Lab Code [10 points]
Filename: MagicSquares.sv
AndrewID: dchan2

```
`default_nettype none

// add a single BCD digit to another
module BCDOneDigitAdd
    (input  logic [3:0] A, B,
     input  logic Cin,
     output logic [3:0] Sum,
     output logic Cout);

    // create internal 5-bit temp_sum
    // 4-bits for the digit, plus a carry
    logic [4:0] temp_sum;

    // get the temporary sum A + B + carryIn
    assign temp_sum = A + B + Cin;

    // use temp_sum to get the final sum and carryOut
    always_comb begin

        if (temp_sum >= 10) begin
            Sum = temp_sum - 4'd10; // sum = temp_sum without carry
            Cout = 1'b1; // carry is 1 since temp_sum >= 10
        end

        else begin
            Sum = temp_sum; // sum = temp_sum
            Cout = 1'b0; // carry is 0 since temp_sum <= 10
        end

    end

endmodule: BCDOneDigitAdd

// test BCDOneDigitAddS
// module BCDOneDigitAdd_test();
//     logic [3:0] A,B,Sum;
//     logic Cin, Cout;
//     BCDOneDigitAdd DUT(.A(A), .B(B),. Sum(Sum), .Cin(Cin), .Cout(Cout));
//         initial begin
//             $monitor($time,,
//             "A=%b, B=%b, Sum=%b, Cin=%b, Cout=%b",
//             A,B,Sum,Cin,Cout);
//             A=4'b0001;
//             B=4'b0000;
//             Cin=1;
//         #10 A=4'b1000; //10
//             B=4'b0010;
//             Cin=0;
//         #10 A=4'b1000; //17
//             B=4'b1000;
//             Cin=1;
//         #10 A=4'b0011; //15
//             B=4'b0100;
//             Cin=0;
//         end

// endmodule: BCDOneDigitAdd_test

// add two 2-BCD digit numbers together
module BCDThreeDigitAdd
    (input  logic [3:0] A, B, C,
     output logic [7:0] Sum);

    // create first adder output wires
    logic [3:0] ab, abc_no_carry;
    logic temp_carry1, temp_carry2;

    // get first sum: A + B
    BCDOneDigitAdd ad0(.A, .B, .Sum(ab), .Cin('0), .Cout(temp_carry1));
```

```
 70
 71     BCDOneDigitAdd ad1(.A(ab), .B(C), .Cin('0),
 72                        .Sum(abc_no_carry), .Cout(temp_carry2));
 73
 74     logic [3:0] tens;
 75     assign tens = temp_carry1 + temp_carry2;
 76     assign Sum = {tens,abc_no_carry};
 77
 78 endmodule: BCDThreeDigitAdd
 79
 80
 81 // test BCDThreeDigitAdd - CHECK TEST CASES!!!
 82 module BCDThreeDigitAdd_test();
 83     logic [3:0] A,B,C;
 84     logic [7:0] Sum;
 85     BCDThreeDigitAdd DUT(.A(A), .B(B), .C(C), .Sum(Sum));
 86
 87     initial begin
 88         $monitor($time,,
 89         "A=%d, B=%d, C=%d, Sum=%b",
 90         A, B, C, Sum);
 91       A=4'b1000;
 92       B=4'b1000;
 93       C=4'b0000;
 94     #10 A=4'b1000; //12
 95       B=4'b0010;
 96       C=4'b0010;
 97     #10 A=4'b1000; //24
 98       B=4'b1000;
 99       C=4'b1000;
100     #10 A=4'b0011; //15
101       B=4'b0100;
102       C=4'b1000;
103     #10 A=4'b1000; //15
104       B=4'b0100;
105       C=4'b0111;
106     end
107
108 endmodule: BCDThreeDigitAdd_test
109
110
111 // compare two values and decide if they are equal
112 module Comparator
113     (input logic [7:0] A, B,
114      output logic equal);
115
116     // check if A = B
117     assign equal = (A == B);
118
119 endmodule: Comparator
120
121 //make test cases for comparator
122 // module Comparator_test();
123 //   logic [3:0] A, B;
124 //   logic equal;
125 //   Comparator C(.A(A), .B(B), .equal(equal));
126 //   initial begin
127 //     $monitor($time,,
128 //     "A=%b, B=%b, equal=%b",
129 //     A,B,equal);
130 //     #10 A=4'd5;
131 //     B=4'd5;
132 //     #10 A=4'd5;
133 //     B=4'd6;
134 //     #10 A=4'd0;
135 //     B=4'd0;
136 //     #10 $finish;
137 //   end
138
139
140 // endmodule: Comparator_test
```

```
141
142
143 // compare three values and decide if they are all equal
144 module ThreeValueComparator
145     (input logic [7:0] A, B, C,
146     output logic equal);
