课程名称：_____实验类型：____综合____

实验项目名称：_____课程设计——贪吃蛇游戏_____

学生姓名：_____ 学号：_____

学生姓名：_____ 学号：_____

学生姓名：_____ 学号：_____

实验地点：_____实验日期：_____ 年____ 月___日

本次实验，我们小组三人组队，选择了“贪吃蛇”游戏作为逻辑设计的主题，通过 verilog 语言进行逻辑模块的设计；同时，在交互设计方面，我们使用了开关的基础交互设计，以及 PS2 接口输入、VGA 接口、蜂鸣器等扩展接口设计；最终完整实现了贪吃蛇游戏的基本功能。

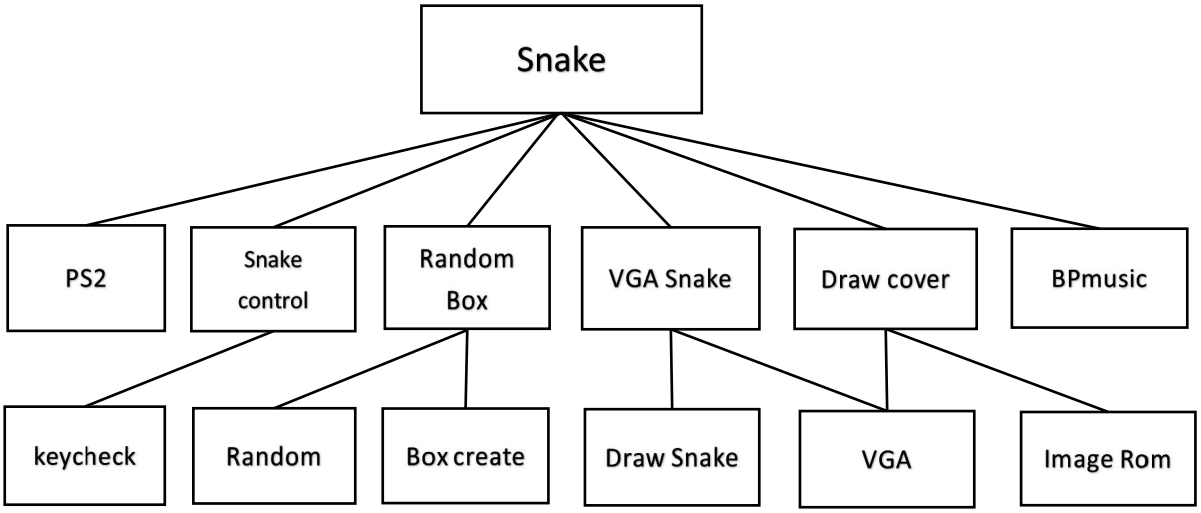
在显示上，我们采用 10*10 的像素块作为蛇身的节，20*20 的像素块作为蛇头，10*10 的像素块作为苹果（食物）。蛇头的颜色设定为黄色，蛇身的颜色设定为绿色，两者颜色不同便于观察蛇的移动状态以及长度变化，苹果颜色为红色，背景色选用黑色，更好的衬托蛇体和苹果。

在操作上，我们绑定了键盘的上下左右四个键用于控制贪吃蛇的相应移动，便于操作，同时提供了难度选择开关，游戏开始、重开开关以运行游戏。

在玩法上，蛇每吃到苹果会变长一节，长度上限为 20 节，蛇在移动中碰撞到自己就会 Game Over，蛇移动无边界限制，左边界出右边界进，其余方向同理。

一、逻辑设计思路

工程结构展示：



下面我们将逐一介绍各个模块的设计思路以及实现方式：

模块名	功能	设计思路
Keycheck	1. 按键去抖动 2. 稳定的按键检测	设计了一个 3 位寄存器“key_cnt”用于保存最近三个时钟周期内的按键状态。每个时钟周期，“key_cnt”右移一位，并将当前 key 信号放在最高位。这样，可以记录连续三个时钟周期的按键状态。使用连续三个周期内的按键状态来判定按键是否稳定按下。当“key_cnt”的值为 3'b110 时，表示按键已经稳定按下，并且无抖动信号。此时，将 key_v 置为高电平（1）。如果按键状态不是稳定按下的状态，则将“key_v”置为低电平（0）。
Random	生成伪随机数	设定一个初始数字，使用 LFSR 通过移位与异或操作来生成新的伪随机数，同时在伪随机数大小超过边界数时对伪随机数进行对边界数取模，以保证生成的坐标参数在屏幕显示范围内。
Box create	生成苹果的随机位置	基于输入的随机数“rand_num”和控制信号“rand_drive”来生成随机坐标。在一个时钟周期内，当 “rand_drive ”信号为高电平时，将当前输入的 “rand_num” 作为“rand_x ”的值；然后在下一个时钟周期，将同一个“rand_num ”作为 rand_y 的值。将新生成的“rand_num”作为下一个新的 x 坐标。
Draw snake	在显示器上绘制蛇和苹果	根据传入的“snake_head”(蛇头像素判断)、“pixel”(蛇身像素判断)，以及盒子（红色）坐标来绘制图像。对屏幕上每个像素进行判断，如果“snake_head”为“1”，该点像素为黄色；否则，如果“pixel”为“1”，该点像素为绿色；否则，如果该点坐标在盒子坐标左十、右十、上十、下十这个矩形内，该点为红色；否则为黑色（背景）

