```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    22:48:08 05/14/06
7   // Design Name:
8   // Module Name:    usercmd
9   // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module usercmd(clk,reset,bad_move,lose,ascii,ascii_ready,movement);
22     input clk, reset, ascii_ready, bad_move, lose;
23     input [7:0] ascii;
24     output [3:0] movement;  //up, down, left, right
25
26     reg [3:0] movement;
27
28     always @ (posedge clk)
29        begin
30           if(reset | bad_move | lose)
31              movement <= 4'b0000;
32           else if(ascii_ready)
33              begin
34                 if(ascii == 8'h52) movement <= 4'b1000;
35                 else if(ascii == 8'h46) movement <= 4'b0100;
36                 else if(ascii == 8'h44) movement <= 4'b0010;
37                 else if(ascii == 8'h47) movement <= 4'b0001;
38              end
39        end
40  endmodule
41
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    20:35:25 05/03/06
7   // Design Name:
8   // Module Name:    apples
9   // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module applesprite(clk,x,y,tar_x,tar_y,RGB);
22
23     input clk;
24     input [9:0] x, y;
25     input [9:0] tar_x, tar_y;
26     output [23:0] RGB;
27
28     reg [23:0] RGB;
29
30     parameter WIDTH = 10;
31     parameter HEIGHT = 10;
32     parameter COLOR = 24'hFF0000;
33
34     always @ (posedge clk)
35        begin
36           if((((x > tar_x) || (x == tar_x)) && (x < tar_x+WIDTH)) &&
37              (((y > tar_y) || (y == tar_y)) && (y < tar_y+HEIGHT)))
38              RGB <= COLOR;
39           else
40              RGB <= 24'h0;
41        end
42
43  endmodule
44
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    23:14:43 05/14/06
7   // Design Name:
8   // Module Name:    colfsm
9   // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module colfsm(clk,reset,grow_idx,head_x,head_y,up,down,left,right,index,x,y,apple_x,apple_
    y,
22                rand_x,rand_y,lose,win,grow,addr,rgb,movement);
23     input clk, reset;
24     input [9:0] head_x, head_y, x, y, rand_x, rand_y;
25     input up, down, left, right;
26     input [2:0] rgb;
27     input [4:0] grow_idx;
28     input [3:0] movement;
29     output [4:0] index;
30     output [18:0] addr;
31     output [9:0] apple_x, apple_y;
32     output lose, win, grow;
33
34     parameter RESET = 0;
35     parameter CHECK_MOVE = 1;
36     parameter CHECK_LVL1 = 2;
37     parameter CHECK_LVL2 = 3;
38     parameter CHECK_SNAKE = 4;
39     parameter CHECK_APPLE = 5;
40     parameter GEN_APPLE1 = 6;
41     parameter GEN_APPLE2 = 7;
42     parameter GEN_APPLE3 = 8;
43     parameter GEN_APPLE4 = 9;
44     parameter GEN_APPLE5 = 10;
45     parameter GAME_OVER = 11;
46
47     reg [3:0] state;
48     reg [3:0] next;
49     reg [4:0] index;
50     reg [18:0] addr;
51     reg [9:0] apple_x, apple_y, tx, ty;
52     reg lose, win, grow, apple_ok;
53
54     always @ (posedge clk)
55        begin
56           if(reset)
57              begin
58                 apple_x <= 10'd100;
59                 apple_y <= 10'd100;
60                 state <= RESET;
61              end
62           else
```

```
63              begin
64                 apple_x <= (apple_ok) ? tx : apple_x;
65                 apple_y <= (apple_ok) ? ty : apple_y;
66                 state <= next;
67              end
68        end
69
70     always @ (state or up or down or left or right or index or rgb or grow_idx or head_x or
71                head_y or x or y or addr or rand_x or rand_y or tx or ty or movement or apple
   _x or apple_y)
72        begin
73           lose = 0;
74           win = 0;
75           grow = 0;
76           apple_ok = 0;
77           case(state)
78              RESET:   begin
79                          index = 5'd31;
80                          tx = 10'd100;
81                          ty = 10'd100;
82                    //    addr = (up) ? head_x + (640 * head_y) : (down) ? head_x + (640 * (
   head_y + 19)) :
83                    //           (left) ? head_x + (640 * head_y) : (right)? head_x + 19 + (
   640 * head_y) :
84                    //                    head_x + (640 * head_y);
85                          addr = head_x + (640 * head_y);
86                          next = CHECK_LVL1;
87                       end
88
89              CHECK_LVL1: if(rgb == 3'b111)
90                          begin
91                       //    lose = 1;
92                             next = GAME_OVER;
93                          end
94                       else
95                          begin
96                       //       addr = (up) ? head_x + 19 + (640 * head_y) : (down) ? head_x
    + 19 + (640 * (head_y + 19)) :
97                       //                (left) ? head_x + (640 * (head_y + 19)) : (right)? he
   ad_x + 19 + (640 * (head_y + 19)) :
98                       //                         head_x + 19 + (640 * head_y);
99                             addr = head_x + 19 + (640 * (head_y + 19));
100                            next = CHECK_LVL2;
101                         end
102
103             CHECK_LVL2: if(rgb == 3'b111)
104                         begin
105                      //    lose = 1;
106                            next = GAME_OVER;
107                         end
108                      else next = CHECK_LVL1;
109
110             CHECK_SNAKE: begin
111                         if(grow_idx == 0)
112                            begin
113                               win = 1;
114                               next = CHECK_LVL1;
115                            end
116                         else if(index == grow_idx + 5)
117                            begin
118                               index = 5'd31;
119                               next = CHECK_APPLE;
```

```
120                                 end
121                             else if(up && (((head_x >= x) && (head_x <= x + 19) && (head_y
    <= y + 19)) ||
122                                     ((head_x <= x) && (head_x + 19 >= x) && (head_y <= y +
     19))))
123                                 begin
124                                     //lose = 1;
125                                     next = GAME_OVER;
126                                 end
127                             else if(down && (((head_x >= x) && (head_x <= x + 19) && (head_
    y + 19 >= y)) ||
128                                     ((head_x <= x) && (head_x + 19 >= x) && (head_y + 19 >
    = y))))
129                                 begin
130                                     //lose = 1;
131                                     next = GAME_OVER;
132                                 end
133                             else if(left && (((head_y + 19 >= y) && (head_y <= y) && (head_
    x <= x + 19)) ||
134                                     ((head_y >= y) && (head_y <= y + 19) && (head_x <= x +
     19))))
135                                 begin
136                                     //lose = 1;
137                                     next = GAME_OVER;
138                                 end
139                             else if(right && (((head_y >= y) && (head_y <= y + 19) && (head
    _x + 19 >= x)) ||
140                                     ((head_y + 19 >= y) && (head_y <= y) && (head_x + 19 >
    = x))))
141                                 begin
142                                     //lose = 1;
143                                     next = GAME_OVER;
144                                 end
145                             else
146                                 begin
147                                     index = index - 1;
148                                     next = CHECK_SNAKE;
149                                 end
150                         end
151             CHECK_APPLE: begin
152                             if((up && (head_x < apple_x) && (head_x + 19 > apple_x + 9) &&
    (head_y <= apple_y + 9)) ||
153                                 (down && (head_x < apple_x) && (head_x + 19 > apple_x + 9) &
    & (head_y + 19 >= apple_y)) ||
154                                 (left && (head_y < apple_y) && (head_y + 19 > apple_y + 9) &
    & (head_x <= apple_x + 9)) ||
155                                 (right && (head_y < apple_y) && (head_y + 19 > apple_y + 9)
    && (head_x + 19 >= apple_x)))
156                                 begin
157                                     grow = 1;
158                                     tx = rand_x;
159                                     ty = rand_y;
160                                     addr = rand_x + (rand_y * 640);
161                                     next = GEN_APPLE1;
162                                 end
163                             next = CHECK_LVL1;
164                         end
165             GEN_APPLE1: begin
166                             if(tx < 10'd620 && ty < 10'd460 && tx > 10'd20 && ty > 10'd20 &
    & rgb == 3'b000)
167                                 begin
168                                     addr = tx + 9 + (640 * ty);
