LAB2

we aim to design and implement the combinational circuit for the following equation: `y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;`

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

```
module example(
    input logic a, b, c,
    output logic y
);

    assign y = (~a & ~b & ~c) | (a & ~b & ~c) | (a & ~b & c);

endmodule
```

✅ **Implementation Steps:**

1. Create a **truth table** to examine all possible inputs and outputs.
2. Implement this equation in **Verilog**, defining the inputs and outputs.
3. Write a **testbench** to simulate and verify the circuit's functionality.
4. Observe the **simulation results** to ensure the circuit operates correctly.

a	b	c	~a	~b	~c	~a & ~b & ~c	a & ~b & ~c	a & ~b & c	y
0	0	0	1	1	1	1	0	0	1
0	0	1	1	1	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0	0	0
1	0	0	0	1	1	0	1	0	1
1	0	1	0	1	0	0	0	1	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0

TESTBENCH

Testbench for Combinational Circuit Simulation

This Verilog **testbench** is designed to verify the functionality of the combinational circuit implemented in **example.sv**. It applies a set of predefined input waveforms and observes the corresponding output.

Key Components:

1️⃣ **Instantiation of DUT (Device Under Test):**

- The **example** module is instantiated and connected to test signals.
- Inputs: `a`, `b`, `c`
- Output: `y`

2️⃣ **Waveform Generation:**

- The `initial` block defines the input test cases, applying different values to `a`, `b`, and `c` at specific time steps to match the required simulation waveform.
- The testbench covers all relevant input transitions to validate circuit behavior.

3️⃣ **Output Monitoring:**

- The `$monitor` statement continuously prints the values of `a`, `b`, `c`, and `y` with timestamps, helping to track changes in real-time.

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

```
`timescale 1ns / 1ps
module tb ();
    // Signal declarations
    logic  a;    // Input a
    logic  b;    // Input b
    logic  c;    // Input c
    logic  y;    // Output y

    // Instantiate the Device Under Test (DUT)
    example dut (
        .a(a),
        .b(b),
        .c(c),
        .y(y)
    );

    // Generate waveform to match the simulation output
    initial
      begin
        // Initialize inputs
        a = 0; b = 0; c = 0;

        // Generate a waveform that matches the given output
        #10 c = 1;
        #20 b = 1; c = 0;
        #10 c = 1;
        #20 a = 1; b = 0; c = 0;
        #10 c = 1;
        #20 b = 1; c = 0;
        #10 c = 1;
        // End simulation
        #20;
        $finish;
      end
    // Monitor outputs
    initial
      begin
        $monitor("Time=%0t | a=%b | b=%b | c=%b | y=%b", $time, a, b, c, y)
      end
endmodule
```

```
if [file exists work] {
    vdel -all
}
vlib work

# compile source files
vlog example.sv example_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
```

- ✅ **Compiles** the Verilog files (`example.sv` and `example_tb.sv`).
- ✅ **Loads** the testbench for simulation.
- ✅ **Displays** all relevant signals in waveform format.
- ✅ **Runs** the simulation for **250 ns** to observe circuit behavior.

This script ensures a **fully automated process** for running and analyzing a **Verilog-based simulation** in **ModelSim/QuestaSim**. 🚀

