TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

Faculty of Automation and Computer Science
Department of Computer Science

COMPUTER ARCHITECTURE

Introduction to Creating a VHDL Test Bench in Xilinx Vivado

COMPUTER ARCHITECTURE

Introduction to Creating a VHDL Test Bench in Xilinx Vivado

February 2021

1 Introduction

Simulation: the imitative representation of the functioning of one system or process by means of the functioning of another *(as defined by Merriam-Webster)*

When it comes to electronic hardware and *Hardware Description Language* (HDL) models, simulators and *Electronic Design Automation* (EDA) simulation tools are used to verify as to whether the *Register-Transfer Level* (RTL) code meets the functional requirements of a given design's specifications.

To achieve this, the necessary signals must be generated alongside the different states and logical input values resulting in a waveform as an output.

It's possible that you may not be able to properly copy-paste the code from the document, therefore a separate GitHub Gist has been created to circumvent any issues.
The link to the UTCN Computer Architecture Test Bench supplementary code is https://gist.github.com/fuzesa/bd6f463f4a8ce7687975eb3e43da064f

2 Basic Terminology

NOTE: When it comes to Verilog, test benches are sometimes referred to as *test fixtures*. Please be aware that two terms are synonymous in concept.

DUT - Device Under Test / UUT - Unit Under Test: The given model or entity undergoing testing.

Input Stimulus: The input signal being fed into the DUT / UUT.

3 Functional vs timing simulation

Functional simulation refers to the concept of testing the underlying operational behavior of the circuit without any consideration for delays associated with placement or routing.

In reality, these aforementioned delays could result in signals not meeting setup time and undesirable output.

Timing simulation on the other hand takes these factors into account based on the speed grade of the component.
Within Vivado, these delays are not configured through any timing analysis tool and they must be incorporated into the device through its design.

4 Common guidelines

– Test benches are basically a part of your project's hierarchy and it is recommended to have them at the top level.

– Within Xilinx, the instantiated components in test benches are often labeled as UUT (*Unit Under Test*).

– In most cases, the same test bench should be applicable for both functional and timing simulations.

5 Simulation of a combinational logic circuit

First, we are going to implement a test bench for basic circuit with simple logic gates.

The device has three inputs, two binary for different values and a vector select control of a *multiplexer* (MUX). The output of the device will be the output from the MUX

The inputs to the MUX are going to be the result of
00 → an *AND* gate
01 → an *OR* gate
10 → and lastly a *XOR* gate

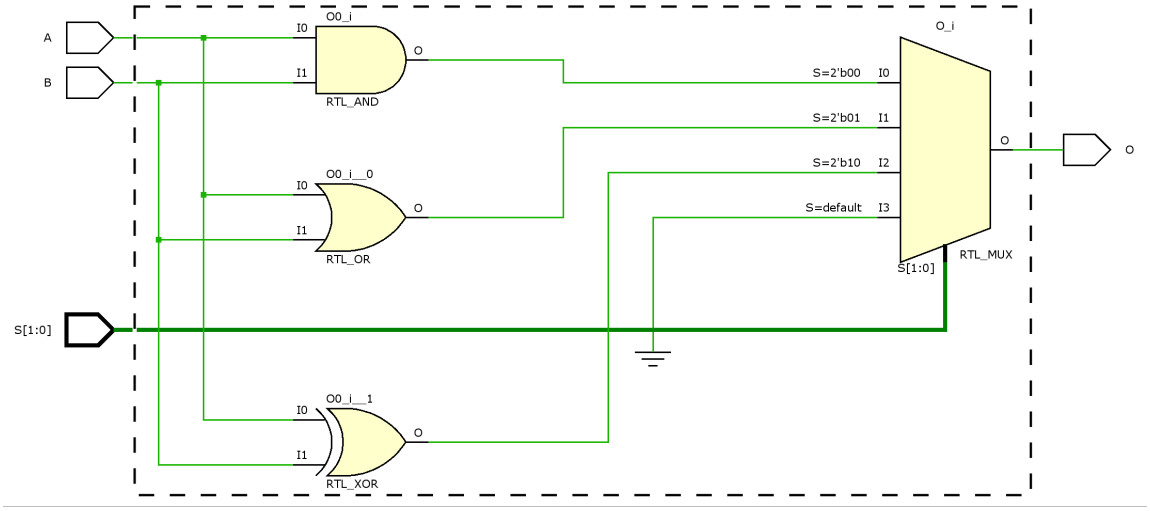

Figure 1: Sample combinational circuit

The VHDL code for the schematic above is the following:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sample_comb_circ is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           S : in STD_LOGIC_VECTOR (1 downto 0);
           O : out STD_LOGIC);
end sample_comb_circ;

architecture Behavioral of sample_comb_circ is

begin

with S select
  O <= A and B when "00",
       A or B  when "01",
       A xor B when "10",
       '0' when others;
```

