基于 Xilinx Nexys4 DDR 的贪吃蛇小游戏设计

2051981 李想

目录

1 设计动机和思路 **2**

2 结果展示 **2**

2.1 实机演示 2

3 各模块说明 **5**

3.1 top：顶层模块 5

3.2 snake：控制蛇身 5

3.3 apple：控制苹果生成 5

3.4 fsm：控制游戏状态 6

3.5 Get_direction：获取方向 6

3.6 score：用时、得分的记录和输出 6

3.7 display：VGA 显示 7

3.8 Audio：音效输出 7

4 心得体会 **8**

4.1 本课程收获 8

4.2 对本课程建议 8

4.3 我的认识和体会 8

5 模块代码 **10**

1　设计动机和思路

动机　想做一个贪吃蛇的动机主要是这是能够利用开发板和 VGA 显示器的相对简单的设计，比较容易在任务众多的期末周中实现。（虽然任务开始之后发现并不容易）

思路　将整个屏幕（1024*768）分割为 48*27 的游戏区域，设置一个二维数组储存游戏区域（蛇、苹果等），然后吃苹果加分、撞到墙游戏结束。

- 游戏的暂停、开始、重启：通过一个 fsm 状态机来控制。
- 方向控制：通过板载的四个按键控制（本来想用三轴加速器控制，但最后没做好）
- 蛇和苹果：分别用两个寄存器存储位置
- VGA 屏幕显示和音效输出：分别写两个模块单独控制