169                                     next = GEN_APPLE2;
```

```
170                                end
171                             else
172                                begin
173                                   tx = rand_x;
174                                   ty = rand_y;
175                                   addr = rand_x + (rand_y * 640);
176                                   next = GEN_APPLE1;
177                                end
178                          end
179             GEN_APPLE2: begin
180                             if(rgb == 3'b000)
181                                begin
182                                   addr = tx + (640 * (ty + 9));
183                                   next = GEN_APPLE3;
184                                end
185                             else
186                                begin
187                                   tx = rand_x;
188                                   ty = rand_y;
189                                   addr = rand_x + (rand_y * 640);
190                                   next = GEN_APPLE1;
191                                end
192                          end
193             GEN_APPLE3: begin
194                             if(rgb == 3'b000)
195                                begin
196                                   addr = tx + 9 + (640 * (ty + 9));
197                                   next = GEN_APPLE4;
198                                end
199                             else
200                                begin
201                                   tx = rand_x;
202                                   ty = rand_y;
203                                   addr = rand_x + (rand_y * 640);
204                                   next = GEN_APPLE1;
205                                end
206                          end
207             GEN_APPLE4: begin
208                             if(rgb == 3'b000)
209                                begin
210                                   index = 5'd31;
211                                   next = GEN_APPLE5;
212                                end
213                             else
214                                begin
215                                   tx = rand_x;
216                                   ty = rand_y;
217                                   addr = rand_x + (rand_y * 640);
218                                   next = GEN_APPLE1;
219                                end
220                          end
221             GEN_APPLE5: begin
222                             if(index == grow_idx)
223                                begin
224                                   index = 5'd31;
225                                   apple_ok = 1;
226                                   next = CHECK_LVL1;
227                                end
228                             else if((tx > x - 10) && (tx + 10 < x + 30) && (ty > y - 10) &&
     (ty + 10 < y + 30))
229                                begin
230                                   tx = rand_x;
231                                   ty = rand_y;
```

```
232                                 addr = rand_x + (rand_y * 640);
233                                 next = GEN_APPLE1;
234                              end
235                           else
236                              begin
237                                 index = index - 1;
238                                 next = GEN_APPLE5;
239                              end
240                        end
241            GAME_OVER:  begin
242                     //    lose = 1;
243                           next = GAME_OVER;
244                        end
245         endcase
246      end
247   endmodule
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    13:46:35 03/24/06
7   // Design Name:
8   // Module Name:    display
9   // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module display(clk,reset,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x1
    9,
22                 x20,x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,y1,y2,y3,y4,y5,y6,y7,y8
    ,y9,
23                 y10,y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26,y27,y28
    ,y29,
24                 y30,y31,y32,apple_x,apple_y,lvl_rgb,pixel_count,line_count,RGB,addr);
25     input clk;
26     input reset;
27     input [9:0] x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21,
28                 x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,
29                 y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26,y27,y28,y29
    ,y30,y31,y32;
30     input [9:0] apple_x;
31     input [9:0] apple_y;
32     input [2:0] lvl_rgb;
33     input [9:0] pixel_count;
34     input [9:0] line_count;
35     output [23:0] RGB;
36     output [18:0] addr;
37
38     reg [23:0] RGB;
39     wire clk;
40     wire [9:0] pixel_count;
41     wire [9:0] line_count;
42     wire [23:0] rgb1, rgb2, rgb3, rgb4, rgb5, rgb6, rgb7, rgb8, rgb9, rgb10, rgb11, rgb12,
43                 rgb13, rgb14, rgb15, rgb16, rgb17, rgb18, rgb19, rgb20, rgb21, rgb22, rgb23
    ,
44                 rgb24, rgb25, rgb26, rgb27, rgb28, rgb29, rgb30, rgb31, rgb32, rgb33;
45  // wire [9:0] x1;
46  // wire [9:0] y1;
47     wire [9:0] apple_x;
48     wire [9:0] apple_y;
49     snakesprite s1(clk,pixel_count,line_count,x1,y1,rgb1);
50     defparam s1.COLOR = 24'h0000FF;
51     snakesprite s2(clk,pixel_count,line_count,x2,y2,rgb2);
52     snakesprite s3(clk,pixel_count,line_count,x3,y3,rgb3);
53     snakesprite s4(clk,pixel_count,line_count,x4,y4,rgb4);
54     snakesprite s5(clk,pixel_count,line_count,x5,y5,rgb5);
55     snakesprite s6(clk,pixel_count,line_count,x6,y6,rgb6);
56     snakesprite s7(clk,pixel_count,line_count,x7,y7,rgb7);
57     snakesprite s8(clk,pixel_count,line_count,x8,y8,rgb8);
58     snakesprite s9(clk,pixel_count,line_count,x9,y9,rgb9);
```

```
59       snakesprite s10(clk,pixel_count,line_count,x10,y10,rgb10);
60       snakesprite s11(clk,pixel_count,line_count,x11,y11,rgb11);
61       snakesprite s12(clk,pixel_count,line_count,x12,y12,rgb12);
62       snakesprite s13(clk,pixel_count,line_count,x13,y13,rgb13);
63       snakesprite s14(clk,pixel_count,line_count,x14,y14,rgb14);
64       snakesprite s15(clk,pixel_count,line_count,x15,y15,rgb15);
65       snakesprite s16(clk,pixel_count,line_count,x16,y16,rgb16);
66       snakesprite s17(clk,pixel_count,line_count,x17,y17,rgb17);
67       snakesprite s18(clk,pixel_count,line_count,x18,y18,rgb18);
68       snakesprite s19(clk,pixel_count,line_count,x19,y19,rgb19);
69       snakesprite s20(clk,pixel_count,line_count,x20,y20,rgb20);
70       snakesprite s21(clk,pixel_count,line_count,x21,y21,rgb21);
71       snakesprite s22(clk,pixel_count,line_count,x22,y22,rgb22);
72       snakesprite s23(clk,pixel_count,line_count,x23,y23,rgb23);
73       snakesprite s24(clk,pixel_count,line_count,x24,y24,rgb24);
74       snakesprite s25(clk,pixel_count,line_count,x25,y25,rgb25);
75       snakesprite s26(clk,pixel_count,line_count,x26,y26,rgb26);
76       snakesprite s27(clk,pixel_count,line_count,x27,y27,rgb27);
77       snakesprite s28(clk,pixel_count,line_count,x28,y28,rgb28);
78    // defparam s28.COLOR = 24'h0000FF;
79       snakesprite s29(clk,pixel_count,line_count,x29,y29,rgb29);
80    // defparam s29.COLOR = 24'hFF0000;
81       snakesprite s30(clk,pixel_count,line_count,x30,y30,rgb30);
82       snakesprite s31(clk,pixel_count,line_count,x31,y31,rgb31);
83       snakesprite s32(clk,pixel_count,line_count,x32,y32,rgb32);
84       //defparam s32.COLOR = 24'h0000FF;
85       applesprite a1(clk,pixel_count,line_count,apple_x,apple_y,rgb33);
86
87       always @ (posedge clk)
88          begin
89             if(reset)   RGB <= 24'h0;
90             else
91                begin
92                   if(lvl_rgb)
93                      RGB <= {lvl_rgb[2],lvl_rgb[2],lvl_rgb[2],lvl_rgb[2],lvl_rgb[2],lvl_rgb[2
      ],lvl_rgb[2],lvl_rgb[2],
94                              lvl_rgb[1],lvl_rgb[1],lvl_rgb[1],lvl_rgb[1],lvl_rgb[1],lvl_rgb[1
      ],lvl_rgb[1],lvl_rgb[1],
95                              lvl_rgb[0],lvl_rgb[0],lvl_rgb[0],lvl_rgb[0],lvl_rgb[0],lvl_rgb[0
      ],lvl_rgb[0],lvl_rgb[0]};
96                   else
97                      RGB <= (rgb1 | rgb2 | rgb3 | rgb4 | rgb5 | rgb6 | rgb7 | rgb8 | rgb9 | r
      gb10 |
98                              rgb11 | rgb12 | rgb13 | rgb14 | rgb15 | rgb16 | rgb17 | rgb18 |
      rgb19 |
99                              rgb20 | rgb21 | rgb22 | rgb23 | rgb24 | rgb25 | rgb26 | rgb27 |
      rgb28 |
100                             rgb29 | rgb30 | rgb31 | rgb32 | rgb33);
101               end
102         end
103
104      assign addr = pixel_count + (640 * line_count);
105
106   endmodule
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    19:26:41 05/03/06
7   // Design Name:
8   // Module Name:    disprf
9   // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module disprf(clk,reset,pixel_count,line_count,en);
22
23     input clk, reset;
24     input [9:0] pixel_count, line_count;
25     output en;
26     reg en;
27
28     always @ (posedge clk)
