EE-421 Digital System Design Lab 02 (Fall 2025)

Lab Manual

Electrical Engineering Department
LUMS

Instructor: Dr. Shahid Masud
TA: Syed Muhammad Tufail

Objectives

- Understanding HDL based design flow
- Identifying the components of a SystemVerilog module definition
- Understanding how to define the port list for a module and declare it in SystemVerilog
- Familiarization with the logic gate primitives provided in SystemVerilog
- Understanding instantiation of primitive gates and construction of a SystemVerilog description from the logic diagram of the circuit
- Writing simple Testbenches in SystemVerilog
- Simulation of basic building blocks of digital circuits in SystemVerilog using Questa simulator

Table of Contents

1 Introduction to SystemVerilog HDL 3
1.1 Why SystemVerilog? 3
1.2 Synthesis or Synthesizable code 3
2 SystemVerilog Basics 3
2.1 Module 3
2.2 Module Declaration 4
2.3 Ports 4
2.4 Testbench/Stimulus 5
2.5 Identifiers and Keywords 5
2.6 Variable declaration 5
3 Gate level or structural modeling 6
3.1 Simulation 6
4 Getting Started Using Mentor Graphics Questa 7
4.1 Testbench Approach 9
5 Lab Tasks 11
6 Appendix 13

1 Introduction to SystemVerilog HDL

SystemVerilog standardized as **IEEE 1800** is a Hardware Description Language (HDL), used to design digital circuits using FPGA.

1.1 Why SystemVerilog?

SystemVerilog is relatively simple and close to **C** while VHDL is complex and close to **Ada.** SystemVerilog has 33% of the world digital design market (larger share in the US) [it varies from designer and organization].

1.2 Synthesis or Synthesizable Code

The portion of code or design that can be converted to physical hardware or can be easily implemented. For FPGAs, the synthesis creates gate patterns and maps the logic described by the program.

The non-synthesizable statements in a code are written to help in modelling and simulation of different hardware blocks.

2 SystemVerilog Basics

NOTE: SystemVerilog is a case sensitive language.

SystemVerilog has four main levels of abstraction:

1. **Switch level modeling:** The lowest level of modeling is the switch level or transistor level modeling. This can only help in designing small specialized circuits. Only few hardware designers work at the switch level.
2. **Gate level modeling:** Logic gates are used to design the program logic. For example, *AND, OR, NOT. Usually the lowest level at which hardware designers work.*
3. **Dataflow modeling:** Here the data flows through the registers. The designer is aware of the process of data flow in the registers. Assign keyword and operators (logic and arithmetic) are used. *This is a commonly used approach in SystemVerilog.*
4. **Behavioral modeling:** It is the highest level of abstraction. Implementation can be done according to the desired algorithm design. Programming at this level is similar to C. Loops can be used to design the module. *This is also popular, though some code could be non-synthesizable.*

2.1 Module

The basic building block of SystemVerilog. It can be a combination of lower-level design blocks. It has a name, port list, and internals (functionality). The keywords **module** and **endmodule** must appear at the beginning and end of the module definition. The Same Module cannot be nested within itself.

