

Lab Code [0 points]  
Filename: Ball.sv

```
1 module Ball
2     (input logic clock, reset, update,
3      input logic [8:0] left_paddle_top, right_paddle_top,
4      input logic [8:0] row,
5      input logic [9:0] col,
6      output logic left_scored, right_scored,
7      output logic display);
8
9     logic x_dir;
10    logic [1:0] y_dir;
11    logic [9:0] left, top;
12
13    always_ff@(posedge clock) begin
14        if (reset) begin
15            left <= 320 - 2;
16            top <= 240 - 2;
17            if (left_scored)
18                x_dir <= 0;
19            else if (right_scored)
20                x_dir <= 1;
21            y_dir <= 2;
22            right_scored <= 0;
23            left_scored <= 0;
24        end
25        else if (update) begin
26            // hits left paddle
27            if (x_dir &
28                top >= left_paddle_top & top < left_paddle_top + 48 &
29                left >= 60 & left < 64) begin
30                y_dir <= (top >= left_paddle_top + 16) +
31                    (top >= left_paddle_top + 32);
32                x_dir <= 0;
33            end
34
35            // hits right paddle
36            else if (~x_dir &
37                top >= right_paddle_top & top < right_paddle_top + 48 &
38                left + 4 >= 577 & left + 4 < 581) begin
39                y_dir <= (top >= right_paddle_top + 16) +
40                    (top >= right_paddle_top + 32);
41                x_dir <= 1;
42            end
43
44            // hits top
45            else if (y_dir == 0 & top <= 0)
46                y_dir <= 2;
47
48            // hits bottom
49            else if (y_dir == 2 & top + 4 > 480)
50                y_dir <= 0;
51
52            // right score
53            else if (left <= 4) begin
54                right_scored <= 1;
55            end
56
57            // left score
58            else if (left > 632) begin
59                left_scored <= 1;
60            end
61
62            // normal
63            else begin
64                if (x_dir)
65                    left <= left - 2;
66                else
67                    left <= left + 2;
68
69                if (y_dir == 2)
70                    top <= top + 1;
```

```
71             else if (y_dir == 0)
72                 top <= top - 1;
73             end
74         end
75     end
76 end
77 end
78
79 assign display = (
80     col >= left &
81     col < left + 4 &
82     row >= top &
83     row < top + 4
84 );
85
86 endmodule: Ball
```

Lab Code [0 points]  
Filename: ChipInterface.sv

```
1 `default_nettype none
2
3 module RangeCheck
4     #(parameter WIDTH = 8)
5     (input logic [WIDTH-1:0] val,high,low,
6      output logic is_between);
7
8     assign is_between = (val<=high) && (val>=low);
9
10 endmodule : RangeCheck
11
12
13 module RangeCheck_test;
14     logic [7:0] val,high,low;
15     logic is_between;
16     RangeCheck #(8) dut (.*);
17     initial begin
18         $monitor("val:%b high:%b low:%b is_between:%b",
19                 val, high, low, is_between);
20         val = 8'd10;
21         high = 8'd20;
22         low = 8'd5;
23         #10
24         high = 8'd7;
25         #10
26         high = 8'd20;
27         low = 8'd15;
28         #10
29         $finish;
30     end
31 endmodule: RangeCheck_test
32
33
34
35 module OffsetCheck
36     #(parameter WIDTH = 8)
37     (input logic [WIDTH-1:0] val,delta,low,
38      output logic is_between);
39
40     assign is_between = (val<=(low+delta)) && (val>=low);
41
42 endmodule: OffsetCheck
43
44
45 module OffsetCheck_test;
46     logic [7:0] val, delta, low;
47     logic is_between;