29        begin
30           if(reset) en <= 0;
31           else en <= ((pixel_count == 639) && (line_count == 479));
32        end
33  endmodule
34
```

```
1   //////////////////////////////////////////////////////////////////////////////
2   //
3   // 6.111 FPGA Labkit -- Template Toplevel Module for Lab 4 (Spring 2006)
4   //
5   //
6   // Created: March 13, 2006
7   // Author: Nathan Ickes
8   //
9   //////////////////////////////////////////////////////////////////////////////
10
11  module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
12            ac97_bit_clock,
13
14            vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
15            vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
16            vga_out_vsync,
17
18            tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
19            tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
20            tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,
21
22            tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
23            tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
24            tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
25            tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,
26
27            ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
28            ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,
29
30            ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
31            ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
32
33            clock_feedback_out, clock_feedback_in,
34
35            flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
36            flash_reset_b, flash_sts, flash_byte_b,
37
38            rs232_txd, rs232_rxd, rs232_rts, rs232_cts,
39
40            mouse_clock, mouse_data, keyboard_clock, keyboard_data,
41
42            clock_27mhz, clock1, clock2,
43
44            disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
45            disp_reset_b, disp_data_in,
46
47            button0, button1, button2, button3, button_enter, button_right,
48            button_left, button_down, button_up,
49
50            switch,
51
52            led,
53
54            user1, user2, user3, user4,
55
56            daughtercard,
57
58            systemace_data, systemace_address, systemace_ce_b,
59            systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
60
61            analyzer1_data, analyzer1_clock,
62            analyzer2_data, analyzer2_clock,
63            analyzer3_data, analyzer3_clock,
```

```
64            analyzer4_data, analyzer4_clock);
65
66      output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
67      input  ac97_bit_clock, ac97_sdata_in;
68
69      output [7:0] vga_out_red, vga_out_green, vga_out_blue;
70      output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
71        vga_out_hsync, vga_out_vsync;
72
73      output [9:0] tv_out_ycrcb;
74      output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
75        tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
76        tv_out_subcar_reset;
77
78      input  [19:0] tv_in_ycrcb;
79      input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
80        tv_in_hff, tv_in_aff;
81      output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
82        tv_in_reset_b, tv_in_clock;
83      inout  tv_in_i2c_data;
84
85      inout  [35:0] ram0_data;
86      output [18:0] ram0_address;
87      output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
88      output [3:0] ram0_bwe_b;
89
90      inout  [35:0] ram1_data;
91      output [18:0] ram1_address;
92      output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
93      output [3:0] ram1_bwe_b;
94
95      input  clock_feedback_in;
96      output clock_feedback_out;
97
98      inout  [15:0] flash_data;
99      output [23:0] flash_address;
100     output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
101     input  flash_sts;
102
103     output rs232_txd, rs232_rts;
104     input  rs232_rxd, rs232_cts;
105
106     input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
107
108     input  clock_27mhz, clock1, clock2;
109
110     output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
111     input  disp_data_in;
112     output  disp_data_out;
113
114     input  button0, button1, button2, button3, button_enter, button_right,
115       button_left, button_down, button_up;
116     input  [7:0] switch;
117     output [7:0] led;
118
119     inout [31:0] user1, user2, user3, user4;
120
121     inout [43:0] daughtercard;
122
123     inout  [15:0] systemace_data;
124     output [6:0]  systemace_address;
125     output systemace_ce_b, systemace_we_b, systemace_oe_b;
126     input  systemace_irq, systemace_mpbrdy;
```

```
127
128     output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
129         analyzer4_data;
130     output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;
131
132     ////////////////////////////////////////////////////////////////////////
133     //
134     // I/O Assignments
135     //
136     ////////////////////////////////////////////////////////////////////////
137
138     // Audio Input and Output
139     assign beep= 1'b0;
140     assign audio_reset_b = 1'b0;
141     assign ac97_synch = 1'b0;
142     assign ac97_sdata_out = 1'b0;
143
144     // Video Output
145     assign tv_out_ycrcb = 10'h0;
146     assign tv_out_reset_b = 1'b0;
147     assign tv_out_clock = 1'b0;
148     assign tv_out_i2c_clock = 1'b0;
149     assign tv_out_i2c_data = 1'b0;
150     assign tv_out_pal_ntsc = 1'b0;
151     assign tv_out_hsync_b = 1'b1;
152     assign tv_out_vsync_b = 1'b1;
153     assign tv_out_blank_b = 1'b1;
154     assign tv_out_subcar_reset = 1'b0;
155
156     // Video Input
157     assign tv_in_i2c_clock = 1'b0;
158     assign tv_in_fifo_read = 1'b0;
159     assign tv_in_fifo_clock = 1'b0;
160     assign tv_in_iso = 1'b0;
161     assign tv_in_reset_b = 1'b0;
162     assign tv_in_clock = 1'b0;
163     assign tv_in_i2c_data = 1'bZ;
164
165     // SRAMs
166     assign ram0_data = 36'hZ;
167     assign ram0_address = 19'h0;
168     assign ram0_adv_ld = 1'b0;
169     assign ram0_clk = 1'b0;
170     assign ram0_cen_b = 1'b1;
171     assign ram0_ce_b = 1'b1;
172     assign ram0_oe_b = 1'b1;
173     assign ram0_we_b = 1'b1;
174     assign ram0_bwe_b = 4'hF;
175     assign ram1_data = 36'hZ;
176     assign ram1_address = 19'h0;
177     assign ram1_adv_ld = 1'b0;
178     assign ram1_clk = 1'b0;
179     assign ram1_cen_b = 1'b1;
180     assign ram1_ce_b = 1'b1;
181     assign ram1_oe_b = 1'b1;
182     assign ram1_we_b = 1'b1;
183     assign ram1_bwe_b = 4'hF;
184     assign clock_feedback_out = 1'b0;
185
186     // Flash ROM
187     assign flash_data = 16'hZ;
188     assign flash_address = 24'h0;
189     assign flash_ce_b = 1'b1;
```

```
190     assign flash_oe_b = 1'b1;
191     assign flash_we_b = 1'b1;
192     assign flash_reset_b = 1'b0;
193     assign flash_byte_b = 1'b1;
194
195     // RS-232 Interface
196     assign rs232_txd = 1'b1;
197     assign rs232_rts = 1'b1;
198
199     // LED Displays
200     assign disp_blank = 1'b1;
201     assign disp_clock = 1'b0;
202     assign disp_rs = 1'b0;
203     assign disp_ce_b = 1'b1;
204     assign disp_reset_b = 1'b0;
205     assign disp_data_out = 1'b0;
206
207     // Buttons, Switches, and Individual LEDs
208     assign led = 8'hFF;
209
210     // User I/Os
211     assign user1 = 32'hZ;
212     assign user2 = 32'hZ;
213     assign user3 = 32'hZ;
214     assign user4 = 32'hZ;
215
216     // Daughtercard Connectors
217     assign daughtercard = 44'hZ;
218
219     // SystemACE Microprocessor Port
220     assign systemace_data = 16'hZ;
221     assign systemace_address = 7'h0;
222     assign systemace_ce_b = 1'b1;
223     assign systemace_we_b = 1'b1;
224     assign systemace_oe_b = 1'b1;
225
226     // Logic Analyzer
227     assign analyzer1_data = 16'h0;
228     assign analyzer1_clock = 1'b1;
229     assign analyzer2_data = 16'h0;
230     assign analyzer2_clock = 1'b1;
231     assign analyzer3_data = 16'h0;
232     assign analyzer3_clock = 1'b1;
233
234
235     ////////////////////////////////////////////////////////////////////////
236     //
237     // Lab 4 Components
238     //
239     ////////////////////////////////////////////////////////////////////////
240
241     //
242     // Generate a 31.5MHz pixel clock from clock_27mhz
243     //
244
245     wire pclk, pixel_clock;
246     DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
247     // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
248     // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
249     // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
250     // synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
251     BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));
252
```

```
253      //
254      // VGA output signals
255      //
256
257      // Inverting the clock to the DAC provides half a clock period for signals
258      // to propagate from the FPGA to the DAC.
