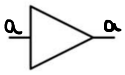
(lab 2 - S2)

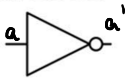
Porți logice

Driver



Input	Output
0	0
1	1

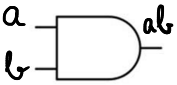
Inverter



Input	Output
0	1
1	0

not (F, a)
F = a' = $\bar{a}$
assign F = ~A;

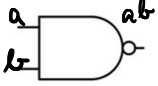
AND



A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

and (F, a, b)
assign F = a && b;

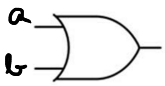
NAND



A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

nand (F, a, b)
assign F = ~(a && b);

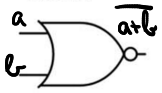
OR



A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

or (F, a, b)
assign F = a || b;

NOR



A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

nor (F, a, b);
assign F = ~(a || b);

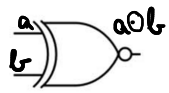
XOR

A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

xor (F, a, b)
assign F = a ^ b;

$a \oplus b = \bar{a}b + \bar{b}a$

XNOR



A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1

xnor (F, a, b);
assign F = ~(a ^ b);

$a \odot b = ab + \bar{a}\bar{b}$

În limbajul de programare Verilog, instructiunile sunt executate în paralel.
Singurul bloc care face exceptie de la aceasta regula este blocul always.

Operatiuni de atribuire:

assign: atunci când valoarea din partea dreapta se schimba, declanseaza modificarea valoarii din partea stânga

<= (mai mic sau egal cu): este o operatie neblocanta care se executa la POSEDGE CLK

= : este o operatiune de blocare

Element neutru

$a + 0 = a$

$a \cdot 1 = a$

Complement

$a + \bar{a} = 1$

$a \cdot \bar{a} = 0$

Legea lui De Morgan

$\overline{ab} = \bar{a} + \bar{b}$

$\overline{a+b} = \bar{a} \cdot \bar{b}$

Majority voter 2 inputs

ex1.v

```
module ex1(output o, input x, input y);
    //porti logice
    wire r1, r2;
    assign r1 = x|y;
    assign r2 = ~y;

    assign o = r1&r2;
endmodule

```

ex1_tb.v

```
module ex1_tb;
    reg x, y;
    wire o;

    ex1 uut(.o(o), .x(x), .y(y));
    initial begin
        x = 0;
        y = 0;
        #100;
    end

    always begin
        #25 x=~x;
        #50 y=~y;
    end
endmodule
```

Majority voter 3 inputs

mj_voter.v

```
module mj_voter(
    input in1,
    input in2,
    input in3,
    output out
);

    wire nand1, nand2, nand3l;

    assign nand1 = ~(in1 & in2);
    assign nand2 = ~(in1 & in3);
    assign nand3 = ~(in2 & in3);

    assign out = ~(nand1 & nand2 & nand3);
endmodule
```

testbench.v

```
module testbench();

    //Inputs
    reg in1;
    reg in2;
    reg in3;

    //Outputs
    wire out;

    mj_voter DUT(
        .in1(in1),
        .in2(in2),
        .in3(in3),
        .out(out)
    );

    initial begin
        //initilize
        in1 = 0;
        in2 = 0;
        in3 = 0;

        #100;
    end

    always begin
        #25 in1 = ~in1;
        #50 in2 = ~in2;
        #75 in3 = ~in3;
    end
endmodule
```

Exemplu laborator 3 inputs, 2 outputs

out1 = $\overline{ab} + \overline{ac} + \overline{bc}$

out2 = $a \oplus b$ (XOR)

a	b	c	$\overline{ab}$	$\overline{ac}$	$\overline{bc}$	out1	out2
0	0	0	1	1	1	1	0
0	0	1	1	1	1	1	0
0	1	0	1	1	1	1	1
0	1	1	1	1	0	1	1
1	0	0	1	1	1	1	1
1	0	1	1	0	1	1	1
1	1	0	0	1	1	1	0
1	1	1	0	0	0	0	0

out = $\bar{a}\bar{b}\bar{c} + \bar{a}bc + a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c} + abc$

a	b	c	out
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Formula out se poate construi pe baza valorilor de adevăr.

3'b101; -> nr. pe 3 biți

gates.v

```
module gates(
  input in1, in2, in3,
  output out1, out2
);

wire nand1, nand2, nand3;

assign nand1 = (~in1 & in2);
assign nand2 = (~in1 & in3);
assign nand3 = (~in2 & in3);

assign out1 = nand1 | nand2 | nand3;
assign out2 = (~in1 & in2) | (in1 & ~in2);

endmodule

```

gates_tb.v

```
module gates_tb;

    //Inputs
    reg in1, in2, in3;

    //Outputs
    wire out1, out2;

    gates
  dut(.in1(in1), .in2(in2), .in3(in3),
      .out1(out1), .out2(out2)
      );

  integer k;

    initial begin
      {in1, in2, in3} = 0;

      for(k = 1; k < 8; k = k + 1)
        #20 {in1, in2, in3} = k;

      //in1, in2, in3 sunt stocate automat pe 1 bit si vor lua toate valorile lui k
      //k = 1 -> 001 -> in1 = 0, in2 = 0, in3 = 1
      //k = 2 -> 010 si tot asa, 3 = 011, 4 = 100, 5 = 101, 6 =110, 7 = 111

      #20
    end

endmodule
```