VGA	在 VGA 显示器上显示图像	直接应用实验 10 的 VGA 模块。VGA 模块利用时钟信号驱动水平和垂直计数器，根据计数器的值确定像素的行和列地址。通过判断当前位置是否在显示区域内，生成像素读取使能信号，并将输入的像素颜色数据输出为红、绿、蓝分量。同时，根据计数器值生成水平和垂直同步信号，确保显示器按照标准 VGA 规范进行扫描和同步。
Image Rom	在 VGA 显示器上显示预加载的图像数据	使用“Verilog”的系统任务 “$readmemh”从“Start_1_data.hex”文件中读取图像数据，并初始化存储在 “rom” 数组中。always 块对 “addr” 的变化敏感，每当 “addr” 发生变化时，从 “rom” 数组中读取对应地址的数据，并将其赋值给 “data” 输出。
PS2	检测键盘上下左右四个方向键的按下和释放状态，并输出对应的信号	通过时钟同步和状态机逻辑实现完整的键盘扫描和数据解析。首先侦测 PS/2 键盘的时钟信号边沿，确定数据采样时机，并使用状态机将 PS/2 数据帧解析为具体的按键操作。模块内部包括状态寄存器用于追踪数据采样过程和解析状态，以及临时存储器存储 PS/2 数据的临时变量。最终，根据解析后的按键扫描码更新上、下、左、右四个方向键的状态信号。
Random Box	生成随机苹果坐标	调用 Random 模块和 Box Create 模块，定义线网将 Random 生成的伪随机数连接到 Box Create 模块的输入，将每个时钟下，Box Create 模块输出的随机坐标数赋值给 box_x、box_y。
VGA Snake	显示游戏画面	调用 VGA 模块、draw_snake 模块，定义线网将 draw_snake 模块输出的 pixel(每个像素点的值，表示颜色)输入到 VGA 中以游戏渲染画面。
Draw cover	显示封面	调用 VGA 模块、Image Rom 模块，定义线网将 Image Rom 模块处理后输出的图片信息（每个点的像素值）输入到 VGA 的输入 Din 中以显示封面。
BPmusic	蜂鸣器，演奏游戏背景音乐	利用计数器和预分频器来控制蜂鸣器的输出频率和节奏，在每个时钟周期内，计数器 “cnt” 递增，当计数器达到预设的预分频值 “prediv” 时，蜂鸣器输出状态取反，实现音调的切换。通过调整 delay 变量和 “prediv” 的值，可以控制不同音符的持续时间和频率，从而生成音乐节奏和音调。

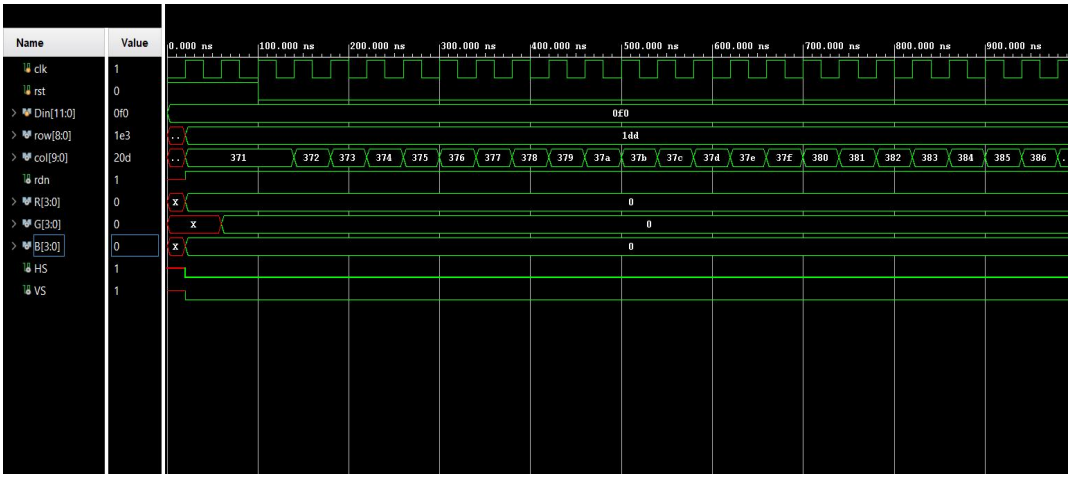
Snake control	1. 实现蛇头的移动 2. 实现蛇体位置更新 3. 实现蛇的增长 4. 速度选择 5. 蛇体检测 6. 死亡检测	1. 首先，使用一个 dir 寄存器存储蛇头当前的移动方向状态，并根据按键输入更新这个状态。然后，在每个时钟周期内，根据当前的移动方向，通过一个 case 语句实现上下左右情况下的对应坐标更新，更新蛇头的位置坐标，实现蛇头在屏幕上的移动。 2. 蛇由许多节构成，每个节有一个坐标，在每次时钟到来时，蛇头的坐标会根据当前方向“div”更新，然后将上一个时钟时上一个点（距离蛇头近的）的坐标作为下一个点（距离蛇头更远）的坐标实现的。 3. 蛇的增长通过吃苹果实现。其中首先有一个碰撞检测模块：当蛇头节点的坐标进入苹果中心点坐标左十、右十、上十、下十，形成的这个 20*20 的矩形中时，我们认为蛇吃到了苹果，此时蛇的节数（length）增 1，蛇增长。 4. 通过改变“update_time”来实现。每当计数器 counter(随时钟 clk 自增 1)到达 update_time 这个值时，我们更新一次蛇头坐标，相当于蛇移动一此。所以 update_time 越大，蛇头更新越慢，蛇移动速度越慢，反之，蛇移动速度越快。通过一个两位的输入来选择 3 个不同大小的 update_time 值来改变更新速度，这就实现了游戏难度的选择。 5. 蛇的长度上线设为 20。（length 最大值为 20）。蛇由节组成，每个节都有一个坐标，所以，我们定义了一个 20 位的向量，每个分量为 10 位，用来表示每个节的 x 坐标值，定义了一个 20 位的向量，每个分量为 9 位，用来表示每个节的 y 坐标值。初始化只设定 snake_x[0]、snake_y[0]的值（蛇头），其余均为 0，length 为 1。每吃一个苹果，length 加一，蛇体多一个节点，多加一个坐标。蛇体渲染时，每个节点为中心 20*20 的正方形渲染为蛇。 6. 每个时钟下，检测蛇头坐标 snake_x[0]、snake_y[0],如果 snake_x[0],snake_y[0]与任意一个节的坐标完全相等，那么蛇碰到了自己，游戏结束。
Snake	1. 实现封面与游戏界面的切换逻辑。 2. 时钟分频。 3. 最顶层模块，整合各个模块，实现整个游戏的逻辑	1. 定义线网存储 VGA Snake 模块和 Draw cover 模块的输出结果，根据 rst_n 的值来进行多路选择，rst_n 置 0 游戏未开始，此时 VGA 屏幕输出信号就与 Draw cover 输出相连，rst_n 置 1，游戏开始，VGA 屏幕输出信号与 VGA Snake 输出相连，显示游戏界面。 2. 处理时钟信号 clk，调整到适合游戏的频率。定义一个双位宽的寄存器 clkdiv，并在初始化阶段将其赋值为 2'b0。使用 always @(posedge clk) 块来监测时钟信号 clk 的上升沿，每次检测到上升沿时，将 clkdiv 的值递增 2'b1,以实现一个简单的时钟分频器。 3. 定义所需线网，调用所有二级模块，将游戏所需所有模块连接整合，形成整体逻辑工程。