259      assign vga_out_pixel_clock = ~pixel_clock;
260
261      // The composite sync signal is used to encode sync data in the green
262      // channel analog voltage for older monitors.  It does not need to be
263      // implemented for the monitors in the 6.111 lab, and can be left at 1'b1.
264      assign vga_out_sync_b = 1'b1;
265
266      // The following assignments should be deleted and replaced with your own
267      // code to implement the Pong game.
268      wire reset_sync;
269      wire redraw_sync;
270      wire [9:0] x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21,
271                 x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,
272                 y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26,y27,y28,y29,
    y30,y31,y32;
273      wire hsync;
274      wire vsync;
275      wire sync_b;
276      wire blank_b;
277      wire [9:0] pixel_count;
278      wire [9:0] line_count;
279      wire [23:0] RGB;
280      wire [2:0] lvl_rgb1;
281      wire [18:0] addr1;
282      wire [18:0] addr2;
283      wire [2:0] lvl_rgb2;
284      wire draw_en;
285      wire [9:0] apple_x, apple_y, head_x, head_y, x, y;
286      wire ascii_ready;
287      wire [7:0] ascii;
288      wire [3:0] movement;
289      wire lose, win, grow, en_lvl1, en_lvl2, en_lvl3;
290      wire up, down, left, right, data_valid;
291      wire [4:0] grow_idx, index;
292      wire [19:0] pd_out;
293      wire bad_move;
294
295      assign analyzer4_data = {5'h0,keyboard_clock,keyboard_data,ascii_ready,ascii};
296      assign analyzer4_clock = clock_27mhz;
297
298      assign vga_out_red = RGB[23:16];
299      assign vga_out_green = RGB[15:8];
300      assign vga_out_blue = RGB[7:0];
301      assign vga_out_blank_b = blank_b;
302      assign vga_out_hsync = hsync;
303      assign vga_out_vsync = vsync;
304
305      debounce db1(1'b0, pixel_clock, ~button0, reset_sync);
306      debounce db2(reset_sync, pixel_clock, ~button1, grow);
307      disprf rf1(pixel_clock,reset_sync,pixel_count,line_count,draw_en);
308      vga vga1(pixel_clock, reset_sync, hsync, vsync, sync_b, blank_b, pixel_count, line_coun
    t);
309      display dp1(pixel_clock, reset_sync,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,
    x16,x17,x18,x19,
310                  x20,x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,y1,y2,y3,y4,y5,y6,y7,y8
    ,y9,
311                  y10,y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26,y27,y28
```

```
311   ,y29,
312                 y30,y31,y32,apple_x,apple_y,lvl_rgb1,pixel_count,line_count,RGB,addr1);
313      romone one(.addra(addr1), .addrb(addr2), .clka(pixel_clock), .clkb(pixel_clock), .douta
      (lvl_rgb1), .doutb(lvl_rgb2),
314               .ena(1'b1), .enb(1'b1));
315      ps2_ascii_input kbd(clock_27mhz,reset_sync,keyboard_clock,keyboard_data,ascii,ascii_rea
      dy);
316      usercmd cntl(pixel_clock,reset_sync,bad_move,lose,ascii,ascii_ready,movement);
317   // romtwo two(.addra(addr1), .addrb(addr2), .clka(pixel_clock), .clkb(pixel_clock), .douta
      (lvl_rgb1), .doutb(lvl_rgb2),
318   //           .ena(1'b1), .enb(1'b1));
319   // romthree three(.addra(addr1), .addrb(addr2), .clka(pixel_clock), .clkb(pixel_clock), .d
      outa(lvl_rgb1), .doutb(lvl_rgb2),
320   //           .ena(en_lvl3), .enb(en_lvl3));
321   // rand rgen(pixel_clock,pd_out,1'b1,data_valid,1'b0);
322   //   colfsm cfsm(pixel_clock,reset_sync,grow_idx,head_x,head_y,up,down,left,right,index,x,
      y,apple_x,apple_y,
323   //               pd_out[9:0],pd_out[19:10],lose,win,grow,addr2,lvl_rgb2,movement);
324      snakeregs sreg(pixel_clock,reset_sync,movement,draw_en,grow,win,head_x,head_y,index,x,y
      ,up,down,left,right,grow_idx,
325                     x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x
      21,
326                     x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,y1,y2,y3,y4,y5,y6,y7,y8,y9,y
      10,
327                     y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26,y27,y28,
      y29,y30,y31,y32,bad_move);
328   // gamefsm gfsm(pixel_clock,reset_sync,lose,win,en_lvl1,en_lvl2,en_lvl3);
329
330   endmodule
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    19:53:20 05/10/06
7   // Design Name:
8   // Module Name:    majorfsm
9   // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module gamefsm(
22     clk,
23     reset,
24     lose,
25     win,
26     en_lvl1,
27     en_lvl2,
28     en_lvl3);
29
30     input clk, reset, lose, win;
31     output en_lvl1, en_lvl2, en_lvl3;
32     reg [1:0] state;
33     reg [1:0] next;
34
35     //the states
36     parameter LVL1 = 0;
37     parameter LVL2 = 1;
38     parameter LVL3 = 2;
39     parameter GAME_OVER = 3;
40
41     always @ (posedge clk)
42        begin
43           if (reset) state <= LVL1;
44           else state <= next;
45        end
46
47     always @ (state or lose or win)
48        begin
49           case(state)
50              LVL1:    begin
51                          if(lose) next = GAME_OVER;
52                          else if(win)   next = LVL2;
53                          else  next = LVL1;
54                       end
55
56              LVL2:    begin
57                          if(lose) next = GAME_OVER;
58                          else if(win)   next = LVL3;
59                          else  next = LVL2;
60                       end
61
62              LVL3:    begin
63                          if(lose || win)   next = GAME_OVER;
```

```
64                          else next = LVL3;
65                      end
66
67              GAME_OVER:  next = GAME_OVER;
68          endcase
69      end
70
71     assign en_lvl1 = (state == LVL1);
72     assign en_lvl2 = (state == LVL2);
73     assign en_lvl3 = (state == LVL3);
74
75  endmodule
```

```
1   //
2   // File:   ps2_kbd.v
3   // Date:   24-Oct-05
4   // Author: C. Terman / I. Chuang
5   //
6   // PS2 keyboard input for 6.111 labkit
7   //
8   // INPUTS:
9   //
10  //   clock_27mhz   - master clock
11  //   reset         - active high
12  //   clock         - ps2 interface clock
13  //   data          - ps2 interface data
14  //
15  // OUTPUTS:
16  //
17  //   ascii         - 8 bit ascii code for current character
18  //   ascii_ready   - one clock cycle pulse indicating new char received
19
20
21  ////////////////////////////////////////////////////////////////////////
22
23  module ps2_ascii_input(clock_27mhz, reset, clock, data, ascii, ascii_ready);
24
25     // module to generate ascii code for keyboard input