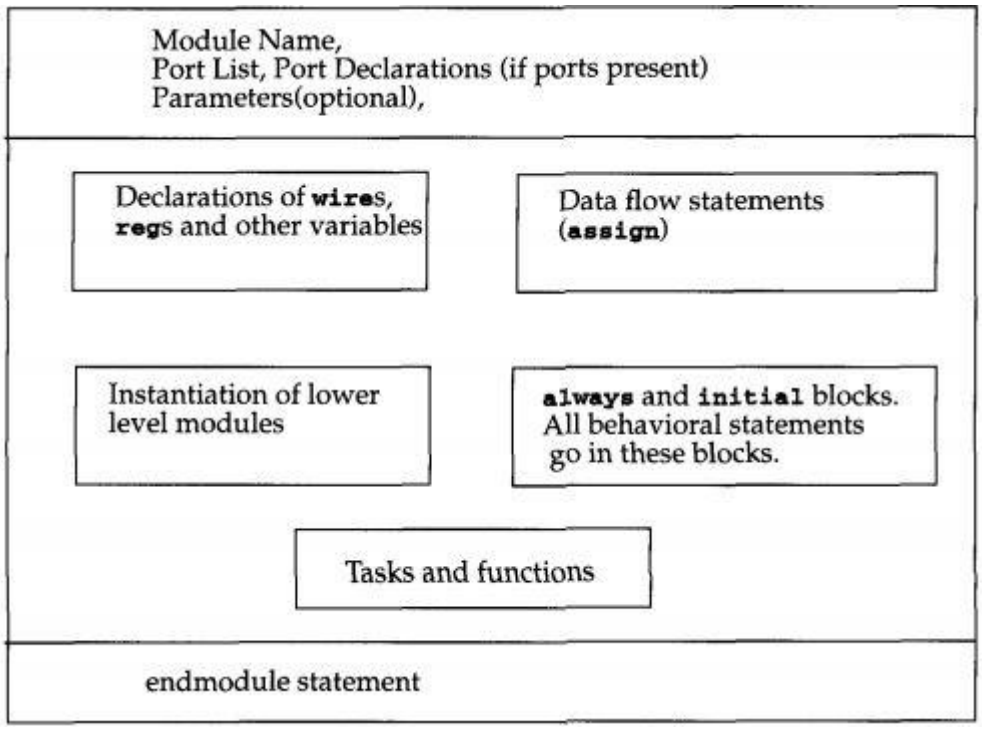


Figure 1: Components of a SystemVerilog Module

2.2 Module Declaration

Example 1.1. SystemVerilog Wrapper

Figure 2 shows the basic skeleton of a generic design module written in SystemVerilog. It consists of module name, port list and the internal details of the module.

```
module <module_name> (<module_terminal_list>);

       - ;

       <module internals>;

       - ;

       - ;

endmodule
```

Figure 2: SystemVerilog Module Wrapper

Common pitfall: endmodule is a single keyword and must not be written like ~~end module~~ as this will allow the parser to treat the module and end separately which are different keywords.

2.3 Ports:

Ports provide the module with a means to communicate with other modules or its environment. A module can have a port list. Ports in the port list must be declared as ***input, output,*** or ***inout.***

When instantiating a module, port connection rules are enforced by the SystemVerilog simulator.

Example 1.2. Port list explained

```
module ds (y, x); // port list

       input x;
       output y;

       // user logic

endmodule
```

```
module my_module
 (     input wire, clk,
       input wire rst_n,
       output wire y_out );

// user logic

endmodule
```

Figure 3: Example of a Basic SystemVerilog Module

SystemVerilog Convention: outputs generally come first in port list than input as shown in Figure 3. New style in SystemVerilog is on the right. Both are acceptable.

2.4 Testbench / Stimulus

The module functionality is tested by creating a stimulus in which the inputs are defined to drive the outputs as shown in Figure 4. **Stimulus is not synthesizable.**

```
module stimulus;
  reg a, b; // inputs to test circuit
  wire c;   // outputs from test circuit

  dut a1 (c, a, b);

  initial
  begin
       a=1'b0;
       b=1'b0;
  end

endmodule
```

Figure 4: Example of a basic SystemVerilog Testbench File

2.5 Identifiers and Keywords

Identifiers are the names given to the objects so that they can be referred anywhere in the design while keywords are special identifiers used to describe the language constructs.

```
reg value;   // reg is a keyword; value is an identifier
input clk;   // input is a keyword; clk is an identifier
```

2.6 Variable Declaration

- **Registers**

Registers (*reg*) are used to store values. Usually, inputs in the testbench /stimulus are declared as *"reg*" as they are to be stored until a new value is passed during simulation.

```
reg [1:0] sum;
```

- **Nets / Wires:**

Nets or wires do not store values they just pass on them to the other part of the design. Outputs in the testbench / stimulus are declared as *"wires"*.

```
wire [1:0] value
```

3 Gate-level or Structural Modeling and Simulation

3.1 Gate-Level Design

- In gate-level modeling, circuits are built from gate-level primitives.
- We can describe the circuit using the logic gates.
- SystemVerilog has built-in gate-level primitives NAND, NOR, AND, OR, XOR, BUF, NOT and some others.

