CS227- Lab 10

Digital design using HDL(Verilog)

The goal of this lab is to get familiar with digital modeling in Verilog Hardware Description Language (HDL) and to learn how to handle the simulator. The goal is also to interpret outputs from a simulator and to understand how Verilog code is interpreted by the simulator.

For this purpose we will use **modelsim** simulator.

Modelsim Installation link for Windows:

https://www.intel.com/content/www/us/en/software-kit/660907/intel-quartus-prime-lite-edition-design-software-version-20-1-1-for-windows.html

Modelsim Installation link for Linux:

https://www.intel.com/content/www/us/en/software-kit/661017/intel-quartus-prime-lite-edition-design-software-version-20-1-for-linux.html

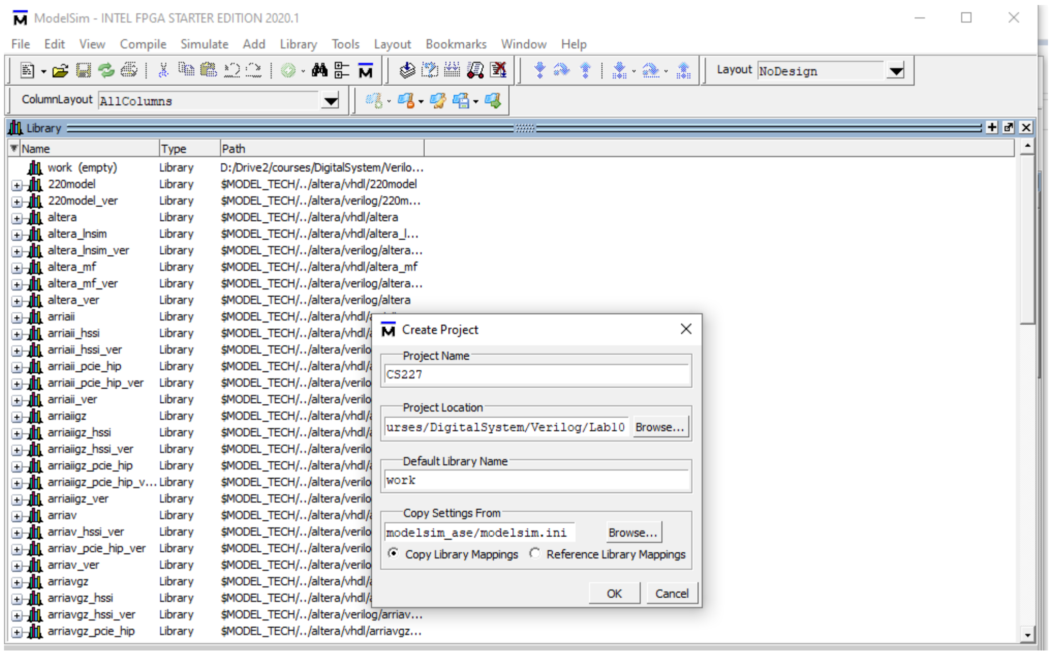
In your local system, create a lab directory for the course (say. D:/CS227/Verilog). Download the Lab10 files .

Start modelsim by the following steps

Start -> programes->modelsim->

Create a project (File->New->Project)

Name the project and project location as shown in the following figure.


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Library Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library

Name	Type	Path
work (empty)	Library	D:/Drive2/courses/DigitalSystem/Verilo…
220model	Library	$MODEL_TECH/../altera/vhdl/220model
220model_ver	Library	$MODEL_TECH/../altera/verilog/220m…
altera	Library	$MODEL_TECH/../altera/vhdl/altera
altera_lnsim	Library	$MODEL_TECH/../altera/vhdl/altera_l…
altera_lnsim_ver	Library	$MODEL_TECH/../altera/verilog/altera…
altera_mf	Library	$MODEL_TECH/../altera/vhdl/altera_mf
altera_mf_ver	Library	$MODEL_TECH/../altera/verilog/altera…
altera_ver	Library	$MODEL_TECH/../altera/verilog/altera
arriaii	Library	$MODEL_TECH/../altera/vhdl/
arriaii_hssi	Library	$MODEL_TECH/../altera/vhdl/
arriaii_hssi_ver	Library	$MODEL_TECH/../altera/verilo
arriaii_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/
arriaii_pcie_hip_ver	Library	$MODEL_TECH/../altera/verilo
arriaii_ver	Library	$MODEL_TECH/../altera/verilo
arriaiigz	Library	$MODEL_TECH/../altera/vhdl/
arriaiigz_hssi	Library	$MODEL_TECH/../altera/vhdl/
arriaiigz_hssi_ver	Library	$MODEL_TECH/../altera/verilo
arriaiigz_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/
arriaiigz_pcie_hip_v…	Library	$MODEL_TECH/../altera/verilo
arriaiigz_ver	Library	$MODEL_TECH/../altera/verilo
arriav	Library	$MODEL_TECH/../altera/vhdl/
arriav_hssi_ver	Library	$MODEL_TECH/../altera/verilo
arriav_pcie_hip_ver	Library	$MODEL_TECH/../altera/verilo
arriav_ver	Library	$MODEL_TECH/../altera/verilo
arriavgz	Library	$MODEL_TECH/../altera/vhdl/
arriavgz_hssi	Library	$MODEL_TECH/../altera/vhdl/
arriavgz_hssi_ver	Library	$MODEL_TECH/../altera/verilog/arriav…
arriavgz_pcie_hip	Library	$MODEL_TECH/../altera/vhdl/arriavgz…

Create Project

Project Name: CS227

Project Location: urses/DigitalSystem/Verilog/Lab10 Browse...

Default Library Name: work

Copy Settings From: modelsim_ase/modelsim.ini Browse...

[✓ Copy Library Mappings] Reference Library Mappings

OK Cancel

Figure 1: Mdelsim: create project option

Then appropriate option to be selected for adding files in the project. The options are shown as below.

