SystemVerilog Testbench

DSD F2019

What is it?

- A test bench is an environment (also some SystemVerilog codes) used to verify the correctness or soundness of a design or model.

- Another way to do the simulation.

- In a word, write some SystemVerilog codes to do the simulation.

Why is it?

- Can run and verify the results automatically.

- For complicated designs, it will take too long to create the input waveforms to test all the cases.

- It is required for Lab 4 Simulation.

How to write it?

- The module to be tested is called Device Under Test (DUT). It may also be called as Unit Under Test (UUT).

- Not Synthesizable

- Three Types
  - Simple
  - Self-checking
  - Self-checking with testvectors

Testbench Example

Write SystemVerilog code to implement the following function in hardware:

$$y = \overline{b}\,\overline{c} + a\overline{b}$$

```
module sillyfunction(input  logic a, b, c,
                     output logic y);
  assign y = ~b & ~c | a & ~b;
endmodule
```

Simple Testbench

```
module testbench1();
  logic a, b, c;
  logic y;
  // instantiate device under test
  sillyfunction dut(a, b, c, y);
  // apply inputs one at a time
  initial begin
    a = 0; b = 0; c = 0; #10;
    c = 1; #10;
    b = 1; c = 0; #10;
    c = 1; #10;
    a = 1; b = 0; c = 0; #10;
    c = 1; #10;
    b = 1; c = 0; #10;
    c = 1; #10;
  end
endmodule
```

Self-checking Testbench

```
module testbench2();
  logic  a, b, c;
  logic y;
  sillyfunction dut(a, b, c, y); //
instantiate dut
  initial begin // apply inputs, check
results one at a time
      a = 0; b = 0; c = 0; #10;
      if (y !== 1) $display("000 failed.");
      c = 1; #10;
      if (y !== 0) $display("001 failed.");
      b = 1; c = 0; #10;
      if (y !== 0) $display("010 failed.");
      c = 1; #10;
      if (y !== 0) $display("011 failed.");
      a = 1; b = 0; c = 0; #10;
      if (y !== 1) $display("100 failed.");
      c = 1; #10;
      if (y !== 1) $display("101 failed.");
      b = 1; c = 0; #10;
      if (y !== 0) $display("110 failed.");
      c = 1; #10;
      if (y !== 0) $display("111 failed.");
  end
endmodule
```

$display(“...”); will display msg inside bouble quotes

Self-checking testbench with testvectors

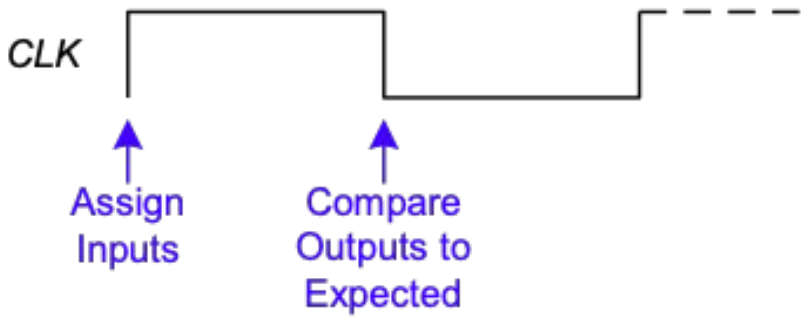
- Testvector file
  - inputs and expected outputs

- Testbench
  - Generate clock for assigning inputs, reading outputs
  - Read testvectors file into array
  - Assign inputs, expected outputs
  - Compare outputs with expected outputs and report errors

Self-checking testbench with testvectors

Testbench Clock

- Assign inputs @ posedge of clk
- Compare outputs with expected outputs @ negedge of clk



Testvector example

- Create a file called example.txt
- Contains vectors line by line, in the format of abc_yexpected

```
000_1
001_0
010_0
011_0
100_1
101_1
110_0
111_0
```

1. Generate Clock

```
module testbench3();
  logic       clk, reset;
  logic       a, b, c, yexpected;
  logic       y;
  logic [31:0] vectornum, errors;   // bookkeeping variables
  logic [3:0]  testvectors[10000:0]; // array of testvectors

  // instantiate device under test
  sillyfunction dut(a, b, c, y);
  // generate clock
  always     // no sensitivity list, so it always executes
 begin
     clk = 1; #5; clk = 0; #5;
 end
```

