Problem 1: [8 points] Drill problem
Filename: library.sv

```
 1 `default_nettype none
 2
 3 module MagComp
 4   # (parameter WIDTH = 1)
 5   (input logic [WIDTH-1:0] A, B,
 6    output logic AltB, AeqB, AgtB);
 7
 8   always_comb begin
 9     if (A < B)
10       begin
11         AeqB = 1'b0;
12         AltB = 1'b1;
13         AgtB = 1'b0;
14       end
15     else if (A == B)
16       begin
17         AeqB = 1'b1;
18         AltB = 1'b0;
19         AgtB = 1'b0;
20       end
21     else
22       begin
23         AgtB = 1'b1;
24         AltB = 1'b0;
25         AeqB = 1'b0;
26       end
27   end
28
29 endmodule: MagComp
30
31
32 module Multiplexer
33   # (parameter WIDTH = 2)
34   // parameter S_WIDTH = $clog2(WIDTH);
35   (input logic [WIDTH-1:0] I,
36    input logic [$clog2(WIDTH)-1:0] S,
37    output logic Y);
38
39   always_comb begin
40     if (S < WIDTH)
41       Y = I[S];
42   end
43
44 endmodule: Multiplexer
45
46 module Mux2to1
47   // input width
48   # (parameter WIDTH = 1)
49   (input logic [WIDTH-1:0] I0, I1,
50    input logic S,
51    output logic [WIDTH-1:0] Y);
52
53   assign Y = S ? I1 : I0;
54
55 endmodule: Mux2to1
56
57 module Decoder
58     # (parameter WIDTH = 1)
59     // parameter I_WIDTH = $clog2(WIDTH);
60     (input logic [$clog2(WIDTH)-1:0] I,
61      input logic en,
62      output logic [WIDTH-1:0] D);
63
64     always_comb begin
65       D = 0;
66       if (en)
67         begin
68           if (I < WIDTH)
69             D[I] = 1'b1;
70         end
```

```
 71       else
 72         D = I;
 73     end
 74
 75 endmodule: Decoder
 76
 77 module Adder
 78   # (parameter WIDTH = 1)
 79   (input logic [WIDTH-1:0] A, B,
 80    input logic Cin,
 81    output logic [WIDTH-1:0] S,
 82    output logic Cout);
 83
 84   assign {Cout, S} = A + B + Cin;
 85
 86 endmodule: Adder
 87
 88 module Register
 89   # (parameter WIDTH = 1)
 90   (input logic [WIDTH-1:0] D,
 91    input logic en, clear,
 92    input logic clock,
 93    output logic [WIDTH-1:0] Q);
 94
 95    always_ff @(posedge clock)
 96      if (en)
 97        Q <= clear ? 0 : D;
 98
 99 endmodule: Register
100
101 module Counter
102   #(parameter WIDTH = 1)
103    (input logic en, clear, load, up,
104     input logic clk,
105     input logic [WIDTH-1:0] D,
106     output logic [WIDTH-1:0] Q);
107
108   always_ff @(posedge clk)
109     if (clear && en)
110       Q <= 0;
111     else if (load && en)
112       Q <= D;
113     else if (up && en)
114       Q <= Q + 1;
115 endmodule: Counter
116
117 module ShiftRegister
118   #(parameter WIDTH = 1)
119    (input logic en, left, load,
120     input logic clk,
121     input logic [WIDTH-1:0] D,
122     output logic [WIDTH-1:0] Q);
123
124   always_ff @(posedge clk)
125     if (load)
126       Q <= D;
127     else if (en && left)
128       Q <= (Q << 1);
129     else if (en && ~left)
130       Q <= (Q >> 1);
131
132 endmodule: ShiftRegister
133
134 module BarrelShiftRegister
135   #(parameter WIDTH = 1)
136    (input logic load, en,
137     input logic [1:0] by,
138     input logic [WIDTH-1:0] D,
139     input logic clk,
140     output logic [WIDTH-1:0] Q);
141
```

```
142   always_ff @(posedge clk)
143     if (load)
144       Q <= D;
145     else if (en)
146       Q <= (Q << by);
147  endmodule: BarrelShiftRegister
148
149  module Memory
150   #(parameter DW = 16,
151               W = 256,
