```
module lab1_and_gate(
input a,
input b,
output c,
output d,
output e,
output f,
output g
);
assign c = a&b; // and gate
assign d = a|b; // or gate
assign e = a^b; // xor gate
assign f = ~(a^b); // xnor gate
assign g = ~a; // not gate
endmodule



module testbench();

reg a,b;
wire c,d,e,f,g;
lab1_and_gate uut(a,b,c,d,e,f,g);
initial
begin
a = 0; b = 0;
#10
a = 0; b = 1;
#10
a = 1; b = 0;
#10
a = 1; b = 1;
#10
$finish;
end
endmodule




// Dataflow code
module lab2_half_adder(
input a,
input b,
output sum,
output carry
);

assign sum = a^b;
assign carry = a&b;

endmodule
```

```
// Behavioral code
// Method 1
module lab2_half_adder(
input a,
input b,
output sum,
output carry
);
reg sum, carry;
always@(a or b)
{carry, sum} = a+b;
endmodule

// Method 2
module lab2_half_adder(
input a,
input b,
output sum,
output carry
);
reg sum, carry;
always@(a or b)
if(a==0 && b==0)
begin
 sum = 0;
 carry = 0;
end
if(a==0 && b==1)
begin
 sum = 1;
 carry = 0;
end
if(a==1 && b==0)
begin
 sum = 1;
 carry = 0;
end
if(a==1 && b==1)
begin
 sum = 0;
 carry = 1;
end
endmodule



// testbench code
module testbench();
reg a,b;
wire sum, carry;
lab2_half_adder uut(a,b,sum,carry);
```

```
initial
begin
a = 0; b = 0;
#10
a = 0; b = 1;
#10
a = 1; b = 0;
#10
a = 1; b = 1;
#10
$finish;
end
endmodule
```

```
// Dataflow code
module lab3_full_adder(
input a,
input b,
input c,
output sum,
output carry
);

assign sum = a^(b^c);
assign carry = (a&b) | (b&c) | (c&a);

endmodule

//Behavioural code
module lab3_full_adder(
input a,
input b,
input c,
output sum,
output carry
);
reg sum, carry;
always@(a or b or c)
begin
case(a|b|c)
 3'b000: begin sum=0; carry=0; end
 3'b001: begin sum=1; carry=0; end
 3'b010: begin sum=1; carry=0; end
 3'b011: begin sum=0; carry=1; end
 3'b100: begin sum=1; carry=0; end
 3'b101: begin sum=0; carry=1; end
 3'b110: begin sum=0; carry=1; end
 3'b111: begin sum=1; carry=1; end
endcase
end
```

```
endmodule


// Structural Modelling
module lab3_full_adder(
input a,
input b,
input c,
output sum,
output carry
);

wire w1,c1,c2,c3,z1;
xor x1(w1, a, b);
xor x2(sum, w1, c);

and a1(c1, a, b);
and a2(c2, b, c);
and a3(c3, c, a);

or o1(z1, c1, c2);
or o2(carry, z1, c3);

endmodule



// Full adder using half adder
//Structural Modelling
module lab3_full_adder(
input a,
input b,
input c,
output sum,
output carry
);
wire sum1,carry1, carry2;
lab2_half_adder h1(
.a(a),
.b(b),
.sum(sum1),
.carry(carry1)
);
lab2_half_adder h2(
.a(sum1),
.b(c),
.sum(sum),
.carry(carry2)
);
or o1(carry, carry1, carry2);
endmodule
```

```
module testbench();
reg a,b,c;
wire sum, carry;
lab3_full_adder uut(a,b,c,sum,carry);
initial
begin
a = 0; b = 0;
#10
a = 0; b = 1;
#10
a = 1; b = 0;
#10
a = 1; b = 1;
#10
$finish
end
endmodule
```

```
module ripple_adder(
input [3:0] a,
input [3:0] b,
input cin,
output [3:0] sum,
output carry
);

wire c1, c2, c3;
lab2_full_adder h1(a[0], b[0], cin, sum[0], c1);
lab2_full_adder h1(a[1], b[1], c1, sum[1], c2);
lab2_full_adder h1(a[2], b[2], c2, sum[2], c3);
lab2_full_adder h1(a[3], b[3], c3, sum[3], carry);

endmodule
```

```
module testbench();
reg [3:0]a,b;
reg cin;
wire [3:0]sum;
wire carry;

ripple_adder uut(a,b,cin, sum, carry);
initial
begin
a=4'b0100;
b=4'b1101;
c=1'b1;

#10
```

```
a=4’b1011;
b=4’b0101;
cin=1’b1;