2. Read testvectors into an array

```
// at start of test, load vectors and pulse reset

initial
begin
    $readmemb("example.txt", testvectors);
    vectornum = 0; errors = 0;
    reset = 1; #27; reset = 0;
end
// Note: $readmemh reads testvector files written in
// hexadecimal
```

3. Assign inputs & expected outputs

```
 // apply test vectors @ posedge of clk
 always @(posedge clk)
begin
   #1; {a, b, c, yexpected} = testvectors[vectornum];
end
```

4. Compare with Expected Outputs

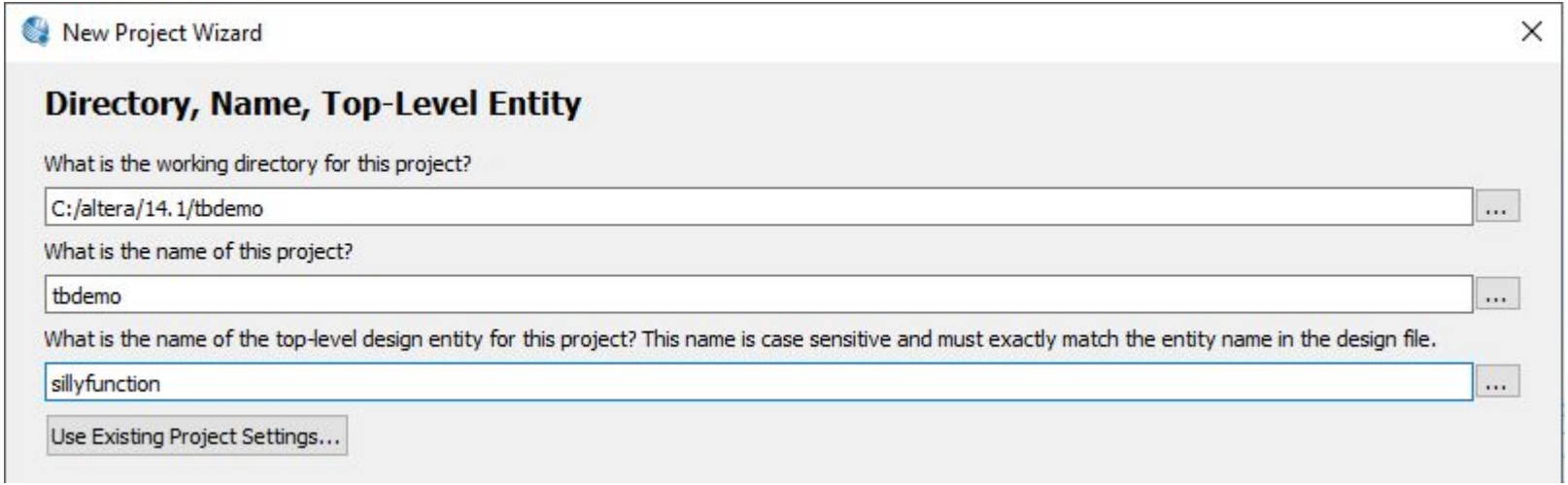
```
// check results on falling edge of clk
   always @(negedge clk)
 if (~reset) begin // skip during reset
     if (y !== yexpected) begin
     $display("Error: inputs = %b", {a, b, c});
     $display("  outputs = %b (%b expected)",y,yexpected);
     errors = errors + 1;
     end
// Note: to print in hexadecimal, use %h. For example,
//       $display("Error: inputs = %h", {a, b, c});
```

4. Compare with Expected Outputs

```
// increment array index and read next testvector
    vectornum = vectornum + 1;
    if (testvectors[vectornum] === 4'bx) begin
        $display("%d tests completed with %d errors",
                vectornum, errors);
    $finish;
    end
 end
endmodule
// === and !== can compare values that are 1, 0, x, or z.
```

