

计算机逻辑设计基础课程设计 小游戏 Flappy Mario 设计报告

Group: 费姚瑞 (3220103113)

仇国智 (3220102181)

Date: 2023.12.22

一、背景介绍

Flappy bird 是一款 2D 横版闯关游戏，画面中一只小鸟始终向右飞行，玩家可以通过一直点击来让小鸟不断向上飞行，一旦停止点击小鸟就会开始下落。这个游戏要求玩家利用这一机制使小鸟保持适当的高度来通过不同高度的障碍物。

在本次课程设计中，我们小组将利用 Verilog HDL 语言和相关实验环境实现基于 Flappy bird 改编的 Flappy Mario 的运行和游玩。

Flappy_Mario 将在 Flappy_bird 的基础上加入双人游玩模式并增加一些细节内容。

二、设计说明

A. 设计开发环境

实验平台：Sword Kintex7

开发环境：Xilinx ISE、Xilinx Vivado

硬件描述语言：Verilog HDL

B. 输入输出交互选择

输入：Sword 平台上 clk 时钟信号与 SW[15]开关、PS2 键盘上 space 按钮及上下方向键

输出：Sword 平台上 VGA 显示器以及 4 位 7 段数码管

C. 游戏状态寄存器

Flappy Mario 游戏本质也是一个有限状态机，游戏中主要的状态寄存器如下：

1. status[1:0]: 01 为开始页面单人模式，10 为开始页面双人模式，00 为游戏中单人模式，11 为游戏中双人模式。
2. score[15:0]: 储存游戏中获得的分数
3. pipe1~pipe3[31:0]: pipe*储存上下一对水管的状态，一共三对

31~28	27~20	19~10	9~0
保留	水管间距	水管水平坐标	上面水管高度

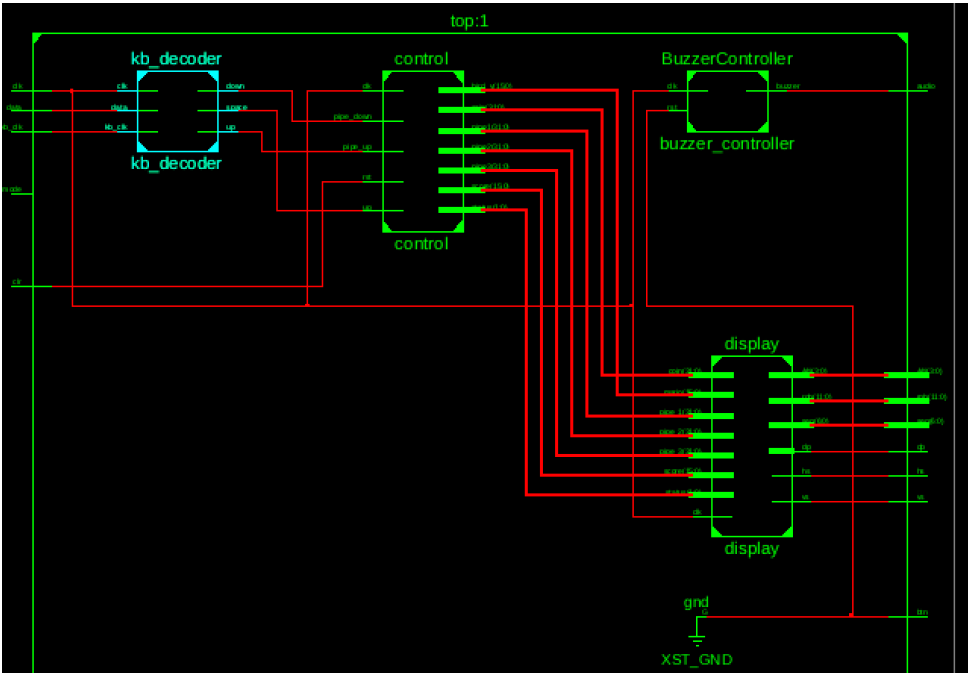
4. Mario[15:0]: 储存角色状态

15	14~10	9~0
状态 1 飞行，0 下降	保留	角色高度

5. coin[31:0]: 储存金币状态

31	30: 20	19: 10	9: 0
金币是否存在	保留	金币 y 坐标	金币 x 坐标

D. 总模块结构



E. 控制模块设计——Control Module

实现 Flappy Mario 这一小游戏的核心模块就是控制模块，即读入玩家的操作、将其反馈到游戏中并结算游戏状态。

Flappy Mario 这一游戏主要包括两种游玩模式，第一种为单人模式游玩，其游玩规则与 Flappy bird 基本一致；第二种则为双人模式游玩，在单人模式的基础上，双人模式将增加第二个玩家的操作，第二个玩家可以控制屏幕最右侧（Mario 在屏幕左侧）的障碍物高度，从而为第一名玩家制造更加复杂的障碍物，从而提高第一名玩家的游戏难度。

我们对这一小游戏的交互想法很简单，玩家的输入需要三个：一是控制，Mario 向上飞的 space 按钮，二是控制游戏是游玩状态还是初始状态的

SW[15]开关，三是用于第二名玩家控制障碍物高度的上下方向键。因此读入玩家的操作只需要三个一位的控制信号（up、rst、pipe_up 和 pipe_down）即可储存。

而对于游戏某一时刻的状态确定，我们需要以下信息：

1. clk: Sword 平台提供的 clk 时钟信号
2. fail: 游戏是否处于游玩状态，在拨动开始开关后游戏便开始游玩状态，倘若 Mario 撞到了障碍物上，则游戏失败，退出游玩状态。
3. rst: 控制游戏启动及重启，倘若 rst 为 1 则将游戏状态清 0，游戏回到起始页面。
4. up: 记录玩家是否按下了控制 Mario 飞起的按钮，若为 1 则 Mario 飞起
5. score: 记录本局游戏至此的分数，Mario 每成功穿过一个障碍物则加一分，Mario 每吃到一个金币则分数加二分，初始为 0 分。
6. bird_y: 记录当前状态下 Mario 的高度位置（以 Mario 左下角为基准点）以及 Mario 的行动状态（飞起与滑翔）
7. pipe: 记录当前障碍物（管子）的横坐标、高度（以管子左上角为基准点）以及这一建筑物供 Mario 穿过的间隙，屏幕中同时只出现三个障碍物。
8. coin: 记录当前金币的横坐标、纵坐标以及是否存在（若被 Mario 吃掉或从屏幕最左边离开则算作不存在）
9. pipe_up 与 pipe_down: 记录玩家二是否按下控制障碍物高低的按钮，若为 1 则使管子相应的上升或下降

而对于控制模块的实现步骤如下：

1. 首先调用 clk_20ms 模块，将 clk 周期放大为 20ms，并以 clk_20ms 输出的时钟作为驱动时钟驱动以下步骤
2. 检测 rst 的值，倘若 rst 的值为 0，说明此时游戏尚未开始，游戏应当显示初始页面。
3. 在这一页面，玩家可通过按 PS2 键盘上的上下方向键，也就是 pipe_up、pipe_down 信号来更改接下来的游玩模式（单人、双人），在代码中将用 status 二位变量记录，status=1 表示此时在初始页面选

择了单人模式，`status=2` 表示此时在初始页面选择了双人模式，`status` 默认为 1，也就是默认为单人模式。

4. 同时在初始页面游戏处于准备状态，会在固定的三个位置生成随机高度的 `pipe`，也就是对 `pipe_x` 赋初值，并对 `pipe_y` 随机赋值，随机的范围为 `pipe_head~310-pipe_head`，`pipe_head` 指的是水管一端的固定贴图，水管的长度至少要保证能使两端的端点贴图能完整显示，310 的值是 480（显示屏垂直分辨率）-170（水管间距最大值）得到的。同时需要随机生成上下水管的间距大小，也就是对 `gap` 随机赋值，范围为 120~170。
5. 在初始页面还要对金币的位置进行初始化，也就是对 `coin[19:10]` 随机赋值，范围为 20~440-`coin_length`，`coin_length` 为金币贴图的边长。同时还需要对 Mario 的飞行状态、位置进行初始化，也就是对 `bird_y[15]` 赋值为 0、`bird[14:0]` 赋值为 240（显示屏垂直分辨率的一半）。随机数的生成将调用 `clk_div` 模块将 `clk` 计数，再将当前时间的 `clk_div` 根据指定范围取模从而获得对应的伪随机数。
6. 在初始页面选择了游玩模式后，可以拨动 `SW[15]` 进入游戏状态，此时检测到 `rst` 的值为 1，则进入游戏状态。此时对 `status` 的值进行修改，倘若 `status` 的值为 1，则将其赋值为 0，表示单人模式正在游玩中；若 `status` 的值为 2，则将其赋值为 3，表示双人模式正在游玩中。
7. 游戏状态下，分别检测 `up`、`fail` 和 `longpress` 信号，`longpress` 信号用于检测是否长按，若 `up` 先前为 1，则 `longpress` 亦为 1 直到 `up` 变为 0。
 - a) 当 `up` 为 1（玩家按下 `space`）、`fail` 为 0（游戏未结束）且 `longpress` 为 0（玩家刚刚按下 `space`），则将 `bird_flying` 赋值为 10（表示 Mario 将持续飞行 10 个时钟周期），将 `bird_y[15]` 赋值为 1（表示 Mario 正在飞行），并将 `longpress` 赋值为 1（防止玩家长按 `space`）。
 - b) 当 `bird_flying` 大于 0（Mario 还在惯性飞行）且 `fail` 为 0（游戏未结束），此时将 `bird_y` 减少 `bird_flying` 的值，相当于 Mario 向上飞了相当于 `bird_flying` 大小的高度，这样可以使 Mario 的飞

行速度先大后小，从而增强游戏手感；然后使 `bird_flying` 的值减一。

c) 当 `bird_flying` 小于等于 0 (Mario 不再飞行) 或 `fail` 为 1 (Mario 撞上障碍物，游戏结束) 时，时 `bird_y` 增加 4，相当于每个时钟周期 Mario 下降 4 个像素点。

d) 在此之外，对 `up` 进行判断，若 `up` 为 0，则令 `longress` 值变回 0

8. 游戏状态下，检测 `fail` 的值，若 `fail` 为 0，游戏未结束，此时 Mario 仍要向右飞行，也就是每个时钟周期将 `pipe1_x`、`pipe2_x`、`pipe3_x` 以及 `coin[9:0]` 减少 2，相当于 Mario 向右飞行了 2 个像素点。

9. 游戏状态下，检测 `pipe1_x`、`pipe2_x`、`pipe3_x` 的值，若 `pipe_x` ≤ 2 (管子将要移出屏幕左端)，则将 `pipe_x` 重置为 640 (屏幕水平分辨率)，并重新利用 `clk_div` 对 `pipe_y` 进行重新的随机初始化。

10. 双人模式游戏状态下，检测 `pipe_up` 和 `pipe_down` 信号，倘若 `fail` 值为 0 (游戏未结束)，且 `status` 值为 3 (双人模式游玩中)，则若 `pipe_up` 信号为 1，则使屏幕最右侧的管子高度升高，若 `pipe_down` 信号为 1，则使屏幕最右侧的管子高度降低。设置 `cnt` 变量，`cnt` 初始值为 3，用于记录当前屏幕最右侧的管子是几号管子，当 `x` 号管子从屏幕左侧离开在右侧出现时，`cnt` 将被赋值为 `x`，再利用 `case` 结构就可以根据 `cnt` 的值使 `pipe_up` 和 `pipe_down` 的效果作用于屏幕最右侧的管子。

11. 游戏状态下，检测 `coin[31]` 和 `coin[9:0]` 的值，倘若 `coin[31]` == 0 (金币被 Mario 吃掉) 或是 `coin[9:0]` ≤ 0 (金币从屏幕左边移出)，将 `coin[9:0]` 的值重置为固定范围的随机值 (范围在最右侧管子右边缘的右侧 0~20 个像素点，最右侧管子由 `cnt` 的值确定，避免出现金币和 `pipe` 重合的问题)，`coin[31]` 重置为 1，`coin[19:10]` 的值重置为固定范围的随机值。

12. 游戏状态下，分别检测对于三根 `pipe` 的表达式 $((\text{bird_y}[14:0] \leq \text{pipe_y}) \parallel (\text{bird_y}[14:0] + \text{bird_height} \geq \text{pipe_y} + \text{gap1})) \& \& (\text{bird_x} \leq \text{pipe_x} + \text{pipe_width}) \& \& (\text{bird_x} + \text{bird_width} \geq \text{pipe_x})$ 的值，若有一个 `pipe` 对