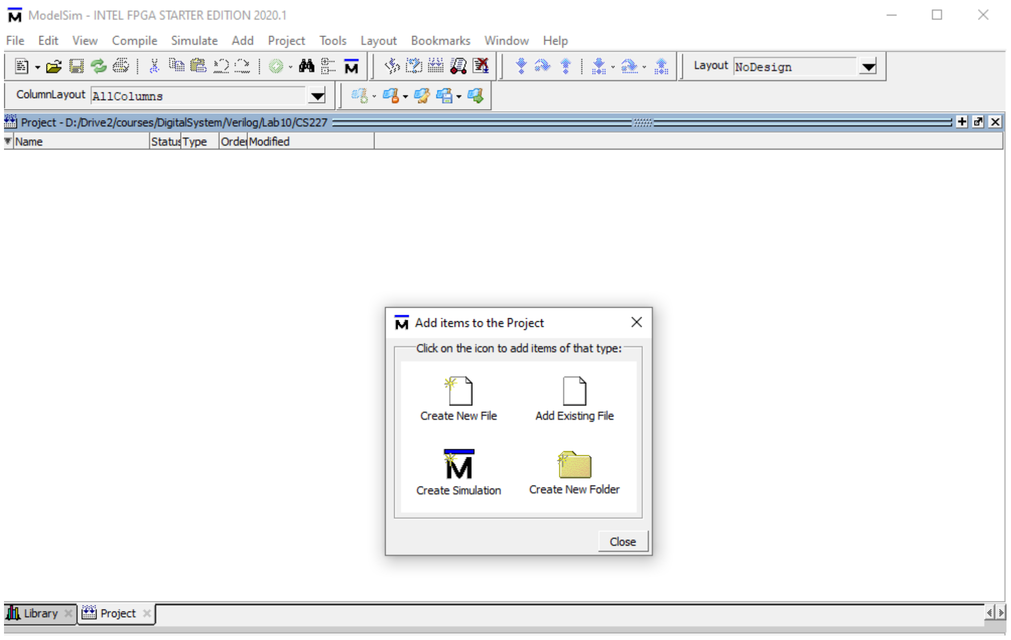
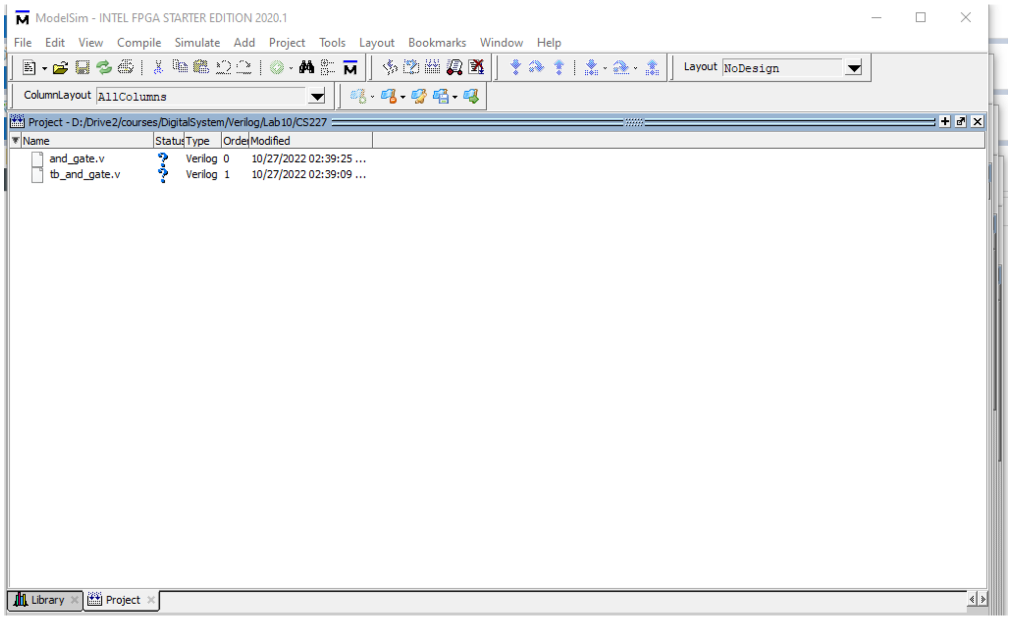

ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Project Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Project - D:/Drive2/courses/DigitalSystem/Verilog/Lab10/CS227
Name
Statu
Type
Orde
Modified
Add items to the Project
Click on the icon to add items of that type:
Create New File
Add Existing File
Create Simulation
Create New Folder
Close
Library
Project

Figure 2: Modelsim: add items to the project option

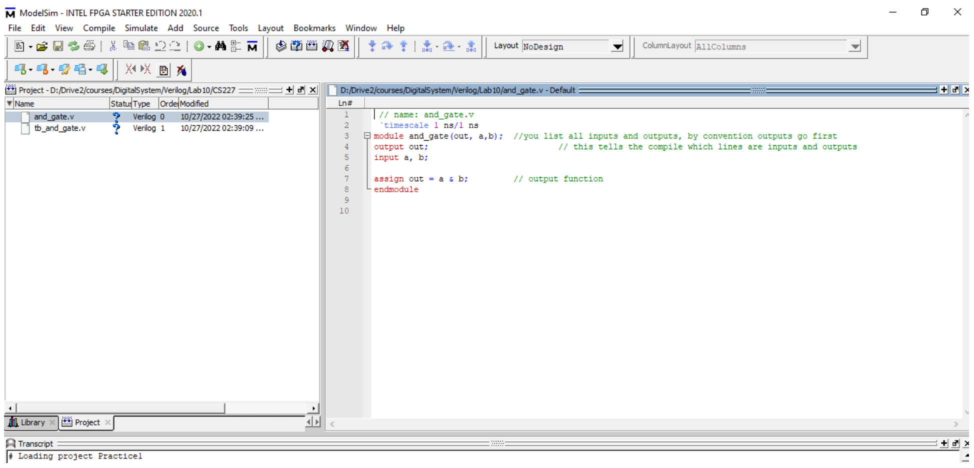
After adding files (say a AND design file and the corresponding testbench file), they will be shown as below.


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Project Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library
Project

```
Project - D:/Drive2/courses/DigitalSystem/Verilog/Lab10/CS227
Name          Status Type    Order Modified
and_gate.v    ?      Verilog 0     10/27/2022 02:39:25 ...
tb_and_gate.v ?      Verilog 1     10/27/2022 02:39:09 ...
```

Figure 3: Modelsim: after adding the files

After adding the files, to show the contents, you need to double click on the file name. The corresponding file will then be available for editing purpose. Double clicking the design file will open it in a separate window as shown below.


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library
Project
Transcript

```
Project - D:/Drive2/courses/DigitalSystem/Verilog/Lab10/CS227
Name          Status Type    Order Modified
and_gate.v    ?      Verilog 0     10/27/2022 02:39:25 ...
tb_and_gate.v ?      Verilog 1     10/27/2022 02:39:09 ...
```

