```
1   /*------------------Top-Level------------------*/
2
3   /*
4      16-bit adder with 4 CLA_4b instances. This module does not incorporate a carry-out
    signal, given that it is solely used in
5      Carry-lookahead adders of higher degree of hierarchy. This also warrants the need of 2nd
    order propagate/generate signals as outputs.
6   */
7
8   module adder_16b (
9      input cin, // Input carry
10     input [15:0] x, y, // Summands
11     output Ppp, Gpp, // 2nd order propagate/generate
12     output [15:0] s // output sum
13     );
14
15     // carry signal for each 4-bit sub-adder. We use 'h' to denote a 'hierarchical' carry.
16     wire [3:0] hc;
17     assign hc[0] = cin;
18
19     // 'hierachical' Generate and Propagate signals.
20     wire [3:0] hP, hG;
21
22     // 4 CLA_4b instances for each 4-bit subset of x and y.
23     genvar i;
24     generate
25     for (i=0; i<4; i = i+1) begin : subadders
26        CLA_4b subadder (hc[i], x[4*i+3: 4*i], y[4*i+3: 4*i], hP[i], hG[i], s[4*i+3: 4*i]);
27     end
28     endgenerate
29
30     // Hierarchical carries according to the lookahead framework.
31     assign hc[1] = hG[0] | hP[0] & cin;
32     assign hc[2] = hG[1] | hP[1] & hG[0] | hP[1] & hP[0] & cin;
33     assign hc[3] = hG[2] | hP[2] & hG[1] | hP[2] & hP[1] & hG[0] | hP[2] & hP[1] & hP[0] &
    cin;
34     // hc[4] not necessary since adder_16b is itself a subadder.
35
36     assign Ppp = hP[3] & hP[2] & hP[1] & hP[0];
37     assign Gpp = hG[3] | hP[3] & hG[2] | hP[3] & hP[2] & hG[1] | hP[3] & hP[2] & hP[1] & hG[0
    ];
38
39  endmodule
40
41  /*------------------Mid-Level------------------*/
42
43  // 4-bit Carry-Lookahead block used in the top-level hierarchy.
44  module CLA_4b (
45     input cin,
46     input [3:0] x, y,
47     output Pp, Gp, // 1st order propagate/generate.
48     output [3:0] s
49     );
50
51     // Carry signal for each bit stage
52     wire [3:0] c;
53     assign c[0] = cin;
54
55     // Generate and Propagate signals for each bit stage
56     wire [3:0] P, G;
57
58     // bcell instances for each bit stage.
59     genvar i;
60     generate
61     for (i=0; i<4; i = i+1) begin : bcells
62        bcell BC (x[i], y[i], c[i], s[i], P[i], G[i]);
63     end
64     endgenerate
65
66     // Verbose expressions for each carry are required to minimize gate delays from fitter's
    perspective.
67     // Compare with c[i+1] = G[i] | P[i] & c[i] in RTL viewer.
68     assign c[1] = G[0] | P[0] & cin;
69     assign c[2] = G[1] | P[1] & G[0] | P[1] & P[0] & cin;
```

```
70       assign c[3] = G[2] | P[2] & G[1] | P[2] & P[1] & G[0] | P[2] & P[1] & P[0] & cin;
71
72       // The 2nd level CL hierarchy will produce c[4] carries under the 'lookahead' framework
    - no need to derive internally.
73       // using Pp and Gp (P', G') below
74       assign Pp = P[3] & P[2] & P[1] & P[0];
75       assign Gp = G[3] | P[3] & G[2] | P[3] & P[2] & G[1] | P[3] & P[2] & P[1] & G[0];
76
77   endmodule
78
79   /*--------------------Bottom-Level--------------------*/
80
81   // Bit cell producing Generate and Propagate signals for each xi, yi, ci tuple as presented
    in lecture.
82   module bcell (
83       input xi, yi, ci,
84       output si, Pi, Gi
85       );
86
87       assign Pi = xi ^ yi;
88       assign si = Pi ^ ci;
89       assign Gi = xi & yi;
90
91   endmodule
92
93
94   /*--------------------Top-Level Testbench--------------------*/
95
96   module adder_16b_testbench();
97
98       reg cin;
99       reg [15:0] x, y;
100      wire Ppp, Gpp;
101      wire [15:0] s;
102
103      adder_16b dut (cin, x, y, Ppp, Gpp, s);
104
105      // Further testcases were validated during the design phase, but only three edge cases
    were kept for submission clarity.
106      initial begin
107          // Test 1: Small positive values
108          cin = 0; x = 16'b0000_0000_0000_0101; y = 16'b0000_0000_0000_1011; #10;
109          $display("Test 1: cin = %b, x = %b, y = %b, s = %b, Ppp = %b, Gpp = %b", cin, x, y, s,
     Ppp, Gpp);
110
111          // Test 2: Mixed range values
112          cin = 0; x = 16'b0000_1000_0000_1111; y = 16'b0001_0111_1100_1100; #10;
113          $display("Test 2: cin = %b, x = %b, y = %b, s = %b, Ppp = %b, Gpp = %b", cin, x, y, s,
     Ppp, Gpp);
114
115          // Test 3: Maximum 16-bit values
116          cin = 0; x = 16'b1111_1111_1111_1111; y = 16'b1111_1111_1111_1111; #10;
117          $display("Test 3: cin = %b, x = %b, y = %b, s = %b, Ppp = %b, Gpp = %b", cin, x, y, s,
     Ppp, Gpp);
118
119          $stop;
120      end
121  endmodule
122
```