**Q1) Ripple carry adder using full adders (Using hierarchical modelling) Verilog code with**

**Top level entity-** four\_bit\_ripple\_adder

module half\_add (input a, b, output s, c);

xor g1(s, a, b);

and g2(c, a, b);

endmodule

module full\_add ( input a, b, cin, output s, c);

wire wire1, wire2, wire3;

half\_add ha1 ( a, b, wire1, wire2);

half\_add ha2 ( wire1, cin, s, wire3);

or or1 (c, wire2, wire3);

endmodule

module four\_bit\_ripple\_adder ( input [3:0] a, b, input cin, output [4:0] s);

wire c1, c2, c3;

full\_add fa1 ( a[0], b[0], 1'b0, s[0], c1);

full\_add fa2 ( a[1], b[1], c1, s[1], c2);

full\_add fa3 ( a[2], b[2], c2, s[2], c3);

full\_add fa4 ( a[3], b[3], c3, s[3], s[4]);

endmodule

**Testbench for Ripple carry adder using full adders**

module four\_bit\_ripple\_adder\_test;

reg [3:0] a, b;

reg cin;

wire [4:0] s;

four\_bit\_ripple\_adder gate1 (a, b, cin, s);

integer i;

initial begin

for (i = 0; i < 512; i = i + 1)

begin

{a, b} = i;

#20;

end

end

endmodule

**Q2) 8 – Bit ripple carry adder using full adders Verilog code with Top level entity -** eight\_bit\_ripple\_carry\_adder

module half\_add (input a, b, output s, c);

xor g1(s, a, b);

and g2(c, a, b);

endmodule

module full\_add ( input a, b, cin, output s, c);

wire wire1, wire2, wire3;

half\_add ha1 ( a, b, wire1, wire2);

half\_add ha2 ( wire1, cin, s, wire3);

or or1 (c, wire2, wire3);

endmodule

module eight\_bit\_ripple\_carry\_adder (input [7:0] a, b, input cin, output [8:0] s);

wire c1, c2, c3, c4, c5, c6, c7;

full\_add fa1 ( a[0], b[0], 1'b0, s[0], c1);

full\_add fa2 ( a[1], b[1], c1, s[1], c2);

full\_add fa3 ( a[2], b[2], c2, s[2], c3);

full\_add fa4 ( a[3], b[3], c3, s[3], c4);

full\_add fa5 ( a[4], b[4], c4, s[4], c5);

full\_add fa6 ( a[5], b[5], c5, s[5], c6);

full\_add fa7 ( a[6], b[6], c6, s[6], c7);

full\_add fa8 ( a[7], b[7], c7, s[7], s[8]);

endmodule

**Test bench for 8 – Bit ripple carry adder using full adders**

module eight\_bit\_ripple\_carry\_adder\_test;

reg [7:0] a, b;

reg cin;

wire [8:0] s;

eight\_bit\_ripple\_carry\_adder gate1 (a, b, cin, s);

integer i;

initial begin

for (i = 0; i < 65536; i = i + 1)

begin

{a, b} = i;

#20;

end

end

endmodule

**Q3) BCD adder with four-bit numbers Verilog code with Top level entity-** BCD\_adder

module half\_add (input a, b, output s, c);

xor g1(s, a, b);

and g2(c, a, b);

endmodule

module full\_add ( input a, b, cin, output s, c);

wire wire1, wire2, wire3;

half\_add ha1 ( a, b, wire1, wire2);

half\_add ha2 ( wire1, cin, s, wire3);

or or1 (c, wire2, wire3);

endmodule

module four\_bit\_ripple\_adder ( input [3:0] a, b, input cin, output [3:0] s, output cout);

wire c1, c2, c3;

full\_add fa1 ( a[0], b[0], 1'b0, s[0], c1);

full\_add fa2 ( a[1], b[1], c1, s[1], c2);

full\_add fa3 ( a[2], b[2], c2, s[2], c3);

full\_add fa4 ( a[3], b[3], c3, s[3], cout);

endmodule

module BCD\_adder\_main(input [3:0] a, b, output [3:0] s1, s2);

wire [3:0] si;

wire w1, w2, w3, x;

four\_bit\_ripple\_adder g1 (a, b, 1'b0, si, w3);

and a1 (w1, si[1], si[3]);

and a2 (w2, si[2], si[3]);

or o1 (x, w1, w2, w3);

wire [3:0] wi = {1'b0, x, x, 1'b0};

wire discard;

four\_bit\_ripple\_adder g2 (si, wi, 1'b0, s1, discard);

assign s2 = {1'b0, 1'b0, 1'b0, x};

endmodule

module BCD\_adder (input [3:0] a, b, output reg [3:0] s1, s2);

wire [3:0] wo1, wo2;