应表达式值为 1，说明 Mario 撞到了障碍物，则游戏失败，fail 赋值为 1。

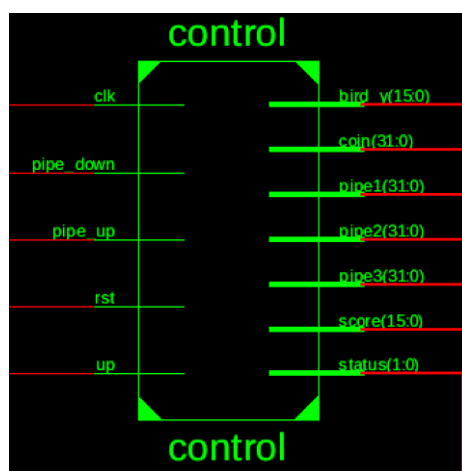
13. 游戏状态下，检测 fail 的值以及 pipe+pipe_width 和 bird_x 的值，若 fail=0 且 pipe+pipe_width==bird_x，说明此时 Mario 成功通过了一个障碍物，则给 score 加一

14. 游戏状态下，在 fail==0 的前提下检测表达式

coin[31]&&((bird_y[14:0]<=coin[19:10]+coin_length)&&(bird_y[14:0]+bird_height>=coin[19:10]))&&(bird_x<=coin[9:0]+coin_length)&&(bird_x+bird_width>=coin[9:0]))的值，若表达式值为 1，说明此时 Mario 成功吃到了金币，则给 score 加二

15. 最终将 gap、pipe_x、pipe_y 合并为 pipe 输出

为了保证游戏的合理性，此模块中需要添加 clk_100ms 模块用于调整游戏速度，由于采用 PS2 键盘输入，不需要防抖模块。若要修改游戏运行速度则修改其中参数即可。



F. 显示模块设计——Display Module

实现这一小游戏的另一重要模块便是显示模块，这一模块需要接收 control 模块运算得到的游戏状态并将其通过 VGA 可视化输出。

模块输入：

1. clk：系统时钟
2. status, pipe, Mario, score, coin：游戏状态输入

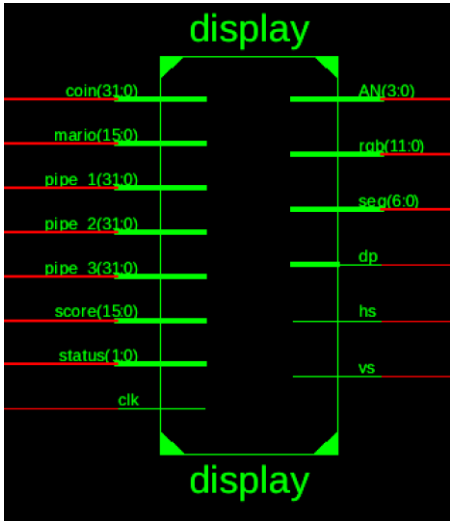
模块输出：

1. seg[6:0]:七段数码管输出
2. dp: 数码管小数点输出
3. AN[3:0]: 数码管共阳使能输出
4. rgb[11:0]: VGA12 位色彩输出
5. hs: VGA 水平同步信号
6. vs: VGA 垂直同步信号

模块内的主要组成部分如下:

1. 首先使用取余运算与截断除法将输入的十六进制分数转换为 BCD 码的十进制数据, 然后调用 DispNum 模块将 BCD 码转换为数码管编码 (seg, dp, AN) 输出。
2. 调用 VGA 接口模块 vga _sync, 模块输出当前对应需要输出的绝对坐标和 VGA 同步信号 hs, vs。注意为了使得 VGA 输出帧率为 60 帧, 调用 clk 分频模块 clkdiv, 输入周期为 25MHZ 的时钟。
3. 我们调用 Mario_rom 等一系列用于储存图像信息的 rom IP 核, 并根据当前绝对坐标和各个显示区域基准点坐标计算出相对坐标输入 rom, 获得 color_Mario 等当前区域的对应像素值。
4. 显示图像大多是不规则平面图形, 而矩形便于显示。为了便捷, 所有图像区域均由不规则图形补全为矩形, 设置透明色 12'h00f 填充补全区域像素点。
5. 为了实现金币转动的效果, 设置寄存器 coin_state[1:0] 以储存金币转动角度的状态, 每隔一定周期进行更新, 通过 coin_state 选择四幅金币旋转状态图中的一张进行显示. 与之相似的还有角色 Mario 的显示, 通过 Mario[15]来选择上升与下降图像。
6. 根据当前绝对位置坐标, 和各个显示区域的位置的大小, 我们使用 assign 语句和逻辑语句, 来确定像素点是否位于标题文本、模式文本、高亮区域、管道头部、管道躯干、角色 Mario、金币图层的区域范围内, 及该区域对应像素点是否为透明色, 并使用 isMario 等一系列 wire 来表明结果。
7. 最后根据图层的覆盖关系和使能, 使用三元运算符进行图层融合生

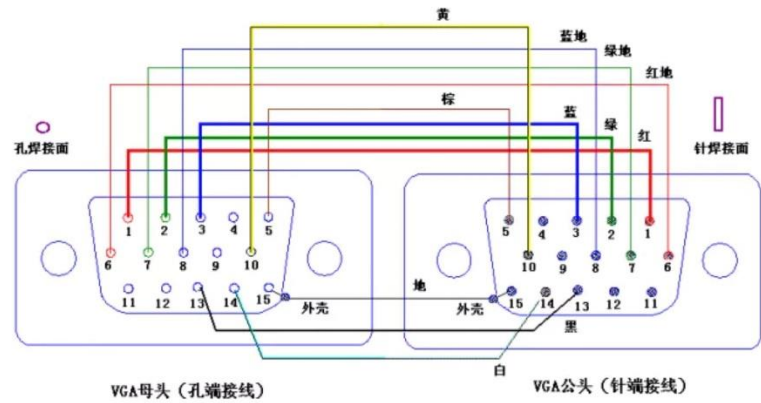
成最终的像素点。其中，为了便于根据 state[1:0]使能起始页面，我们先融合了标题文本、模式文本、高亮区域生成起始页面图层 color_start。



G. VGA 接口模块——vga_sync

VGA（Video Graphics Array）是一种模拟电脑视频接口标准，旨在连接显示器或电视设备，以便接收和显示来自电脑的图形信息。

- 1. VGA 使用模拟方式传输信号。其 15 针接口能分别传输红色、绿色、蓝色三种基色的模拟信号，以及相应的水平和垂直同步信号。
SWORD 实验板会根据 12 位颜色信号引脚的输入，在蓝绿红线上输出对应的电压强度。



VGA接线图

注：7根线中的红绿蓝需要焊屏蔽层
不用线空着不接
外壳接外屏蔽线

1	2	3	4	5	6	7	8	9	10
红基色	绿基色	蓝基色	地址码	自测试	红地	绿地	蓝地	保留	数字地
11	12	13	14	15					
地址码	地址码	行同步	场同步	地址码					

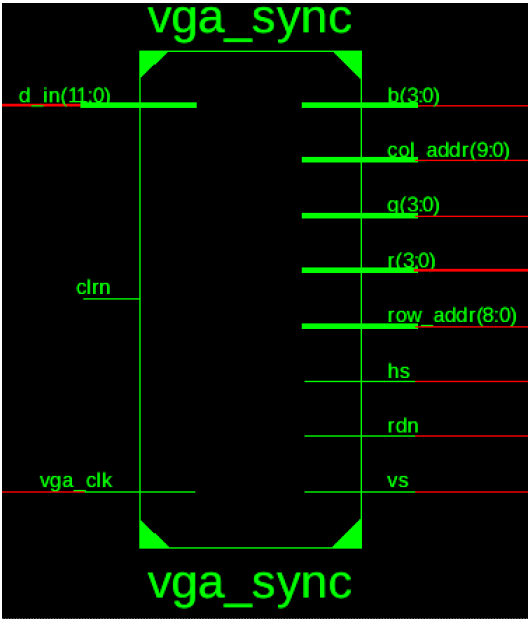
- VGA 通过行场同步信号来确定输出像素点的位置。行同步信号标志着视频扫描线的开始。每一行像素在显示时都会有一次行同步信号拉高，通过这个信号，显示器知道何时开始新的一行的扫描。场同步信号标志着一帧图像的开始。当显示器接收到场同步信号后，它知道一个新的画面开始，显示器即返回到屏幕的左上角，开始绘制新的一帧图像。行场同步信号时序如下，其中有效图像区间输出像素点。



VGA 可以显示不同分辨率的图像，不同分辨率及帧率对应参数如下：

显示模式	时钟 (MHz)	行同步信号时序(像素)							场同步信号时序(行数)						
		同步	后沿	左边 框	有效图 像	右边 框	前沿	行扫描周 期	同步	后沿	上边 框	有效 图像	底边 框	前沿	场扫描周 期
640x480@60	25.175	96	40	8	640	8	8	800	2	25	8	480	8	2	525
640x480@75	31.5	64	120	0	640	0	16	840	3	16	0	480	0	1	500
800x600@60	40.0	128	88	0	800	0	40	1056	4	23	0	600	0	1	628
800x600@75	49.5	80	160	0	800	0	16	1056	3	21	0	600	0	1	625
1024x768@60	65	136	160	0	1024	0	24	1344	6	29	0	768	0	3	806
1024x768@75	78.8	176	176	0	1024	0	16	1312	3	28	0	768	0	1	806
1280x1024@60	108.0	112	248	0	1280	0	48	1688	3	38	0	1024	0	1	1066

3. 最后我们进行 vga_sync 模块的设计，根据输入的 25MHZ 时钟和 640*480@60 行场同步信号的时序模式，输出 hs,vs 行场同步信号，并在有效图像区间根据时钟和输入颜色值，同步更新输出的绝对坐标（x,y）和 12 位颜色值。



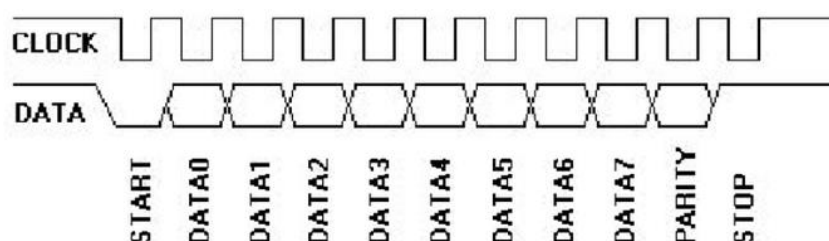
H. PS2 键盘输入模块——keyboard、kb_decoder

SWORD 板与键盘是通过 USB 相连接，SWORD 板内部将其转换为 PS2 协议输入。

1. PS2 主要用于设计的两个引脚为输入时钟 kb_clk,移位输入数据 data。主机读取 PS2 的 data 均在 kb_clk 下降沿发生。如下为 data 读取时 11 位数据的时序结构。

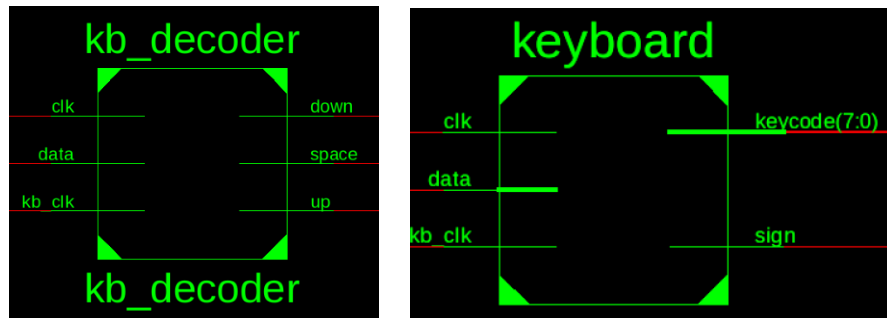
0	起始位，总为 0
---	----------

1~8	数据位，低位在前
9	校验位，奇校验
10	停止位，总为 1



2. 在模块的输入端，我们可以看到有三个输入信号，包括系统时钟 `clk`，键盘时钟 `kb_clk` 和键盘数据 `data`。在输出端，我们可以看到有两个输出信号，包括键盘码 `keycode` 和松开/按下信号 `sign`。注意为了达到同步赋值的效果，我们使用寄存器 `kb_clk_old` 存储键盘时钟的历史值，通过 `kb_clk_old` 检测 `kb_clk` 的上升沿，并在 `clk` 的上升沿进行同步的赋值和更新。
3. 在模块内部，定义了一些寄存器，包括状态寄存器 `state`，临时键盘码寄存器 `code_temp`，计数寄存器 `cnt`，确认寄存器 `confirm` 和键盘时钟历史寄存器 `kb_clk_old`。`state` 用于表示当前的状态，包括等待（00）、输入（01）、确认（10）和结束（11）。`code_temp` 用于暂存正在接收的键盘码。`cnt` 用于计数接收到的有效位数。`confirm` 用于存储奇偶校验位。
4. 在模块的主体部分，使用 `always` 语句在每个系统时钟上升沿更新寄存器和输出信号。首先，更新 `kb_clk_old`。然后，根据 `state` 的值执行不同的操作。在等待状态，如果检测到键盘时钟的下降沿，并且数据位为 0，表示开始接收一个新的键盘码，此时将 `state` 设置为输入状态，将 `confirm` 清零，将 `sign` 设置为 0 或 1，（`sign` 取决于前一位是否为 F0 即松开的前一半状态码）。在输入状态，如果检测到键盘时钟的下降沿，将 `cnt` 加 1，如果 `cnt` 的所有位都为 1，表示已经接收到 8 位数据，此时将 `state` 设置

