```
1   // Top-level module that defines the I/Os for the DE-1 SoC + LED expansion boards
2
3   module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, GPIO_1);
4       input logic CLOCK_50; // 50MHz clock.
5       output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
6       output logic [9:0] LEDR;
7       input logic [3:0] KEY; // True when not pressed, False when pressed
8       input logic [9:0] SW;
9       output logic [35:0] GPIO_1;
10
11      logic reset, start_game;
12      assign reset = SW[9];
13      assign start_game = SW[8];
14
15      // Slower clock for Game of Life
16      logic [31:0] clk;
17      parameter slow = 6;
18      clock_divider cdiv (.clock(CLOCK_50), .reset(reset), .divided_clocks(clk));
19
20      // Game of Life modules and signals
21      logic [15:0][15:0] RedPixels, GrnPixels, RedPixels_next;
22      logic [7:0] row_select, col_select;
23      logic cell_state, set_initial, game_running;
24      logic [3:0] pattern_count;
25
26      // Instantiate the interface for handling user configs to the startpos of GoL
27      userInput user_interface (
28          .clk(clk[slow]),
29          .reset(reset),
30          .KEY(KEY),
31          .SW0(SW[0]),
32          .start_game(start_game),
33          .row_select(row_select),
34          .col_select(col_select),
35          .set_initial(set_initial),
36          .cell_state(cell_state),
37          .GrnPixels(GrnPixels)
38      );
39
40      // Instantiate the Grid Module
41      grid grid_inst (
42          .clk(clk[slow]),
43          .reset(reset),
44          .row_select(row_select),
45          .col_select(col_select),
46          .set_initial(set_initial),
47          .new_state(cell_state),
48          .enable_update(game_running),
49          .grid(RedPixels),
50          .grid_next(RedPixels_next),
51          .pattern_count(pattern_count)
52      );
53
54      // Update Logic (when the game actually runs, after start signal is given, start cell
    config is done)
55      updateLogic game_of_life (
56          .clk(clk[slow]),
57          .reset(reset),
58          .grid(RedPixels),
59          .grid_next(RedPixels_next)
60      );
61
62      // Standard LED Driver instantiation (directly from LED driver tutorial)
63      LEDDriver Driver (.CLK(clk[slow]), .RST(reset), .EnableCount(1'b1), .RedPixels(RedPixels
    ), .GrnPixels(GrnPixels), .GPIO_1(GPIO_1));
64
65      // Control Unit
66      controlUnit control_unit (
67          .clk(clk[slow]),
68          .reset(reset),
69          .start(start_game),
70          .enable_update(game_running)
71      );
```

```
72
73      // Display pattern count on HEX4 and HEX5
74      patternCounterDisplay counter_display (
75          .pattern_count(pattern_count),
76          .HEX4(HEX4),
77          .HEX5(HEX5)
78      );
79
80      // Turn off unused HEX displays
81      assign HEX0 = '1;
82      assign HEX1 = '1;
83      assign HEX2 = '1;
84      assign HEX3 = '1;
85  endmodule
86
87
88
89
90
91
92  module DE1_SoC_testbench();
93     logic clk, reset, start_game;
94     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
95     logic [9:0] LEDR;
96     logic [9:0] SW;
97     logic [35:0] GPIO_1;
98     logic [3:0] KEY;
99     //logic SW0;
100    logic [7:0] row_select;
101    logic [7:0] col_select;
102    logic set_initial;
103    logic cell_state;
104    logic [15:0][15:0] RedPixels, RedPixels_next, GrnPixels, grid_initial;
105    logic [3:0] pattern_count;
106
107    DE1_SoC dut (clk, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, GPIO_1);
108
109    // Set up a simulated clock.
110    parameter CLOCK_PERIOD = 100;
111    initial begin
112       clk <= 0;
113       forever #(CLOCK_PERIOD/2) clk <= ~clk; // Forever toggle the clock
114    end
115
116     // Test the design.