```
end Behavioral;
```

In the example above, we have named our entity as `sample_comb_circ`

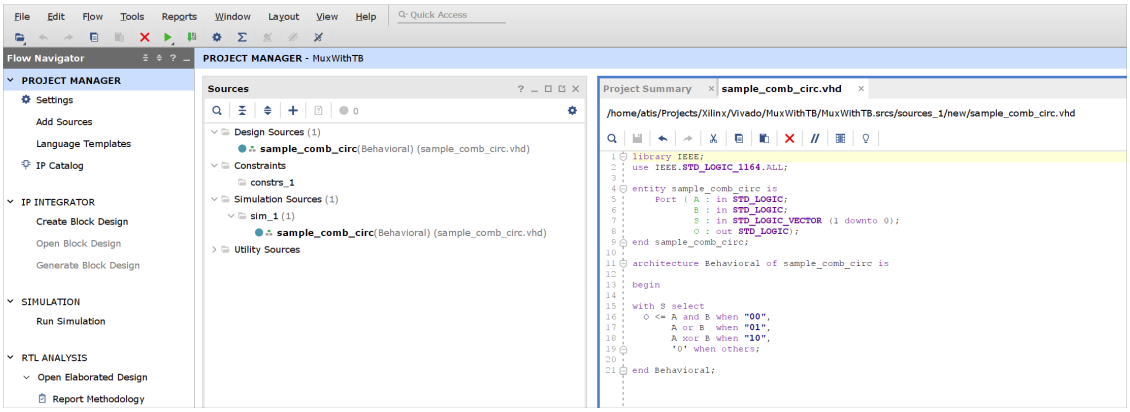

Figure 2: A new design source has been added to the project

Next we will add a corresponding test bench to the design

First, left click on the `sim_1 (1)` folder within `Simulation Sources` and click on the plus sign (+) above `Design Sources` sources to bring up the dialog.

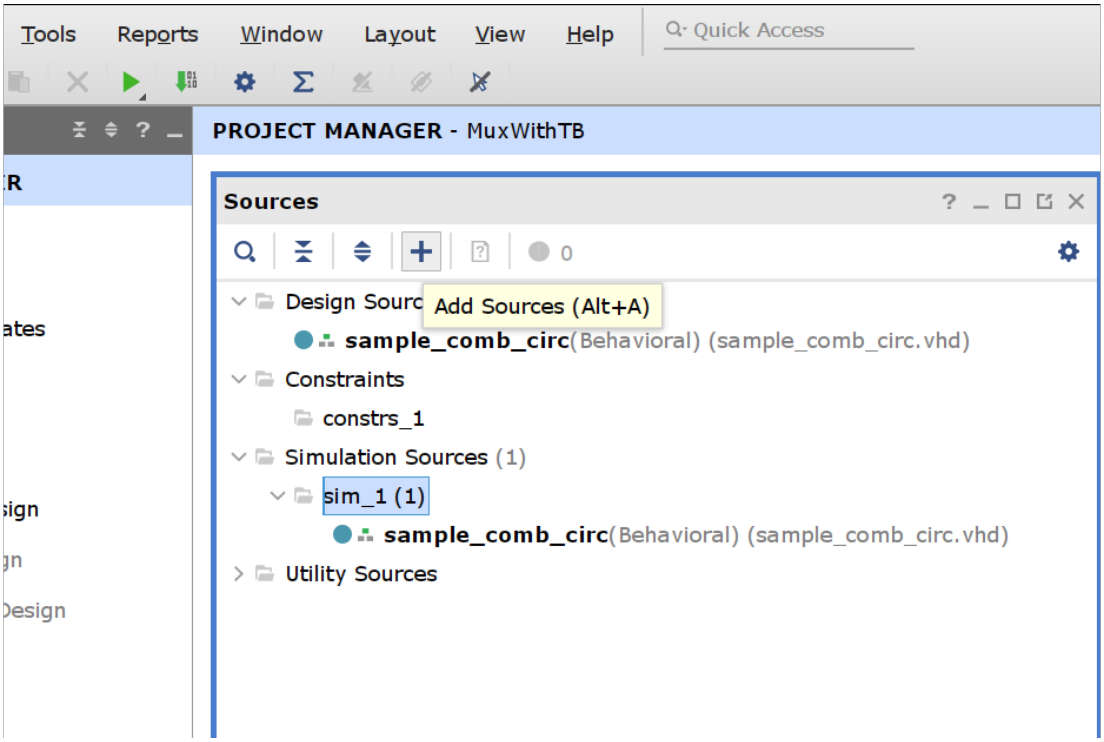

Figure 3: Add new source file

Then select `Add or create simulation sources`

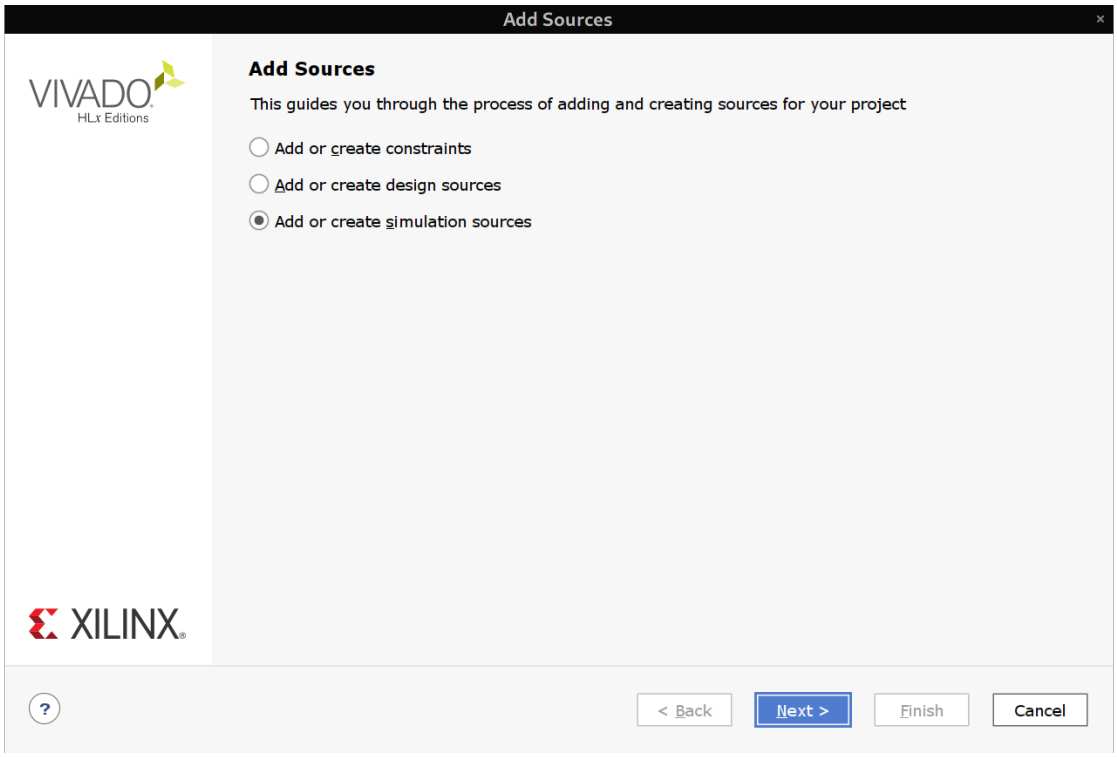

Figure 4: Add Sources dialog options

Then click on `Create File`

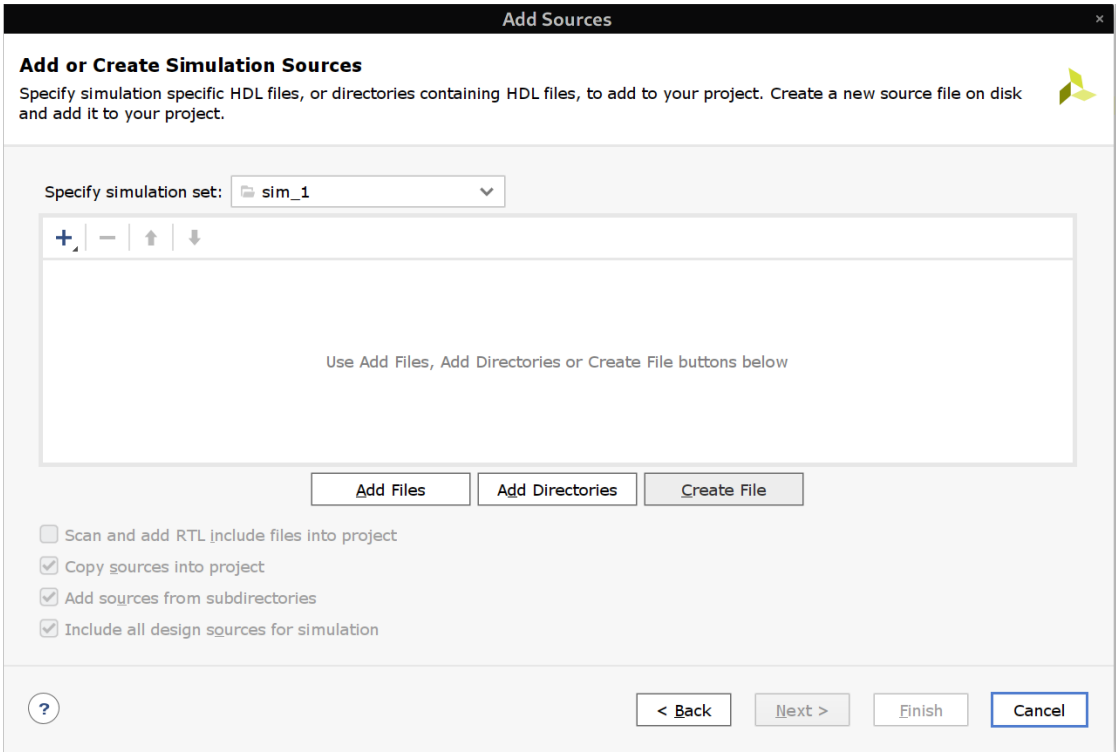

Figure 5: Create File from Add Sources dialog

Name the file as `sample_comb_circ_tb` and click on the `OK` button.

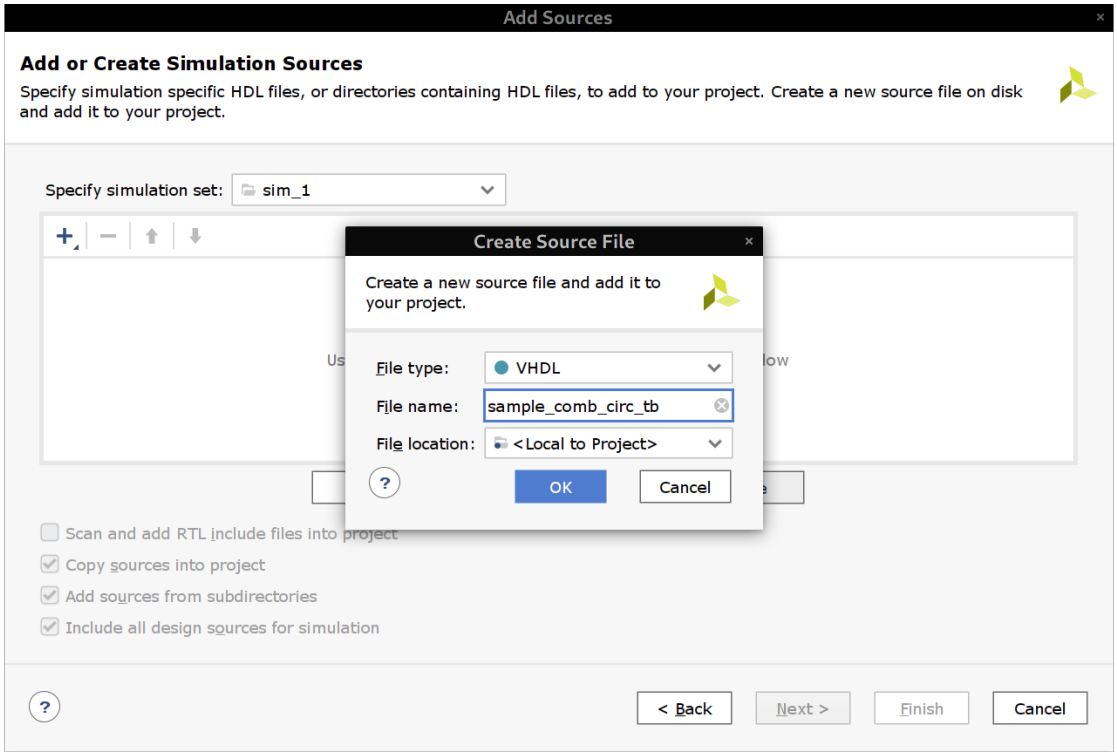

Figure 6: Assign a name to the new file

After the file has been added to the table, click the `Finish` button.

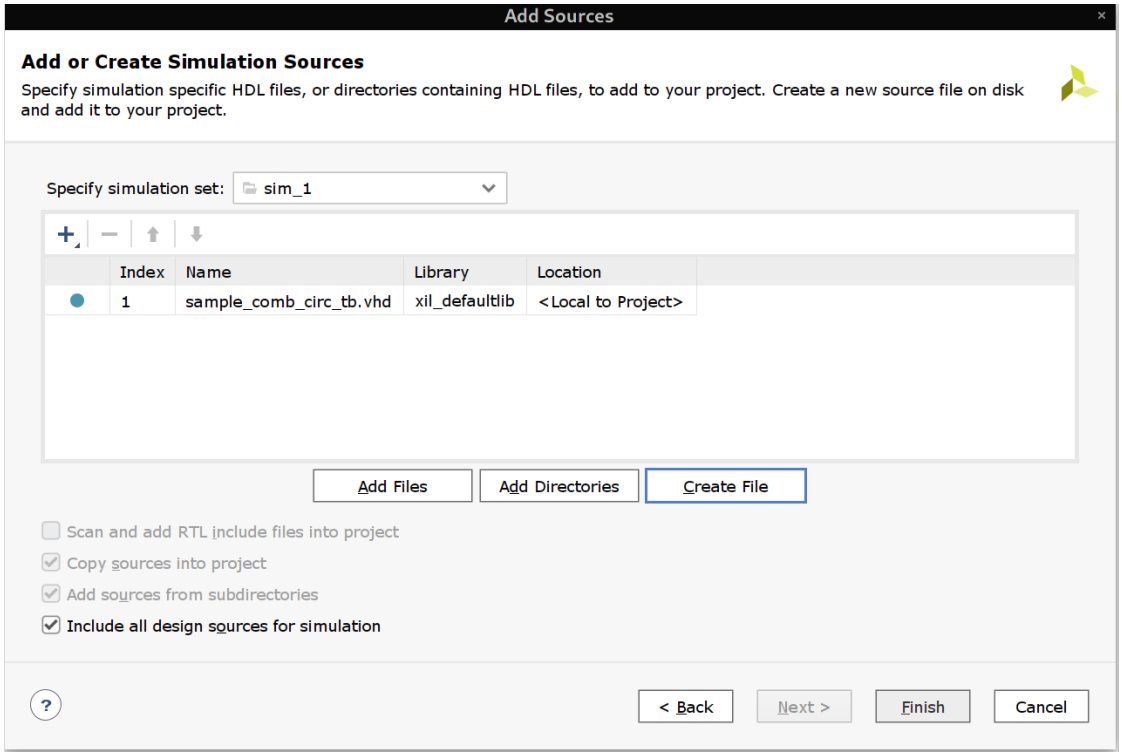

Figure 7: Finish creating a new file