Testbench in Quartus II

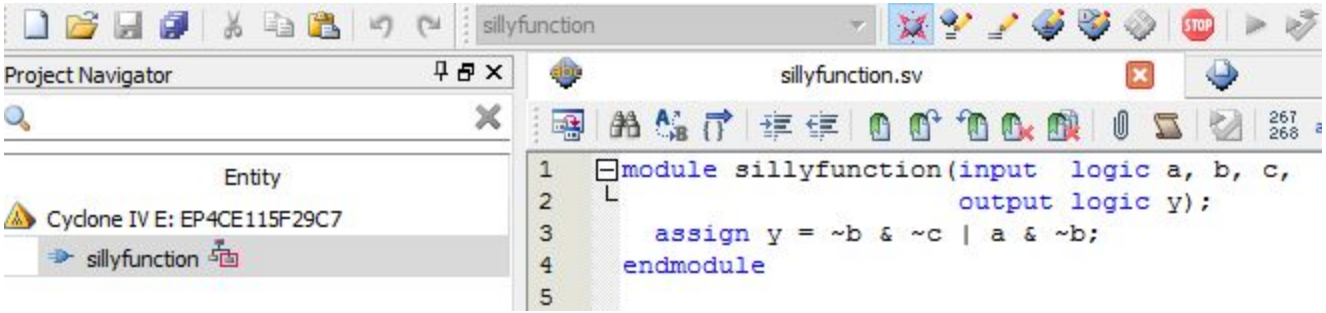
First, we create a project named “*tbdemo*” and its top-level design entity is called “*sillyfunction*” using New Project Wizard.



Testbench in Quartus II

Create *sillyfunction.sv* and copy & paste the codes.

Then compile it to make sure you do this correctly.



Testbench in Quartus II

It is a good practice to include the testbench module in the DUT.

So we copy & paste the codes *testbench2* module under *sillyfunction*, and rename it to *sillyfunction_TB*.

Make everything compiled successfully.

sillyfunction.sv | Compilation Report - sillyfunction

```
1  module sillyfunction(input  logic a, b, c,
2                       output logic y);
3    assign y = ~b & ~c | a & ~b;
4  endmodule
5
6  module sillyfunction_TB();
7    logic  a, b, c;
8    logic y;
9    sillyfunction dut(a, b, c, y);  // instantiate dut
10   initial begin // apply inputs, check results one at a time
11    a = 0; b = 0; c = 0; #10;
12    if (y !== 1) $display("000 failed.");
13    c = 1; #10;
14    if (y !== 0) $display("001 failed.");
15    b = 1; c = 0; #10;
16    if (y !== 0) $display("010 failed.");
17    c = 1; #10;
18    if (y !== 0) $display("011 failed.");
19    a = 1; b = 0; c = 0; #10;
20    if (y !== 1) $display("100 failed.");
21    c = 1; #10;
22    if (y !== 1) $display("101 failed.");
23    b = 1; c = 0; #10;
24    if (y !== 0) $display("110 failed.");
25    c = 1; #10;
26    if (y !== 0) $display("111 failed.");
27   end
28 endmodule
29
```

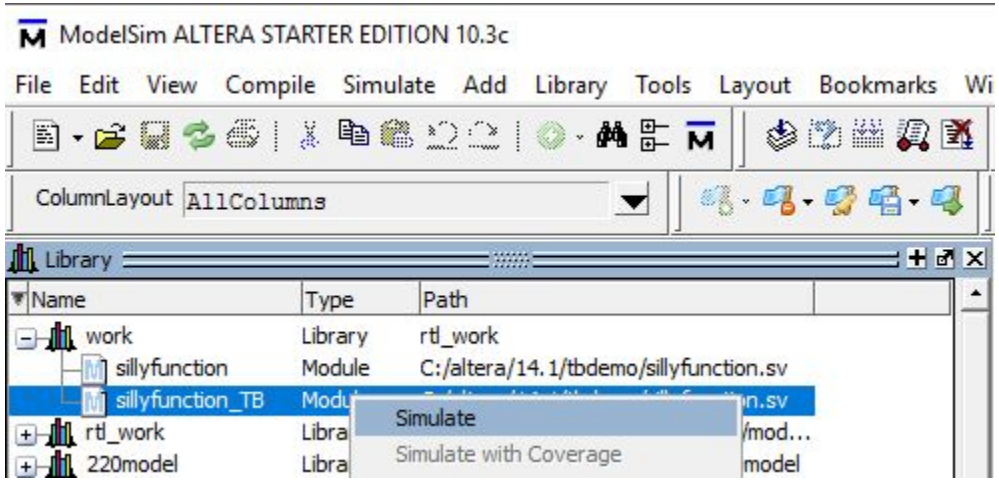
Testbench in Quartus II

Testbench module is not Synthesizable. To simulate DUT, we have to use Altera-ModelSim. To open it, choose Tools -> Run Simulation Tool -> RTL Simulation.

Make sure you have everything compiled successfully before you open it.