为确认状态，将 cnt 清零，将 confirm 取反，将 code_temp 左移并加上数据位。在确认状态，如果检测到键盘时钟的下降沿，并且确认位与数据位不同，表示接收到的键盘码正确，此时将 state 设置为结束状态，否则将 state 设置为等待状态，将 code_temp 清零。在结束状态，将 state 设置为等待状态，将 keycode 设置为 code_temp 的值。



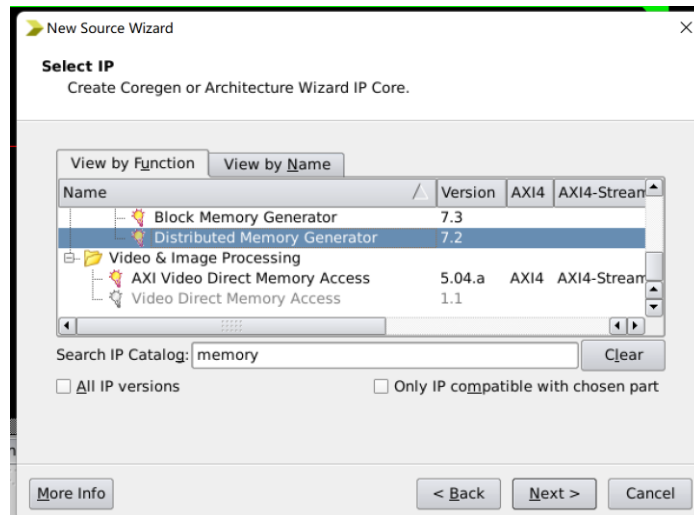
I. ROM 寄存器——IP 核设计

ROM 全称为只读存储器（Read-Only Memory），是一种在断电后数据仍然不会丢失的存储器。ROM 有几种不同的类型，包括掩模 ROM（Mask ROM，被程序预先硬编码，不能被用户改写）、可编程 ROM（PROM，可以被用户写入数据，但只能写一次）、可擦写可编程 ROM（EPROM，可以通过紫外线进行擦写）、电可擦写可编程 ROM（EEPROM，可以通过电流进行擦写）等。在实际使用中，这些只读存储器主要用在一些需要长期储存数据，且数据不常更改的场合。基于此特点，ROM 在本实验中主要用于储存大量的图像数据的元件设计

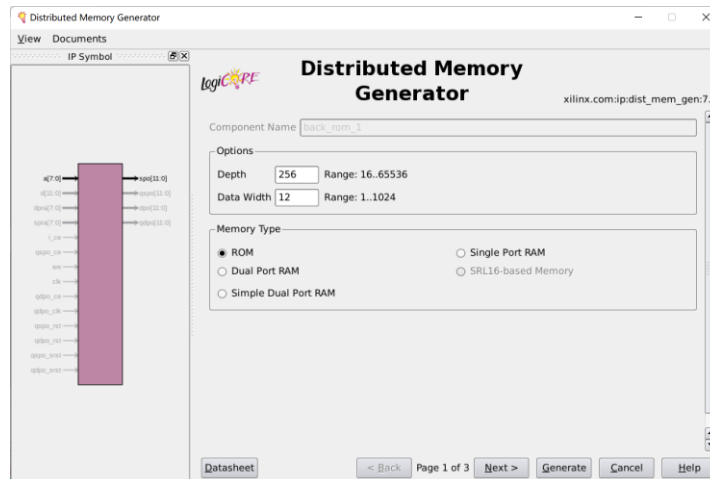
本实验中采取 IP 核进行 ROM 设计。IP 核通常是指在使用 FPGA（现场可编程门阵列）设计时，预先设计好并可以重复利用的电子设计单元。

首先从网络上搜集游戏动画素材，转换合适大小后导出处理为 .coe 文件格式。

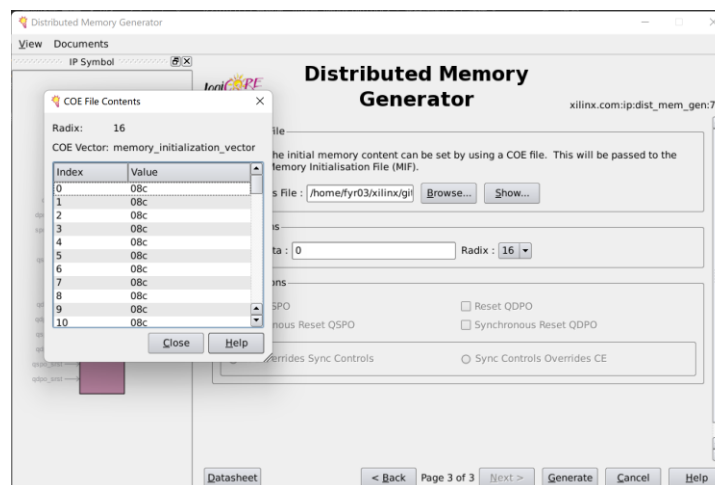
然后选择 IP 核中的 ROM 模块。

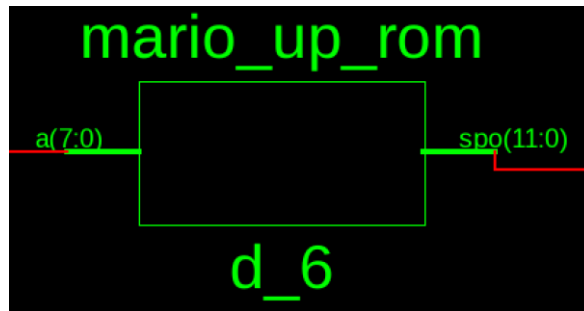


调整 IP 核参数，途中 a 为位置输入引脚，spo 为数据输出引脚



导入.coe 文件。





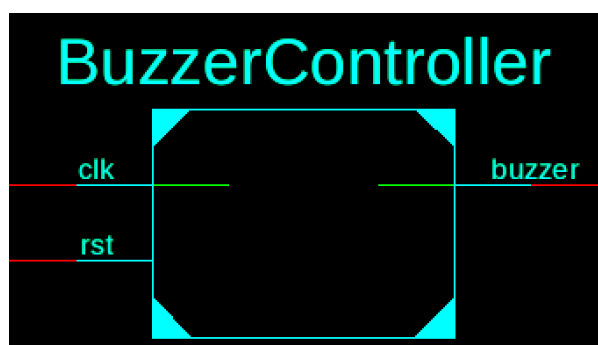
J. 音频模块设计——buzzer_controller

为增强游戏的游玩体验，课程设计中增加了音乐模块，实验台的 buzzer 按照设定好的频率和节奏发声即可放出音乐。

对于 buzzer 而言，放出音乐只需要两个信息，一是当前音符对应的发声频率，二是当前音符需要持续多久，因此我们需要两个数组分别是 `note_periods[]` 记录每个音符的振动周期和 `note_duration[]` 记录每个音符需要发声多久。

1. 首先需要对 `note_period[]` 和 `note_duration[]` 进行初始化，也就是保存要放的音乐的乐谱信息。
2. 接着只需要执行一个 `always` 循环，当如果当前音符的一个周期已经完成，反转 buzzer 的状态；如果当前音符已经播放完毕，切换到下一个音符；否则，增加计数器。

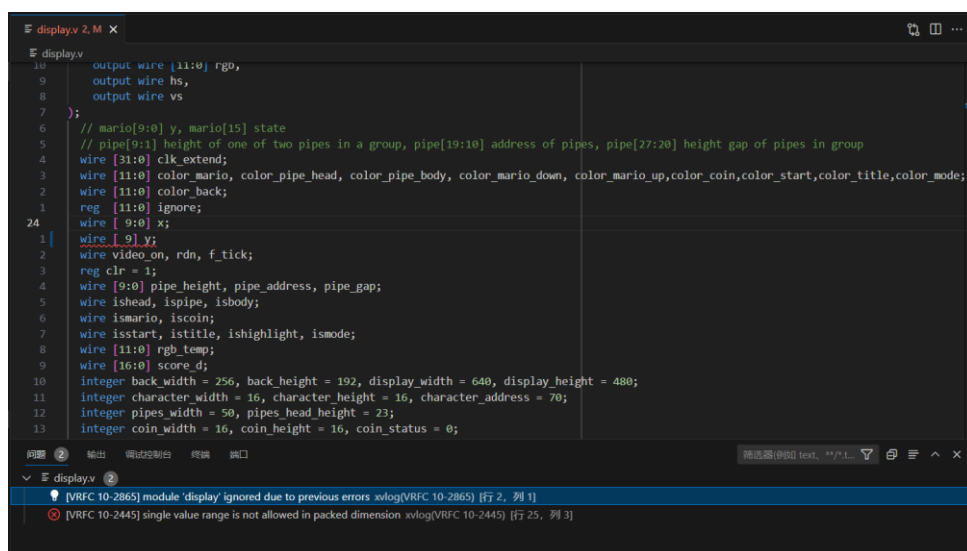
以上，便可实现一个音频 buzzer 模块。



三、调试过程分析

1. 首先是语法调试，语法调试阶段是非常重要的，这部分与其他现代编程语言的调试相似，错误定位通常是非常准确的。我会使用各种开发工具进行语法检查，包括但不限于代码编辑器的内建语法检查功能或者专门的

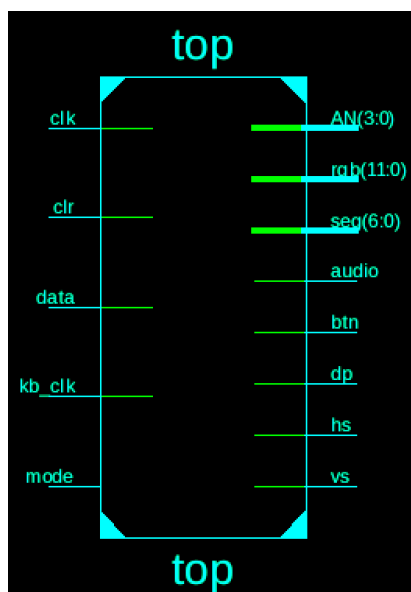
静态代码分析工具。如果我们正在处理一块大的代码，而且其中有很多错误，那么可能会一次性得到很多错误消息。这样的情况下，最好的策略是从上到下，一次解决一个错误，然后重新编译，再继续下一个。同时，让别人审查你的代码，或者对别人的代码进行审查，都是很好的学习和发现错误的方式。



```
display.v
10 output wire [11:0] rgb,
11 output wire hs,
12 output wire vs
13 );
14 // mario[9:0] y, mario[15] state
15 // pipe[9:1] height of one of two pipes in a group, pipe[19:10] address of pipes, pipe[27:20] height gap of pipes in group
16 wire [31:0] clk_extnd;
17 wire [11:0] color_mario, color_pipe_head, color_pipe_body, color_mario_down, color_mario_up, color_coin, color_start, color_title, color_mode;
18 wire [11:0] color_back;
19 reg [11:0] ignore;
20 wire [9:0] x;
21 {
22   wire [9:0] y;
23   wire video_on, rdn, f_tick;
24   reg clr = 1;
25   wire [9:0] pipe_height, pipe_address, pipe_gap;
26   wire ishead, ispipe, isbody;
27   wire ismario, iscoin;
28   wire isstart, istitle, ishighlight, ismode;
29   wire [11:0] rgb_temp;
30   wire [16:0] score_d;
31   integer back_width = 256, back_height = 192, display_width = 640, display_height = 480;
32   integer character_width = 16, character_height = 16, character_address = 70;
33   integer pipes_width = 50, pipes_head_height = 23;
34   integer coin_width = 16, coin_height = 16, coin_status = 0;
```

2. 然后，我们会运行顶层代码的 RTL 综合，分析 RTL 综合电路，看综合结果是否有什么特别之处，这将确保设计功能是否一致。同时，我也会分析和浏览系统综合报告，以清楚地理解系统资源使用情况，检查各个模块的错误信息（包括警告信息），并一一排除。以下是我在分析综合结果时通常会执行的步骤：分析资源使用情况：我会检查设计对 FPGA 资源（如逻辑单元、内存块、I/O 单元等）的使用，以确保设计不会超出器件的容量。如果资源利用率过高，我可能需要优化设计以提高资源利用率。检查时序报告：这个报告会告诉我设计是否满足时序限制。我会查看数据路径以找出可能存在的关键路径，并如果需要，进行时序优化。查看功耗报告：设计的耗电量是一项重要的指标，过高的功耗可能导致热问题，甚至影响设备的寿命。我会查看各个模块的功耗，特别是大单元块的功耗。检查编译器的警告和错误：编译器在综合过程中可能会返回一些警告或错误信息。我会仔细检查这些信息，以确保我们的设计能正确地实现预期的功能。验证设计满足功能需求：我会参照设计的功能描述，检查 RTL 综合结果是否满足这些要求。我也会通过运行模拟来进一步检