152               AW = $clog2(W))
153    (input logic re, we, clk,
154     input logic [AW-1:0] Address,
155     inout tri   [DW-1:0] Data);
156
157   logic [DW-1:0] M[W];
158   logic [DW-1:0] out;
159
160   assign Data = (re) ? out : 'z;
161
162   always_ff @(posedge clk)
163     if (we)
164       M[Address] <= Data;
165
166   always_comb begin
167     out = M[Address];
168   end
169
170 endmodule: Memory
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
```

Problem 1: [8 points] Drill problem
Filename: library_tests.sv

```
 1 `default_nettype none
 2 module MagComp_test ();
 3   logic [4:0] A1, B1;
 4   logic AltB, AgtB, AeqB;
 5
 6   MagComp #(5) five (.A(A1), .B(B1), .AltB, .AgtB, .AeqB);
 7
 8   initial begin
 9     $monitor($time,, "A1=%d, B1=%d, AltB=%b, AgtB=%b, AeqB=%b",
10              A1, B1, AltB, AgtB, AeqB);
11     #5 A1 = 5'b10000;
12        B1 = 5'b00010;
13     #5 A1 = 5'b00001;
14        B1 = 5'b00001;
15     #5 B1 = 5'b01111;
16     #1 $finish;
17   end
18 endmodule: MagComp_test
19
20 module Multiplexer_test ();
21   logic [6:0] I;
22   logic Y;
23   logic [2:0] S;
24
25   Multiplexer #(7) m1 (.*);
26
27   initial begin
28     $monitor($time,, "I=%b, Y=%b, S=%d",
29              I, Y, S);
30     #5 I = 7'b0000000;
31        S = 3'b000;
32     #5 S = 3'b101;
33     #5 I = 7'b0100000;
34     #5 S = 3'b111;
35     #5 I = 7'b1111000;
36        S = 3'b001;
37     #5 S = 3'b100;
38     #1 $finish;
39   end
40
41 endmodule: Multiplexer_test
42
43 module Mux2to1_test ();
44   logic [6:0] I0, I1, Y;
45   logic S;
46
47   Mux2to1 #(7) m1 (.*);
48
49   initial begin
50     $monitor($time,, "I0=%b, I1=%b, Y=%b, S=%b",
51              I0, I1, Y, S);
52     #5 I0 = 7'b0000000;
53        I1 = 7'b1111111;
54        S = 1'b1;
55     #5 S = 1'b0;
56     #1 $finish;
57   end
58 endmodule: Mux2to1_test
59
60 module Decoder_test ();
61   // `define WIDTH1 3
62   // `define I_WIDTH1 2
63
64   logic [2:0] D;
65   logic [1:0] I;
66   logic en;
67
68   Decoder #(3) dec (.*);
69
70   initial begin
```

```
 71     $monitor($time,, "I=%d, en=%b, D=%b", I, en, D);
 72     #5 I = 2'b11;
 73        en = 0;
 74     #5 en = 1;
 75     #5 I = 2'b01;
 76     #1 $finish;
 77   end
 78 endmodule: Decoder_test
 79
 80 module Adder_test ();
 81   // `define 8 8
 82   logic Cin, Cout;
 83   logic [7:0] A, B, S;
 84
 85   Adder #(8) add (.*);
 86
 87   initial begin
 88     $monitor($time,, "A=%d, B=%d, Cin=%b, Cout=%b, S=%d", A, B, Cin, Cout, S);
 89     #5 A = 27;
 90        B = 0;
 91        Cin = 1'b0;
 92     #5 A = 38;
 93        B = 200;
 94     #5 Cin = 1'b1;
 95     #5 A = 129;
 96        B = 129;
 97     #1 $finish;
 98   end
 99 endmodule: Adder_test
100
101
102 module Register_test ();
103   //`define 3 3
104   logic [2:0] D, Q;
105   logic en, clear;
106   logic clk, reset_L;
107
108   Register #(3) regis (.*);
109
110   initial begin
111     $monitor($time,, "Q=%d, D=%d, en=%d, clear=%b",
112              Q, D, en, clear);
113     clk = 0;
114     forever #5 clk = ~clk;
115   end
116
117   initial begin
118     D = 3'b000;
119     en = 0;
120     clear = 0;
121     reset_L <= 0;
122     @(posedge clk);
123     reset_L <= 1;
124     @(posedge clk);
125     D = 3'b100;
126   // reset is synchronous, so must wait for a clk edge
127     @(posedge clk);
128     en = 1;
129     @(posedge clk);
