

```

1  `timescale 1ns / 1ns
2
3  module drawSquare
4  (
5      CLOCK_50,                // On Board 50 MHz
6      // Your inputs and outputs here
7      KEY,                     // On Board Keys
8      SW,
9      // The ports below are for the VGA output. Do not change.
10     VGA_CLK,                 // VGA Clock
11     VGA_HS,                  // VGA H_SYNC
12     VGA_VS,                  // VGA V_SYNC
13     VGA_BLANK_N,             // VGA BLANK
14     VGA_SYNC_N,              // VGA SYNC
15     VGA_R,                   // VGA Red[9:0]
16     VGA_G,                   // VGA Green[9:0]
17     VGA_B                    // VGA Blue[9:0]
18 );
19
20 input          CLOCK_50;      // 50 MHz
21 input [3:0]    KEY;
22 // Declare your inputs and outputs here
23 input [9:0]    SW;
24 // Do not change the following outputs
25 output        VGA_CLK;       // VGA Clock
26 output        VGA_HS;        // VGA H_SYNC
27 output        VGA_VS;        // VGA V_SYNC
28 output        VGA_BLANK_N;    // VGA BLANK
29 output        VGA_SYNC_N;     // VGA SYNC
30 output [7:0]   VGA_R;         // VGA Red[7:0] Changed from 10 to 8-bit DAC
31 output [7:0]   VGA_G;         // VGA Green[7:0]
32 output [7:0]   VGA_B;         // VGA Blue[7:0]
33
34 wire resetn;
35 assign resetn = KEY[0]; // Active low, so don't invert
36
37 // Create the colour, x, y and writeEn wires that are inputs to the controller.
38
39 wire [2:0] colour;
40 wire [7:0] x;
41 wire [6:0] y;
42 wire writeEn;
43
44 // Create an Instance of a VGA controller - there can be only one!
45 // Define the number of colours as well as the initial background
46 // image file (.MIF) for the controller.
47 vga_adapter VGA(
48     .resetn(resetn),
49     .clock(CLOCK_50),
50     .colour(colour),
51     .x(x),
52     .y(y),
53     .plot(writeEn),
54     /* Signals for the DAC to drive the monitor. */
55     .VGA_R(VGA_R),
56     .VGA_G(VGA_G),
57     .VGA_B(VGA_B),
58     .VGA_HS(VGA_HS),
59     .VGA_VS(VGA_VS),
60     .VGA_BLANK(VGA_BLANK_N),
61     .VGA_SYNC(VGA_SYNC_N),
62     .VGA_CLK(VGA_CLK));
63 defparam VGA.RESOLUTION = "160x120";
64 defparam VGA.MONOCHROME = "FALSE";
65 defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
66 defparam VGA.BACKGROUND_IMAGE = "black.mif";
67
68 // Put your code here. Your code should produce signals x,y,colour and writeEn
69 // for the VGA controller, in addition to any other functionality your design may

```

```

70     require.
71     // Input wires
72     wire[6:0] DATA_IN;
73     wire[2:0] C_DATA;
74     wire go,clear,plot;
75
76     // Control wires
77     wire ld_x,ld_y,ld_colour;
78     wire ld_xpos,ld_ypos;
79     wire set_black,draw_pixel;
80     wire[7:0] dx,dy;
81
82     // Assign inputs
83     assign plot      = ~KEY[1];
84     assign go        = ~KEY[2];
85     assign clear     = ~KEY[3];
86     assign DATA_IN = SW[6:0];
87     assign C_DATA    = SW[9:7];
88
89     // Control module
90     control c0(
91         .clock(CLOCK_50),.resets(resets),
92         .go(go),.plot(plot),.clear(clear),
93         .ld_x(ld_x),.ld_y(ld_y),.ld_colour(ld_colour),
94         .ld_xpos(ld_xpos),.ld_ypos(ld_ypos),
95         .set_black(set_black),.draw_pixel(draw_pixel),
96         .dx_out(dx),.dy_out(dy)
97     );
98
99     // Controls plotting on VGA
100    assign writeEn = draw_pixel;
101
102    // Datapath module
103    datapath d0(
104        .clock(CLOCK_50),.resets(resets),
105        .DATA_IN(DATA_IN),.COLOUR_DATA(C_DATA),
106        .ld_x(ld_x),.ld_y(ld_y),.ld_colour(ld_colour),
107        .ld_xpos(ld_xpos),.ld_ypos(ld_ypos),
108        .set_black(set_black),.dx(dx),.dy(dy),
109        .xpos(x),.ypos(y),.colour(colour)
110    );
111    endmodule
112
113    // Tracks state and datapath control signals depending on state
114    module control(
115        input clock,resets,
116        input go,plot,clear,
117        output reg ld_x,ld_y,ld_colour,
118        output reg ld_xpos,ld_ypos,
119        output reg set_black,draw_pixel,
120        output reg[7:0] dx,dy
121    );
122
123    // Parameters for counters
124    localparam X_SHAPE      = 8'd4,
125               Y_SHAPE      = 8'd4,
126               X_SCREEN     = 8'd160,
127               Y_SCREEN     = 8'd120;
128
129    // Keeps track of state
130    reg[3:0] current_state,next_state;
131    // dx and dy are counters that count up to x and y size
132    // Used to draw shapes and clear screen
133    reg[7:0] x_size,y_size;
134    reg reset_dx,reset_dy,inc_dx,inc_dy;
135    reg set_size,size_select;
136
137    // Assigning state variables