BCD\_adder\_main g1 (a, b, wo1, wo2);

always @\* begin

if (a > 4'b1001 || b > 4'b1001)

begin

s1 = 4'bxxxx;

s2 = 4'bxxxx;

end

else

begin

s1 = wo1;

s2 = wo2;

end

end

endmodule

**Test bench for BCD adder**

module BCD\_adder\_test;

reg[3:0] a, b;

wire [3:0] s1, s2;

BCD\_adder g1( a, b, s1, s2);

integer i, j;

initial begin

for (i = 0; i < 16; i = i + 1)

begin

a = i;

for (j = 0; j < 16; j = j + 1)

begin

b = j;

#20;

end

end

end

endmodule

**Q4) BCD to Excess-3 converter Verilog code with Top level entity-** bcd\_to\_excess\_three

module bcd\_to\_excess\_three(input [3:0] bcd, output reg [3:0] ex3);

wire [3:0] wex3;

bcd\_to\_x3 g1 (bcd, wex3);

always @\* begin

if (bcd > 4'b1001)

begin

ex3 = 4'bxxxx;

end

else

begin

ex3 = wex3;

end

end

endmodule

module bcd\_to\_x3(input [3:0] bcd, output [3:0] ex3);

wire w1, w2, w3, w4, w5, b\_, c\_, d\_;

not n1 (b\_, bcd[2]);

not n2 (c\_, bcd[1]);

not n3 (d\_, bcd[0]);

and a1 (w1, b\_, bcd[1]);

and a2 (w2, b\_, bcd[0]);

and a3 (w3, bcd[2], c\_, d\_);

and a4 (w4, bcd[2], bcd[1]);

and a5 (w5, bcd[2], bcd[0]);

or o1 (ex3[3], bcd[3], w4, w5);

or o2 (ex3[2], w1, w2, w3);

xnor xn1 (ex3[1], bcd[1], bcd[0]);

assign ex3[0] = d\_;

endmodule

**Test bench for BCD to excess 3 converter**

module bcd\_to\_excess\_three\_test;

reg [3:0] bcd;

wire [3:0] ex3;

bcd\_to\_excess\_three g1 (bcd, ex3);

integer i = 0;

initial begin

for (i = 0; i < 16; i = i + 1)

begin

{bcd} = i;

#20;

end

end

endmodule

**Q5) Half Subtractor Verilog code with Top level entity-** half\_subtractor

module half\_subtractor(input a, b, output diff, borr);

wire \_b;

xor x1 (diff, a, b);

not n1 (\_b, b);

and a1(borr, a, \_b);

endmodule

**Test bench for half Subtractor**

module half\_subtractor\_test;

reg a, b;

wire diff, borr;

half\_subtractor h1(a, b, diff, borr);

initial begin

a = 0; b = 0;

#10 a = 0; b = 1;

#10 a = 1; b = 0;

#10 a = 1; b = 1;

end

endmodule

**Q6) Full Subtractor Verilog code with Top level entity -** full\_subtractor

module half\_subtractor(input a, b, output diff, borr);

wire \_b;

xor x1 (diff, a, b);

not n1 (\_b, b);

and a1(borr, a, \_b);

endmodule

module full\_subtractor(input a, b, borrin, output diff, borrout);

wire diff1, borr1;

half\_subtractor h1(a, b, diff1, borr1);

half\_subtractor h2(diff1, borrin, diff, borrout);

endmodule

**Test bench for Full Subtractor**

module full\_subtractor\_test;

reg a, b, borrin;

wire diff, borrout;

full\_subtractor f1(a, b, borrin, diff, borrout);

initial begin

borrin = 0;

a = 0; b = 0;

#10 a = 0; b = 1;

#10 a = 1; b = 0;

#10 a = 1; b = 1;

#10;

borrin = 1;

a = 0; b = 0;

#10 a = 0; b = 1;

#10 a = 1; b = 0;

#10 a = 1; b = 1;

end

endmodule