同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2251079

姓　　名 隋建政

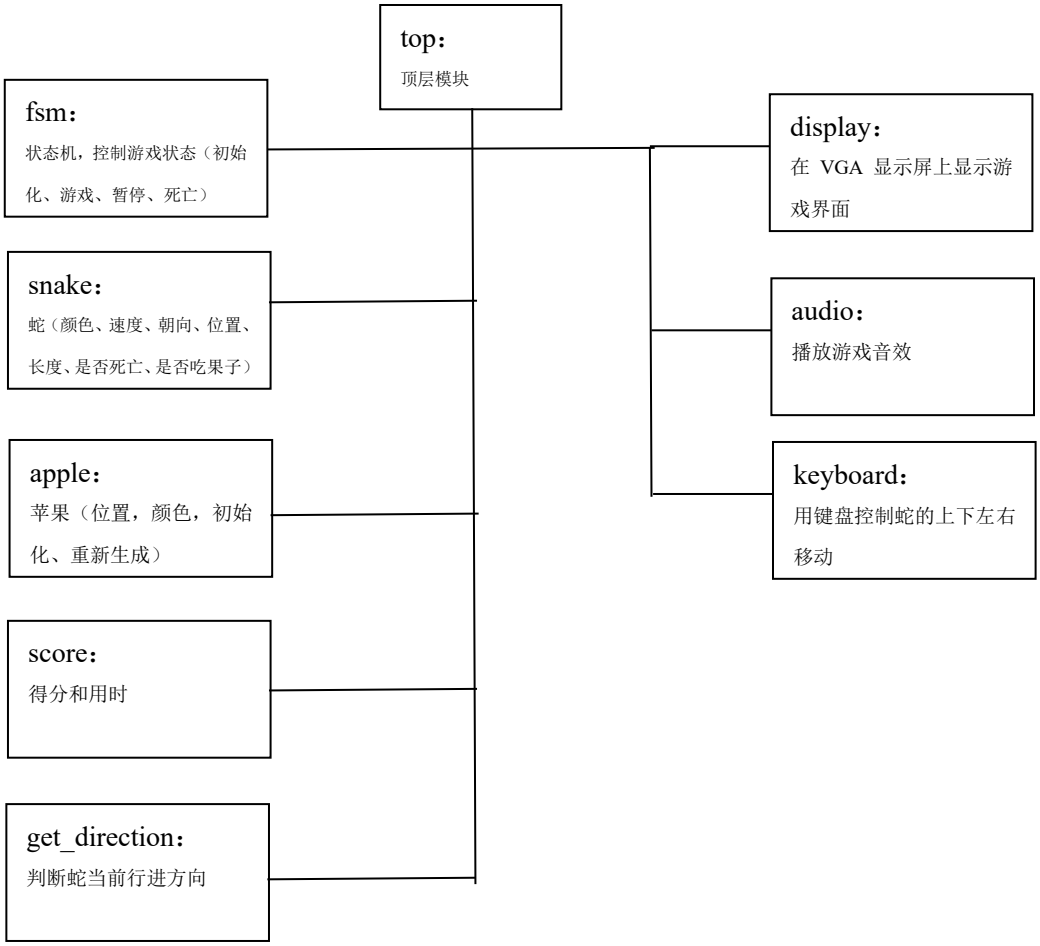
专　　业 计算机科学与技术

授课老师 郭玉臣

一、实验内容

基于 NEXYS A7 开发板，利用 VGA 显示屏和键盘制作的贪吃蛇小游戏。

二、贪吃蛇游戏数字系统总框图



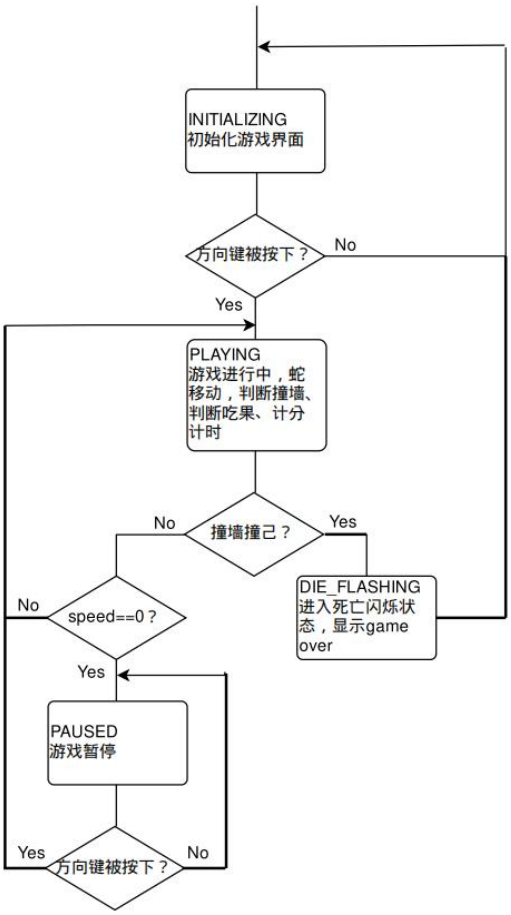
三、系统控制器设计

1、游戏状态控制

（1） 状态设置

```
localparam PAUSED=2'b00;        //暂停
localparam PLAYING=2'b01;       //游戏中
localparam DIE_FLASHING=2'b10; //死亡闪烁
localparam INITIALIZING=2'b11; //初始化
```

（2）ASM 流程图



（3）状态转移真值表

Bn	An	Bn+1	An+1	条件
0	0	0	0	
		0	1	up||down||left||right
0	1	0	0	speed==0
		0	1	
		1	0	hit_wall||hit_self
1	0	1	1	
1	1	1	1	
		0	1	up||down||left||right

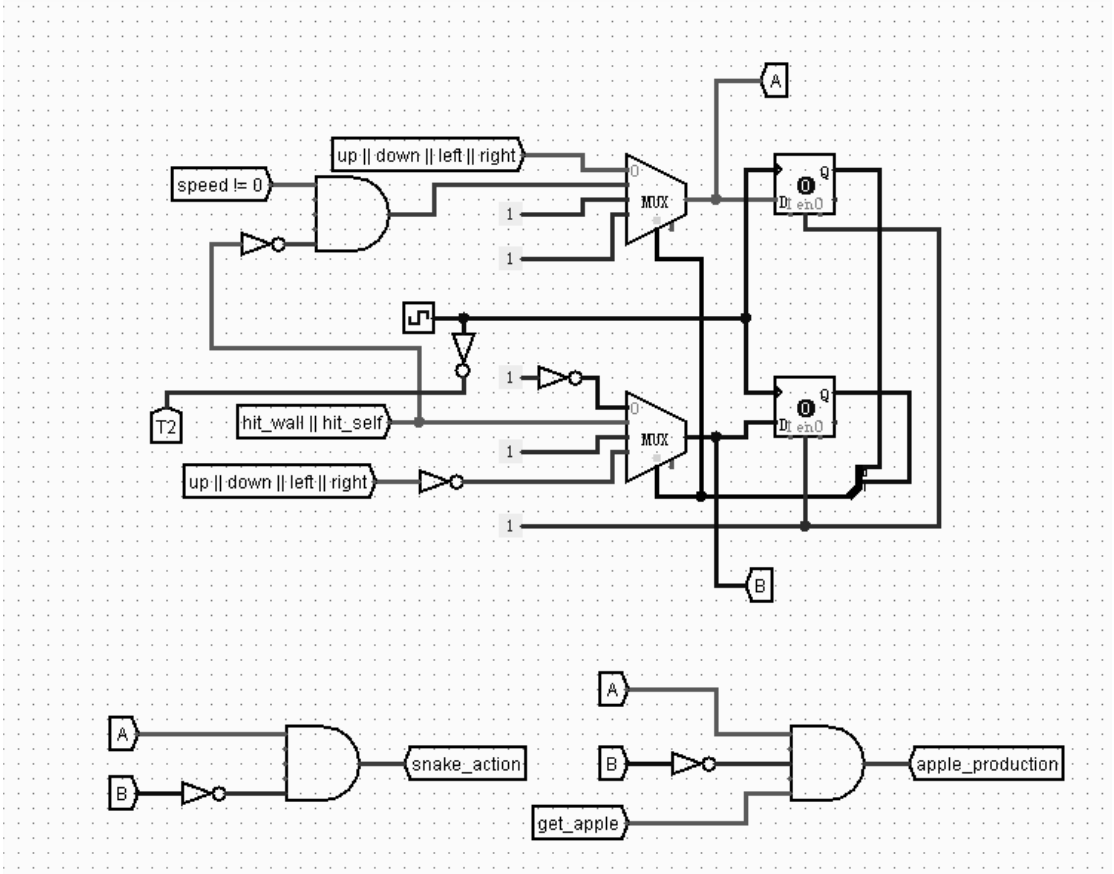
（4）次态激励函数表达式和控制命令逻辑表达式

$$B^{n+1} = A^n(hit\_wall + hit\_self) + B^n\overline{(hit\_wall + hit\_self)}$$

$$A^{n+1} = \overline{B^n}\overline{A^n}(up + down + left + right) + \overline{B^n}A^n(speed == 0)\overline{\left(hit_{wall} + hit_{self}\right)} + B^n$$

$$snake\_action = \overline{B^n}A^n \qquad apple\_production = \overline{B^n}A^n get\_apple$$

（5）系统控制器逻辑方案图



四、子系统模块建模

（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模，图文描述，描述清楚设计及实现的思路，不要在此处放 verilog 代码，所有的 verilog 代码在附录部分）