A new dialog window will pop-up asking you to add **ports** to the *entity*.
Simply click on the `OK` button and when asked to verify, click `Yes`.

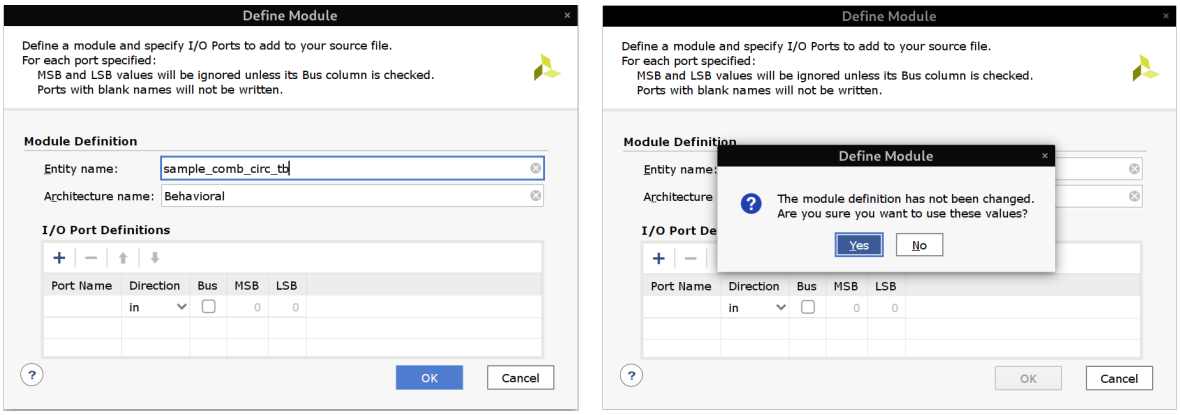

Figure 8: Verify that no ports will be added to the newly created simulation file

The new simulation source file should now be included within the `Simulation Sources`

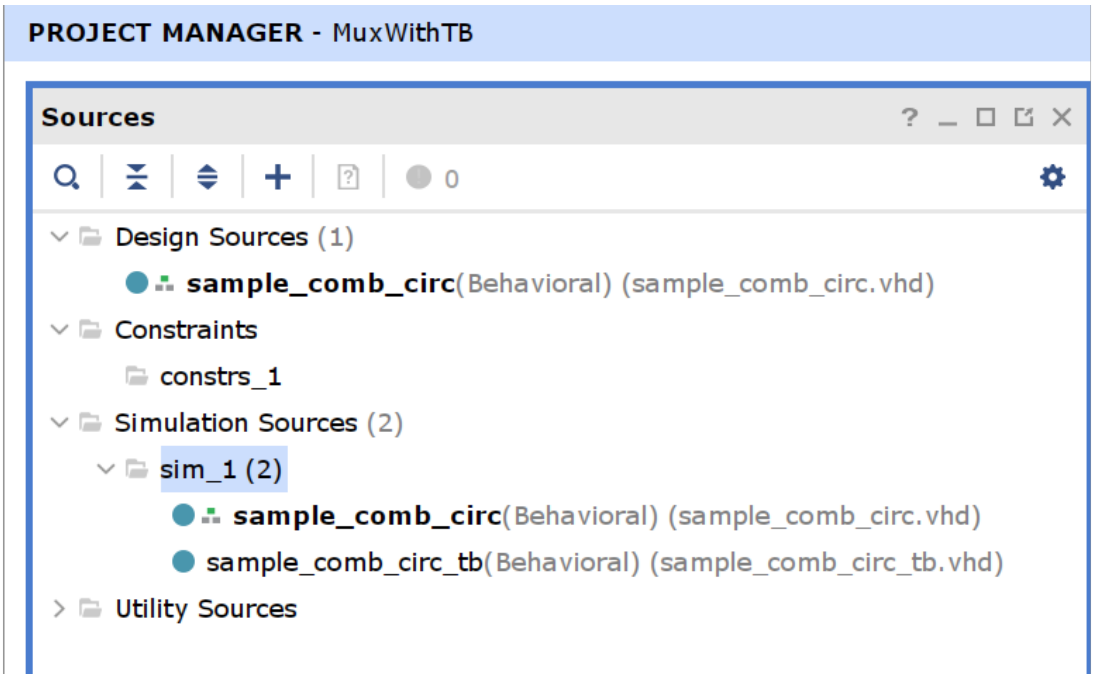

Figure 9: Newly created file has been added under Simulation Sources

Next we will modify the default vhd file generated from the template.

Clear the entire file (delete all the text), so that it would be clutter free and add the following code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- No port declarations necessary
entity sample_comb_circ_tb is
end sample_comb_circ_tb;

architecture Behavioral of sample_comb_circ_tb is

-- Declare component within the scope of the test bench
COMPONENT sample_comb_circ
PORT(
     A : in STD_LOGIC;
     B : in STD_LOGIC;
     S : in STD_LOGIC_VECTOR (1 downto 0);
     O : out STD_LOGIC
    );
END COMPONENT;

-- Add the necessary input signals
signal A : STD_LOGIC := '0';
signal B : STD_LOGIC := '0';
signal S : STD_LOGIC_VECTOR(1 downto 0) := "00";

-- Add the necessary output signals
signal O : STD_LOGIC;

begin

    -- Instantiate the declared component
    uut: sample_comb_circ PORT MAP (
      A => A,
      B => B,
      S => S,
      O => O
    );

    -- Begin combinational logic flow
    stim_proc: process
    begin
      -- hold reset state for 100ns
      wait for 100ns;

      -- insert stimulus here

      -- by default the MUX's selection vector is set to "00"
      -- so the it will out put the result from the AND gate
      A <= '1';
      B <= '1';
      wait for 100ns;
      A <= '1';
      B <= '0';
      wait for 100ns;
```

```
    -- Set the MUX to output the result from the OR gate
    S <= "01";
    A <= '0';
    B <= '1';
    wait for 100ns;
    B <= '0';
    wait for 100ns;

    -- and lastly the XOR gate
    S <= "10";
    A <= '0';
    B <= '1';
    wait for 100ns;
    A <= '1';
    B <= '0';
    wait for 100ns;
    A <= '1';
    B <= '1';
    wait;
  end process;

end Behavioral;
```