二、仿真与调试过程分析

如前一部分所述，我们的工程共由上面的 11 个模块组成。我将对其中的一部分模块进行仿真并给出分析。

1. VGA 模块

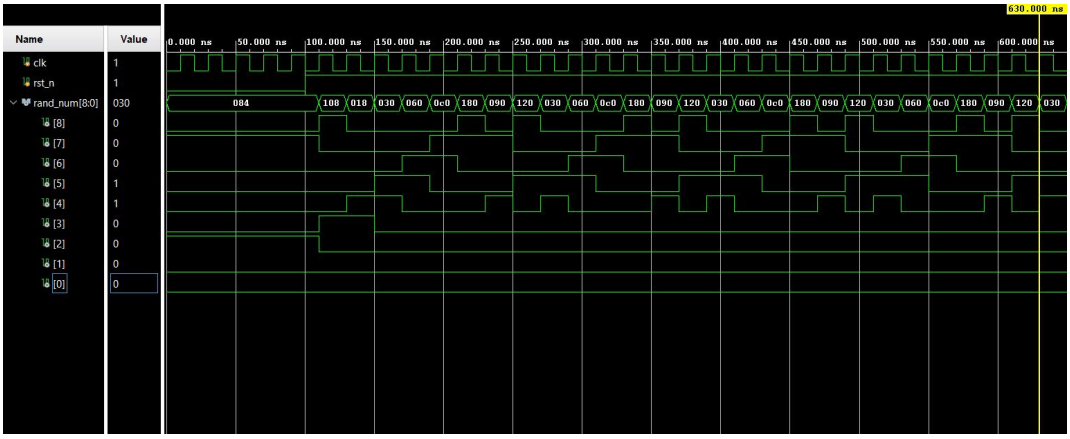
我们的 VGA 模块是从实验十中取得，应用到本游戏中时做了一些较小的调整。下图是 VGA 模块的仿真结果。