1、设备连接模块

(1) display 模块

(a) 建模

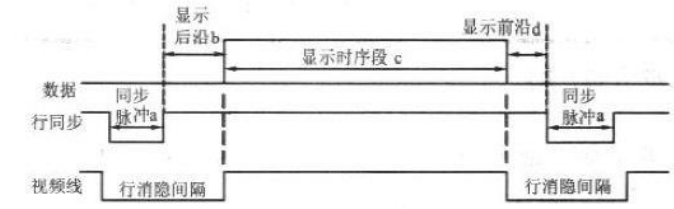
```
module display(
input clock, // 148.5MHZ，用于输出 1920x1080@60Hz 的 VGA 信号
input [5:0] apple_x, apple2_x, apple3_x, apple4_x, apple5_x, // 苹果位置
input [5:0] apple_y, apple2_y, apple2_y, apple4_y, apple5_y,
input [5:0] wall_x, wall2_x, wall3_x, wall4_x, // 障碍物位置
input [5:0] wall_y, wall2_y, wall3_y, wall4_y,
input [5:0] wall5_x, wall6_x, wall7_x, wall8_x,
input [5:0] wall5_y, wall6_y, wall7_y, wall8_y,
input [32*6-1:0] snake_x_temp, // 蛇的临时信号
input [32*6-1:0] snake_y_temp,
input [31:0] snake_piece_is_display, // 需要显示的蛇身（长度）
```

```
input [1:0] game_status, //  游戏状态
input [2:0] snake_color, //蛇的颜色
output h_sync,v_sync, //  行、场扫描信号
output reg [11:0] vga // vga 显示输出
);
```

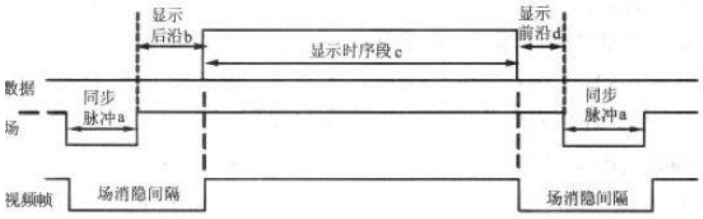
(b) 功能描述

该模块根据游戏中各个状态的信息，将相应的游戏画面显示在 VGA 屏幕上。

VGA 显示图像使用的是扫描的方式，从第一行的第一个像素点开始逐个填充，一直到最后一行的最后一个像素点。每一帧图像都通过这种扫描的方式构成，当扫描的速度足够快，由于人眼的视觉暂留特性，人们就会看到一幅完整的图像。VGA 显示需要两个时序相互配合，分别为行同步信号与场同步信号，通过这两个信号完成一帧的扫描。



VGA 的行时序



VGA 的场时序

VGA **的两种时序（图片来源：**https://blog.csdn.net/liuligui5200）

在该模块的功能实现中，首先需要判断游戏状态。若游戏状态处于初始化，则除默认的围墙、障碍物、蛇以及苹果的初始位置外，还需要在屏幕上方生成游戏标题 SNAKE；当玩家按下按键进入游戏状态时，屏幕上方的游戏标题会消失，蛇也会进行移动，当玩家操控蛇吃掉某个苹果后，该苹果会消失，并在随机一个位置生成新的苹果，同时蛇身的长度也会增加一格。

当游戏处于暂停状态时，整体界面与游戏进行中时相同但蛇不会进行移动；当蛇死亡时即进入死亡闪烁状态时，蛇身会闪烁持续一段时间，而屏幕中央会显示 GAME OVER 的标识，一段时间后自动进入初始化界面。

在实际游戏内容物的演示上，首先蛇的颜色可以进行操控，其由三位输入控制，总共有七种选择，且蛇头与蛇身做出差异（蛇头更亮一些让玩家更好区分）。墙和障碍物的颜色均为白色。而苹果有两种颜色，红色的苹果为普通苹果，而金

色的苹果为超级苹果，当蛇吃掉某种苹果时，会生成一个新的与老苹果属性相同的苹果。

(2) keyboard 模块

(a) 建模

```
module keyboard (clk,rst,key_clk,key_data,kup,kdown,kleft,kright);
input clk; //  系统时钟
input rst; //  系统复位，低有效
input key_clk; // PS2 键盘时钟
input key_data; // PS2 键盘数据输入
output reg kup,kdown,kleft,kright; //  输出对应的上下左右信号
```

(b) 功能描述

即使不用键盘，也是可以控制蛇的行进的，在最初的开发中就是利用开发板上的 M18、P18、P17、M17 四个按键控制蛇的方向。当考虑到玩家的方便，遂增加键盘控制的方式。

PS/2 键盘履行一种双向同步串行协议，即每次数据线上发送以为数据，且每在时钟线上发一个脉冲就被读入，键盘与主机可以双向通信。键盘的扫描码有两种不同的类型：通码和断码。当键被按下就发送通码，当键被释放就发送断码。传输信号时，键盘使用一种每帧包含 11 位的串行协议，只需要分析 8 位数据位

第 N 位	属性
0	开始位
1~8	数据位
9	校验位
10	结束位

键盘传输的 11 位数据（图片来源：https://zhuanlan.zhihu.com/p/384221079）

就可以获取我们需要的键盘输入信息。

在该模块中，对键盘输入的数据进行处理，并将产生相应的操作指令，输出到 kup、kdown、kleft 和 kright 中。这四个方向信号与默认输入的四个信号地位等同，共同控制游戏的开始与蛇的行进。

(3) audio 模块

(a) 建模

```
module Audio(
input clk, //  时钟信号
input hit_wall, //  撞墙信号
input hit_itself, //  撞己信号
input get_apple, get_apple2, get_apple3, get_apple4, get_apple5,
//  吃到苹果的信号
input up,right,down,left, //  方向键信号
output AUD_PWM, //  音频输出波形
output AUD_SD //  音频输出控制
);
```

(b) 功能描述

该模块通过游戏进行中发生事件的信号来控制 AUDIO 音频的输出，当发生不同的事件时，会产生不同音高的提示音加以区分。例如吃到超级苹果的提示音高于吃到普通苹果，转向时的提示音较第而撞墙时的提示音最高。

2、游戏相关模块

(1) fsm 模块

(a) 建模

```
module fsm(
input reset, //  复位信号
input clock, //  时钟信号
input hit_wall, //  撞墙
input hit_itself, //  撞自身
input up,right,down,left, //  上下左右信号
input kup,kright,kdown,kleft,
input pause, //  暂停信号
output reg [1:0] game_status //  输出游戏状态
 );
```

(b) 功能描述

该模块实现的是整个游戏状态的时序转移，game_status 共有四个状态，游戏初始化（INITIALIZING）、游戏进行中（PLAYING）、游戏暂停（PAUSED）以及死亡闪烁（DIE_FLASHING）。

程序运行时先进入初始化状态，此时若方向键有输入则会立即进入游戏状态。在游戏状态中，要随时检测暂停信号以及蛇的状态，若这些信号都没有变化则持续游戏，若暂停信号置高位，则进入暂停信号，若检测到蛇死亡（撞到墙或者撞到自己的身体），就进入死亡闪烁状态）。暂停状态与初始化状态类似，遇到方向键信号则回到游戏状态，否则就一直保持。而死亡闪烁状态一旦进入，会持续一段时间，之后自行回到初始化状态。

(2) snake 模块

(a) 建模

```
module snake(
input clock, //  时钟信号
input pause, //  暂停信号
input [2:0]speed, //  蛇的行进速度
input [1:0] next_direction, //  蛇的下一个方向
input [1:0] game_status, //  游戏状态
input [5:0] apple_x, apple2_x, apple3_x, apple4_x, apple5_x, //  苹果位置
input [5:0] apple_y, apple2_y, apple2_y, apple4_y, apple5_y,
input [5:0] wall_x, wall2_x, wall3_x, wall4_x, //  障碍物位置
input [5:0] wall_y, wall2_y, wall3_y, wall4_y,
input [5:0] wall5_x, wall6_x, wall7_x, wall8_x,
```

```
input [5:0] wall5_y, wall6_y, wall7_y, wall8_y,
output reg [1:0] current_direction, // 蛇的当前方向
output [32*6-1:0] snake_x_temp, // 蛇的临时信号
output [32*6-1:0] snake_y_temp,
output reg [31:0] snake_piece_is_display, // 控制蛇体长
output reg get_apple, get_apple2, get_apple3, get_apple4, get_apple5,
// 吃到苹果的信号
output reg hit_wall, // 撞墙信号
output reg hit_itself // 撞己信号
);
```

(b) 功能描述

snake 模块完成对蛇的控制，判断在某一个状态下蛇的对应状态，并生成对应的输出信号。

当游戏状态处于初始化时，初始化蛇的出生位置，固定在屏幕左下方，长度为 3；当游戏状态处于死亡闪烁时，进行计数并使蛇的样式处于一明一暗；当游戏状态处于进行中的状态时，首先判断蛇的当前位置有没有撞到墙壁、障碍物或者撞到自己的身体，然后需要判断蛇的当前位置有没有与苹果的位置重合，如果有重合则需要将蛇身加长一格，前两项判断完毕后，需要对蛇的位置进行更新，根据当前蛇的朝向，将蛇身的每个方块向相应方向推进一格即可。以上检查的状态均被输出到标志信号中，便于其他模块进行相应的操作。

(3) apple 模块

(a) 建模

```
module apple(
input clock, // 时钟信号
// 蛇吃到苹果的信号
input reg get_apple, get_apple2, get_apple3, get_apple4, get_apple5,
input [1:0] game_status, // 游戏状态
// 输出新的苹果坐标
output reg [5:0] apple_x, apple2_x, apple3_x, apple4_x, apple5_x,
output reg [5:0] apple_y, apple2_y, apple3_y, apple4_y, apple5_y,
);
```

(b) 功能描述

该模块控制苹果的生成。当游戏状态处于初始化时，按默认的状态生成五个苹果的位置；当游戏状态处于进行状态时，随时检测五个苹果中哪个苹果被蛇吃掉，若有苹果被吃掉，则在随机位置生成一个新的苹果（用五个信号分别表示是因为苹果的状态有所不同，有吃掉加 1 分的普通苹果和吃掉加 3 分的超级苹果，因此需要对每个苹果加以区分），其余没有被吃掉的苹果还按原位置不变。新的五个苹果的位置作为状态输出，在 display 和其他模块中参与代码。

(4) score 模块

(a) 建模

```
module score(
input clock, // 时钟信号
input reset, // 复位信号
input get_apple, get_apple2, get_apple3, get_apple4, get_apple5,
input [1:0] game_status, // 游戏状态
output [7:0] an, // 数码管使能
output [7:0] seg // 数码管输出
 );
```