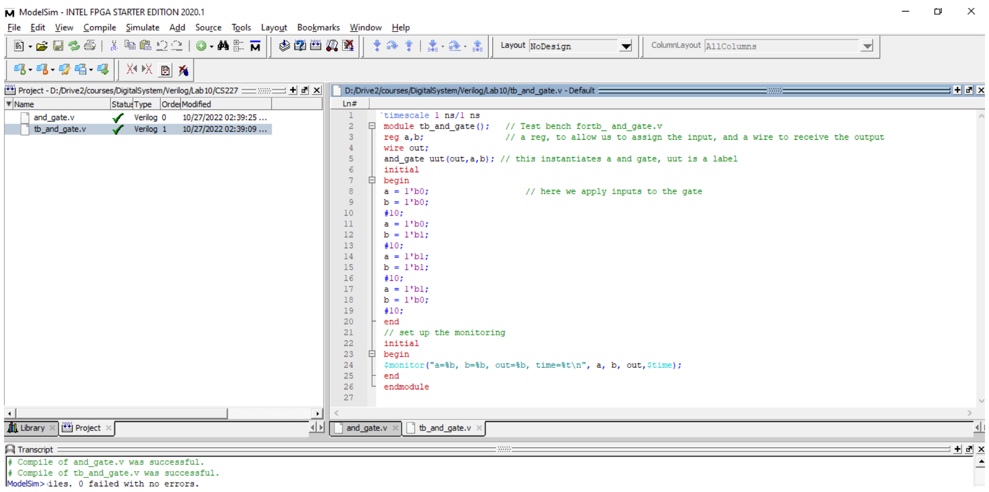
D:/Drive2/courses/DigitalSystem/Verilog/Lab10/and_gate.v - Default

```verilog
 // name: and_gate.v
 `timescale 1 ns/1 ns
module and_gate(out, a,b);  //you list all inputs and outputs, by convention outputs go first
output out;                           // this tells the compile which lines are inputs and outputs
input a, b;

assign out = a & b;         // output function
endmodule


```

```
# Loading project Practice1
```

Figure 4: Modelsim: Displaying the design file

In a similar way, the test bench file will be available in a separate window.

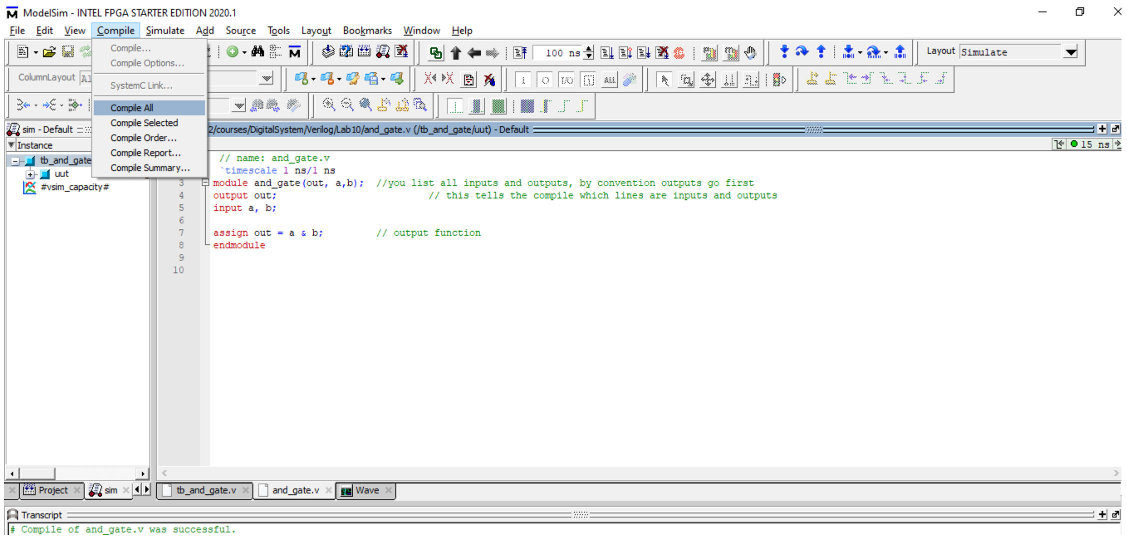


```
`timescale 1 ns/1 ns
module tb_and_gate();   // Test bench fortb_ and_gate.v
reg a,b;                // a reg, to allow us to assign the input, and a wire to receive the output
wire out;
and_gate uut(out,a,b); // this instantiates a and gate, uut is a label
initial
begin
a = 1'b0;                    // here we apply inputs to the gate
b = 1'b0;
#10;
a = 1'b0;
b = 1'b1;
#10;
a = 1'b1;
b = 1'b1;
#10;
a = 1'b1;
b = 1'b0;
#10;
end
// set up the monitoring
initial
begin
$monitor("a=%b, b=%b, out=%b, time=%t\n", a, b, out,$time);
end
endmodule
```

```
# Compile of and_gate.v was successful.
# Compile of tb_and_gate.v was successful.
ModelSim> iles. 0 failed with no errors.
```

Figure 5: Modelsim: Displaying the testbench file

The codes can be compiled using the compile all option as shown below.

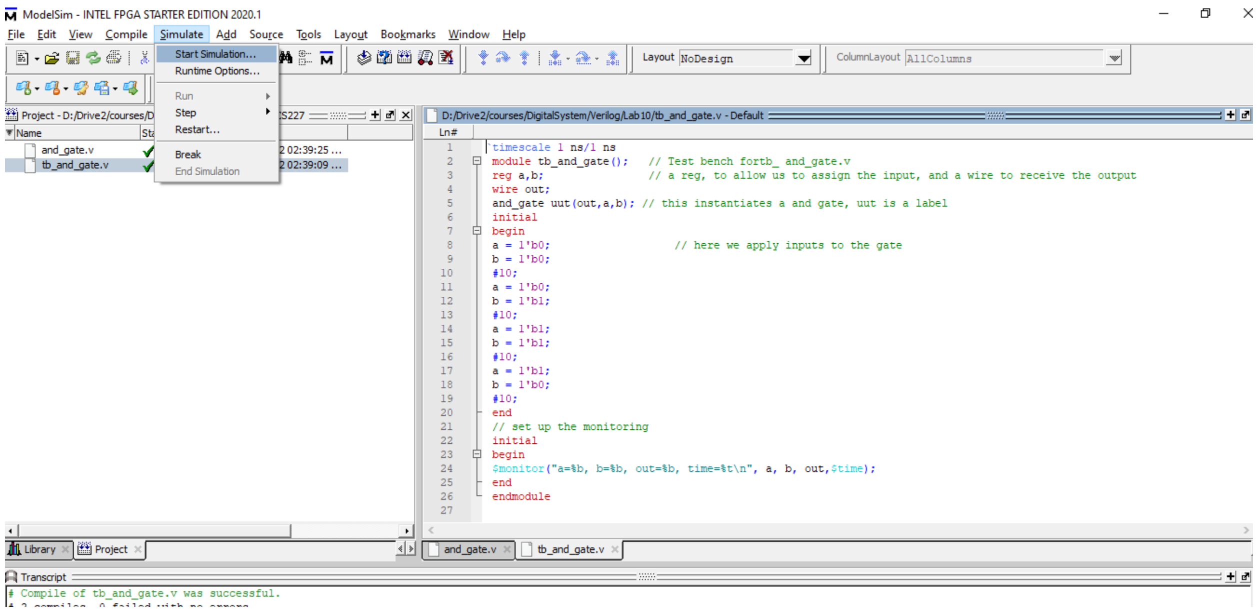


```
// name: and_gate.v
`timescale 1 ns/1 ns
module and_gate(out, a,b);  //you list all inputs and outputs, by convention outputs go first
output out;                             // this tells the compile which lines are inputs and outputs
input a, b;

assign out = a & b;         // output function
endmodule
```

```
# Compile of and_gate.v was successful.
```

Figure 6: Modelsim: Compile All option

Once compilation is successfully done then you are ready for the simulation. Go to simulation menu and select Start Simulation as shown below.

