# Lab1

**Digital Logic Design**

**SystemVerilog (SV)**

# Welcome to Dr. Das's Lab!

- We hope you have a pleasant learning experience while working with this software.
- In this session, we will design **basic logic gates: NOT, AND, and OR** together.
- In the previous session, yo successfully installed **Modelsim software**. Now, it's time to start **writing code in the SystemVerilog** .

- For this purpose, we will need **three main files**:

1. **Verilog file** (to describe the logic circuit).
2. **Testbench file** (to test and evaluate the circuit's performance).
3. **DO file** (to automate the simulation process).

# What is a DO File?

- A **DO file** in ModelSim is a **script file** that automates the simulation process by executing a sequence of commands. Instead of manually entering each command in the ModelSim command line, a DO file allows you to **run multiple commands automatically** with a single execution.
- **Why is a DO File Important?**
- Without a **DO file**, we would have to **manually type every command** in ModelSim, such as:
- Compiling the Verilog files
- Loading the testbench
- Adding signals to the waveform
- Running the simulation
- This process would be **time-consuming and error-prone**, especially for large projects. By using a **DO file**, we can **automate** these tasks, ensuring **consistency** and **efficiency** in the simulation workflow.

```
onbreak {resume}

# create library
if [file exists work] {
    vdel -all
}
vlib work
# compile source files
vlog inv.sv inv_tb.sv

# start and run simulation
vsim -voptargs=+acc work.tb
view list
view wave
-- display input and output signals as hexidecimal values
# Diplays All Signals recursively
add wave -hex -r /tb/*
# add wave -noupdate -divider -height 32 "Datapath"
# add wave -hex /tb/dut/part1/*
# add wave -noupdate -divider -height 32 "Control"
# add wave -hex /tb/dut/part2/*
# add wave -noupdate -divider -height 32 "Note for Speaker"
# add wave -hex /tb/dut/part1/note1/*
# add wave -hex /tb/dut/part1/note2/*
# add wave -hex /tb/dut/part1/note3/*
# add wave -hex /tb/dut/part1/note4/*
add list -hex -r /tb/*
add log -r /*
-- Set Wave Output Items
TreeUpdate [SetDefaultTree]
WaveRestoreZoom {0 ps} {75 ns}
configure wave -namecolwidth 150
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
-- Run the Simulation
run 250 ns
```

**Handling Interruptions in the Simulation( ◇ Ensures that if the simulation pauses due to an error or manual stop, it resumes instead of terminating.)**

- Checks if the work library already exists.
- If it does, it deletes (vdel -all) everything inside it.
- Then, it creates a new library (vlib work) to store compiled simulation files.

- Compiles the Verilog/SystemVerilog source files:

inv.sv → The file (DUT - Device Under Test).

inv_tb.sv → The testbench file (stimulates the DUT).

- The command vlog is used in ModelSim for compilation.

- Loads the compiled testbench (tb) into ModelSim for simulation.
- Breakdown of options:

vsim → Starts the simulator.

-voptargs=+acc → Enables signal accessibility for debugging.

work.tb → Specifies the compiled testbench to simulate.

**Setting Up the Waveform View:**

- Opens two essential views in ModelSim:

List View → Displays signal values in tabular format.

Wave View → Displays signals as timing waveforms.

**Adding Signals to the Waveform:**

- Adds all signals inside the testbench (tb) to the waveform view.
- -hex → Displays signal values in hexadecimal format (useful for multi-bit signals).
- -r → Recursive addition, meaning all sub-signals inside tb will be added.

Using this DO file automates the entire simulation process, eliminating the need to manually enter commands in Modelsim!

- **Key Actions of This DO File**

1. Deletes and recreates the simulation library (work).
2. Compiles the Verilog files (inv.sv and inv_tb.sv).
3. Loads the testbench (tb) for simulation.
4. Adds signals to the waveform and list views.
5. Customizes the waveform display.
6. Runs the simulation for 250 ns.

# NOT GATE

inv.sv

C: > Users > Maryam > Desktop > Lab > Lab > inv.sv

```
1  module inv(input logic a, output logic y);
2
3    assign  y = ~a;
4
5  endmodule
6
```

- **The Code Begins with the "`module`" Keyword that** indicates the start of defining a new module.
- **Module Name inv** which is short for inverter.

**`input logic a:`**

- The input to this module is defined as a, which is of type logic.
- logic represents a binary signal that can take values of 0 or 1.

**`output logic y:`**

- The output of the module is defined as y, which is also of type logic, similar to the input.
- This output will be the inverted (NOT) value of the input a.

```
assign y = ~a;
```

**`assign:`**
This keyword is used to define combinational relationships between inputs and outputs.

**y:**
This is the output of the module.

**~a:**
The ~ operator is the logical NOT operator. It inverts the value of a:

- If a = 0, the output y = 1.
- If a = 1, the output y = 0.

**`endmodule:`**

- This line marks the end of the module definition. All the code related to this module is enclosed between the `module` and `endmodule` keywords.