从仿真结果中可以看出，我们的 VGA 模块可以正常工作，扫描屏幕上的每个像素点并产生输出 VGA 信号。

2. random 模块

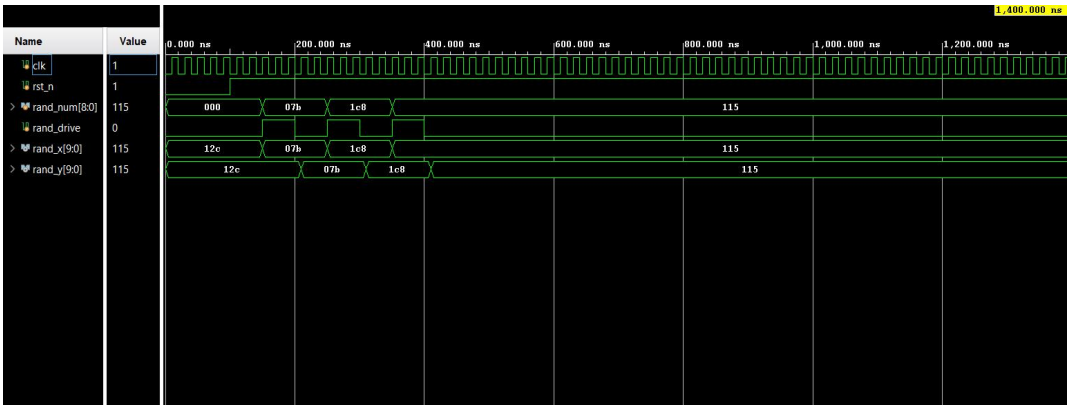
如前所述，random 模块的作用就是产生伪随机数。给出仿真结果如下：



在 rst_n 为 1 时，本模块随着时钟 clk 的节奏生成伪随机数；在 rst_n 为 0 时停止生成伪随机数。我们的 random 模块可以正常工作。

3. box_create 模块

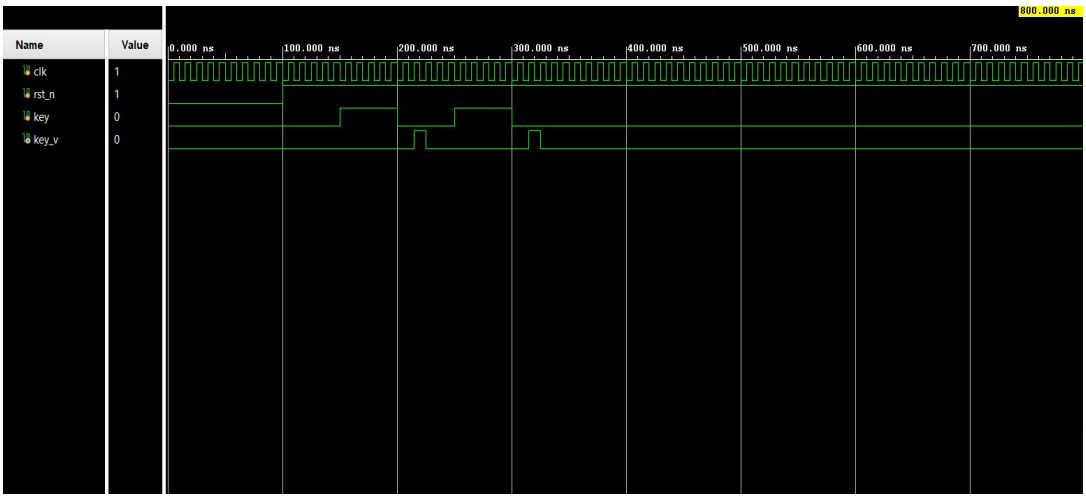
如前所述，box_create 模块的作用是根据 random 模块产生的随机数产生 box 的中心坐标。给出仿真结果如下：



可以看到，在 rand_num 发生改变之后，rand_x 和 rand_y 也会相应发生变化。我们成功根据 rand_num 的值确定了 box 的中心位置的坐标。我们的 box_create 模块可以正常工作。

4. keycheck 模块

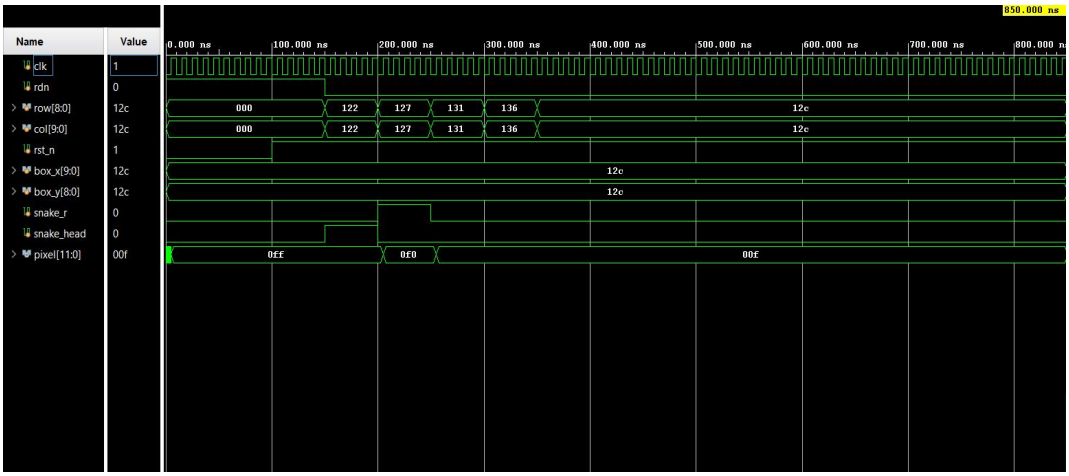
如前所述，keycheck 模块的功能就是检测某个按键是否按下。仿真结果如下：



从仿真结果中可以看出，每当 key 变为 1（模拟按下按键）之后，我们的输出 key_v 也会变为 1，表示检测到按键按下；一个时钟周期之后将 key_v 将变回 0，表示抬起。整个模块在 rst_n 为 1 时工作。我们的 keycheck 模块可以正常工作。