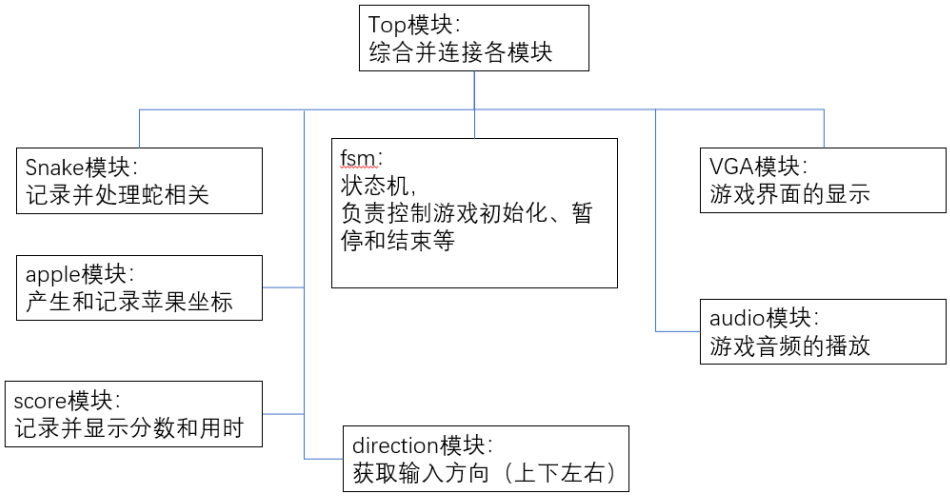

图 1: 具体连接设计

2　结果展示

2.1　实机演示

以下为图片展示部分：

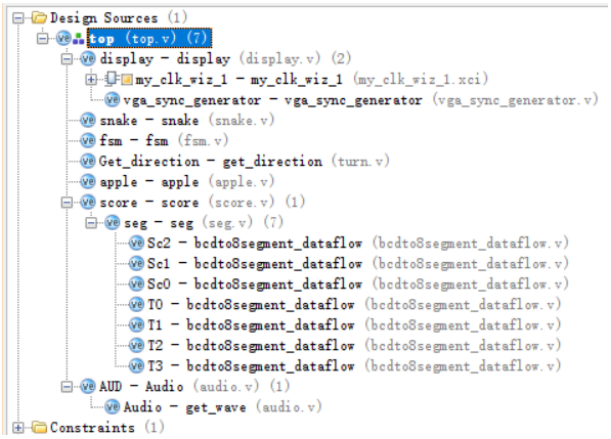

图 2: 实际连接

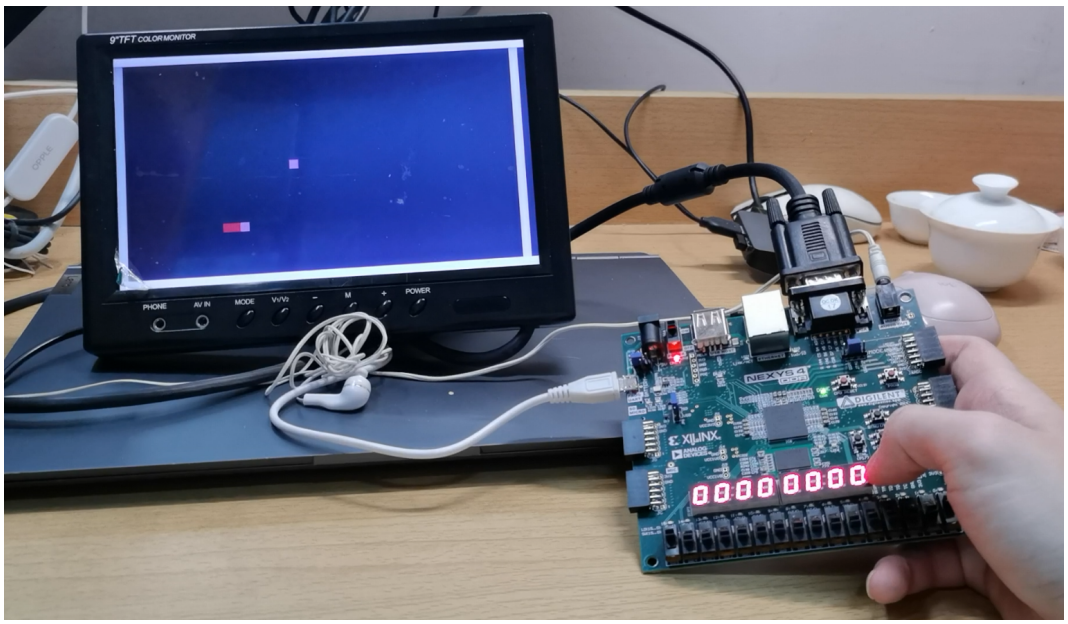

图 3: 开始游戏

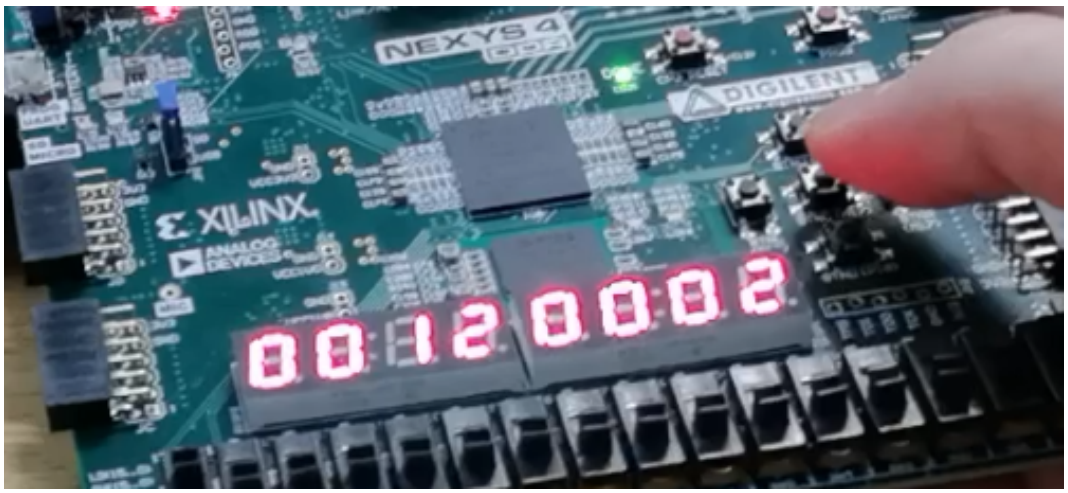

图 4: 左：当前用时；右：当前得分

图 5: 触墙判定死亡、闪烁 2s 后返回至新一局游戏

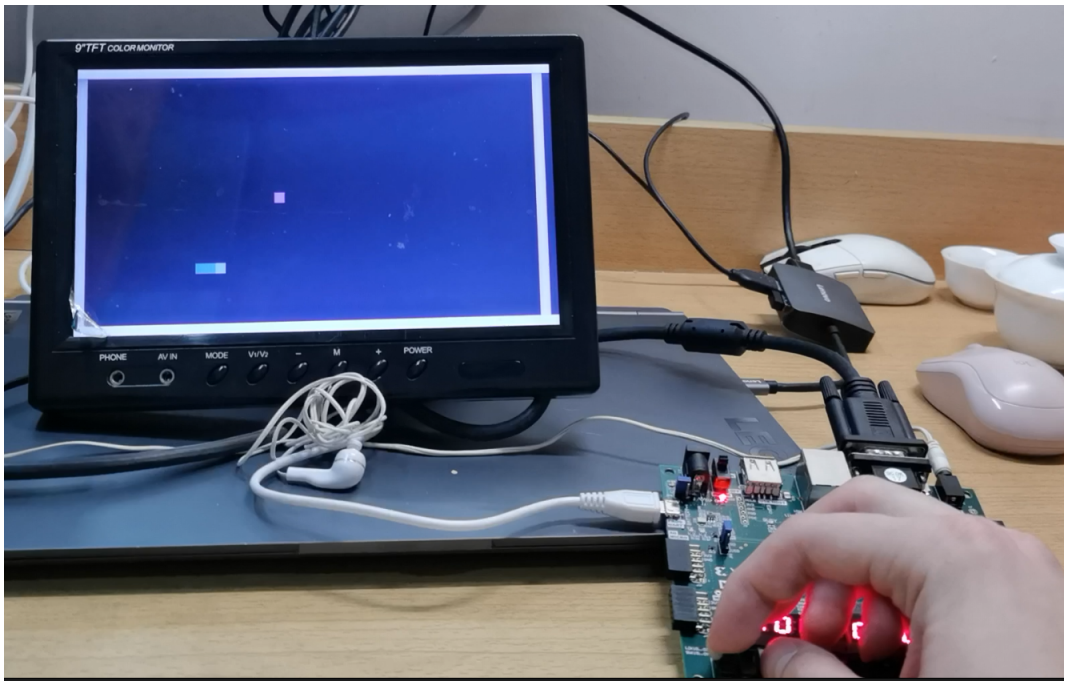

图 6: 可以更换蛇的颜色（共八种）

注：**图片形式未能展示的**内容包括：

1 具体游戏过程（包括按上下左右拐弯，吃苹果后蛇身变长等）

2 游戏音效（每个按键均有不同音效，吃苹果、死亡也有相应音效）

3 可以调节游戏速度，共七档

4 可以暂停游戏

5 可以按键直接重新开始一局新游戏

3 各模块说明

3.1 top：顶层模块

代码部分： 点击跳转

3.2 snake：控制蛇身

输入

- 时钟
- 暂停、减速、转向命令
- 苹果位置
- 游戏状态

输出：

- 当前方向
- 蛇身数据（坐标、体长）
- 是否吃苹果、撞墙、撞自身

代码部分： 点击跳转

3.3 apple：控制苹果生成

输入

- 时钟
- 是否获取新苹果坐标
- 游戏状态

输出：

- 苹果坐标

代码部分： 点击跳转

3.4 fsm：控制游戏状态

输入

- 时钟
- 是否重置、撞墙、撞自己
- 上下左右（用于按任意键开始）

输出：

- 游戏状态

代码部分： 点击跳转

3.5 Get_direction：获取方向

输入

- 时钟
- 上下左右，当前方向

输出：

- 下一个（经核验过，合理的）方向

代码部分： 点击跳转

3.6 score：用时、得分的记录和输出

输入

- 时钟
- 是否重置、吃苹果
- 游戏状态

输出：

- 七段数码管相关输出（左四：时间；右三：得分）

代码部分： 点击跳转

3.7 display：VGA 显示

输入

- 时钟 (经 IP 核调整为 148.5mhz)
- 苹果坐标
- 蛇坐标、体长
- 游戏状态

输出：

- VGA 相关信息（颜色；行、场扫描信号）

代码部分： 点击跳转

3.8 Audio：音效输出

输入

- 时钟
- 撞墙、吃苹果、方向键等信号

输出：

- 用于音频输出的波形和使能信号

代码部分： 点击跳转

4 心得体会

4.1 本课程收获

上完本课程之后，我掌握了从门级到行为级的硬件设计能力，确实学到了很多，对集成电路有了更深刻的认识！最重要的是掌握了利用 verilog 的硬件设计流程，具体如下：