147     always_comb begin
148
149         // check if A = B = C
150         if(A==B && A==C) begin
151             equal = 1;
152         end
153
154         else begin
155             equal = 0;
156         end
157     end
158 endmodule: ThreeValueComparator
159
160 // // test ThreeValueComparator
161 // module ThreeValueComparator_test();
162 //   logic [3:0] A, B, C;
163 //   logic equal;
164 //   ThreeValueComparator TVC(.A(A), .B(B), .C(C), .equal(equal));
165 //   initial begin
166 //     $monitor($time,,
167 //     "A=%b, B=%b, C=%b, equal=%b",
168 //     A,B,C,equal);
169 //     #10 A=4'd5;
170 //     B=4'd5;
171 //     C=4'd5;
172 //     #10 A=4'd5;
173 //     B=4'd6;
174 //     C=4'd5;
175 //     #10 A=4'd6;
176 //     B=4'd5;
177 //     C=4'd5;
178 //     #10 A=4'd5;
179 //     B=4'd5;
180 //     C=4'd6;
181 //     #10 $finish;
182 //   end
183
184 // endmodule: ThreeValueComparator_test
185
186
187 // compare all row, column, and diagonal sums
188 module FullComparator
189     (input logic [7:0] c1,c2,c3,
190     input logic [7:0] r1,r2,r3,
191     input logic [7:0] d1,d2,
192     output logic equal);
193
194     // create some internal lines to connect the sub-comparators
195     logic c,r,d,cdr;
196
197     // check that all columns are equal
198     ThreeValueComparator col(.equal(c),.A(c1),.B(c2),.C(c3));
199
200     // check that all rows are equal
201     ThreeValueComparator row(.equal(r),.A(r1),.B(r2),.C(r3));
202
203     // check that both diagonals are equal
204     Comparator diag(.equal(d),.A(d1),.B(d2));
205
206     // check that all columns = all rows = all diagonals, by commutativity
207     ThreeValueComparator all(.equal(cdr),.A(c1),.B(r1),.C(d1));
208
209     // check that all previous checks have passed and all is equal
210     always_comb begin
211         if(cdr && r && c && d)begin
```

```
212             equal = 1'b1;
213         end
214 
215         else begin
216             equal = 1'b0;
217         end
218     end
219 
220 endmodule: FullComparator
221 
222 // //test cases for FullComparator
223 // module FullComparator_test();
224 //   logic [7:0] c1,c2,c3;
225 //   logic [7:0] r1,r2,r3;
226 //   logic [7:0] d1,d2;
227 //   logic equal;
228 //   FullComparator FC(.equal, .c1(c1), .c2(c2), .c3(c3), .r1(r1),
229 //                     .r2(r2), .r3(r3), .d1(d1), .d2(d2));
230 //   initial begin
231 //     $monitor($time,,
232 //     "c1=%b, c2=%b, c3=%b, r1=%b, r2=%b, r3=%b, d1=%b, d2=%b, equal=%b",
233 //     c1,c2,c3,r1,r2,r3,d1,d2,equal);
234 //     #10 c1=4'd4;
235 //     c2=4'd4;
236 //     c3=4'd4;
237 //     r1=4'd4;
238 //     r2=4'd4;
239 //     r3=4'd4;
240 //     d1=4'd4;
241 //     d2=4'd4;
242 //     #10 c1=4'd4;
243 //     c2=4'd4;
244 //     c3=4'd4;
245 //     r1=4'd5;
246 //     r2=4'd5;
247 //     r3=4'd5;
248 //     d1=4'd6;
249 //     d2=4'd6;
250 //     #10 c1=4'd4;
251 //     c2=4'd4;
252 //     c3=4'd4;
253 //     r1=4'd4;
254 //     r2=4'd5;
255 //     r3=4'd4;
256 //     d1=4'd4;
257 //     d2=4'd4;
258 //     #10 $finish;
259 //   end
260 
261 // endmodule:FullComparator_test
262 
263 
264 // make sums, then compare them to see if they are equal
265 module EquivalenceChecker
266     (input  logic [35:0] values,
267      output logic [7:0]  magic_constant,
268      output logic        equal);
269 
270     // define internal sum variables
271     logic [7:0] r1, r2, r3, c1, c2, c3, d1, d2;
272     logic carry1, carry2, carry3, carry4, carry5, carry6, carry7, carry8;
273 
274 
275     // get row sums
276     BCDThreeDigitAdd row1(.A(values[3:0]), .B(values[7:4]),
277                           .C(values[11:8]), .Sum(magic_constant));
278     BCDThreeDigitAdd row2(.A(values[15:12]), .B(values[19:16]),
279                           .C(values[23:20]), .Sum(r2));
280     BCDThreeDigitAdd row3(.A(values[27:24]), .B(values[31:28]),
