```
Problem 3: [20 points]
Filename: hw7prob3.sv
AndrewID: jtbell
```

```
`default_nettype none

module MagComp
  #(parameter   WIDTH = 8)
  (output logic             AltB, AeqB, AgtB,
   input  logic [WIDTH-1:0] A, B);

  assign AeqB = (A == B);
  assign AltB = (A <  B);
  assign AgtB = (A >  B);

endmodule: MagComp

// An Adder is a combinational sum generator.
module Adder
  #(parameter WIDTH=8)
  (input  logic [WIDTH-1:0] A, B,
   input  logic             cin,
   output logic [WIDTH-1:0] sum,
   output logic             cout);

   assign {cout, sum} = A + B + cin;

endmodule : Adder

// The Multiplexer chooses one of WIDTH bits
module Multiplexer
  #(parameter WIDTH=8)
  (input  logic [WIDTH-1:0]         I,
   input  logic [$clog2(WIDTH)-1:0] S,
   output logic                     Y);

   assign Y = I[S];

endmodule : Multiplexer

// The 2-to-1 Multiplexer chooses one of two multi-bit inputs.
module Mux2to1
  #(parameter WIDTH = 8)
  (input  logic [WIDTH-1:0] I0, I1,
   input  logic             S,
   output logic [WIDTH-1:0] Y);

  assign Y = (S) ? I1 : I0;

endmodule : Mux2to1

// The Decoder converts from binary to one-hot codes.
module Decoder
  #(parameter WIDTH=8)
  (input  logic [$clog2(WIDTH)-1:0] I,
   input  logic                     en,
   output logic [WIDTH-1:0]         D);

  always_comb begin
    D = '0;
    if (en)
      D[I] = 1'b1;
  end

endmodule : Decoder

// A DFlipFlop stores the input bit synchronously with the clock signal.
// preset and reset are asynchronous inputs.
module DFlipFlop
  (input  logic D,
   input  logic preset_L, reset_L, clock,
   output logic Q);

```

```
 70   always_ff @(posedge clock, negedge preset_L, negedge reset_L)
 71     if (~preset_L & reset_L)
 72       Q <= 1'b1;
 73     else if (~reset_L & preset_L)
 74       Q <= 1'b0;
 75     else if (~reset_L & ~preset_L)
 76       Q <= 1'bX;
 77     else
 78       Q <= D;
 79
 80 endmodule : DFlipFlop
 81
 82 // A Register stores a multi-bit value.
 83 // Enable has priority over Clear
 84 module Register
 85   #(parameter WIDTH=8)
 86   (input  logic [WIDTH-1:0] D,
 87    input  logic             en, clear, clock,
 88    output logic [WIDTH-1:0] Q);
 89
 90   always_ff @(posedge clock)
 91     if (en)
 92       Q <= D;
 93     else if (clear)
 94       Q <= '0;
 95
 96 endmodule : Register
 97
 98 // A binary up-down counter.
 99 // Clear has priority over Load, which has priority over Enable
100 module Counter
101   #(parameter WIDTH=8)
102   (input  logic [WIDTH-1:0] D,
103    input  logic             en, clear, load, clock, up,
104    output logic [WIDTH-1:0] Q);
105
106   always_ff @(posedge clock)
107     if (clear)
108       Q <= {WIDTH {1'b0}};
109     else if (load)
110       Q <= D;
111     else if (en)
112       if (up)
113         Q <= Q + 1'b1;
114       else
115         Q <= Q - 1'b1;
116
117 endmodule : Counter
118
119 // A Synchronizer takes an asynchronous input and changes it to synchronized
120 module Synchronizer
121   (input  logic async, clock,
122    output logic sync);
123
124   logic metastable;
125
126   DFlipFlop one(.D(async),
127                 .Q(metastable),
128                 .clock,
129                 .preset_L(1'b1),
130                 .reset_L(1'b1)
131                );
132
133   DFlipFlop two(.D(metastable),
134                 .Q(sync),
135                 .clock,
136                 .preset_L(1'b1),
137                 .reset_L(1'b1)
138                );
139
140 endmodule : Synchronizer
```

```
141
142 // A PIPO Shift Register, with controllable shift direction
143 // Load has priority over shifting.