#10

$finish;
end
endmodule



// Dataflow code

module lab_5_mux_4to1(
input a,
input b,
input c,
input d,
input [1:0] in,
output out
);
assign out = in[1] ? (in[0] ? d : c) : (in[0] ? b : a);
endmodule


// Testbench code for dataflow code

module testbench();
reg a,b,c,d;
reg [1:0] in;
wire out;
lab_5_mux_4to1 uut(a,b,c,d,in,out);
initial
begin
a=1;b=0;c=0;d=0;in=2’b00;
#10
a=0;b=1;c=0;d=0;in=2’b01;
#10
a=0;b=0;c=1;d=0;in=2’b10;
#10
a=0;b=0;c=0;d=1;in=2’b11;
#10
$finish;
end
endmodule



// Behavioural Modelling
module lab_5_mux_4to1(
input a,
input b,
input c,
```

```
input d,
input in0,
input in1,
output out,
);
reg out2
always @(a or b or c or d or in0 or in1)
begin
case(in0|in1)
2’b00: out2 <= a;
2’b01: out2 <= b;
2’b10: out2 <= c;
2’b11: out2 <= d;
endcase
end
endmodule



// Testbench code for behavioural code
module testbench();
reg a,b,c,d,in0,in1;
wire out;
initial
begin
a=1;b=0;c=0;d=0;in0=0;in1=0;
#10
a=0;b=1;c=0;d=0;in0=0;in1=1;
#10
a=0;b=0;c=1;d=0;in0=1;in1=0;
#10
a=0;b=0;c=0;d=1;in0=1;in1=1;
#10
$finish;
end
endmodule




//3 to 8 decoder

module dataflow_decoder(
    input [2:0] x,
    output [7:0] y
    );

    assign y[0] = ~x[2] & ~x[1] & ~x[0];
    assign y[1] = ~x[2] & ~x[1] & x[0];
    assign y[2] = ~x[2] & x[1] & ~x[0];
    assign y[3] = ~x[2] & x[1] & x[0];
    assign y[4] = ~x[2] & x[1] & x[0];
    assign y[5] = x[2] & ~x[1] & x[0];
    assign y[6] = x[2] & x[1] & ~x[0];
```

```
    assign y[7] = x[2] & x[1] & x[0];
endmodule



module stimuli;
reg [2:0] x;
wire [7:0] y;
dataflow_decoder dut(
.x(x),
.y(y)
);
reg clk;
always #5 clk = ~clk;

initial begin
clk = 0;
x = 0;

#10 x = 3’b000;
#10 x = 3’b001;
#10 x = 3’b010;
#10 x = 3’b011;
#10 x = 3’b100;
#10 x = 3’b101;
#10 x = 3’b110;
#10 x = 3’b111;

$display("input x = %b, Output y = %b",x,y);
$finish;
end
endmodule
```

8 to 3 encoder

```
module encoder_implementation(o,a,en);
input [7:0] a;
input en;
output [2:0] o;
reg [2:0] o;
always @(a,en)
begin
if(en == 1’b1)
case(a)
8’b00000001 : o = 3’b000;
8’b00000010 : o = 3’b001;
8’b00000100 : o = 3’b010;
8’b00001000 : o = 3’b011;
8’b00010000 : o = 3’b100;
8’b00100000 : o = 3’b101;
8’b01000000 : o = 3’b110;
8’b10000000 : o = 3’b111;
endcase
else
```

```
o = 3’bzzz;
end
endmodule


module stimuli( );
reg [7:0] a;
reg en;
wire [2:0] o;
encoder_implementation uut(o,a,en);
initial begin
a=8’b00000000;
en=1’b1;
#10 a= 8’b00000001;
#10 a= 8’b00000010;
#10 a= 8’b00000100;
#10 a= 8’b00001000;
#10 a= 8’b00010000;
#10 a= 8’b00100000;
#10 a= 8’b01000000;
#10 a= 8’b10000000;
end
always #90 en = ~en;
initial #120 $stop;
endmodule
```

sr flip flop

```
module sr_flipflop(
    output reg q,qbar,
    input s,r,clk
    );
    always@(posedge clk)
    begin
    q=1’b0 ; qbar = 1’b1;
    if(clk ==1) begin
    if(s==0 && r==0)begin
    q=q;qbar=qbar;
end
    else if(s==0 && r==1) begin
    q = 1’b0;qbar = 1’b1;
end
    else if(s==1 && r==0) begin
    q=1’b1;qbar=1’b0;
end
    else if(s==1 && r==1) begin
    q = 1’bx ; qbar = 1’bx;
end
    end
    if(clk ==0) begin
    q=q; qbar= qbar;
end
    end
```

```
endmodule



module stimuli( );
wire q,qbar;
reg s,r,clk;
sr_flipflop uut(q,qbar,s,r,clk);

initial begin
s =0;r=0;clk=1;
#500 $finish;
end

always #10 clk = ~clk;
always #50 s = ~s;
always #100 r = ~r;

endmodule
```