Simulation Lab 7
Use ISE to Auto-Generate Testbench Simulations

National Science Foundation
Funded in part, by a grant from the National Science Foundation DUE 1003736 and 1068182

Step 1: steps should follow learning objectives

Task A:

Acknowledgements
Developed by Craig Kief, Alonzo Vera, Alexandria Haddad, and Quinlan Cao, at the Configurable Space Microsystems Innovations & Applications Center (COSMIAC). Based on original tutorial developed by Bassam Matar, Engineering Faculty at Chandler-Gilbert Community College, Chandler, Arizona. *Funded by the National Science Foundation (NSF).*

Lab Summary
The critical (and often most overlooked) part of any large VHDL FPGA project is the testing phase. In the early days of student's learning, they will often develop a project and then drop it onto the hardware for testing. Although this is great when a student is beginning, it is unacceptable in larger or more professional projects. When delivering commercial products, it will be expected that complete and accurate simulations will have been accomplished (where possible). The designer should always strive to write a module, test a module, write a module, test a module and then, combine the modules into a system and test the system. An average VHDL module should probably be no more than a few hundred lines of code so writing an exhaustive testbench is more manageable. It is our vision that writing testbenches is an excellent starting place for beginner engineers or technicians. Creating properly documented testbenches will save a design team countless hours of a very expensive senior engineer's time.

Lab Goal
The goal of this lab is to show the designer how to use the ISE wizard to create a testbench that can be used to test modules in a larger system. At the end of the lab, the designer should have gained the skills necessary to take any project, of any size, and create the structure and files necessary for testing.

Learning Objectives
1. Understand the hierarchical "tree" structure of projects and their test modules.
2. Using the wizard, create a testbench for the modules.
3. Create a testbench for the system.

Grading Criteria
Your grade is determined by your instructor.

Time Required
2 hours

Lab Preparation
- Read this document completely before starting the lab.
- On the COSMIAC FPGA webpage (http://www.cosmiac.org/Projects_FPGA.html), locate and download the file Lab7.zip file.
- Extract the source files onto your hard drive.

NOTE: Extract the source files to a folder (directory) without spaces in any part of the name.

Equipment and Materials

Students should work in teams of two or three. Each team of students will need the following supplies:

Supplies	Quantity
1. ISE® Design Suite (or WebPACK™) software from the Xilinx website, www.xilinx.com, if you don't already have it installed. Your classroom should have a full working version of Xilinx ISE® Design Suite. 2. FPGA kit including download and power cable(s). 3. Free Digilent Adept software (instructions for download and installation are included at the beginning of this lab): http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2	1

Additional References

The FPGA reference manual, data sheet, and any other supporting documents that may be of use.

ISE Design Suite User Guide :
http://www.xilinx.com/support/index.htm#nav=sd-nav-link-106173&tab=tab-dt

Digilent Adept User Guide:
http://www.digilentinc.com/Data/Software/Adept/Adept%20Users%20Manual.pdf

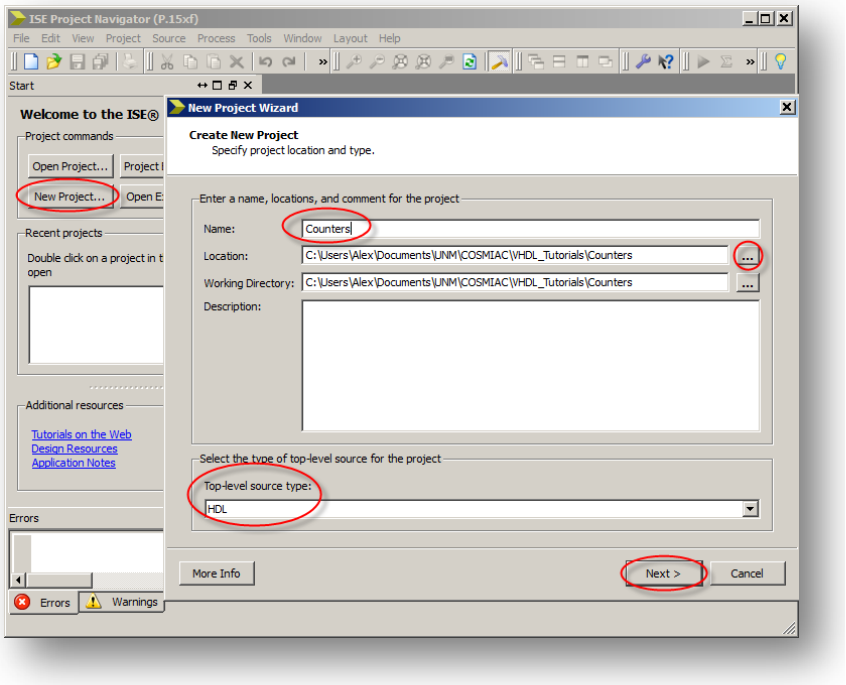
Lab Procedure 1: Create an ISE Project and Add Source Files