# TESTBENCH

C: > Users > Maryam > Desktop > Lab > Lab > inv_tb.sv

```
1
2   `timescale 1ns / 1ps
3   module tb ();
4
5       logic a;  // Input to DUT
6       logic y;  // Output from DUT
7
8       // Instantiate the Device Under Test (DUT)
9       inv dut (
10          .a(a),
11          .y(y)
12      );
13
14      // Test vectors
15      initial begin
16          // Case 1: a = 0
17          a = 0;
18          #10; // Wait for 10 time units
19          if (y !== 1)
20              $display("Error: For a = 0, y = %b (expected 1)", y);
21          else
22              $display("Pass: For a = 0, y = %b", y);
23
24          // Case 2: a = 1
25          a = 1;
26          #10; // Wait for 10 time units
27          if (y !== 0)
28              $display("Error: For a = 1, y = %b (expected 0)", y);
29          else
30              $display("Pass: For a = 1, y = %b", y);
31
32          $finish; // End simulation
33      end
34
35  endmodule
36
```

**Line 1:**

- **timescale:** This line specifies the time unit in the simulation.
- **1ns:** Means each time unit in the simulation is equal to 1 nanosecond.
- **1ps:** Indicates that the simulation precision is up to 1 picosecond.

---

**Line 2:**

- **module tb ();:** This line indicates the start of the testbench. The name tb is short for Testbench.
- This module has no inputs or outputs since its sole purpose is to test the main circuit.

**Lines 3 and 4:**

- **logic a:** This line defines the input to the main module (Device Under Test, or DUT).
- **logic y:** This line defines the output from the main module (DUT).

---

**Lines 7 to 11:**

- **In this section:**
  - The main module inv, which was previously designed, is connected to the testbench.
  - The name dut is chosen for instantiating the DUT.
- **Connections:**
  - **Input:** The testbench input a is connected to the DUT input.
  - **Output:** The DUT output y is connected to the testbench output.

## Lines 24 to 31 (Case 2)

- In this section, the input value a is set to 1.
- After 10 time units:
  - If the output y is correct (0), it displays a "Pass" message.
  - If the output y is incorrect, it displays an "Error" message.

---

**Line 33:**

- **$finish;:**
  This command indicates the end of the simulation and terminates it.

---

**Line 35:**

- **endmodule:**
  This line marks the end of the testbench definition.

---

**Summary of Testbench Functionality:**

1. Tests the main module (inv).
2. Verifies the output in two scenarios:
   - When the input a = 0, the expected output is y = 1.
   - When the input a = 1, the expected output is y = 0.
3. Displays the result for each case as either "Pass" or "Error".

To simulate the testbench using **ModelSim**, follow these **step-by-step instructions**:

## Step 1: Navigate to the Folder Containing the Required Files

(Make sure that your three necessary files are in the same folder):

**Open the Terminal in the Correct Folder:**

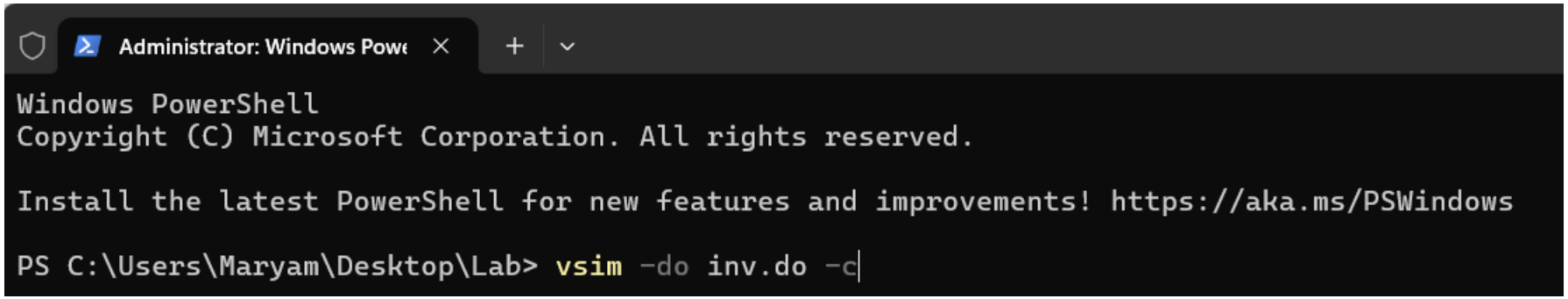
- ❑ Go to the folder containing these files (inv.sv, inv_tb.sv, inv.do).
- ❑ Right-click inside the folder and select "Open in Terminal" (or "Open PowerShell Here" in Windows).
- ❑ You should now be inside the correct directory in the terminal