26     // this is module works synchronously with the system clock
27
28     input clock_27mhz;
29     input reset;    // Active high asynchronous reset
30     input clock;    // PS/2 clock
31     input data;     // PS/2 data
32     output [7:0] ascii;   // ascii code (1 character)
33     output ascii_ready;   // ascii ready (one clock_27mhz cycle active high)
34
35     reg [7:0]   ascii_val;  // internal combinatorial ascii decoded value
36     reg [7:0]   lastkey;    // last keycode
37     reg [7:0]   curkey;     // current keycode
38     reg [7:0]   ascii;      // ascii output (latched & synchronous)
39     reg          ascii_ready;  // synchronous one-cycle ready flag
40
41     // get keycodes
42
43     wire        fifo_rd;    // keyboard read request
44     wire [7:0]  fifo_data;  // keyboard data
45     wire        fifo_empty; // flag: no keyboard data
46     wire        fifo_overflow; // keyboard data overflow
47
48     ps2 myps2(reset, clock_27mhz, clock, data, fifo_rd, fifo_data,
49          fifo_empty,fifo_overflow);
50
51     assign      fifo_rd = ~fifo_empty;  // continous read
52     reg          key_ready;
53
54     always @(posedge clock_27mhz)
55       begin
56
57     // get key if ready
58
59     curkey <= ~fifo_empty ? fifo_data : curkey;
60     lastkey <= ~fifo_empty ? curkey : lastkey;
61     key_ready  <= ~fifo_empty;
62
63     // raise ascii_ready for last key which was read
```

```
64
65       ascii_ready <= key_ready & ~(curkey[7]|lastkey[7]);
66       ascii <=  (key_ready & ~(curkey[7]|lastkey[7])) ? ascii_val : ascii;
67
68         end
69
70       always @(curkey) begin //convert PS/2 keyboard make code ==> ascii code
71         case (curkey)
72           8'h1C: ascii_val = 8'h41;    //A
73           8'h32: ascii_val = 8'h42;    //B
74           8'h21: ascii_val = 8'h43;    //C
75           8'h23: ascii_val = 8'h44;    //D
76           8'h24: ascii_val = 8'h45;    //E
77           8'h2B: ascii_val = 8'h46;    //F
78           8'h34: ascii_val = 8'h47;    //G
79           8'h33: ascii_val = 8'h48;    //H
80           8'h43: ascii_val = 8'h49;    //I
81           8'h3B: ascii_val = 8'h4A;    //J
82           8'h42: ascii_val = 8'h4B;    //K
83           8'h4B: ascii_val = 8'h4C;    //L
84           8'h3A: ascii_val = 8'h4D;    //M
85           8'h31: ascii_val = 8'h4E;    //N
86           8'h44: ascii_val = 8'h4F;    //O
87           8'h4D: ascii_val = 8'h50;    //P
88           8'h15: ascii_val = 8'h51;    //Q
89           8'h2D: ascii_val = 8'h52;    //R
90           8'h1B: ascii_val = 8'h53;    //S
91           8'h2C: ascii_val = 8'h54;    //T
92           8'h3C: ascii_val = 8'h55;    //U
93           8'h2A: ascii_val = 8'h56;    //V
94           8'h1D: ascii_val = 8'h57;    //W
95           8'h22: ascii_val = 8'h58;    //X
96           8'h35: ascii_val = 8'h59;    //Y
97           8'h1A: ascii_val = 8'h5A;    //Z
98
99           8'h45: ascii_val = 8'h30;    //0
100          8'h16: ascii_val = 8'h31;    //1
101          8'h1E: ascii_val = 8'h32;    //2
102          8'h26: ascii_val = 8'h33;    //3
103          8'h25: ascii_val = 8'h34;    //4
104          8'h2E: ascii_val = 8'h35;    //5
105          8'h36: ascii_val = 8'h36;    //6
106          8'h3D: ascii_val = 8'h37;    //7
107          8'h3E: ascii_val = 8'h38;    //8
108          8'h46: ascii_val = 8'h39;    //9
109
110          8'h0E: ascii_val = 8'h60;    // `
111          8'h4E: ascii_val = 8'h2D;    // -
112          8'h55: ascii_val = 8'h3D;    // =
113          8'h5C: ascii_val = 8'h5C;    // \
114          8'h29: ascii_val = 8'h20;    // (space)
115          8'h54: ascii_val = 8'h5B;    // [
116          8'h5B: ascii_val = 8'h5D;    // ]
117          8'h4C: ascii_val = 8'h3B;    // ;
118          8'h52: ascii_val = 8'h27;    // '
119          8'h41: ascii_val = 8'h2C;    // ,
120          8'h49: ascii_val = 8'h2E;    // .
121          8'h4A: ascii_val = 8'h2F;    // /
122
123          8'h5A: ascii_val = 8'h0D;    // enter (CR)
124          8'h66: ascii_val = 8'h08;    // backspace
125
126          8'h75: ascii_val = 8'h10;    //up
```

```
127         8'h72: ascii_val = 8'h11;    //down
128         8'h74: ascii_val = 8'h12;    //right
129         8'h6B: ascii_val = 8'h13;    //left
130
131         //  8'hF0: ascii_val = 8'hF0;      // BREAK CODE
132
133         default: ascii_val = 8'h23;     // #
134       endcase
135     end
136   endmodule // ps2toascii
137
138   ////////////////////////////////////////////////////////////////////////
139   // new synchronous ps2 keyboard driver, with built-in fifo, from Chris Terman
140
141   module ps2(reset, clock_27mhz, ps2c, ps2d, fifo_rd, fifo_data,
142         fifo_empty,fifo_overflow);
143
144     input clock_27mhz,reset;
145     input ps2c;       // ps2 clock
146     input ps2d;       // ps2 data
147     input fifo_rd;    // fifo read request (active high)
148     output [7:0] fifo_data; // fifo data output
149     output   fifo_empty; // fifo empty (active high)
150     output   fifo_overflow; // fifo overflow - too much kbd input
151
152    reg [3:0] count;      // count incoming data bits
153    reg [9:0] shift;      // accumulate incoming data bits
154
155    reg [7:0] fifo[7:0];   // 8 element data fifo
156    reg fifo_overflow;
157    reg [2:0] wptr,rptr;   // fifo write and read pointers
158
159    wire [2:0] wptr_inc = wptr + 1;
160
161    assign fifo_empty = (wptr == rptr);
162    assign fifo_data = fifo[rptr];
163
164    // synchronize PS2 clock to local clock and look for falling edge
165    reg [2:0] ps2c_sync;
166    always @ (posedge clock_27mhz) ps2c_sync <= {ps2c_sync[1:0],ps2c};
167    wire sample = ps2c_sync[2] & ~ps2c_sync[1];
168
169    always @ (posedge clock_27mhz) begin
170      if (reset) begin
171        count <= 0;
172        wptr <= 0;
173        rptr <= 0;
174        fifo_overflow <= 0;
175      end
176      else if (sample) begin
177            // order of arrival: 0,8 bits of data (LSB first),odd parity,1
178            if (count==10) begin
179               // just received what should be the stop bit
180               if (shift[0]==0 && ps2d==1 && (^shift[9:1])==1) begin
181         fifo[wptr] <= shift[8:1];
182         wptr <= wptr_inc;
183         fifo_overflow <= fifo_overflow | (wptr_inc == rptr);
184               end
185               count <= 0;
186        end else begin
187               shift <= {ps2d,shift[9:1]};
188               count <= count + 1;
189        end
```

```
190           end
191       // bump read pointer if we're done with current value.