Once you have saved the file, observe that the instantiated component now appears within the test bench file.

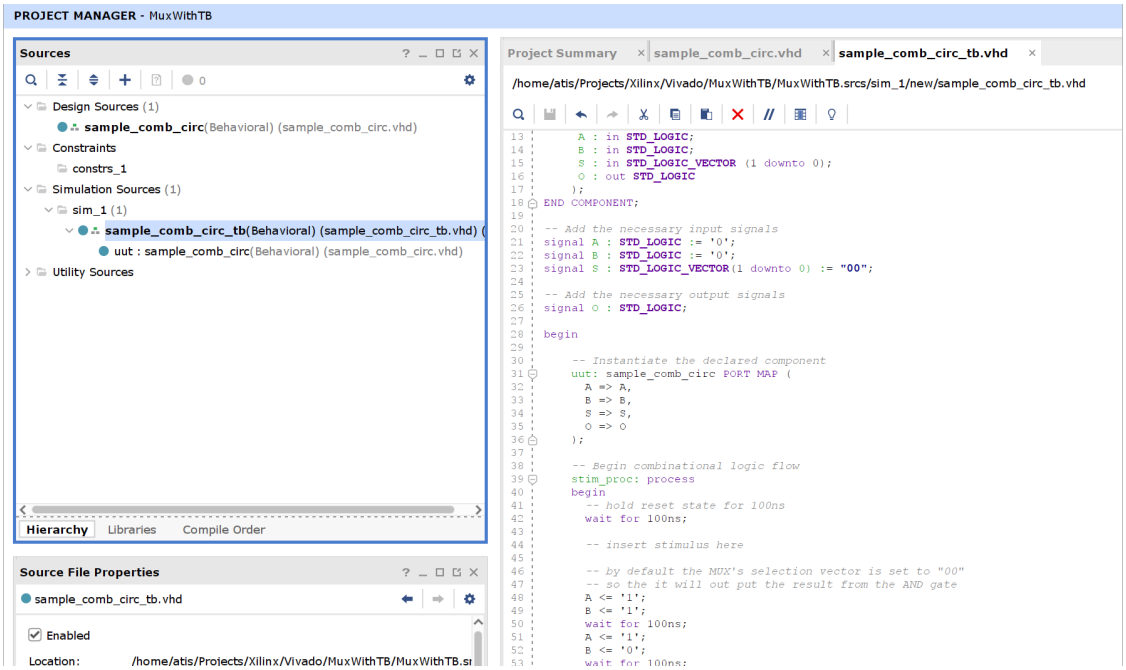

Figure 10: Component has been instantiated within the simulation file

Now that we have the simulation file ready, let's run it using Vivado's built-in simulator.
Within the `Flow Navigator` on the left side, expand SIMULATION and click on `Run Behavioral Simulation`.

