module comp(

input [5:0] ai,bi,

output logic [5:0] ao,bo

);

always\_comb begin : comp\_module

ao = (ai < bi) ? ai : bi;

bo = (ai < bi) ? bi : ai;

end

endmodule

module Sort (

input [5:0] in\_num0, in\_num1, in\_num2, in\_num3, in\_num4,

output [5:0] out\_num0, out\_num1, out\_num2, out\_num3, out\_num4

);

//inter\_wire[layer][row]

logic [5:0] inter\_wire [4:0][4:0];

// 1st layer

comp c1(in\_num0, in\_num1, inter\_wire[0][0],inter\_wire[0][1]);

comp c2(in\_num2, in\_num3, inter\_wire[0][2],inter\_wire[0][3]);

assign inter\_wire[0][4] = in\_num4;

// 2nd layer

assign inter\_wire[1][0] = inter\_wire[0][0];

comp c3(inter\_wire[0][1], inter\_wire[0][2], inter\_wire[1][1],inter\_wire[1][2]);

comp c4(inter\_wire[0][3], inter\_wire[0][4], inter\_wire[1][3], inter\_wire[1][4]);

// 3nd layer

assign inter\_wire[2][4] = inter\_wire[1][4];

comp c5(inter\_wire[1][0], inter\_wire[1][1], inter\_wire[2][0],inter\_wire[2][1]);

comp c6(inter\_wire[1][2], inter\_wire[1][3], inter\_wire[2][2], inter\_wire[2][3]);

// 4nd layer

assign inter\_wire[3][0] = inter\_wire[2][0];

comp c7(inter\_wire[2][1], inter\_wire[2][2], inter\_wire[3][1],inter\_wire[3][2]);

comp c8(inter\_wire[2][3], inter\_wire[2][4], inter\_wire[3][3], inter\_wire[3][4]);

// 5th layer

assign inter\_wire[4][4] = inter\_wire[3][4];

comp c9(inter\_wire[3][0], inter\_wire[3][1], inter\_wire[4][0],inter\_wire[4][1]);

comp c10(inter\_wire[3][2], inter\_wire[3][3], inter\_wire[4][2], inter\_wire[4][3]);

// output

assign out\_num0 = inter\_wire[4][0];

assign out\_num1 = inter\_wire[4][1];

assign out\_num2 = inter\_wire[4][2];

assign out\_num3 = inter\_wire[4][3];

assign out\_num4 = inter\_wire[4][4];

endmodule

module SMJ(

// Input signals

hand\_n0,

hand\_n1,

hand\_n2,

hand\_n3,

hand\_n4,

// Output signals

out\_data

);

//---------------------------------------------------------------------

// INPUT AND OUTPUT DECLARATION

//---------------------------------------------------------------------

input [5:0] hand\_n0;

input [5:0] hand\_n1;

input [5:0] hand\_n2;

input [5:0] hand\_n3;

input [5:0] hand\_n4;

output logic [1:0] out\_data;

//---------------------------------------------------------------------

// LOGIC DECLARATION

//---------------------------------------------------------------------

logic [5:0] sorted\_hand[4:0];

logic invalid\_check[4:0];

logic invalid\_term\_exist;

logic tri\_plus\_pair;

logic seq\_plus\_pair;

logic is\_honor [4:0];

logic equal\_01, equal\_24, equal\_34, equal\_02, equal\_13, diff\_01, diff\_12, diff\_23, diff\_34;

logic pair\_01, pair\_34;

logic seq\_012, seq\_234, seq\_01\_34;

logic tri\_012, tri\_123, tri\_234;

//---------------------------------------------------------------------

// Your design

//---------------------------------------------------------------------

// Sort all input

Sort sort\_ins(.in\_num0 (hand\_n0), .in\_num1 (hand\_n1), .in\_num2 (hand\_n2), .in\_num3 (hand\_n3),.in\_num4 (hand\_n4),

.out\_num0 (sorted\_hand[0]), .out\_num1 (sorted\_hand[1]), .out\_num2 (sorted\_hand[2]), .out\_num3 (sorted\_hand[3]), .out\_num4 (sorted\_hand[4]));

always\_comb begin

// these indicate a pair

equal\_01 = sorted\_hand[0]==sorted\_hand[1];

equal\_34 = sorted\_hand[3]==sorted\_hand[4];

// these indicate a triplet

equal\_02 = sorted\_hand[0]==sorted\_hand[2];

equal\_13 = sorted\_hand[1]==sorted\_hand[3];

equal\_24 = sorted\_hand[2]==sorted\_hand[4];

// 01\_12(!is\_honor[2]), 12\_23(!is\_honor[2]), 23\_34(!is\_honor[2]) indicates a seq

diff\_01 = sorted\_hand[1]==sorted\_hand[0]+1;

diff\_12 = sorted\_hand[2]==sorted\_hand[1]+1;

diff\_23 = sorted\_hand[3]==sorted\_hand[2]+1;

diff\_34 = sorted\_hand[4]==sorted\_hand[3]+1;

// Check if input contains invalid terms

for(int i=0;i<5;i++) begin

is\_honor[i] = (sorted\_hand[i][5:4]==2'b00/\*&2'b11\*/);

invalid\_check[i] = (is\_honor[i] ? sorted\_hand[i][3:0]>6 : sorted\_hand[i][3:0]>8);

end

invalid\_term\_exist = invalid\_check[0] || invalid\_check[1] || invalid\_check[2] || invalid\_check[3] || invalid\_check[4] || (equal\_02 & equal\_24);

/\*

// Check tri\_plus\_pair case

tri\_plus\_pair = ((equal\_01) && (equal\_24)) || ((equal\_34) && (equal\_02));

//Check seq\_plus\_pair case

seq\_plus\_pair = (diff\_01 && diff\_12 && equal\_34 && !is\_honor[0])

|| (equal\_01 && diff\_23 && diff\_34 && !is\_honor[2])

|| (equal\_13 && diff\_01 && diff\_34 && !is\_honor[0]);

\*/

pair\_01 = equal\_01;

pair\_34 = equal\_34;

seq\_012 = diff\_01 & diff\_12;

seq\_234 = diff\_23 & diff\_34;

seq\_01\_34 = diff\_01 & diff\_34 & equal\_13;

tri\_012 = equal\_02;

tri\_123 = equal\_13;

tri\_234 = equal\_24;

if(invalid\_term\_exist)begin

out\_data = 2'b01;

end

else if((tri\_012 && pair\_34)||(tri\_234 && pair\_01))begin

out\_data = 2'b11;

end

else if(is\_honor[2])begin

out\_data = 2'b00;

end

else if((pair\_01 && seq\_234)||(seq\_012 && pair\_34)||(seq\_01\_34 &&tri\_123))begin

out\_data = 2'b10;

end

else begin

out\_data = 2'b00;

end

/\*else if(tri\_plus\_pair)begin

out\_data = 2'b11;

end

else if(seq\_plus\_pair)begin

out\_data = 2'b10;

end

else begin

out\_data = 2'b00;

end\*/

end

endmodule