192       // Read also resets the overflow indicator
193       if (fifo_rd && !fifo_empty) begin
194         rptr <= rptr + 1;
195         fifo_overflow <= 0;
196       end
197     end
198
199   endmodule
200
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    21:36:59 05/04/06
7   // Design Name:
8   // Module Name:    snakeseg
9   // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module snakeregs(clk,reset,movement,draw_en,grow,win,head_x,head_y,index,x,y,up,down,left,
    right,grow_idx,
22                   x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x2
    1,
23                x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,
24                y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26,y27,y28,y29,
    y30,y31,y32,bad_move);
25     input clk, reset, draw_en, grow, win;
26     input [3:0] movement;
27     input [4:0] index;
28     output [4:0] grow_idx;
29     output [9:0] head_x, head_y, x, y;
30     output up, down, left, right, bad_move;
31     output [9:0] x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21
    ,
32                 x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10
    ,
33                 y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26,y27,y28,y2
    9,y30,y31,y32;
34
35     reg [9:0] xpos[31:0];
36     reg [9:0] ypos[31:0];
37     reg [4:0] grow_idx;     //grow index
38     reg up;
39     reg down;
40     reg left;
41     reg right;
42     reg bad_move;
43
44     //assign bad_move = (up & movement[2]) | (down & movement[3]) | (left & movement[0]) |
45     //                (right & movement[1]) | lose;
46
47     assign head_x = xpos[0];
48     assign head_y = ypos[0];
49     assign x = xpos[index];
50     assign y = ypos[index];
51
52     assign x1 = xpos[0];
53     assign x2 = xpos[1];
54     assign x3 = xpos[2];
55     assign x4 = xpos[3];
56     assign x5 = xpos[4];
57     assign x6 = xpos[5];
```

```
58       assign x7 = xpos[6];
59       assign x8 = xpos[7];
60       assign x9 = xpos[8];
61       assign x10 = xpos[9];
62       assign x11 = xpos[10];
63       assign x12 = xpos[11];
64       assign x13 = xpos[12];
65       assign x14 = xpos[13];
66       assign x15 = xpos[14];
67       assign x16 = xpos[15];
68       assign x17 = xpos[16];
69       assign x18 = xpos[17];
70       assign x19 = xpos[18];
71       assign x20 = xpos[19];
72       assign x21 = xpos[20];
73       assign x22 = xpos[21];
74       assign x23 = xpos[22];
75       assign x24 = xpos[23];
76       assign x25 = xpos[24];
77       assign x26 = xpos[25];
78       assign x27 = xpos[26];
79       assign x28 = xpos[27];
80       assign x29 = xpos[28];
81       assign x30 = xpos[29];
82       assign x31 = xpos[30];
83       assign x32 = xpos[31];
84       assign y1 = ypos[0];
85       assign y2 = ypos[1];
86       assign y3 = ypos[2];
87       assign y4 = ypos[3];
88       assign y5 = ypos[4];
89       assign y6 = ypos[5];
90       assign y7 = ypos[6];
91       assign y8 = ypos[7];
92       assign y9 = ypos[8];
93       assign y10 = ypos[9];
94       assign y11 = ypos[10];
95       assign y12 = ypos[11];
96       assign y13 = ypos[12];
97       assign y14 = ypos[13];
98       assign y15 = ypos[14];
99       assign y16 = ypos[15];
100      assign y17 = ypos[16];
101      assign y18 = ypos[17];
102      assign y19 = ypos[18];
103      assign y20 = ypos[19];
104      assign y21 = ypos[20];
105      assign y22 = ypos[21];
106      assign y23 = ypos[22];
107      assign y24 = ypos[23];
108      assign y25 = ypos[24];
109      assign y26 = ypos[25];
110      assign y27 = ypos[26];
111      assign y28 = ypos[27];
112      assign y29 = ypos[28];
113      assign y30 = ypos[29];
114      assign y31 = ypos[30];
115      assign y32 = ypos[31];
116
117      always @ (posedge clk)
118         begin
119            if (reset | win)
120               begin
```

```
121                 xpos[0] <= 10'd380;
122                 xpos[1] <= 10'd380;
123                 xpos[2] <= 10'd380;
124                 xpos[3] <= 10'd380;
125                 xpos[4] <= 10'd380;
126                 xpos[5] <= 10'd380;
127                 xpos[6] <= 10'd380;
128                 xpos[7] <= 10'd380;
129                 xpos[8] <= 10'd380;
130                 xpos[9] <= 10'd380;
131                 xpos[10] <= 10'd380;
132                 xpos[11] <= 10'd380;
133                 xpos[12] <= 10'd380;
134                 xpos[13] <= 10'd380;
135                 xpos[14] <= 10'd380;
136                 xpos[15] <= 10'd380;
137                 xpos[16] <= 10'd380;
138                 xpos[17] <= 10'd380;
139                 xpos[18] <= 10'd380;
140                 xpos[19] <= 10'd380;
141                 xpos[20] <= 10'd380;
142                 xpos[21] <= 10'd380;
143                 xpos[22] <= 10'd380;
144                 xpos[23] <= 10'd380;
145                 xpos[24] <= 10'd380;
146                 xpos[25] <= 10'd380;
147                 xpos[26] <= 10'd380;
148                 xpos[27] <= 10'd360;
149                 xpos[28] <= 10'd340;
150                 xpos[29] <= 10'd320;
151                 xpos[30] <= 10'd300;
152                 xpos[31] <= 10'd280;
153                 ypos[0] <= 10'd200;
154                 ypos[1] <= 10'd200;
155                 ypos[2] <= 10'd200;
156                 ypos[3] <= 10'd200;
157                 ypos[4] <= 10'd200;
158                 ypos[5] <= 10'd200;
159                 ypos[6] <= 10'd200;
160                 ypos[7] <= 10'd200;
161                 ypos[8] <= 10'd200;
162                 ypos[9] <= 10'd200;
163                 ypos[10] <= 10'd200;
164                 ypos[11] <= 10'd200;
165                 ypos[12] <= 10'd200;
166                 ypos[13] <= 10'd200;
167                 ypos[14] <= 10'd200;
168                 ypos[15] <= 10'd200;
169                 ypos[16] <= 10'd200;
170                 ypos[17] <= 10'd200;
171                 ypos[18] <= 10'd200;
172                 ypos[19] <= 10'd200;
173                 ypos[20] <= 10'd200;
174                 ypos[21] <= 10'd200;
175                 ypos[22] <= 10'd200;
176                 ypos[23] <= 10'd200;
177                 ypos[24] <= 10'd200;
178                 ypos[25] <= 10'd200;
179                 ypos[26] <= 10'd200;
180                 ypos[27] <= 10'd200;
181                 ypos[28] <= 10'd200;
182                 ypos[29] <= 10'd200;
183                 ypos[30] <= 10'd200;
```

```
184                ypos[31] <= 10'd200;
185                grow_idx <= 5'd26;
186                up <= 0;
187                down <= 0;
188                left <= 0;
189                right <= 0;
190                bad_move <= 0;
191             end
192          else if(draw_en & ~grow)
193             begin
194                if((movement[3] & down) | (movement[2] & up) | (movement[1] & right) | (mov
    ement[0] & left) | bad_move)
195                   bad_move <= 1;
196                else if(movement[3] | movement[2] | movement[1] | movement[0])
197                begin
198                   if(movement[3])
199                      begin
200                         up <= 1;
201                         down <= 0;
202                         left <= 0;
203                         right <= 0;
204                      end
205                   else if(movement[2])
206                      begin
207                         up <= 0;
208                         down <= 1;
209                         left <= 0;
210                         right <= 0;
211                      end
212                   else if(movement[1])
213                      begin
214                         up <= 0;
215                         down <= 0;
216                         left <= 1;
217                         right <= 0;
218                      end
219                   else if(movement[0])
220                      begin
221                         up <= 0;
222                         down <= 0;
223                         left <= 0;
224                         right <= 1;
225                      end
226                xpos[0] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5 : xpos
    [0];
227                ypos[0] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5 : ypos
    [0];
228                if(grow_idx >= 5'd1)
229                   begin
230                      xpos[1] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
231                      ypos[1] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
232                   end
233                else
234                   begin
235                      xpos[1] <= xpos[0];
236                      ypos[1] <= ypos[0];
237                   end
238                if(grow_idx >= 5'd2)
239                   begin
240                      xpos[2] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
```

```
241                     ypos[2] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
242                  end
243               else
244                  begin
245                     xpos[2] <= xpos[1];
246                     ypos[2] <= ypos[1];
247                  end
248               if(grow_idx >= 5'd3)
249                  begin
250                     xpos[3] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
251                     ypos[3] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
252                  end
253               else
254                  begin
255                     xpos[3] <= xpos[2];
256                     ypos[3] <= ypos[2];
257                  end
258               if(grow_idx >= 5'd4)
259                  begin
260                     xpos[4] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
261                     ypos[4] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
262                  end
263               else
264                  begin
265                     xpos[4] <= xpos[3];
266                     ypos[4] <= ypos[3];
267                  end
268               if(grow_idx >= 5'd5)
269                  begin
270                     xpos[5] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
271                     ypos[5] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
272                  end
273               else
274                  begin
275                     xpos[5] <= xpos[4];
276                     ypos[5] <= ypos[4];
277                  end
278               if(grow_idx >= 5'd6)
279                  begin
280                     xpos[6] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
281                     ypos[6] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
282                  end
283               else
284                  begin
285                     xpos[6] <= xpos[5];
286                     ypos[6] <= ypos[5];
287                  end
288               if(grow_idx >= 5'd7)
289                  begin
290                     xpos[7] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
291                     ypos[7] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
292                  end
```

```
293                 else
294                    begin
295                       xpos[7] <= xpos[6];
296                       ypos[7] <= ypos[6];
297                    end
298                 if(grow_idx >= 5'd8)
299                    begin
300                       xpos[8] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
301                       ypos[8] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
302                    end
303                 else