NOTE: Make sure you have completed the Lab Preparation before starting this Lab Procedure.

NOTE: If you've recently completed Lab 4, try creating the project and adding the source files on your own, without using the instructions. The project name is **Counter**, and the source files are **display_controller.vhd**, **counter.vhd**, and **counter4.ucf**.

NOTE: If you have just completed Lab 6, you can skip this Lab Procedure and start at Lab Procedure 2.

1. Open **Xilinx ISE Design Tool Project Navigator**.
2. Click **OK** to close the Tip of the Day.
3. Start a new project by selecting **File → New Project** from the menu or click the **New Project** button. The New Project Wizard starts.
   a. Type **Counters** in the Name text box.
   b. Select a location on your computer to save your project files by clicking the ellipsis (**...**) button to the right of the **Location** text box.
   c. Under **Top-level source type**, select **HDL**.
   d. Click **Next**.



NOTE: File names must start with a letter. Use underscores (_) for readability. Do not use hyphens (-); although the file name will work, the entity name will not.

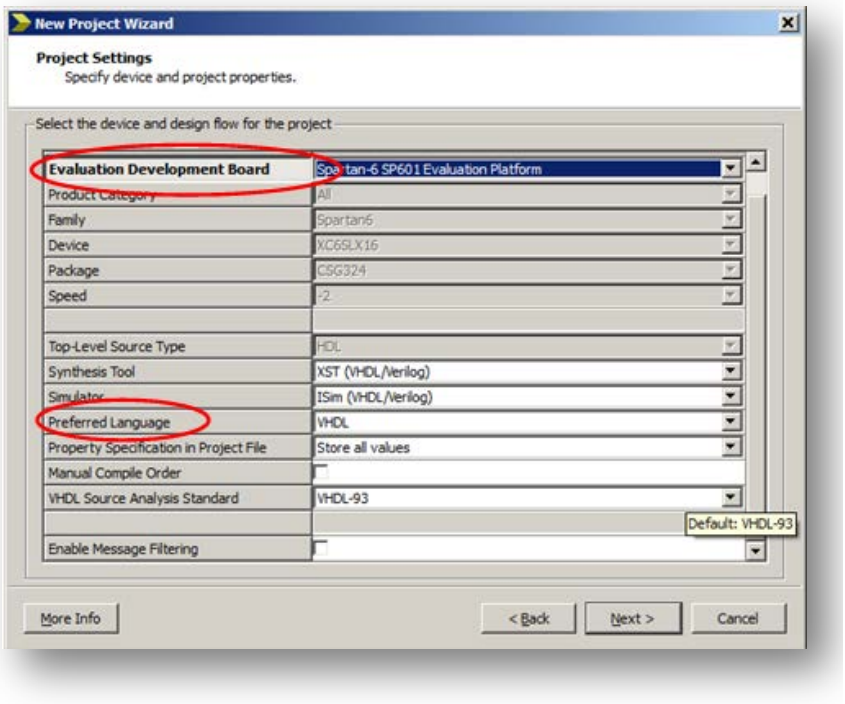
4. The Project Settings dialog box opens.

5. Select **Spartan-6 SP601 Evaluation Platform** from the **Evaluation Development Board** drop down menu.

   The **Product Category**, **Family**, **Device**, **Package**, and **Speed** should all automatically populate (top half of the screen). You will need to set some options in the lower half of the dialog box.

   Select **VHDL** from the **Preferred Language** drop down menu.