首先是理解需求：不要被数字电路的外表迷惑了，以为这是个写代码的工作。它本质上还是电路设计，只不过用行为级的方法描述了出来。所以理解电路的需求就非常重要。首先要清楚电路从哪来，到哪去，用它想做什么，实现了什么功能，这样才能更好的设计。

其次是找寻主干：要找到最重要或者说最核心的模块。就是顶层模块和关键的逻辑模块，输入输出模块这两大类。

最后是对时序电路的理解。电路是全并行的，要想象面前的代码就是一堆分立元件，同时工作。搞清楚之后，再到如何写代码，理逻辑。

4.2 对本课程建议

可以加快有关模块手册指导资料的更新，还有在 mips246 网站上更新大作业相关器件的相关资料，以便于学习。

4.3 我的认识和体会

上完数字逻辑课程后，我对数字芯片设计有了更深刻的认识。

对于国内外数字芯片设计行业现状的认识和看法 数字芯片设计行业最关键的就是三个部分：

1、CPU：cpu 这种东西本身就比较复杂，它的研发不可能一蹴而就，需要时间和经验的积累。普通企业压根不可能会自主研发这类产品。因为企业是以盈利为目的的，不可能投资个几亿甚至更多，然后等个十年再来看回报的。因此，大部分企业宁可花钱买授权，省心省事（自己做的万一出来的产品有 bug 怎么办）。所以你能看到在研究 CPU 这玩意的大都是研究所，高校，国企等背景的单位。这部分国家其实也一直在加大投入研发，只是需要时间。

2、EDA 工具：对于数字芯片设计来说，主要的 EDA 工具均来自于美国，分别是 cadence 和 synopsys。自贸易战开打，美国就开始限制各大 EDA 工具的授权使用和服务。虽然国内大公司被限制 eda 工具的使用和服务，会一定程度上影响他们的产品，比如开发一款新产品，可能需要更多的时间，可能用别的 EDA 工具会有很多问题等，但是如果美国一直严格限制甚至全面禁止 EDA 工具的使用，其实反而会极大加速国内自主研发的 EDA 工具的发展。比如国内华大九天的产品，其实早就能用，只不过没他们好用。这个问题就跟上面造 cpu 面临一样的问题。

3、代工厂：目前国际上主要的 foundary 有台积电 TSMC，UMC，Global Foundary，三星，SMIC，华宏等。当然 inter 也有，不过他们是给自己的产品做加工。其中 TSMC 当属全球第一，Global Foundary 排行老三。大陆最好的算是 SMIC 了，国家也一直在大力扶持本土 foundary 的发展。典型的案例就是国家投入巨资让 smic 买光刻机，国家让国内各大名企参与 SMIC 14nm 工艺的研发和量产。预计 14nm 今年会正式量产。看到这里，应该知道现在的 TSMC 已经在研发 3nm 了。这就是差距所在。要知道华为大部分的产品其实都是在 TSMC 加工量产的，如果哪天 TSMC 不给华为产能或者不给他们加工，那么短期内其实会面临着严重的挑战。因此，扶持大陆 smic 的举动是必然的。

虽然国内芯片发展现状还与国际水平存在一定的差距，但是最近十年国内 IC 发展非常迅猛，国家也出台了一系列的政策和措施来扶持国内企业和补齐行业人才的缺口。相信在不远的将来，以中国人的头脑，一定可以实现赶超!

并祝国家早日统一，让台积电公司为中华民族伟大复兴的道路添砖加瓦!!!

5 模块代码

top 模块:

```
`timescale 1ns / 1ps

module top(
    input CLK100MHZ, // 100MHz时钟
    input reset, // 重新开始
    input up,right,down,left, // 方向键
    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)

    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
//  // FOR DEBUG
//  output [11:0] led
//  output display_hit_wall,
//  output display_hit_itself,
//  output display_get_apple,
//  output display_is_launching
    );

    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 == 0) ? 1'b1 : 1'b0;

  // 用于模块间传递二维数组 [5:0] snake_x/y [31:0]
  // 参考自 https://stackoverflow.com/questions/16369698/how-to-pass-array-
    structure-between-two-verilog-modules
    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

```

```
38  wire [31:0] snake_piece_is_display;  // 控制体长
39
40  wire [5:0] apple_x; // 苹果坐标
41  wire [5:0] apple_y; // 苹果坐标
42
43  // 游戏状态 00: PAUSED  01: PLAYING   10:DIE_FLASHING   11: INITIALIZING
44  wire [1:0] game_status;
45
46  // 方向 00: UP   01: Right   10: DOWN   11: LEFT
47  wire [1:0] current_direction;
48  wire [1:0] next_direction;
49
50  wire get_apple; // 吃到苹果
51
52  wire hit_wall; // 撞墙否
53  wire hit_itself; // 撞己否
54
55
56
57
58    display (
59        .clock(CLK100MHZ), // 在display内部做时钟转换
60        .h_sync(h_sync),
61        .v_sync(v_sync),
62        .vga(vga),
63        .game_status(game_status),
64        .apple_x(apple_x),
65        .apple_y(apple_y),
66        .snake_x_temp(snake_x_temp),
67        .snake_y_temp(snake_y_temp),
68        .snake_piece_is_display(snake_piece_is_display),
69        .snake_color(snake_color)
70        );
71
72    snake (
73      .clock(CLK100MHZ),
74      .speed(speed),
75      .snake_x_temp(snake_x_temp),
76      .snake_y_temp(snake_y_temp),
77      .apple_x(apple_x),
78      .apple_y(apple_y),
```

```
79          .snake_piece_is_display(snake_piece_is_display),
80          .get_apple(get_apple),
81          .game_status(game_status),
82            .current_direction(current_direction),
83          .next_direction(next_direction),
84          .hit_wall(hit_wall),
85          .hit_itself(hit_itself)
86          );
87
88      fsm (
89            .reset(reset),
90            .clock(CLK100MHZ),
91            .game_status(game_status),
92            .hit_wall(hit_wall),
93            .hit_itself(hit_itself),
94          .up(up),
95            .right(right),
96            .down(down),
97            .left(left),
98            .pause(is_pause)
99            );
100
101     get_direction Get_direction (
102       .clock(CLK100MHZ),
103       .up(up),
104       .right(right),
105       .down(down),
106       .left(left),
107       .current_direction(current_direction),
108       .next_direction(next_direction)
109       );
110
111     apple (
112           .clock(CLK100MHZ),
113           .apple_x(apple_x),
114           .apple_y(apple_y),
115           .get_apple(get_apple),
116           .game_status(game_status)
117           );
118
119     score (
```

```
        .reset(reset),
        .clock(CLK100MHZ),
        .get_apple(get_apple),
      .game_status(game_status),
        .an(an),
        .seg(seg)
          );


       Audio AUD (
            .clk(CLK100MHZ),
            .hit_wall(hit_wall),
            .get_apple(get_apple),
            .hit_itself(hit_itself),
            .up(up),
            .right(right),
            .down(down),
            .left(left),
            .AUD_PWM(AUD_PWM),
            .AUD_SD(AUD_SD)            );

endmodule
```