VSIM -DO NAME.DO _C

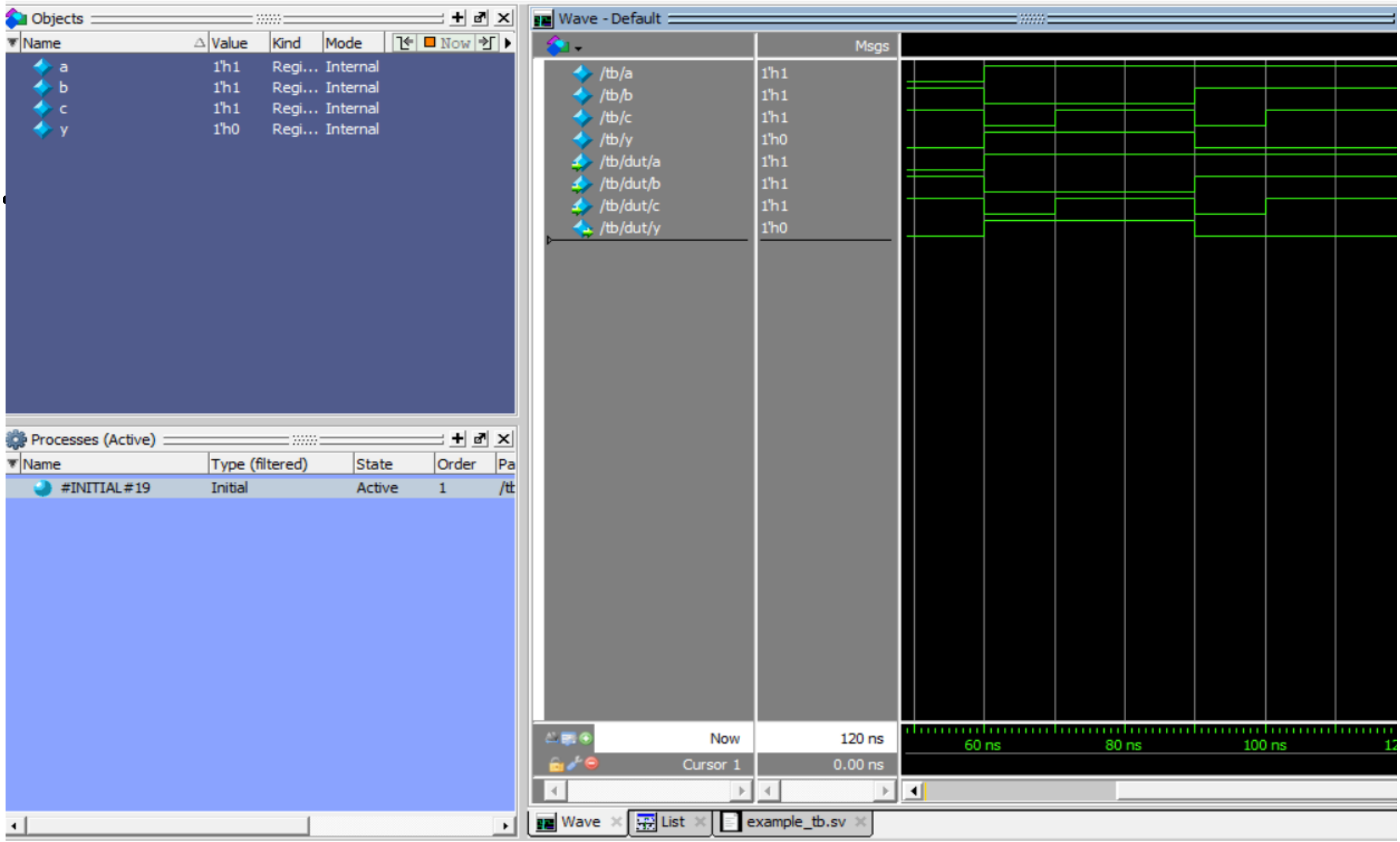
- Typically, you will also run SV in two modes
  - Text-based: `vsim -do file.do -c`
  - Graphical: `vsim -do file.do`
- Graphical should be used to figure out specific problems and may involve examining signals very closely.
- This is not easy and takes a load of practice, but I personally do not think it is that difficult.
- The key is to use your brain on figuring out what things "should" be.