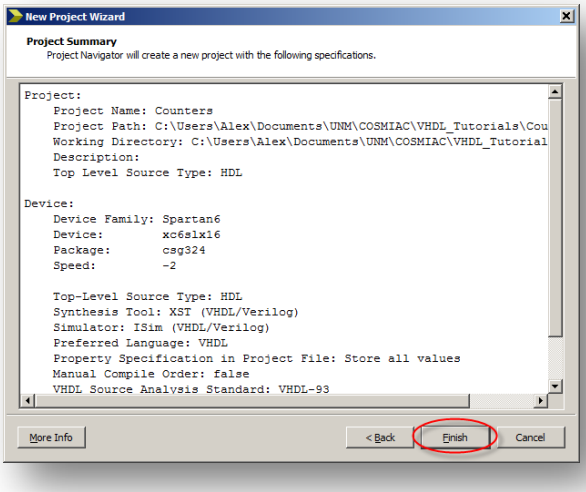
   The Project Settings should resemble the figure to the right.



6. Click **Next**.

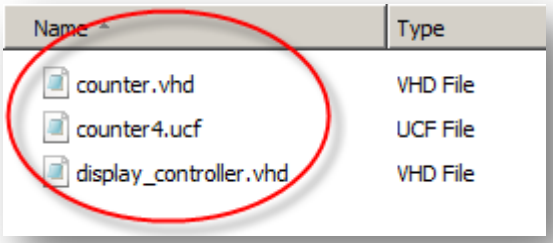
NOTE: The options specified are for the **Nexys 3, Spartan-6 FPGA**. Your board might be different than the board used for these instructions. Board specifications are printed on the FPGA chip in the middle of the board. The board information is also listed on the box it came in.

7. The Project Summary displays. Verify the file type and name are correct then click **Finish** to complete the New Project creation process.



NOTE: There are three main files associated with this tutorial. The project files are in the **counter.zip** file.

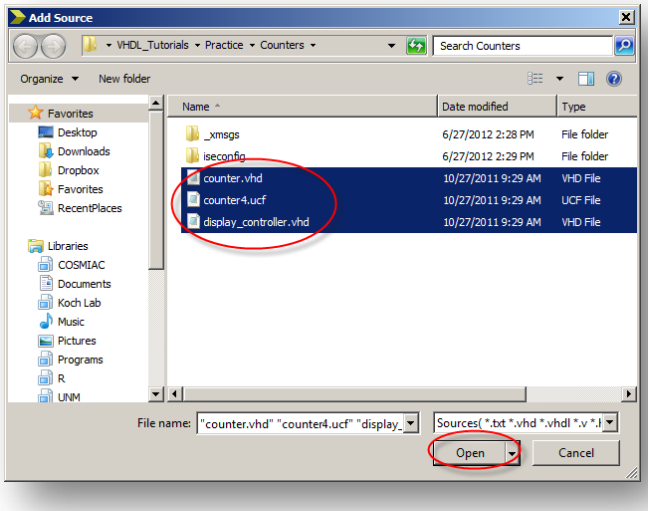
We will be adding the files from the counter.zip file to our new project.



8. Using Windows Explorer **move** these three files to your project folder.

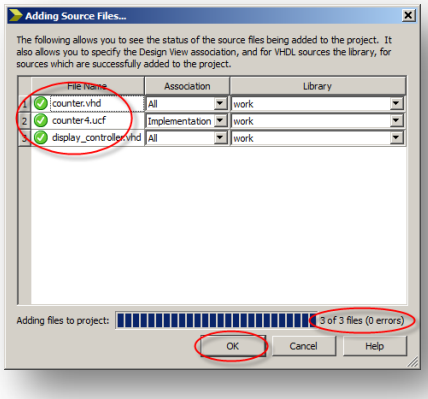
9. In Xilinx ISE® select **Project → Add Source** from the menu or **Right-click** in the Hierarchy pane and select **Add Source** from the shortcut menu. The **Add Source** dialog box opens.

10. **Select** the three files that were moved to the project directory earlier. **Click** the **Open** button. The **Adding Source Files...** dialog box opens.

