```
1   `timescale 1ns / 1ns // `timescale time_unit/time_precision
2
3   /* top-level entity for the 8-bit counter */
4   module lab5Part1 (input [1:0] SW, input [0:0] KEY, output [9:0] LEDR,
5                     output [6:0] HEX0, HEX1);
6
7               assign LEDR[9:0] = 10'b0000000000; // all LEDs remain off
8               wire [7:0] Q_state; // current state of the counter circuit
9
10              /*
11               * KEY[0] is the clock signal
12               * SW[0] is the clear signal
13               * SW[1] is the enable signal
14               * HEX0, HEX1 are for display of inputs and output
15               * Rest HEX[k] stay off (un-initialized)
16               */
17
18               eightBit_Counter C1 (.clock(~KEY[0]), .clear(SW[0]),
19                                    .enable(SW[1]), .Q(Q_state)); // instantiating 8-bit counter
20
21               BCD_to_HEX_Decoder D1 (.C(Q_state[3:0]), .HEX(HEX0)); // display LSB of Q
22               BCD_to_HEX_Decoder D2 (.C(Q_state[7:4]), .HEX(HEX1)); // display MSB of Q
23
24  endmodule // lab5part1
25
26  /* 8-bit counter using 8 serial t_flip-flops */
27  module eightBit_Counter (input clock, clear, enable, output [7:0] Q);
28
29              wire [7:0] T; // (N-1) = 7-bit bus to track toggle inputs. T[0] not used
30              t_flipFlop T0 (.T(enable), .clock(clock), .clear(clear), .Q(Q[0]), .Q_p1(T[1]));
     // bit 0
31              t_flipFlop T1 (.T(T[1]), .clock(clock), .clear(clear), .Q(Q[1]), .Q_p1(T[2]));
    // bit 1
32              t_flipFlop T2 (.T(T[2]), .clock(clock), .clear(clear), .Q(Q[2]), .Q_p1(T[3]));
    // bit 2
33              t_flipFlop T3 (.T(T[3]), .clock(clock), .clear(clear), .Q(Q[3]), .Q_p1(T[4]));
    // bit 3
34              t_flipFlop T4 (.T(T[4]), .clock(clock), .clear(clear), .Q(Q[4]), .Q_p1(T[5]));
    // bit 4
35              t_flipFlop T5 (.T(T[5]), .clock(clock), .clear(clear), .Q(Q[5]), .Q_p1(T[6]));
    // bit 5
36              t_flipFlop T6 (.T(T[6]), .clock(clock), .clear(clear), .Q(Q[6]), .Q_p1(T[7]));
    // bit 6
37              t_flipFlop T7 (.T(T[7]), .clock(clock), .clear(clear), .Q(Q[7]), .Q_p1(T[0]));
    // bit 7
38
39  endmodule // eightBit_Counter
40
41  /* t-flip flop module with asynchronous clear */
42  module t_flipFlop (input T, clock, clear, output reg Q, Q_p1);
43
44              // triggered on rising edge of the clock signal and falling edge of clear
45              always @(posedge clock, negedge clear)
46              begin
47
48                 if (clear == 1'b0)
49                    Q <= 1'b0; // active-low, asynchronous reset to 0
50                 else if (T == 1'b0)
51                    Q <= Q; // if toggle is 0, t_flip-flop maintains state
52                 else
53                    Q <= ~Q; // if toggle is 1, t_flip-flop changes state to T'
54
55                 Q_p1 = Q & T; // T for the next t_flip-flop
56
57              end
58
59  endmodule // t_flipFlop
60
61
62  /* BCD to common-anode seven-segment display decoder */
63  module BCD_to_HEX_Decoder (input [3:0] C, output [6:0] HEX);
64
65              // maxterms for every segment LEDs with common anode
66              assign HEX[0] = !((C[3]|C[2]|C[1]|!C[0]) & (C[3]|!C[2]|C[1]|C[0]) &
67                           (!C[3]|C[2]|!C[1]|!C[0]) & (!C[3]|!C[2]|C[1]|!C[0]));
68
```

```
69                assign HEX[1] = !((C[3]||!C[2]|C[1]||!C[0]) & (C[3]||!C[2]||C[1]|C[0]) &
70                                  (!C[3]|C[2]||!C[1]||!C[0]) & (!C[3]||C[2]|C[1]|C[0]) &
71                                  (!C[3]||!C[2]||C[1]|C[0]) & (!C[3]||C[2]||!C[1]||!C[0]));
72
73                assign HEX[2] = !((C[3]|C[2]||!C[1]|C[0]) & (!C[3]||!C[2]|C[1]|C[0]) &
74                                  (!C[3]||!C[2]||!C[1]|C[0]) & (!C[3]||C[2]||!C[1]||!C[0]));
75
76                assign HEX[3] = !((C[3]|C[2]|C[1]||!C[0]) & (C[3]||!C[2]|C[1]|C[0]) &
77                                  (C[3]||!C[2]||!C[1]||!C[0]) & (!C[3]|C[2]|C[1]||!C[0]) &
78                                  (!C[3]|C[2]||!C[1]|C[0]) & (!C[3]||!C[2]||!C[1]||!C[0]));
79
80                assign HEX[4] = !((C[3]|C[2]|C[1]||!C[0]) & (C[3]|C[2]||!C[1]||!C[0]) &
81                                  (C[3]||!C[2]|C[1]|C[0]) & (C[3]||!C[2]|C[1]|C[0]) &
82                                  (C[3]||!C[2]|C[1]||!C[0]) & (C[3]||!C[2]||!C[1]||!C[0]) &
83                                  (!C[3]|C[2]|C[1]||!C[0]));
84
85                assign HEX[5] = !((C[3]|C[2]|C[1]||!C[0]) & (C[3]|C[2]||!C[1]|C[0]) &
86                                  (C[3]|C[2]||!C[1]||!C[0]) & (C[3]||!C[2]||!C[1]||!C[0]) &
87                                  (!C[3]||!C[2]|C[1]||!C[0]));
88
89                assign HEX[6] = !((C[3]|C[2]|C[1]|C[0]) & (C[3]|C[2]|C[1]||!C[0]) &
90                                  (C[3]||!C[2]||!C[1]||!C[0]) & (!C[3]||!C[2]|C[1]|C[0]));
91
92    endmodule // BCD_to_HEX_Decoder
```