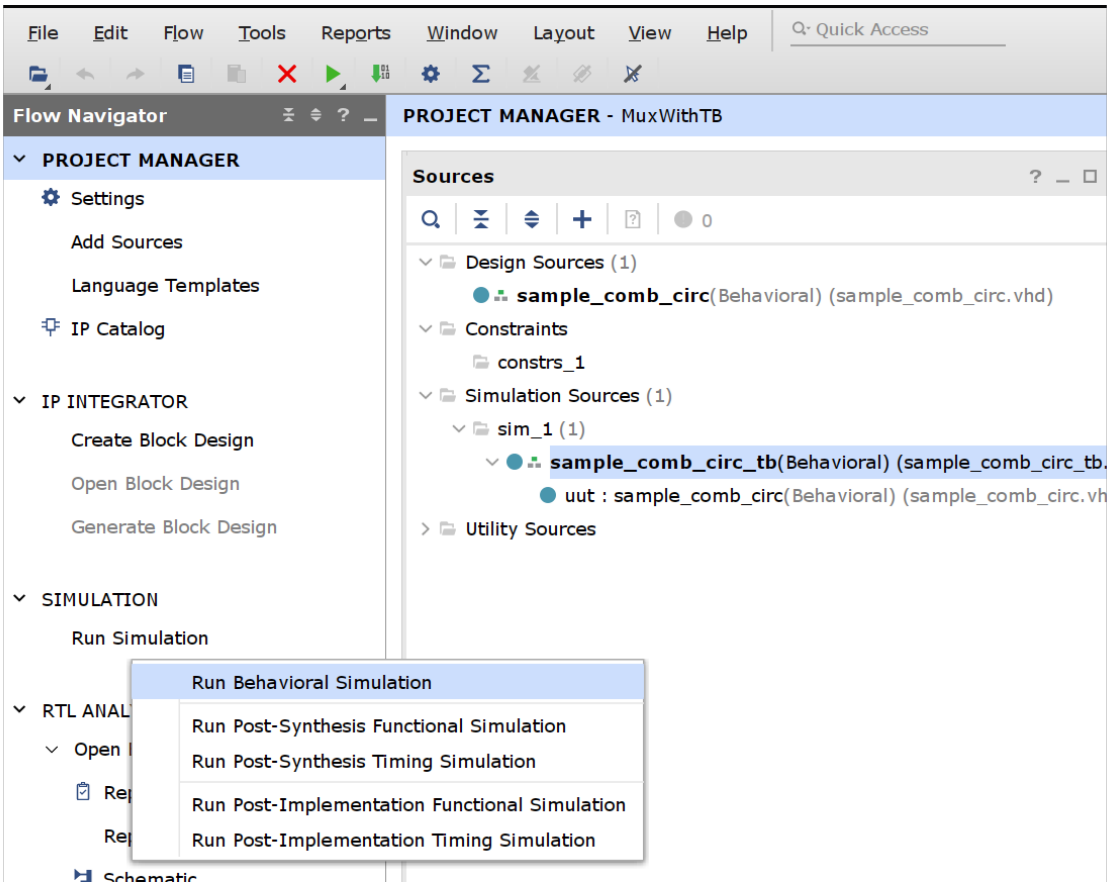

Figure 11: Click on Run Behavioral Simulation

After the simulation is complete, you will be taken to the resulting view where you can observe the signals that you have described within your test bench. Since the default time scale isn't for the duration of the entire simulation, click on the `Zoom Fit` button (highlighted in red) to get a better view

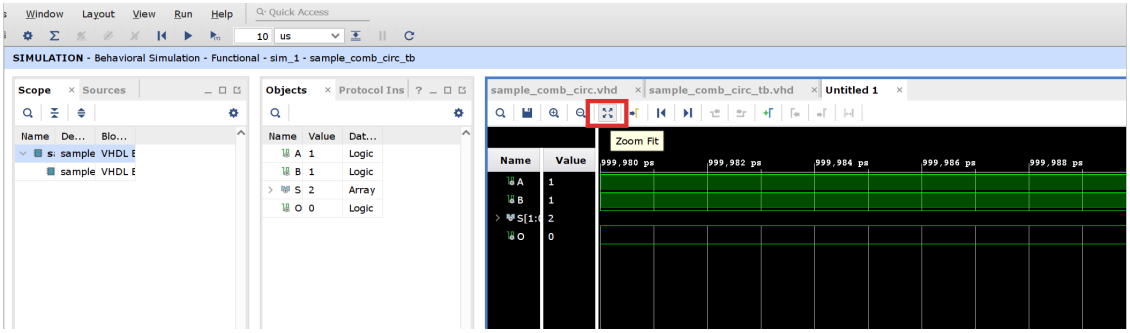

Figure 12: Press Zoom Fit to get a better view of the signals

Observe the signals to verify that the simulation has yielded the desired results.

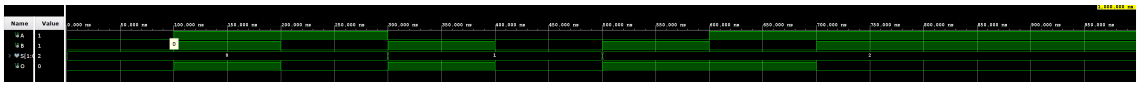

Figure 13: Verify the behavior of the component based on the results

6 Simulation of a sequential logic circuit

In this example, we are going to detail how to write a test bench for a circuit with a basic clock and without any additional delays included in the clock mechanism.

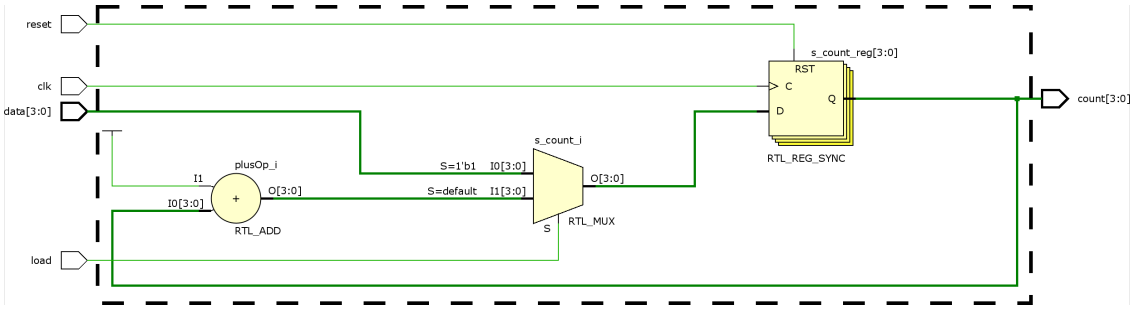

Figure 14: Sample sequential circuit

This is simple counter with a synchronous reset and the capability to load a pre-defined value into it. It has three inputs, one for the *clock* signal, one for the *reset*, and another two for the *load*ing a value.
The code for this schematic is as follows:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.STD_LOGIC_UNSIGNED.all;

entity sample_counter is
    PORT (
          clk   : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          load  : in  STD_LOGIC;
          data  : in  STD_LOGIC_VECTOR (3 downto 0);
          count : out STD_LOGIC_VECTOR (3 downto 0)
         );
end sample_counter;

architecture Behavioral of sample_counter is
signal s_count: STD_LOGIC_VECTOR (3 downto 0);
begin

    process(clk)
    begin
       if rising_edge(clk) then
          if reset = '1' then
             s_count <= (others => '0');
          elsif load = '1' then
```