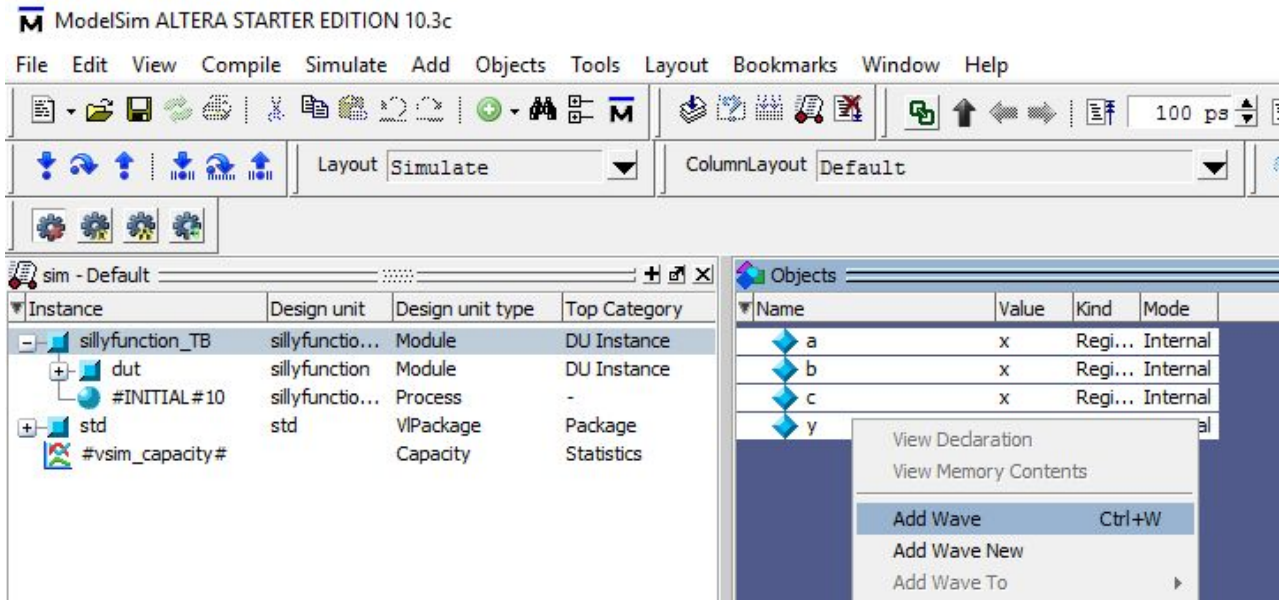
Testbench in Quartus II

Expand *work*. Right click *sillyfunction_TB*, then click *Simulate*.