snake 模块:

```
`timescale 1ns / 1ps

module snake(
  // 变量说明见top模块
  input clock,
//  input pause,
  input [2:0]speed,
  input [1:0] next_direction,
  input [1:0] game_status,
  input [5:0] apple_x,
  input [5:0] apple_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
```

```
17    output reg get_apple,
18    output reg hit_wall,
19    output reg hit_itself
20      );
21
22      wire is_pause;
23      assign is_pause = (speed == 0) ? 1'b1 : 1'b0;
24
25      reg [25:0] count;
26      reg [25:0] count_two;
27      reg [31:0] snake_piece_is_display_origin; // 存储体长的旧值，用于死亡闪
        烁
28
29    localparam PAUSED=2'b00;
30      localparam PLAYING=2'b01;
31      localparam DIE_FLASHING=2'b10;
32      localparam INITIALIZING=2'b11;
33
34      localparam UP=2'b00;
35      localparam RIGHT=2'b01;
36      localparam DOWN=2'b10;
37      localparam LEFT=2'b11;
38
39    // snake_x[0]：头的横坐标  snake_y[0]:头的纵坐标
40      reg [5:0] snake_x [31:0];
41      reg [5:0] snake_y [31:0];
42
43
44    // 用于模块间传递二维数组 [5:0] snake_x/y [31:0]
45    // 参考自 https://stackoverflow.com/questions/16369698/how-to-pass-array-
        structure-between-two-verilog-modules
46      genvar i;
47      generate for (i=0;i<32;i=i+1)
48      begin
49        assign snake_x_temp[6*i+:6]=snake_x[i];
50        assign snake_y_temp[6*i+:6]=snake_y[i];
51      end endgenerate
52
53
54    initial
55      begin
```

```
56      count<=0;
57      count_two<=0;
58      end
59
60      always @(posedge clock)
61      begin
62        if (count_two>=40000000) count_two<=0;
63        else count_two<=count_two+1;
64
65        if (game_status==INITIALIZING)
66        begin
67      snake_piece_is_display<=32'b00000000_00000000_00000000_00000111;
68        snake_x[0]<=14;
69        snake_y[0]<=20;
70        snake_x[1]<=13;
71        snake_y[1]<=20;
72        snake_x[2]<=12;
73        snake_y[2]<=20;
74        current_direction<=RIGHT;
75        hit_wall<=0;
76        hit_itself<=0;
77        end
78
79        else if (game_status==DIE_FLASHING)
80        begin
81        // 闪烁
82        if (count_two==20000000) snake_piece_is_display<=0;
83        else if (count_two==0) snake_piece_is_display<=
    snake_piece_is_display_origin;
84        end
85
86        else if (game_status==PLAYING && (is_pause == 0))
87        begin
88          snake_piece_is_display_origin<=snake_piece_is_display; // 存储体长
    的旧值，用于死亡闪烁(若死亡)
89
90          if (snake_x[0]==0 || snake_x[0]==47 || snake_y[0]==0 || snake_y
    [0]==26) hit_wall<=1;
91          else hit_wall<=0; // 是否撞墙
92
93          if (
```

```
94              (snake_x[0]==snake_x[1] && snake_y[0]==snake_y[1] &&
    snake_piece_is_display[1]==1) ||
95          (snake_x[0]==snake_x[2] && snake_y[0]==snake_y[2] &&
    snake_piece_is_display[2]==1) ||
96          (snake_x[0]==snake_x[3] && snake_y[0]==snake_y[3] &&
    snake_piece_is_display[3]==1) ||
97          (snake_x[0]==snake_x[4] && snake_y[0]==snake_y[4] &&
    snake_piece_is_display[4]==1) ||
98          (snake_x[0]==snake_x[5] && snake_y[0]==snake_y[5] &&
    snake_piece_is_display[5]==1) ||
99          (snake_x[0]==snake_x[6] && snake_y[0]==snake_y[6] &&
    snake_piece_is_display[6]==1) ||
100         (snake_x[0]==snake_x[7] && snake_y[0]==snake_y[7] &&
    snake_piece_is_display[7]==1) ||
101         (snake_x[0]==snake_x[8] && snake_y[0]==snake_y[8] &&
    snake_piece_is_display[8]==1) ||
102         (snake_x[0]==snake_x[9] && snake_y[0]==snake_y[9] &&
    snake_piece_is_display[9]==1) ||
103         (snake_x[0]==snake_x[10] && snake_y[0]==snake_y[10] &&
    snake_piece_is_display[10]==1) ||
104         (snake_x[0]==snake_x[11] && snake_y[0]==snake_y[11] &&
    snake_piece_is_display[11]==1) ||
105         (snake_x[0]==snake_x[12] && snake_y[0]==snake_y[12] &&
    snake_piece_is_display[12]==1) ||
106         (snake_x[0]==snake_x[13] && snake_y[0]==snake_y[13] &&
    snake_piece_is_display[13]==1) ||
107         (snake_x[0]==snake_x[14] && snake_y[0]==snake_y[14] &&
    snake_piece_is_display[14]==1) ||
108         (snake_x[0]==snake_x[15] && snake_y[0]==snake_y[15] &&
    snake_piece_is_display[15]==1) ||
109         (snake_x[0]==snake_x[16] && snake_y[0]==snake_y[16] &&
    snake_piece_is_display[16]==1) ||
110         (snake_x[0]==snake_x[17] && snake_y[0]==snake_y[17] &&
    snake_piece_is_display[17]==1) ||
111         (snake_x[0]==snake_x[18] && snake_y[0]==snake_y[18] &&
    snake_piece_is_display[18]==1) ||
112         (snake_x[0]==snake_x[19] && snake_y[0]==snake_y[19] &&
    snake_piece_is_display[19]==1) ||
113         (snake_x[0]==snake_x[20] && snake_y[0]==snake_y[20] &&
    snake_piece_is_display[20]==1) ||
114         (snake_x[0]==snake_x[21] && snake_y[0]==snake_y[21] &&
```

```
    snake_piece_is_display[21]==1) ||
115         (snake_x[0]==snake_x[22] && snake_y[0]==snake_y[22] &&
    snake_piece_is_display[22]==1) ||
116         (snake_x[0]==snake_x[23] && snake_y[0]==snake_y[23] &&
    snake_piece_is_display[23]==1) ||
