```
1   //
2   // Part 5, Basic Seqential Logic Lab
3   // Neil Nie, (c) Nov 7th, 2018
4   //
5   //
6
7   module seven_segment_display (
8
9      in,          // accepts a four bit input
10     out          // returns a six bit output
11  );
12
13  input [3:0] in;   // A B C D
14  output [6:0] out; // F E D C B A
15
16  assign out[6] = (~in[3] & ~in[2] & ~in[1]) + (~in[3] & in[2] & in[1] & in[0]);
17  assign out[5] = (in[3] & in[2] & ~in[1]) + (~in[3] & in[1] & in[0]) + (~in[3] & ~in[2] & in[
    0]) + (~in[3] & ~in[2] & in[1]);
18  assign out[4] = (~in[1] & in[2]) | (in[0]);
19  assign out[3] = (in[2] & in[1] & in[0]) + (~in[3] & in[2] & ~in[1] & ~in[0]) + (in[3] & ~in[
    2] & in[1] & ~in[0]) + (~in[2] & ~in[1] & in[0]);
20  assign out[2] = (in[3] & in[2] & ~in[0]) + (~in[3] & ~in[2] & in[1] & ~in[0]) + (in[3] & in[
    2] & in[1]);
21  assign out[1] = (in[2] & in[1] & ~in[0]) + (in[3] & in[1] & in[0]) + (~in[3] & in[2] & ~in[1
    ] & in[0]) + (in[3] & in[2] & ~in[0]);
22  assign out[0] = (in[2] & ~in[1] & ~in[0]) + (in[3] & in[2] & ~in[1]) + (~in[3] & ~in[2] & ~
    in[1] & in[0]) + (in[3] & ~in[2] & in[1] & in[0]);
23
24  endmodule
25
26  // Gated D Latch
27
28  module gated_d_latch(clk, D, Q);
29
30  input clk, D;
31  output reg Q;
32
33  always @ (D, clk)
34     if (clk)
35        Q = D;
36
37  endmodule
38
39  // Positive Edge Gated Data Flip-Flop
40
41  module ped_flip_flop(clk, D, Q);
42
43  input D, clk;
44  output Q;
45
46  wire Qm, Qs;
47
48  gated_d_latch latch_1(~clk, D, Qm);
49  gated_d_latch latch_2(clk, Qm, Qs);
50
51  assign Q = Qs;
52
53  endmodule
54
55  // Part 5
56
57  module Part5(
58     SW, KEY,
59     HEX0, HEX1, HEX2, HEX3,
60     HEX4, HEX5, HEX6, HEX7
61  );
62
63  input [15:0] SW;
64  input [1:0] KEY;  // key[0] is clock, key[1] is reset
65  output [6:0] HEX0, HEX1, HEX2, HEX3;
66  output [6:0] HEX4, HEX5, HEX6, HEX7;
67
68  reg [15:0] input1, input2;
69  reg clk, key_state;
70  wire [15:0] latch_out;
71  initial key_state <= 0;
```

```
72    initial clk <= 0;
73    initial input2 <= 16'b0;
74
75    always @ (posedge KEY[0]) begin
76       key_state <= 1;
77    end
78
79    always @ (SW or KEY[1]) begin
80
81       if (KEY[1] == 0) begin
82          input1 <= 16'b0;
83       end else begin
84          input1 <= SW;
85       end
86
87       if (key_state == 1)
88          input2 <= SW;
89    end
90
91    // simply storing a 16 bit number
92    ped_flip_flop bit0(KEY[0], input1[0], latch_out[0]);
93    ped_flip_flop bit1(KEY[0], input1[1], latch_out[1]);
94    ped_flip_flop bit2(KEY[0], input1[2], latch_out[2]);
95    ped_flip_flop bit3(KEY[0], input1[3], latch_out[3]);
96    ped_flip_flop bit4(KEY[0], input1[4], latch_out[4]);
97    ped_flip_flop bit5(KEY[0], input1[5], latch_out[5]);
98    ped_flip_flop bit6(KEY[0], input1[6], latch_out[6]);
99    ped_flip_flop bit7(KEY[0], input1[7], latch_out[7]);
100   ped_flip_flop bit8(KEY[0], input1[8], latch_out[8]);
101   ped_flip_flop bit9(KEY[0], input1[9], latch_out[9]);
102   ped_flip_flop bit10(KEY[0], input1[10], latch_out[10]);
103   ped_flip_flop bit11(KEY[0], input1[11], latch_out[11]);
104   ped_flip_flop bit12(KEY[0], input1[12], latch_out[12]);
105   ped_flip_flop bit13(KEY[0], input1[13], latch_out[13]);
106   ped_flip_flop bit14(KEY[0], input1[14], latch_out[14]);
107   ped_flip_flop bit15(KEY[0], input1[15], latch_out[15]);
108   // ==============================
109
110   // first four displays
111   seven_segment_display disp0(latch_out[3:0], HEX0);
112   seven_segment_display disp1(latch_out[7:4], HEX1);
113   seven_segment_display disp2(latch_out[11:8], HEX2);
114   seven_segment_display disp3(latch_out[15:12], HEX3);
115
116   // second four displays
117   seven_segment_display disp4(input2[3:0], HEX4);
118   seven_segment_display disp5(input2[7:4], HEX5);
119   seven_segment_display disp6(input2[11:8], HEX6);
120   seven_segment_display disp7(input2[15:12], HEX7);
121
122   endmodule
123
```