```
            s_count <= data;
         else
            s_count <= s_count + '1';
         end if;
      end if;
   end process;

   count <= s_count;

end Behavioral;
```

Now, as for the test bench, here we need to pay attention to *two certain details*.
We would like the behavior of the clock to be independent of the rest of the functionality. One advantage of VHDL, is that anything that's *defined* as a basic *process* gets executed in parallel.
Therefore, we simply need to define a separate process for the clock and another one for the stimulus, just like in the example code.
Another important thing to keep in mind is also that **one control signal** should be changed only in **one process**, in other words avoid assigning values to signals in 2 or more processes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sample_counter_tb is
end sample_counter_tb;

architecture Behavioral of sample_counter_tb is

-- Add constants
constant T : time := 50 ns;

-- Declare component within the scope of the test bench
COMPONENT sample_counter
    PORT (
          clk :   in  STD_LOGIC;
          reset : in  STD_LOGIC;
          load  : in  STD_LOGIC;
          data  : in  STD_LOGIC_VECTOR (3 downto 0);
          count : out STD_LOGIC_VECTOR (3 downto 0)
         );
END COMPONENT;

-- Add the necessary input signals
signal clk   : STD_LOGIC := '0';
signal reset : STD_LOGIC := '0';
signal load  : STD_LOGIC := '0';
signal data  : STD_LOGIC_VECTOR (3 downto 0) := X"D";

-- Add the necessary output signals
signal count : STD_LOGIC_VECTOR (3 downto 0);

begin

    -- Instantiate the declared component
    uut : sample_counter PORT MAP (
                                   clk => clk,
                                   reset => reset,
```

```
                                    load => load,
                                    data => data,
                                    count => count
                                );

    -- continuous clock
    clk_proc: process
    begin
        clk <= '0';
        wait for T/2;
        clk <= '1';
        wait for T/2;
    end process;

    -- stimuli
    stim_proc: process
    begin
        -- hold reset for one clock cycle
        reset <= '1';
        wait for T;
        reset <= '0';

        -- initialize the value for load
        load <= '0';

        -- wait for four cycles and then reset again
        wait for 4 * T;
        reset <= '1';
        wait for T;
        reset <= '0';

        -- wait for six cycles and then load the value in the data vector
        wait for 6 * T;
        load <= '1';
        wait for T;
        load <= '0';
        wait;
    end process;

end Behavioral;
```

After running the simulation, make sure to verify the desired output.

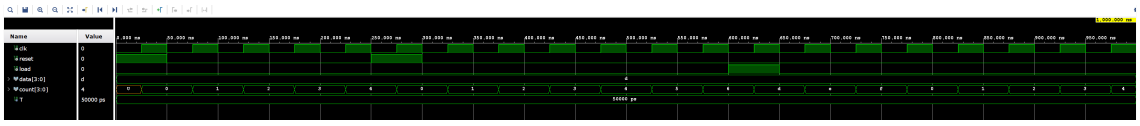

Figure 15: Simulation result

In Vivado, the **default simulation time** is set to **1000ns**.
We are able to change that within the simulation settings.

In order to do that, first *right-click on Run Simulation* which can be found under *Simulation* within the *Flow Navigator*.