281                           .C(values[35:32]), .Sum(r3));
282 
```

```
283     // get column sums
284     BCDThreeDigitAdd col1(.A(values[3:0]), .B(values[15:12]),
285                           .C(values[27:24]), .Sum(c1));
286     BCDThreeDigitAdd col2(.A(values[7:4]), .B(values[19:16]),
287                           .C(values[31:28]), .Sum(c2));
288     BCDThreeDigitAdd col3(.A(values[11:8]), .B(values[23:20]),
289                           .C(values[35:32]), .Sum(c3));
290 
291     // get diagonal sums
292     BCDThreeDigitAdd diag1(.A(values[3:0]), .B(values[19:16]),
293                            .C(values[35:32]), .Sum(d1));
294     BCDThreeDigitAdd diag2(.A(values[27:24]), .B(values[19:16]),
295                            .C(values[11:8]), .Sum(d2));
296 
297 
298     // compare sums to see whether they are all equal; set equal accordingly
299     FullComparator fc0(.r1(magic_constant), .r2, .r3, .c1, .c2,
300                        .c3, .d1, .d2, .equal);
301 
302 endmodule: EquivalenceChecker
303 
304 // test EquivalenceChecker
305 // module EquivalenceChecker_test
306 //     ();
307 
308 //     // create internal wires, instantiate DUT
309 //     logic [35:0] values;
310 //     logic [7:0] magic_constant;
311 //     logic equal;
312 //     EquivalenceChecker ec0(.values, .magic_constant, .equal);
313 
314 //     // test desired cases
315 //     initial begin
316 
317 //         // start monitor to see results
318 //         $monitor("values: %b, magic constant: %b, equal: %b",
319 //                  values, magic_constant, equal);
320 
321 //         // test case 1: 0000_0000_0000_0000_0000_0000_0000_0000_0000
322 //         // expect to see magic_constant 0, equal 1 (even if invalid)
323 //         #10 values = 36'b0000_0000_0000_0000_0000_0000_0000_0000_0000;
324 
325 //         // test case 2: 0100_1001_0010_0011_0101_0111_1000_0001_0110
326 //         // expect to see magic_constant 1111, equal 1
327 //         #10 values = 36'b0100_1001_0010_0011_0101_0111_1000_0001_0110;
328 
329 //     end
330 
331 // endmodule: EquivalenceChecker_test
332 
333 
334 // validate a single cell's BCD input
335 module OneDigitInputValidator
336     (input  logic [3:0]D,
337      input  logic [8:0] found_in,
338      output logic [8:0] found_out);
339 
340     always_comb begin
341 
342         // check if a previous value was invalid
343 
344         // depending on the value of D, check whether it has been seen before
345         // if so, make input invalid; otherwise, write that it has been seen
346         //case (invalid_in)
347         case (D)
348             4'd1: begin
349                 if (found_in[0] == 1) begin
350                     found_out=found_in;
351                 end
352                 else begin
353                     found_out = found_in+9'b000_000_001;
```

```
354                 end
355                     //$display("invalid_out1=%b", invalid_out);
356             end
357             4'd2: begin
358                 if (found_in[1] == 1) begin
359                     found_out=found_in;
360                 end
361                 else begin
362                     found_out = found_in+9'b000_000_010;
363                 end
364                 //$display("invalid_out2=%b", invalid_out);
365             end
366             4'd3: begin
367                 if (found_in[2] == 1)begin
368                     found_out=found_in;
369                 end
370                 else begin
371                     found_out = found_in+9'b000_000_100;
372                 end
373                 //$display("invalid_out3=%b", invalid_out);
374             end
375             4'd4: begin
376                 if (found_in[3] == 1)begin
377                     found_out=found_in;
378                 end
379                 else begin