117         (snake_x[0]==snake_x[24] && snake_y[0]==snake_y[24] &&
    snake_piece_is_display[24]==1) ||
118         (snake_x[0]==snake_x[25] && snake_y[0]==snake_y[25] &&
    snake_piece_is_display[25]==1) ||
119         (snake_x[0]==snake_x[26] && snake_y[0]==snake_y[26] &&
    snake_piece_is_display[26]==1) ||
120         (snake_x[0]==snake_x[27] && snake_y[0]==snake_y[27] &&
    snake_piece_is_display[27]==1) ||
121         (snake_x[0]==snake_x[28] && snake_y[0]==snake_y[28] &&
    snake_piece_is_display[28]==1) ||
122         (snake_x[0]==snake_x[29] && snake_y[0]==snake_y[29] &&
    snake_piece_is_display[29]==1) ||
123         (snake_x[0]==snake_x[30] && snake_y[0]==snake_y[30] &&
    snake_piece_is_display[30]==1) ||
124         (snake_x[0]==snake_x[31] && snake_y[0]==snake_y[31] &&
    snake_piece_is_display[31]==1)
125         )
126         hit_itself<=1;
127         else hit_itself<=0; // 是否撞自己
128
129         if (snake_x[0]==apple_x && snake_y[0]==apple_y) get_apple<=1;
130         else get_apple<=0; // 是否吃到苹果
131
132         if (get_apple == 1 )
133         begin
134         snake_piece_is_display<=2*snake_piece_is_display+1; // 增加体长（把
    snake_piece_is_display最后一个0变成1）
135         get_apple<=0;
136         end
137
138         current_direction<=next_direction; // 更新方向
139
140
141         if(is_pause == 1)
142             count <= count;
143         else if (count < 5*1000000*(8 - speed)) // 控制速度
```

```
144             count <= count+1;
145         else
146         begin
147         count <= 0;
148 
149         // 头前进
150         case (next_direction)
151         UP:
152             begin
153             snake_x[0]<=snake_x[0];
154             snake_y[0]<=snake_y[0]-1;
155             end
156         RIGHT:
157             begin
158             snake_x[0]<=snake_x[0]+1;
159             snake_y[0]<=snake_y[0];
160             end
161         DOWN:
162             begin
163             snake_x[0]<=snake_x[0];
164             snake_y[0]<=snake_y[0]+1;
165             end
166         LEFT:
167             begin
168             snake_x[0]<=snake_x[0]-1;
169             snake_y[0]<=snake_y[0];
170             end
171         default:
172             begin
173             snake_x[0]<=snake_x[0]+1;
174             snake_y[0]<=snake_y[0];
175             end
176         endcase
177 
178         // 身体前进
179         snake_x[1]<=snake_x[0];
180         snake_y[1]<=snake_y[0];
181 
182         snake_x[2]<=snake_x[1];
183         snake_y[2]<=snake_y[1];
184 
```

```
185             snake_x[3]<=snake_x[2];
186             snake_y[3]<=snake_y[2];
187
188             snake_x[4]<=snake_x[3];
189             snake_y[4]<=snake_y[3];
190
191             snake_x[5]<=snake_x[4];
192             snake_y[5]<=snake_y[4];
193
194             snake_x[6]<=snake_x[5];
195             snake_y[6]<=snake_y[5];
196
197             snake_x[7]<=snake_x[6];
198             snake_y[7]<=snake_y[6];
199
200             snake_x[8]<=snake_x[7];
201             snake_y[8]<=snake_y[7];
202
203             snake_x[9]<=snake_x[8];
204             snake_y[9]<=snake_y[8];
205
206             snake_x[10]<=snake_x[9];
207             snake_y[10]<=snake_y[9];
208
209             snake_x[11]<=snake_x[10];
210             snake_y[11]<=snake_y[10];
211
212             snake_x[12]<=snake_x[11];
213             snake_y[12]<=snake_y[11];
214
215             snake_x[13]<=snake_x[12];
216             snake_y[13]<=snake_y[12];
217
218             snake_x[14]<=snake_x[13];
219             snake_y[14]<=snake_y[13];
220
221             snake_x[15]<=snake_x[14];
222             snake_y[15]<=snake_y[14];
223
224             snake_x[16]<=snake_x[15];
225             snake_y[16]<=snake_y[15];
```

```
226
227             snake_x[17]<=snake_x[16];
228             snake_y[17]<=snake_y[16];
229
230             snake_x[18]<=snake_x[17];
231             snake_y[18]<=snake_y[17];
232
233             snake_x[19]<=snake_x[18];
234             snake_y[19]<=snake_y[18];
235
236             snake_x[20]<=snake_x[19];
237             snake_y[20]<=snake_y[19];
238
239             snake_x[21]<=snake_x[20];
240             snake_y[21]<=snake_y[20];
241
242             snake_x[22]<=snake_x[21];
243             snake_y[22]<=snake_y[21];
244
245             snake_x[23]<=snake_x[22];
246             snake_y[23]<=snake_y[22];
247
248             snake_x[24]<=snake_x[23];
249             snake_y[24]<=snake_y[23];
250
251             snake_x[25]<=snake_x[24];
252             snake_y[25]<=snake_y[24];
253
254             snake_x[26]<=snake_x[25];
255             snake_y[26]<=snake_y[25];
256
257             snake_x[27]<=snake_x[26];
258             snake_y[27]<=snake_y[26];
259
260             snake_x[28]<=snake_x[27];
261             snake_y[28]<=snake_y[27];
262
263             snake_x[29]<=snake_x[28];
264             snake_y[29]<=snake_y[28];
265
266             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
```