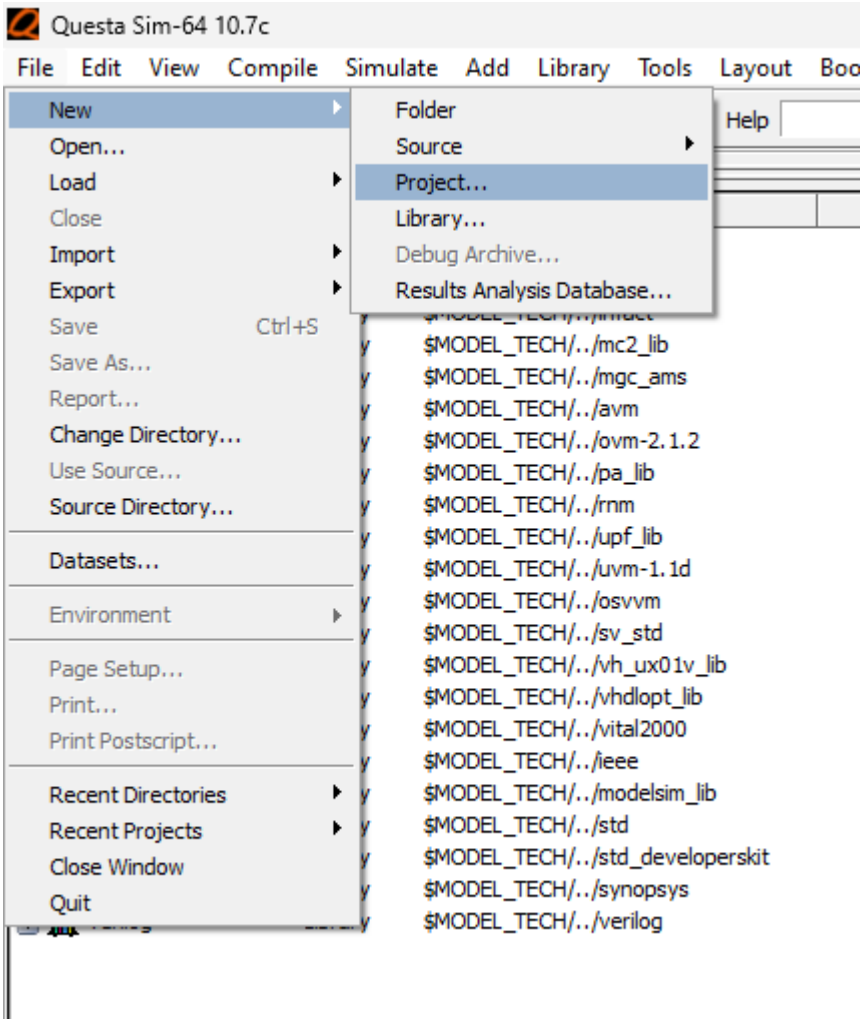
3.2 Simulation

Simulators are available from many vendors for various prices. For desktop/personal use, Aldec, Mentor, LogicSim, SynaptiCAD, TarangEDA Altera Quartus and others offer > $5000+ USD tool-suites.

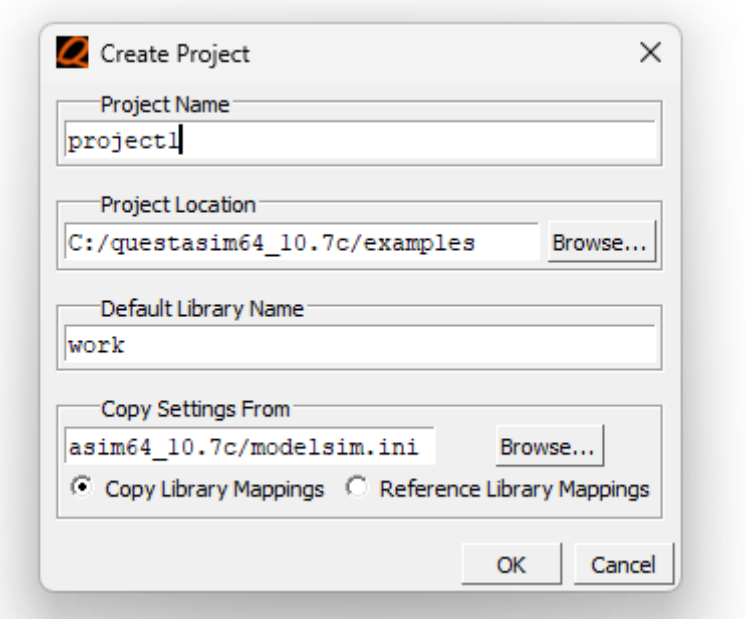
In this lab, we will be using ***Questa® by Mentor Graphics (now Siemens)***. *Questa* is an easy- to-use yet versatile VHDL/(System)Verilog/C simulator by Mentor Graphics. It supports behavioral, register transfer level (RTL), and gate-level modeling. Questa can be found preinstalled on the lab computers.