48     OffsetCheck #(8) dut (.*);
49     initial begin
50         $monitor("val:%b delta:%b low:%b is_between:%b",
51                 val, delta, low, is_between);
52         val = 8'd5;
53         low = 8'd1;
54         delta = 8'd9;
55         #10
56         delta = 8'd2;
57         #10
58         low = 8'd5;
59         #10
60         low = 8'd6;
61         #10
62         $finish;
63     end
64 endmodule: OffsetCheck_test
65
66
67 module vga
68     (input logic CLOCK_50, reset,
69      output logic HS, VS, blank,
70      output logic [8:0] row,
```

```

71     output logic [9:0] col);
72
73     logic blank_en; //enable/disable the clock
74     logic [15:0] clock_count; //how many clocks
75
76     logic pulse_width;
77     // logic back_porch;
78     logic display;
79     // logic front_porch;
80
81     logic colcountclear_n;
82     logic cc_en;
83
84     logic [19:0] line_counter;
85     logic line_counter_reset_n;
86
87     assign blank = ~(display & blank_en);
88     assign HS = ~pulse_width;
89
90     OffsetCheck #(16) ocpw (clock_count,191,0, pulse_width);
91
92     // OffsetCheck #(16) ocbp (clock_count, 96, 192, back_porch);
93
94     OffsetCheck #(16) ocd (clock_count, 1279, 288, display);
95
96     // OffsetCheck #(16) ocfp (clock_count, 32, 1568, front_porch);
97
98     //if clock passes one timeline, we reset the column count
99     RangeCheck #(16) colrange (clock_count, 1598, 0, colcountclear_n);
100
101     //incrementing clock
102     Counter #(16) c ( , 1, ~colcountclear_n | reset, 0,
103                     CLOCK_50, 1'b1, clock_count);
104
105     //checking col range and incrementing col
106     assign cc_en = display && clock_count[0];
107     Counter #(16) col_count ( , cc_en, ~colcountclear_n | reset,
108                             0, CLOCK_50, 1'b1, col);
109
110     Counter #(16) row_count ( , ~colcountclear_n & blank_en,
111                             ~line_counter_reset_n | reset, 0, CLOCK_50, 1, row);
112
113     OffsetCheck #(20) VS_check (line_counter, 900000, 3200, VS);
114
115     OffsetCheck #(20) clock_check (line_counter, 817399, 49600, blank_en);
116
117     // define line counter
118     RangeCheck #(20) line_counter_check (line_counter,
119                                         833598, 0, line_counter_reset_n);
120     Counter #(20) ( , 1, ~line_counter_reset_n | reset, 0,
121                   CLOCK_50, 1, line_counter);
122
123 endmodule : vga
124
125 /*
126 module vga_test();
127
128     logic clock, reset, HS, VS, blank;
129     logic [8:0] row;
130     logic [9:0] col;
131     logic [31:0] i;
132
133     vga v (clock, reset, HS, VS, blank, row, col);
134
135     initial begin
136         // $monitor("%d, clock: %d, row: %d, col: %d, blank: %d, reset: %d",
137         $time, clock, row, col, blank, reset);
138         clock = 1;
139         #1800000
140         reset = 1;
141         #5

```

```

142         reset = 0;
143         $finish;
144     end
145     initial begin
146         reset = 1;
147         @(posedge clock);
148         @(posedge clock);
149         reset = 0;
150         @(posedge clock);
151         @(posedge clock);
152         @(posedge clock);
153         @(posedge clock);
154         @(posedge clock);
155         @(posedge clock);
156         @(posedge clock);
157         @(posedge clock);
158         @(posedge clock);
159         @(posedge clock);
160     end
161     always
162         #1 clock = ~clock;
163
164 endmodule: vga_test;
165 */
166
167 module ChipInterface
168     (input logic CLOCK_50,
169     input logic [3:0] KEY,
170     input logic [17:0] SW,