(b) 功能描述

该模块实现游戏得分和游戏时间的记录，并将其显示在数码管上。首先要检测复位信号，若复位信号有效，则游戏得分和游戏时间都清零；然后还要检测得分情况，在该设计中，苹果 1245 均为普通苹果，若检测到他们的获取信号，则总分加 1，而苹果 3 为超级苹果，若检测到 get_apple3 有效，则得分加 3；最后要对了游戏时间进行更新，若游戏不处在暂停状态，则在总时间以 s 为单位加 1。最后将游戏得分和时间两个数显示在数码管上，每个数据占 4 位。

(5) get_direction 模块

(a) 建模

```
module get_direction(
 input clock, // 时钟信号
 input up,right,down,left, // 方向信号
 input kup,kdown,kleft,kright,
 input [1:0] current_direction, // 当前行进方向
 output reg [1:0] next_direction // 输出下一个行进方向
 );
```

(b) 功能描述

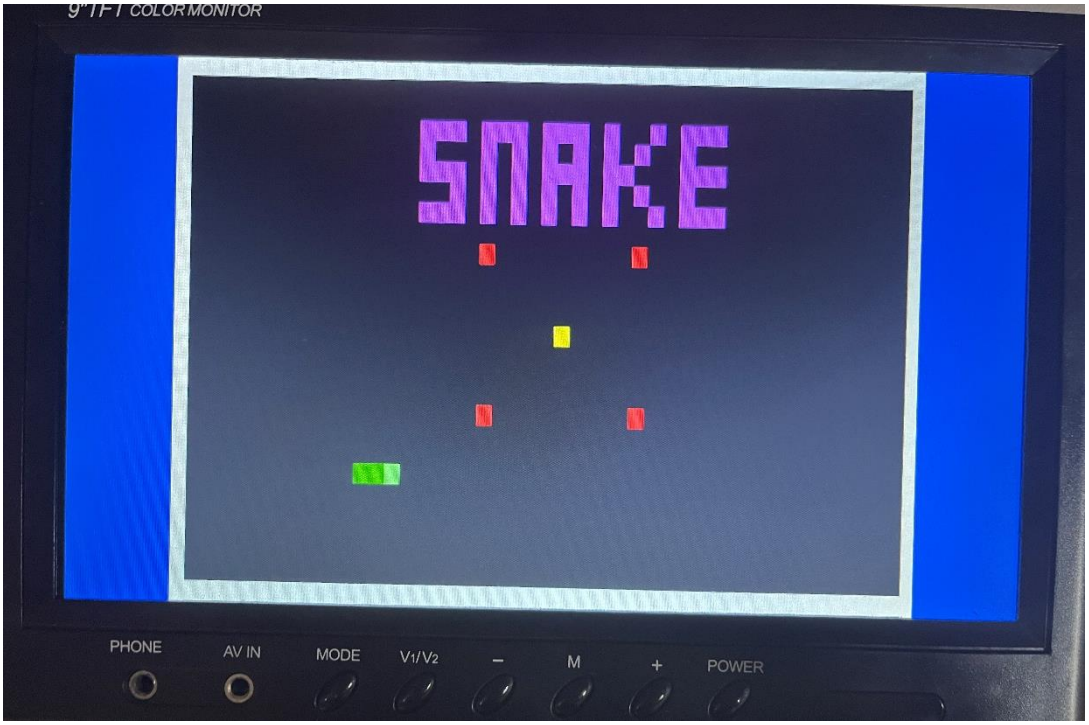
该模块实现了蛇的行进方向的更新，通过当前行进方向与玩家输入的方向键信号相结合，需要注意的是，若玩家新输入的方向与当前行进方向平行时（相同方向或相反方向），下一个行进状态不需要更新，即与当前行进方向相同；而若玩家新输入的行进方向与当前行进方向垂直，则将下一个行进方向更新为玩家输入的方向

五、测试模块建模

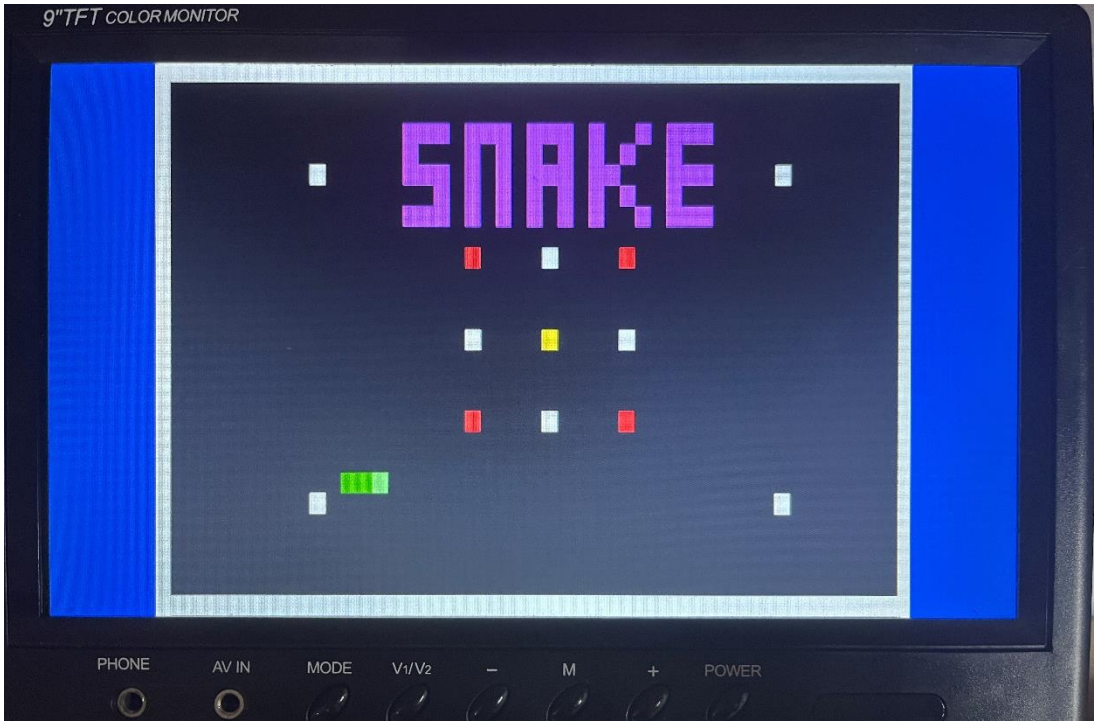
无 test_bench，在完成该项目的过程中，一直利用直接观察 VGA 显示屏上的运行结果的方式来进行测试与验证。

六、实验结果

1、初始化界面

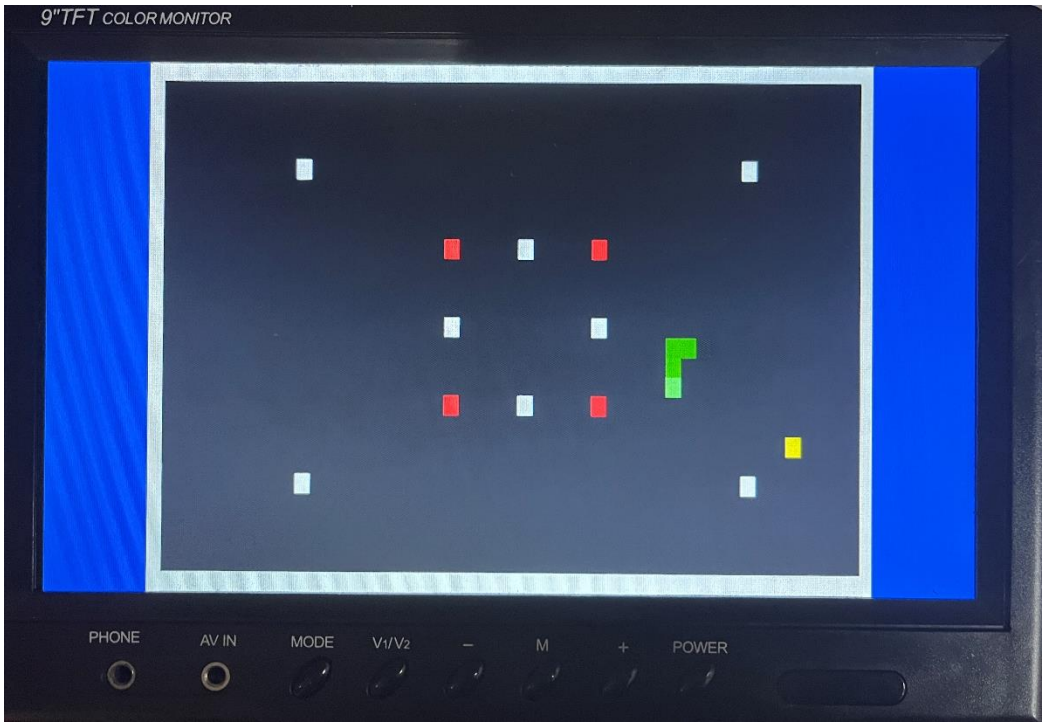


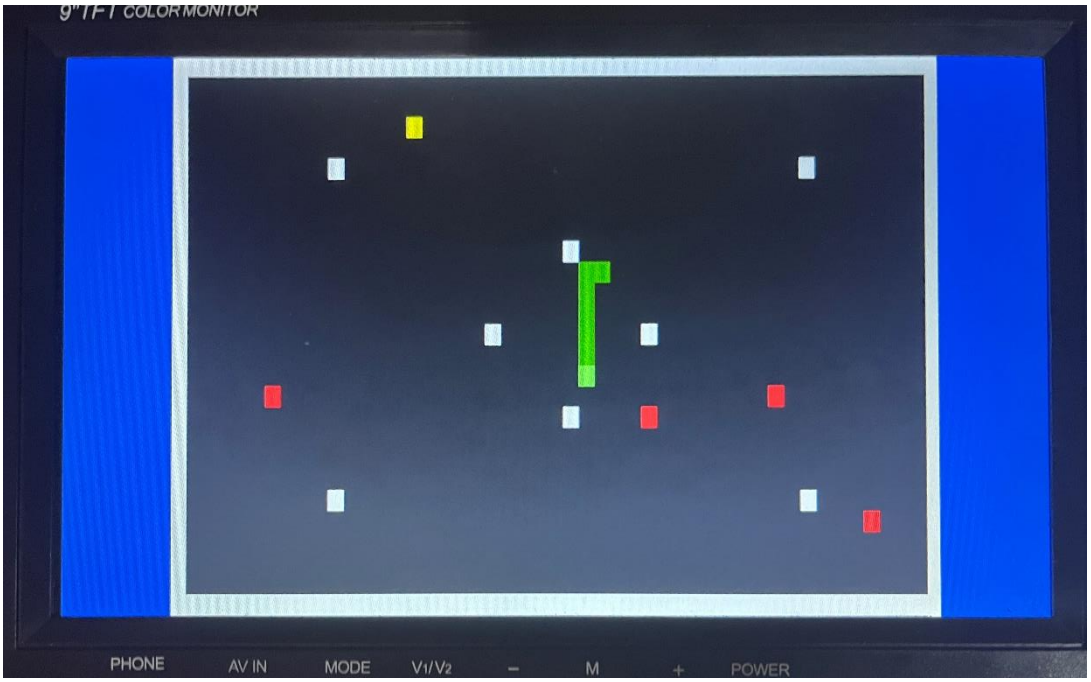
初始化界面除了正常的游戏画面外，还在屏幕上方显示游戏的标题 SNAKE。可以选择不同的游戏难度，上图为没有障碍物的简单模式，下面两图分别为带有 4 个障碍物的中等模式和带有 8 个障碍物的困难模式。（难度模式通过开发板上的开关控制）

