_x + 98 && row\_addr >= door3\_y && row\_addr <= door3\_y + 4)

                    vga\_data <= door3\_rgb; // 显示门3颜色

                if (col\_addr >= door4\_x && col\_addr <= door4\_x + 98 && row\_addr >= door4\_y && row\_addr <= door4\_y + 4)

                    vga\_data <= door4\_rgb; // 显示门4颜色

                // 矿块的显示逻辑

                if (col\_addr >= mine1\_x && col\_addr <= mine1\_x + 49 && row\_addr >= mine1\_y && row\_addr <= mine1\_y + 49)

                    vga\_data <= mine1\_rgb; // 显示矿块1颜色

                if (col\_addr >= mine2\_x && col\_addr <= mine2\_x + 49 && row\_addr >= mine2\_y && row\_addr <= mine2\_y + 49)

                    vga\_data <= mine2\_rgb; // 显示矿块2颜色

                if (col\_addr >= mine3\_x && col\_addr <= mine3\_x + 49 && row\_addr >= mine3\_y && row\_addr <= mine3\_y + 49)

                    vga\_data <= mine3\_rgb; // 显示矿块3颜色

                if (col\_addr >= mine4\_x && col\_addr <= mine4\_x + 49 && row\_addr >= mine4\_y && row\_addr <= mine4\_y + 49)

                    vga\_data <= mine4\_rgb; // 显示矿块4颜色

                if (col\_addr >= mine5\_x && col\_addr <= mine5\_x + 49 && row\_addr >= mine5\_y && row\_addr <= mine5\_y + 49)

                    vga\_data <= mine5\_rgb; // 显示矿块5颜色

                // 特殊区域的显示逻辑

                if (col\_addr >= loopy\_x && col\_addr <= loopy\_x + 29 && row\_addr >= loopy\_y && row\_addr <= loopy\_y + 29 && loopy\_rgb != 12'hfff)

                    vga\_data <= loopy\_rgb; // 显示特殊区域颜色

            end

            else begin // 如果状态为结束状态

                if (col\_addr >= 0 && col\_addr <= 639 && row\_addr >= 0 && row\_addr <= 479)

                    vga\_data <= end\_background\_rgb; // 显示结束状态下的背景色

            end

        end

        //---------------------------------------------

        endmodule

## 10.3 ucf引脚文件

    NET "clk"                                        LOC = AC18      | IOSTANDARD = LVCMOS18 ;

    NET "rstn"                                       LOC = W13       | IOSTANDARD = LVCMOS18 ;

    NET "ps2\_clk"                                    LOC = N18       | IOSTANDARD = LVCMOS33 | PULLUP;

    NET "ps2\_data"                                   LOC = M19       | IOSTANDARD = LVCMOS33 | PULLUP;

    NET "hs"                                         LOC = M22       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "vs"                                         LOC = M21       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "b[0]"                                       LOC = T20       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "b[1]"                                       LOC = R20       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "b[2]"                                       LOC = T22       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "b[3]"                                       LOC = T23       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "g[0]"                                       LOC = R22       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "g[1]"                                       LOC = R23       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "g[2]"                                       LOC = T24       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "g[3]"                                       LOC = T25       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "r[0]"                                       LOC = N21       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "r[1]"                                       LOC = N22       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "r[2]"                                       LOC = R21       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "r[3]"                                       LOC = P21       | IOSTANDARD = LVCMOS33 | SLEW = FAST ;

    NET "SEGLED\_CLK"                                 LOC = M24       | IOSTANDARD = LVCMOS33 ;

    NET "SEGLED\_CLR"                                 LOC = M20       | IOSTANDARD = LVCMOS33 ;

    NET "SEGLED\_DO"                                  LOC = L24       | IOSTANDARD = LVCMOS33 ;

    NET "SEGLED\_PEN"                                 LOC = R18       | IOSTANDARD = LVCMOS33 ;