4 Getting Started using Mentor Graphics Questa

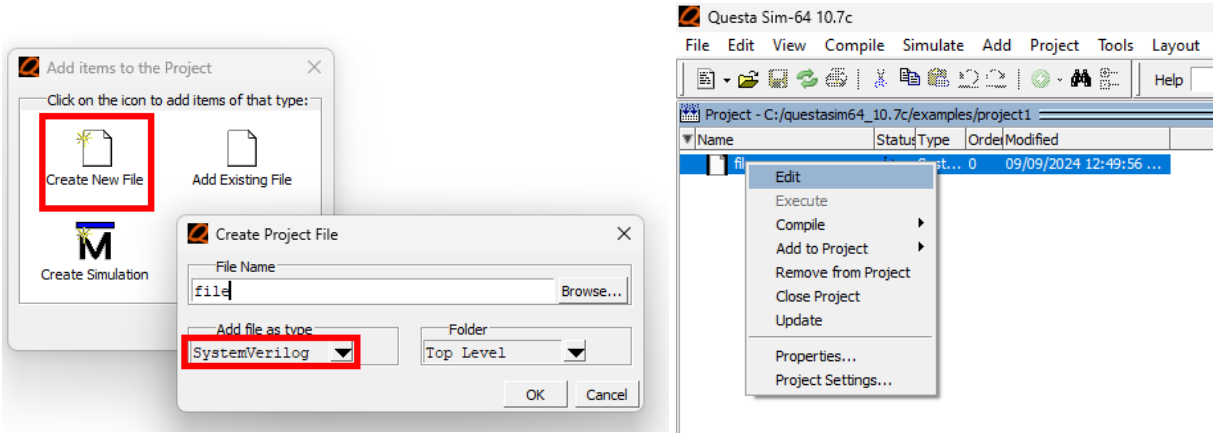
1. Open Questa
2. Go to File → New → Project and click Project.



3. This window will pop up. Name your project something useful, in this case, **`project1`** was the name chosen. Add the name of the project onto the project location. This keeps the project in an easy to find a place.



4. If this is the first project most likely you will want to create a new file. Click the `Create New File` button and a window pops up. A project can have more than one files so name this file according to its functionality.



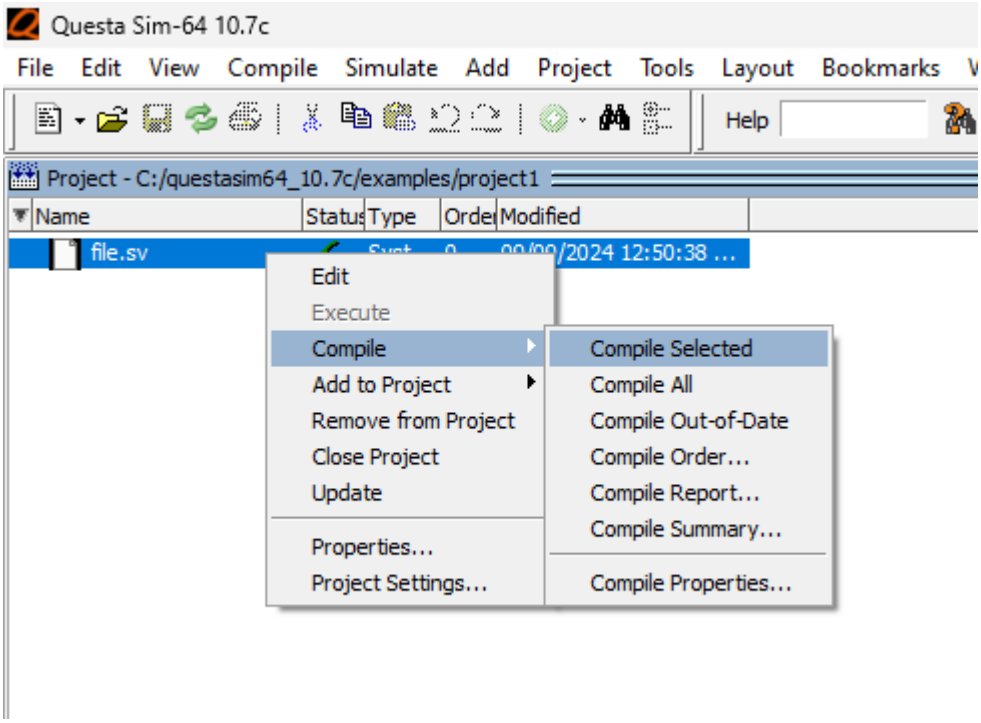
5. Right click the file name and select "`Edit`". Now you are going to write SystemVerilog code in this file, Use the following code, this is 2-input and gate.

```
//This module designs the 2 input and one //output and gate.
module andgate (y, a, b);

input a, b; output y;

and a2(y, a, b); endmodule
```