117     initial begin
118         reset <= 1;          @(posedge clk); // reset every time we start
119         @(posedge clk);
120         reset <= 0;          @(posedge clk);
121         @(posedge clk);
122
123         // Configure Pattern 1
124         KEY <= 4'b1111; SW[0] <= 1;
125         row_select <= 8'd8; col_select <= 8'd8; @(posedge clk); @(posedge clk); // Set cell
126         KEY[3] <= 1'b0; @(posedge clk); @(posedge clk); // Move left
127         SW[0] <= 0; @(posedge clk); @(posedge clk); // Clear cell
128         KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
129         SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
130         KEY[1] <= 1'b0; @(posedge clk); @(posedge clk); // Move down
131         SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
132
133         // Start game for Pattern 1
134         SW[8] <= 1; @(posedge clk); @(posedge clk);
135         SW[8] <= 0; @(posedge clk); @(posedge clk);
136
137         // Let the game run for a few cycles
138         repeat (5) @(posedge clk);
139
140         // Reset for next pattern
141         reset <= 1; @(posedge clk); reset <= 0; @(posedge clk);
142
143         // Configure Pattern 2
144         KEY <= 4'b1111; SW[0] <= 1;
```

```
145             row_select <= 8'd8; col_select <= 8'd8; @(posedge clk); @(posedge clk); // Set cell
146             KEY <= 4'b1011; @(posedge clk); @(posedge clk); // Move left
147             SW[0] <= 0; @(posedge clk); @(posedge clk); // Clear cell
148             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
149             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
150             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
151             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
152
153             // Start game for Pattern 2
154             SW[8] <= 1; @(posedge clk); @(posedge clk);
155             SW[8] <= 0; @(posedge clk); @(posedge clk);
156
157             // Let the game run for a few cycles
158             repeat (5) @(posedge clk);
159
160             // Reset for next pattern
161             reset <= 1; @(posedge clk); reset <= 0; @(posedge clk);
162
163             // Configure Pattern 3
164             KEY <= 4'b1111; SW[0] <= 1;
165             row_select <= 8'd8; col_select <= 8'd8; @(posedge clk); @(posedge clk); // Set cell
166             KEY <= 4'b1011; @(posedge clk); @(posedge clk); // Move left
167             SW[0] <= 0; @(posedge clk); @(posedge clk); // Clear cell
168             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
169             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
170             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
171             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
172
173             // Start game for Pattern 3
174             SW[8] <= 1; @(posedge clk); @(posedge clk);
175             SW[8] <= 0; @(posedge clk); @(posedge clk);
176
177             // Let the game run for a few cycles
178             repeat (5) @(posedge clk);
179
180             // Reset for next pattern
181             reset <= 1; @(posedge clk); reset <= 0; @(posedge clk);
182
183             // Configure Pattern 4
184             KEY <= 4'b1111; SW[0] <= 1;
185             row_select <= 8'd8; col_select <= 8'd8; @(posedge clk); @(posedge clk); // Set cell
186             KEY <= 4'b1011; @(posedge clk); @(posedge clk); // Move left
187             SW[0] <= 0; @(posedge clk); @(posedge clk); // Clear cell
188             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
189             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
190             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
191             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
192
193             // Start game for Pattern 4
194             SW[8] <= 1; @(posedge clk); @(posedge clk);
195             SW[8] <= 0; @(posedge clk); @(posedge clk);
196
197             // Let the game run for a few cycles
198             repeat (5) @(posedge clk);
199
200             // Test Pattern 1: Vertical Blinker
201             // Initial configuration for vertical blinker
202             KEY <= 4'b1111; SW[0] <= 1;
203             row_select <= 8'd7; col_select <= 8'd8; @(posedge clk); @(posedge clk); // Set cell
204             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
205             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
206             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
207             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
208
209             // Start game for Pattern 1
210             SW[8] <= 1; @(posedge clk); @(posedge clk);
211             SW[8] <= 0; @(posedge clk); @(posedge clk);
212
213             // Let the game run for a few cycles
214             repeat (5) @(posedge clk);
215
216
217             // Reset for next pattern
```

```
218             reset <= 1; @(posedge clk); reset <= 0; @(posedge clk);
219
220             // Test Pattern 2: Horizontal Blinker
221             // Initial configuration for horizontal blinker
222             KEY <= 4'b1111; SW[0] <= 1;
223             row_select <= 8'd8; col_select <= 8'd7; @(posedge clk); @(posedge clk); // Set cell
224             KEY <= 4'b1011; @(posedge clk); @(posedge clk); // Move right
225             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
226             KEY <= 4'b1011; @(posedge clk); @(posedge clk); // Move right
227             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
228
229             // Start game for Pattern 2
230             SW[8] <= 1; @(posedge clk); @(posedge clk);
231             SW[8] <= 0; @(posedge clk); @(posedge clk);
232
233             // Let the game run for a few cycles
234             repeat (5) @(posedge clk);
235
236
237             // Reset for next pattern
238             reset <= 1; @(posedge clk); reset <= 0; @(posedge clk);
239
240             // Test Pattern 3: Both Vertical and Horizontal Blinker Oscillations
241             // Initial configuration for both vertical and horizontal blinker oscillations
242             KEY <= 4'b1111; SW[0] <= 1;
243             row_select <= 8'd7; col_select <= 8'd8; @(posedge clk); @(posedge clk); // Set cell
244             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
245             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
246             KEY <= 4'b1101; @(posedge clk); @(posedge clk); // Move down
247             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
248
249             KEY <= 4'b1111; SW[0] <= 1;
250             row_select <= 8'd8; col_select <= 8'd7; @(posedge clk); @(posedge clk); // Set cell
251             KEY <= 4'b1011; @(posedge clk); @(posedge clk); // Move right
252             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
253             KEY <= 4'b1011; @(posedge clk); @(posedge clk); // Move right
254             SW[0] <= 1; @(posedge clk); @(posedge clk); // Set cell
255
256             // Start game for Pattern 3
257             SW[8] <= 1; @(posedge clk); @(posedge clk);
258             SW[8] <= 0; @(posedge clk); @(posedge clk);
259
260             // Let the game run for a few cycles
261             repeat (10) @(posedge clk);
262
263             $stop;
264         end
265     endmodule
266
```