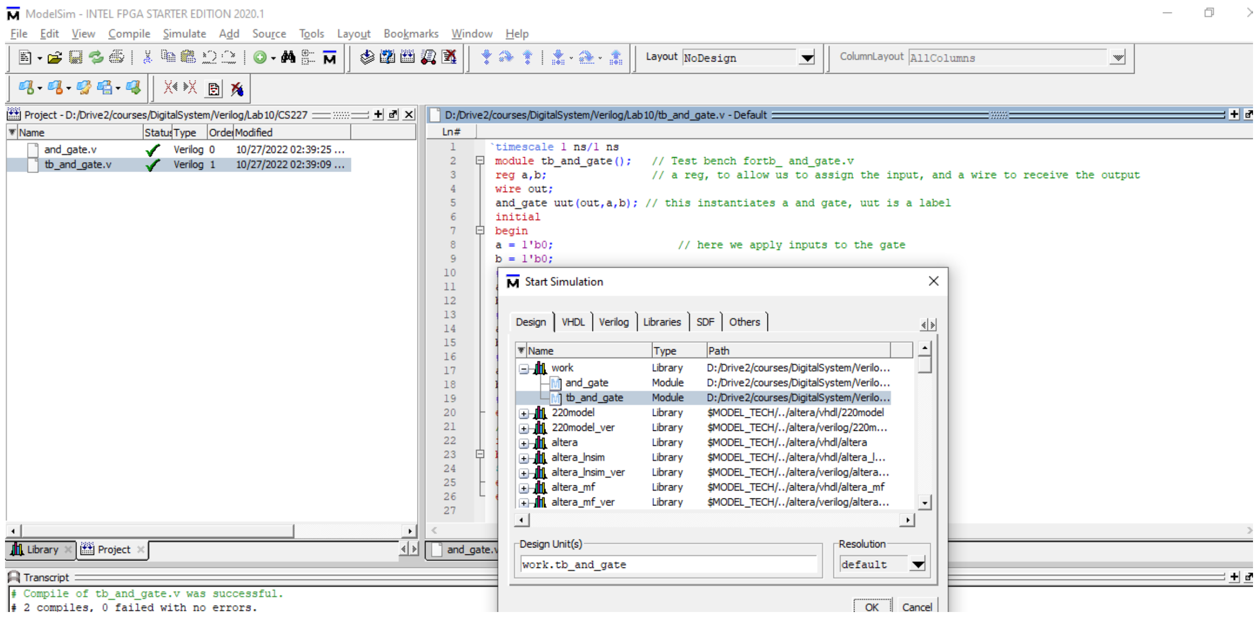


```
`timescale 1 ns/1 ns
module tb_and_gate();   // Test bench forth_ and_gate.v
reg a,b;                // a reg, to allow us to assign the input, and a wire to receive the output
wire out;
and_gate uut(out,a,b); // this instantiates a and gate, uut is a label
initial
begin
a = 1'b0;                    // here we apply inputs to the gate
b = 1'b0;
#10;
a = 1'b0;
b = 1'b1;
#10;
a = 1'b1;
b = 1'b1;
#10;
a = 1'b1;
b = 1'b0;
#10;
end
// set up the monitoring
initial
begin
$monitor("a=%b, b=%b, out=%b, time=%t\n", a, b, out,$time);
end
endmodule
```

```
# Compile of tb_and_gate.v was successful.
```

Figure 7: Modelsim: start simulation

Expand the work folder and select the testbench file for starting the simulation as shown in the following figure.



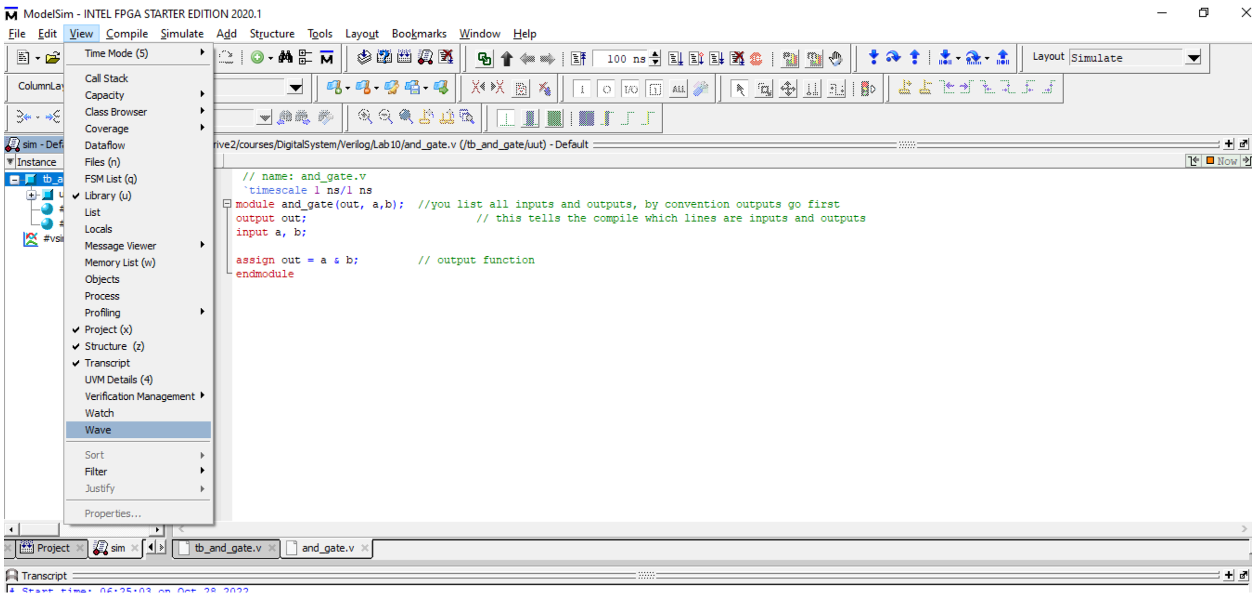
```
# Compile of tb_and_gate.v was successful.
# 2 compiles, 0 failed with no errors.
```

Name	Type	Path
work	Library	D:/Drive2/courses/DigitalSystem/Verilo...
and_gate	Module	D:/Drive2/courses/DigitalSystem/Verilo...
tb_and_gate	Module	D:/Drive2/courses/DigitalSystem/Verilo...
220model	Library	$MODEL_TECH/../altera/vhdl/220model
220model_ver	Library	$MODEL_TECH/../altera/verilog/220m...
altera	Library	$MODEL_TECH/../altera/vhdl/altera
altera_lnsim	Library	$MODEL_TECH/../altera/vhdl/altera_l...
altera_lnsim_ver	Library	$MODEL_TECH/../altera/verilog/altera...
altera_mf	Library	$MODEL_TECH/../altera/vhdl/altera_mf
altera_mf_ver	Library	$MODEL_TECH/../altera/verilog/altera...