Inputs

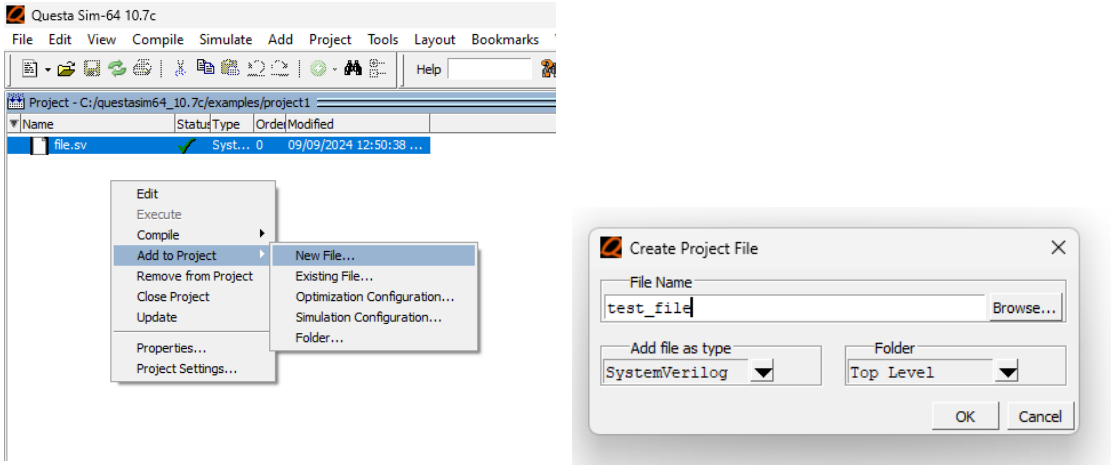
Output

6. After you complete the code, Right click on file name and select `Compile → Compile Selected`. This step will compile the SystemVerilog file. Remove the bugs if any highlighted by the compiler output.



4.1 Testbench / Stimulus

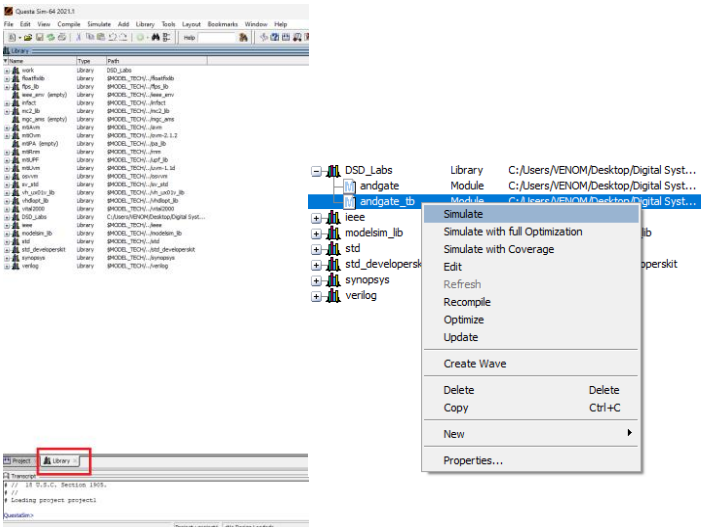
7. Now create another SystemVerilog file to write the Testbench for your code.



