GATE LEVEL MODELLING

```
module  gates ( A, B, x, y, P, n, w, q, r);
 input   A, B;
output   x, y, P, n, w, q, r;
  and (x, A, B);
   or  (Y, A, B);
  xor (P, A, B);
   not (n, A, B);
  nand (w, A, B);
   nor (q, A, B);
  xnor (r, A, B);
   endmodule
```

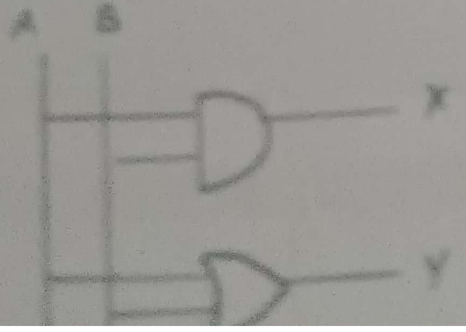
DATA FLOW MODELLING

```
module  gates  ( A, B, x, Y, P, n, w, q, r);
  input    A, B;
  output   x, Y, P, n, w, q, r;
  assign   x = A&B;
  assign   Y = A|B;
  assign   P = A^B;
  assign   n = A~B;
  assign   w = A~&B;
   assign  q = A~|B;
    assign r = A~^B;
  end module
```

VERILOG TEST FIXTURE

```
initial begin
/ initalize Input
a=0;
b=0;
#100;
a= 0;
b 1;
# 100;
a= 1;
b= 0;
# 100;
a=1;
b=1;
# 200;
end
```

SIMULATION OF LOGIC GATES:



Full Adder

```
module  sh1 (a,b,c ,s,y);
input  a,b,c;
output  s,y;
wire    m,n,p;
 xor (s, a,b, c);
and (m, a, b);
and(n, a,b);
and (p, a,b);
 or  (y,m,n,p);
end module
```

Data flow modeling:

```
module  adder (a,b,c,d,e)
input  a,b,c;
output sd,e;
wire   x,y,z;
assign  x = a&b;
assign  y = b&c;
assign  z = c&a;
assign  e = b|c;
endmodule
```

Full Subractor:

```
module sub (a,b,c, diff, box);
input a,b,c;
output diff, box;
wire w1, w2, w3, w4, w5;
xor (w1, a,b);
not (w2,a);
xor (diff, w1, c);
and (w3, w2,b);
and (w4, b, c);
and (w5, w2, c);
or (box, w3, w4, w5);
endmodule;
```

Data flow

```
module sub (input, A, B, Bin. out diff, Bout);
difference = A xor B xor Bin
assign diff = A^B^Bin;
Borrow out (Bout) = (NOT A AND B) or (B AND Bin) or (NOT A AND Bin)
assign Bout = (~A&B) |(B& Bin) | (~A & Bin);
endmodule.
```

8:1 Mux

```
module mux (a,b,c,e,d0, d1,d2, d3, d4, d5, d6, d7, o);
input a,b,c, e, d0, d1 ,d2, d3, d4, d5, d6, d7;
output o;
wire x1, x2, x3, x4, x5, x6, x7, x8, x, y, z;
and(x1, e, x, y, z, d0);
and(x2, e, x, y, c, d1);
and(x3, e, x, b, z, d2);
and(x4, e, x, b, c, d3);
and(x5, e, a, y, z, d4);
and(x6, e, a, y, c, d5);
and(x7, e, a, b, z, d6);
and(x8, e, a, b, c, d7);
not(x, a);
not(y, b);
not(z, c);
 or(o, x1, x2, x3, x4, x5, x6, x7, x8);
end module
```

Data flow

```
module ms, (out, input D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2);
assign S1 bar
assign S0 bar
assign S2 bar
assign out = (D0 & S2 bar & S1 bar & S0 bar)|(D1 & S2 bar & S1 ba
      & S0) | (D2 & S2 bar & S1 & S0bar) + (D3 & S2 bar &
      S1 & S0) + (D4 & S2 & S1 bar & S0) + (D6 & S2 & S1) &
      (S0bar) + (D7 & S2 & S1 & S0);
 end module
```