Design Unit(s): work.tb_and_gate
Resolution: default

Figure 8: Modelsim: selecting the testench file for starting the simulation

For showing the simulation in wave form, go to view and select the wave option.



```
// name: and_gate.v
`timescale 1 ns/1 ns
module and_gate(out, a,b);   //you list all inputs and outputs, by convention outputs go first
output out;                          // this tells the compile which lines are inputs and outputs
input a, b;

assign out = a & b;          // output function
endmodule
```

Figue 9: Modelsim: wave view opening

Once the wave view option is clicked, it will pop up another window as shown below.

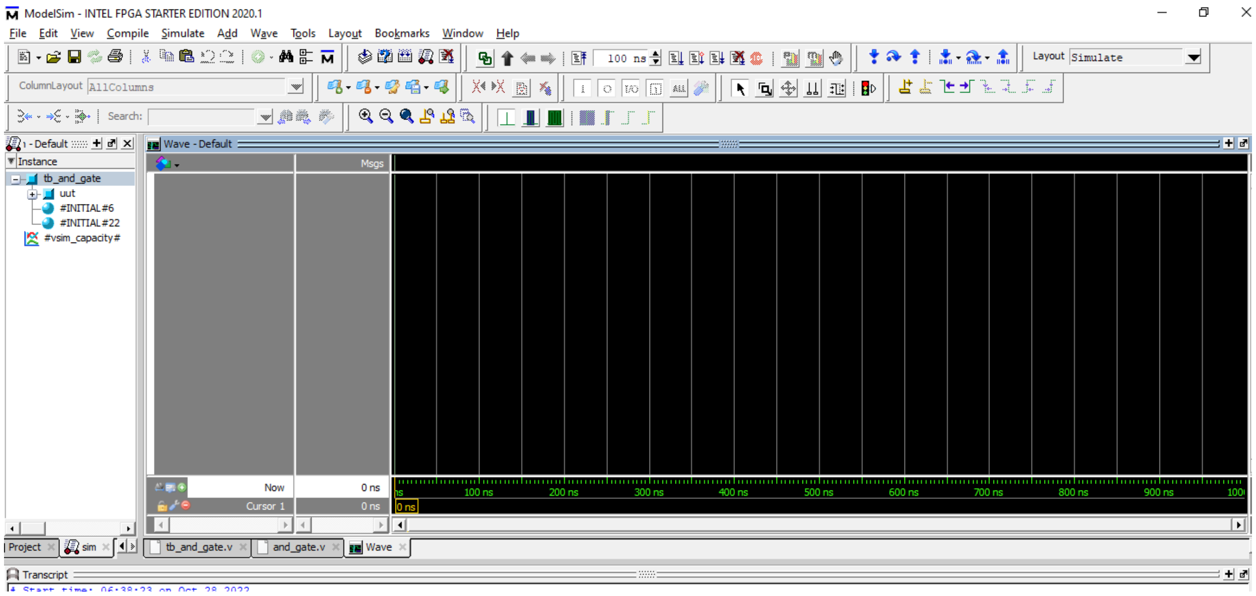
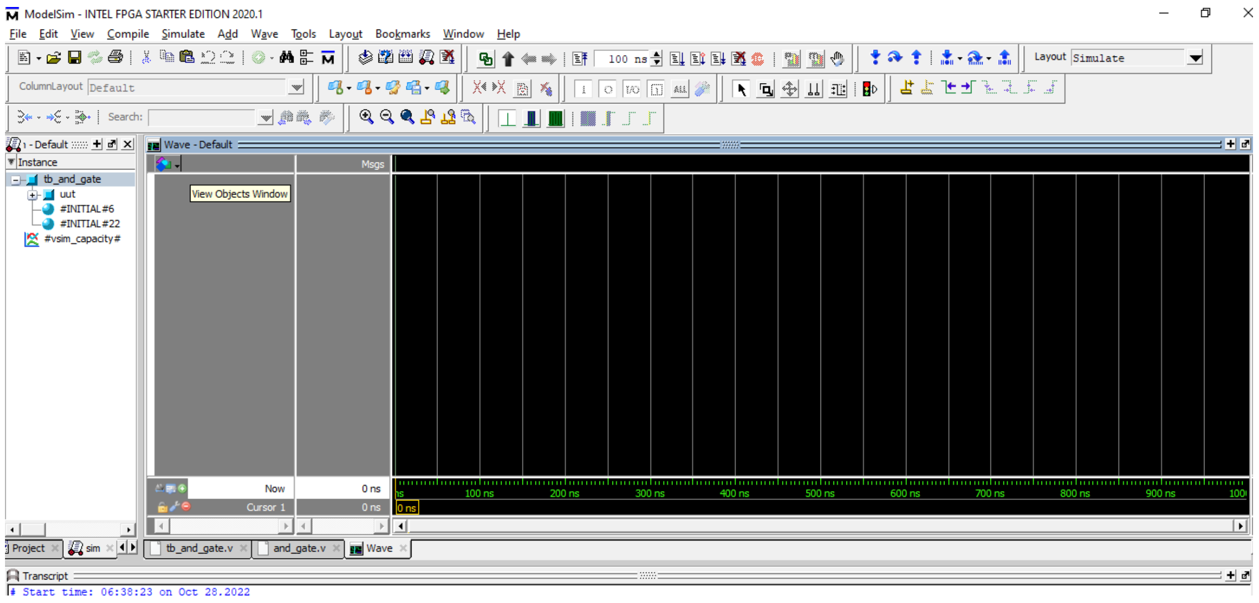