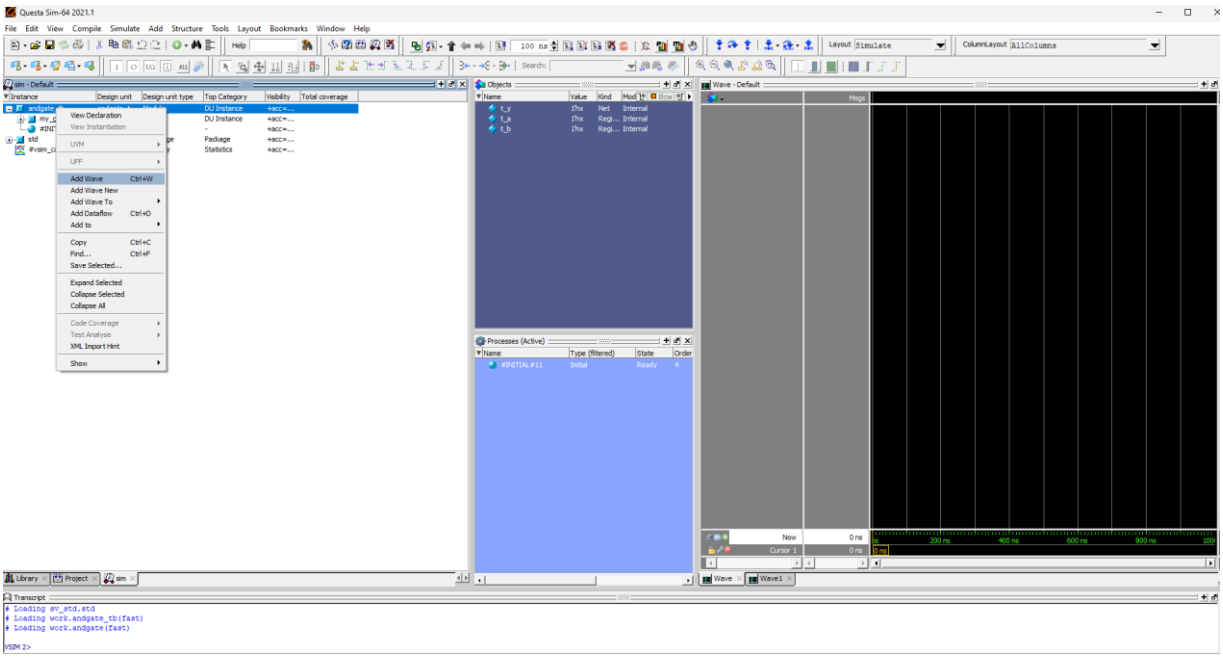
8. Use the following code for the Testbench of and gate SystemVerilog file. Compile the testbench file using step 6 again. **Make sure you get no error.**

```
// add the code for testfile here
`timescale 1ns/1ns

module andgate_tb;

	wire t_y;
	reg t_a, t_b;

	andgate my_gate(t_y, t_a, t_b);

	initial
	begin

		t_a = 1'b0;
		t_b = 1'b0;

		#5
		t_a = 1'b0;
		t_b = 1'b1;

		#5
		t_a = 1'b1;
		t_b = 1'b0;

		#5
		t_a = 1'b1;
		t_b = 1'b1;
	end

endmodule
```