5. draw_snake 模块

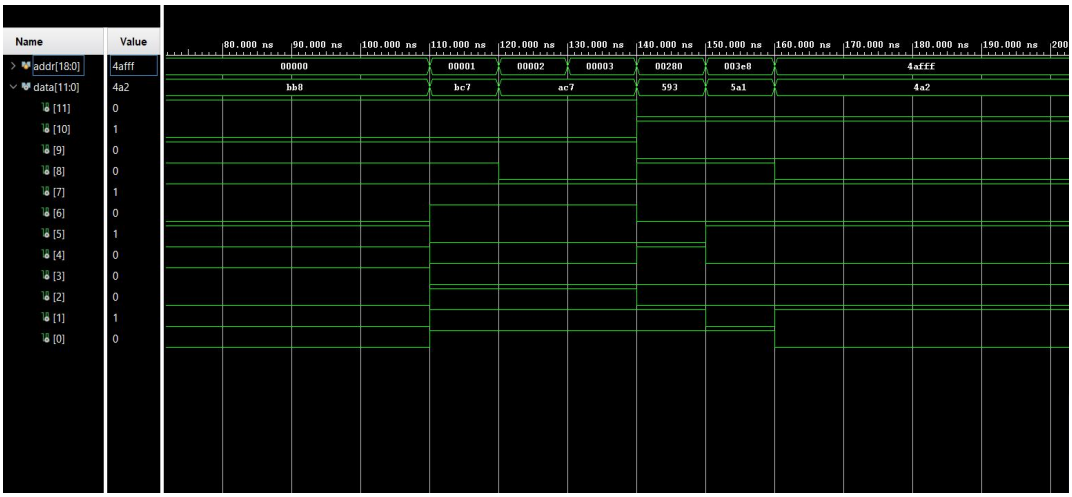
如前所述，draw_snake 模块的作用主要是产生用于显示蛇的信号。此外，它还有一个附加功能——接受 box_create 模块产生的 box 中心的坐标，用于产生用于显示 box 的信号；仿真结果如下：



可以看到，我们的 draw_snake 模块产生了 row，col 的输出，根据不同的 snake_r 产生了不同的 pixel 输出。pixel 信号将用于显示蛇和 box。我们的 draw_snake 模块可以正常工作。

6. image_rom 模块

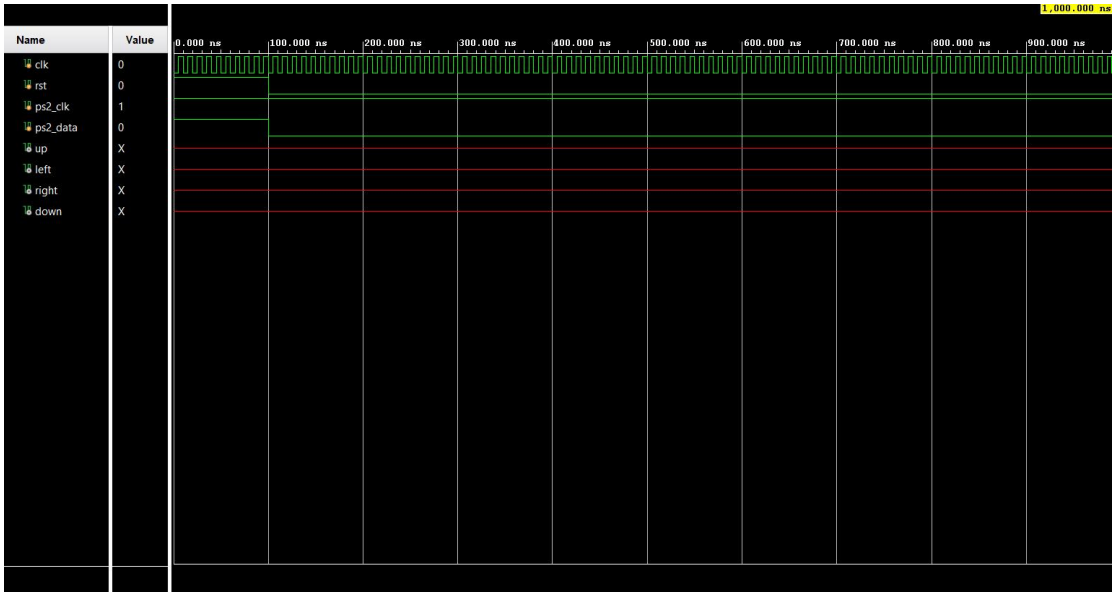
image_rom 模块用于读取开始界面的.hex 文件，生成 VGA 信号用于显示开始界面。在我们的影片中，开始界面能够正常显示，开关能够正常操控，已经能说明该模块能够正常工作。下面给出仿真结果：



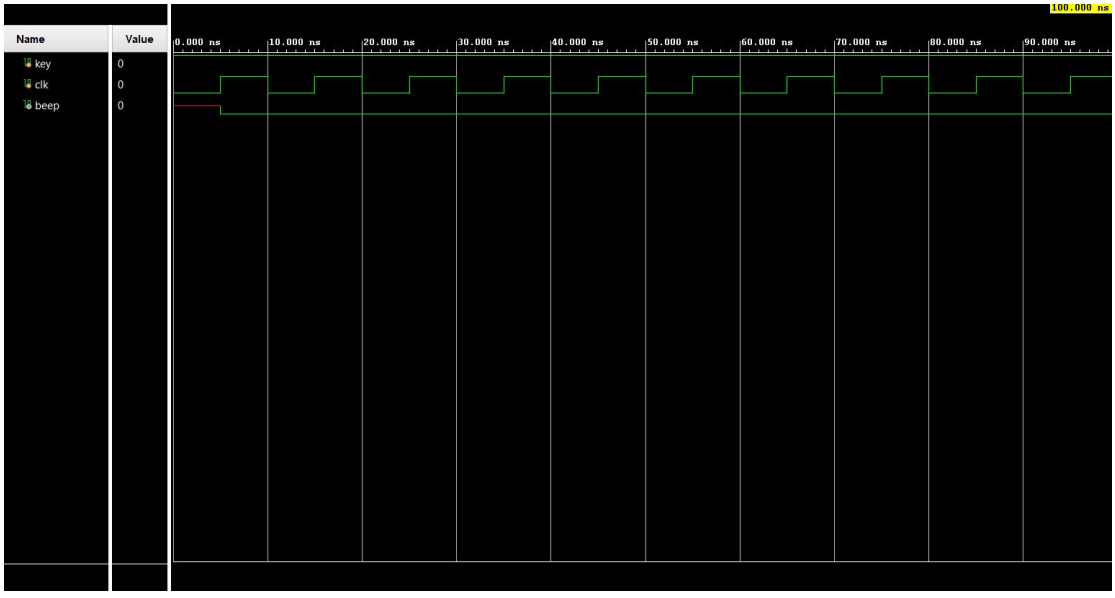
到此为止，我们已经对除了 PS2 和 Bpmusic 之外的所有最底层模块进行了仿真并给出了结果和分析。