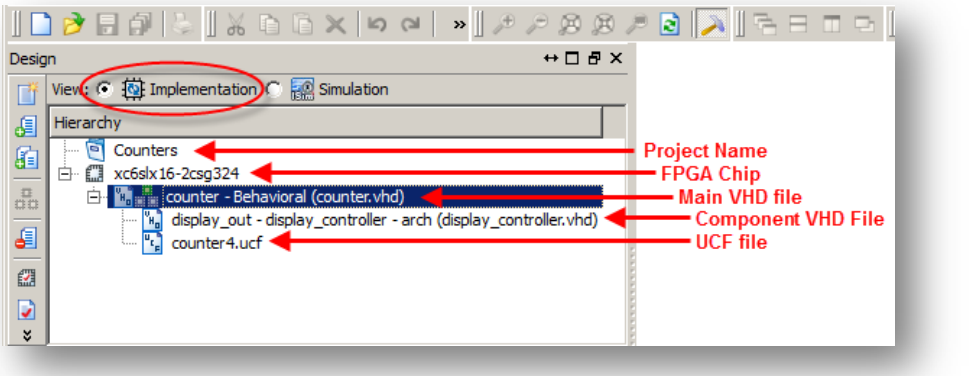


11. Click the **OK** button to complete the add files process.

NOTE: All three project files display with a green checkmark to the left of the file name. This lets us know that ISE understands the design association of each of the files.



12. The files are added to the **Counters** project.



NOTE: It is important to understand the file Hierarchy pane. The files display in a tree structure similar to the file and directory structure on a computer.

When Implementation is selected, the UCF file should be below the associated design file.

This image shows that at the top level is the project, then the chip type. Under the chip are the VHDL file, component VHDL file, and the associated UCF file.

Lab Procedure 2: Use the ISE Wizard to Create and Run Testbenches

Testbench Introduction

Senior FPGA engineers are expensive and their time is better spent creating the hardware design. We envision that the creation and generation of testbenches, for the senior engineer to evaluate, would be an excellent job for a technician or junior engineer. Additionally, this would provide a real cost savings for the company and provide more resources to the design engineer to develop product. Testing VHDL designs is an art and skill but one that can be mastered. The first step to this mastery is in this tutorial.

Before we jump into creating a testbench, let's briefly discuss what VHDL is and what it is used for:

- VHDL is an acronym for VHSIC Hardware Description Language.
- It is an IEEE standard.
- It was not initially created for FPGAs.

It was originally designed to document ASICs for the US DoD. Eventually the FPGA community realized that it was very useful for their work. For the purposes of these Labs, there are two types of VHDL: synthesizable and non-synthesizable. Synthesizable VHDL is used to program an FPGA. Non-synthesizable VHDL is used for testing the synthesizable code before it is programmed to the FPGA board. We create files called testbenches for testing our synthesizable designs.

Synthesizable VHDL is often thought of as something you can go to a parts store and purchase, i.e., you can purchase AND gates. Commands such as "wait for 50 ns" are viable VHDL code but it is impossible to buy "50 ns" in hardware. These are the things we use when we create a VHDL testbench. Their function is not to go to hardware, but to test the hardware design files.

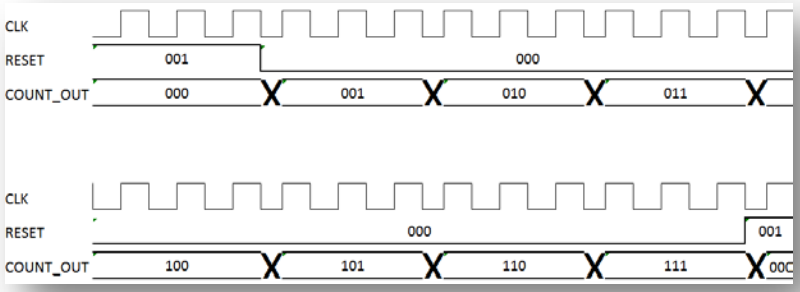
Testbenches are designed to ONLY stimulate inputs in order to determine if the outputs are as expected. There are two types of simulations that can be run, Behavioral and Timing. The behavioral simulation is where 90% of all simulations are performed.

As a designer, you should be aware that it takes a certain amount of time for signals to get from the input to the output of a chip. This time is called delta T (or ΔT). This ΔT is different for each chip and each design. The behavioral simulation sets ΔT equal to zero. This gives us the benefit of running the simulation to check for flaws in the logic of the design.

