```
1   // Implements rules of the GoL, helps to update the grid to grid_next after each clk cycle
2   // Is called when SW[8] i.e. start_game is pushed to 1
3
4   module updateLogic (
5       input logic clk,
6       input logic reset,
7       input logic [15:0][15:0] grid,
8       output logic [15:0][15:0] grid_next
9   );
10      logic [2:0] num_neighbors;
11      logic current_state;
12      logic [4:0] neighbor_row, neighbor_col;
13
14      always_ff @(posedge clk or posedge reset) begin
15          if (reset) begin
16              grid_next <= '{default: 0};
17          end else begin
18              for (int row = 0; row < 16; row++) begin
19                  for (int col = 0; col < 16; col++) begin
20                      num_neighbors = 0;
21                      for (int i = -1; i <= 1; i++) begin
22                          for (int j = -1; j <= 1; j++) begin
23                              if (i != 0 || j != 0) begin
24                                  neighbor_row = row + i;
25                                  neighbor_col = col + j;
26                                  if (neighbor_row >= 0 && neighbor_row < 16 && neighbor_col
    >= 0 && neighbor_col < 16) begin
27                                      num_neighbors += grid[neighbor_row][neighbor_col];
28                                  end
29                              end
30                          end
31                      end
32                      current_state = grid[row][col];
33                      if (current_state == 1) begin
34                          if (num_neighbors < 2 || num_neighbors > 3) begin
35                              grid_next[row][col] = 0;
36                          end else begin
37                              grid_next[row][col] = 1;
38                          end
39                      end else begin
40                          if (num_neighbors == 3) begin
41                              grid_next[row][col] = 1;
42                          end else begin
43                              grid_next[row][col] = 0;
44                          end
45                      end
46                  end
47              end
48          end
49      end
50  endmodule
51
52
53
54
55  module updateLogic_testbench();
56      logic clk, reset;
57      logic [15:0][15:0] grid;
58      logic [15:0][15:0] grid_next;
59
60      updateLogic dut (clk, reset, grid, grid_next);
61
62      // Set up a simulated clock.
63      parameter CLOCK_PERIOD = 100;
64      initial begin
65          clk <= 0;
66          forever #(CLOCK_PERIOD/2) clk <= ~clk; // Forever toggle the clock
67      end
68
69      // Test the design.
70      initial begin
71          // Reset the design
72          reset <= 1;          @(posedge clk); // reset every time we start
```

```
73            @(posedge clk);
74            reset <= 0;          @(posedge clk);
75            @(posedge clk);
76
77            // Test case 1: Initial grid state
78            grid = '{default: 0};
79            grid[8][7] = 1; grid[8][8] = 1; grid[8][9] = 1; // Horizontal blinker
80            @(posedge clk); @(posedge clk);
81
82            // Test case 2: Next grid state
83            grid = grid_next;
84            @(posedge clk); @(posedge clk);
85
86            // Test case 3: Another next grid state
87            grid = grid_next;
88            @(posedge clk); @(posedge clk);
89
90            // Test case 4: Reset and new pattern
91            reset <= 1; @(posedge clk); reset <= 0; @(posedge clk);
92            grid = '{default: 0};
93            grid[7][8] = 1; grid[8][8] = 1; grid[9][8] = 1; // Vertical blinker
94            @(posedge clk); @(posedge clk);
95
96            // Test case 5: Next grid state
97            grid = grid_next;
98            @(posedge clk); @(posedge clk);
99
100           // Test case 6: Another next grid state
101           grid = grid_next;
102           @(posedge clk); @(posedge clk);
103
104           $stop;
105       end
106   endmodule
107
```