PS2 模块涉及到从键盘处取得输入，在这里不予分析；Bpmusic 模块涉及到向蜂鸣器输出，在这里也不予分析。从我们的影片中可以得知：我们可以通过键盘操作蛇的移动，从而 PS2 模块能够正常工作；我们可以通过开关控制背景音乐，从而 Bpmusic 模块能够正常工作。直接给出这两个模块的仿真结果如下：

PS2 模块的仿真结果：

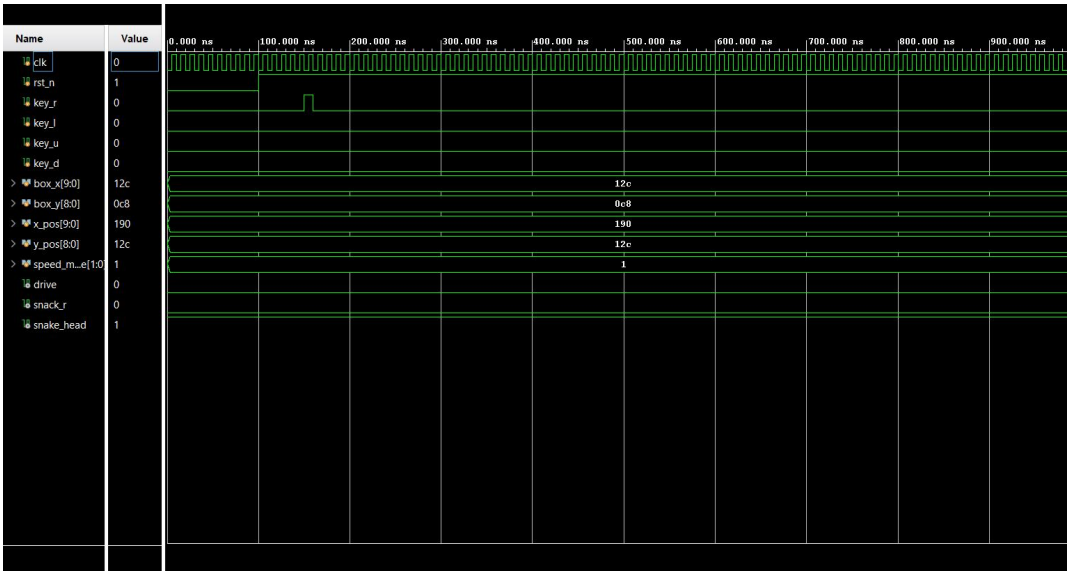


Bpmusic 模块的仿真结果：

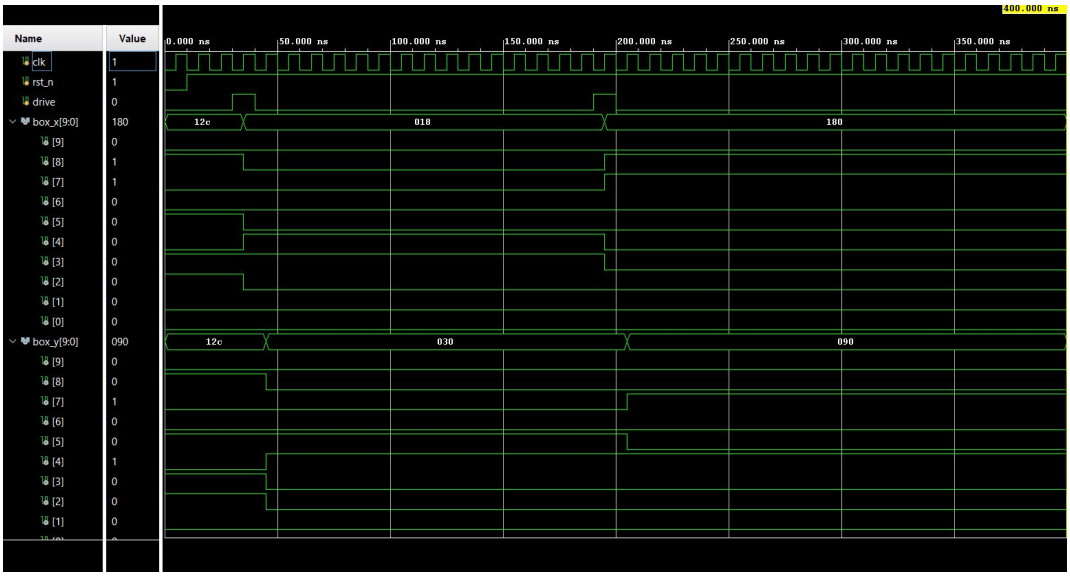


综合上述底层模块的仿真结果，我们可以得知 snake_control，random_box 和 draw_cover 模块，乃至顶层 snake 模块的大致情况。下面给出三个二级模块的仿真结果：