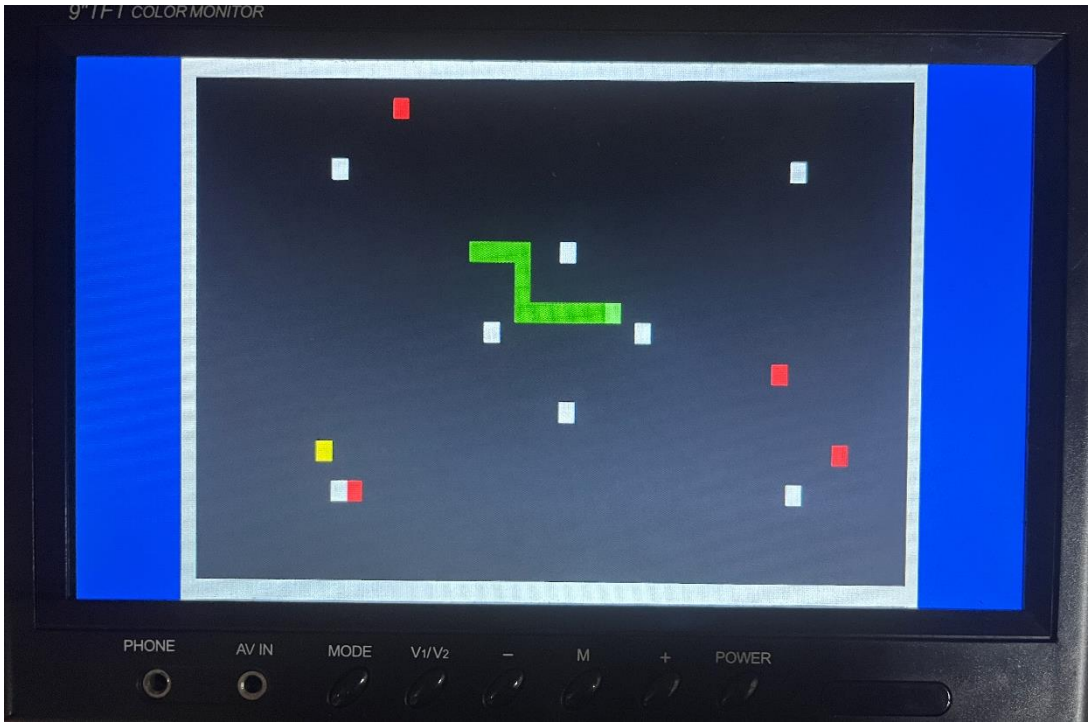


2、游戏进行界面

按下方向键即可开始游戏界面，通过方向键控制蛇行进的方向，界面中四个红色和一个黄色的像素点代表苹果，红色的普通苹果吃掉可以加 1 分，黄色的超级苹果吃掉可以加 3 分，吃掉苹果后蛇的身长会增加一格。界面中白色的部分为围墙和障碍物，蛇头一旦撞上它们就会死亡。





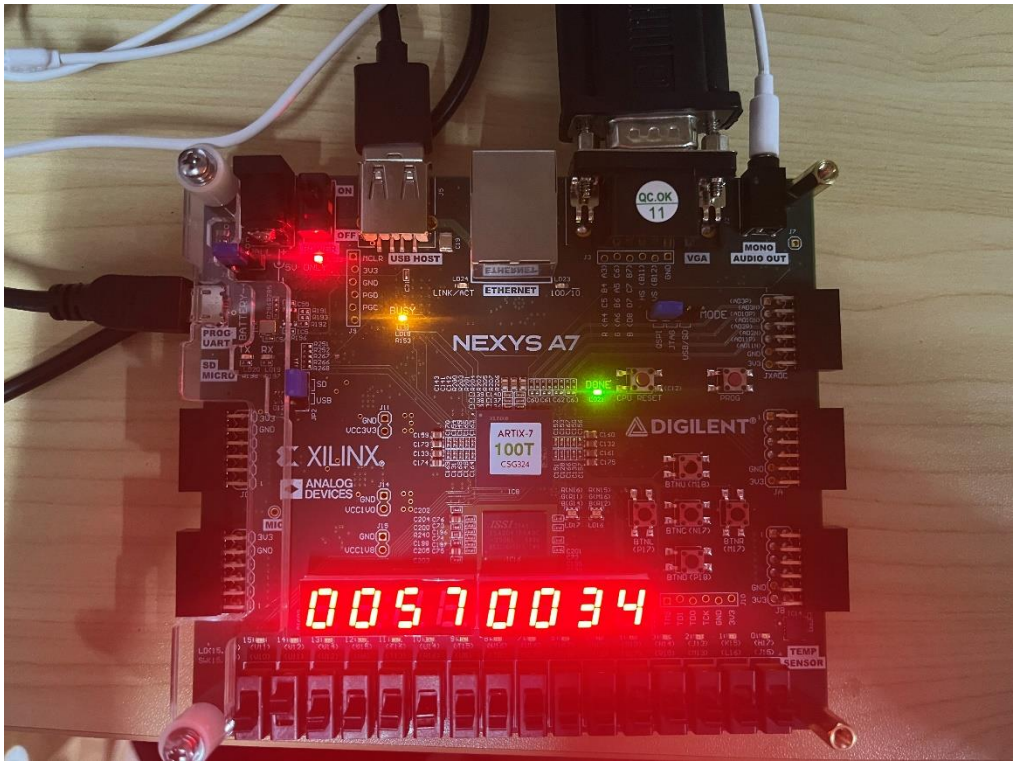


3、死亡闪烁

当蛇头触碰到墙壁、障碍物或者自己的身体时，蛇便会死亡，游戏也因此结束。在这个状态下，蛇身会闪烁一段时间，而屏幕正中央会出现显示 GAME OVER 的提示框。只需等待一段时间，该提示框便会消失，游戏界面重新回到最初的初始化界面，等待下一次游戏的开始。

4、计分与计时

计分与计时通过开发板上的数码管显示实现，其中左侧为记录的游戏时间，右侧为记录的游戏分数。



七、结论

该项目使用 NEXYS A7 开发板、VGA 显示屏以及键盘实现了一个贪吃蛇小游戏。在数字系统的设计上，采用自顶向下的设计模式，将众多模块集成在 top 模块中，再分别设计每个子模块。在众多子模块中，最核心的起到枢纽作用的无疑是调控整个苏数字系统各个状态的控制器模块，在该模块中通过输入以及游戏中各个参数，进行游戏状态的转移，再以不同的游戏状态调控其他模块的分配与运作。

在游戏设计方面，将游戏中出现的多个元素分别进行建模，分成 snake、apple、score、get_direction 等几个模块，在状态时序的调控下对游戏中的各个属性进行更新与处理，并显示在 VGA 屏幕上。

游戏软件部分的设计完成后，就是开发板与外设硬件的连接模块了。首先利用两种时序配合，完成 VGA 屏幕的图像显示，将整个图像划分为小的像素块，并通过为像素块的 RGB 赋值完成颜色的区分。当游戏系统中的属性由不同状态更迭而产生变化时，VGA 显示屏也会实时的显示出来。使用 AUDIO 为游戏中的事件增加了不同音高的音效处理，但只有戴上耳机时才可以接收到声音信号。最后还为游戏操作增添了键盘输入的方式，通过对 PS/2 键盘协议中传输数据的处理读入键盘的输入，并以此控制游戏的进程，使得操作更加简便和人性化。

综合以上诸多模块，完成了最终的游戏系统的制作。

八、心得体会及建议

通过本项目的开发，我体会了自己设计数字系统的完整体验，切实感受了什么叫“自顶向下”的设计思路。在总体设计时，首先要对系统中可能出现的各个状态进行考量，为不同的状态分配相应的任务。为了实现状态之间的转移，需要设计 ASM 流程图、状态转移表等一系列图表来对逻辑进行提取和化简，最终得以完成控制器的设计，可以说这部分是一个数字系统的灵魂所在。

而在整个项目之下，又需要将多个小模块进行建模，对要解决的问题进行抽象，减少问题的复杂度。子模块需要根据不同的大状态做出相应的操作，因此在子模块的建模实现时不能忽略一些细节的小问题。当所有子模块都设计完成后，整个项目才可以运行并达到自己的预期。可以切实体会到整个设计流程是十分严谨且有规律性的，在开发过程中，每个开发环节的任务和目的都十分明确，而各个开发环节之间也不应存在过多的交叉，否则就会容易产生混乱或者错误。

最后是还学习了一些硬件设备与开发板这种类似 CPU 的设备之间的通讯协议，大体了解一些设备诸如 VGA 显示屏和 AUDIO 音频设备的实现原理。并利用这些通信协议中的特殊规定，完成了不同设备之间的互通，例如完成数字信号向 VGA 显示设备的输出，也完成了 PS/2 键盘向开发板中数字信号的输入。

