📘 **Day 4 Topic: Verilog codes for Universal NAND and Universal NOR implementations for logic gates Using Cadence (Xcelium)**

- Why NAND & NOR are Universal Gates?
  - A universal gate means you can implement any Boolean function (AND, OR, NOT, XOR, XNOR, etc.) using only that gate.
  - NAND and NOR are called universal because:
    - From NAND → you can build NOT, AND, OR, XOR, etc.
    - From NOR → you can build NOT, OR, AND, XOR, etc.
  - Advantage: In real hardware (IC design), using only NAND or only NOR reduces cost and simplifies manufacturing.

1)NAND Implementations

- **1.1) Design Code**

nand_uni.v

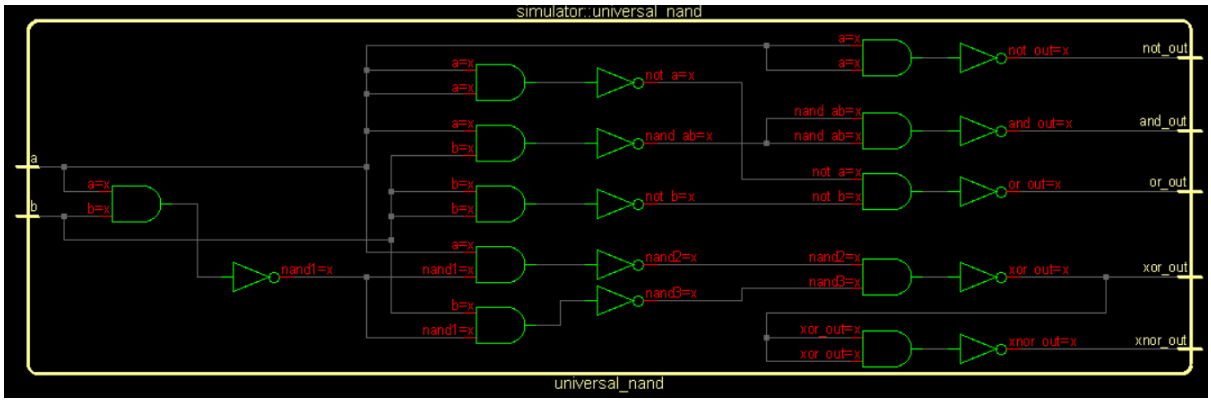
```
// Universal NAND Gate Implementations
module universal_nand(a, b, not_out, and_out, or_out, xor_out, xnor_out);
    input  a, b;                          // Inputs
    output not_out, and_out, or_out, xor_out, xnor_out; // Outputs

    // NOT using NAND: A NAND A = NOT(A)
    assign not_out = ~(a & a);

    // AND using NAND: (A NAND B) NAND (A NAND B) = A AND B
    wire nand_ab;
    assign nand_ab = ~(a & b);
    assign and_out = ~(nand_ab & nand_ab);

    // OR using NAND: (NOT A) NAND (NOT B) = A OR B
    wire not_a, not_b;
    assign not_a = ~(a & a);    // NOT A
    assign not_b = ~(b & b);    // NOT B
    assign or_out = ~(not_a & not_b);

    // XOR using NAND gates
    wire nand1, nand2, nand3;
    assign nand1 = ~(a & b);
    assign nand2 = ~(a & nand1);
    assign nand3 = ~(b & nand1);
    assign xor_out = ~(nand2 & nand3);

    // XNOR = NOT(XOR)
    assign xnor_out = ~(xor_out & xor_out);
endmodule
```

- **1.2) Test Bench Code**

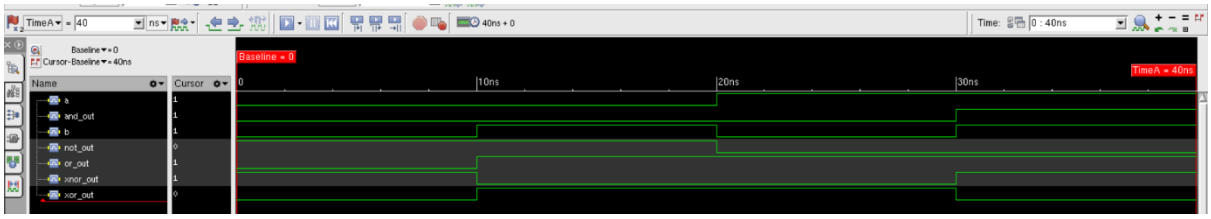
nand_uni_tb.v

```
// Testbench for Universal NAND Gate
module universal_nand_tb;
    reg a, b;                    // Test inputs
    wire not_out, and_out, or_out, xor_out, xnor_out; // Outputs

    // DUT (Device Under Test)
    universal_nand uut (.a(a), .b(b), .not_out(not_out), .and_out(and_out),
                        .or_out(or_out), .xor_out(xor_out), .xnor_out(xnor_out));

    initial begin
        // Monitor all signals
        $monitor("a=%b b=%b | NOT=%b AND=%b OR=%b XOR=%b XNOR=%b",
                 a, b, not_out, and_out, or_out, xor_out, xnor_out);

        // Test cases
        a=0; b=0; #10;
        a=0; b=1; #10;
        a=1; b=0; #10;
        a=1; b=1; #10;

        $finish;
    end
endmodule
```