171     output logic [6:0] HEX0, HEX1, HEX2, HEX3,
172     HEX4, HEX5, HEX6, HEX7,
173     output logic [7:0] VGA_R, VGA_G, VGA_B,
174     output logic VGA_BLANK_N, VGA_CLK, VGA_SYNC_N,
175     output logic VGA_VS, VGA_HS);
176
177     logic [8:0] row;
178     logic [9:0] col;
179
180     logic serve, reset, update, left_scored, right_scored,
181     ball_display,
182     left_paddle_display, right_paddle_display,
183     left_score_display, right_score_display,
184     left_scored_display, right_scored_display,
185     left_win, right_win;
186
187     logic [9:0] left_paddle_top, right_paddle_top, reset_counter;
188     logic [9:0] left_score_counter, right_score_counter;
189
190     always_ff@(posedge CLOCK_50) begin
191
192         update <= (col == 639) & (row == 479) & (~update);
193
194         if (~KEY[0]) begin
195             reset <= 1;
196             serve <= 1;
197             reset_counter <= 0;
198         end
199
200         else if (reset_counter < 8) begin
201             reset <= 1;
202             serve <= 1;
203             reset_counter <= reset_counter + 1;
204             left_score_counter <= 20;
205             right_score_counter <= 20;
206         end
207
208         else if (left_score_counter < 40 | left_win) begin
209             if (update)
210                 left_score_counter <= left_score_counter + 1;
211             left_scored_display <= ~left_score_counter[3];
212         end

```

```
213
214     else if (right_score_counter < 40 | right_win) begin
215         if (update)
216             right_score_counter <= right_score_counter + 1;
217         right_scored_display <= ~right_score_counter[3];
218     end
219
220     else begin
221         reset <= 0;
222         left_scored_display <= 0;
223         right_scored_display <= 0;
224
225         if (left_scored) begin
226             serve <= 1;
227             left_score_counter <= 0;
228         end
229
230         else if (right_scored) begin
231             serve <= 1;
232             right_score_counter <= 0;
233         end
234
235         else if (~KEY[3]) begin
236             serve <= 0;
237         end
238     end
239 end
240
241 Ball b(
242     .clock(CLOCK_50),
243     .reset(serve | reset),
244     .update(update),
245     .left_paddle_top(left_paddle_top),
246     .right_paddle_top(right_paddle_top),
247     .row(row),
248     .col(col),
249     .left_scored(left_scored),
250     .right_scored(right_scored),
251     .display(ball_display)
252 );
253
254 Paddle left_paddle(
255     .clock(CLOCK_50),
256     .reset(reset),
257     .input_up(SW[17]),
258     .input_down(SW[16]),
259     .left(10'd60),
260     .update(update),
261     .row(row),
262     .col(col),
263     .top(left_paddle_top),
264     .display(left_paddle_display)
265 );
266
267 Paddle right_paddle(
268     .clock(CLOCK_50),
269     .reset(reset),
270     .input_up(SW[1]),
271     .input_down(SW[0]),
272     .left(10'd577),
273     .update(update),
274     .row(row),
275     .col(col),
276     .top(right_paddle_top),
277     .display(right_paddle_display)
278 );
279
280 Score left_score(
281     .clock(CLOCK_50),
282     .reset(reset),
283     .scored(left_scored),
```

```
284         .row(row),
285         .col(col),
286         .top(16),
287         .left(280),
288         .win(left_win),
289         .display(left_score_display),
290         .segs(HEX6)
291     );
292
293     Score right_score(
294         .clock(CLOCK_50),
295         .reset(reset),
296         .scored(right_scored),
297         .row(row),
298         .col(col),
299         .top(16),
300         .left(328),
301         .win(right_win),