查设计的正确性。



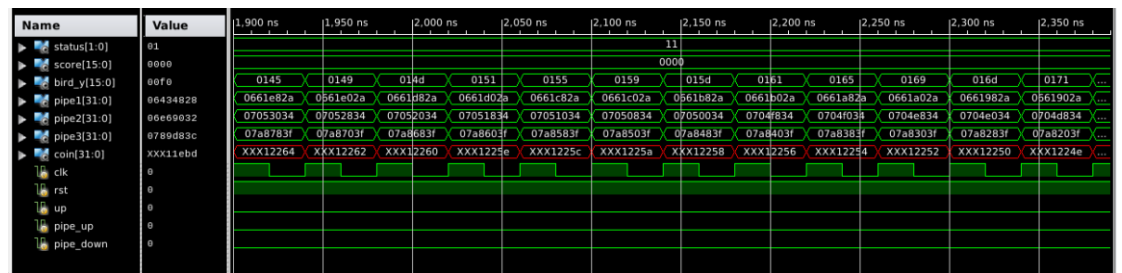
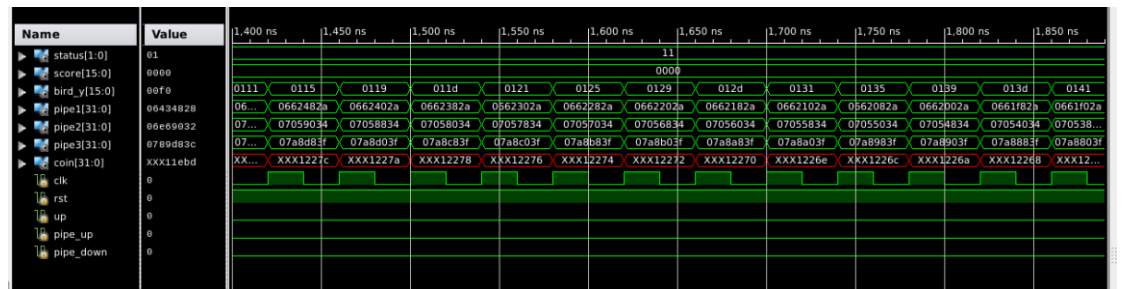
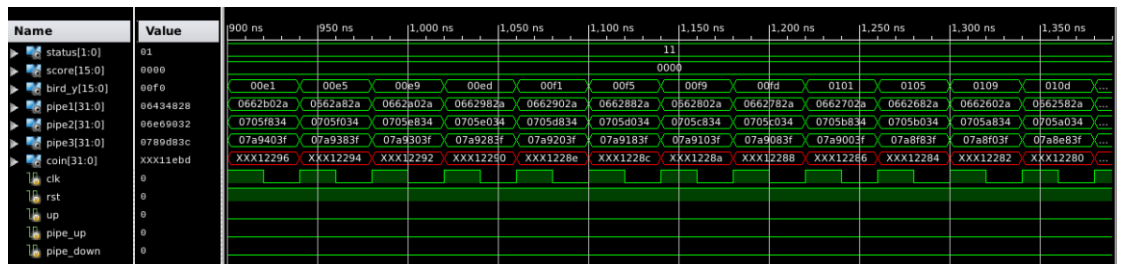
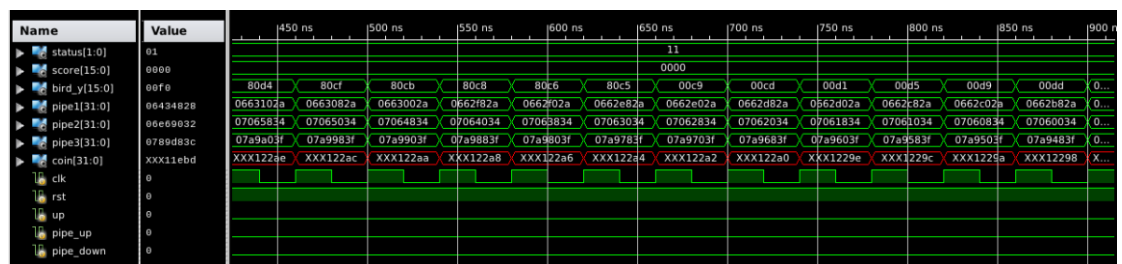
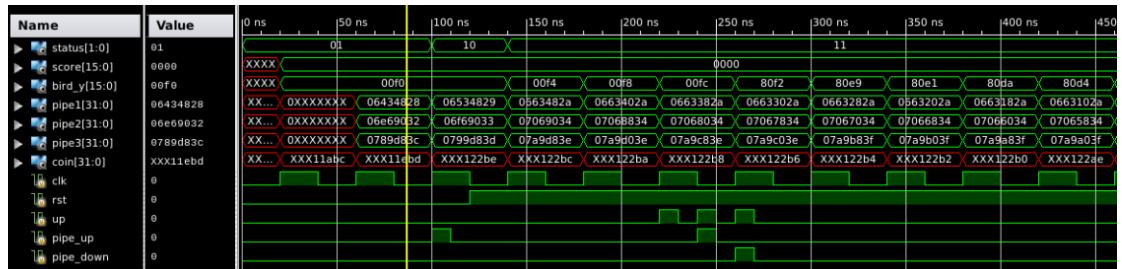
3. 接下来是各个模块的调试过程，包括控制模块，显示模块，VGA 接口模块，PS/2 接口模块，蜂鸣器接口模块等等。我会编写良好的仿真激励代码，根据相应的仿真波形，认真分析，针对其中出现的变量定义错误、语句错误等错综复杂的问题进行逐一修改。具体细节见核心模块仿真部分。
4. 编写 ucf 文件对引脚进行定义，下载至 SWORD 实验板进行验证。在编写 UCF 文件时，需要参考实验板的手册，依据手册上的 FPGA 引脚分配表，将设计中的每一个输入输出信号逐一分配到对应的物理引脚上。注意输入输出信号的名称和信号类型必须跟设计中保持一致。接下来就是观察实验板上的表现，如角色马里奥上下移动情况，管子是否正常随机生成等实际现象。若实验板上的结果与预期一致，那么说明电路设计的功能已经成功实现，否则需要回过头去查找问题的原因，并进行代码的修正和优化，然后重复上述的验证过程。

四、核心模块仿真分析

1. Control 模块仿真

初始状态下，rst、up、pipe_up、pipe_down 均为 0，然后按下 pipe_up 按钮可以看到 status 从 1 变为 2，说明此时在标题页面从默认的单人模式变为了双人模式。然后在 120ns 拨动 rst 为 1，status 由 2 变为 3，说明此时以双人模

式开始游戏，接下来 bird_y 不断减少，说明 Mario 在下落。而在 220ns 处连续按下三次 Up 后 bird_y 增大了数个周期，说明 Mario 飞起。同时 240ns 时按下了 pipe_up，260ns 时按下了 pipe_down，最右侧的管子 pipe3 高度相应地变大变小了。



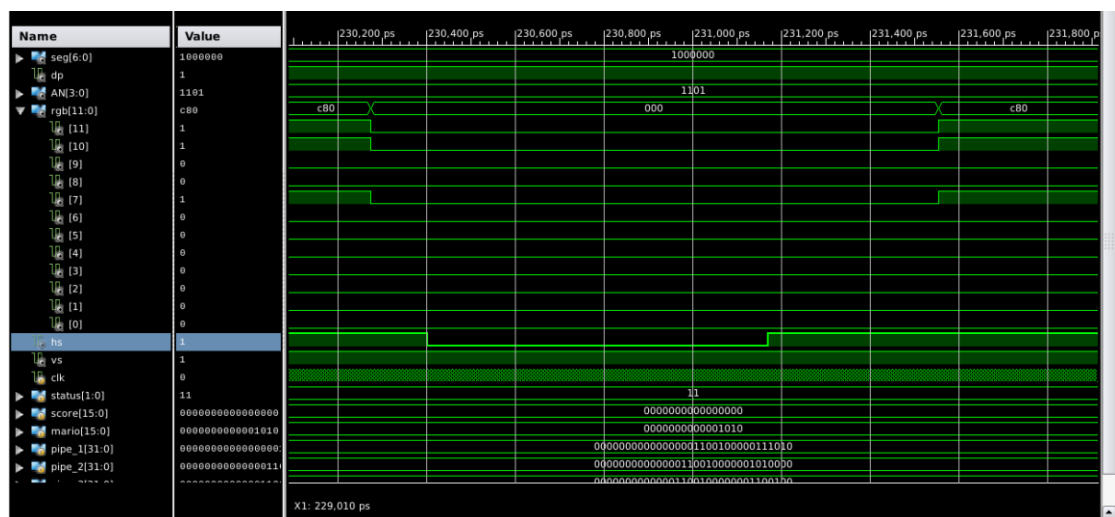
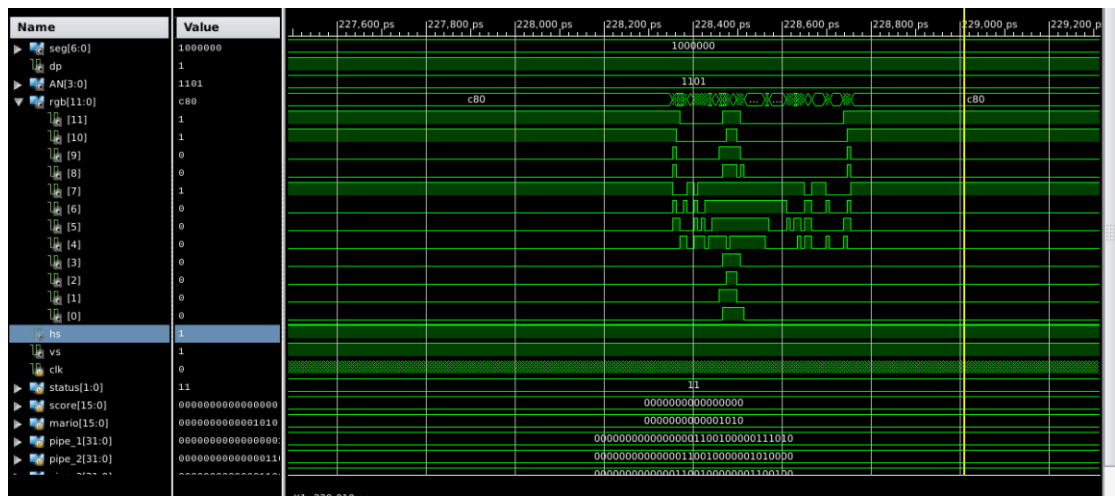
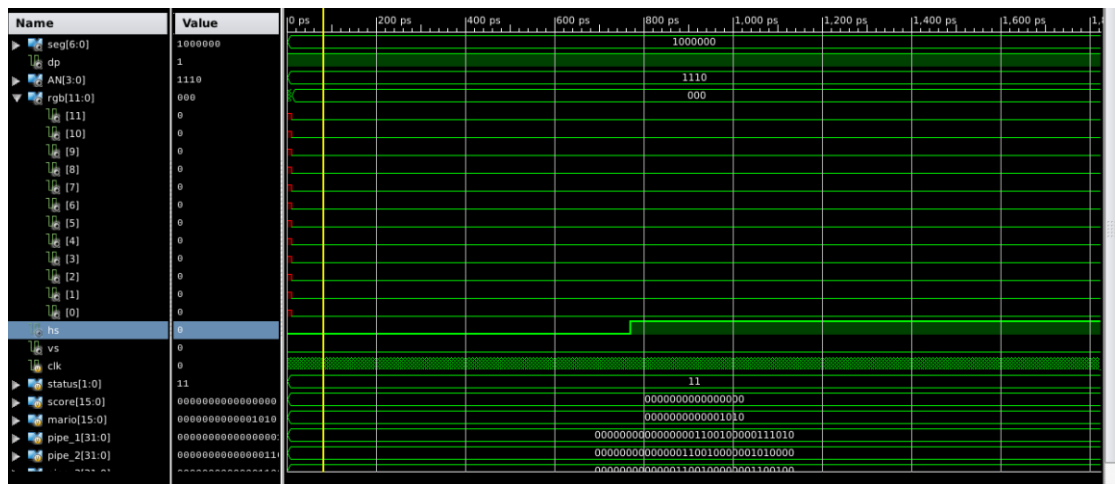
在不操作后，从 2620ns 开始 coin、pipe 等值开始不变，说明此时 Mario 撞到障碍物，游戏停止。