Figure 10: Modelsim wave window

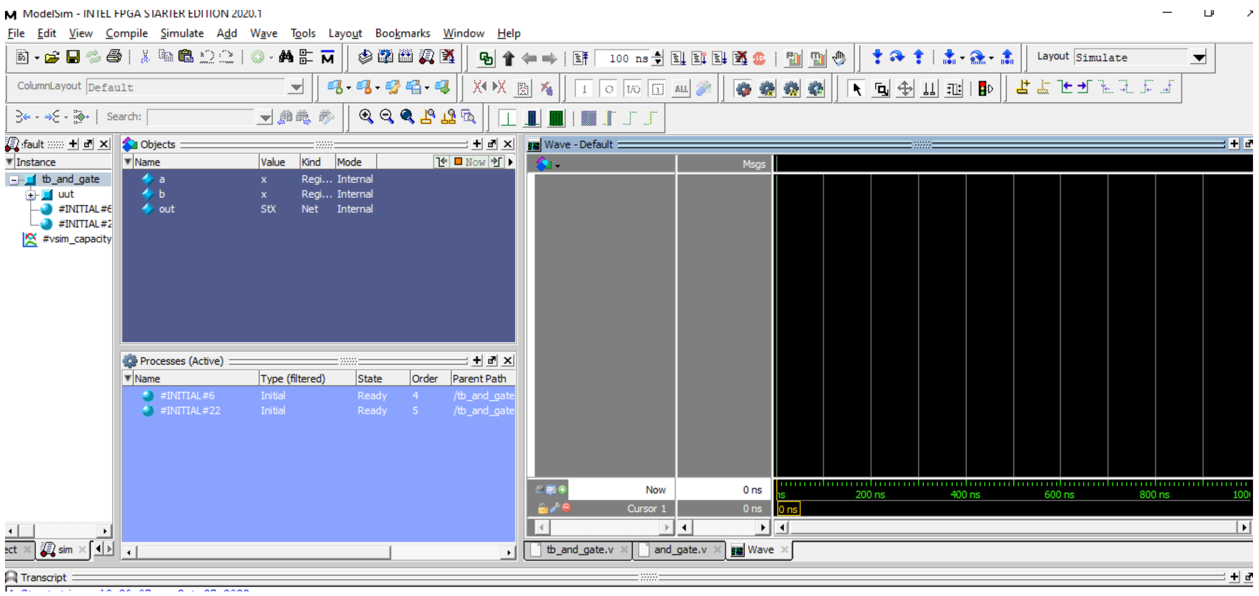
To view the objects, go to view object as shown in the following



```
# Start time: 06:38:23 on Oct 28,2022
```

Figure 11: Modelsim view object

Clicking the view object option will open another window as shown below



Name	Value	Kind	Mode
a	x	Regi...	Internal
b	x	Regi...	Internal
out	StX	Net	Internal

Name	Type (filtered)	State	Order	Parent Path
#INITIAL#6	Initial	Ready	4	/tb_and_gate
#INITIAL#22	Initial	Ready	5	/tb_and_gate

Figure 12: Modelsim: objects and wave window view

Now select the input output objects by pressing control key and drag them in the wave window (grey color area)

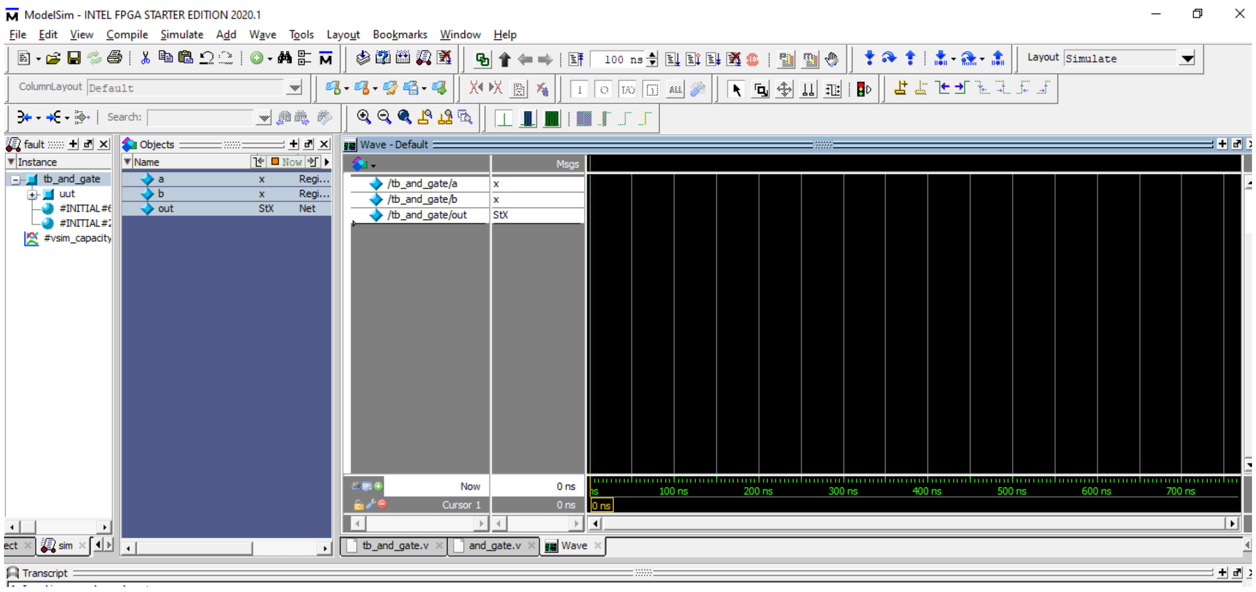
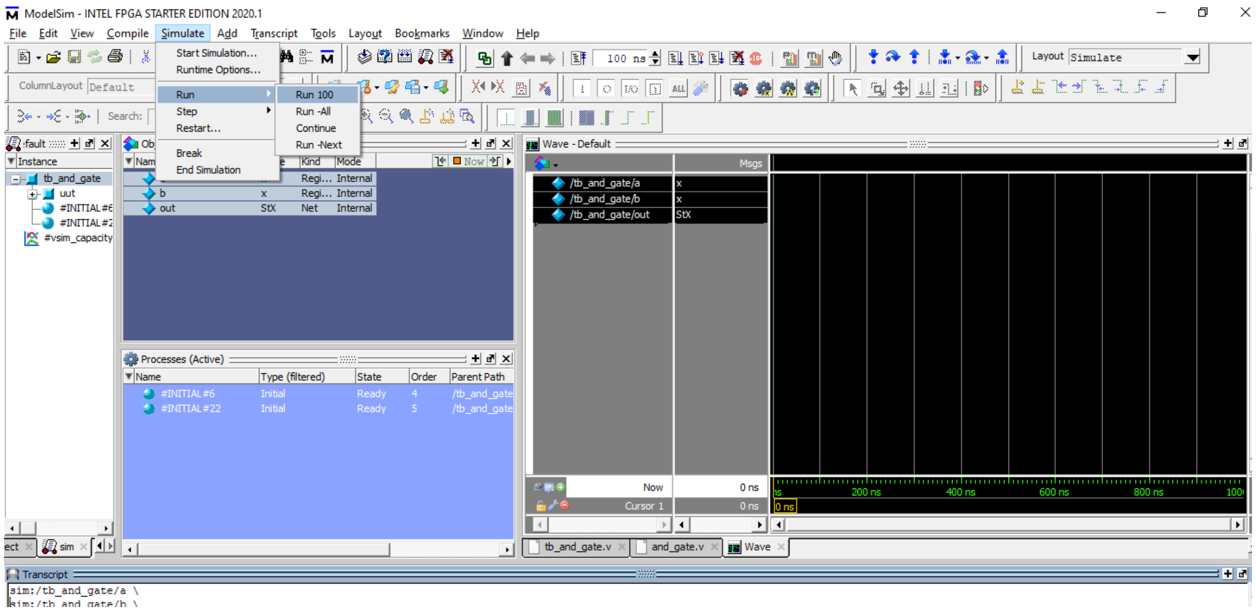

ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Objects
Wave - Default
/tb_and_gate/a
/tb_and_gate/b
/tb_and_gate/out
Now 0 ns
Cursor 1 0 ns
Transcript

Figure 13: Modelsim: After selecting the objects and dragging them in the wave panel

We can now run the simulation. To run it for 100 ns, we select the appropriate run option in the Simulate menu as shown below.



```
sim:/tb_and_gate/a \
sim:/tb_and_gate/b \
```

Figure 14: Modelsim: run simulation for 100 ns option

Clicking in the wave window and pressing ‘f’ will fit the wave in the available screen area. The yellow line highlights the time and the corresponding input and output values as shown in the following figure.

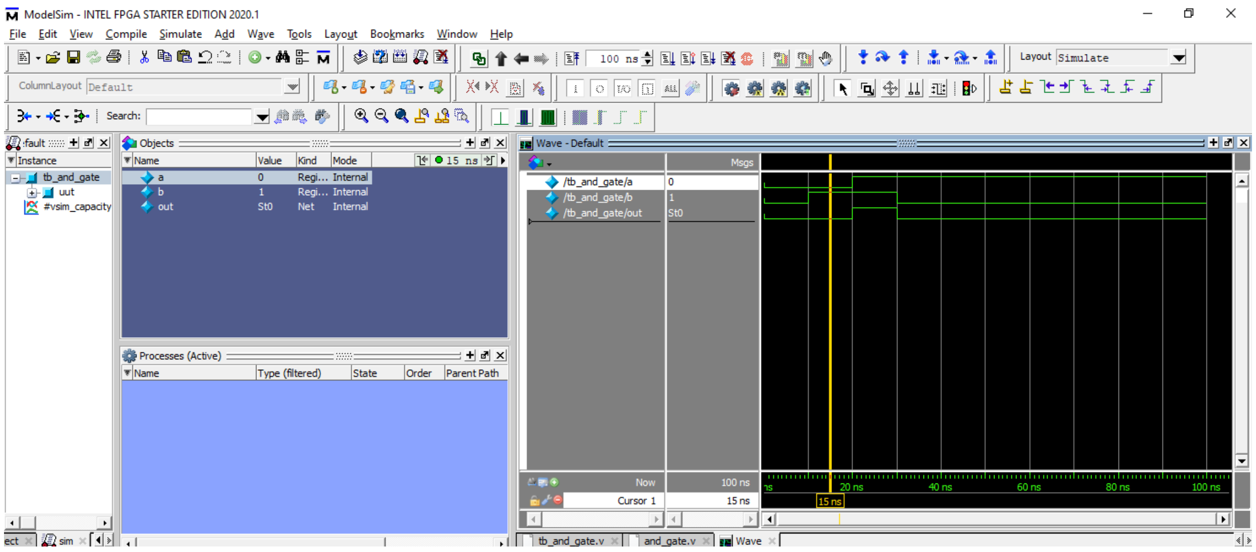


Figure 15: Modelsim: Timing behavior

Task1: Simulating an AND gate

In this exercise at first simulate an AND gate (Figure 16). Consider the code in listing. This code declares a single module, which you can think of as a class. The module's name is *and_gate*, which is descriptive, as we want to simulate an AND gate.

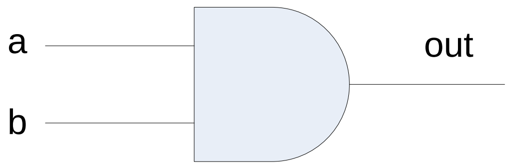


Figure 16

The design file is given below.

```
// name: and_gate.v

module and_gate(out, a,b);  //you list all inputs and outputs, by convention outputs go first
output out;                 // this tells the compile which lines are inputs and outputs
input a, b;

assign out = a & b;         // output function
endmodule
```

Now that we have a module (and_gate), we need to make an instance and simulate it for testing purposes. We only need to simulate and test, so we need to write a test bench. The terminology goes back to the old days of hardwired testing (usually with wire wrap), where you would have a physical testing bench, which typically was set up to rapidly connect and test circuits. The test bench creates virtual environment to verify the correctness design (in this case “and” gate). See Figure 17 for a conceptual representation.

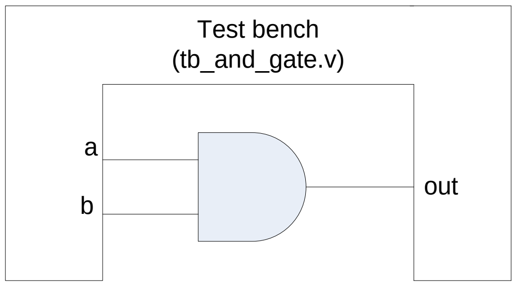


Figure 17

The testbench file is given below.

```
//name: tb_ and_gate.v

module tb_and_gate();   // Test bench for tb_ and_gate.v
reg a,b;                           // a reg, to allow us to assign the input, and a wire to receive the output
wire out;
and_gate uut(out,a,b); // this instantiates a and gate, uut is a label
initial
begin
a = 1'b0;                       // here we apply inputs to the gate
b = 1'b0;
#10;
a = 1'b0;
b = 1'b1;
#10;
a = 1'b1;
b = 1'b1;
#10;
a = 1'b1;
b = 1'b0;
#10;
end
// set up the monitoring
initial
begin
$monitor("a=%b, b=%b, out=%b, time=%t\n", a, b, out, $time);
end
endmodule
```

The $monitor command continuously monitors the values of the variables or signals specified in the parameter list and displays all parameters in the list whenever the values of any one variable or signal changes. $monitor only needs to be invoked once. Only one monitor list can be active at a time.

Step by step intructions:

From modelsim window:

File-> Add to Project->Existing Files: Select file ‘and_gate.v’ and tb_and_gate.v

Select *and_gate.v* in the project tab

Compile

Now compile *tb_and_gate.v*

To simulate:

Simulate -> Start simulation

Click the plus sign (‘+’) near work

Choose tb_and_gate and click OK

In the structural window (labeled as ‘sim’) you can see the structure of the design. The source code for the module that is chosen in the structure window is displayed in the source window.

In the object window: Right click and choose Add to Wave -> signals in Region

To simulate select; Simulation -> Run -> Run -all

In the wave window, you can see the waveform for the selected signals

In this simulation outputs are also visible at main console window.