Demux

```
module demux (d, [0:1] s, [0:7] y);
input d, so, s1, s2;
output y0, y1, y2, y3, y4, y5, y6, y7;
not (son, so) (s1n, s1) (s2n, s2);
and (y0, d, son, s1n, s2n), (y1, d, so, s1, s2n);
        (y2, d, son, s1, s2n), (y3, d, so, s1 s2n);
        (y4, d, son, s1n, s2), (y5, d, so, s1n, s2);
        (y6, d, son, s1, s2), (y7, d, so, s, s2);
end module
```

Data flow

```
module demux (d, so, s1, s2, y0, y1, y2, y3, y4, y5, y6, y7);
input d1, so, s1, s2;
output y0, y1, y2, y3, y4, y5, y6, y7;
assign son = ~so;
assign s1n = ~s1;
assign s2n = ~s2;
assign y0 = d & son & s1n & s2n;
assign y1 = d & so & s1n & s2n;
assign y2 = d & son & s1 & s2n;
assign y3 = d & so & s1 & s2n;
assign y4 = d & so & s1n & s2;
endmodule
```

Encoder

```
module encoder ( D, Y1, Yo);
input [3:0] D;
or (y1, D[2], D[3]);
or (yo, D[1], D[3]);

end module'
```

Data flow

```
module encoder (D, Y1, Yo)
Input [3:0],D;
output y1, yo;
assign  Y1= D[2]| D[3];
asogn  Yo= D[1] | D[3];
end module
```

Decoder

```
module decoder _2 _to 2 C
Input[1:0] A;
output Yo, Y1 Y2, Y3;
wire not Ao, not A1;
not (not A1, A[0]);
not (not A1. A[1]);
and (Yo, not A, not Ao);
and (Y1, not A, not A[0]);
and (Y2, A[1], not Ao);
and (Y3, A[1], A[0]);
endmodule
```

Data flow

```
module decoder (A, Yo, Y1, Y2, Y3)
Input [1:0] A;
output Yo, Y1, Y2, Y3;
assign  Yo = ~A[1] & ~A[0];
assign  Y1 = ~A[1] & A[0];
assign  Y2 = A[1] & ~A[0];
assign  Y3 = A[1] & A[0];
endmodule
```

Data flow

```
module adder ( a, b, cin , carry, sum);
assign  sum =  a^b ^cin;
assign  carry= a&b| b&cin | cin & a;
input  a,b, cin;
output  carry, sum;
end module;
module rca (a,b, cout , c)
input a,b;
output  sum, cout;
wire  c0 , c1 , c2,  c3 , c4, c5, c6, c7 ;
adder fa0 ( a[0], b[0], cin, c0, sum[0]);
adder fa1 (a[1], b[1], c0 ,  c1, sum[1]);
adder fa2 ( a[2], b[2), c1,  c2, sum[2]);
adder fa3 ( a[3], b[3] , c2.  c3. sum[3]);
adder fa4 ( a[4], b[4] , c3, c4, sum[4]);
adder fa5 (a[5], b[5], c4, c5 , sum[5]);
adder fa6 ( a[6] , b[6], c5, c6, sum[6]);
adder fa7 ( a[7], b[7], c6 , cout , sum[7]);
end module
```

```
Text Fixture

initial begain;

a = 0;

b = 0;

cin = 0;

#100

a = 0

b = 1

cin = 0;

#100;
```

Program

```
module multiplier (a,b,c)
input [3:0] a,b;
output [7:0] c;
wire [7:0] P1, P2, P3, P4;
assign P1 = {4'b0, b[0]&a[3], b[0]&a[2], b[0]&a[1], b[0]&a[0]}
assign P2 = {3'b0, b[1]&a[3], b[1]&a[2], b[1]&a[1], b[1]&a[0], 1'b0}
assign P3 = {2'b0, b[2]&a[3], b[2]&a[2], b[2]&a[1], b[2]&a[0], 2'b0
assign P4 = {     b[3]&a[3], b[3]&a[2], b[3]&a[1], b[3]&a[0], 4'b0

assign c = P1 + P2 + P3 + P4;
endmodule
```