380                     found_out = found_in+9'b000_001_000;
381                 end
382                 //$display("invalid_out4=%b", invalid_out);
383             end
384             4'd5: begin
385                 if (found_in[4] == 1)begin
386                     found_out=found_in;
387                 end
388                 else begin
389                     found_out = found_in+9'b000_010_000;
390                 end
391                 //$display("invalid_out5=%b", invalid_out);
392             end
393             4'd6: begin
394                 if (found_in[5] == 1)begin
395                     found_out=found_in;
396                 end
397                 else begin
398                     found_out = found_in+9'b000_100_000;
399                 end
400                 //$display("invalid_out6=%b", invalid_out);
401             end
402             4'd7: begin
403                 if (found_in[6] == 1)begin
404                     found_out=found_in;
405                 end
406                 else begin
407                     found_out = found_in+9'b001_000_000;
408                 end
409                 //$display("invalid_out7=%b", invalid_out);
410             end
411             4'd8: begin
412                 if (found_in[7] == 1)begin
413                     found_out=found_in;
414                 end
415                 else begin
416                     found_out = found_in+9'b010_000_000;
417                 end
418                 //$display("invalid_out8=%b", invalid_out);
419             end
420             4'd9: begin
421                 if (found_in[8] == 1)begin
422                     found_out=found_in;
423                 end
424                 else begin
```

```
425                     found_out = found_in+9'b100_000_000;
426                 end
427                 //$display("invalid_out9=%b", invalid_out);
428             end
429             default: begin
430                 found_out = found_in;
431                 //$display("default, invalid_out=%b",invalid_out);
432             end
433         endcase
434         // begin
435         //     $display("D=%b, invalid_in: %b, invalid_out: %b,
436         //               found_in:%b", D, invalid_in, invalid_out, found_in);
437         // end
438     end
439 
440 
441 endmodule: OneDigitInputValidator
442 
443 // test OneDigitInputValidator
444 // module OneDigitInputValidator_test();
445 //     logic [3:0]D;
446 //     logic invalid_in;
447 //     logic [8:0] found_in;
448 //     logic invalid_out;
449 //     logic [8:0] found_out;
450 //     OneDigitInputValidator ODIV(.D(D),.invalid_in(invalid_in),
451 //                                 .found_in(found_in),
452 //                                 .invalid_out(invalid_out),
453 //                                 .found_out(found_out));
454 //     initial begin
455 //         $monitor($time,,
456 //         "D=%b,invalid_in=%b,found_in=%b,invalid_out=%b,found_out=%b",
457 //         D,invalid_in,found_in,invalid_out,found_out);
458 //         D=4'b0010;
459 //         invalid_in=1'b0;
460 //         found_in=9'b011_000_000;
461 //     #10 D=4'b0001;
462 //         invalid_in=1'b0;
463 //         found_in=9'b011_000_001;
464 //     #10 D=4'b0001;
465 //         invalid_in=1'b0;
466 //         found_in=9'b111_111_110;
467 //     #10 D=4'b0001;
468 //         invalid_in=1'b1;
469 //         found_in=9'b011_000_000;
470 //     #10 D=4'b0001;
471 //         invalid_in=1'b1;
472 //         found_in=9'b011_000_001;
473 //     #10 D=4'b0001;
474 //         invalid_in=1'b0;
475 //         found_in=9'b000_000_000;
476 //     #10 D=4'b0000;
477 //         invalid_in=1'b0;
478 //         found_in=9'b000_000_000;
479 //     end
480 
481 
482 // endmodule: OneDigitInputValidator_test
483 
484 
485 module InputValidator