最后是一些小小建议，在申请外设时我并没有申请 PS/2 键盘，最终的实现我是使用了自己购买的三模键盘，其效果也是非常的好。但是根据一些申领了 PS/2 键盘的同学的反馈，领取到的 PS/2 键盘并不能很好的使用，因此我也将自己的键盘借给了一些需要的同学，直接使用自带的 USB-TypeC 线连接就可以很好的实现需求。因此若还希望设计使用键盘输入的项目，可以考虑就用自己的键盘，目前市面上的键盘基本都是兼容 PS/2 协议的，因此不必担心协议不互通的问题。

九、附录

1、top 模块

```
`timescale 1ns / 1ps
module top(
    input CLK100MHZ, // 100MHz 时钟
    input reset, // 复位信号
    input up,right,down,left, // 方向键，上下左右
    input [2:0] speed, // 速度(0 为暂停)
    input [2:0] snake_color, //蛇的颜色(低位至高位分别为 R、G、B)
    input difficult, //难度
    input difficult2,
    output [7:0] an, // 数码管使能
    output [7:0] seg, // 数码管输出
    output [11:0] vga, // vga 显示输出，顺序为 R,G,B 各 4 位
    output h_sync,v_sync, // 行、列扫描信号
    output AUD_PWM,      //音频输出波形
    output AUD_SD,       //音频输出控制
    input key_data,
    input key_clk
    );

    localparam PAUSED=2'b00;          //暂停
    localparam PLAYING=2'b01;         //游戏中
    localparam DIE_FLASHING=2'b10; //死亡闪烁
    localparam INITIALIZING=2'b11; //初始化

    wire is_pause;
    assign is_pause = (speed == 3'b0) ? 1'b1 : 1'b0;
    wire [32*6-1:0] snake_x_temp; // 蛇身坐标临时变量
    wire [32*6-1:0] snake_y_temp; // 蛇身坐标临时变量

    wire [31:0] snake_piece_is_display;
    wire [5:0] apple_x; // 苹果坐标
    wire [5:0] apple_y; // 苹果坐标
    wire [5:0] apple2_x;
    wire [5:0] apple2_y;
    wire [5:0] apple3_x;
    wire [5:0] apple3_y;
    wire [5:0] apple4_x;
    wire [5:0] apple4_y;
    wire [5:0] apple5_x;
    wire [5:0] apple5_y;
```

```
wire [5:0] wall_x;
wire [5:0] wall_y;
assign wall_x = difficult?10:0;
assign wall_y = difficult?5:0;
wire [5:0] wall2_x;
wire [5:0] wall2_y;
assign wall2_x = difficult?10:0;
assign wall2_y = difficult?21:0;
wire [5:0] wall3_x;
wire [5:0] wall3_y;
assign wall3_x = difficult?40:0;
assign wall3_y = difficult?5:0;
wire [5:0] wall4_x;
wire [5:0] wall4_y;
assign wall4_x = difficult?40:0;
assign wall4_y = difficult?21:0;

wire [5:0] wall5_x;
wire [5:0] wall5_y;
assign wall5_x = (difficult & difficult2)?25:0;
assign wall5_y = (difficult & difficult2)?9:0;
wire [5:0] wall6_x;
wire [5:0] wall6_y;
assign wall6_x = (difficult & difficult2)?25:0;
assign wall6_y = (difficult & difficult2)?17:0;
wire [5:0] wall7_x;
wire [5:0] wall7_y;
assign wall7_x = (difficult & difficult2)?20:0;
assign wall7_y = (difficult & difficult2)?13:0;
wire [5:0] wall8_x;
wire [5:0] wall8_y;
assign wall8_x = (difficult & difficult2)?30:0;
assign wall8_y = (difficult & difficult2)?13:0;
wire [1:0] game_status;

// 方向  00: UP      01: Right      10: DOWN      11: LEFT
wire [1:0] current_direction;
wire [1:0] next_direction;

wire get_apple; // 吃到苹果
wire get_apple2;
wire get_apple3;
wire get_apple4;
wire get_apple5;
```

```
wire hit_wall;
wire hit_itself;

wire kup,kdown,kleft,kright;//键盘控制的方向键

display (
    .clock(CLK100MHZ), //  在 display 内部做时钟转换
    .h_sync(h_sync),
    .v_sync(v_sync),
    .vga(vga),
    .game_status(game_status),
    .apple_x(apple_x),
    .apple_y(apple_y),
    .apple2_x(apple2_x),
    .apple2_y(apple2_y),
    .apple3_x(apple3_x),
    .apple3_y(apple3_y),
    .apple4_x(apple4_x),
    .apple4_y(apple4_y),
    .apple5_x(apple5_x),
    .apple5_y(apple5_y),
    .wall_x(wall_x),
    .wall_y(wall_y),
    .wall2_x(wall2_x),
    .wall2_y(wall2_y),
    .wall3_x(wall3_x),
    .wall3_y(wall3_y),
    .wall4_x(wall4_x),
    .wall4_y(wall4_y),
    .wall5_x(wall5_x),
    .wall5_y(wall5_y),
    .wall6_x(wall6_x),
    .wall6_y(wall6_y),
    .wall7_x(wall7_x),
    .wall7_y(wall7_y),
    .wall8_x(wall8_x),
    .wall8_y(wall8_y),
    .snake_x_temp(snake_x_temp),
    .snake_y_temp(snake_y_temp),
    .snake_piece_is_display(snake_piece_is_display),
    .snake_color(snake_color)
    );

snake (
```

```
        .clock(CLK100MHZ),
        .speed(speed),
        .snake_x_temp(snake_x_temp),
        .snake_y_temp(snake_y_temp),
        .apple_x(apple_x),
        .apple_y(apple_y),
        .apple2_x(apple2_x),
        .apple2_y(apple2_y),
        .apple3_x(apple3_x),
        .apple3_y(apple3_y),
        .apple4_x(apple4_x),
        .apple4_y(apple4_y),
        .apple5_x(apple5_x),
        .apple5_y(apple5_y),
        .wall_x(wall_x),
        .wall_y(wall_y),
        .wall2_x(wall2_x),
        .wall2_y(wall2_y),
        .wall3_x(wall3_x),
        .wall3_y(wall3_y),
        .wall4_x(wall4_x),
        .wall4_y(wall4_y),
        .wall5_x(wall5_x),
        .wall5_y(wall5_y),
        .wall6_x(wall6_x),
        .wall6_y(wall6_y),
        .wall7_x(wall7_x),
        .wall7_y(wall7_y),
        .wall8_x(wall8_x),
        .wall8_y(wall8_y),
        .snake_piece_is_display(snake_piece_is_display),
        .get_apple(get_apple),
        .get_apple2(get_apple2),
        .get_apple3(get_apple3),
        .get_apple4(get_apple4),
        .get_apple5(get_apple5),
        .game_status(game_status),
        .current_direction(current_direction),
        .next_direction(next_direction),
        .hit_wall(hit_wall),
        .hit_itself(hit_itself)
        );

    fsm (
```

```
        .reset(reset),
        .clock(CLK100MHZ),
        .game_status(game_status),
        .hit_wall(hit_wall),
        .hit_itself(hit_itself),
        .up(up),
        .right(right),
        .down(down),
        .left(left),
        .kup(kup),
        .kdown(kdown),
        .kleft(kleft),
        .kright(kright),
        .pause(is_pause)
        );

    get_direction Get_direction (
        .clock(CLK100MHZ),
        .up(up),
        .right(right),
        .down(down),
        .left(left),
        .kup(kup),
        .kdown(kdown),
        .kleft(kleft),
        .kright(kright),
        .current_direction(current_direction),
        .next_direction(next_direction)
        );

    apple (
        .clock(CLK100MHZ),
        .apple_x(apple_x),
        .apple_y(apple_y),
        .apple2_x(apple2_x),
        .apple2_y(apple2_y),
        .apple3_x(apple3_x),
        .apple3_y(apple3_y),
        .apple4_x(apple4_x),
        .apple4_y(apple4_y),
        .apple5_x(apple5_x),
        .apple5_y(apple5_y),
        .get_apple(get_apple),
        .get_apple2(get_apple2),
```

```
            .get_apple3(get_apple3),
            .get_apple4(get_apple4),
            .get_apple5(get_apple5),
            .game_status(game_status)
            );
        score (
            .reset(reset),
            .clock(CLK100MHZ),
            .get_apple(get_apple),
            .get_apple2(get_apple2),
            .get_apple3(get_apple3),
            .get_apple4(get_apple4),
            .get_apple5(get_apple5),
            .game_status(game_status),
            .an(an),
            .seg(seg)
            );
         Audio AUD (
             .clk(CLK100MHZ),
             .hit_wall(hit_wall),
             .get_apple(get_apple),
             .get_apple2(get_apple2),
             .get_apple3(get_apple3),
             .get_apple4(get_apple4),
             .get_apple5(get_apple5),
             .hit_itself(hit_itself),
             .up(up),
             .right(right),
             .down(down),
             .left(left),
             .AUD_PWM(AUD_PWM),
             .AUD_SD(AUD_SD)                );

         keyboard (
            .clk(CLK100MHZ),
            .rst(1),
            .key_clk(key_clk),
            .key_data(key_data),
            .kup(kup),
            .kdown(kdown),
            .kleft(kleft),
            .kright(kright)
         );
endmodule
```

2、 fsm 模块

```
`timescale 1ns / 1ps
module fsm(
    input reset,
    input clock,
    input hit_wall,
    input hit_itself,
    input up,right,down,left,
    input kup,kright,kdown,kleft,
    input pause,
    output reg [1:0] game_status
    );
    localparam PAUSED=2'b00;          //暂停
    localparam PLAYING=2'b01;         //游戏中
    localparam DIE_FLASHING=2'b10; //死亡闪烁
    localparam INITIALIZING=2'b11; //初始化
    reg [27:0] count_two; // count_two 用来做死亡闪烁的延时

    initial
    begin
    game_status<=INITIALIZING;
    count_two<=0;
    end

    always @(posedge clock)
    begin
        if (reset==1)
            game_status<=INITIALIZING;
        else if (pause == 1)
            game_status <= PAUSED;
        else if (game_status==PLAYING && (hit_wall==1 || hit_itself==1))
        begin
        game_status<=DIE_FLASHING;
        count_two<=0;
        end
        else if (game_status==DIE_FLASHING)
        begin
            if (count_two==200000000)
            begin game_status<=INITIALIZING; count_two<=0; end
            else count_two <= count_two+1; // count_two 用来做死亡闪烁的延时
        end
        else if ((game_status == INITIALIZING || game_status == PAUSED)
    && ( up==1 || right==1 || down==1 || left==1 || kup==1||kdown==1||kleft==1||kright==1))
```

```
    begin
        game_status <= PLAYING; //  按下任意按键时游戏开始
        end
    end
endmodule
```

