**Tutorial on VHDL**

Table of Contents

[Setup 2](#_Toc63382425)

[Source Code 3](#_Toc63382426)

[Testbench 5](#_Toc63382427)

[Running the Code 8](#_Toc63382428)

## Setup

In this tutorial, we will be using ModelSim by Intel FPGA (Starter Edition). Version 16.1 of the software for Windows 10 can be downloaded from [here](https://download.altera.com/akdlm/software/acdsinst/16.1/196/ib_installers/ModelSimSetup-16.1.0.196-windows.exe).

Once installed, run ModelSim and create a new project from the File tab. The project name should be related to the work, for example mux\_4\_to\_1 for a project that is meant to hold the code for a 4-to-1 Multiplexer.

Next, you should be prompted to create a new file. The first file we create is meant to hold the source code for our entity. It should have the same name as the project. Ensure that the file type is VHDL.

## Source Code

There should now be a .vhd file visible. Double click this to open the code editor. In the code editor, place the source code for the entity. The entity name should, again, be the same as the project name and the file name.

For example, consider the code for a 4-to-1 multiplexer:

library ieee;  
use ieee.std\_logic\_1164.all;  
  
entity mux4\_to\_1 is  
 port (  
 I0 : in std\_logic;  
 I1 : in std\_logic;  
 I2 : in std\_logic;  
 I3 : in std\_logic;  
 sel : in std\_logic\_vector (1 downto 0);  
 output : out std\_logic  
 );  
end entity;  
  
architecture behavioral of mux4\_to\_1 is  
begin  
 process(sel) -- process and sensitivity list  
 begin  
 case sel is  
 when "00" => output <= I0;  
 when "01" => output <= I1;  
 when "10" => output <= I2;  
 when others => output <= I3;  
 end case;  
 end process;  
end architecture behavioral;

VHDL

In the entity section, we have all of the inputs and outputs for the multiplexer. sel is particularly interesting, since it is a 2-bit array with indices from 1 to 0. Also, the output, the last member in the list, does not have a semicolon after it, as do the other elements in the list.

The part with case simply states which input should be given as output for different values of sel.

Once the code is written, save it. Next, we need to compile it. Select Compile All from the Compile tab.

## Testbench

Once we are done with our source code, we need to write the testbench code. This will reside in another file within the same project. To add the file, right-click on the project pane and click on Add to Project and then New File. Generally, testbench codes have the same name as the source code file with a tb prefix. Thus, for the example above, the filename would be tb\_mux\_4\_to\_1.

For the previous example, the testbench code looks like this:

library ieee;  
use ieee.std\_logic\_1164.all;  
  
entity tb\_mux4\_to\_1 is  
end tb\_mux4\_to\_1;  
  
architecture behavioral of tb\_mux4\_to\_1 is  
  
component mux4\_to\_1  
 port (  
 I0 : in std\_logic;  
 I1 : in std\_logic;  
 I2 : in std\_logic;  
 I3 : in std\_logic;  
 sel : in std\_logic\_vector (1 downto 0);  
 output : out std\_logic  
 );  
end component;  
  
signal tb\_I0, tb\_I1, tb\_I2, tb\_I3 : std\_logic := '0';  
signal tb\_sel : std\_logic\_vector (1 downto 0) := (others => '0');  
signal tb\_output : std\_logic;

begin  
 uut: mux4\_to\_1 port map (  
 I0 => tb\_I0,  
 I1 => tb\_I1,  
 I2 => tb\_I2,  
 I3 => tb\_I3,  
 sel => tb\_sel,  
 output => tb\_output  
 );  
   
 -- stimulus process  
 stim\_process: process  
 begin  
 wait for 50 ns;  
 tb\_sel <= "00";  
 tb\_I0 <= '1'; tb\_I1 <= '0'; tb\_I2 <= '1'; tb\_I3 <= '1';  
 wait for 50 ns;  
 tb\_sel <= "01";  
 wait for 50 ns;  
 tb\_sel <= "10";  
 wait for 50 ns;  
 tb\_sel <= "11";  
 wait for 50 ns;  
 tb\_I0 <= '0'; tb\_I1 <= '1'; tb\_I2 <= '0'; tb\_I3 <= '0';  
 tb\_sel <= "11";  
 wait for 50 ns;  
 end process;  
  
end architecture behavioral;

VHDL

Notice that the entity section is left empty. Instead, in the architecture section, the component from our source code is declared with all its ports. If we have multiple entities in the project, we will also have multiple components in our testbench.

Next, we have the signals, which are like the connecting wires.

Finally, we have the unit under test, uut, which maps each of the ports from the component onto a signal.

Lastly, the actual body of the architecture section begins. In the stimulus process, we test our component by giving different inputs to tb\_sel­. We wait a while between different inputs to ensure there are no problems due to jitters.

Of course, once we are done with this, we must save the file and compile it as before.

## Running the Code

To actually simulate the code we have written, go to the Simulate tab and choose Start Simulation. You should be greeted with a new window with lots of files. Under the Design tab, go to work and then select the testbench file. Ensure that the resolution is set to nano-seconds. Click on OK to get the simulation running.

In the new view that appears, in the left-most pane, the sim pane that is, right-click and select Add to Wave All items in region. At the top of the new view, you should also see a box for Run Length. Ensure this is a large enough value that you can see all the changes you want to. For the example code above, it requires at least , due to the wait times. We should probably use a little more, say . Finally, click the Run button next to the Run Length text field to see the output.