302         .display(right_score_display),
303         .segs(HEX4)
304     );
305
306     Color color(
307         .left(left_paddle_display | left_score_display |
308             left_scored_display),
309         .right(right_paddle_display | right_score_display |
310             right_scored_display),
311         .ball(ball_display),
312         .R(VGA_R),
313         .G(VGA_G),
314         .B(VGA_B)
315     );
316
317     logic blank;
318     assign VGA_BLANK_N = ~blank;
319     vga dut(CLOCK_50, 0, VGA_HS, VGA_VS, blank, row, col);
320
321     assign HEX7 = 7'b1111111;
322     assign HEX5 = 7'b1111111;
323     assign HEX3 = 7'b1111111;
324     assign HEX2 = 7'b1111111;
325     assign HEX1 = 7'b1111111;
326     assign HEX0 = 7'b1111111;
327     assign VGA_SYNC_N = 1;
328     assign VGA_CLK = ~CLOCK_50;
329 endmodule: ChipInterface
330
```

Lab Code [0 points]  
Filename: Color.sv

```
1 `default_nettype none
2
3 module Color(
4     input  logic left, right, ball,
5     output logic [7:0] R, G, B);
6
7     always_comb begin
8
9         if (left) begin
10             R = 255;
11             G = 255;
12             B = 0;
13         end
14
15         else if (right) begin
16             R = 0;
17             G = 255;
18             B = 255;
19         end
20
21         else if (ball) begin
22             R = 255;
23             G = 255;
24             B = 255;
25         end
26
27         else begin
28             R = 0;
29             G = 0;
30             B = 0;
31         end
32     end
33 endmodule: Color
```



Lab Code [0 points]  
Filename: Paddle.sv

```
1 `default_nettype none
2
3 module Paddle(
4     input  logic clock, reset,
5     input  logic input_up, input_down,
6     input  logic [9:0] left,
7     input  logic update,
8     input  logic [9:0] row, col,
9     output logic [9:0] top,
10    output logic display);
11
12    logic up, down;
13
14    always_ff@(posedge clock) begin
15        up <= input_up;
16        down <= input_down;
17
18        if (reset)
19            top <= 240 - 24;
20        else if (update & up & ~down & (top < (480 - 48)))
21            top <= top + 4;
22        else if (update & down & ~up & (top >= 4))
23            top <= top - 4;
24    end
25
26    assign display = (
27        col >= left &
28        col < left + 4 &
29        row >= top &
30        row < top + 48
31    );
32
33 endmodule: Paddle
```

Lab Code [0 points]  
Filename: Score.sv

```
1 `default_nettype none
2
3 module Score(
4     input  logic clock, reset,
5     input  logic scored,
6     input  logic [9:0] row, col, top, left,
7     output logic [3:0] score,
8     output logic win, display,
9     output logic [6:0] segs);
10
11     logic [4:0] scoreX2;
12
13     assign score = scoreX2[4:1];
14
15     always_ff@(posedge clock) begin
16         if (reset) begin
17             win <= 0;
18             scoreX2 <= 0;
19         end
20         else if (score == 9)
21             win <= 1;
22         else if (scored)
23             scoreX2 <= scoreX2 + 1;
24     end
25
26     /*
27     ----s0----
28     |          |
29     | s5       | s1
30     |          |
31     ----s6----
32     |          |
33     | s4       | s2
34     |          |
35     ----s3----
36
37     */
38
39
40     logic [9:0] rx, ry;
41     logic s0, s1, s2, s3, s4, s5, s6, s7;
42     always_comb begin
43         rx = col - left;
44         ry = row - top;
45
46
47         s0 = rx >= 0 & rx < 32 & ry >= 0 & ry < 4;
48         s1 = rx >= 28 & rx < 32 & ry >= 0 & ry < 24;
49         s2 = rx >= 28 & rx < 32 & ry >= 24 & ry < 48;
50         s3 = rx >= 0 & rx < 32 & ry >= 44 & ry < 48;