486     (input  logic [35:0] values,
487      output logic        invalid);
488 
489     // create wires for one-digit validator chaining
490     logic [8:0] found1, found2, found3, found4, found5, found6, found7,
491                 found8, found9;
492 
493     // chain multiple one-digit validators to validate all values
494     OneDigitInputValidator iv0(
495                                 .D(values[3:0]),
```

```
496                                 .found_in('0),
497                                 .found_out(found1)
498                                );
499
500     OneDigitInputValidator iv1(
501                                 .D(values[7:4]),
502                                 .found_in(found1),
503                                 .found_out(found2)
504                                );
505
506     OneDigitInputValidator iv2(
507                                 .D(values[11:8]),
508                                 .found_in(found2),
509                                 .found_out(found3)
510                                );
511
512     OneDigitInputValidator iv3(
513                                 .D(values[15:12]),
514                                 .found_in(found3),
515                                 .found_out(found4)
516                                );
517
518     OneDigitInputValidator iv4(
519                                 .D(values[19:16]),
520                                 .found_in(found4),
521                                 .found_out(found5)
522                                );
523
524     OneDigitInputValidator iv5(
525                                 .D(values[23:20]),
526                                 .found_in(found5),
527                                 .found_out(found6)
528                                );
529
530     OneDigitInputValidator iv6(
531                                 .D(values[27:24]),
532                                 .found_in(found6),
533                                 .found_out(found7)
534                                );
535
536     OneDigitInputValidator iv7(
537                                 .D(values[31:28]),
538                                 .found_in(found7),
539                                 .found_out(found8)
540                                );
541
542     OneDigitInputValidator iv8(
543                                 .D(values[35:32]),
544                                 .found_in(found8),
545                                 .found_out(found9)
546                                );
547     assign invalid =(found9===9'b111_111_111) ? 0:1;
548
549 endmodule: InputValidator
550
551 // test InputValidator
552 // module InputValidator_test();
553 //     logic [35:0] values;
554 //     logic invalid;
555 //     InputValidator DUT(.values(values),.invalid(invalid));
556 //     initial begin
557 //         $monitor($time,,
558 //         "values=%b, invalid=%b",
559 //         values,invalid);
560 //         values=36'd0;
561 //     #10 values=36'b0001_0010_0011_0100_0101_0110_0111_1000_1001;
562 //     #10 values=36'b0011_1000_0100_0011_0001_1100_1001_0110_0000;
563 //     #10 values=36'b1000_1100_0100_0010_0001_1010_1001_0110_0000;
564 //     #10 values=36'b1000_0100_0010_0001_0011_0111_1111_1001_0000;
565 //     #10 values=36'b1000_1000_1000_1000_1000_1000_1000_1000_1000;
566 //     #10 values=36'b1001_0001_0010_0011_0100_0101_0110_0111_1000;
```

```
//     #10 values=36'b1101_0001_0010_0011_0100_0101_0110_0111_1000;
//     end

// endmodule: InputValidator_test


// full implementation module: combine validate input and compare sum modules
module IsMagic
    (input logic [3:0] num1, num2, num3, //top row, L to R
     input logic [3:0] num4, num5, num6, //middle row
     input logic [3:0] num7, num8, num9, //bottom row
     output logic [7:0] magic_constant, //2 BCD digits