fsm 模块:

```
`timescale 1ns / 1ps

module fsm(
  // 变量说明见top模块
    input reset,
    input clock,
    input hit_wall,
    input hit_itself,
    input up,right,down,left,
    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

    // 初始化状态和计数器
  initial
  begin
  game_status<=INITIALIZING;

  count_two<=0;
  end

    always @(posedge clock)
    begin
```

```
        //任何状态下，按下reset恢复到INITIALIZING
        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 && ( up==1 || right==1 || down==1
     || left==1))
      begin
      game_status <= PLAYING; // 按下任意按键时游戏开始
      end
      else if (game_status == PAUSED && ( up==1 || right==1 || down==1 ||
     left==1))
          begin
          game_status <= PLAYING; // 暂停状态下，按下任意按键时游戏继续
          end
      end
endmodule
```

Get_direction 模块:

```
`timescale 1ns / 1ps

module get_direction(
  // 变量说明见top模块
    input clock,
    input up,right,down,left,
    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) next_direction<=LEFT;
      else if (right==1) next_direction<=RIGHT;
      else next_direction<=current_direction;
      end
    RIGHT:
      begin
      if (up==1) next_direction<=UP;
      else if (down==1) next_direction<=DOWN;
      else next_direction<=current_direction;
      end
    DOWN:
      begin
      if (left==1) next_direction<=LEFT;
      else if (right==1) next_direction<=RIGHT;
      else next_direction<=current_direction;
      end
    LEFT:
      begin
      if (up==1) next_direction<=UP;
      else if (down==1) next_direction<=DOWN;
      else next_direction<=current_direction;
      end
    default: next_direction<=current_direction;
    endcase
    end
endmodule
```

apple 模块:

```
`timescale 1ns / 1ps

module apple(
  // 变量说明见top模块
    input clock,
    input get_apple,
    input [1:0] game_status,

    // 输出苹果坐标
    output reg [5:0] apple_x,
    output reg [5:0] apple_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
      apple_x<=20;
      apple_y<=13;
      random_for_x<=521;
      random_for_y<=133;
      end
```

```
41        else
42        if (game_status==PLAYING && get_apple==1)
43      begin
44        // 防止苹果x和y坐标超范围
45        apple_x<=(random_for_x[5:0]+1>46?(random_for_x[5:0]+1-20):(
    random_for_x[5:0]+1));
46        apple_y<=(random_for_y[4:0]+1>25?(random_for_y[4:0]+1-10):(
    random_for_y[4:0]+1));
47      end
48        else
49        begin
50        apple_x<=apple_x;
51        apple_y<=apple_y;
52        end
53
54      end
55 endmodule
```

score 模块:

```
1 `timescale 1ns / 1ps
2
3 module score(
4   // 变量说明见top模块
5   input clock,
6   input reset,
7   input get_apple,
8   input [1:0] game_status,
9
10  output [7:0] an,
11  output [7:0] seg
12    );
13
14    localparam LAUNCHING=2'b00;
15    localparam PLAYING=2'b01;
16    localparam DIE_FLASHING=2'b10;
17    localparam INITIALIZING=2'b11;
18
19    wire real_enable;
20    wire real_reset;
21
```

```
//    assign real_enable = (get_apple==1) && (game_status==PLAYING); // 计分
//    assign real_reset = (reset==1) | (game_status==INITIALIZING);  // 清空

    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;
    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



    // 将千位、百位、十位、个位数字的4位BDC码在数码管上显示
    seg (
        .reset(reset),
        .clock(clock),
        .an(an),
        .seg(seg),
```

```
        .score(Myscore),
        .total_time(total_time)
          );

endmodule
```

子模块 seg：七段数码管显示部分

```
`timescale 1ns / 1ps

module seg(
  // 变量说明见top模块
    input reset,
    input clock,
    input [31:0] score,       //要输出的分数
    input [31:0] total_time,  //要输出的用时
    output reg [7:0] seg,    //八位数码管（的某一位）输出
    output reg [7:0] an      //选择输出为八位数码管其中一个，低电平有效

    );

    wire [3:0] score_a,score_b,score_c,score_d; // 分别为千位、百位、十位、个位数字的4位BDC码
    assign score_d = score % 10;
    assign score_c = (score % 100) / 10;
    assign score_b = score / 100;

    wire [7:0] b_seg,c_seg,d_seg;
    //依次为百位，十位，个位

    bcdto8segment_dataflow Sc2(.num(score_b),.seg(b_seg));
    bcdto8segment_dataflow Sc1(.num(score_c),.seg(c_seg));
    bcdto8segment_dataflow Sc0(.num(score_d),.seg(d_seg));
    //将它们转换为供八位数码管显示用的格式

    //用时的四位数据
    wire [3:0] total_time_decs0;
    wire [3:0] total_time_decs1;
    wire [3:0] total_time_decs2;
    wire [3:0] total_time_decs3;
    //用时的七段数码管流
```

```
33      wire [7:0] total_time_segflow0;
34      wire [7:0] total_time_segflow1;
35      wire [7:0] total_time_segflow2;
36      wire [7:0] total_time_segflow3;
37      assign total_time_decs0 = total_time % 10;
38      assign total_time_decs1 = (total_time % 100) / 10;
39      assign total_time_decs2 = (total_time % 1000) / 100;
40      assign total_time_decs3 = (total_time / 1000);
41
42      bcdto8segment_dataflow T0(.num(total_time_decs0),.seg(
        total_time_segflow0));
43      bcdto8segment_dataflow T1(.num(total_time_decs1),.seg(
        total_time_segflow1));
44      bcdto8segment_dataflow T2(.num(total_time_decs2),.seg(
        total_time_segflow2));
45      bcdto8segment_dataflow T3(.num(total_time_decs3),.seg(
        total_time_segflow3));
46
47
48  /*---------以下为刷新数码管所用代码--------*/
49      reg [18:0] count;
50
51      initial
52      begin
53      count<=0;
54      end
55
56      always @(posedge clock or posedge reset)
57      begin
58        if (reset==1)
59        begin
60          an<=8'b00000000;
61          seg<=8'b10000000;
62          count<=0;
63        end
64        else
65        begin
66          // count每数100000切换数码管，相当于每次切换的时间为100k/100M
        =1/1000s=1ms
67        if (count==80000)
68              begin
```

```
                an <= 8'b11101111;
                seg <= total_time_segflow0;
                count <= 0;
            end
            else if (count==70000)
            begin
                an <= 8'b11011111;
                seg <= total_time_segflow1;
                count <= count + 1;
            end
            else if (count==60000)
            begin
                an <= 8'b10111111;
                seg <= total_time_segflow2;
                count <= count + 1;
            end
            else if (count == 50000)
            begin
                an <= 8'b01111111;
                seg <= total_time_segflow3;
                count <= count + 1;
            end
        else if (count==40000)
        begin
          an<=8'b11111110;
          seg<=d_seg;
          count <= count + 1;
        end
        else if (count==30000)
        begin
          an<=8'b11111101;
          seg<=c_seg;
          count<=count+1;
        end
        else if (count==20000)
        begin
          an<=8'b11111011;
          seg<=b_seg;
          count<=count+1;
        end
        else if (count==10000)
```

```
            begin
                an<=8'b11110111;
                seg<= 8'b11000000;
                count<=count+1;
            end
        else count<=count+1;
    end
    end
endmodule
```