144 module ShiftRegister_PIPO
145   #(parameter WIDTH=8)
146   (input  logic [WIDTH-1:0] D,
147    input  logic             en, left, load, clock,
148    output logic [WIDTH-1:0] Q);
149
150   always_ff @(posedge clock)
151     if (load)
152       Q <= D;
153     else if (en)
154       if (left)
155         Q <= {Q[WIDTH-2:0], 1'b0};
156       else
157         Q <= {1'b0, Q[WIDTH-1:1]};
158
159 endmodule : ShiftRegister_PIPO
160
161 // A SIPO Shift Register, with controllable shift direction
162 // Load has priority over shifting.
163 module ShiftRegister_SIPO
164   #(parameter WIDTH=8)
165   (input  logic             serial,
166    input  logic             en, left, clock,
167    output logic [WIDTH-1:0] Q);
168
169   always_ff @(posedge clock)
170     if (en)
171       if (left)
172         Q <= {Q[WIDTH-2:0], serial};
173       else
174         Q <= {serial, Q[WIDTH-1:1]};
175
176 endmodule : ShiftRegister_SIPO
177
178 // A BSR shifts bits to the left by a variable amount
179 module BarrelShiftRegister
180   #(parameter WIDTH=8)
181   (input  logic [WIDTH-1:0] D,
182    input  logic             en, load, clock,
183    input  logic [      1:0] by,
184    output logic [WIDTH-1:0] Q);
185
186   logic [WIDTH-1:0] shifted;
187   always_comb
188     case (by)
189       default: shifted = Q;
190       2'b01: shifted = {Q[WIDTH-2:0], 1'b0};
191       2'b10: shifted = {Q[WIDTH-3:0], 2'b0};
192       2'b11: shifted = {Q[WIDTH-4:0], 3'b0};
193     endcase
194
195   always_ff @(posedge clock)
196     if (load)
197         Q <= D;
198     else if (en)
199         Q <= shifted;
200
201 endmodule : BarrelShiftRegister
202
203 module RangeCheck
204     #(parameter  WIDTH = 8)
205     (input logic [WIDTH-1:0] low,
206     input logic [WIDTH-1:0] high,
207     input logic [WIDTH-1:0] val,
208     output logic is_between);
209
210     assign is_between = (low <= val  &&  val <= high) ? 1'b1: 1'b0;
211 endmodule
```

```
212
213 module hw7prob3
214     (input logic [6:0] score,
215     input logic [1:0] score_type,
216     input logic start, grade_it,
217     output logic grade_A, grade_B, grade_C, grade_D, grade_R,
218     input logic clock, reset_L);
219
220     logic [6:0] new_score,start_H,start_E,start_L,start_P,
221                 add_H,add_E,add_L,add_P,
222                 divided_H,divided_E_1,divided_E_2,divided_L,
223                 score_H,score_E,score_L,score_P,
224                 divided_P,fin_1,fin_2,final_val,combined_E_val;
225     logic hw_en, lab_en, exam_en, class_p_en,reset_all;
226
227
228     my_fsm fsm (.*);
229     //set en to be edited at start
230     Mux2to1 #(7) choose_gradeIt(.I0(7'b0),.I1(score),.Y(new_score),
231         .S(grade_it));
232     MagComp #(2) hw_comp (.A(2'b00),.B(score_type), .AeqB(hw_en),
233                 .AgtB(), .AltB());
234     MagComp #(2) lab_comp (.A(2'b01),.B(score_type), .AeqB(lab_en),
235                 .AgtB(), .AltB());
236     MagComp #(2) class_partipation_comp (.A(2'b10),.B(score_type),
237                 .AeqB(exam_en), .AgtB(), .AltB());
238     MagComp #(2) exam_comp (.A(2'b11),.B(score_type), .AeqB(class_p_en),
239                 .AgtB(), .AltB());
240     //scpre_H,E,L,P need values initially
241
242     Adder #(7) hw_add (.A(new_score), .B(score_H),.cin(1'd0),.cout(),
243         .sum(add_H));
244     Adder #(7) exam_add (.A(new_score), .B(score_E),.cin(1'd0),.cout(),
245         .sum(add_E));
246     Adder #(7) lab_add (.A(new_score), .B(score_L),.cin(1'd0),.cout(),
