📘 **Day 5 Topic: Verilog codes for Universal MUX implementations for logic gates Using Cadence (Xcelium)**

🔹 Why is a MUX called "Universal"?

Because any logic function (AND, OR, NOT, XOR, full adder, etc.) can be implemented using only a multiplexer.

Think of a multiplexer as a hardware truth-table selector:

- The select lines act as the input variables.
- The data inputs (I0, I1, ...) are fixed to either 0 or 1 (or signals) to generate the desired logic function.

So, instead of building many gates, you just configure the inputs of the MUX.

1) MUX Implementations

🔹 **1.1) Design Code**

mux_logic_gates.v

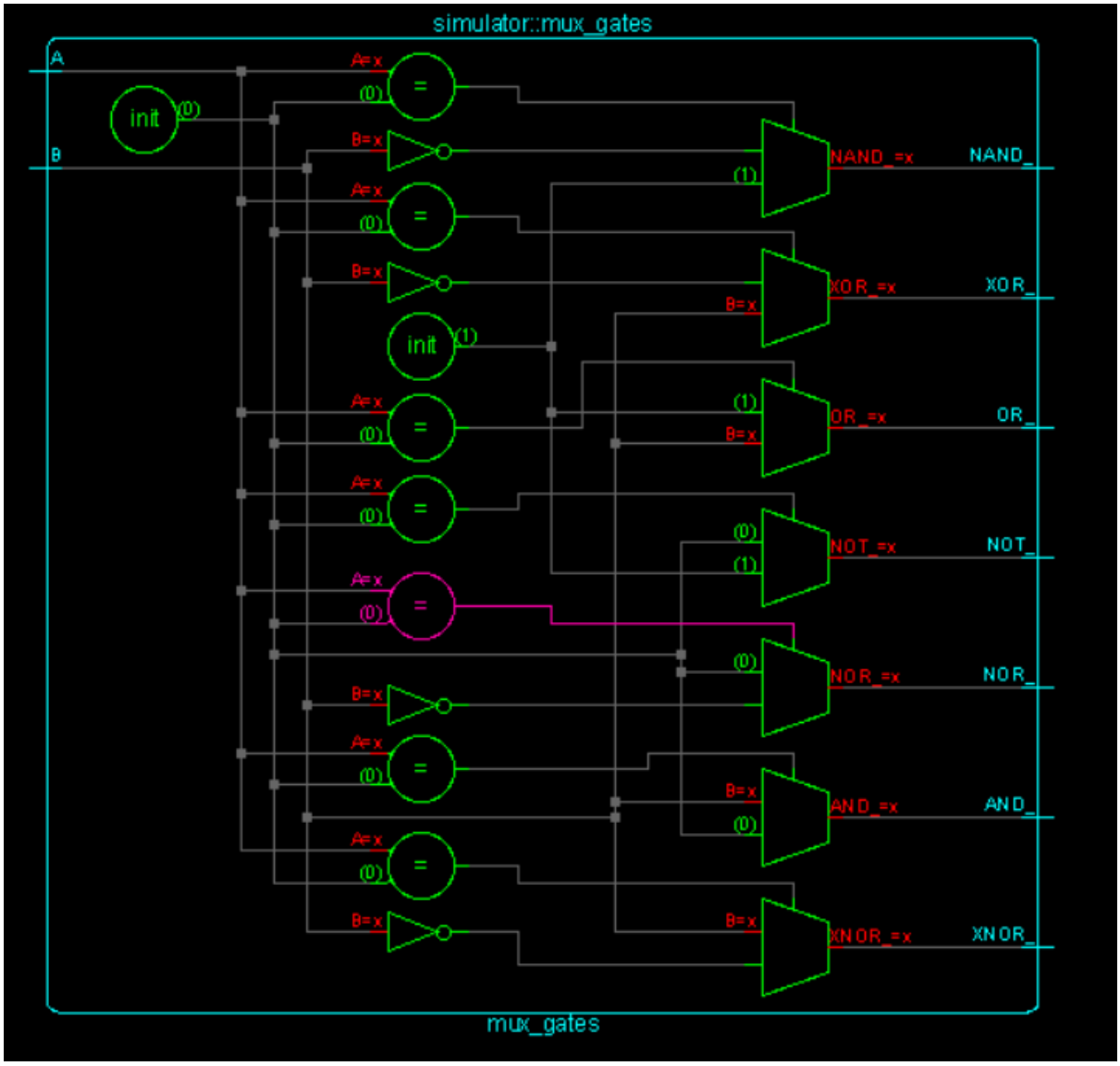
```
// Universal Gates using 2x1 MUX
// A = select line, I0/I1 values chosen to realize logic

module mux_gates(input A, B, output AND_, OR_, NOT_, NAND_, NOR_, XOR_, XNOR_);

    // NOT gate: Y = ~A
    assign NOT_  = (A == 0) ? 1 : 0;

    // AND gate: Y = A & B
    assign AND_  = (A == 0) ? 0 : B;

    // OR gate: Y = A | B
    assign OR_   = (A == 0) ? B : 1;

    // NAND gate: Y = ~(A & B)
    assign NAND_ = (A == 0) ? 1 : ~B;

    // NOR gate: Y = ~(A | B)
    assign NOR_  = (A == 0) ? ~B : 0;

    // XOR gate: Y = A ^ B
    assign XOR_  = (A == 0) ? B : ~B;

    // XNOR gate: Y = ~(A ^ B)
    assign XNOR_ = (A == 0) ? ~B : B;

endmodule
```

