***VERILOG CODE***

***HALF ADDER***

**Data flow :**

module ha(s,ca,a,b);

input a,b;

output s,ca;

assign s=a^b;

assign ca=a&b;

endmodule

**Gate model :**

module hagate(s,c,a,b);

input a,b;

output c,s;

xor x1(s,a,b);

and x2(c,a,b);

endmodule

**Behaviour model :**

module habehavior(sum,carry,a,b);

input a,b;

output sum,carry;

reg sum,carry;

always@(a or b )

begin

case({a,b})

2'b00:begin

sum=0;carry=0; end

2'b01:begin

sum=1;carry=0; end

2'b10:begin

sum=1;carry=0; end

2'b11:begin

sum=0;carry=1; end

endcase

end

endmodule

***FULL ADDER***

**Data flow :**

module fadata(s,ca,a,b,c);

input a,b,c;

output s,ca;

assign s=a^b^c;

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

endmodule

**Gate model :**

module fagate(s,ca,a,b,c);

input a,b,c;

output s,ca;

wire x,y,z;

xor x1(s,a,b,c);

and x2(x,a,b);

and x3(y,b,c);

and x4(z,c,a);

or x5(ca,x,y,z);

endmodule

**Behaviour model :**

module fabehav(s,ca,a,b,c);

output s,ca;

reg s,ca;

input a,b,c;

always@(a or b or c)

begin case({a,b,c})

3'b000:begin

s=0;ca=0;end

3'b001:begin

s=1;ca=0;end

3'b010:begin

s=1;ca=0;end

3'b011:begin

s=0;ca=1;end

3'b100:begin

s=1;ca=0;end

3'b101:begin

s=0;ca=1;end

3'b110:begin

s=0;ca=1;end

3'b111:begin

s=1;ca=1;end

endcase

end

endmodule

***4 X 1 MUX***

**Data flow :**

module mux(y,s1,s0,a,b,c,d);

output y;

input a,b,c,d,s1,s0;

wire p,q,r,m,n,o;

assign p=~s1;

assign q=~s0;

assign r=p&q&a;

assign m=p&s0&b;

assign n=s1&q&c;

assign o=s1&s0&d;

assign y=r|m|n|o;

endmodule

**Gate model :**

module muxgate(y,a,b,c,d,s1,s0);

input a,b,c,d,s1,s0;

output y;

wire p,q,r,s,t,u;

not x1(p,s1);

not x2(q,s0);

and x3(r,p,q,a);

and x4(s,p,s0,b);

and x5(t,s1,q,c);

and x6(u,s1,s0,d);

or x7(y,r,s,t,u);

endmodule

**Behaviour model :**

module muxbeha(y,a,b,c,d,s1,s2);

input a,b,c,d,s1,s2;

output y;

reg y;

always@(s1 or s2)

begin

case({s1,s2})

2'b00:begin

y=a;end

2'b01:begin

y=b;end

2'b10:begin

y=c;end

2'b11:begin

y=d;end

endcase

end

endmodule

***1 X 4 DEMUX***

**Data flow :**

module demuxdata(a,b,c,d,y,s1,s0);

output a,b,c,d;

input y,s1,s0;

wire p,q;

assign p=~s1;

assign q=~s0;

assign a=p&q&y;

assign b=p&s0&y;

assign c=s1&q&y;

assign d=s1&s0&y;

endmodule

**Gate model :**

module demuxgate(y,a,b,c,d,s1,s0);

output a,b,c,d;

input y,s1,s0;

wire p,q;

not x1(p,s1);

not x2(q,s0);

and x3(a,p,q,y);

and x4(b,p,s0,y);

and x5(c,s1,q,y);

and x6(d,s1,s0,y);

endmodule

**Behaviour model :**

module demuxbehav(y,a,b,c,d,s1,s2);

output a,b,c,d;

input y,s1,s2;

reg a,b,c,d;

always@(s1 or s2)

begin

case({s1,s2})

2'b00:begin

a=y;b=0;c=0;d=0;end

2'b01:begin

a=0;b=y;c=0;d=0;end

2'b10:begin

a=0;b=0;c=y;d=0;end

2'b11:begin

a=0;b=0;c=0;d=y;end

endcase

end

endmodule

***ENCODER 8:3***

**Data flow :**

module Encoder(a,b,c,y7,y6,y5,y4,y3,y2,y1,y0);

output a,b,c;

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

assign a=y7|y6|y4|y5;

assign b=y7|y6|y3|y2;

assign c=y7|y5|y3|y1;

endmodule

**Gate model :**

module Encodergate(a,b,c,y7,y6,y5,y4,y3,y2,y1,y0);

output a,b,c;

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

or x1(a,y7,y6,y5,y4);

or x2(b,y7,y6,y3,y2);

or x3(c,y7,y5,y3,y1);

endmodule

**Behaviour model :**

module encoderbehav(a,b,c,y7,y6,y5,y4,y3,y2,y1,y0);

output a,b,c;

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

reg a,b,c;

always@(y7 or y6 or y5 or y4 or y3 or y2 or y1 or y0)

begin

case({y7,y6,y5,y4,y3,y2,y1,y0})