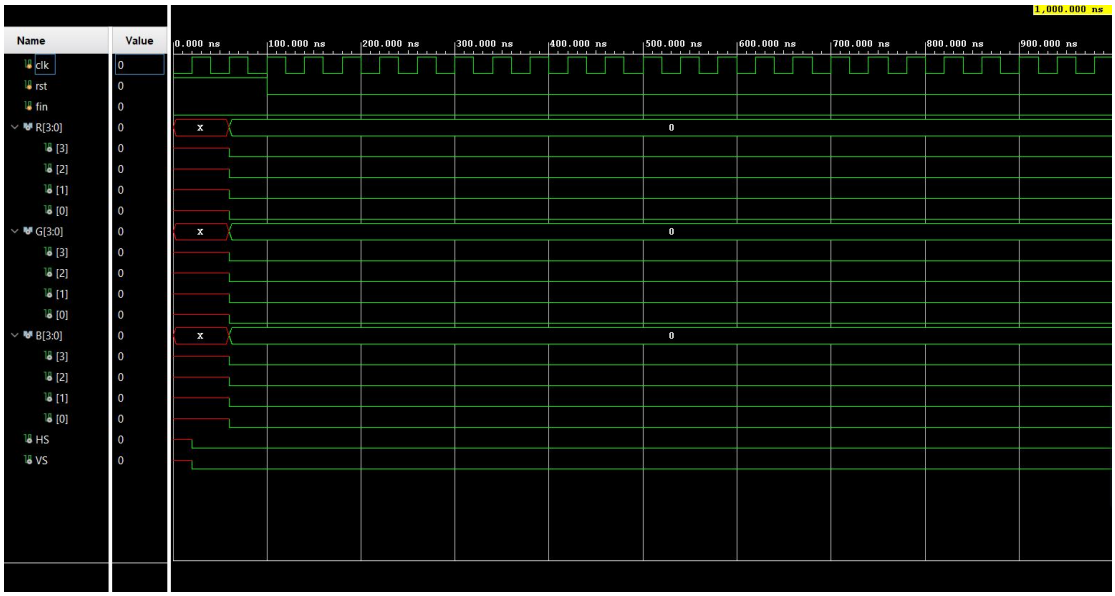
1. snake_control 模块



2. random_box 模块

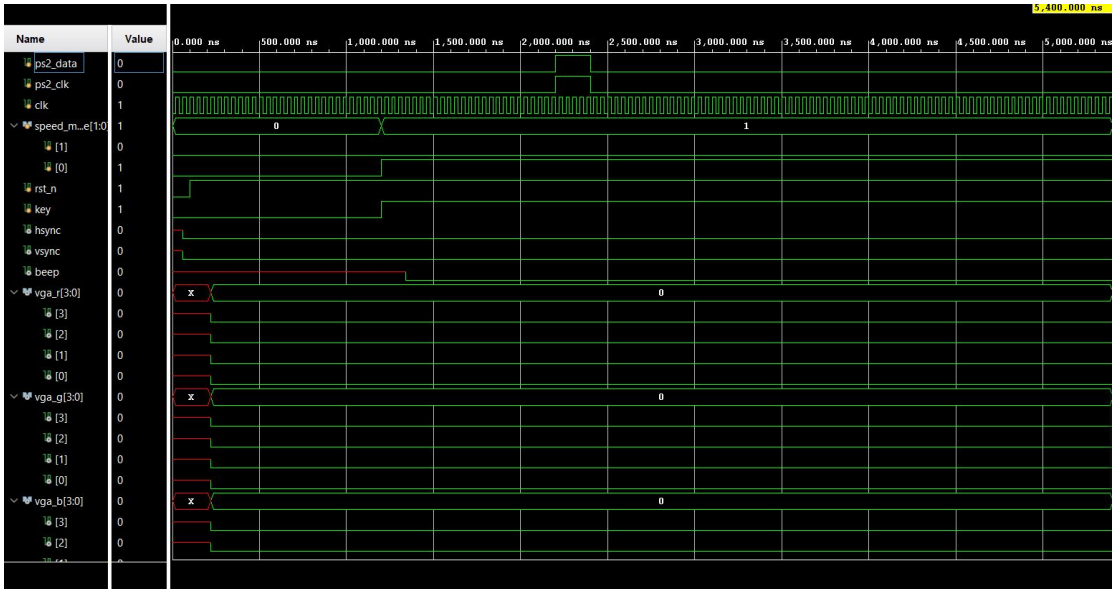


3. draw_cover 模块



最后给出顶层模块的仿真结果：

Snake 模块：



三、 上版结果与分析

1. 上版设备

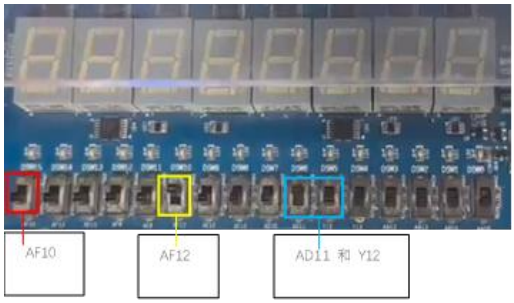
在这个实验中，我们的交互界面由板上的按钮和 PS2 控制器的四个方向按键（上、下、左、右）这两种设备组成。