247         .sum(add_L));
248     Adder #(7) class_particiaption_add (.A(new_score), .B(score_P),.cin(1'd0),
249         .cout(),.sum(add_P));
250
251     Mux2to1 #(7) choose_start_H(.I1(new_score),.I0(add_H),.Y(start_H),
252         .S(start));
253     Mux2to1 #(7) choose_start_E(.I1(new_score),.I0(add_E),.Y(start_E),
254         .S(start));
255     Mux2to1 #(7) choose_start_L(.I1(new_score),.I0(add_L),.Y(start_L),
256         .S(start));
257     Mux2to1 #(7) choose_start_P(.I1(new_score),.I0(add_P),.Y(start_P),
258         .S(start));
259
260
261
262     //issue is with either with add or score
263     Register #(7) reg_H(.D(start_H), .Q(score_H), .en(hw_en),.clock(clock),
264         .clear(reset_all));
265     Register #(7) reg_E(.D(start_E), .Q(score_E), .en(exam_en),.clock(clock),
266         .clear(reset_all));
267     Register #(7) reg_L(.D(start_L), .Q(score_L), .en(lab_en),.clock(clock),
268         .clear(reset_all));
269     Register #(7) reg_P(.D(start_P), .Q(score_P), .en(class_p_en),.clock(clock),
270         .clear(reset_all));
271
272     assign divided_H = score_H >> 2;
273     assign divided_E_1 = score_E >> 2;
274     assign divided_E_2 = score_E >> 3;
275     assign divided_L =  score_L  >> 2;
276     assign divided_P = score_P  >> 3;
277
278     Adder #(7) add_combine_E (.A(divided_E_1), .B(divided_E_2),.cin(1'd0),
279         .cout(),.sum(combined_E_val));
280
281     Adder #(7) add_fin1 (.A(combined_E_val), .B(divided_H),.cin(1'd0),
282         .cout(),.sum(fin_1));
```

```
283     Adder #(7) add_fin2 (.A(fin_1), .B(divided_L),.cin(1'd0),
284         .cout(),.sum(fin_2));
285     Adder #(7) add_fin3 (.A(fin_2), .B(divided_P),.cin(1'd0),
286         .cout(),.sum(final_val));
287 
288     RangeCheck #(7) R_Checker(.low(7'd0),.high(7'd59),.val(final_val),
289         .is_between(grade_R));
290     RangeCheck#(7) D_Checker(.low(7'd60),.high(7'd69),.val(final_val),
291         .is_between(grade_D));
292     RangeCheck #(7) C_Checker(.low(7'd70),.high(7'd79),.val(final_val),
293         .is_between(grade_C));
294     RangeCheck #(7) B_Checker(.low(7'd80),.high(7'd89),.val(final_val),
295         .is_between(grade_B));
296     RangeCheck #(7) A_Checker(.low(7'd90),.high(7'd100),.val(final_val),
297         .is_between(grade_A));
298 endmodule
299 
300 module my_fsm
301     (input logic start, grade_it, reset_L, clock,
302     input logic [1:0] score_type,
303     output logic reset_all);
304   enum {INIT, SHIFTING} state, nextState;
305 
306   always_ff @(posedge clock, negedge reset_L) begin
307     if (~reset_L)
308       state <= INIT;
309     else
310       state <= nextState;
311   end
312     always_comb begin
313         reset_all = 1'b1;
314         case(state)
315             INIT:begin
316               if(~start) begin
317                 nextState = INIT;
318                 reset_all = 1'b1;
319               end
320               if(start) begin
321                 nextState = SHIFTING;
322                 reset_all = 1'b1;
323               end
324             end
325             SHIFTING:begin
326                 if(~start) begin
327                 nextState = SHIFTING;
328                 reset_all = 1'b0;
329               end
330                 if(start) begin
331                 nextState = SHIFTING;
332                 reset_all = 1'b1;
333               end
334             end
335         endcase
336     end
337 
338 endmodule
```