     output logic it_is_magic);

    // define internal logic wires
    logic equal, invalid;

    // make input numbers --> values
    logic [35:0] values;
    assign values = ({num1, num2, num3, num4, num5, num6, num7, num8, num9});

    // instantiate EquivalenceChecker and InputValidator and connect them
    EquivalenceChecker ec0(.values, .equal, .magic_constant);
    InputValidator iv0(.values, .invalid);

    // setup the logic for whether it is magic or not
    assign it_is_magic = (~invalid) & (equal);

endmodule: IsMagic

// test IsMagic
module IsMagic_test
    ();  // no inputs and outputs; using bench style 2 (self-contined)

    // create internal wires, instantiate DUT
        logic [3:0] num1, num2, num3, num4, num5, num6, num7, num8, num9;
        logic [7:0] magic_constant;
        logic it_is_magic;
        IsMagic im0(.num1, .num2, .num3,
                    .num4, .num5, .num6,
                    .num7, .num8, .num9, .magic_constant, .it_is_magic);

        // test desired cases
        initial begin

            // start monitor to see results
            $monitor($time,
                     " %b_%b_%b_%b_%b_%b_%b_%b_%b, constant: %b, magic: %b",
                     num1, num2, num3,
                     num4, num5, num6,
                     num7, num8, num9,
                     magic_constant, it_is_magic);

            // test case 1: 6 1 8  7 5 3  2 9 4
            // expect to see magic_constant 1111, it_is_magic 1
            #10 num1 = 4'd6;
                num2 = 4'd1;
                num3 = 4'd8;
                num4 = 4'd7;
                num5 = 4'd5;
                num6 = 4'd3;
                num7 = 4'd2;
                num8 = 4'd9;
                num9 = 4'd4;

            // test case 2: 8 1 6  3 5 7  4 9 2
            // expect to see magic_constant 1111, it_is_magic 1
            #10 num1 = 4'd8;
                num2 = 4'd1;
                num3 = 4'd6;
                num4 = 4'd3;
```

```
638                 num5 = 4'd5;
639                 num6 = 4'd7;
640                 num7 = 4'd4;
641                 num8 = 4'd9;
642                 num9 = 4'd2;
643 
644             // test case 3: 6 7 2  1 5 9  8 3 4
645             // expect to see magic_constant 1111, it_is_magic 1
646             #10 num1 = 4'd6;
647                 num2 = 4'd7;
648                 num3 = 4'd2;
649                 num4 = 4'd1;
650                 num5 = 4'd5;
651                 num6 = 4'd9;
652                 num7 = 4'd8;
653                 num8 = 4'd3;
654                 num9 = 4'd4;
655 
656             // test case 4: 8 3 4  1 5 9  6 7 2
657             // expect to see magic_constant 1111, it_is_magic 1
658             #10 num1 = 4'd8;
659                 num2 = 4'd3;
660                 num3 = 4'd4;
661                 num4 = 4'd1;
662                 num5 = 4'd5;
663                 num6 = 4'd9;
664                 num7 = 4'd6;
665                 num8 = 4'd7;
666                 num9 = 4'd2;
667 
668             // test case 5: 2 7 6  9 5 1  4 3 8
669             // expect to see magic_constant 1111, it_is_magic 1
670             #10 num1 = 4'd2;
671                 num2 = 4'd7;
672                 num3 = 4'd6;
673                 num4 = 4'd9;
674                 num5 = 4'd5;
675                 num6 = 4'd1;
676                 num7 = 4'd4;
677                 num8 = 4'd3;
678                 num9 = 4'd8;
679 
680             // test case 6: 4 3 8  9 5 1  2 7 6
681             // expect to see magic_constant 1111, it_is_magic 1
682             #10 num1 = 4'd4;
683                 num2 = 4'd3;
684                 num3 = 4'd8;
685                 num4 = 4'd9;
686                 num5 = 4'd5;
687                 num6 = 4'd1;
688                 num7 = 4'd2;
689                 num8 = 4'd7;
690                 num9 = 4'd6;
691 
