

```
1  //
2  // Digital Logic Lab Part 4
3  // Neil Nie
4  // (C) Yongyang Nie
5  // Oct 31th, 2018
6  //
7
8  module full_adder(
9
10     a,
11     b,
12     out,
13     cin,
14     cout
15 );
16
17 input a;
18 input b;
19 input cin;
20 output out;
21 output cout;
22
23 assign cout = (a & b) | cin & (~a & b | ~b & a);
24 assign out = cin & ~a & ~b | ~cin & ~a & b | cin & a & b | ~cin & a & ~b;
25
26 endmodule
27
28 module four_bit_adder(
29
30     x,
31     y,
32     out,
33     carry
34 );
35
36 input [3:0] x;
37 input [3:0] y;
38 output [4:0] out;
39 output carry;
40
41 wire [3:0] cout;
42
43 full_adder(x[0], y[0], out[0], 0, cout[0]);
44 full_adder(x[1], y[1], out[1], cout[0], cout[1]);
45 full_adder(x[2], y[2], out[2], cout[1], cout[2]);
46 full_adder(x[3], y[3], out[3], cout[2], cout[3]);
47 assign out[4] = cout[3];
48
49 endmodule
50
51 // this display only shows one or nothing
52
53 module half_display(
54
55     in,          // accepts a four bit input
56     out          // returns a six bit output
57 );
58
59 input in;
60 output [6:0] out; // D C B A G
61
62
63 assign out[6] = 1;
64 assign out[5] = 1;
65 assign out[4] = 1;
66 assign out[3] = 1;
67 assign out[2] = ~in;
68 assign out[1] = ~in;
69 assign out[0] = 1;
70
71 endmodule
72
73 // a full full seven segment display module.
74
75 module full_display(
76     in,          // accepts a four bit input
```

```

77     out          // returns a six bit output
78 );
79
80 input [3:0] in;    // A B C D
81 output [6:0] out;  // F E D C B A
82
83 assign out[6] = (~in[3] & ~in[2] & ~in[1]) + (~in[3] & in[2] & in[1] & in[0]);
84 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]);
85 assign out[4] = (~in[1] & in[2]) | (in[0]);
86 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]);
87 assign out[2] = (in[3] & in[2] & ~in[0]) + (~in[3] & ~in[2] & in[1] & ~in[0]) + (in[3] & in[
2] & in[1]);
88 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]);
89 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]);
90
91 endmodule
92
93 module two_one_four_bit_mux(a, b, sel, out);
94
95 input [3:0] a;
96 input [3:0] b;
97 output [3:0] out;
98 input sel;
99
100 assign out[3] = (~sel & a[3]) | (sel & b[3]);
101 assign out[2] = (~sel & a[2]) | (sel & b[2]);
102 assign out[1] = (~sel & a[1]) | (sel & b[1]);
103 assign out[0] = (~sel & a[0]) | (sel & b[0]);
104
105 endmodule
106
107 // returns 1 if in > 9
108 // returns 0 if in < 9
109
110 module comparator(
111     in,
112     out
113 );
114
115 );
116
117 input [4:0] in;
118 output out;
119
120 assign out = in[4] | in[3] & (in[2] | in[1]);
121
122 endmodule
123
124 // this converter module convert number > 9
125 // to the appropriate number
126
127 module converter(
128     in,
129     out
130 );
131
132 );
133
134 input [3:0] in;
135 output [3:0] out;
136
137 assign out[3] = (in[3] & ~in[2] & ~in[1]);
138 assign out[2] = (in[2] & in[1]) | (~in[3] & ~in[2] & ~in[1] & ~in[0]);
139 assign out[1] = (in[3] & in[2] & ~in[1]) | (~in[3] & in[2] & in[1]) | (~in[3] & ~in[2] & ~in
[1] & ~in[0]);
140 assign out[0] = (in[3] & in[0]) | (in[2] & in[1] & in[0]);
141
142 endmodule
143
144 // this is the display unit for part 4.
145 // the module is nearly identitcal to part 3.
146 /* Parameters

```

```
147
148     in: four bit binary input
149     hex0: display 'in' if in < 9
150           display 10 - 'in' if in > 9
151     hex1: display nothing when in < 9
152           display 1 when in > 9
153
154 */
155 module display_unit(
156     in,
157     hex0,
158     hex1
159 );
160
161 input  [4:0] in;
162 output [6:0] hex0;
163 output [6:0] hex1;
164
165 // declare internal variables
166 wire    compare_out;
167 wire [3:0] convert_out;
168 wire [3:0] mux_out;
169
170 comparator(in, compare_out);
171 converter(in[3:0], convert_out); // you only need the last four bits of the input
172
173 two_one_four_bit_mux(in[3:0], convert_out, compare_out, mux_out);
174 full_display(mux_out, hex0);
175 half_display(compare_out, hex1);
176
177 endmodule
178
179 // -----
180
181 module Part4(
182     SW,
183     HEX0,
184     HEX1
185 );
186
187 input  [7:0] SW;
188 output [6:0] HEX0;
189 output [6:0] HEX1;
190
191 // assign x & y
192 wire [3:0] x;
193 wire [7:4] y;
194 wire [4:0] sum;
195 wire [6:0] display0;
196 wire [6:0] display1;
197
198 assign x = SW[3:0];
199 assign y = SW[7:4];
200
201 four_bit_adder(x, y, sum);
202
203 display_unit(sum, display0, display1);
204
205 assign HEX0 = display0;
206 assign HEX1 = display1;
207
208 endmodule
209
210
211
212
```