51         s4 = rx >= 0 & rx < 4 & ry >= 24 & ry < 48;
52         s5 = rx >= 0 & rx < 4 & ry >= 0 & ry < 24;
53         s6 = rx >= 0 & rx < 32 & ry >= 22 & ry < 26;
54
55         if (score == 0) begin
56             display = s0 | s1 | s2 | s3 | s4 | s5;
57             segs = {1'b1, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0}; end
58         else if (score == 1) begin
59             display = s1 | s2;
60             segs = {1'b1, 1'b1, 1'b1, 1'b1, 1'b0, 1'b0, 1'b1}; end
61         else if (score == 2) begin
62             display = s0 | s1 | s3 | s4 | s6;
63             segs = {1'b0, 1'b1, 1'b1, 1'b0, 1'b0, 1'b1, 1'b0}; end
64         else if (score == 3) begin
65             display = s0 | s1 | s2 | s3 | s6;
66             segs = {1'b0, 1'b1, 1'b1, 1'b0, 1'b0, 1'b0, 1'b0}; end
67         else if (score == 4) begin
68             display = s1 | s2 | s5 | s6;
69             segs = {1'b0, 1'b0, 1'b1, 1'b1, 1'b0, 1'b0, 1'b1}; end
70         else if (score == 5) begin
```

```
71         display = s0 | s2 | s3 | s5 | s6;
72         segs = {1'b0, 1'b0, 1'b1, 1'b0, 1'b0, 1'b1, 1'b0}; end
73     else if (score == 6) begin
74         display = s0 | s2 | s3 | s4 | s5 | s6;
75         segs = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b1, 1'b0}; end
76     else if (score == 7) begin
77         display = s0 | s1 | s2;
78         segs = {1'b1, 1'b1, 1'b1, 1'b1, 1'b0, 1'b0, 1'b0}; end
79     else if (score == 8) begin
80         display = s0 | s1 | s2 | s3 | s4 | s5 | s6;
81         segs = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0}; end
82     else if (score == 9) begin
83         display = s0 | s1 | s2 | s3 | s5 | s6;
84         segs = {1'b0, 1'b0, 1'b1, 1'b0, 1'b0, 1'b0, 1'b0}; end
85     else begin
86         display = 0;
87         segs = {1'b1, 1'b1, 1'b1, 1'b1, 1'b1, 1'b1, 1'b1}; end
88     end
89
90 endmodule: Score
```

Lab Code [0 points]  
Filename: library.sv

```
1 `default_nettype none
2
3 module MagComp
4     #(parameter WIDTH = 8)
5     (output logic AltB, AeqB, AgtB,
6      input logic [WIDTH-1:0] A, B);
7
8     assign AeqB = (A == B);
9     assign AltB = (A < B);
10    assign AgtB = (A > B);
11
12 endmodule: MagComp
13
14 // module MagComp_test;
15
16 //     logic AltB, AeqB, AgtB;
17 //     logic [1:0] A, B;
18 //     logic [3:0] vector;
19
20 //     assign {A, B} = vector;
21
22 //     MagComp #(2) dut(.);
23
24 //     initial begin
25 //         $monitor("A:%b B:%b ->> AltB(%b) AeqB(%b) AgtB(%b)",
26 //                 //A, B, AltB, AeqB, AgtB);
27 //         for (vector = 4'b0; vector != 4'b1111; vector++)
28 //             #1;
29 //         #1
30 //         $finish;
31 //     end
32 // endmodule : MagComp_test
33
34 module Adder
35     #(parameter WIDTH=8)
36     (input logic [WIDTH-1:0] A, B,
37      input logic Cin,
38      output logic [WIDTH-1:0] S,
39      output logic Cout);
40
41     assign {Cout, S} = A + B + Cin;
42
43 endmodule : Adder
44
45 // module Adder_test;
46
47 //     logic [3:0] A, B;
48 //     logic Cin;
49 //     logic [3:0] S;
50 //     logic Cout;
51
52 //     logic [8:0] vector;
53 //     assign {Cin, A, B} = vector;
54
55 //     Adder #(4) dut(.);
56
57 //     initial begin
58 //         $monitor("Cin:%b A:%b B:%b ->> Cout:%b S:%b", Cin, A, B, Cout, S);
59 //         for (vector = 9'b0; vector != 9'b1_1111_1111; vector++)
60 //             #1;
61 //         #1;
62 //         $finish;
63 //     end
64
65 // endmodule : Adder_test
66
67 module Multiplexer
68     #(parameter WIDTH=8)
69     (input logic [WIDTH-1:0] I,
70      input logic [$clog2(WIDTH)-1:0] S,
```

```