```

```

138 localparam S_LOAD_X = 4'd0,
139          S_LOAD_X_WAIT = 4'd1,
140          S_LOAD_Y = 4'd2,
141          S_LOAD_Y_WAIT = 4'd3,
142          S_LOAD_COLOUR = 4'd4,
143          S_LOAD_COLOUR_WAIT = 4'd5,
144          S_CLEAR_SCREEN = 4'd6,
145          S_CYCLE_0 = 4'd7,
146          S_CYCLE_1 = 4'd8,
147          S_CYCLE_2 = 4'd9,
148          S_CYCLE_3 = 4'd10,
149          S_CYCLE_4 = 4'd11,
150          S_CYCLE_5 = 4'd12;
151
152 // Counter registers for dx and dy
153 always @(posedge clock)
154 begin
155     if(!resetsn) // Active low reset
156     begin
157         dx <= 8'b0;
158         dy <= 8'b0;
159     end
160     else
161     begin
162         // dx counter
163         if(reset_dx)
164             dx <= 8'b0;
165         else if(inc_dx)
166             dx <= dx+1;
167
168         // dy counter
169         if(reset_dy)
170             dy <= 8'b0;
171         else if(inc_dy)
172             dy <= dy+1;
173     end
174 end
175
176 // Registers for x and y size
177 always @(posedge clock)
178 begin
179     if(!resetsn) // Active low reset to shape size
180     begin
181         x_size <= X_SHAPE;
182         y_size <= Y_SHAPE;
183     end
184     else if(set_size) // Set the size to selected size
185     begin
186         x_size <= size_select ? X_SHAPE:X_SCREEN;
187         y_size <= size_select ? Y_SHAPE:Y_SCREEN;
188     end
189 end
190
191 // State table
192 always @(*)
193 begin
194     case(current_state)
195         S_LOAD_X: // Loop in state until value is input
196             next_state = go ? S_LOAD_X_WAIT:S_LOAD_X;
197         S_LOAD_X_WAIT: // Loop in state until go signal goes low
198             next_state = go ? S_LOAD_X_WAIT:S_LOAD_Y;
199         S_LOAD_Y: // Loop in state until value is input
200             next_state = go ? S_LOAD_Y_WAIT:S_LOAD_Y;
201         S_LOAD_Y_WAIT: // Loop in state until go signal goes low
202             next_state = go ? S_LOAD_Y_WAIT:S_LOAD_COLOUR;
203         S_LOAD_COLOUR: // Loop in state until value is input
204             next_state = go ? S_LOAD_COLOUR_WAIT:S_LOAD_COLOUR;
205         S_LOAD_COLOUR_WAIT: // Loop until go goes low and plot goes high
206             next_state = (go | !plot) ? S_LOAD_COLOUR_WAIT:S_CYCLE_0;