304                    begin
305                       xpos[8] <= xpos[7];
306                       ypos[8] <= ypos[7];
307                    end
308                 if(grow_idx >= 5'd9)
309                    begin
310                       xpos[9] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
311                       ypos[9] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
312                    end
313                 else
314                    begin
315                       xpos[9] <= xpos[8];
316                       ypos[9] <= ypos[8];
317                    end
318                 if(grow_idx >= 5'd10)
319                    begin
320                       xpos[10] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
     : xpos[0];
321                       ypos[10] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
     : ypos[0];
322                    end
323                 else
324                    begin
325                       xpos[10] <= xpos[9];
326                       ypos[10] <= ypos[9];
327                    end
328                 if(grow_idx >= 5'd11)
329                    begin
330                       xpos[11] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
     : xpos[0];
331                       ypos[11] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
     : ypos[0];
332                    end
333                 else
334                    begin
335                       xpos[11] <= xpos[10];
336                       ypos[11] <= ypos[10];
337                    end
338                 if(grow_idx >= 5'd12)
339                    begin
340                       xpos[12] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
     : xpos[0];
341                       ypos[12] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
     : ypos[0];
342                    end
343                 else
344                    begin
345                       xpos[12] <= xpos[11];
```

```
346                     ypos[12] <= ypos[11];
347                  end
348               if(grow_idx >= 5'd13)
349                  begin
350                     xpos[13] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
 : xpos[0];
351                     ypos[13] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
352                  end
353               else
354                  begin
355                     xpos[13] <= xpos[12];
356                     ypos[13] <= ypos[12];
357                  end
358               if(grow_idx >= 5'd14)
359                  begin
360                     xpos[14] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
 : xpos[0];
361                     ypos[14] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
362                  end
363               else
364                  begin
365                     xpos[14] <= xpos[13];
366                     ypos[14] <= ypos[13];
367                  end
368               if(grow_idx >= 5'd15)
369                  begin
370                     xpos[15] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
 : xpos[0];
371                     ypos[15] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
372                  end
373               else
374                  begin
375                     xpos[15] <= xpos[14];
376                     ypos[15] <= ypos[14];
377                  end
378               if(grow_idx >= 5'd16)
379                  begin
380                     xpos[16] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
 : xpos[0];
381                     ypos[16] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
382                  end
383               else
384                  begin
385                     xpos[16] <= xpos[15];
386                     ypos[16] <= ypos[15];
387                  end
388               if(grow_idx >= 5'd17)
389                  begin
390                     xpos[17] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
 : xpos[0];
391                     ypos[17] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
392                  end
393               else
394                  begin
395                     xpos[17] <= xpos[16];
396                     ypos[17] <= ypos[16];
397                  end
398               if(grow_idx >= 5'd18)
```

```
399                     begin
400                        xpos[18] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
401                        ypos[18] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
402                     end
403                  else
404                     begin
405                        xpos[18] <= xpos[17];
406                        ypos[18] <= ypos[17];
407                     end
408                  if(grow_idx >= 5'd19)
409                     begin
410                        xpos[19] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
411                        ypos[19] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
412                     end
413                  else
414                     begin
415                        xpos[19] <= xpos[18];
416                        ypos[19] <= ypos[18];
417                     end
418                  if(grow_idx >= 5'd20)
419                     begin
420                        xpos[20] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
421                        ypos[20] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
422                     end
423                  else
424                     begin
425                        xpos[20] <= xpos[19];
426                        ypos[20] <= ypos[19];
427                     end
428                  if(grow_idx >= 5'd21)
429                     begin
430                        xpos[21] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
431                        ypos[21] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
432                     end
433                  else
434                     begin
435                        xpos[21] <= xpos[20];
436                        ypos[21] <= ypos[20];
437                     end
438                  if(grow_idx >= 5'd22)
439                     begin
440                        xpos[22] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
441                        ypos[22] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
    : ypos[0];
442                     end
443                  else
444                     begin
445                        xpos[22] <= xpos[21];
446                        ypos[22] <= ypos[21];
447                     end
448                  if(grow_idx >= 5'd23)
449                     begin
450                        xpos[23] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
    : xpos[0];
```

```
451                     ypos[23] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
452                  end
453               else
454                  begin
455                     xpos[23] <= xpos[22];
456                     ypos[23] <= ypos[22];
457                  end
458               if(grow_idx >= 5'd24)
459                  begin
460                     xpos[24] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
 : xpos[0];
461                     ypos[24] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
462                  end
463               else
464                  begin
465                     xpos[24] <= xpos[23];
466                     ypos[24] <= ypos[23];
467                  end
468               if(grow_idx >= 5'd25)
469                  begin
470                     xpos[25] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
 : xpos[0];
471                     ypos[25] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
472                  end
473               else
474                  begin
475                     xpos[25] <= xpos[24];
476                     ypos[25] <= ypos[24];
477                  end
478               if(grow_idx == 5'd26)
479                  begin
480                     xpos[26] <= (movement[1]) ? xpos[0] - 5 : (movement[0]) ? xpos[0] + 5
 : xpos[0];
481                     ypos[26] <= (movement[3]) ? ypos[0] - 5 : (movement[2]) ? ypos[0] + 5
 : ypos[0];
482                  end
483               else
484                  begin
485                     xpos[26] <= xpos[25];
486                     ypos[26] <= ypos[25];
487                  end
488               xpos[27] <= xpos[26];
489               ypos[27] <= ypos[26];
490               xpos[28] <= xpos[27];
491               ypos[28] <= ypos[27];
492               xpos[29] <= xpos[28];
493               ypos[29] <= ypos[28];
494               xpos[30] <= xpos[29];
495               ypos[30] <= ypos[29];
496               xpos[31] <= xpos[30];
497               ypos[31] <= ypos[30];
498               end
499            end
500         else if(draw_en && grow && (grow_idx > 0))
501            begin
502               xpos[0] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 10 : xp
os[0];
503               ypos[0] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 10 : yp
os[0];
504               if(grow_idx > 5'd1)
```

```
505                     begin
506                        xpos[1] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
507                        ypos[1] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
508                     end
509                  if(grow_idx > 5'd2)
510                     begin
511                        xpos[2] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
512                        ypos[2] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
513                     end
514                  if(grow_idx > 5'd3)
515                     begin
516                        xpos[3] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
517                        ypos[3] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
518                     end
519                  if(grow_idx > 5'd4)
520                     begin
521                        xpos[4] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
522                        ypos[4] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
523                     end
524                  if(grow_idx > 5'd5)
525                     begin
526                        xpos[5] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
527                        ypos[5] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
528                     end
529                  if(grow_idx > 5'd6)
530                     begin
531                        xpos[6] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
532                        ypos[6] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
533                     end
534                  if(grow_idx > 5'd7)
535                     begin
536                        xpos[7] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
537                        ypos[7] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
538                     end
539                  if(grow_idx > 5'd8)
540                     begin
541                        xpos[8] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
542                        ypos[8] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
543                     end
544                  if(grow_idx > 5'd9)
545                     begin
546                        xpos[9] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] + 1
    0 : xpos[0];