130     D = 3'b010;
131     @(posedge clk);
132     D = 3'b011;
133     clear = 1;
134     @(posedge clk);
135     en = 0;
136     @(posedge clk);
137     clear = 0;
138     @(posedge clk);
139     @(posedge clk);
140     D = 3'b111;
141     en = 1;
```

```
142     @(posedge clk);
143     D = 3'b101;
144     en = 0;
145     @(posedge clk);
146     #1 $finish;
147   end
148
149 endmodule: Register_test
150
151 module Counter_test ();
152   // WIDTH = 6
153   logic en, clear, load, up;
154   logic clk;
155   logic [5:0] D, Q;
156
157   Counter #(6) counti (.*);
158
159   initial begin
160     $monitor($time,, "Q=%d, D=%d, en=%d, clear=%b \
161                       load=%b, up=%b",
162              Q, D, en, clear, load, up);
163     clk = 0;
164     forever #5 clk = ~clk;
165   end
166
167
168   initial begin
169     en = 0;
170     clear = 0;
171     load = 0;
172     up = 0;
173     D = 6'b011101;
174     @(posedge clk);
175     en = 1;
176     load = 1;
177     @(posedge clk);
178     load = 0;
179     up = 1;
180     @(posedge clk);
181     @(posedge clk);
182     clear = 1;
183     @(posedge clk);
184     clear = 0;
185     @(posedge clk);
186     load = 1;
187     @(posedge clk);
188     #1 $finish;
189   end
190
191 endmodule: Counter_test
192
193 module ShiftRegister_test ();
194   // WIDTH = 9
195   logic en, load, left;
196   logic clk;
197   logic [8:0] D, Q;
198
199   initial begin
200     $monitor($time,, "Q=%d, D=%d, en=%d, load=%b, left=%b",
201              Q, D, en, load, left);
202     clk = 0;
203     forever #5 clk = ~clk;
204   end
205
206   ShiftRegister #(9) shifti (.*);
207
208   initial begin
209     en = 0;
210     left = 0;
211     load = 0;
212     // up = 0;
```

```
213     D = 9'b00000001;
214     @(posedge clk);
215     en = 1;
216     load = 1;
217     left = 1;
218     @(posedge clk);
219     load = 0;
220     @(posedge clk);
221     @(posedge clk);
222     left = 0;
223     @(posedge clk);
224     @(posedge clk);
225     en = 0;
226     @(posedge clk);
227     @(posedge clk);
228     load = 1;
229     en = 1;
230     left = 1;
231     @(posedge clk);
232     @(posedge clk);
233     left = 0;
234     @(posedge clk);
235     #1 $finish;
236   end
237
238 endmodule: ShiftRegister_test
239
240 module BarrelShiftRegister_test ();
241   // WIDTH = 5
242   logic load, en, clk;
243   logic [1:0] by;
244   logic [4:0] D, Q;
245
246   initial begin
247     $monitor($time,, "Q=%d, D=%d, en=%d, load=%b, by=%d",
248              Q, D, en, load, by);
249     clk = 0;
250     forever #5 clk = ~clk;
251   end
252
253
254   BarrelShiftRegister #(5) barri (.*);
255
256   initial begin
257     en = 0;
258     by = 2'b01;
259     load = 0;
260     // up = 0;
261     D = 5'b00001;
262     @(posedge clk);
263     en = 1;
264     load = 1;
265     @(posedge clk);
266     load = 0;
267     @(posedge clk);
268     @(posedge clk);
269     @(posedge clk);
270     @(posedge clk);
271     @(posedge clk);
272     load = 1;
273     @(posedge clk);
274     en = 0;
275     @(posedge clk);
276     load = 0;
277     by = 2'b10;
278     en = 1;
279     @(posedge clk);
280     load = 1;
281     @(posedge clk);
282     @(posedge clk);
283     load = 0;
```

```
284     by = 2'b11;
285     @(posedge clk);
286     load = 1;
287     @(posedge clk);
288     by = 2'b00;
289     @(posedge clk);
290     @(posedge clk);