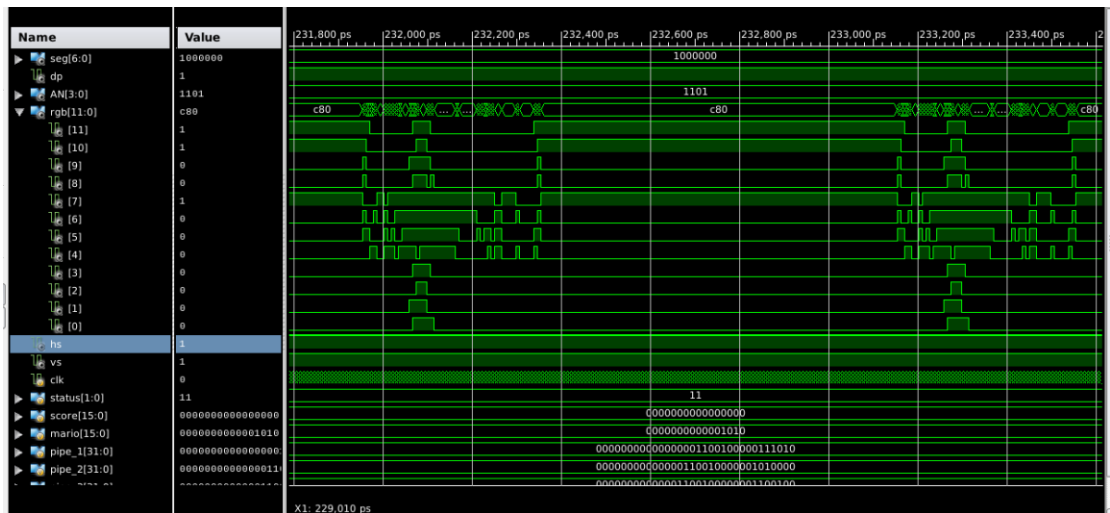

```

        .pipe1(pipe1),
        .pipe2(pipe2),
        .pipe3(pipe3),
        .coin(coin)
    );
    always begin
        clk = ~clk;
        #20;
    end
    initial begin
        clk=0;
        rst=1'b0;
        pipe_up = 1'b0;
        pipe_down = 1'b0;
        up = 1'b0;
        #100;
        pipe_up = 1'b1;#10;
        pipe_up = 1'b0;#10;
        rst=1'b1;#100;
        up=1'b1;#10;
        up=1'b0;#10;
        up=1'b1;pipe_up=1'b1;#10;
        up=1'b0;pipe_up=1'b0;#10;
        up=1'b1;pipe_down=1'b1;#10;
        up=1'b0;pipe_down=1'b0;#10;
        #1000;
        rst=1'b0;
    end
endmodule

```

2. Display 模块仿真





seg 为 7 位数码管输出均为 0，AN 交错使能四位数码管，score 数码管输出正常。VGA 行场同步信号 vs、hs 符合时序逻辑图，rgb 输出信号经过分析符合所属部分区域的像素点。

激励代码：

```
`timescale 1ps / 1ps
module dsisplay_test1 ();
    reg clk;
    reg [1:0] status;
    reg [15:0] score;
    reg [15:0] mario;
    reg [31:0] pipe_1;
    reg [31:0] pipe_2;
    reg [31:0] pipe_3;
    reg [31:0] coin;
    wire [6:0] seg;
    wire dp;
    wire [3:0] AN;
    wire [11:0] rgb;
    wire hs;
    wire vs;
    display test1 (
        .clk(clk),
        .score(score),
        .mario(mario),
        .pipe_1(pipe_1),
        .pipe_2(pipe_2),
        .pipe_3(pipe_3),
        .seg(seg),
        .dp(dp),
```

```

        .AN(AN),
        .rgb(rgb),
        .hs(hs),
        .vs(vs),
        .coin(coin),
        .status(status)
    );
    always begin
        #1 clk = 1'b1;
        #1 clk = 1'b0;
    end
    initial begin
        score = 16'h0000;
        mario = 16'd10;
        pipe_1 = 32'd51258;
        pipe_2 = 32'd204880;
        pipe_3 = 32'd409700;
        status = 2'b11;
        coin = 32'd0;
    end
endmodule

```

五、游玩说明

游戏目标：

控制 Mario 飞行，避开障碍物，吃到金币，获得更高分

游戏游玩方式：

将游戏下载到实验板上后，将 SW[15]拨至 0 状态，此时游戏位于初始页面，此时玩家可以按动 PS2 键盘上的上下方向键在初始页面选择游玩模式（单人模式、双人模式），默认为单人模式。选择游玩模式后将 SW[15]拨至 1 状态即可开始游戏：

单人模式：只有一位玩家可以参与游戏，按动 PS2 键盘的空格键可以使 Mario 飞起，若不按则 Mario 会因重力下降，玩家要靠这一机制使 Mario 保持合适高度通过障碍物、吃到金币，每通过一个障碍物则分数加 1，每吃到一个金币则分数加 2，分数将被显示在实验台的 4 位数码管上。

双人模式：有两位玩家参与游戏，玩家 1 的操作同单人模式的玩家，而

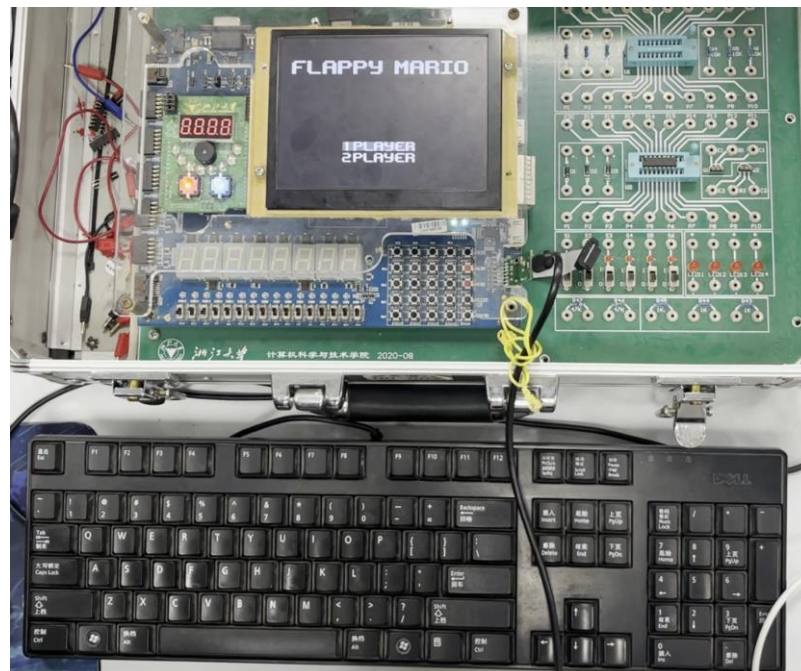
玩家二可以按动 PS2 键盘上的上下方向键调整屏幕最右边管子的高度，从而为玩家一制造更高难度的障碍，分数规则同单人模式，同样会显示在 4 位数码管上。

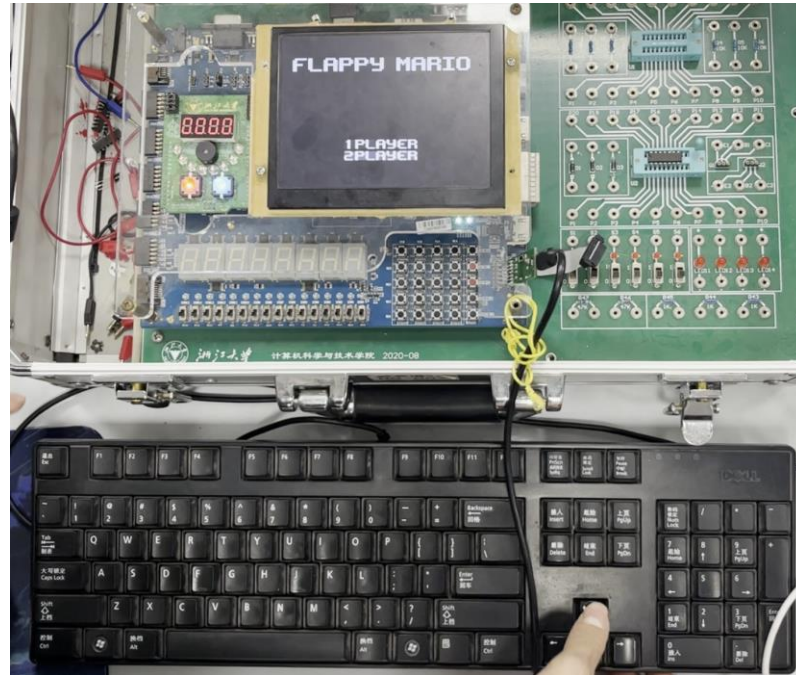
注意：

1. 在游玩的任何时候都可以通过拨动 SW[15]来使游戏回到初始页面。
2. 游玩模式只有在初始界面可以选择，开始游玩后游玩模式将无法改变。

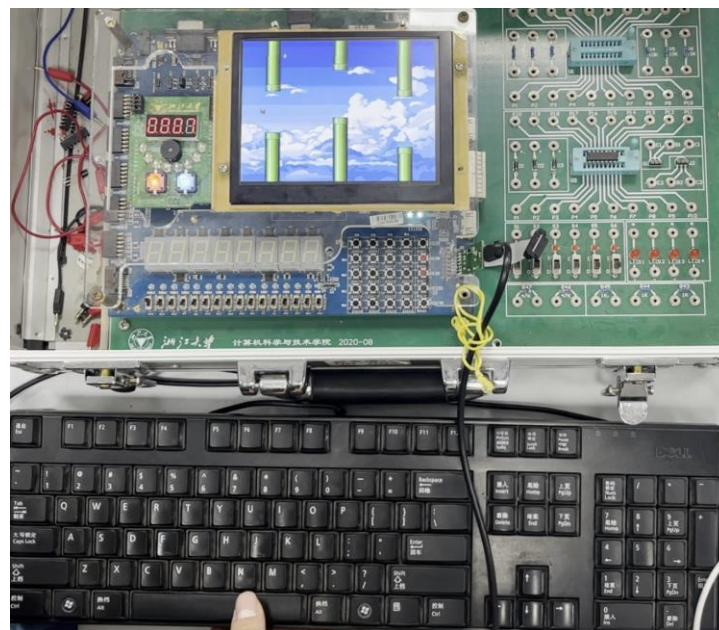
六、验证过程展示

初始页面切换模式高亮、显示均正常

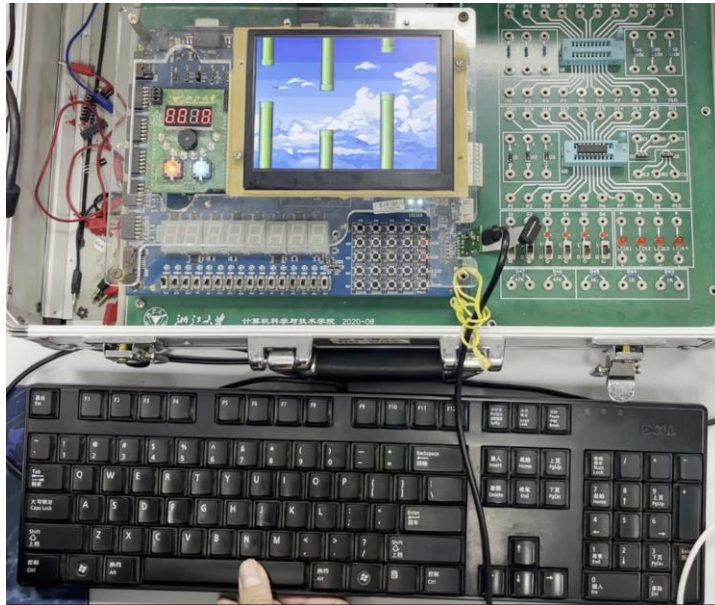




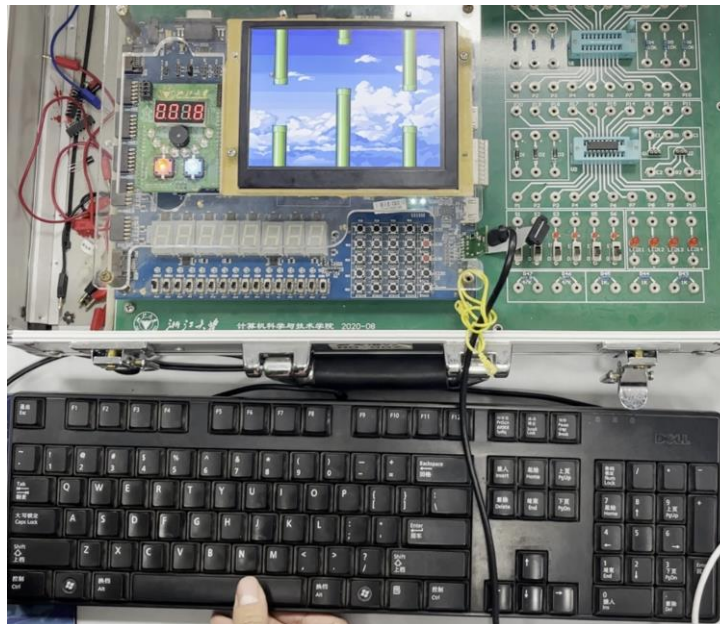
Mario 过管得分正常



Mario 吃金币得分正常



Mario 撞到障碍物后游戏正常结束



七、成员及分工说明

3220103113 费姚瑞 50%: control、buzzer 模块设计, 论文撰写

3220102181 仇国智 50%: display、VGA、PS2 接口模块设计, 论文撰写

八、源代码

top:

```
module top (  
    input wire clk,    // 时钟信号  
    input wire clr,    // 清除信号  
    input wire mode,   // 模式信号
```

```

input wire kb_clk, // 键盘时钟信号
input wire data, // 数据信号
output wire [6:0] seg, // 七段显示器信号
output wire dp, // 小数点信号
output wire [3:0] AN, // 公共阳极信号
output wire [11:0] rgb, // RGB 信号
output wire hs, // 水平同步信号
output wire vs, // 垂直同步信号
output wire btn, // 按钮信号
output wire audio // 音频信号
);

wire [1:0] status; // 状态信号
assign btn=1'b0; // 按钮信号赋值为 0
wire [15:0] score; // 分数信号
wire [15:0] mario; // 马里奥信号
wire [31:0] pipe_1; // 管道 1 信号
wire [31:0] pipe_2; // 管道 2 信号
wire [31:0] pipe_3; // 管道 3 信号
wire [31:0] coin; // 硬币信号
wire space, up, down; // 空格, 上, 下信号
kb_decoder kb_decoder ( // 键盘解码器模块
    .clk(clk),
    .kb_clk(kb_clk),
    .data(data),
    .up(up),
    .down(down),
    .space(space)
);

control control ( // 控制模块
    .clk(clk),
    .rst(clr),
    .up(space),
    .coin(coin),
    .status(status),
    .score(score),
    .bird_y(mario),
    .pipe1(pipe_1),
    .pipe2(pipe_2),
    .pipe3(pipe_3),
    .pipe_up(up),
    .pipe_down(down)
);

display display ( // 显示模块
    .clk(clk),