```

```

207     S_CLEAR_SCREEN: // Clears (x and y size = screen size), reset dy
208         next_state = S_CYCLE_1;
209     S_CYCLE_0: // Sets x and y size to shape size, reset dy
210         next_state = S_CYCLE_1;
211     S_CYCLE_1: // Reset dx
212         next_state = S_CYCLE_2;
213     S_CYCLE_2: // Set actual x and y positions
214         next_state = S_CYCLE_3;
215     S_CYCLE_3: // Fill pixel with colour
216         next_state = S_CYCLE_4;
217     S_CYCLE_4: // Increment dx
218         next_state = (dx==x_size-1) ? S_CYCLE_5:S_CYCLE_2;
219     S_CYCLE_5: // Increment dy
220         next_state = (dy==y_size-1) ? S_LOAD_X:S_CYCLE_1;
221     default: next_state = S_LOAD_X;
222 endcase
223 end
224
225 // Changing data control signals
226 always @(*)
227 begin
228     // Initializing signals to 0 to avoid latches
229     // Internal controls
230     reset_dx = 0; reset_dy = 0; inc_dx = 0; inc_dy = 0;
231     set_size = 0; size_select = 0;
232     // External controls
233     ld_x = 0; ld_y = 0; ld_colour = 0;
234     ld_xpos = 0; ld_ypos = 0;
235     set_black = 0; draw_pixel = 0;
236
237     case(current_state)
238     S_LOAD_X: // Load x
239     begin
240         ld_x = 1;
241     end
242     S_LOAD_Y: // Load y
243     begin
244         ld_y = 1;
245     end
246     S_LOAD_COLOUR: // Load colour
247     begin
248         ld_colour = 1;
249     end
250     S_CLEAR_SCREEN: // Clears (x and y size = screen size), reset dy
251     begin
252         reset_dy = 1;
253         set_size = 1; size_select = 0; // Screen size
254         set_black = 1;
255     end
256     S_CYCLE_0: // Sets x and y size to shape size, reset dy
257     begin
258         reset_dy = 1;
259         set_size = 1; size_select = 1; // Shape size
260     end
261     S_CYCLE_1: // Reset dx
262     begin
263         reset_dx = 1;
264     end
265     S_CYCLE_2: // Set actual x and y positions
266     begin
267         ld_xpos = 1;
268         ld_ypos = 1;
269     end
270     S_CYCLE_3: // Fill pixel with colour
271     begin
272         draw_pixel = 1;
273     end
274     S_CYCLE_4: // Increment dx
275     begin

```

```

276         inc_dx = 1;
277     end
278     S_CYCLE_5: // Increment dy
279     begin
280         inc_dy = 1;
281     end
282 endcase
283 end
284
285 // Register for current state
286 always @(posedge clock)
287 begin
288     if(!resetsn) // Reset to value input, active low
289         current_state <= S_LOAD_X;
290     else if(clear)
291         current_state <= S_CLEAR_SCREEN;
292     else // Load next state
293         current_state <= next_state;
294     end
295 endmodule
296
297 // Modifies data and outputs depending on control signals
298 module datapath(
299     input clock, resetsn,
300     input[6:0] DATA_IN,
301     input[2:0] COLOUR_DATA,
302     input ld_x, ld_y, ld_colour,
303     input ld_xpos, ld_ypos,
304     input set_black,
305     input[7:0] dx, dy,
306     output reg[7:0] xpos,
307     output reg[6:0] ypos,
308     output reg[2:0] colour
309 );
310
311 // Internal registers
312 reg[7:0] x, y;
313
314 // Registers x, y, xpos, ypos and colour with input logic
315 always @(posedge clock)
316 begin
317     if(!resetsn) // Active low reset
318     begin
319         x <= 8'b0;
320         y <= 8'b0;
321         xpos <= 8'b0;
322         ypos <= 7'b0;
323         colour <= 3'b0;
324     end
325     else
326     begin
327         // x register
328         if(ld_x)
329             x <= {1'b0, DATA_IN};
330         else if(set_black)
331             x <= 8'b0;
332         // y register
333         if(ld_y)
334             y <= {1'b0, DATA_IN};
335         else if(set_black)
336             y <= 8'b0;
337         // xpos register
338         if(ld_xpos)
339             xpos <= x+dx;
340         // ypos register
341         if(ld_ypos)
342             ypos <= y[6:0]+dy[6:0];
343         // colour register
344         if(ld_colour)

```

```
345         colour <= COLOUR_DATA;
346     else if(set_black)
347         colour <= 3'b0;
348     end
349 end
350 endmodule
351
```