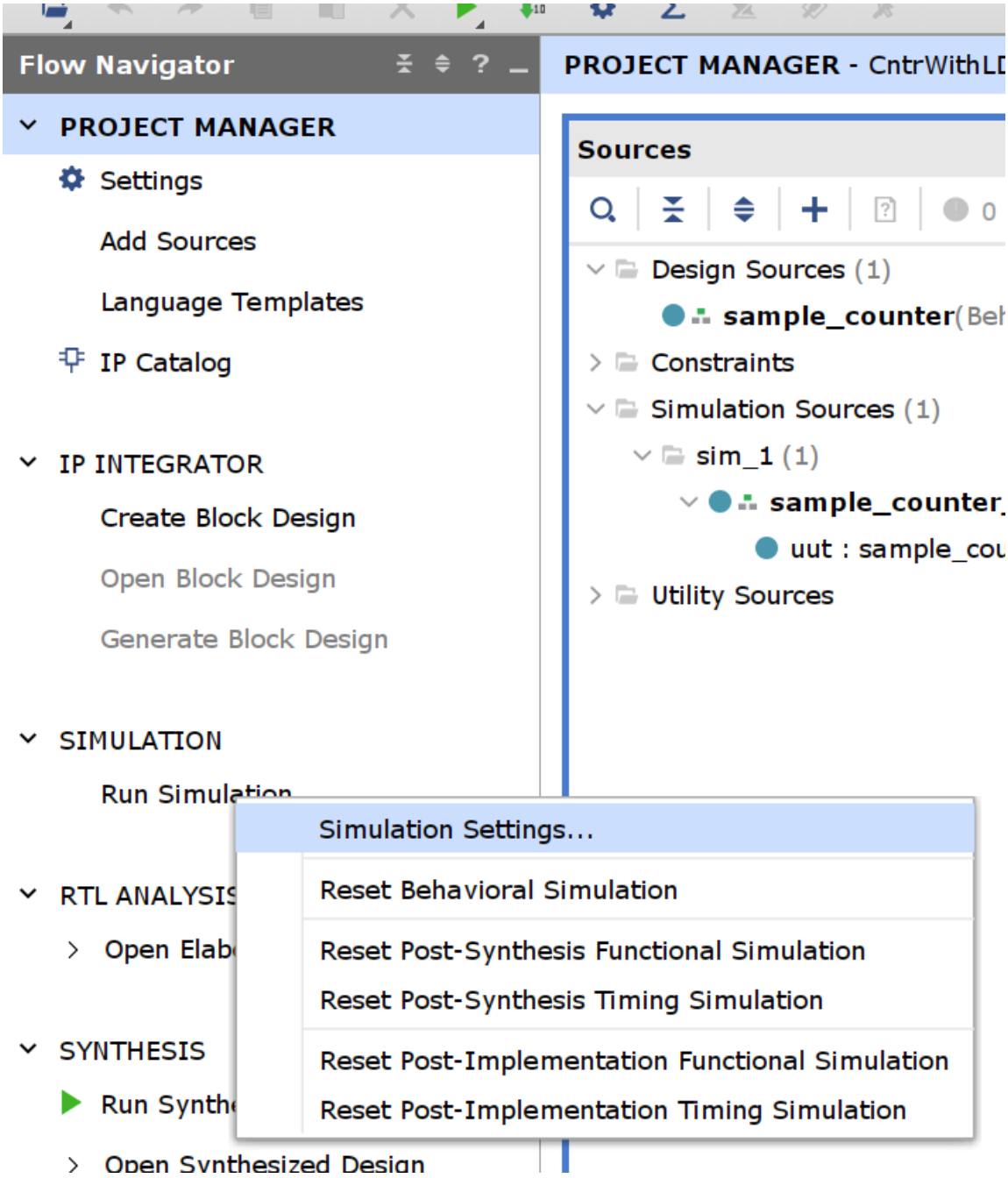

Figure 16: Right-click on Run Simulation

Then within the new dialog window
(1) First click on Simulation within the **Project Settings** area
(2) then click on the **Simulation** tab
(3) lastly, modify the value for the **xsim.simulate.runtime** key

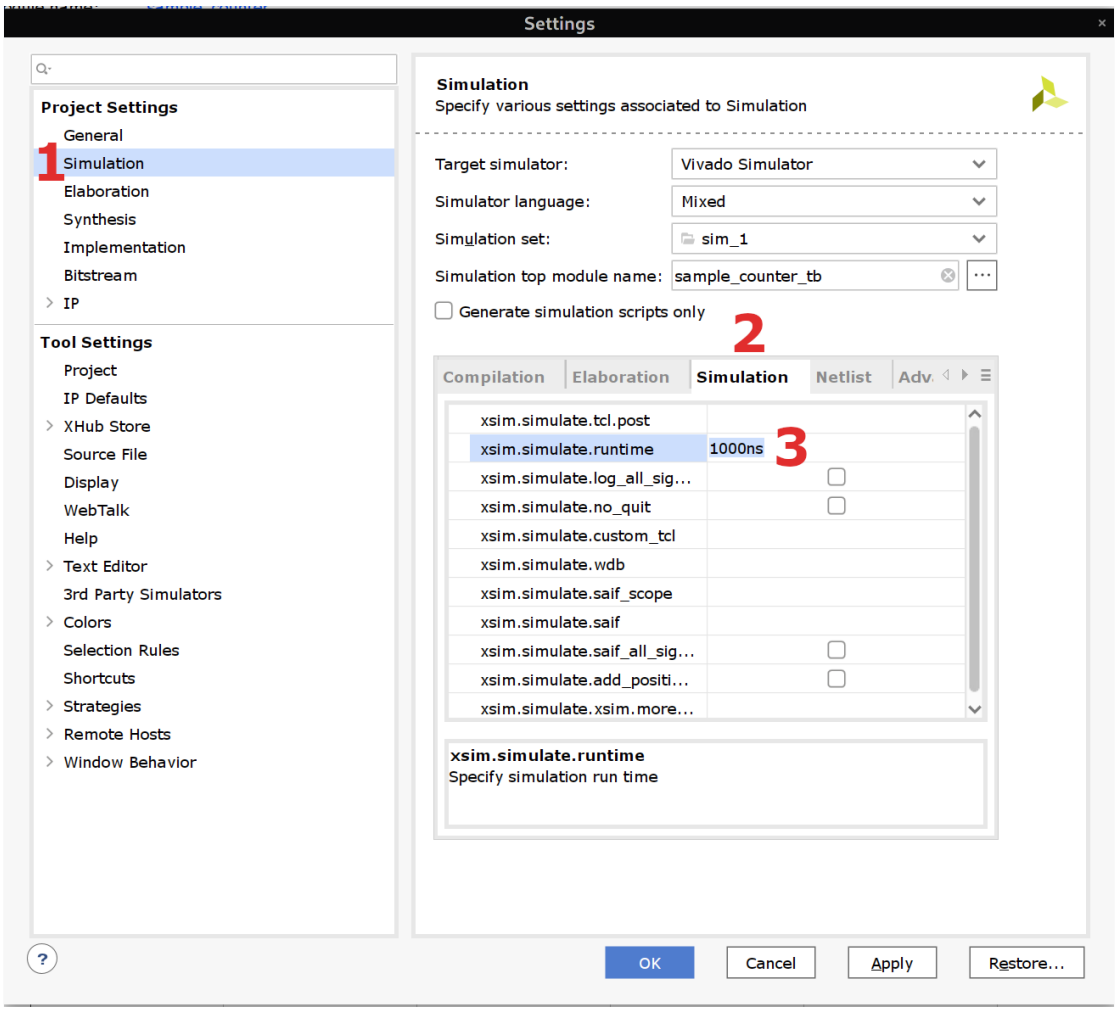

Figure 17: Steps to modify the simulation time

When you run the behavioral simulation like previously, you can make sure that now the simulation time corresponds to the value that has just been set.

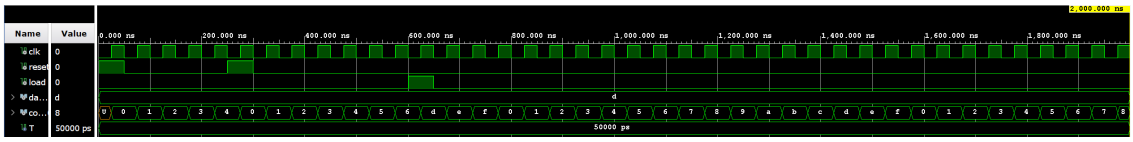

Figure 18: Simulation with the new run time value