547                        ypos[9] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] + 1
    0 : ypos[0];
548                     end
549                  if(grow_idx > 5'd10)
```

```
550                     begin
551                        xpos[10] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
552                        ypos[10] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
553                     end
554                  if(grow_idx > 5'd11)
555                     begin
556                        xpos[11] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
557                        ypos[11] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
558                     end
559                  if(grow_idx > 5'd12)
560                     begin
561                        xpos[12] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
562                        ypos[12] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
563                     end
564                  if(grow_idx > 5'd13)
565                     begin
566                        xpos[13] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
567                        ypos[13] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
568                     end
569                  if(grow_idx > 5'd14)
570                     begin
571                        xpos[14] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
572                        ypos[14] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
573                     end
574                  if(grow_idx > 5'd15)
575                     begin
576                        xpos[15] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
577                        ypos[15] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
578                     end
579                  if(grow_idx > 5'd16)
580                     begin
581                        xpos[16] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
582                        ypos[16] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
583                     end
584                  if(grow_idx > 5'd17)
585                     begin
586                        xpos[17] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
587                        ypos[17] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
588                     end
589                  if(grow_idx > 5'd18)
590                     begin
591                        xpos[18] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
592                        ypos[18] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
593                     end
594                  if(grow_idx > 5'd19)
```

```
595                     begin
596                        xpos[19] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
597                        ypos[19] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
598                     end
599                  if(grow_idx > 5'd20)
600                     begin
601                        xpos[20] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
602                        ypos[20] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
603                     end
604                  if(grow_idx > 5'd21)
605                     begin
606                        xpos[21] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
607                        ypos[21] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
608                     end
609                  if(grow_idx > 5'd22)
610                     begin
611                        xpos[22] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
612                        ypos[22] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
613                     end
614                  if(grow_idx > 5'd23)
615                     begin
616                        xpos[23] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
617                        ypos[23] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
618                     end
619                  if(grow_idx > 5'd24)
620                     begin
621                        xpos[24] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
622                        ypos[24] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
623                     end
624                  if(grow_idx > 5'd25)
625                     begin
626                        xpos[25] <= (movement[1]) ? xpos[0] - 10 : (movement[0]) ? xpos[0] +
    10 : xpos[0];
627                        ypos[25] <= (movement[3]) ? ypos[0] - 10 : (movement[2]) ? ypos[0] +
    10 : ypos[0];
628                     end
629                  xpos[26] <= xpos[26];
630                  ypos[26] <= ypos[26];
631                  xpos[27] <= xpos[27];
632                  ypos[27] <= ypos[27];
633                  xpos[28] <= xpos[28];
634                  ypos[28] <= ypos[28];
635                  xpos[29] <= xpos[29];
636                  ypos[29] <= ypos[29];
637                  xpos[30] <= xpos[30];
638                  ypos[30] <= ypos[30];
639                  xpos[31] <= xpos[31];
640                  ypos[31] <= ypos[31];
641                  grow_idx <= grow_idx - 1;
642               end
643            else
```

```
644             begin
645                xpos[0] <= xpos[0];
646                ypos[0] <= ypos[0];
647                xpos[1] <= xpos[1];
648                ypos[1] <= ypos[1];
649                xpos[2] <= xpos[2];
650                ypos[2] <= ypos[2];
651                xpos[3] <= xpos[3];
652                ypos[3] <= ypos[3];
653                xpos[4] <= xpos[4];
654                ypos[4] <= ypos[4];
655                xpos[5] <= xpos[5];
656                ypos[5] <= ypos[5];
657                xpos[6] <= xpos[6];
658                ypos[6] <= ypos[6];
659                xpos[7] <= xpos[7];
660                ypos[7] <= ypos[7];
661                xpos[8] <= xpos[8];
662                ypos[8] <= ypos[8];
663                xpos[9] <= xpos[9];
664                ypos[9] <= ypos[9];
665                xpos[10] <= xpos[10];
666                ypos[10] <= ypos[10];
667                xpos[11] <= xpos[11];
668                ypos[11] <= ypos[11];
669                xpos[12] <= xpos[12];
670                ypos[12] <= ypos[12];
671                xpos[13] <= xpos[13];
672                ypos[13] <= ypos[13];
673                xpos[14] <= xpos[14];
674                ypos[14] <= ypos[14];
675                xpos[15] <= xpos[15];
676                ypos[15] <= ypos[15];
677                xpos[16] <= xpos[16];
678                ypos[16] <= ypos[16];
679                xpos[17] <= xpos[17];
680                ypos[17] <= ypos[17];
681                xpos[18] <= xpos[18];
682                ypos[18] <= ypos[18];
683                xpos[19] <= xpos[19];
684                ypos[19] <= ypos[19];
685                xpos[20] <= xpos[20];
686                ypos[20] <= ypos[20];
687                xpos[21] <= xpos[21];
688                ypos[21] <= ypos[21];
689                xpos[22] <= xpos[22];
690                ypos[22] <= ypos[22];
691                xpos[23] <= xpos[23];
692                ypos[23] <= ypos[23];
693                xpos[24] <= xpos[24];
694                ypos[24] <= ypos[24];
695                xpos[25] <= xpos[25];
696                ypos[25] <= ypos[25];
697                xpos[26] <= xpos[26];
698                ypos[26] <= ypos[26];
699                xpos[27] <= xpos[27];
700                ypos[27] <= ypos[27];
701                xpos[28] <= xpos[28];
702                ypos[28] <= ypos[28];
703                xpos[29] <= xpos[29];
704                ypos[29] <= ypos[29];
705                xpos[30] <= xpos[30];
706                ypos[30] <= ypos[30];
```

```
707                 xpos[31] <= xpos[31];
708                 ypos[31] <= ypos[31];
709             end
710       end
711
712   endmodule
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    20:30:27 05/03/06
7   // Design Name:
8   // Module Name:    snakes
9   // Project Name:
10  // Target Device:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module snakesprite(clk,x,y,tar_x,tar_y,RGB);
22
23     input clk;
24     input [9:0] x, y;
25     input [9:0] tar_x, tar_y;
26     output [23:0] RGB;
27
28     reg [23:0] RGB;
29
30     parameter WIDTH = 20;
31     parameter HEIGHT = 20;
32     parameter COLOR = 24'h00FF00;
33
34     always @ (posedge clk)
35        begin
36           if((((x > tar_x) || (x == tar_x)) && (x < tar_x+WIDTH)) &&
37              (((y > tar_y) || (y == tar_y)) && (y < tar_y+HEIGHT)))
38              RGB <= COLOR;
39           else
40              RGB <= 24'h0;
41        end
42
43  endmodule
44
```

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    13:32:17 03/24/06
7  // Design Name:
8  // Module Name:    vga
9  // Project Name:
10 // Target Device:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module vga (pixel_clock, reset, hsync, vsync, sync_b, blank_b, pixel_count, line_count);
22    input pixel_clock; // 31.5 MHz pixel clock
23    input reset; // system reset
24    output hsync; // horizontal sync
25    output vsync; // vertical sync
26    output sync_b; // hardwired to Vdd
27    output blank_b; // composite blank
28    output [9:0] pixel_count; // number of the current pixel
29    output [9:0] line_count; // number of the current line
30
31    // 640x480 75Hz parameters
32    parameter PIXELS = 800;
33    parameter LINES = 525;
34    parameter HACTIVE_VIDEO = 640;
35    parameter HFRONT_PORCH = 16;
36    parameter HSYNC_PERIOD = 96;
37    parameter HBACK_PORCH = 48;
38    parameter VACTIVE_VIDEO = 480;
39    parameter VFRONT_PORCH = 11;
40    parameter VSYNC_PERIOD = 2;
41    parameter VBACK_PORCH = 32;
42
43    // current pixel count
44    reg [9:0] pixel_count = 10'b0;
45    reg [9:0] line_count = 10'b0;
46
47    // registered outputs
48    reg hsync = 1'b1;
49    reg vsync = 1'b1;
50    reg blank_b = 1'b1;
51    wire sync_b; // connected to Vdd
52    wire pixel_clock;
53    wire [9:0] next_pixel_count;
54    wire [9:0] next_line_count;
55
56    always @ (posedge pixel_clock)
57       begin
58          if (reset)
59             begin
60                pixel_count <= 10'b0;
61                line_count <= 10'b0;
62                hsync <= 1'b1;
63                vsync <= 1'b1;
```

```
64                blank_b <= 1'b1;
65             end
66          else
67             begin
68                pixel_count <= next_pixel_count;
69                line_count <= next_line_count;
70                hsync <= (next_pixel_count < HACTIVE_VIDEO + HFRONT_PORCH) |
71                         (next_pixel_count >= HACTIVE_VIDEO+HFRONT_PORCH+HSYNC_PERIOD);
72                vsync <= (next_line_count < VACTIVE_VIDEO+VFRONT_PORCH) |
73                         (next_line_count >= VACTIVE_VIDEO+VFRONT_PORCH+VSYNC_PERIOD);
74
75                // this is the and of hblank and vblank
76                blank_b <= (next_pixel_count < HACTIVE_VIDEO) & (next_line_count < VACTIVE_
   VIDEO);
77             end
78       end
79
80    // next state is computed with combinational logic
81    assign next_pixel_count = (pixel_count == PIXELS-1) ? 10'h000 : pixel_count + 1'b1;
82    assign next_line_count = (pixel_count == PIXELS-1) ? (line_count == LINES-1) ? 10'h000
   :
83                                line_count + 1'b1 : line_count;
84
85    // since we are providing hsync and vsync to the display, we
86    // can hardwire composite sync to Vdd.
87    assign sync_b = 1'b1;
88
89 endmodule
```