3、 snake 模块

```
`timescale 1ns / 1ps
module snake(
    input clock,
    input [2:0]speed,
    input [1:0] next_direction,
    input [1:0] game_status,
    input [5:0] apple_x,
    input [5:0] apple_y,
    input [5:0] apple2_x,
    input [5:0] apple2_y,
    input [5:0] apple3_x,
    input [5:0] apple3_y,
    input [5:0] apple4_x,
    input [5:0] apple4_y,
    input [5:0] apple5_x,
    input [5:0] apple5_y,
    input [5:0] wall_x,
    input [5:0] wall_y,
    input [5:0] wall2_x,
    input [5:0] wall2_y,
    input [5:0] wall3_x,
    input [5:0] wall3_y,
    input [5:0] wall4_x,
    input [5:0] wall4_y,
    input [5:0] wall5_x,
    input [5:0] wall5_y,
    input [5:0] wall6_x,
    input [5:0] wall6_y,
    input [5:0] wall7_x,
    input [5:0] wall7_y,
    input [5:0] wall8_x,
    input [5:0] wall8_y,

    output reg [1:0] current_direction,
    output [32*6-1:0] snake_x_temp,
    output [32*6-1:0] snake_y_temp,
```

```
output reg [31:0] snake_piece_is_display, //  控制蛇体长
output reg get_apple,
output reg get_apple2,
output reg get_apple3,
output reg get_apple4,
output reg get_apple5,
output reg hit_wall,
output reg hit_itself
);

wire is_pause;
assign is_pause = (speed == 0) ? 1'b1 : 1'b0;

reg [25:0] count;
reg [25:0] count_two;
reg [31:0] snake_piece_is_display_origin; //  存储体长的旧值，用于死亡闪烁

localparam PAUSED=2'b00;
localparam PLAYING=2'b01;
localparam DIE_FLASHING=2'b10;
localparam INITIALIZING=2'b11;

localparam UP=2'b00;
localparam RIGHT=2'b01;
localparam DOWN=2'b10;
localparam LEFT=2'b11;

// snake_x[0]：头的横坐标    snake_y[0]:头的纵坐标
reg [5:0] snake_x [31:0];
reg [5:0] snake_y [31:0];

genvar i;
generate for (i=0;i<32;i=i+1)
begin
      assign snake_x_temp[6*i+:6]=snake_x[i];
      assign snake_y_temp[6*i+:6]=snake_y[i];
end endgenerate

initial
begin
count<=0;
count_two<=0;
end
```

```
integer k;
integer hitflag;
always @(posedge clock)
begin
    hitflag=0;
    if (count_two>=40000000) count_two<=0;
    else count_two<=count_two+1;

    if (game_status==INITIALIZING)
    begin
    snake_piece_is_display<=32'b00000000_00000000_00000000_00000111;
    snake_x[0]<=14;
    snake_y[0]<=20;
    snake_x[1]<=13;
    snake_y[1]<=20;
    snake_x[2]<=12;
    snake_y[2]<=20;
    current_direction<=RIGHT;
    hit_wall<=0;
    hit_itself<=0;
    end

    else if (game_status==DIE_FLASHING)
    begin
    // 闪烁
    if (count_two==20000000) snake_piece_is_display<=0;
    else if (count_two==0) snake_piece_is_display<=snake_piece_is_display_origin;
    end

    else if (game_status==PLAYING && (is_pause == 0))
    begin
        snake_piece_is_display_origin<=snake_piece_is_display;

        if  (snake_x[0]==0  ||  snake_x[0]==47  ||  snake_y[0]==0  ||  snake_y[0]==26)
            hit_wall<=1;
        else if(snake_x[0] == wall_x && snake_y[0] == wall_y) hit_wall<=1;
        else if(snake_x[0] == wall2_x && snake_y[0] == wall2_y) hit_wall<=1;
        else if(snake_x[0] == wall3_x && snake_y[0] == wall3_y) hit_wall<=1;
        else if(snake_x[0] == wall4_x && snake_y[0] == wall4_y) hit_wall<=1;
        else if(snake_x[0] == wall5_x && snake_y[0] == wall5_y) hit_wall<=1;
        else if(snake_x[0] == wall6_x && snake_y[0] == wall6_y) hit_wall<=1;
        else if(snake_x[0] == wall7_x && snake_y[0] == wall7_y) hit_wall<=1;
        else if(snake_x[0] == wall8_x && snake_y[0] == wall8_y) hit_wall<=1;
        else hit  wall<=0; //  是否撞墙
```

```
if(snake_x[0]==snake_x[k]
&& snake_y[0]==snake_y[k]
&& snake_piece_is_display[k]==1) hitflag=1;
      end
if(hitflag==1) hit_itself<=1;
else hit_itself<=0; //  是否撞自己
if (snake_x[0]==apple_x && snake_y[0]==apple_y) get_apple<=1;
else get_apple<=0; //  是否吃到苹果
if(snake_x[0]==apple2_x && snake_y[0]==apple2_y) get_apple2<=1;
else get_apple2<=0;
if(snake_x[0]==apple3_x && snake_y[0]==apple3_y) get_apple3<=1;
else get_apple3<=0;
if(snake_x[0]==apple4_x && snake_y[0]==apple4_y) get_apple4<=1;
else get_apple4<=0;
if(snake_x[0]==apple5_x && snake_y[0]==apple5_y) get_apple5<=1;
else get_apple5<=0;

if (get_apple == 1 )
begin
snake_piece_is_display<=2*snake_piece_is_display+1; //  增长
get_apple<=0;
end
if(get_apple2 == 1)
begin
snake_piece_is_display<=2*snake_piece_is_display+1;
get_apple2<=0;
end
if(get_apple3 == 1)
begin
snake_piece_is_display<=2*snake_piece_is_display+1;
get_apple3<=0;
end
if(get_apple4 == 1)
begin
snake_piece_is_display<=2*snake_piece_is_display+1;
get_apple4<=0;
end
if(get_apple5 == 1)
begin
snake_piece_is_display<=2*snake_piece_is_display+1;
get_apple5<=0;
end

current  direction<=next  direction; //  更新方向
```

```
if(is_pause == 1)
    count <= count;
else if (count < 5*1000000*(8 - speed)) //  控制速度
    count <= count+1;
else
begin
    count <= 0;
    //  头前进
    case (next_direction)
    UP:
        begin
        snake_x[0]<=snake_x[0];
        snake_y[0]<=snake_y[0]-1;
        end
    RIGHT:
        begin
        snake_x[0]<=snake_x[0]+1;
        snake_y[0]<=snake_y[0];
        end
    DOWN:
        begin
        snake_x[0]<=snake_x[0];
        snake_y[0]<=snake_y[0]+1;
        end
    LEFT:
        begin
        snake_x[0]<=snake_x[0]-1;
        snake_y[0]<=snake_y[0];
        end
    default:
        begin
        snake_x[0]<=snake_x[0]+1;
        snake_y[0]<=snake_y[0];
        end
    endcase
    //  身体前进
    snake_x[1]<=snake_x[0];
    snake_y[1]<=snake_y[0];

    snake_x[2]<=snake_x[1];
    snake_y[2]<=snake_y[1];

    snake_x[3]<=snake_x[2];
    snake_y[3]<=snake_y[2];
```

```
        snake_x[4]<=snake_x[3];snake_y[4]<=snake_y[3];snake_x[5]<=snake_x[4];
        snake_y[5]<=snake_y[4];snake_x[6]<=snake_x[5];snake_y[6]<=snake_y[5];
        snake_x[7]<=snake_x[6];snake_y[7]<=snake_y[6];snake_x[8]<=snake_x[7];
        snake_y[8]<=snake_y[7];snake_x[9]<=snake_x[8];snake_y[9]<=snake_y[8];
        snake_x[10]<=snake_x[9];snake_y[10]<=snake_y[9];snake_x[11]<=snake_x[10];
        snake_y[11]<=snake_y[10];snake_x[12]<=snake_x[11];snake_y[12]<=snake_y[11];
        snake_x[13]<=snake_x[12];snake_y[13]<=snake_y[12];snake_x[14]<=snake_x[13];
        snake_y[14]<=snake_y[13];snake_x[15]<=snake_x[14];snake_y[15]<=snake_y[14];
        snake_x[16]<=snake_x[15];snake_y[16]<=snake_y[15];snake_x[17]<=snake_x[16];
        snake_y[17]<=snake_y[16];snake_x[18]<=snake_x[17];snake_y[18]<=snake_y[17];
        snake_x[19]<=snake_x[18];snake_y[19]<=snake_y[18];snake_x[20]<=snake_x[19];
        snake_y[20]<=snake_y[19];snake_x[21]<=snake_x[20];snake_y[21]<=snake_y[20];
        snake_x[22]<=snake_x[21];snake_y[22]<=snake_y[21];snake_x[23]<=snake_x[22];
        snake_y[23]<=snake_y[22];snake_x[24]<=snake_x[23];snake_y[24]<=snake_y[23];
        snake_x[25]<=snake_x[24];snake_y[25]<=snake_y[24];snake_x[26]<=snake_x[25];
        snake_y[26]<=snake_y[25];snake_x[27]<=snake_x[26]snake_y[27]<=snake_y[26];
        snake_x[28]<=snake_x[27];snake_y[28]<=snake_y[27];snake_x[29]<=snake_x[28];
        snake_y[29]<=snake_y[28];snake_x[30]<=snake_x[29];snake_y[30]<=snake_y[29];
        snake_x[31]<=snake_x[30];snake_y[31]<=snake_y[30];
                end
            end
        end
endmodule
```