- **1.3) Schematic**



- **1.4) Wave Forms**



2) NOR Implementations

2.1) Design Code

nor_uni.v

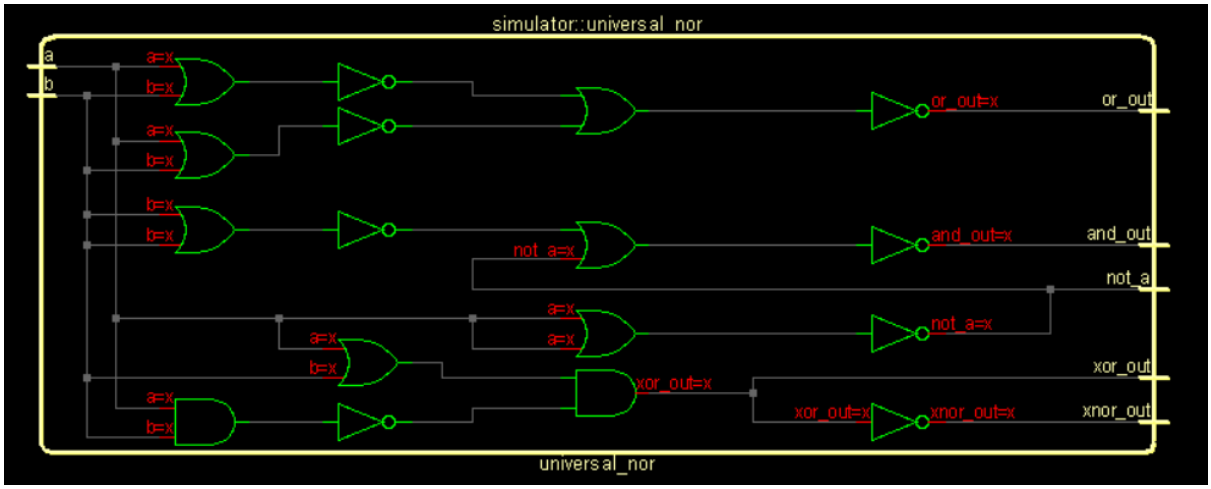
```
module universal_nor (
    input  wire a, b,
    output wire not_a,
    output wire and_out,
    output wire or_out,
    output wire xor_out,
    output wire xnor_out
);

    assign not_a   = ~(a | a);                  // NOT A
    assign or_out  = ~(~(a | b) | ~(a | b));    // OR = (A NOR B)'
    assign and_out = ~(not_a | ~(b | b));       // AND = ~(~A | ~B)
    assign xor_out = (a | b) & ~(a & b);        // XOR formula
    assign xnor_out= ~xor_out;                  // XNOR = NOT(XOR)

endmodule
```

2.2) Test Bench Code

nor_uni_tb.v

```
module universal_nor_tb;
    reg a, b;
    wire not_a, and_out, or_out, xor_out, xnor_out;

    universal_nor uut (a, b, not_a, and_out, or_out, xor_out, xnor_out);

    initial begin
        $display("a b | NOT AND OR XOR XNOR");
        a=0; b=0; #10 $display("%b %b |  %b   %b   %b   %b   %b", a,b,not_a,and_out,or_out,xor_out,xnor_out);
        a=0; b=1; #10 $display("%b %b |  %b   %b   %b   %b   %b", a,b,not_a,and_out,or_out,xor_out,xnor_out);
        a=1; b=0; #10 $display("%b %b |  %b   %b   %b   %b   %b", a,b,not_a,and_out,or_out,xor_out,xnor_out);
        a=1; b=1; #10 $display("%b %b |  %b   %b   %b   %b   %b", a,b,not_a,and_out,or_out,xor_out,xnor_out);
        $finish;
    end
endmodule
```

🔹 2.3) Schematic



🔹 2.4) Wave Forms

✨ Summary:

- **NAND & NOR are universal gates → can implement any logic.**
- **We built NOT, AND, OR, XOR, XNOR using only NAND and only NOR.**
- **Testbenches check for all input cases.**