`define MY\_DESIGN

`define GATE\_ADD3

module add\_3\_if\_at\_least\_5(

input [3:0] in,

output [3:0] out);

`ifdef GATE\_ADD3

assign out[0] = !in[0];

assign out[1] = in[1] ~^ in[0];

assign out[2] = (in[0] == 1'b0) ? in[2] ~^ in[1] : !(in[2]);

assign out[3] = in[3]||(in[2]&&(in[1]||in[0]));

`else

if(in>4) begin

assign out = in + 3;

end

else begin

assign out = in;

end

`endif

endmodule

module BCD(

// Input signals

in\_bin,

// Output signals

out\_hundred,

out\_ten,

out\_unit

);

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

// INPUT AND OUTPUT DECLARATION

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

//Input Ports

input [8:0] in\_bin;

//output Ports

output logic [2:0]out\_hundred;

output logic [3:0]out\_ten;

output logic [3:0]out\_unit;

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

// LOGIC DECLARATION

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

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

// Your DESIGN

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

`ifdef MY\_DESIGN

// the wires that will be used to connect diffrent levels of add3 circuits

// 0 is connected to in\_bin while 6 will be connected to the output reg

logic [10:0] inter\_wire [6:0];

assign inter\_wire[0] = {2'b0,in\_bin};

genvar i;

genvar j;

//integer num\_diag;

generate

// the number of diagonal lines

//integer num\_diag = 0;

for(i=3;i<9;i=i+1)begin

// the idx of the inter\_wire element that the add3 module will be connected to

integer input\_idx = i-3;

integer output\_idx = (i-3) + 1;

integer num\_diag = (i/3) -1;

// creating a new diagonal series

if(i%3 == 0)begin

add\_3\_if\_at\_least\_5 add3(.in(inter\_wire[input\_idx][(9- num\_diag) : (9- num\_diag-3)]),.out(inter\_wire[(output\_idx)][(9- num\_diag) : (9- num\_diag-3)]));

end

// lay down next element of the existing diagnal series

//integer j;

for(j=0;j<num\_diag;j=j+1)begin

integer init\_pos = 9 + j;

integer num\_shift = i - 3 \* (j+1);

add\_3\_if\_at\_least\_5 add3(.in(inter\_wire[input\_idx][(init\_pos - num\_shift):(init\_pos - num\_shift - 3)]),.out(inter\_wire[output\_idx][(init\_pos - num\_shift):(init\_pos - num\_shift - 3)]));

end

/\*if(i%3 == 0)begin

num\_diag = num\_diag + 1;

end\*/

end

// assign the val to the output regs

assign out\_unit = inter\_wire[6][3:0];

assign out\_ten = inter\_wire[6][7:4];

assign out\_hundred = inter\_wire[6][10:8];

endgenerate

`else

integer i;

always @(in\_bin) begin

out\_hundred = 0;

out\_ten = 0;

out\_unit = 0;

//genvar i;

for(i=0;i<9;i=i+1)begin

if(out\_hundred > 4)begin

out\_hundred = out\_hundred + 3;

end

if(out\_ten > 4)begin

out\_ten = out\_ten + 3;

end

if(out\_unit > 4)begin

out\_unit = out\_unit + 3;

end

// left shift

out\_hundred = {out\_hundred[1:0], out\_ten[3]};

out\_ten = {out\_ten[2:0], out\_unit[3]};

out\_unit = {out\_unit[2:0], in\_bin[8-i]};

end

end

`endif

endmodule