291     #1 $finish;
292   end
293
294 endmodule: BarrelShiftRegister_test
295
296 module Memory_test();
297   logic re, we, clk;
298   logic [7:0] Address;
299   tri [15:0] Data;
300   logic en1, en2;
301
302   Memory memi (.*);
303   initial begin
304     $monitor($time,, "Address=%d, Data=%d, we=%b, re=%b, en1=%b",
305              Address, Data, we, re, en1);
306     clk = 0;
307     forever #5 clk = ~clk;
308   end
309
310   assign Data = (en1) ? 16'b1 : 'z;
311
312   initial begin
313     re = 0;
314     we = 1;
315     en1 = 1;
316     Address = 7'b1101101;
317     @(posedge clk);
318     @(posedge clk);
319     en1 = 0;
320     re = 1;
321     we = 0;
322     @(posedge clk);
323     @(posedge clk);
324     Address = 7'b1000001;
325     @(posedge clk);
326     re = 0;
327     we = 1;
328     en1 = 1;
329     // Data = 16'b1111111111111111;
330     @(posedge clk);
331     en1 = 0;
332     re = 1;
333     we = 0;
334     @(posedge clk);
335     Address = 7'b1101101;
336     @(posedge clk);
337     #1 $finish;
338   end
339
340 endmodule: Memory_test
341
342
343
344
345
346
347
348
349
350
```

Problem 3: [10 points]
Filename: hw8prob3.sv

```
 1 `default_nettype none
 2 module OnesCount
 3  #(parameter w = 30)
 4  (input  logic         d_in_ready, clk, reset,
 5   input  logic [w-1:0] d_in,
 6   output logic         dor,
 7   output logic [$clog2(w)-1:0] d_out);
 8
 9   logic lowBit, done, Cclr_L, Cinc_L;
10   logic Sload_L, Sshift_L, Oclr_L;
11   logic Oinc_L;
12   logic dummy1, dummy2;
13   logic [29:0] shifted;
14
15   logic [$clog2(w)-1:0] SC;
16
17   assign lowBit = shifted[0];
18
19   fsm #(w) control (.*);
20
21   ShiftRegister #(w) sr (.Q(shifted), .D(d_in), .clk,
22                          .load(~Sload_L), .en(~Sshift_L), .left(1'b0));
23
24   Counter #($clog2(w)) sc (.clk, .en(1'b1), .D(5'b0), .load(1'b0),
25                            .clear(~Cclr_L), .up(~Cinc_L), .Q(SC));
26
27   MagComp #($clog2(w)) cmp (.AltB(dummy1), .AeqB(done),
28                             .AgtB(dummy2), .A(SC), .B(5'd30));
29
30   Counter #($clog2(w)) oct (.clk, .clear(~Oclr_L), .up(~Oinc_L),
31                       .Q(d_out), .en(1'b1), .D(5'b0), .load(1'b0));
32
33 endmodule: OnesCount
34
35 module fsm
36  #(parameter w = 30)
37   (input  logic clk, reset, done,
38    input  logic d_in_ready, lowBit,
39    output logic Cclr_L, Cinc_L, Sload_L, Sshift_L, Oclr_L, Oinc_L, dor);
40
41   enum logic {A = 1'b0, B = 1'b1} cur_state, n_state;
42
43   always_comb begin
44     case (cur_state)
45       A: begin  //State A
46          n_state = d_in_ready ? B : A;
47          Cclr_L  = d_in_ready ? 0 : 1;
48          Sload_L = d_in_ready ? 0 : 1;
49          Oclr_L  = d_in_ready ? 0 : 1;
50          Sshift_L = 1;
51          Cinc_L   = 1;
52          Oinc_L   = 1;
53          dor = 0; // D_out_ready
54          end
55       B: begin  //State B
56          n_state  = (done)? A : B;
57          dor      = (done)? 1 : 0;
58          Cclr_L   = 1;
59          Sload_L  = 1;
60          Oclr_L   = 1;
61          Cinc_L   = (done) ? 1 : 0;
62          Sshift_L = (done) ? 1 : 0;
63          Oinc_L = (done)? 1:~lowBit;
64          end
65     endcase
66   end
67
68   always_ff @(posedge clk, posedge reset)
69     if (reset)
70       cur_state <= A;
```

```
 71     else
 72       cur_state <= n_state;
 73
 74 endmodule: fsm
```