692             // test case 7: 82 9 4  7 5 3  6 1 8
693             // expect to see magic_constant 1111, it_is_magic 1
694             #10 num1 = 4'd2;
695                 num2 = 4'd9;
696                 num3 = 4'd4;
697                 num4 = 4'd7;
698                 num5 = 4'd5;
699                 num6 = 4'd3;
700                 num7 = 4'd6;
701                 num8 = 4'd1;
702                 num9 = 4'd8;
703 
704             // test case 8: 4 9 2  3 5 7  8 1 6
705             // expect to see magic_constant 1111, it_is_magic 1
706             #10 num1 = 4'd4;
707                 num2 = 4'd9;
708                 num3 = 4'd2;
```

```
709                 num4 = 4'd3;
710                 num5 = 4'd5;
711                 num6 = 4'd7;
712                 num7 = 4'd8;
713                 num8 = 4'd1;
714                 num9 = 4'd6;
715 
716             // test case 9: edge case; all 5's
717             // expect to see magic_constant 15 (1111), it_is_magic 0
718             #10 num1 = 4'd5;
719                 num2 = 4'd5;
720                 num3 = 4'd5;
721                 num4 = 4'd5;
722                 num5 = 4'd5;
723                 num6 = 4'd5;
724                 num7 = 4'd5;
725                 num8 = 4'd5;
726                 num9 = 4'd5;
727 
728             // test case 10: edge case; all 0's
729             // expect to see magic_constant 0, it_is_magic 0
730             #10 num1 = 4'd0;
731                 num2 = 4'd0;
732                 num3 = 4'd0;
733                 num4 = 4'd0;
734                 num5 = 4'd0;
735                 num6 = 4'd0;
736                 num7 = 4'd0;
737                 num8 = 4'd0;
738                 num9 = 4'd0;
739 
740             // test case 11: test case 2, -1 in all spots
741             // expect to see magic_constant 12 (1100), it_is_magic 0
742             #10 num1 = 4'd7;
743                 num2 = 4'd0;
744                 num3 = 4'd5;
745                 num4 = 4'd2;
746                 num5 = 4'd4;
747                 num6 = 4'd6;
748                 num7 = 4'd3;
749                 num8 = 4'd8;
750                 num9 = 4'd1;
751 
752             // test case 12: test case 2, +1 in all spots
753             // expect to see magic_constant 15 (1111 overfow), it_is_magic 0
754             #10 num1 = 4'd9;
755                 num2 = 4'd2;
756                 num3 = 4'd7;
757                 num4 = 4'd4;
758                 num5 = 4'd6;
759                 num6 = 4'd8;
760                 num7 = 4'd5;
761                 num8 = 4'd10;
762                 num9 = 4'd3;
763 
764             // test case 13: test case 2, swapping 2 random cells
765             // expect to see magic_constant (15), it_is_mgic 0
766             #10 num1 = 4'd8;
767                 num2 = 4'd1;
768                 num3 = 4'd6;
769                 num4 = 4'd3;
770                 num5 = 4'd5;
771                 num6 = 4'd7;
772                 num7 = 4'd2; // swapped
773                 num8 = 4'd9;
774                 num9 = 4'd4; // swapped
775 
776 
777             #10 $finish;
778 
779         end
```

```
780
781 endmodule: IsMagic_test
782
783 // helper module for enabling all LEDs if the square is magical
784 module MagicLEDs
785     (input  logic it_is_magic,
786      output logic [7:0] LEDG);
787
788     always_comb begin
789         if (it_is_magic) LEDG = 8'b1111_1111;
790         else LEDG = 8'b0000_0000;
791     end
792
793 endmodule: MagicLEDs
794
795
796 // helper module for displaying a BCD digit via 7-segment display
797 module BCDToSevenSegment
798     (input  logic [3:0] BCD,
799      output logic [6:0] seg);
800
801     // recall that 7-segment displays are active low; 0's for ON, 1's for OFF
802     always_comb begin
803         case (BCD)
804             4'd0: seg = 7'b100_0000;
805             4'd1: seg = 7'b111_1001;