子模块 becto8segment_dataflow： 负责将数字转换为七段数码管所用的数据流

```
`timescale 1ns / 1ps
//八位数码管显示
module bcdto8segment_dataflow(
    input [3:0] num,           //要输出的数字 (0-9)
  output reg [7:0] seg      //转换为数码管编码
);

always @(num)
    begin
        case(num)
        4'b0000 : seg <= 8'b11000000;
        4'b0001 : seg <= 8'b11111001;
        4'b0010 : seg <= 8'b10100100;
        4'b0011 : seg <= 8'b10110000;
        4'b0100 : seg <= 8'b10011001;
        4'b0101 : seg <= 8'b10010010;
        4'b0110 : seg <= 8'b10000010;
        4'b0111 : seg <= 8'b11111000;
        4'b1000 : seg <= 8'b10000000;
        4'b1001 : seg <= 8'b10010000;
        default : seg <= 8'b1xxxxxxx;
        endcase
    end

endmodule
```

display 模块: VGA 显示

```
module display(
```

```
// 变量说明见top模块
  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 [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;
```

```
43
44  vga_sync_generator(
45      .clock(VGA_CLK[1]),
46      .h_sync(h_sync),
47      .v_sync(v_sync),
48      .x_counter(x_counter),
49      .y_counter(y_counter),
50      .in_display_area(in_display_area)
51    );
52
53
54  // 用于模块间传递二维数组 [5:0] snake_x/y [31:0]
55  // 参考自 https://stackoverflow.com/questions/16369698/how-to-pass-array-
     structure-between-two-verilog-modules
56  integer i;
57  always @(snake_x_temp,snake_y_temp)
58  begin
59  for (i=0;i<32;i=i+1)
60    begin
61      // 片选
62      snake_x[i]<=snake_x_temp[6*i+:6];
63      snake_y[i]<=snake_y_temp[6*i+:6];
64    end
65  end
66
67  always @(x_counter or y_counter)
68  begin
69    current_x<=x_counter/21;
70    current_y<=y_counter/28;
71  end
72
73  wire [11:0]RGB_snake_head;
74  wire [11:0]RGB_snake_body;
75
76  assign RGB_snake_head[0] = snake_color[0];
77  assign RGB_snake_head[1] = snake_color[0];
78  assign RGB_snake_head[2] = 1;
79  assign RGB_snake_head[3] = snake_color[0];
80  assign RGB_snake_head[4] = snake_color[1];
81    assign RGB_snake_head[5] = snake_color[1];
82    assign RGB_snake_head[6] = 1;
```

```
    assign RGB_snake_head[7] = snake_color[1];
  assign RGB_snake_head[8] = snake_color[2];
    assign RGB_snake_head[9] = snake_color[2];
    assign RGB_snake_head[10] = 1;
    assign RGB_snake_head[11] = snake_color[2];

  assign RGB_snake_body[0] = snake_color[0];
  assign RGB_snake_body[1] = snake_color[0];
  assign RGB_snake_body[2] = snake_color[0];
  assign RGB_snake_body[3] = 0;
  assign RGB_snake_body[4] = snake_color[1];
    assign RGB_snake_body[5] = snake_color[1];
    assign RGB_snake_body[6] = snake_color[1];
    assign RGB_snake_body[7] = 0;
  assign RGB_snake_body[8] = snake_color[2];
    assign RGB_snake_body[9] = snake_color[2];
    assign RGB_snake_body[10] = snake_color[2];
    assign RGB_snake_body[11] = 0;


    always @(posedge VGA_CLK[1])
    begin
      begin
        if (in_display_area==0) vga<=0;
        else if (current_x==0 || current_x==47 || current_y==0 || current_y
    == 26) vga<=12'b1111_1111_1111; //边框
        else if (current_x == apple_x && current_y == apple_y) vga<=12'
    b1011_0100_1001; //苹果
        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
        (
        (current_x == snake_x[1] && current_y == snake_y[1] &&
    snake_piece_is_display[1]==1) ||
        (current_x == snake_x[2] && current_y == snake_y[2] &&
    snake_piece_is_display[2]==1) ||
        (current_x == snake_x[3] && current_y == snake_y[3] &&
    snake_piece_is_display[3]==1) ||
        (current_x == snake_x[4] && current_y == snake_y[4] &&
    snake_piece_is_display[4]==1) ||
        (current_x == snake_x[5] && current_y == snake_y[5] &&
```

```
    snake_piece_is_display[5]==1) ||
117         (current_x == snake_x[6] && current_y == snake_y[6] &&
    snake_piece_is_display[6]==1) ||
118         (current_x == snake_x[7] && current_y == snake_y[7] &&
    snake_piece_is_display[7]==1) ||
119         (current_x == snake_x[8] && current_y == snake_y[8] &&
    snake_piece_is_display[8]==1) ||
120         (current_x == snake_x[9] && current_y == snake_y[9] &&
    snake_piece_is_display[9]==1) ||
121         (current_x == snake_x[10] && current_y == snake_y[10] &&
    snake_piece_is_display[10]==1) ||
122         (current_x == snake_x[11] && current_y == snake_y[11] &&
    snake_piece_is_display[11]==1) ||
123         (current_x == snake_x[12] && current_y == snake_y[12] &&
    snake_piece_is_display[12]==1) ||
124         (current_x == snake_x[13] && current_y == snake_y[13] &&
    snake_piece_is_display[13]==1) ||
125         (current_x == snake_x[14] && current_y == snake_y[14] &&
    snake_piece_is_display[14]==1) ||
126         (current_x == snake_x[15] && current_y == snake_y[15] &&
    snake_piece_is_display[15]==1) ||
127         (current_x == snake_x[16] && current_y == snake_y[16] &&
    snake_piece_is_display[16]==1) ||
128         (current_x == snake_x[17] && current_y == snake_y[17] &&
    snake_piece_is_display[17]==1) ||
129         (current_x == snake_x[18] && current_y == snake_y[18] &&
    snake_piece_is_display[18]==1) ||
130         (current_x == snake_x[19] && current_y == snake_y[19] &&
    snake_piece_is_display[19]==1) ||
131         (current_x == snake_x[20] && current_y == snake_y[20] &&
    snake_piece_is_display[20]==1) ||
132         (current_x == snake_x[21] && current_y == snake_y[21] &&
    snake_piece_is_display[21]==1) ||
133         (current_x == snake_x[22] && current_y == snake_y[22] &&
    snake_piece_is_display[22]==1) ||
134         (current_x == snake_x[23] && current_y == snake_y[23] &&
    snake_piece_is_display[23]==1) ||
135         (current_x == snake_x[24] && current_y == snake_y[24] &&
    snake_piece_is_display[24]==1) ||
136         (current_x == snake_x[25] && current_y == snake_y[25] &&
    snake_piece_is_display[25]==1) ||
```

```
        (current_x == snake_x[26] && current_y == snake_y[26] &&
    snake_piece_is_display[26]==1) ||
        (current_x == snake_x[27] && current_y == snake_y[27] &&
    snake_piece_is_display[27]==1) ||
        (current_x == snake_x[28] && current_y == snake_y[28] &&
    snake_piece_is_display[28]==1) ||
        (current_x == snake_x[29] && current_y == snake_y[29] &&
    snake_piece_is_display[29]==1) ||
        (current_x == snake_x[30] && current_y == snake_y[30] &&
    snake_piece_is_display[30]==1) ||
        (current_x == snake_x[31] && current_y == snake_y[31] &&
    snake_piece_is_display[31]==1)
        )
        vga <= RGB_snake_body;//身子
        else vga<=0;
      end
    end
endmodule
```