- **1.2) Test Bench Code**

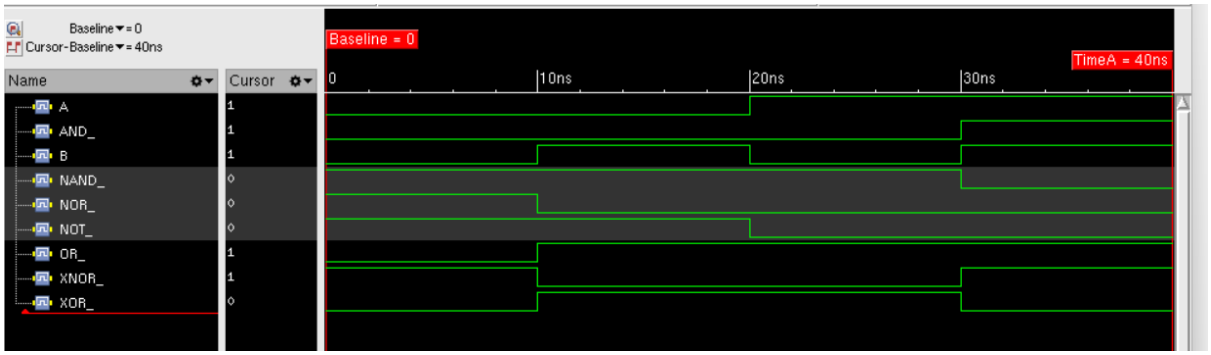
tb.v

```
module tb_mux_gates;
    reg A, B;
    wire AND_, OR_, NOT_, NAND_, NOR_, XOR_, XNOR_;

    // Instantiate DUT
    mux_gates uut(.A(A), .B(B), .AND_(AND_), .OR_(OR_), .NOT_(NOT_),
                  .NAND_(NAND_), .NOR_(NOR_), .XOR_(XOR_), .XNOR_(XNOR_));

    initial begin
        $display("A B | NOT AND OR NAND NOR XOR XNOR");
        $display("-----------------------------------");

        // Apply all input combinations
        A=0; B=0; #10;
        $display("%b %b |  %b    %b   %b    %b    %b   %b    %b",
                  A,B,NOT_,AND_,OR_,NAND_,NOR_,XOR_,XNOR_);

        A=0; B=1; #10;
        $display("%b %b |  %b    %b   %b    %b    %b   %b    %b",
                  A,B,NOT_,AND_,OR_,NAND_,NOR_,XOR_,XNOR_);

        A=1; B=0; #10;
        $display("%b %b |  %b    %b   %b    %b    %b   %b    %b",
                  A,B,NOT_,AND_,OR_,NAND_,NOR_,XOR_,XNOR_);

        A=1; B=1; #10;
        $display("%b %b |  %b    %b   %b    %b    %b   %b    %b",
                  A,B,NOT_,AND_,OR_,NAND_,NOR_,XOR_,XNOR_);

        $stop;
    end
endmodule
```

- **1.3) Schematic**



- **1.4) Wave Forms**



🔹 **Why is this "best"?**

- ✅ **Reduces hardware types → instead of carrying many gates, you just carry MUX blocks.**
- ✅ **Flexible → reconfigure a MUX to behave like any gate.**
- ✅ **Used in FPGAs → LUTs (Lookup Tables) inside FPGAs are basically tiny multiplexers storing truth tables. That's why MUX-based design is powerful in reconfigurable logic.**

⚡ **So, NAND/NOR are universal at the gate level,**
while MUX is universal at the circuit/system level → because you can build *any* Boolean function with it.