```

```

        .score(score),
        .status(status),
        .mario(mario),
        .pipe_1(pipe_1),
        .pipe_2(pipe_2),
        .pipe_3(pipe_3),
        .seg(seg),
        .dp(dp),
        .AN(AN),
        .rgb(rgb),
        .hs(hs),
        .vs(vs),
        .coin(coin)
    );
    BuzzerController buzzer_controller ( // 蜂鸣器控制器模块
        .clk(clk),
        .buzzer(audio)
    );
endmodule

```

control:

```

`timescale 1ps / 1ps
module control (
    input wire clk, //时钟
    input wire rst, //rst 为 1 时游戏重新开始
    input up, //up 为 1 时小鸟飞起, 连接 btn
    //input mode, //mode 为 1 时双人模式, 为 0 时单人模式
    input pipe_up,
    input pipe_down,
    output reg [1:0]status,
    output reg [15:0]score, //分数
    output reg [15:0]bird_y, //bird_y 为小鸟下侧 y 坐标, 第 16 位记录小鸟是否
    在下落, 为 0 时小鸟下落, 为 1 时小鸟上升
    output reg [31:0]pipe1, //pipe1 储存管道 1 的 x 和 y 和 gap, x 占高 10 位, y
    占低 10 位, x 为管子左边界坐标, y 为管子上侧坐标
    output reg [31:0]pipe2, //pipe2 储存管道 2 的 x 和 y 和 gap, x 占高 10 位, y
    占低 10 位, x 为管子左边界坐标, y 为管子上侧坐标
    output reg [31:0]pipe3, //pipe3 储存管道 3 的 x 和 y 和 gap, x 占高 10 位, y
    占低 10 位, x 为管子左边界坐标, y 为管子上侧坐标
    output reg [31:0]coin, //coin 储存金币的 x 和 y, y 占高 10 位, x 占低 10 位,
    x 为金币左边界坐标, y 为金币下侧坐标
    //output reg [3:0]bird_falling
);

```

```

wire [31:0] clk_div;
// wire up1;
// wire pipeup1;
// wire pipedown1;
reg [1:0]cnt=3;
reg fail;//fail 为 1 游戏结束
//reg pass1;
//reg pass2;
//reg pass3; //pass1,pass2,pass3 为小鸟是否通过管道，为 1 时通过
reg longpress;
reg [9:0] pipe1_x; //pipe1_x 为管 1 的 x 左边界坐标
reg [9:0] pipe1_y; //pipe1_y 为管 1 的 y 上边界坐标
reg [9:0] pipe2_x; //pipe2_x 为管 2 的 x 左边界坐标
reg [9:0] pipe2_y; //pipe2_y 为管 2 的 y 上边界坐标
reg [9:0] pipe3_x; //pipe3_x 为管 3 的 x 左边界坐标
reg [9:0] pipe3_y; //pipe3_y 为管 3 的 y 上边界坐标
wire [9:0] bird_x = 10'd70; //bird_x 为小鸟左边界坐标，小鸟是不动的，所以 x 坐标为常数
//wire [9:0] gap = 10'd50; //gap 为管道间隙，供小鸟通过
wire [9:0] bird_width = 10'd16; //bird_width 为小鸟宽度
wire [9:0] bird_height = 10'd16; //bird_height 为小鸟高度
wire [9:0] pipe_width = 10'd50; //pipe_width 为管道宽度
wire [7:0] pipe_head=8'd40;
wire [7:0] coin_length=8'd16;
reg [7:0]gap1; //gap1 为管道 1 的间隙，供小鸟通过
reg [7:0]gap2; //gap2 为管道 2 的间隙，供小鸟通过
reg [7:0]gap3; //gap3 为管道 3 的间隙，供小鸟通过
reg [15:0] bird_flying; //bird_flying 为小鸟距离开始下落的时间，为 0 时小鸟下落
reg old;
wire clk_100ms;
//reg bird_falltime;

clk_div m0 (
    .clk(clk),
    .rst(1'b0),
    .clk_div(clk_div)
); //时钟分频，将时钟分频 100ms
// pbdebounce m1 (
//     .clk_1ms(clk_div[17]),
//     .btn(up),
//     .pbreg(up1)
// ); //按钮去抖动，将按钮信号去抖动
// pbdebounce m3 (

```

```

//      .clk_1ms(clk_div[17]),
//      .btn(pipe_up),
//      .pbreg(pipeup1)
// ); //按钮去抖动, 将按钮信号去抖动
// pbdebounce m4 (
//      .clk_1ms(clk_div[17]),
//      .btn(pipe_down),
//      .pbreg(pipedown1)
// ); //按钮去抖动, 将按钮信号去抖动
clk_100ms m2 (
    .clk(clk),
    .clk_100ms(clk_100ms)
); //100ms 时钟
initial begin
    status <= 2'b01;
end
always @(posedge clk_100ms) begin
    if (!rst) begin
        if(status==0||status==3)status<=1;
        if(pipe_up||pipe_down) begin
            if(old==0)begin
                if(status==1)status <= 2'b10;
                else status <= 2'b01;
            end
            old <= 1;
        end
        else old <= 0;
        bird_y[14:0] <= 16'd240; //小鸟初始位置
        bird_y[15] <= 1'b0;
        pipe1_x <= 20'd210;
        pipe2_x <= 20'd420;
        pipe3_x <= 20'd630;
        cnt <= 3;
        //pass1 <= 0;
        //pass2 <= 0;
        //pass3 <= 0;//pass1,pass2,pass3 为小鸟是否通过管道, 为 1 时通过
        //bird_falltime <= 0;
        longpress <= 0;
        fail <= 0;
        score <= 16'd0;
        bird_flying <= 16'b0000; //小鸟距离开始下落的时间, 为 0 时小鸟下落
        coin[31] <= 1'b1;
        coin[9:0] <= 10'd650+pipe_width+clk_div % 20;
        coin[19:10] <= 20+(clk_div+50)%(440-coin_length);
    end
end

```



```

    pipe1_y <= pipe_head + clk_div % (310-pipe_head-pipe_head); // 生成 pipe_head 到 480-150-pipe_head 的随机数
    pipe2_y <= pipe_head + (clk_div+10) % (310-pipe_head-pipe_head); // 生成 pipe_head 到 480-150-pipe_head 的随机数
    pipe3_y <= pipe_head + (clk_div+20) % (310-pipe_head-pipe_head); // 生成 pipe_head 到 480-150-pipe_head 的随机数
    gap1 <= 100 + clk_div % 50;
    gap2 <= 100 + (clk_div+10) % 50;
    gap3 <= 100 + (clk_div+20) % 50;
    pipe1 <= {gap1,pipe1_x, pipe1_y};
    pipe2 <= {gap2,pipe2_x, pipe2_y};
    pipe3 <= {gap3,pipe3_x, pipe3_y};
end else begin
    if(status==1)status <= 2'b00;
    if(status==2)status <= 2'b11;
    if (pipe_up&&!fail&&status==3)begin
        case(cnt)
            1:if(pipe1_y>=pipe_head)pipe1_y<=pipe1_y-1;
            2:if(pipe2_y>=pipe_head)pipe2_y<=pipe2_y-1;
            3:if(pipe3_y>=pipe_head)pipe3_y<=pipe3_y-1;
        endcase
    end
    if (pipe_down&&!fail&&status==3)begin
        case(cnt)
            1:if(pipe1_y+pipe_head<=330)pipe1_y<=pipe1_y+1;
            2:if(pipe2_y+pipe_head<=330)pipe2_y<=pipe2_y+1;
            3:if(pipe3_y+pipe_head<=330)pipe3_y<=pipe3_y+1;
        endcase
    end
    if (up && !fail &&!longpress) begin //按钮按下时 up 为 1, 小鸟飞行 4 个周期
        bird_flying <= 16'd10;
        bird_y[15] <= 1'b1;
        longpress <= 1;
        //bird_falltime <= 0;
    end
    if ((bird_flying<=0)||fail) begin //小鸟下落
        bird_y[14:0] <= bird_y[14:0] + 4;
        bird_y[15] <= 1'b0;
        //bird_falltime <= bird_falltime + 1;
    end
    if ((bird_flying>0)&&(!fail)) begin //小鸟按惯性向上飞
        bird_y[14:0] <= bird_y[14:0] - bird_flying;
        bird_flying <= bird_flying - 1;
    end
end

```

```

        bird_y[15] <= 1'b1;
    end
    if (!up) begin
        longpress <= 0;
    end
    if (!fail) begin //若游戏未结束，管道移动
        pipe1_x <= pipe1_x - 2;
        pipe2_x <= pipe2_x - 2;
        pipe3_x <= pipe3_x - 2;
        coin[9:0] <= coin[9:0] - 2;
    end
    if (pipe1_x<= 2) begin //管道移出屏幕后重新生成
        pipe1_x <= 20'd640;
        pipe1_y <= pipe_head + clk_div % (310-pipe_head-pipe_head); //
生成 pipe_head 到 480-150-pipe_head 的随机数
        gap1 <= 120 + clk_div % 50;
        //pass1 <= 0;
        cnt <= 1;
    end
    if (pipe2_x<= 2) begin //管道移出屏幕后重新生成
        pipe2_x <= 20'd640;
        pipe2_y <= pipe_head + clk_div % (310-pipe_head-pipe_head); //
生成 pipe_head 到 480-150-pipe_head 的随机数
        gap2 <= 120 + clk_div % 50;
        //pass2 <= 0;
        cnt <= 2;
    end
    if (pipe3_x<= 2) begin //管道移出屏幕后重新生成
        pipe3_x <= 20'd640;
        pipe3_y <= pipe_head + clk_div % (310-pipe_head-pipe_head); //
生成 pipe_head 到 480-150-pipe_head 的随机数
        gap3 <= 120 + clk_div % 50;
        //pass3 <= 0;
        cnt <= 3;
    end
    end
    if(coin[31]==0||coin[9:0]<=0) begin
        coin[31] <= 1'b1;
        case(cnt)
            1:coin[9:0] <= pipe1_x+pipe_width+clk_div % 20;
            2:coin[9:0] <= pipe2_x+pipe_width+clk_div % 20;
            3:coin[9:0] <= pipe3_x+pipe_width+clk_div % 20;
        endcase
        coin[19:10] <= 20+(clk_div+50)%(440-coin_length);
    end
end

```



```

        if (((bird_y[14:0] <=
pipe1_y)||((bird_y[14:0]+bird_height>=pipe1_y+gap1))&&(bird_x<=pipe1_x+pipe_width)&&(bird_x+bird_width>=pipe1_x))
        ||(((bird_y[14:0] <=
pipe2_y)||((bird_y[14:0]+bird_height>=pipe2_y+gap2))&&(bird_x<=pipe2_x+pipe_width)&&(bird_x+bird_width>=pipe2_x))
        ||(((bird_y[14:0] <=
pipe3_y)||((bird_y[14:0]+bird_height>=pipe3_y+gap3))&&(bird_x<=pipe3_x+pipe_width)&&(bird_x+bird_width>=pipe3_x)))) begin
            //若小鸟与管道相撞，游戏结束
            fail <= 1;
        end
        if (!fail) begin
            if(bird_x==pipe1_x+pipe_width) begin
                score <= score + 1;
                //pass1 <= 1;
            end
            if(bird_x==pipe2_x+pipe_width) begin
                score <= score + 1;
                //pass2 <= 1;
            end
            if(bird_x==pipe3_x+pipe_width) begin
                score <= score + 1;
                //pass3 <= 1;
            end
            //若小鸟通过管道，分数加 1
        end
        if(coin[31]&&((bird_y[14:0]<=coin[19:10]+coin_length)&&(bird_y[14:0]+bird_height>=coin[19:10])&&(bird_x<=coin[9:0]+coin_length)&&(bird_x+bird_width>=coin[9:0]))) begin
            coin[31] <= 1'b0;
            score <= score + 2;
        end
        pipe1 <= {gap1,pipe1_x, pipe1_y};
        pipe2 <= {gap2,pipe2_x, pipe2_y};
        pipe3 <= {gap3,pipe3_x, pipe3_y};
    end
end
endmodule