The timing simulation is performed after Synthesis and Implementation are completed, after the design has been placed into the FPGA and routed to the input and output pins. At this point, the ISE suite has been able to accurately compute the ΔT. We now know exactly how long it takes to go from the input pins, through the internal components, and then to the output pins. Once ΔT is known we can run the simulation much more accurately. Timing simulations are often not run in undergraduate courses, however it is important for the designer to know that they exist and to understand the difference between a behavioral and timing simulation. We can use the same testbench for both types of simulations.

When creating a testbench it is a good idea to visualize beforehand what you think the waveforms should look like. For our counter design, the clock will run off and on at regular intervals. When the reset button is activated (high or 1) the count is set to zero. Once the reset button is inactive, the count will start

incrementing on a rising edge of the clock. Because we have created a slow clock signal, the count increments after several clock cycles. Our testbench waveform should look something similar to this:



We will need to create two testbenches, one for the counter and once for the display_controller. The initial creation of the testbench is fairly simple using ISE. For each testbench we will create a new VHD Testbench source and associate the appropriate vhd file with that source. ISE will auto-generate a testbench template that we will modify based on our previous visualization. The modification of the testbench takes a little practice. Often if the junior engineer can get to this point, the senior engineer can give them more precise timing based on the final design of the module or system.

Because VHDL is such a versatile language there are several ways to define the testbench inputs. The following are examples of basic approaches that can be expanded upon for more complex designs.

Step 1: Create a Testbench for the Counter

1. Because our code has created a slow clock, we need to change two settings.

   Change the **SLOW_CLK <= CLK_DIVIDER** setting to zero (**0**) in the **counter.**

counter.vhd

```
73    SLOW_CLK <= CLK_DIVIDER(0);                -- SLOW_CLK gets
74    -- ********************************************************
75    -- for the test bench change SLOW_CLK <= CLK_DIVIDER to 0,
76    -- ********************************************************
77    COUNT_BLINK <= CLK_DIVIDER(23);            -- COUNT_BLINK ge
78                                               -- COUNT_BLINK is
```

   Change the **constant N: integer** setting in the **display_controller**, to **2**.

display_controller.vhd

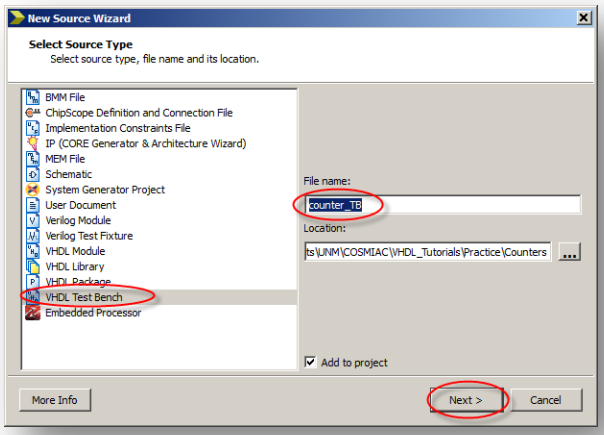
```
29    -- ***************************************
30       -- CHANGE N value to 2 for test bench,
31    -- ***************************************
32       constant N: integer := 2;
33
```

2. Create a testbench for your file. **Right-click** in the Hierarchy Pane and select **New Source** from the short cut menu.

