```
1   `timescale 1ns / 1ns // `timescale time_unit/time_precision
2
3   /* top-level entity for the speed controlled counter */
4   module lab5Part2 (input [2:0] SW, input CLOCK_50, output [9:0] LEDR, output [6:0] HEX0);
5
6               assign LEDR[9:0] = 10'b0000000000; // all LEDs remain off
7               wire [4:0] disp_Count;
8
9               speedController S1 (.f_Select(SW[1:0]), .clock(CLOCK_50),
10                                  .reset(SW[2]), .countState(disp_Count));
11              BCD_to_HEX_Decoder D1 (.C(disp_Count[3:0]), .HEX(HEX0));
12
13  endmodule // lab5Part2
14
15  /*  */
16  module speedController (input [1:0] f_Select, input clock, reset,
17                          output [4:0] countState);
18
19              wire [25:0] m_Cycles; // to connect max cycles.
20              wire dwnClk_Enable; // downclocked, synchronous enable for counter
21
22              speedSelect S1 (.sel(f_Select), .maxCycles(m_Cycles));
23              rateDivider R1 (.maxCycles(m_Cycles), .clock(clock), .reset(reset),
24                              .downClock(dwnClk_Enable));
25              fourBit_Counter C1 (.clock(clock), .reset(reset),
26                                  .enable(dwnClk_Enable), .Q(countState));
27
28  endmodule // speedController
29
30  /**/
31  module speedSelect (input [1:0] sel, output reg [25:0] maxCycles);
32
33              always @(*)
34              begin
35
36                 case(sel)
37                    2'b00: maxCycles = 26'd1; // 50MHz
38                    2'b01: maxCycles = 26'd12500000; // 4Hz
39                    2'b10: maxCycles = 26'd25000000; // 2Hz
40                    2'b11: maxCycles = 26'd50000000; // 1Hz
41                    default: maxCycles = 26'd50000000; // dwef
42                 endcase
43
44              end
45
46  endmodule // speedSelect
47
48  /* downclocks input 50 MHz clock for feeding into other modules */
49  module rateDivider(input [25:0] maxCycles, input clock, reset, output downClock);
50
51              reg [25:0] cycleCount;
52
53              always @(posedge clock) // triggered on edges of clock
54              begin
55
56                 if (reset == 1'b0) // synchronous active -low
57                    cycleCount <= 26'd0;
58                 else if (cycleCount == 26'd0)
59                    cycleCount <= maxCycles; // reset counter to 50M
60                 else
61                    cycleCount <= cycleCount - 1'b1; // decrement state
62
63              end
64
65              assign downClock = (cycleCount == 26'd0) ? (1'b1):(1'b0);
66
67  endmodule // rateDivider
68
69  /* 4-bit counter to which accepts the downclocked enable */
70  module fourBit_Counter (input clock, reset, enable, output reg [4:0] Q);
71
72              always @(posedge clock) // triggered on rising edge of clock
73              begin
74
75                 if (reset == 1'd0) // synch reset active-low
76                    Q <= 5'd0;
```

```
77                  else if (Q == 5'd16) // max vvalll
78                     Q <= 5'd0;
79                  else if (enable == 1'd1) // increment on enable
80                     Q <= Q + 1;
81                  else
82                     Q <= Q;
83
84               end
85
86   endmodule // fourBit_Counter
87
88   /* BCD to common-anode seven-segment display decoder */
89   module BCD_to_HEX_Decoder (input [3:0] C, output [6:0] HEX);
90
91               // maxterms for every segment LEDs with common anode
92               assign HEX[0] = !((C[3]|C[2]|C[1]|!C[0]) & (C[3]|!C[2]|C[1]|C[0]) &
93                           (!C[3]|C[2]|!C[1]|!C[0]) & (!C[3]|!C[2]|C[1]|!C[0]));
94
95               assign HEX[1] = !((C[3]|!C[2]|C[1]|!C[0]) & (C[3]|!C[2]|!C[1]|C[0]) &
96                           (!C[3]|C[2]|!C[1]|!C[0]) & (!C[3]|!C[2]|C[1]|C[0]) &
97                           (!C[3]|!C[2]|!C[1]|C[0]) & (!C[3]|!C[2]|!C[1]|!C[0]));
98
99               assign HEX[2] = !((C[3]|C[2]|!C[1]|C[0]) & (!C[3]|!C[2]|C[1]|C[0]) &
100                          (!C[3]|!C[2]|!C[1]|C[0]) & (!C[3]|!C[2]|!C[1]|!C[0]));
101
102              assign HEX[3] = !((C[3]|C[2]|C[1]|!C[0]) & (C[3]|!C[2]|C[1]|C[0]) &
103                          (C[3]|!C[2]|!C[1]|!C[0]) & (!C[3]|C[2]|C[1]|!C[0]) &
104                          (!C[3]|C[2]|!C[1]|C[0]) & (!C[3]|!C[2]|!C[1]|!C[0]));
105
106              assign HEX[4] = !((C[3]|C[2]|C[1]|!C[0]) & (C[3]|C[2]|!C[1]|!C[0]) &
107                          (C[3]|!C[2]|C[1]|C[0]) & (C[3]|!C[2]|C[1]|C[0]) &
108                          (C[3]|!C[2]|C[1]|!C[0]) & (C[3]|!C[2]|!C[1]|!C[0]) &
109                          (!C[3]|C[2]|C[1]|!C[0]));
110
111              assign HEX[5] = !((C[3]|C[2]|C[1]|!C[0]) & (C[3]|C[2]|!C[1]|C[0]) &
112                          (C[3]|C[2]|!C[1]|!C[0]) & (C[3]|!C[2]|!C[1]|!C[0]) &
113                          (!C[3]|!C[2]|C[1]|!C[0]));
114
115              assign HEX[6] = !((C[3]|C[2]|C[1]|C[0]) & (C[3]|C[2]|C[1]|!C[0]) &
116                          (C[3]|!C[2]|!C[1]|!C[0]) & (!C[3]|!C[2]|C[1]|C[0]));
117
118  endmodule // BCD_to_HEX_Decoder
```