806             4'd2: seg = 7'b010_0100;
807             4'd3: seg = 7'b011_0000;
808             4'd4: seg = 7'b001_1001;
809             4'd5: seg = 7'b001_0010;
810             4'd6: seg = 7'b000_0010;
811             4'd7: seg = 7'b111_1000;
812             4'd8: seg = 7'b000_0000;
813             4'd9: seg = 7'b001_1000;
814             default: seg = 7'b111_1111;
815         endcase
816     end
817
818
819 endmodule: BCDToSevenSegment
820
821
822 // ChipInterface - interface IsMagic and FPGA IO
823 module ChipInterface
824     (output logic [6:0] HEX7, HEX6, // magic_constant
825                         HEX5, HEX4, HEX3, HEX2, HEX1, HEX0,
826      output logic [7:0] LEDG,
827      input  logic [17:0] SW,
828      input  logic [3:0] KEY,
829      input  logic CLOCK_50); // needed for enter_9_bcd
830
831     logic [3:0] num1, num2, num3,
832                 num4, num5, num6,
833                 num7, num8, num9;
834     logic [7:0] magic_constant;
835     logic it_is_magic;
836     enter_9_bcd e(.entry(SW[3:0]),
837                   .selector(SW[7:4]),
838                   .enableL(KEY[0]),
839                   .zeroL(KEY[2]),
840                   .set_defaultL(KEY[1]),
841                   .clock(CLOCK_50),
842                   .*);
843
844     IsMagic im(.*);
845
846     // setup LEDs to light up if it_is_magic
847     MagicLEDs ml0(.it_is_magic, .LEDG);
848     // assign LEDG = 'b1;
849
850     // output BCD digits into the segment display
```

```
851     BCDToSevenSegment b0(.BCD(magic_constant[3:0]), .seg(HEX6)); // LSB "1's"
852     BCDToSevenSegment b1(.BCD(magic_constant[7:4]), .seg(HEX7)); // MSB "10's"
853
854 endmodule : ChipInterface
855
```

Lab Code [10 points]
Filename: enter_9_bcd.sv
AndrewID: dchan2

```
1 //
2
3 //
4
5 //  Change Log:
6
7 //  Added Synchronization for Buttons - 26 Sep 2023
8
9 //
10
11
12
13 module enter_9_bcd
14
15   (input  logic [3:0] entry,
16
17    input  logic [3:0] selector,
18
19    input  logic       enableL, zeroL, set_defaultL, clock,
20
21    output logic [3:0] num1, num2, num3, num4, num5, num6, num7, num8, num9);
22
23
24
25   logic enableL_async, enableL_sync;
26
27   logic zeroL_async, zeroL_sync;
28
29   logic set_defaultL_async, set_defaultL_sync;
30
31
32
33   // 2FF Synchronization
34
35   always_ff @(posedge clock) begin
36
37     enableL_async      <= enableL;
38
39     enableL_sync       <= enableL_async;
40
41     zeroL_async        <= zeroL;
42
43     zeroL_sync         <= zeroL_async;
44
45     set_defaultL_async <= set_defaultL;
46
47     set_defaultL_sync  <= set_defaultL_async;
48
49   end
50
51
52
53   always_ff @(posedge clock) begin
54
55     if (~zeroL_sync) begin
56
57       num1 <= 4'b0000;
58
59       num2 <= 4'b0000;
60
61       num3 <= 4'b0000;
62
63       num4 <= 4'b0000;
64
65       num5 <= 4'b0000;
66
67       num6 <= 4'b0000;
68
69       num7 <= 4'b0000;
```

```
 70
 71       num8 <= 4'b0000;
 72
 73       num9 <= 4'b0000;
 74
 75     end
 76
 77     else if (~set_defaultL_sync) begin
 78
 79       num1 <= 4'b1000;
 80
 81       num2 <= 4'b0001;
 82
 83       num3 <= 4'b0110;
 84
 85       num4 <= 4'b0011;
 86
 87       num5 <= 4'b0101;
 88
 89       num6 <= 4'b0111;
 90
 91       num7 <= 4'b0100;
 92
 93       num8 <= 4'b1001;
 94
 95       num9 <= 4'b0010;
 96
 97     end
 98
 99     else if (~enableL_sync)
100
101     unique case (selector)
102
103       4'b0001: num1 <= entry;
104
105       4'b0010: num2 <= entry;
106
107       4'b0011: num3 <= entry;
108
109       4'b0100: num4 <= entry;
110
111       4'b0101: num5 <= entry;
112
113       4'b0110: num6 <= entry;
114
115       4'b0111: num7 <= entry;
116
117       4'b1000: num8 <= entry;
118
119       4'b1001: num9 <= entry;
120
121     endcase
122
123   end
124
125
126
127 endmodule: enter_9_bcd
```