```

display:

```

`timescale 1ps / 1ps
module display (
    input wire clk, // 输入时钟信号

```

```

    input wire [1:0] status, // 输入状态信号
    input wire [15:0] score, // 输入得分信号
    input wire [15:0] mario, // 输入马里奥信号
    input wire [31:0] pipe_1, // 输入管道 1 信号
    input wire [31:0] pipe_2, // 输入管道 2 信号
    input wire [31:0] pipe_3, // 输入管道 3 信号
    input wire [31:0] coin, // 输入金币信号
    output wire [6:0] seg, // 输出段信号
    output wire dp, // 输出小数点信号
    output wire [3:0] AN, // 输出 AN 信号
    output wire [11:0] rgb, // 输出 RGB 信号
    output wire hs, // 输出水平同步信号
    output wire vs // 输出垂直同步信号
);
    // mario[9:0] y, mario[15] state
    // pipe[9:1] height of one of two pipes in a group, pipe[19:10]
address of pipes, pipe[27:20] height gap of pipes in group
    wire [31:0] clk_extend; // 扩展时钟信号
    wire [11:0] color_mario, color_pipe_head, color_pipe_body,
color_mario_down,
color_mario_up,color_coin,color_start,color_title,color_mode; // 颜色信
号
    wire [11:0] color_back; // 背景颜色信号
    reg [11:0] ignore; // 忽略信号
    wire [ 9:0] x; // x 坐标信号
    wire [ 9:0] y; // y 坐标信号
    wire video_on, rdn, f_tick; // 视频开关信号, 读取使能信号, 帧刷新信号
    reg clr = 1; // 清除信号
    wire [9:0] pipe_height, pipe_address, pipe_gap; // 管道高度信号, 管道
地址信号, 管道间隙信号
    wire ishead, ispipe, isbody; // 是否为头部信号, 是否为管道信号, 是否为身
体信号
    wire ismario, iscoin; // 是否为马里奥信号, 是否为金币信号
    wire isstart, istitle, ishighlight, ismode; // 是否为开始信号, 是否为标
题信号, 是否为高亮信号, 是否为模式信号
    wire [11:0] rgb_temp; // RGB 临时信号
    wire [16:0] score_d; // 得分显示信号
    integer
        back_width = 256,
        back_height = 192,
        display_width = 640,
        display_height = 480; // 背景宽度, 背景高度, 显示宽度, 显示高度
    integer
        character_width = 16,

```

```

        character_height = 16,
        character_address = 70; // 字符宽度, 字符高度, 字符地址
integer pipes_width = 50, pipes_head_height = 23; // 管道宽度, 管道头部高度
integer
    coin_width = 16,
    coin_height = 16,
    coin_status = 0; // 定义硬币的宽度、高度和状态

integer
    title_width = 183, // 标题的宽度
    title_height = 14, // 标题的高度
    title_width_start = 45, // 标题的起始宽度
    title_height_start = 80, // 标题的起始高度
    title_multi = 3; // 标题的倍数

integer
    mode_width = 112, // 模式的宽度
    mode_height = 32, // 模式的高度
    mode_width_start = 208, // 模式的起始宽度
    mode_height_start = 340, // 模式的起始高度
    mode_line_height = 14, // 模式的行高
    mode_multi = 2; // 模式的倍数
// title x:65-613 y: 40-81
// mode x:228-451 y:300-363
initial begin
    ignore = 12'h00f; // 初始化 ignore 变量为十六进制的 00f
end
assign y[9] = 1'b0; // 将 y 的第 9 位赋值为 0
assign pipe_gap[9:8] = 0; // 将 pipe_gap 的第 9 位和第 8 位赋值为 0

// 定义一个 DispNum 模块实例 d_0, 用于处理分数的显示
DispNum d_0 (
    .clk(clk),
    .RST(1'b0),
    .LES(4'b0000),
    .points(4'b0000),
    .AN(AN),
    .Segment({dp, seg}),
    .HEXS(score_d)
);

// 定义一个 clkdiv 模块实例 d_1, 用于处理时钟分频
clkdiv d_1 (

```

```

        .clk(clk),
        .rst(1'b0),
        .clkdiv(clk_extend)
    );

    // 定义一个 back_rom 模块实例 d_2, 用于处理背景的显示
    back_rom d_2 (
        .a (((y * back_height) / display_height) * back_width) + (x *
back_width / display_width)),
        .spo(color_back)
    );

    // 定义一个 mario_rom 模块实例 d_3, 用于处理马里奥的显示
    mario_rom d_3 (
        .a ((y - mario[9:0]) * character_width + (x -
character_address))),
        .spo(color_mario_down)
    );

    // 定义一个 mario_up_rom 模块实例 d_6, 用于处理马里奥跳跃的显示
    mario_up_rom d_6 (
        .a ((y - mario[9:0]) * character_width + (x -
character_address))),
        .spo(color_mario_up)
    );

    // 定义一个 pipe_head_rom 模块实例 d_4, 用于处理管道头部的显示
    pipe_head_rom d_4 (
        .a((((y>=pipe_height-pipes_head_height&&y<pipe_height)?
y-
pipe_height+pipes_head_height:pipe_height+pipe_gap+pipes_head_height-2-
y)*pipes_width
        +(x - pipe_address))),
        .spo(color_pipe_head)
    );

    // 定义一个 pipe_body_rom 模块实例 d_5, 用于处理管道身体的显示
    pipe_body_rom d_5 (
        .spo(color_pipe_body),
        .a (x - pipe_address)
    );

    // 定义一个 coin_rom 模块实例 d_7, 用于处理硬币的显示
    coin_rom d_7 (

```

```

        .spo(color_coin),
        .a ((y - coin[19:10]) * coin_width * 4 + coin_width *
coin_status + x - coin[9:0])
    );

    // 定义一个 title_rom 模块实例 d_8, 用于处理标题的显示
    title_rom d_8 (
        .spo(color_title),
        .a ((y - title_height_start)/title_multi * title_width + (x -
title_width_start)/title_multi)
    );

    // 定义一个 mode_rom 模块实例 d_9, 用于处理模式的显示
    mode_rom d_9 (
        .spo(color_mode),
        .a ((y - mode_height_start) / mode_multi * mode_width + (x -
mode_width_start) / mode_multi)
    );
    // 调用 VGA 接口模块
    vga_sync vga_sync (
        .vga_clk(clk_extend[1]),
        .clrn(clr),
        .d_in(rgb_temp),
        .hs(hs),
        .vs(vs),
        .r(rgb[11:8]),
        .g(rgb[7:4]),
        .b(rgb[3:0]),
        .rdn(rdn),
        .col_addr(x),
        .row_addr(y[8:0])
    );
    // 金币 4 个旋转状态的周期性变更
    always @(posedge clk_extend[22]) begin
        if (coin_status[0] && coin_status[1]) begin
            coin_status <= 0;
        end else begin
            coin_status <= coin_status + 1;
        end
    end
    // 将 16 进制的 score 转换为 10 进制的 score_d
    assign score_d[3:0] = score % 10;
    assign score_d[7:4] = score / 10 % 10;
    assign score_d[11:8] = score / 100 % 10;

```

```

assign score_d[15:12] = score / 1000 % 10;
//判断是否在游戏开始界面
assign isstart = (status[0] && ~status[1]) || (~status && status[1]);
//判断坐标是否在标题区域
assign istitle = (x >= title_width_start && x < title_width_start +
title_multi * title_width)
                && (y >= title_height_start && y < title_height_start
+ title_multi * title_height)&&(!(color_title^ignore));
//判断坐标是否在模式选择区域
assign ismode = (x >= mode_width_start && x < mode_width_start +
mode_multi * mode_width)
                && (y >= mode_height_start && y < mode_height_start +
mode_multi * mode_height)&&(!(color_mode^ignore));
//判断坐标是否在高亮区域
assign ishighlight = (status[0] && ~status[1] && ((x >=
mode_width_start && x < mode_width_start + mode_multi * mode_width)
                && (y >=
mode_height_start && y < mode_height_start + mode_line_height *
mode_multi)))
                || (~status[0] && status[1] && ((x >=
mode_width_start && x < mode_width_start + mode_multi * mode_width)
                && (y >=
mode_height_start + mode_multi * (mode_height - mode_line_height)
                && y <
mode_height_start + mode_multi * mode_height))));
//判断坐标是否在管子区域，并且确定该管子的参数
assign {ispipe, pipe_gap[7:0], pipe_address, pipe_height} =
((0 <= x - pipe_1[19:10]) && (x - pipe_1[19:10] < pipes_width)) ?
{1'b1, pipe_1[27:20], pipe_1[19:10], pipe_1[9:0]} :
((0 <= x - pipe_2[19:10]) && (x - pipe_2[19:10] < pipes_width)) ?
{1'b1, pipe_2[27:20], pipe_2[19:10], pipe_2[9:0]} :
((0 <= x - pipe_3[19:10]) && (x - pipe_3[19:10] < pipes_width)) ?
{1'b1, pipe_3[27:20], pipe_3[19:10], pipe_3[9:0]} :
{1'b0, 28'b0};
//判断马里奥的状态
assign color_mario = mario[15] ? color_mario_up : color_mario_down;
//判断坐标是否在管子头部区域
assign ishead = ((y >= pipe_height - pipes_head_height) && (y <
pipe_height))
                || ((-y + pipe_height + pipe_gap + pipes_head_height -
2 >= 10'd0)
                && (-y + pipe_height + pipe_gap - 2 +
pipes_head_height < pipes_head_height))
                && (!(color_pipe_head ^ ignore))) ? 1'b1 : 1'b0;

```

```

//判断坐标是否在管子身体区域
assign isbody = (y < pipe_height - pipes_head_height)
                || (pipe_height + pipe_gap + pipes_head_height - 2 < y)
                && (!(color_pipe_body ^ ignore)) ? 1'b1 : 1'b0;
//判断坐标是否在马里奥区域
assign ismario = ((character_address <= x) && (x < character_address
+ character_width)
                && (y >= mario[9:0]) && (y < mario[9:0] +
character_height)
                && (!(color_mario ^ ignore))) ? 1'b1 : 1'b0;
//判断坐标是否在金币区域
assign iscoin = coin[31] && (!(color_coin ^ ignore))
                && (x - coin[9:0] < coin_width)
                && (y - coin[19:10] < coin_height)
                && (x - coin[9:0] >= 0)
                && (y - coin[19:0] >= 0));

//融合开始界面图层
assign color_start = istitle ? color_title : ismode ? color_mode :
ishighlight ? 12'h888 : 12'h000;
//融合所有图层
assign rgb_temp =
    rdn ? 12'h000 :
    isstart ? color_start :
    ismario ? color_mario :
    iscoin ? color_coin :
    ispipe && ishead ? color_pipe_head :
    ispipe && isbody ? color_pipe_body :
    color_back;
endmodule

```

BuzzerController:

```

`define c3 191970
`define d3 171154
`define e3 152651
`define f3 144170
`define g3 127980
`define le 121041
`define a3 114103
`define b3 101767
`define c4 95600
`define d4 84806
`define me 80180
`define e4 75554
`define f4 71700

```

```

`define g4 63990
`define a4 56280
`define b4 50112
`define c5 47800
`define d5 42403
`define e5 37777
`define f5 35464
`define g5 31609
`define a5 27754
`define b5 24670
`define xz 770
`timescale 1ns/1ps
module BuzzerController(
    input wire clk,
    input wire rst,
    output reg buzzer
);

// 音符的数量
parameter NOTE_COUNT = 79;

// 音符的振动周期和持续时间
reg [19:0] note_periods[78:0];
reg [19:0] note_durations[78:0];

// 当前播放的音符和时间计数器
integer current_note;
integer note_counter;
integer duration_counter;

// 初始化 buzzer 为 0
initial begin
    buzzer = 0;
    current_note = 0;
    note_counter = 0;
    duration_counter = 0;
end

initial begin
    note_periods[0] <= `c4; note_periods[1] <= `d4; note_periods[2] <=
`e4; note_periods[3] <= `xz; note_periods[4] <= `e4;
    note_periods[5] <= `xz; note_periods[6] <= `e4; note_periods[7] <=
`d4; note_periods[8] <= `e4; note_periods[9] <= `f4;