Study the simulation output and compare them with the code. Examine the output signal of the gate.

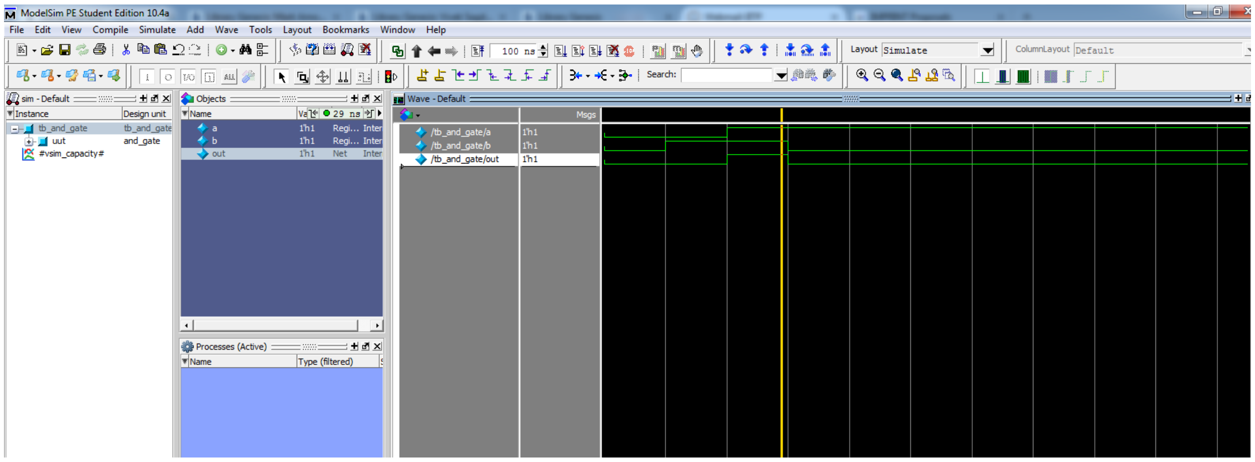


Fig4: Simulation output

Task2: Repeat the above steps by creating or_gate.v and tb_or_gate.v

```
// name: or_gate
 module or_gate(out,a,b);  //you list all inputs and outputs, by convention outputs go first
output out;                // this tells the compile which lines are inputs and outputs
input a, b;

assign out = a | b;        // output function
endmodule

//name: tb_or_gate
module tb_or_gate();   // Test bench for and_gate.v
reg a,b;               // a reg, to allow us to assign the input, and a wire to receive the output
wire out;
or_gate uut (out,a,b); // this instantiates a and gate, uut is a label
initial
begin
a = 1'b0;              // here we apply inputs to the gate
b = 1'b0;
#10;
a = 1'b0;
b = 1'b1;
#10;
a = 1'b1;
b = 1'b1;
#10;
a = 1'b1;
b = 1'b0;
#10;
end
// set up the monitoring
initial
```

```
begin
$monitor("a=%b, b=%b, out=%b, time=%t\n", a, b, out, $time);
end
endmodule.
```

Exercise: Simulate 3input / input AND gate and OR gate. Write appropriate testbench.

Task 3: Types Modeling in Verilog

In this part we will model a logic function in different ways. Consider the following table

sel	In2	In1	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Out put can be simplied as

Out=in1. sel +in2. sel' and the implementation is shown in figure 4

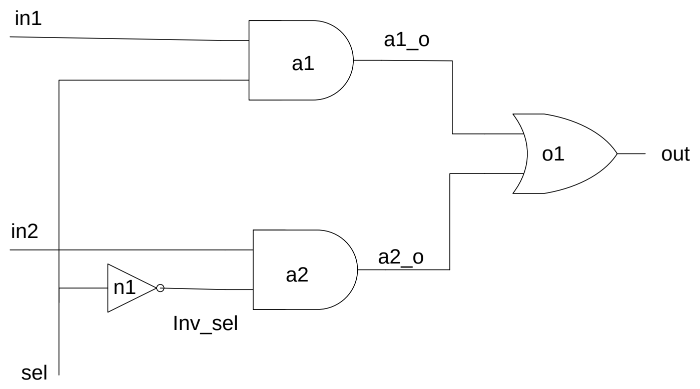


Figure 4

There are various ways of modeling in Verilog.

(1) Structural
(2) Behavioural level

Let us call this function as mux and now we model the function in the above 3 methods.

(1) Structural

Structural Style: *The circuit is specified in terms of lower level components (in this example logic gates, which are Verilog primitives) connected with internal signals. The translation of such a specification into a physical circuit is straight forward.*

```
// lines start with "//" is a comment
// name: mux_gate.v   // gate level model

module mux_struct(out, in1,in2,sel );  //you list all inputs and outputs, by convention outputs go first
output out;                            // this tells the compile which lines are inputs and outputs
input in1,in2,sel;

and a1 (a1_o,in1,sel);          // defines the a1 gate, see figure
not n1 (inv_sel,sel);          // defines the inverter gate, see figure
and a2 (a2_o,in2,inv_sel);          // defines the a2 gate, see figure
or  o1 (out, a1_o, a2_o);          // defines the 01 gate, see figure

endmodule
```

(2) Behavioural level

1. **Behavioural Style:** *It specifies the circuit in terms of its expected behaviour. It is the closest to a natural language description of the circuit functionality.*

```
// lines start with "//" is a comment
// name: mux_behav.v   // Behavior: event-driven behavior description construct

module mux_behav(out, in1,in2,sel );  //you list all inputs and outputs, by convention outputs go first
output out;                           // this tells the compile which lines are inputs and outputs
input in1,in2,sel;
reg out;
always@(in1 or in2 or sel)
begin
        if (sel)
        out =in1;
       else
        out=in2;
        end

endmodule
```

2. **Behavioural Style:**

// lines start with “//” is a comment

***module** mux_ behav2(out, in1,in2,sel);* //you list all inputs and outputs, by convention outputs go first
***output** out;*   *// this tells the compile which lines are inputs and outputs*
***input** in1,in2,sel;*

***assign** out=sel ? in1: in2;*
endmodule

```
module tb_mux_behav();

reg in1,in2,sel;

wire out;

mux_behav UUT (out,in1,in2,sel);

initial

begin

in1 = 1'b0;                // here we apply inputs to the logic

in2 = 1'b0;

sel = 1'b0;

#10;

in1 = 1'b0;

in2 = 1'b1;

sel = 1'b0;

#10;

in1 = 1'b1;

in2 = 1'b0;

sel = 1'b0;

#10;   end

// set up the monitoring

initial  begin

$monitor("sel=%b, in1=%b, in2=%b, out=%b, time=%t\n", sel, in1,in2, out, $time); end

 endmodule
```