4、 apple 模块

```
`timescale 1ns / 1ps
module apple(
        input clock,
        input get_apple,
        input get_apple2,
        input get_apple3,
        input get_apple4,
        input get_apple5,
        input [1:0] game_status,

        output reg [5:0] apple_x,
        output reg [5:0] apple_y,
        output reg [5:0] apple2_x,
        output reg [5:0] apple2_y,
        output reg [5:0] apple3_x,
        output reg [5:0] apple3_y,
```

```
output reg [5:0] apple4_x,
output reg [5:0] apple4_y,
output reg [5:0] apple5_x,
output reg [5:0] apple5_y
);

localparam LAUNCHING=2'b00;
localparam PLAYING=2'b01;
localparam DIE_FLASHING=2'b10;
localparam INITIALIZING=2'b11;
    // 用于随机（伪）生成苹果坐标
    reg [11:0] random_for_x;
    reg [11:0] random_for_y;

    initial
    begin
        random_for_x<=521;
        random_for_y<=133;
    end

    always @(posedge clock)
    begin
        random_for_x<=random_for_x+997;
        random_for_y<=random_for_x+793;

        if (game_status==INITIALIZING)
            begin
            //苹果的初始位置，四个红苹果围一个金苹果
            apple_x<=20;
            apple_y<=9;
            apple2_x<=20;
            apple2_y<=17;
            apple3_x<=25;
            apple3_y<=13;
            apple4_x<=30;
            apple4_y<=9;
            apple5_x<=30;
            apple5_y<=17;
            random_for_x<=521;
            random_for_y<=133;
            end
        else
        if (game_status==PLAYING && get_apple==1)
```

```
    begin
                    // 防止苹果 x 和 y 坐标超范围
                    apple_x<=(random_for_x[5:0]+1>46?(random_for_x[5:0]+1-
20):(random_for_x[5:0]+1));
                    apple_y<=(random_for_y[4:0]+1>25?(random_for_y[4:0]+1-
10):(random_for_y[4:0]+1));
                end
                else
                if(game_status==PLAYING && get_apple2==1)
                begin
                    apple2_x<=(random_for_x[5:0]+1>46?(random_for_x[5:0]+1-
20):(random_for_x[5:0]+1));
                    apple2_y<=(random_for_y[4:0]+1>25?(random_for_y[4:0]+1-
10):(random_for_y[4:0]+1));
                end
                else
                if(game_status==PLAYING && get_apple3==1)
                begin
                    apple3_x<=(random_for_x[5:0]+1>46?(random_for_x[5:0]+1-
20):(random_for_x[5:0]+1));
                    apple3_y<=(random_for_y[4:0]+1>25?(random_for_y[4:0]+1-
10):(random_for_y[4:0]+1));
                end
                else
                if(game_status==PLAYING && get_apple4==1)
                begin
                    apple4_x<=(random_for_x[5:0]+1>46?(random_for_x[5:0]+1-
20):(random_for_x[5:0]+1));
                    apple4_y<=(random_for_y[4:0]+1>25?(random_for_y[4:0]+1-
10):(random_for_y[4:0]+1));
                end
                else
                if(game_status==PLAYING && get_apple5==1)
                begin
                    apple5_x<=(random_for_x[5:0]+1>46?(random_for_x[5:0]+1-
20):(random_for_x[5:0]+1));
                    apple5_y<=(random_for_y[4:0]+1>25?(random_for_y[4:0]+1-
10):(random_for_y[4:0]+1));
                end
                else
                begin
                    apple_x<=apple_x;
                    apple_y<=apple_y;
```

```
                    apple2_x<=apple2_x;
                    apple2_y<=apple2_y;
                    apple3_x<=apple3_x;
                    apple3_y<=apple3_y;
                    apple4_x<=apple4_x;
                    apple4_y<=apple4_y;
                    apple5_x<=apple5_x;
                    apple5_y<=apple5_y;
                end
            end
        endmodule
```

5、 score 模块

```
    `timescale 1ns / 1ps
    module score(
        input clock,
        input reset,
        input get_apple,
        input get_apple2,
        input get_apple3,
        input get_apple4,
        input get_apple5,
        input [1:0] game_status,

        output [7:0] an,
        output [7:0] seg
        );

        localparam LAUNCHING=2'b00;
        localparam PLAYING=2'b01;
        localparam DIE_FLASHING=2'b10;
        localparam INITIALIZING=2'b11;
        wire real_enable;
        wire real_reset;

        reg [31:0]Myscore;

        always @(posedge clock)
        begin
            if((reset==1) | ((game_status==INITIALIZING) == 1))
                Myscore = 0;
            else if((get_apple==1) && (game_status==PLAYING) == 1)
```

```
                Myscore = Myscore + 1;
            else if((get_apple2==1) && (game_status==PLAYING)==1)
                Myscore = Myscore + 1;
            else if((get_apple3==1) && (game_status==PLAYING)==1)
                Myscore = Myscore + 3;
            else if((get_apple4==1) && (game_status==PLAYING) == 1)
                Myscore = Myscore + 1;
            else if((get_apple5==1) && (game_status==PLAYING)==1)
                Myscore = Myscore + 1;
        end

        reg [31:0]total_time;
        reg [127:0]total_time_10ns;
        always @(posedge clock)
        begin
            if(game_status == PLAYING)
            begin
                total_time_10ns <= total_time_10ns + 1;
                if(total_time_10ns % 100000000 == 1)
                    total_time <= total_time + 1;
            end
            else if(game_status == INITIALIZING)
            begin
                total_time_10ns <= 0;
                total_time <= 0;
            end
        end

        seg (
            .reset(reset),
            .clock(clock),
            .an(an),
            .seg(seg),
            .score(Myscore),
            .total_time(total_time)
            );
    endmodule
```

6、 get_direction 模块

```
`timescale 1ns / 1ps
module get_direction(
    input clock,
    input up,right,down,left,
    input kup,kdown,kleft,kright,
    input [1:0] current_direction,
    output reg [1:0] next_direction
    );

    localparam UP=2'b00;
    localparam RIGHT=2'b01;
    localparam DOWN=2'b10;
    localparam LEFT=2'b11;

    always @(posedge clock)
    begin

        case (current_direction)
        UP:
            begin
            if (left==1 || kleft==1) next_direction<=LEFT;
            else if (right==1 || kright==1) next_direction<=RIGHT;
            else next_direction<=current_direction;
            end
        RIGHT:
            begin
            if (up==1||kup==1) next_direction<=UP;
            else if (down==1||kdown==1) next_direction<=DOWN;
            else next_direction<=current_direction;
            end
        DOWN:
            begin
            if (left==1||kleft==1) next_direction<=LEFT;
            else if (right==1||kright==1) next_direction<=RIGHT;
            else next_direction<=current_direction;
            end
        LEFT:
            begin
            if (up==1||kup==1) next_direction<=UP;
            else if (down==1||kdown==1) next_direction<=DOWN;
            else next_direction<=current_direction;
            end
        default: next_direction<=current_direction;
        endcase
    end
endmodule
```