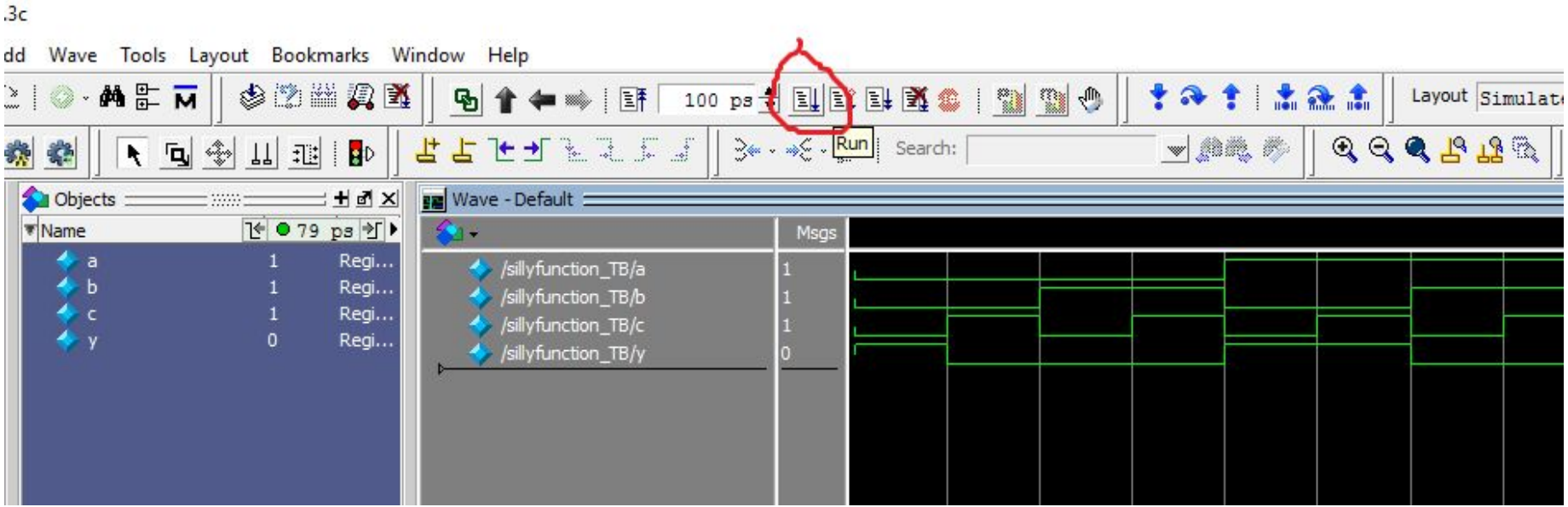
Testbench in Quartus II

Choose all the *Objects* you want to simulate, and then right click them, choose *Add Wave*.



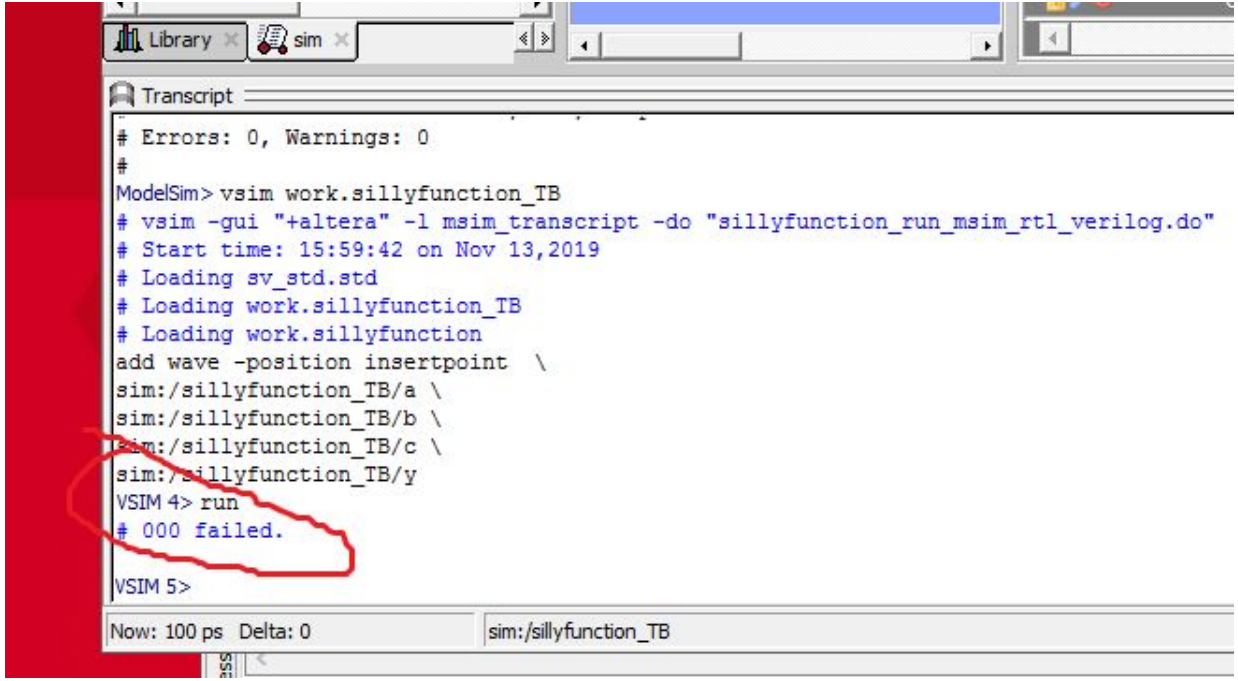
Testbench in Quartus II

Click *Run* button to start simulation. *Zoom in* to see your waves. *100ps* means the simulator will run *100ps* everytime you click the *Run* button.



Testbench in Quartus II

Messages sent by *$display* will be shown in the *Transcript* window.