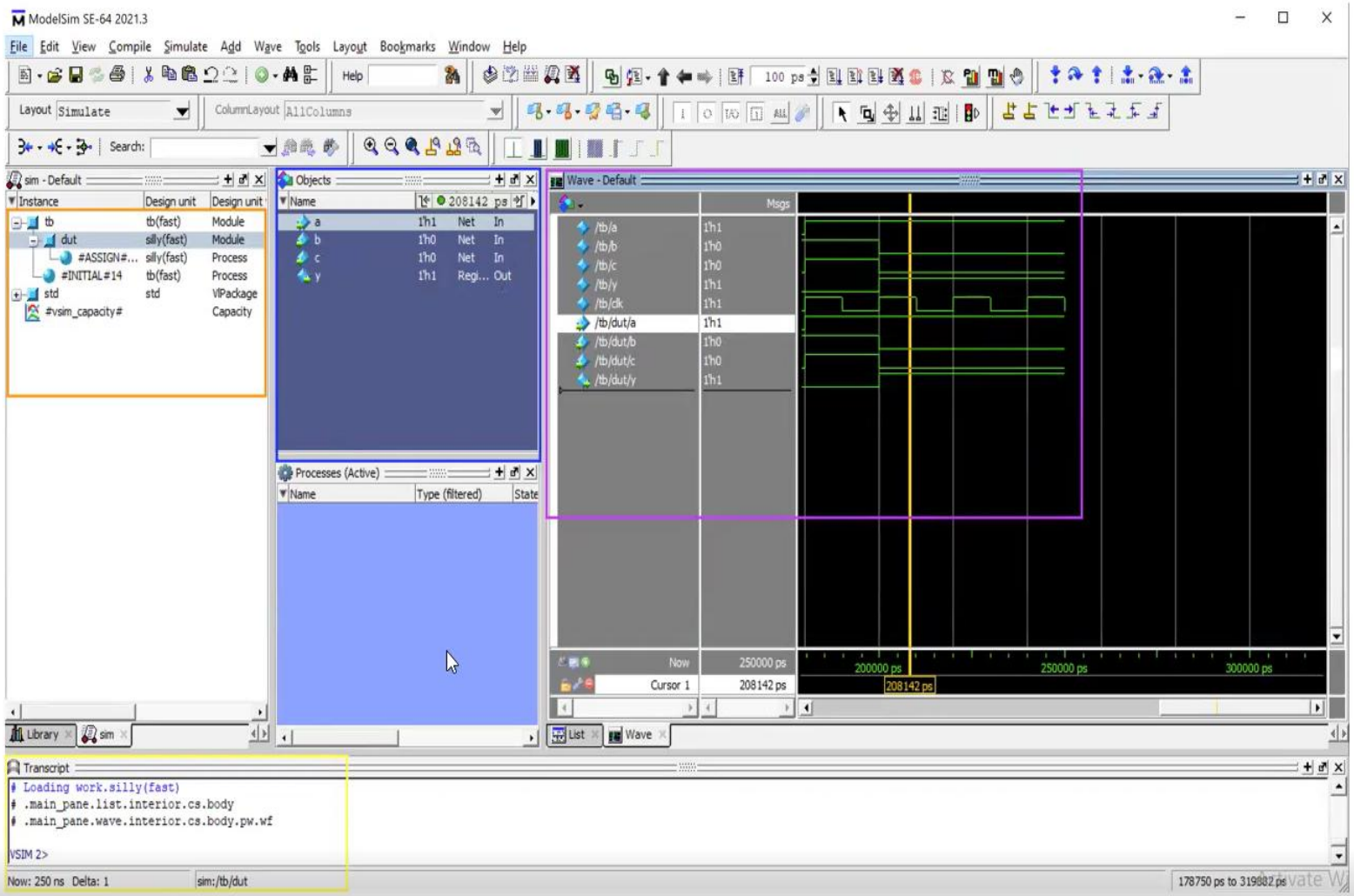
```
Administrator: Windows Powe
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Maryam\Desktop\Lab> vsim -do example.do -c
```

```
PS C:\Users\Maryam\Desktop\Lab> vsim -do example.do
```


Objects
Name
Value
Kind
Mode
Now
a 1'h1 Regi... Internal
b 1'h1 Regi... Internal
c 1'h1 Regi... Internal
y 1'h0 Regi... Internal
Processes (Active)
Name
Type (filtered)
State
Order
Pa
#INITIAL#19 Initial Active 1 /tt
Wave - Default
Msgs
/tb/a 1'h1
/tb/b 1'h1
/tb/c 1'h1
/tb/y 1'h0
/tb/dut/a 1'h1
/tb/dut/b 1'h1
/tb/dut/c 1'h1
/tb/dut/y 1'h0
Now 120 ns
Cursor 1 0.00 ns
60 ns
80 ns
100 ns
Wave
List
example_tb.sv



1. Wave window finds where you want to examine something.
2. The sim window examines your hierarchy from the DUT by instance names.
3. The Objects shows what the signals are at that that time instance.
4. The transcript window indicates messages and where you can interact with commands

- Not all items may be visible unless those signals are indicated *within* the DO file!.
- There are many cool interfaces within this environment that I am not 100% sure of – but I know they leverage much of software and can be made quite powerful!