```



```

    note_periods[10] <= `xz; note_periods[11] <= `f4; note_periods[12]
<= `xz;note_periods[13] <= `f4;note_periods[14] <= `e4;
    note_periods[15] <= `f4; note_periods[16] <= `g4; note_periods[17]
<= `xz;note_periods[18] <= `g4;note_periods[19] <= `d4;
    note_periods[20] <= `f4; note_periods[21] <= `e4; note_periods[22]
<= `d4;note_periods[23] <= `e4;note_periods[24] <= `d4;
    note_periods[25] <= `c4; note_periods[26] <= `xz; note_periods[27]
<= `c4;note_periods[28] <= `xz;note_periods[29] <= `c4;
    note_periods[30] <= `a3; note_periods[31] <= `c4; note_periods[32]
<= `d4;note_periods[33] <= `xz;note_periods[34] <= `d4;
    note_periods[35] <= `xz; note_periods[36] <= `e4; note_periods[37]
<= `c4;note_periods[38] <= `e4;note_periods[39] <= `f4;
    note_periods[40] <= `g4; note_periods[41] <= `d4; note_periods[42]
<= `xz;note_periods[43] <= `d4;note_periods[44] <= `e4;
    note_periods[45] <= `f4; note_periods[46] <= `g4; note_periods[47]
<= `xz;note_periods[48] <= `g4;note_periods[49] <= `xz;
    note_periods[50] <= `g4; note_periods[51] <= `a4; note_periods[52]
<= `b4;note_periods[53] <= `g4;note_periods[54] <= `xz;
    note_periods[55] <= `g4; note_periods[56] <= `a4; note_periods[57]
<= `b4;note_periods[58] <= `c5;note_periods[59] <= `g4;
    note_periods[60] <= `xz; note_periods[61] <= `g4; note_periods[62]
<= `f4;note_periods[63] <= `e4;note_periods[64] <= `g4;
    note_periods[65] <= `d5; note_periods[66] <= `g4; note_periods[67]
<= `c5;note_periods[68] <= `b4;note_periods[69] <= `c5;
    note_periods[70] <= `a4; note_periods[71] <= `d5; note_periods[72]
<= `c5;note_periods[73] <= `b4;note_periods[74] <= `a4;
    note_periods[75] <= `b4; note_periods[76] <= `xz; note_periods[77]
<= `c5;note_periods[78] <= `xz;
end
initial begin
    note_durations[0] <= 12500;note_durations[1] <=
6250;note_durations[2] <= 12500;note_durations[3] <= 6250;
    note_durations[4] <= 18750;note_durations[5] <=
10937;note_durations[6] <= 6250;note_durations[7] <= 12500;
    note_durations[8] <= 6250;note_durations[9] <=
18750;note_durations[10] <= 12500;note_durations[11] <= 6250;
    note_durations[12] <= 12500;note_durations[13] <=
6250;note_durations[14] <= 12500;note_durations[15] <= 6250;
    note_durations[16] <= 18750;note_durations[17] <=
6250;note_durations[18] <= 17187.5;note_durations[19] <= 17187.5;
    note_durations[20] <= 18750;note_durations[21] <=
12500;note_durations[22] <= 6250;note_durations[23] <= 37500;
    note_durations[24] <= 18750;note_durations[25] <=
12500;note_durations[26] <= 6250;note_durations[27] <= 17187.5;

```

```

    note_durations[28] <= 6250;note_durations[29] <=
18750;note_durations[30] <= 12500;note_durations[31] <= 6250;
    note_durations[32] <= 12500;note_durations[33] <=
6250;note_durations[34] <= 12500;note_durations[35] <= 6250;
    note_durations[36] <= 18750;note_durations[37] <=
6250;note_durations[38] <= 17187.5;note_durations[39] <= 18750;
    note_durations[40] <= 12500;note_durations[41] <=
6250;note_durations[42] <= 37500;note_durations[43] <= 18750;
    note_durations[44] <= 12500;note_durations[45] <=
6250;note_durations[46] <= 17187.5;note_durations[47] <= 6250;
    note_durations[48] <= 18750;note_durations[49] <=
12500;note_durations[50] <= 6250;note_durations[51] <= 18750;
    note_durations[52] <= 18750;note_durations[53] <=
18750;note_durations[54] <= 18750;note_durations[55] <= 90000;
    note_durations[56] <= 6250;note_durations[57] <=
12500;note_durations[58] <= 6250;note_durations[59] <= 17187.5;
    note_durations[60] <= 6250;note_durations[61] <=
17187.5;note_durations[62] <= 6250;note_durations[63] <= 18750;
    note_durations[64] <= 12500;note_durations[65] <=
6250;note_durations[66] <= 27500;note_durations[67] <= 12500;
    note_durations[68] <= 6250;note_durations[69] <=
12500;note_durations[70] <= 6250;note_durations[71] <= 18750;
    note_durations[72] <= 18750;note_durations[73] <=
6250;note_durations[74] <= 17187.5;note_durations[75] <= 18750;
    note_durations[76] <= 37500;note_durations[77] <=
18750;note_durations[78] <= 12500;
end

always @(posedge clk or posedge rst) begin
    if (rst) begin
        // 当复位信号为高时, 将 buzzer 和计数器重置
        buzzer <= 0;
        current_note <= 0;
        note_counter <= 0;
        duration_counter <= 0;
    end else begin
        // 播放音乐
        if (duration_counter >= 1000*note_durations[current_note])
begin
            // 如果当前音符已经播放完毕, 切换到下一个音符
            current_note <= (current_note + 1) % NOTE_COUNT;
            note_counter <= 0;
            duration_counter <= 0;
        end else if (note_counter >= note_periods[current_note]) begin

```

```

        // 如果当前音符的一个周期已经完成，反转 buzzer 的状态
        buzzer <= ~buzzer;
        note_counter <= 0;
    end else begin
        // 否则，增加计数器
        note_counter <= note_counter + 1;
        duration_counter <= duration_counter + 1;
    end
end
end
end

endmodule

```

kb_decoder:

```

module kb_decoder (
    clk,    // 输入时钟信号
    kb_clk, // 输入键盘时钟信号
    data,   // 输入数据
    up,     // 输出上键信号
    down,   // 输出下键信号
    space   // 输出空格键信号
);
    input wire clk;
    input wire kb_clk;
    input wire data;
    output reg up = 0; // 初始化上键信号为 0
    output reg down = 0; // 初始化下键信号为 0
    output reg space = 0; // 初始化空格键信号为 0
    wire [7:0] keycode; // 键码
    keyboard d_1 ( // 实例化 keyboard 模块
        .clk(clk),
        .kb_clk(kb_clk),
        .data(data),
        .keycode(keycode),
        .sign(sign)
    );
    always @(posedge clk) begin // 在时钟上升沿时
        case (keycode) // 根据键码执行不同的操作
            8'h29: // 如果键码为 8'h29 (空格键)
            begin
                space <= sign; // 更新空格键信号
            end
            8'h75: // 如果键码为 8'h75 (上键)
            begin
                up <= sign; // 更新上键信号
            end
        endcase
    end
endmodule

```

```

        end
        8'h72: // 如果键码为 8'h72 (下键)
        begin
            down <= sign; // 更新下键信号
        end
    endcase
end
endmodule

```

keyboard:

```

module keyboard (
    input wire clk, // 输入时钟信号
    input wire kb_clk, // 输入键盘时钟信号
    input wire data, // 输入数据
    output reg [7:0] keycode, // 输出键码
    output reg sign // 输出信号
);
    reg [1:0] state = 0; // 状态机的状态, 00 等待, 01 输入, 10 确认, 11 结束
    reg [7:0] code_temp = 0; // 临时存储键码
    reg [2:0] cnt = 0; // 计数器
    reg confirm = 0; // 确认信号
    reg [2:0] kb_clk_old = 0; // 存储旧的键盘时钟信号
    initial begin
        keycode = 8'h00; // 初始化键码为 0
        sign = 0; // 初始化信号为 0
    end
    always @(posedge clk) begin // 在时钟上升沿时
        kb_clk_old <= {kb_clk_old[1:0], kb_clk}; // 更新键盘时钟信号
        case (state) // 根据状态机的状态执行不同的操作
            2'b00: begin // 等待状态
                if (kb_clk_old[2] & ~kb_clk_old[1]) begin
                    if (!data) begin
                        if ((&code_temp[7:4]) && ~(|code_temp[3:0])) begin
                            sign <= 0;
                        end else begin
                            sign <= 1;
                        end
                    end
                    state <= 2'b01; // 转到输入状态
                    confirm <= 0;
                end
            end
            2'b01: begin // 输入状态
                code_temp <= code_temp[7:4] | (data < 8'h0f) < 8'h0f;
                if (data < 8'h0f) begin
                    sign <= 1;
                end
                state <= 2'b10; // 转到确认状态
            end
            2'b10: begin // 确认状态
                keycode <= code_temp;
                sign <= 1;
                state <= 2'b11; // 转到结束状态
            end
            2'b11: begin // 结束状态
                keycode <= 8'h00; // 清空键码
                sign <= 0;
                state <= 2'b00; // 回到等待状态
            end
        endcase
    end
end

```

```

2'b01: begin // 输入状态
    if (kb_clk_old[2] & ~kb_clk_old[1]) begin
        if (&cnt) begin
            state <= 2'b10; // 转到确认状态
            cnt <= 0;
        end else begin
            cnt <= cnt + 1; // 计数器加1
        end
        if (data) begin
            confirm <= ~confirm; // 翻转确认信号
        end
        code_temp <= {code_temp[6:0], data}; // 更新临时键码
    end
    keycode <= 8'h00; // 清空键码
end
2'b10: begin // 确认状态
    if (kb_clk_old[2] & ~kb_clk_old[1]) begin
        if ((confirm ^ data)) begin
            state <= 2'b11; // 转到结束状态
        end else begin
            state <= 2'b00; // 转到等待状态
            code_temp <= 8'h00; // 清空临时键码
        end
    end
    keycode <= 8'h00; // 清空键码
end
2'b11: begin // 结束状态
    state <= 2'b00; // 转到等待状态
    {keycode[0],keycode[1],keycode[2],keycode[3],keycode[4],keycode
[5],keycode[6],keycode[7]} <= code_temp; // 更新键码
    {code_temp[0],code_temp[1],code_temp[2],code_temp[3],code_temp[
4],code_temp[5],code_temp[6],code_temp[7]} <= code_temp; // 更新临时键码
    end
endcase
end
endmodule

```

vga_sync:

```

`timescale 1ns / 1ps
module vga_sync (vga_clk,clrn,d_in,row_addr,col_addr,rdn,r,g,b,hs,vs);
// vgac
    input [11:0] d_in; // bbbb_gggg_rrrr, pixel
    input vga_clk; // 25MHz
    input clrn;

```

```

output reg [8:0] row_addr; // pixel ram row address, 480 (512) lines
output reg [9:0] col_addr; // pixel ram col address, 640 (1024)
pixels
output reg [3:0] r,g,b; // red, green, blue colors
output reg      rd_n;    // read pixel RAM (active_low)
output reg      hs,vs;    // horizontal and vertical
synchronization
// h_count: VGA horizontal counter (0-799)
reg [9:0] h_count; // VGA horizontal counter (0-799): pixels
always @ (posedge vga_clk) begin
    if (!clrn) begin
        h_count <= 10'h0;
    end else if (h_count == 10'd799) begin
        h_count <= 10'h0;
    end else begin
        h_count <= h_count + 10'h1;
    end
end
// v_count: VGA vertical counter (0-524)
reg [9:0] v_count; // VGA vertical counter (0-524): lines
initial begin
    h_count = 10'h0;
    v_count = 10'h0;
end
always @ (posedge vga_clk or negedge clrn) begin
    if (!clrn) begin
        v_count <= 10'h0;
    end else if (h_count == 10'd799) begin
        if (v_count == 10'd524) begin
            v_count <= 10'h0;
        end else begin
            v_count <= v_count + 10'h1;
        end
    end
end
// signals, will be latched for outputs
wire [9:0] row    = v_count - 10'd35;    // pixel ram row addr
wire [9:0] col    = h_count - 10'd143;    // pixel ram col addr
wire      h_sync  = (h_count > 10'd95);    // 96 -> 799
wire      v_sync  = (v_count > 10'd1);     // 2 -> 524
wire      read    = (h_count > 10'd142) && // 143 -> 782
                    (h_count < 10'd783) && // 640 pixels
                    (v_count > 10'd34) && // 35 -> 514
                    (v_count < 10'd515);    // 480 lines

```

```

// vga signals
always @ (posedge vga_clk) begin
    row_addr <= row[8:0]; // pixel ram row address
    col_addr <= col;      // pixel ram col address
    rdn      <= ~read;    // read pixel (active low)
    hs       <= h_sync;   // horizontal synchronization
    vs       <= v_sync;   // vertical synchronization
    r        <= rdn ? 4'h0 : d_in[3:0]; // 3-bit red
    g        <= rdn ? 4'h0 : d_in[7:4]; // 3-bit green
    b        <= rdn ? 4'h0 : d_in[11:8]; // 2-bit blue
end
endmodule

```

其余部分模块为先前上课所设计模块，此处略去。