71     output logic                                Y);
72
73     assign Y = I[S];
74
75 endmodule : Multiplexer
76
77 // module Multiplexer_test;
78
79 //     logic [7:0] I;
80 //     logic [2:0] S;
81 //     logic      Y;
82
83 //     Multiplexer dut(.);
84
85 //     initial begin
86 //         $monitor("I(%b), Sel(%b) --> Y(%b)", I, S, Y);
87 //         I = 8'b1011_0011;
88 //         for (S=3'b000; S != 3'b111; S++)
89 //             #1;
90 //         #1;
91 //         $finish;
92 //     end
93
94 // endmodule : Multiplexer_test
95
96 module Mux2to1
97     #(parameter WIDTH = 8)
98     (input  logic [WIDTH-1:0] I0, I1,
99      input  logic      S,
100     output logic [WIDTH-1:0] Y);
101
102     assign Y = (S) ? I1 : I0;
103
104 endmodule : Mux2to1
105
106 // module Mux2to1_test;
107
108 //     logic [1:0] I0, I1;
109 //     logic      S;
110 //     logic [1:0] Y;
111
112 //     logic [4:0] vector;
113 //     assign {S, I1, I0} = vector;
114
115 //     Mux2to1 #(2) dut(.);
116
117 //     initial begin
118 //         $monitor("Sel(%b) I1(%h) I0(%h) -> Y(%h)", S, I1, I0, Y);
119 //         for(vector = 5'b0; vector != 5'b11111; vector++)
120 //             #1;
121 //         #1;
122 //         $finish;
123 //     end
124
125 // endmodule : Mux2to1_test
126
127 module Decoder
128     #(parameter WIDTH=8)
129     (input  logic [$clog2(WIDTH)-1:0] I,
130      input  logic      en,
131      output logic [WIDTH-1:0]      D);
132
133     always_comb begin
134         D = 0;
135         if (en)
136             D = 1'b1 << I;
137     end
138
139 endmodule : Decoder
140
141 // module Decoder_test;

```

```
142
143 //    logic [2:0] I;
144 //    logic      en;
145 //    logic [7:0] D;
146
147 //    logic [3:0] vector;
148 //    assign {en, I} = vector;
149
150 //    Decoder #(8) dut(.*);
151
152 //    initial begin
153 //        $monitor("I(%b) en(%b) -> D(%b)", I, en, D);
154 //        for(vector = 4'd0; vector != 4'b1111; vector++)
155 //            #1;
156 //        #1;
157 //        $finish;
158 //    end
159
160 // endmodule : Decoder_test
161
162 module Register
163     #(parameter WIDTH=8)
164     (input logic [WIDTH-1:0] D,
165      input logic      en, clear, clock,
166      output logic [WIDTH-1:0] Q);
167
168     always_ff @(posedge clock)
169         if (en)
170             Q <= D;
171         else if (clear)
172             Q <= 0;
173
174 endmodule : Register
175
176 // module Register_test;
177
178 //    logic [7:0] D;
179 //    logic      en, clear, clock;
180 //    logic [7:0] Q;
181
182 //    Register dut(.*);
183
184 //    initial begin
185 //        clock = 0;
186 //        forever #5 clock = ~clock;
187 //    end
188
189 //    initial begin
190 //        $monitor("D(%b) clear(%b) en(%b) -> Q(%b)", D, clear, en, Q);
191 //        D <= 8'b0111_0001; clear <= 0; en <= 1;
192 //        #7;
193 //        D <= 8'b1000_1110; en <= 0;
194 //        #20;
195 //        clear <= 1;
196 //        #10;
197 //        $finish;
198 //    end
199
200 // endmodule : Register_test
201
202 module Counter
203     #(parameter WIDTH=8)
204     (input logic [WIDTH-1:0] D,
205      input logic en, clear, load, clock, up,
206      output logic [WIDTH-1:0] Q);
207
208     always_ff @(posedge clock)
209         if(clear)
210             Q <= 0;
211         else if(load)
212             Q <= D;
```

```
213 else if(en&up)
214   Q <= Q+1;
215 else if(en&~up)
216   Q <= Q-1;
217
218 endmodule: Counter
219
220 module Counter_test;
221   logic [7:0] D;
222   logic en, clear, load, clock, up;
223   logic [7:0] Q;
224   Counter dut(.);
225   initial begin
226     clock = 0;
227     forever #5 clock = ~clock;