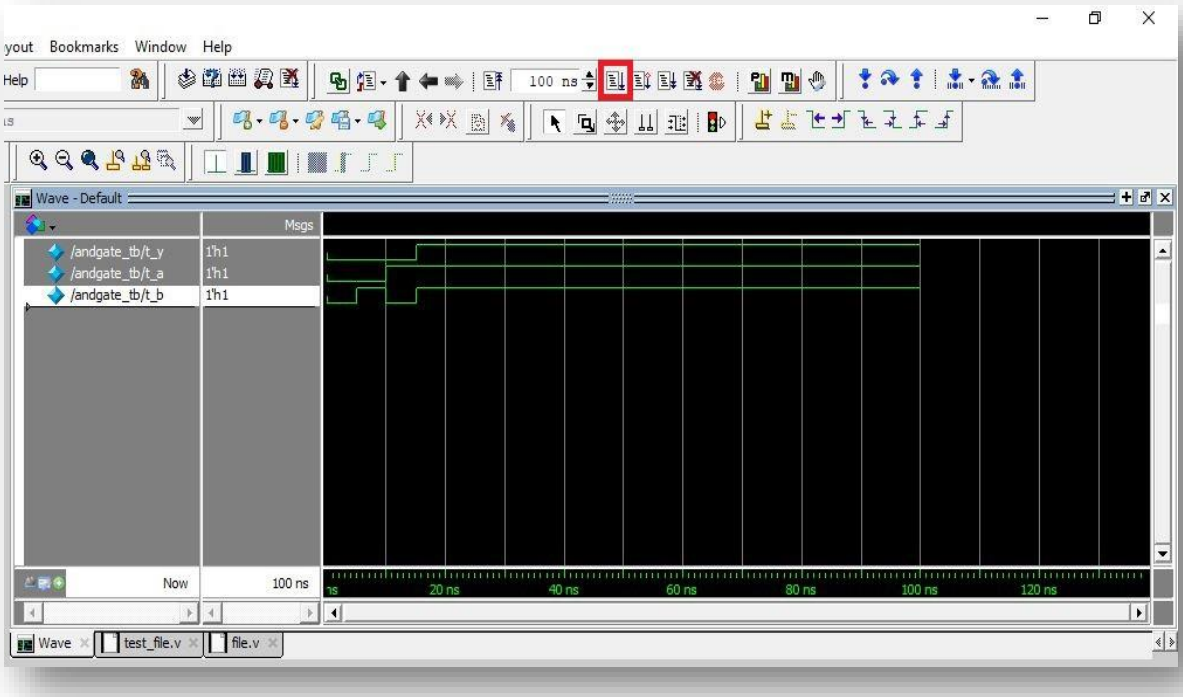
9. Go to Library Tab and then Select `testbench` file from `work` directory / library to start simulation. Right Click and select `Simulate`.



10. Once a file for simulation is selected, the layout will change to "`Simulation`". Following window will pop up. Right click on testbench file and select "`Add Wave`".



11. Select "Run". Make sure you get green colored wave forms and check your logic as well. You can press "I" to zoom in, "O" to zoom out and "F" for zooming upto full simulation time.



5 Lab Tasks

Lab Task Instructions:

1. **Each Lab Task should be a separate Project in SystemVerilog and Questa.**
2. **The document containing lab deliverables should be named as DSD_Deliverable_Lab1_RegNumber .**
3. **The template of Lab Deliverable document is provided on lab LMS page.**

Submission Instructions on LMS:

- **Submission files for each lab deliverable document must be placed in a separate folder.**
- **Usually there will be multiple tasks in a lab. Submission files for each lab task:**
  a. **Truth table (Design details) [if required]**
  b. **Circuit Diagram**
  c. **SystemVerilog code + Testbench code**
  d. **Simulation waveform (annotated to show correct working of circuit)**
  e. **Conclusion/Summary/Analysis [if required]**
- **Finally, a single compressed Lab Deliverable file should be uploaded on LMS. Name of the folder should contain student's reg number.**

Note:

- **For any circuit or table drawn on paper, students can take a picture of their work and insert in lab deliverable document.**
- **The code files for design and Testbench must be uploaded.**
- **For timing waveform, screenshot of the simulation should be inserted.**

Deadline for Submitting Complete Lab Deliverable file on LMS dropbox is 11:55pm on Friday, 12 September 2025 (today)

See the Lab Deliverables file for a list of tasks for today's lab.

6 Appendix

- **Gate Primitives in SystemVerilog**

The gates have one scalar output and multiple scalar inputs. The 1st terminal in the list of gate terminals is an output and the other terminals are inputs.

Gate	Description
and	N-input AND gate
nand	N-input NAND gate
or	N-input OR gate
nor	N-input NOR gate
xor	N-input XOR gate
xnor	N-input XNOR gate
Not	One input NOT gate

The name of these gates are keywords and cannot be used as identifiers (like variable name in c).

- **How to use these Primitive gates**

```
<<name of primitive gate>> <<instance name>> (<<output identifier>>, <<input identifiers>>, …)
```

Example:

Write SystemVerilog description for 3 input one output and gate using gate primitive

```
and a2 (y, a, b, c)
```