1. **Full Adder:**

**Behavioral Modelling**

module full\_adder( input a,b,c, output reg sum,cout );

always @(\*)

begin case ({a,b,c})

3'b000: sum = 0;

3'b001: sum = 1;

3'b010: sum = 1;

3'b011: sum = 0;

3'b100: sum = 1;

3'b101: sum = 0;

3'b110: sum = 0;

3'b111: sum = 1;

default : sum = 0;

endcase case ({a,b,c})

3'b000: cout = 0;

3'b001: cout = 0;

3'b010: cout = 0;

3'b011: cout = 1;

3'b100: cout = 0;

3'b101: cout = 1;

3'b110: cout = 1;

3'b111: cout = 1;

default : cout = 0;

endcase

end

endmodule

**Gate level Modelling:**

module full\_adder( input a,b,c, output sum,cout );

wire w1,c1,c2,c3,out1;

xor x1(w1,a,b);

xor x2(sum,w1,c);

and a1(c1,a,b);

and a2(c2,b,c);

and a3(c3,a,c);

or o1(out1,c1,c2);

or o2(cout,out1,c3);

endmodule

**Data Flow Modelling:**

module full\_adder( input a,b,c, output sum,cout );

assign sum = (a ^ b ^ c );

assign cout = (a & b ) | (b & c) | (a & c);

endmodule

**Test Bench**

module testbench\_fulladder;

reg a,b,c;

wire sum,cout;

full\_adder f1(a,b,c,sum,cout);

initial begin

a = 1'b0; b = 1'b0; c = 1'b0; #20;

a = 1'b0; b = 1'b0; c = 1'b1; #20;

a = 1'b0; b = 1'b1; c = 1'b0; #20;

a = 1'b0; b = 1'b1; c = 1'b1; #20;

a = 1'b1; b = 1'b0; c = 1'b0; #20;

a = 1'b1; b = 1'b0; c = 1'b1; #20;

a = 1'b1; b = 1'b1; c = 1'b0; #20;

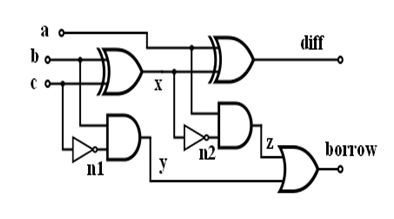
a = 1'b1; b = 1'b1; c = 1'b1; #20;

$finish;

end

endmodule

**ii) Full Subtractor (HDL implementation: data flow-level)**

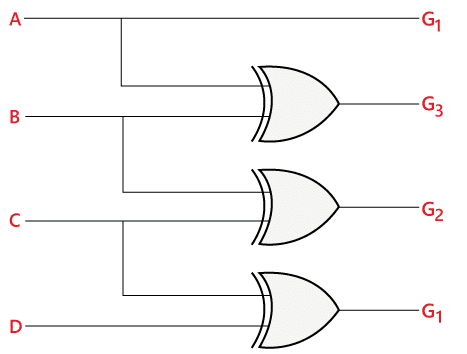


module full\_subtractor( input a, b, bin, output diff , borr);  
assign x = a ^ b ;  
assign y = (~x) & bin ;  
assign z = (~a) & b ;

assign diff = x ^ bin ;  
assign borr = y | z ;

endmodule

**iii) Binary to Gray (HDL implementation: gate-level)**



module Binary\_to\_Gray( input [3:0] b, output [3:0] g );

xor A1(g[0],b[1],b[0]);

xor A2(g[1],b[2],b[1]);

xor A3(g[2],b[3],b[2]);

buf A4(g[3],b[3]);

endmodule

**iv) HDL implementation: Data flow-level of 3:8 decoder**

module decoder3\_to\_8(input x,y,z, output d0, d1, d2, d3, d4, d5, d6, d7);

assign d0 = xn & yn & zn;

assign d1 = xn & yn & z;

assign d2 = xn & y & zn;

assign d3 = xn & y & z;

assign d4 = x & yn & z;

assign d5 = x & yn & z;

assign d6 = x & y & zn;

assign d7 = x & y & z;

assign xn = ~ x;

assign yn = ~ y;

assign zn = ~ z;

endmodule

**v) 4:1 MUX using 2:1 MUX, HDL implementation: gate-level**

**Data Flow Model:**

module mux\_con(out,s0,s1,i);

input s0,s1;

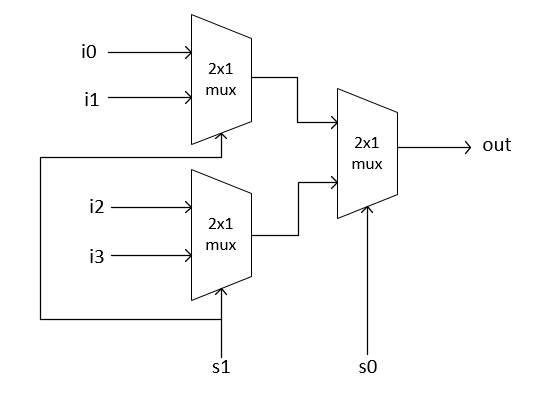
input [3:0]i;

output out;

wire out;

assign out = s1 ? ( s0 ? i[3]:i[2]) : (s0 ? i[1]:i[0]) ;

endmodule



module mux\_2\_1(

input sel,

input i0, i1,

output y);

assign y = sel ? i1 : i0;

endmodule

module mux\_4\_1(

input sel0, sel1,

input i0,i1,i2,i3,

output reg y);

wire y0, y1;

mux\_2\_1 m1(sel1, i2, i3, y1);

mux\_2\_1 m2(sel1, i0, i1, y0);

mux\_2\_1 m3(sel0, y0, y1, y);

endmodule

**Gate Level Modelling**

module mux\_2\_1(s,a,b,out);

input s,a,b;

wire and\_1,and\_2,s\_c;

output out;

not (s\_c,s);

and (and\_1,a,s\_c);

and (and\_2,b,s);

or (out,and\_1,and\_2);

endmodule