**Q1) A 4-bit adder subtractor Verilog code – Top level entity:** four\_bit\_add\_and\_sub

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\_add\_and\_sub ( input [3:0] a, b, input cin, output [4:0] s, output overflow);

wire c1, c2, c3, x1, x2, x3, x4;

xor xor1 (x1, b[0], cin);

xor xor2 (x2, b[1], cin);

xor xor3 (x3, b[2], cin);

xor xor4 (x4, b[3], cin);

full\_add fa1 ( a[0], x1, cin, s[0], c1);

full\_add fa2 ( a[1], x2, c1, s[1], c2);

full\_add fa3 ( a[2], x3, c2, s[2], c3);

full\_add fa4 ( a[3], x4, c3, s[3], s[4]);

xor xor5 (overflow, c, c3);

endmodule

**Testbench for a 4 bit adder and subtractor**

module four\_bit\_add\_and\_sub\_test;

reg [3:0] a, b;

reg cin;

wire [4:0] s;

wire overflow;

four\_bit\_add\_and\_sub gate1 (a, b, cin, s, overflow);

integer i;

initial begin

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

begin

{a, b, cin} = i;

#20;

end

end

endmodule

**Q2) An 8:1 mux Verilog code with Top level entity -** eight\_by\_one\_mux

module eight\_by\_one\_mux ( input i0, i1, i2, i3, i4, i5, i6, i7, a, b, c, output y);

wire wire0, wire1, wire2, wire3, wire4, wire5, wire6, wire7, na, nb, nc;

not n1 (na, a);

not n2 (nb, b);

not n3 (nc, c);

and a0 ( wire0, na, nb, nc, i0);

and a1 ( wire1, na, nb, c, i1);

and a2 ( wire2, na, b, nc, i2);

and a3 ( wire3, na, b, c, i3);

and a4 ( wire4, a, nb, nc, i4);

and a5 ( wire5, a, nb, c, i5);

and a6 ( wire6, a, b, nc, i6);

and a7 ( wire7, a, b, c, i7);

or or1 (y, wire0, wire1, wire2, wire3, wire4, wire5, wire6, wire7);

endmodule

**Testbench for 8:1 mux**

module eight\_by\_one\_mux\_test;

reg i0, i1, i2, i3, i4, i5, i6, i7, a, b, c;

wire y;

eight\_by\_one\_mux gate1 (i0, i1, i2, i3, i4, i5, i6, i7, a, b, c, y);

integer i;

initial begin

for (i = 0; i < 2048; i = i + 1)

begin

{i0, i1, i2, i3, i4, i5, i6, i7, a, b, c} = i;

#20;

end

end

endmodule

**Q3) A 1:8 demux Verilog code with Top level entity:** one\_by\_eight\_demux

module one\_by\_eight\_demux ( input d, s0, s1, s2, output y0, y1, y2, y3, y4, y5, y6, y7);

wire ns0, ns1, ns2;

not n1 (ns0, s0);

not n2 (ns1, s1);

not n3 (ns2, s2);

and (y0, d, ns0, ns1, ns2);

and (y1, d, ns0, ns1, s2);

and (y2, d, ns0, s1, ns2);

and (y3, d, ns0, s1, s2);

and (y4, d, s0, ns1, ns2);

and (y5, d, s0, ns1, s2);

and (y6, d, s0, s1, ns2);

and (y7, d, s0, s1, s2);

endmodule

**Testbench for 1:8 demux**

module one\_by\_eight\_demux\_test;

reg s0, s1, s2, d;

wire y0, y1, y2, y3, y4, y5, y6, y7;

one\_by\_eight\_demux gate1 (d, s0, s1, s2, y0, y1, y2, y3, y4, y5, y6, y7);

integer i;

initial begin

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

begin

{s0, s1, s2, d} = i;

#20;

end

end

endmodule

**Q4) A 4 bit adder using half adders Verilog code with top level entity:** four\_bit\_half\_adder

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

xor g1(s, a, b);

and g2(c, a, b);

endmodule

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

wire wc1, wc2, wc3, wc4, w2c1, w2c2, w2c3, w2c4, co0, co1, co2, ws0, ws1, ws2, ws3;

half\_add ha1 ( a[0], b[0], ws0, wc1);

half\_add ha2 ( ws0, 1'b0, s[0], w2c1);

or or1 (co0, wc1, w2c1);

half\_add ha3 ( a[1], b[1], ws1, wc2);

half\_add ha4 ( ws1, co0, s[1], w2c2);

or or2 (co1, wc2, w2c2);

half\_add ha5 ( a[2], b[2], ws2, wc3);

half\_add ha6 ( ws2, co1, s[2], w2c3);

or or3 (co2, wc3, w2c3);

half\_add ha7 ( a[3], b[3], ws3, wc4);

half\_add ha8 ( ws3, co2, s[3], w2c4);

or or4 (s[4], wc4, w2c4);

endmodule

**Testbench for a 4 bit adder using half adders**

module four\_bit\_half\_adder\_test;

reg [3:0] a, b;

wire [4:0] s;

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

integer i;

initial begin

for (i = 0; i < 256; i = i + 1)

begin

{a, b} = i;

#20;

end

end

endmodule

**Q5) A 4 bit ripple carry adder Verilog code**

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 a 4 bit ripple carry adder**

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