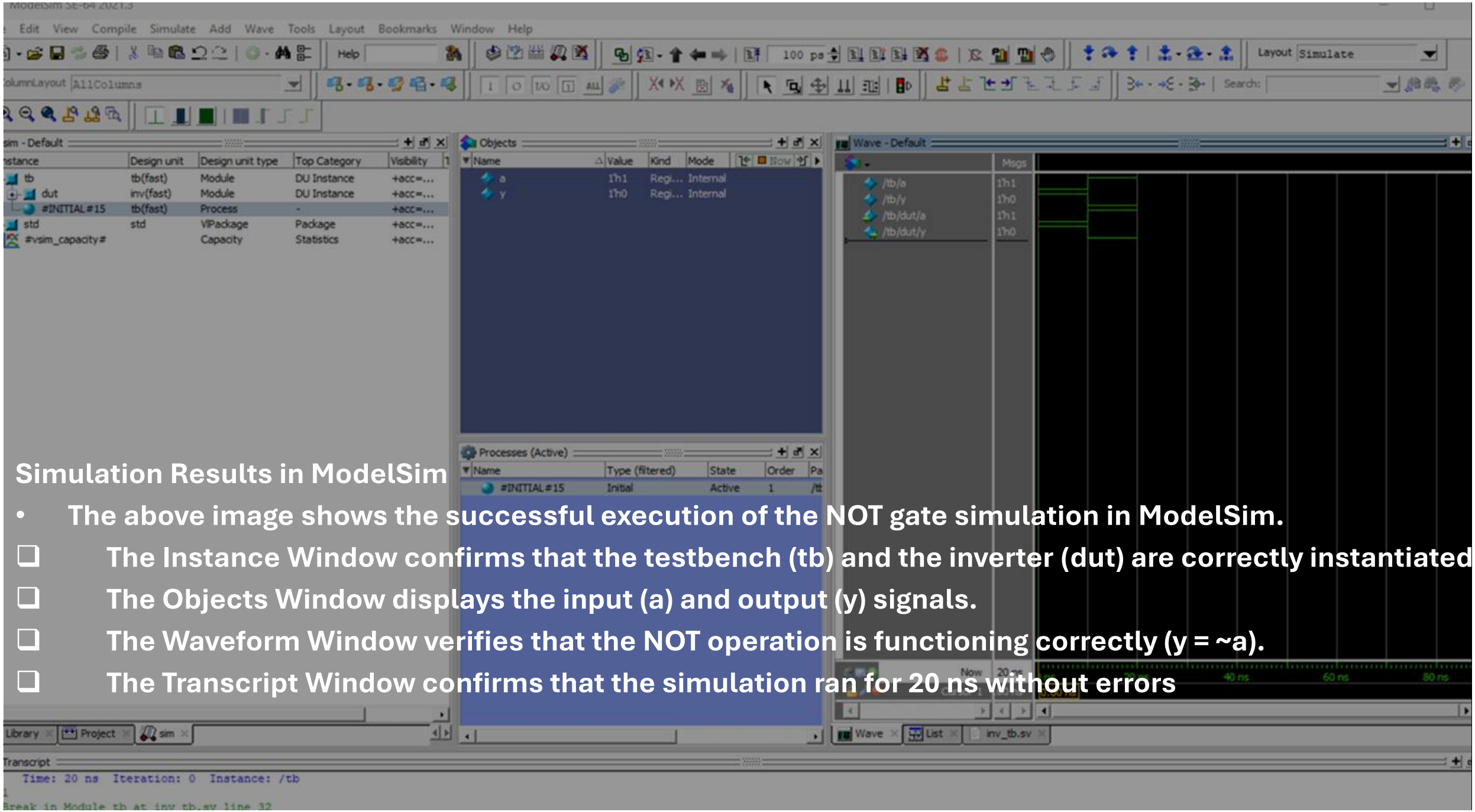
## Step 2: Run the DO File in ModelSim (In the terminal, enter the following command and press Enter): **vsim -do inv.do -c**

- ❑ **-c** → Runs ModelSim in command-line mode (without opening the GUI).
- ❑ **vsim** → Runs the ModelSim simulator.
- ❑ **-do inv.do** → Executes the DO file (inv.do), which automates the simulation.

If you want to see the GUI while running the simulation, remove -c: **vsim -do inv.do**

The graphical environment of ModelSim opens, and you can view signals in the Waveform window.






## Simulation Results in ModelSim

- The above image shows the successful execution of the NOT gate simulation in ModelSim.
  - ❑ The Instance Window confirms that the testbench (tb) and the inverter (dut) are correctly instantiated
  - ❑ The Objects Window displays the input (a) and output (y) signals.
  - ❑ The Waveform Window verifies that the NOT operation is functioning correctly (y = ~a).
  - ❑ The Transcript Window confirms that the simulation ran for 20 ns without errors

# Practice More – Implement AND & OR Gates

To further strengthen your understanding of combinational logic, **please implement the AND and OR gates** in Verilog.

- Design and test both gates following the same approach as the NOT gate.
- Use **SystemVerilog** and simulate your circuits in **ModelSim**.

- For additional examples and reference codes, you can explore the following repository:
  **https://github.com/MaryamShahangian/DLD_DrDas.git**

🔗 **GitHub Repository – DLD Dr. Das**