8'b00000001:begin

a=0;b=0;c=0;end

8'b00000010:begin

a=0;b=0;c=1;end

8'b00000100:begin

a=0;b=1;c=0;end

8'b00001000:begin

a=0;b=1;c=1;end

8'b00010000:begin

a=1;b=0;c=0;end

8'b00100000:begin

a=1;b=0;c=1;end

8'b01000000:begin

a=1;b=1;c=0;end

8'b10000000:begin

a=1;b=1;c=1;end

endcase

end

endmodule

***DECODER 3:8***

**Data flow :**

module decoderdataflow(a,b,c,y7,y6,y5,y4,y3,y2,y1,y0);

input a,b,c;

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

wire x,y,z;

assign x=~a;

assign y=~b;

assign z=~c;

assign y0=x&y&z;

assign y1=x&y&c;

assign y2=x&b&z;

assign y3=x&b&c;

assign y4=a&y&z;

assign y5=a&y&c;

assign y6=a&b&z;

assign y7=a&b&c;

endmodule

**Gate model :**

module decodegate(a,b,c,y7,y6,y5,y4,y3,y2,y1,y0);

input a,b,c;

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

wire x,y,z;

not x1(x,a);

not x2(y,b);

not x3(z,c);

and x4(y0,x,y,z);

and x5(y1,x,y,c);

and x6(y2,x,b,z);

and x7(y3,x,b,c);

and x8(y4,a,y,z);

and x9(y5,a,y,c);

and x10(y6,a,b,z);

and x11(y7,a,b,c);

endmodule

**Behaviour model :**

module decoderbehav(a,b,c,y7,y6,y5,y4,y3,y2,y1,y0);

input a,b,c;

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

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

always@(a or b or c)

begin

case({a,b,c})

3'b000:begin

y7=0;y6=0;y5=0;y4=0;y3=0;y2=0;y1=0;y0=1;end

3'b001:begin

y7=0;y6=0;y5=0;y4=0;y3=0;y2=0;y1=1;y0=0;end

3'b010:begin

y7=0;y6=0;y5=0;y4=0;y3=0;y2=1;y1=0;y0=0;end

3'b011:begin

y7=0;y6=0;y5=0;y4=0;y3=1;y2=0;y1=0;y0=0;end

3'b100:begin

y7=0;y6=0;y5=0;y4=1;y3=0;y2=0;y1=0;y0=0;end

3'b101:begin

y7=0;y6=0;y5=1;y4=0;y3=0;y2=0;y1=0;y0=0;end

3'b110:begin

y7=0;y6=1;y5=0;y4=0;y3=0;y2=0;y1=0;y0=0;end

3'b111:begin

y7=1;y6=0;y5=0;y4=0;y3=0;y2=0;y1=0;y0=0;end

endcase

end

endmodule

***4 BIT ADDER USING STRUCTURAL MODELLING :***

module bitfa(a0,a1,a2,a3,b0,b1,b2,b3,cin,s0,s1,s2,s3,c3);

output s0,s1,s2,s3,c3;

input a0,a1,a2,a3,b0,b1,b2,b3,cin;

wire c0,c1,c2;

fadata x1(a0,b0,cin,s0,c0);

fadata x2(a1,b1,c0,s1,c1);

fadata x3(a2,b2,c1,s2,c2);

fadata x4(a3,b3,c2,s3,c3);

endmodule

***Sub-program :***

module fadata(s,ca,a,b,c);

input a,b,c;

output s,ca;

assign s=a^b^c;

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

endmodule

***4 BIT 2CUMS 1 ADDER USING STRUCTURAL MODELLING :***

module bitfs(a0,a1,a2,a3,b0,b1,b2,b3,cin,s0,s1,s2,s3,c3);

output s0,s1,s2,s3,c3;

input a0,a1,a2,a3,b0,b1,b2,b3,cin;

wire x,y,z,w;

assign w=~b0;

assign x=~b1;

assign y=~b2;

assign z=~b3;

wire c0,c1,c2;

fsdata x1(a0,w,cin,s0,c0);

fsdata x2(a1,x,c0,s1,c1);

fsdata x3(a2,y,c1,s2,c2);

fsdata x4(a3,z,c2,s3,c3);

endmodule

***Sub-program :***

module fsdata(a,b,c,d,bo);

input a,b,c;

output d,bo;

assign d=a^b^c;

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

endmodule

***T flip flop :***

module Tff(q,q1,t,clk);  
output q,q1;  
reg q=0,q1;  
input t,clk;  
  
always@(posedge clk)  
begin  
if(t)  
q=~q;  
q1=~q;  
end  
endmodule

***D flip flop :***

module Dff(d,clk,reset,do);  
output do;  
reg do;  
input d;  
input clk;  
input reset;  
always @(posedge clk)  
begin  
if(reset)  
do<=1;  
else  
do<=d;  
end  
endmodule

***J K flip flop :***

module JKff(j,k,q,q1,clk);  
input j,k,clk;  
    output q,q1;  
reg q=1'b0,q1=1'b1;  
always @(posedge clk or j or k) begin  
case({j,k})  
2'b00:begin  
end  
 2'b01:begin  
 q=1'b0;q1=~q;end  
 2'b10:begin  
 q=1'b1;q1=~q;end  
  2'b11:begin  
 q=q1;q1=~q;end  
endcase  
end  
endmodule