子模块：vga_sync_generator：生成行场同步信号

```
`timescale 1ns / 1ps

module vga_sync_generator(
  // 变量说明见top模块
  input clock,
  output reg h_sync,v_sync,
  output reg [11:0] x_counter, // 列计数
  output reg [10:0] y_counter, // 行计数
  output reg in_display_area // 是否在显示区域 (x_counter<1920 && y_counter
   <1080)
  );

  localparam h_active_pixels= 1024;
  localparam h_front_porch=24;
  localparam h_sync_width=136;
  localparam h_back_porch=160;
  localparam h_total_piexls=(h_active_pixels+h_front_porch+h_back_porch+
   h_sync_width);

  localparam v_active_pixels=768;
```

```
19  localparam v_front_porch = 3;
20  localparam v_sync_width = 6;
21  localparam v_back_porch = 29;
22  localparam v_total_piexls=(v_active_pixels+v_front_porch+v_back_porch+
     v_sync_width);
23
24
25  // counter是否计满
26  wire x_counter_max = (x_counter == h_total_piexls);
27  wire y_counter_max = (y_counter == v_total_piexls);
28
29  always @(posedge clock)
30    if (x_counter_max)
31    x_counter<=0;
32    else
33    x_counter<=x_counter+1;
34
35  always @(posedge clock)
36    if (x_counter_max)
37    begin
38      if (y_counter_max)
39      y_counter<=0;
40      else
41      y_counter<=y_counter+1; // y_counter只在x_counter满而y_counter未满时
     才加1
42    end
43
44  always @(posedge clock)
45    begin
46    h_sync<=!(x_counter>h_active_pixels+h_front_porch && x_counter<
     h_active_pixels+h_front_porch+h_sync_width);
47    v_sync<=!(y_counter>v_active_pixels+v_front_porch && y_counter<
     v_active_pixels+v_front_porch+v_sync_width);
48    end
49
50  always @(posedge clock)
51    begin
52    in_display_area<=(x_counter<h_active_pixels) && (y_counter<
     v_active_pixels);
53    end
54
```

```
55  endmodule
```

Audio 模块：音效输出

```
1  `timescale 1ns / 1ps
2
3  module Audio(
4      input clk,
5  //    input rst,
6  //    input SD_ctrl,
7
8      input hit_wall,
9      input hit_itself,
10     input get_apple,
11     input up,right,down,left,
12
13     output AUD_PWM,
14     output AUD_SD
15
16     );
17
18 //assign AUD_SD = (hit_wall | hit_itself | get_apple | up | right | down |
       left);
19
20 reg [31:0] n_2;//分频系数
21 reg [7:0]pitch;
22 reg counter_ena;     // 用于开始计时
23 reg [127:0]counter; // 用于计时响多久
24
25 assign AUD_SD = counter_ena;
26
27     //控制音高，音长
28     always @(posedge clk)
29     begin
30     if(counter_ena)
31     begin
32         if(counter == 50000000)
33         begin
34             counter_ena = 0;
35             counter = 0;
36         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)
        begin
            counter_ena = 1;
            counter = 0;
            pitch = 5;
        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);

```

```
78
79 endmodule
```

子模块 get_wave：获取波形

```
1
2  module get_wave(
3      input clk_in,
4      input [31:0]n_2,
5      input rst_n,
6
7      output reg clk_out //分频后的时钟
8  );
9
10 reg [31:0] counter;
11 wire [31:0] n;
12 assign n = n_2 / 2;
13
14 initial
15 begin
16     clk_out <= 0;
17     counter <= 32'b0;
18 end
19
20 always@(posedge clk_in or negedge rst_n)
21 begin
22     if(rst_n == 0)
23     begin
24         clk_out = 0;
25         counter = 0;
26     end
27     else if(n != 32'b0)
28     begin
29         if(counter < n)
30             counter <= counter + 1;
31         else
32         begin
33             clk_out <= ~clk_out;
34             counter <= 32'b0;
35         end
36     end
```

```
37
38 end
39
40 endmodule
```

/*————-代码部分完毕————*/