版上的按钮

PS2

2. 按钮的介绍

2.1. 版上的按钮



面板上按钮的操作如下：

1. AF10 按钮：开始界面和重新开始按钮
   - 当按钮显示为 0 时，电脑会显示开始界面。
   - 当按钮显示为 1 时，电脑会显示游戏界面。
2. AF12 按钮：蜂鸣器控制按钮
   - 当按钮显示为 1 时，蜂鸣器不会响。
   - 当按钮显示为 0 时，蜂鸣器会按照我们设置的代码响起。
3. AD11 和 Y12 按钮：游戏难度控制按钮
   - 游戏难度分为简单模式、中等模式和困难模式三个等级。
   - 当 AD11 和 Y12 分别为 0 和 0 时，游戏为简单模式。
   - 当 AD11 和 Y12 分别为 0 和 1 时，游戏为中等模式。
   - 当 AD11 和 Y12 分别为 1 和 0 时，游戏为困难模式。

2.2. PS2 按钮

PS2 按钮的操作如下：

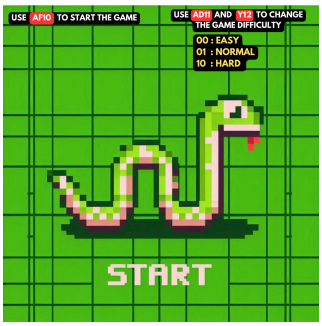
1. 上下左右按钮 ：这四个按钮用于控制蛇的移动方向。

3. 开始界面和游戏界面

3.1. 开始界面

像上面的按钮介绍，当开始界面按钮为 0 的时候，电脑会 显 示开始界面（开始界面如下）：

上板后的图



原图

3.2. 游戏界面

游戏界面介绍：

- 红点：游戏界面显示的红点为 random box 也就是蛇吃的东西
- 黄色：蛇头
- 绿色：蛇身
- 黑色：背景

游戏开始的蛇

蛇变长

- 当蛇头撞到蛇的身体时，显示蛇死亡（游戏结束）

蛇死亡

四、 进一步的优化

1. **我们的游戏只有开始界面。在蛇死亡之后，游戏画面将直接保持静止，并没有结束的界面。**
   实际上，在我们小组的设计中，是有结束界面的考虑的。我们准备的结束界面如下：

但是我们遇到了如下错误：

[DRC UTLZ-1] Resource utilization: LUT as Logic over-utilized in Top Level Design (This design requires more LUT as Logic cells than are available in the target device. This design requires 112822 of such cell types but only 101400 compatible sites are available in the target device. Please analyze your synthesis results and constraints to ensure the design is mapped to Xilinx primitives as expected. If so, please consider targeting a larger device. Please set tcl parameter "drc.disableLUTOverUtilError" to 1 to change this error to warning.)

最后很遗憾未能实现结束界面。
理论上，我们只需要在游戏结束（判定死亡触发）时调用有关模块，显示结束界面即可。显示结束界面的思路与显示开始界面的思路实际上是一样的。

2. **我们的游戏背景是纯黑**
   如果我们能实现 3.中的 VGA 显示分数，很容易地，我们就也能显示特定图片作为游戏背景。这将使得我们的游戏更加美观。

3. **我们没有计分模块**
   实际上，我们已经有了计分模块的核心。每当蛇头与 box 足够接近时，我们判定蛇“吃到”了 box，蛇增长，重新生成 box，这是我们已经实现了的。其实我们只需要另外实现一个计数器，在判定蛇“吃到”了 box 时，触发计数器自增，即可实现计分功能。一个较为简单的输出分数的方式是调用七段数码管，这在先前的实验中已经完成过。另一个更加高级的输出方式是调用 VGA，在屏幕的某个固定位置显示当前分数。

4. **我们没能实现游戏暂停**
   我们希望的效果是，在某个条件触发（玩家主动暂停）时，显示暂停界面，保留当前进度；当玩家选择开始时，回到游戏并继续。显示暂停界面实际上与显示开始界面和 1.中的显示结束界面的思路是相同的。但游戏暂停的实现十分复杂，主要困难在于保持当前游戏状态。比游戏暂停更加简单的是直接重新开始。

5. **我们在难度选择和重新开始的时候依旧使用了按钮交互**
   既然使用了 PS2 从键盘输入，我们实际上是可以将所有输入都用 PS2 实现的

6. **我们的背景音乐只是单音的罗列。整体上并不是很好听。其实可以结合音乐知识丰富一下我们的背景音乐，从而增强游戏的可玩性。**

五、 小组成员分工及贡献比例

姓名	学号	占比	签名

六、 实现过程中所参考的所有代码的链接

蜂鸣器参考：

https://blog.csdn.net/weixin_46049055/article/details/121201679?ops_request_misc=%257B%2522request%255Fid%2522%253A%25221718624053168002150273 63%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=171862405316800215027363&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-4-121201679-null-null.142^v100^pc_search_result_base4&utm_term=%E8%9C%82%E9%B8%A3%E5%99%A8FGPA&spm=1018.2226.3001.4187

PS2 参考：

https://github.com/PAN-Ziyue/FPGA--JOJO/blob/master/JOJO/Framework/PS2.v

VGA 参考：