228   end
229
230   initial begin
231     $monitor("D(%b) clear(%b) en(%b) load(%b) up(%b) -> Q(%b)",
232       D, clear, en, load, up, Q);
233
234     D <= 8'd10;
235     clear <= 1;
236     load <= 0;
237     en <= 1;
238     up <= 1;
239     @(posedge clock);
240     @(posedge clock);
241     clear <= 0;
242     @(posedge clock);
243     @(posedge clock);
244     @(posedge clock);
245     @(posedge clock);
246     @(posedge clock);
247     @(posedge clock);
248     @(posedge clock);
249     @(posedge clock);
250
251     #10
252     $finish;
253   end
254 endmodule: Counter_test
255
256 module ShiftRegister
257   #(parameter WIDTH = 8)
258   (input logic [WIDTH-1:0] D,
259    input logic en, left, load, clock,
260    output logic [WIDTH-1:0] Q);
261
262   always_ff @(posedge clock)
263     if(load)
264       Q <= D;
265     else if(en && left)
266       Q <= {Q[WIDTH-2:0], 1'b0};
267     else if(en && ~left)
268       Q <= {1'b0, Q[WIDTH-1:1]};
269
270 endmodule: ShiftRegister
271 /*
272 module ShiftRegister_test;
273   logic [7:0] D;
274   logic en, left, load, clock;
275   logic [7:0] Q;
276   ShiftRegister dut(.);
277
278   initial begin
279     clock = 0;
280     forever #5 clock = ~clock;
281   end
282
283   initial begin
```

```
284 $monitor("D(%b) en(%b) load(%b) left(%b) -> Q(%b)", D, en, load, left, Q);
285 D <= 8'b1010_1010;
286 en <= 1;
287 load <= 1;
288 left <= 1;
289 @(posedge clock);
290 @(posedge clock);
291 load <= 0;
292 @(posedge clock);
293 left <= 0;
294 @(posedge clock);
295 en <= 0;
296 @(posedge clock);
297 @(posedge clock);
298 #10
299 $finish;
300 end
301 endmodule: ShiftRegister_test
302 */
303 module BarrelShiftRegister
304 #(parameter WIDTH = 8)
305 (input logic [WIDTH-1:0] D,
306 input logic load, en, clock,
307 input logic [1:0] by,
308 output logic [WIDTH-1:0] Q);
309
310 always_ff @(posedge clock)
311 if(load)
312 Q <= D;
313 else if(en)
314 Q <= Q << by;
315 endmodule: BarrelShiftRegister
316 /*
317 module BarrelShiftRegister_test;
318 logic [7:0] D;
319 logic load, en, clock;
320 logic [1:0] by;
321 logic [7:0] Q;
322 BarrelShiftRegister dut(.*);
323
324 initial begin
325 clock = 0;
326 clock = 0;
327 forever #5 clock = ~clock;
328 end
329
330 initial begin
331 $monitor("D(%b) en(%b) load(%b) by(%b) -> Q(%b)", D, en, load, by, Q);
332 D <= 8'b1111_1111;
333 en <= 0;
334 load <= 0;
335 by <= 2'd3;
336 @(posedge clock);
337 en <= 1;
338 @(posedge clock);
339 by <= 2'd1;
340 @(posedge clock);
341 by <= 2'd0;
342 @(posedge clock);
343 load <= 1;
344 @(posedge clock);
345 @(posedge clock);
346 #10
347 $finish;
348 end
349 endmodule: BarrelShiftRegister_test
350 */
351 module Memory
352 #(parameter DW=16,
353 W=256,
354 AW=$clog2(W))
```



```
355 (input logic re,we,clock,
356 input logic [AW-1:0] Addr,
357 inout wire [DW-1:0] Data);
358
359 logic [DW-1:0] M[W];
360 logic [DW-1:0] out;
361
362 assign Data = (re) ? out:'bz;
363
364 always_ff @(posedge clock)
365 if(we) M[Addr] <= Data;
366
367 always_comb
368 out = M[Addr];
369
370 endmodule: Memory
371 /*
372 module Memory_test;
373 logic re, we, clock, tri_en;
374 logic [7:0] Addr;
375 tri [3:0] Data;
376 logic [3:0] driveData;
377
378 Memory #(.AW(8), .DW(4)) dut(.*);
379
380 assign Data = (tri_en) ? driveData : 4'bz;
381
382 initial begin
383 clock = 0;
384 forever #5 clock = ~clock;
385 end
386 initial begin
387 for (Addr = 8'd0; Addr < 8'd4; Addr += 1) begin
388     re <= 1'b0;
389     we <= 1'b1;
390     driveData <= Addr[3:0];
391     tri_en = 1'b1;
392     @(posedge clock);
393 end
394 #5 $finish;
395 end
396 endmodule: Memory_test
397 */
```