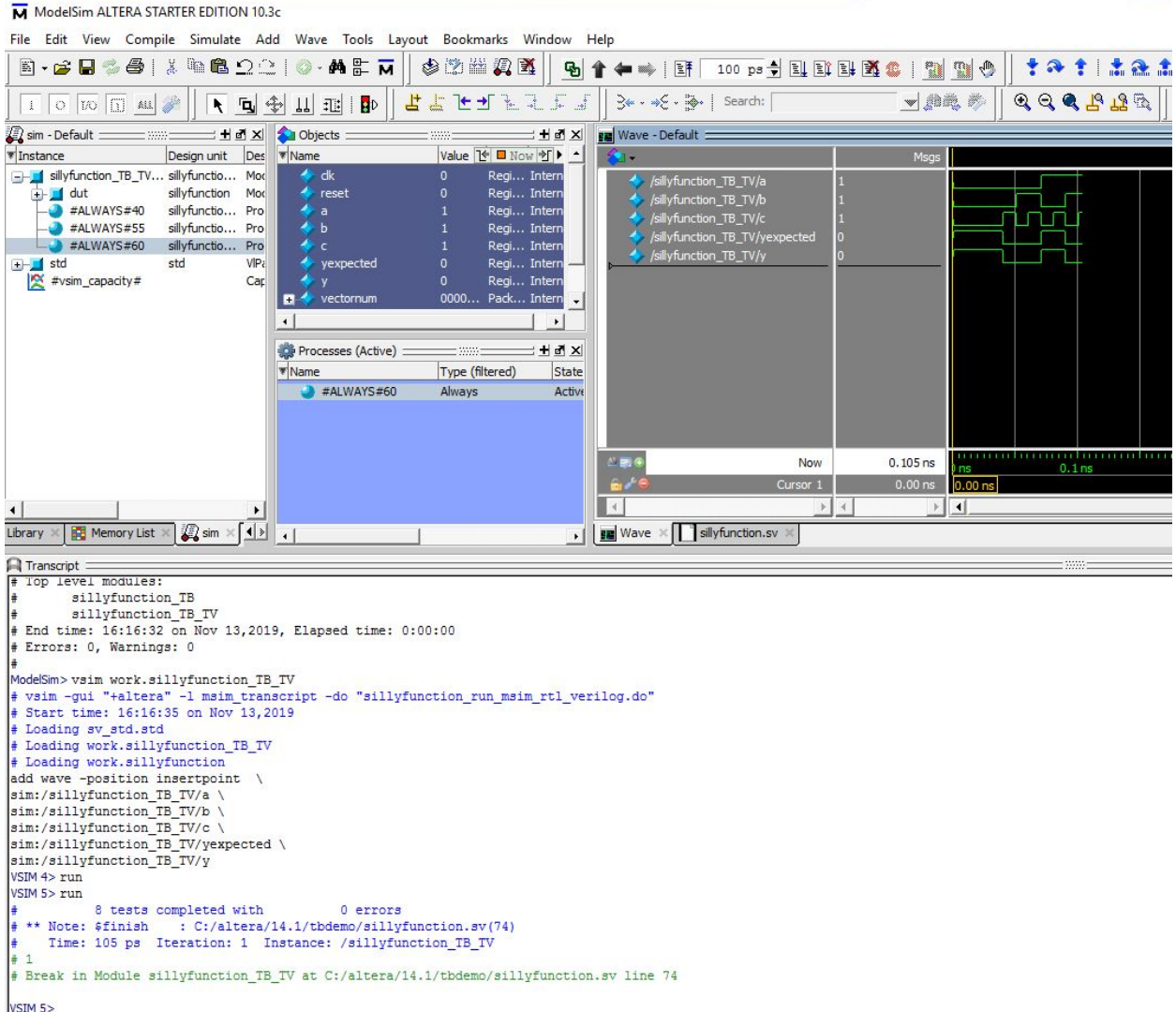


Testbench in Quartus II

For the testbench with testvectors, you have to create your textvector file called *example.txt* and then copy & paste the file to $your_project_directory$\simulation\modelsim.

In this project, it is C:\altera\14.1\tbdemo\simulation\modelsim

A simulation example of testbench with testvectors.



Simulation of Lab 4

- **MUST** use testbench to do the simulation, test all the cases.

- Choose the type of testbench you want, but if you use a more difficult version of testbench (difficulty: Simple < Self-checking < Self-checking with testvectors), you will get higher grades of your simulation report.

References

- Textbook, Chapter 4.9 (Also check the corresponding slides)

- https://class.ece.uw.edu/271/peckol/doc/DE1-SoC-Board-Tutorials/ModelsimTutorials/QuartusII-Testbench-Tutorial.pdf