7、 display 模块

```
module display(
    input clock, // 148.5MHZ，用于输出 1920x1080@60Hz 的 VGA 信号
    input [5:0] apple_x,
    input [5:0] apple_y,
    input [5:0] apple2_x,
    input [5:0] apple2_y,
    input [5:0] apple3_x,
    input [5:0] apple3_y,
    input [5:0] apple4_x,
    input [5:0] apple4_y,
    input [5:0] apple5_x,
    input [5:0] apple5_y,
    input [5:0] wall_x,
    input [5:0] wall_y,
    input [5:0] wall2_x,
    input [5:0] wall2_y,
    input [5:0] wall3_x,
    input [5:0] wall3_y,
    input [5:0] wall4_x,
    input [5:0] wall4_y,
    input [5:0] wall5_x,
    input [5:0] wall5_y,
    input [5:0] wall6_x,
    input [5:0] wall6_y,
    input [5:0] wall7_x,
    input [5:0] wall7_y,
    input [5:0] wall8_x,
    input [5:0] wall8_y,
    input [32*6-1:0] snake_x_temp,
    input [32*6-1:0] snake_y_temp,
    input [31:0] snake_piece_is_display,
    input [1:0] game_status,

    input [2:0] snake_color, //蛇的颜色(低位至高位分别为 R、G、B)

    output h_sync,v_sync,
    output reg [11:0] vga
    );

    wire VGA_CLK[1:0];
    my_clk_wiz_1 (clock,VGA_CLK[1]);
    localparam PAUSED=2'b00;
```

```
localparam PLAYING=2'b01;
localparam DIE_FLASHING=2'b10;
localparam INITIALIZING=2'b11;

localparam UP=2'b00;
localparam RIGHT=2'b01;
localparam DOWN=2'b10;
localparam LEFT=2'b11;

localparam h_active_pixels=1920;
localparam v_active_pixels=1080;

wire [11:0] x_counter;
wire [10:0] y_counter;
wire in_display_area;

// snake_x[0]：头的横坐标   snake_y[0]:头的纵坐标
reg [5:0] snake_x [31:0];
reg [5:0] snake_y [31:0];

// 当前 x，y 坐标，0<=x<=47,0<=y<=26
reg [5:0] current_x;
reg [5:0] current_y;

vga_sync_generator(
      .clock(VGA_CLK[1]),
      .h_sync(h_sync),
      .v_sync(v_sync),
      .x_counter(x_counter),
      .y_counter(y_counter),
      .in_display_area(in_display_area)
   );

integer i;
always @(snake_x_temp,snake_y_temp)
begin
for (i=0;i<32;i=i+1)
      begin
            // 片选
            snake_x[i]<=snake_x_temp[6*i+:6];
            snake_y[i]<=snake_y_temp[6*i+:6];
      end
```

```
end

always @(x_counter or y_counter)
begin
      current_x<=x_counter/21;
      current_y<=y_counter/28;
end

reg [11:0]RGB_snake_head;
reg [11:0]RGB_snake_body;

integer j;
always @(*)
begin
for(j=0;j<12;j=j+1)
    begin
    RGB_snake_head[j] = snake_color[j/4];
    RGB_snake_body[j] = snake_color[j/4];
    if(j == 2 || j == 6 || j == 10) RGB_snake_head[j] = 1;
    if(j == 3 || j == 7 || j == 11) RGB_snake_body[j] = 0;
    end
end

integer k;
reg flag;
always @(posedge VGA_CLK[1])
begin
            begin
                  flag <= 0;
                  if (in_display_area==0) vga<=0;
                  else if ((game_status==DIE_FLASHING) &&
                  (((current_x==8 || current_x==12 || current_x==14
                  || current_x==16 ||current_x==18 ||current_x==20
                  ||current_x==22||current_x==27||current_x==29||
                  current_x==35||current_x==39)
                  && (current_y >=11 &&current_y<=15))||
                  ((current_x==23 || current_x==24||current_x==36||current_x==37)
                  &&(current_y==11||current_y==13||current_y==15))
                  ||((current_x==31||current_x==33)
                  &&(current_y>=11 &&current_y<=14))
                  ||((current_y==11)&&(current_x==9||current_x==10
                  ||current_x==13||current_x==17||current_x==19
                  ||current  x==28||current  x==40||current  x==41))
```

```
                ||(current_x==41&&current_y==12)
                ||((current_x==10||current_x==13||current_x==40
                ||current_x==41)&&(current_y==13))
                ||(current_y==14&&(current_x==10||current_x==40))
                ||((current_x==9||current_x==10||current_x==28||
                current_x==32||current_x==41)
                &&current_y==15))) vga<=12'b0000_0000_1111;
else if ((game_status==DIE_FLASHING)
&& (current_x>=7 && current_x <= 42 && current_y>=10
&& current_y<=16)) vga<=12'b1111_1111_1111;
else if ((game_status==DIE_FLASHING)
&& (current_x>=6 && current_x <= 43 && current_y>=9
&& current_y<=17)) vga<=12'b0111_0111_0111;
else if (current_x==0 || current_x==48 || current_y==0
|| current_y == 26) vga<=12'b1111_1111_1111; //边框
else if (current_x==wall_x && current_y==wall_y)
vga<=12'b1111_1111_1111;
else if (current_x==wall2_x && current_y==wall2_y)
vga<=12'b1111_1111_1111;
else if (current_x==wall3_x && current_y==wall3_y)
vga<=12'b1111_1111_1111;
else if (current_x==wall4_x && current_y==wall4_y)
  vga<=12'b1111_1111_1111;
else if (current_x==wall5_x && current_y==wall5_y)
  vga<=12'b1111_1111_1111;
else if (current_x==wall6_x && current_y==wall6_y)
  vga<=12'b1111_1111_1111;
else if (current_x==wall7_x && current_y==wall7_y)
  vga<=12'b1111_1111_1111;
else if (current_x==wall8_x && current_y==wall8_y)
  vga<=12'b1111_1111_1111;
else if ((current_x == apple_x && current_y == apple_y) ||
(current_x == apple2_x && current_y == apple2_y))
vga<=12'b0000_0000_1111; //苹果(红色)
else if (current_x == apple3_x && current_y == apple3_y)
  vga<=12'b0000_1111_1111;
else if ((current_x == apple4_x && current_y == apple4_y) ||
(current_x == apple5_x && current_y == apple5_y))
vga<=12'b0000_0000_1111; //苹果(红色)
else if (current_x == snake_x[0] && current_y == snake_y[0]
&& snake_piece_is_display[0]==1) vga <= RGB_snake_head; //蛇头
else if(game_status==INITIALIZING && ((current_x==16
&& current  y==3) || (current  x==16 && current  y==4) ||
```

```
                    (current_x==16 && current_y==5) || (current_x==16 && current_y==7)
                    || ((current_x==17||current_x==34||current_x==35)
                    && (current_y==3||current_y==5||current_y==7))
                    || (current_x==18 && current_y==3) ||
                    (current_x==18 && current_y==5) || (current_x==18 && current_y==6)
                    ||(current_x==18 && current_y==7)
                    || ((current_x==20||current_x==22||current_x==24
                    ||current_x==26||current_x==28||current_x==33)
                    && (current_y==3||current_y==4||current_y==5||
                    current_y==6||current_y==7))|| ((current_x==21||
                    current_x==25||current_x==31) && current_y==3)
                    || (current_x==25 && current_y==5) || (current_x==30&&current_y==4)
                    || (current_x==30&&current_y==6) ||
                    (current_x==31 && current_y==7) ||
                    (current_x==29&&current_y==5))) vga<=12'b1111_0000_1111;
                    else begin
                        for(k=0;k<32;k=k+1)
                            begin
                            if(current_x == snake_x[k] && current_y == snake_y[k]
                            && snake_piece_is_display[k] == 1) flag = 1;
                            end
                         if(flag == 1) vga <= RGB_snake_body;
                         else vga <= 0;
                     end
                end
    end
endmodule
```

8、 keyboard 模块

```
module keyboard (clk,rst,key_clk,key_data,kup,kdown,kleft,kright);
input clk;                          //系统时钟
input rst;                         //系统复位，低有效
input key_clk;                     //PS2 键盘时钟
input key_data;                        //PS2 键盘数据输入
output reg kup,kdown,kleft,kright;

reg [3:0] now_bit = 0;              //记录已获取的 bit 数
reg [7:0] store = 0;                //存储键盘数据输入
reg [7:0] key_info = 0;             //存储键盘数据信息
reg break = 0;                         //断码标志
```

```
always @ (posedge clk or negedge rst)
begin
    if(!rst)
    begin
        key_clk_new <= 1'b1;
        key_clk_old <= 1'b1;
        key_data_new <= 1'b1;
        key_data_old <= 1'b1;
    end
    else
    begin
        //交替存储实现延时锁存
        key_clk_new <= key_clk;
        key_clk_old <= key_clk_new;
        key_data_new <= key_data;
        key_data_old <= key_data_new;
    end
end

wire key_clk_neg = key_clk_old & (~key_clk_new);

//PS2 键盘时钟信号下降沿读取数据
always @ (posedge clk or negedge rst)
begin
    if(!rst)
    begin
        now_bit <= 4'd0;
        store <= 8'd0;
    end
    else
    begin
        if(key_clk_neg)
        begin
            if(now_bit >= 4'd10) now_bit <= 4'd0;
            else now_bit <= now_bit + 1'b1;
            case (now_bit)
                0: ;                            //起始位
                // bit 数据位
                1: store[0] <= key_data_old;
                2: store[1] <= key_data_old;
                3: store[2] <= key_data_old;
                4: store[3] <= key_data_old;
                5: store[4] <= key_data_old;
```

```
                    6: store[5] <= key_data_old;
                    7: store[6] <= key_data_old;
                    8: store[7] <= key_data_old;
                    9: ;                              //校验位
                    10:;                              //结束位
                    default: ;
                endcase
            end
        end
end

//判断断码
always @ (posedge clk or negedge rst)
begin
        if(!rst)
        begin
            break <= 0;             //重置标记
            key_info <= 0;
        end
        else
        begin
            if(now_bit == 4'd10 && key_clk_neg)
            begin
                if(store == 8'hf0) break <= 1;              //断码
                else if(!break) key_info <= store;       //不为断码
                else                                              //该段数据是断码，舍弃
                begin
                    break <= 0;             //重置标记
                    key_info <= 0;
                end
            end
        end
end


//将从键盘得到的数据转换为 ASCII 码
always @ (key_info)
begin
        kup=0;
        kdown=0;
        kleft=0;
        kright=0;
        case (key_info)
```

```
            8'h75: kup <= 1;                //上箭头
            8'h72: kdown <= 1;              //下箭头
            8'h6b: kleft <= 1;              //左箭头
            8'h74: kright <= 1;             //右箭头
            8'h1d: kup <= 1;                // w
            8'h1b: kdown <= 1;              // s
            8'h1c: kleft <= 1;              // a
            8'h23: kright <= 1;             // d
        endcase
end
endmodule
```

9、 audio 模块

```
`timescale 1ns / 1ps
module get_wave(
        input clk_in,
        input [31:0]n_2,
        input rst_n,
        output reg clk_out //分频后的时钟
);
reg [31:0] counter;
wire [31:0] n;
assign n = n_2 / 2;

initial
begin
        clk_out <= 0;
        counter <= 32'b0;
end
always@(posedge clk_in or negedge rst_n)
begin
        if(rst_n == 0)
        begin
                clk_out = 0;
                counter = 0;
        end
        else if(n != 32'b0)
            begin
                    if(counter < n)
                            counter <= counter + 1;
                    else
                    begin
```

```
                clk_out <= ~clk_out;
                counter <= 32'b0;
            end
        end
end
endmodule
module Audio(
        input clk,
        input hit_wall,
        input hit_itself,
        input get_apple,
        input get_apple2,
        input get_apple3,
        input get_apple4,
        input get_apple5,
        input up,right,down,left,
        output AUD_PWM,
        output AUD_SD
        );

reg [31:0] n_2;//分频系数
reg [7:0]pitch;
reg counter_ena;       // 用于开始计时
reg [127:0]counter; // 用于计时响多久

assign AUD_SD = counter_ena;
        //控制音高，音长
        always @(posedge clk)
        begin
        if(counter_ena)
        begin
            if(counter == 50000000)
            begin
                counter_ena = 0;
                counter = 0;
            end
            else
                counter = counter + 1;
        end
        else
        begin
            if(hit_wall | hit_itself)
            begin
                counter_ena = 1;
```

```
                counter = 0;
                pitch = 7;
            end
            else if(get_apple || get_apple2 || get_apple4 || get_apple5)
            begin
                counter_ena = 1;
                counter = 0;
                pitch = 5;
            end
            else if(get_apple3)
            begin
                counter_ena = 1;
                counter = 0;
                pitch = 6;
            end
            else if(up | right | left | down)
            begin
                counter_ena = 1;
                counter = 0;
                pitch = 1;
            end
            case(pitch)
                8'd0:n_2<=32'd0;
                8'd1:n_2<=32'd191095;
                8'd2:n_2<=32'd170270;
                8'd3:n_2<=32'd151676;
                8'd4:n_2<=32'd143163;
                8'd5:n_2<=32'd127551;
                8'd6:n_2<=32'd113636;
                8'd7:n_2<=32'd101235;
                default: n_2<=32'd191095;
            endcase
        end
        end
get_wave Audio(clk, n_2, counter_ena, AUD_PWM);
endmodule
```