3. Select **VHDL Testbench** as the source type.

4. Enter **counter_TB** as the file name.

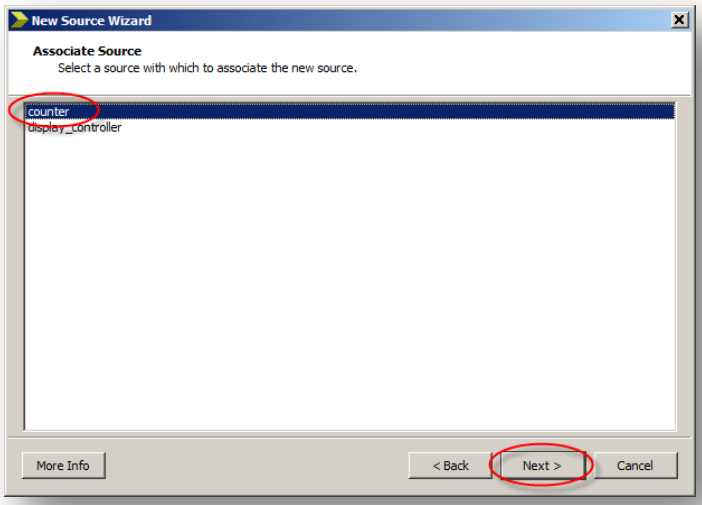
NOTE: Keeping track of a module's vhd file and its associated testbench is much easier when you name the testbench the same name as the vhd file with _TB appended to the end of the file name.



5. Click the **Next** button. The Associate Source dialog box opens. Select **counter**, since that is the module we are currently testing.

6. Click the **Next** Button.

7. The New Source Summary dialog box displays. Click the **Finish** button.



8. ISE auto-generates a testbench based on the associated source file. We will need to make some minor modifications to this file.

9. Make the following changes (highlighted) to the counter_TB.vhd file:

```
-- ensure your test bench declairs the same libraries and packages used with your source files
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use ieee.numeric_std.all;
--------------------------------------------------------------------------------------------------------
ENTITY counter_TB IS
END counter_TB;
--------------------------------------------------------------------------------------------------------
ARCHITECTURE behavior OF counter_TB IS
    -- Component Declaration for the Unit Under Test (UUT);
    -- component is counter since we are testing the counter.vhd file
    -- ports are declared here, they are copies of the i/o ports from counter
    COMPONENT counter
    PORT(
         clk : IN  std_logic;
         reset : IN  std_logic;
         pause : IN  std_logic;
         count_out : OUT  std_logic_vector(15 downto 0);
         count_blink : OUT  std_logic;
         an : OUT  std_logic_vector(3 downto 0);
         sseg : OUT  std_logic_vector(6 downto 0)
        );
    END COMPONENT;

   --Inputs; set to the desired starting value
   signal clk : std_logic := '0';
   signal reset : std_logic := '1';    -- change to 1, we want to start the test with the reset active
   signal pause : std_logic := '0';

   --Outputs
   signal count_out : std_logic_vector(15 downto 0);
   signal count_blink : std_logic;
   signal an : std_logic_vector(3 downto 0);
   signal sseg : std_logic_vector(6 downto 0);

    -- Clock period definitions
    constant clk_period : time := 10 ns;

BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: counter PORT MAP (
            clk => clk,
            reset => reset,
            pause => pause,
            count_out => count_out,
            count_blink => count_blink,
            an => an,
            sseg => sseg
          );

    -- Clock process definitions
    clk_proc : process
    begin
       clk <= '0';
       wait for clk_period/2;
       clk <= '1';
       wait for clk_period/2;
    end process;

    -- Reset process
    reset_proc : process
    begin
       -- hold reset state for 22 ns.
       wait for 22 ns;
         reset <='0';
         wait for 1000 ns;
         reset <='1';
         wait for 18 ns;
         reset <='0';
         wait;
    end process;

    -- Pause process
    pause_proc : process
    begin
       wait for 500 ns;
       pause <= '1';
       wait for 50 ns;
       pause <= '0';
       wait for 1000 ns;
    end process;

END;
```

10. Click the **Save** button and save your work.

For the counter we are testing three inputs, CLK, RESET, and PAUSE. We will test the inputs using three stimulus processes that run concurrently. The clock process oscillates the clock from zero to one, each for half of the defined clk_period.

The reset process starts with reset active (equaling one) for the first 22 ns, which should hold our count at zero for this time period. Then we release the reset, setting it equal to zero, and let the counter run for 1000 ns. Next we activate RESET again for 18 ns to make sure that the counter will restart at zero. Finally we set reset back to zero.

NOTE: Choosing 1000 ns was arbitrary. Before creating the testbench, during the visualization process, we assumed that if the counter was working for a certain long period of time, then it was probably working all the time. We could have chosen 50,000 or 65,535 ns. The last choice would have tested every count option but would have taken a really long time to simulate. For a classroom exercise, this probably isn't necessary. For a real-life application testing, that test would probably be needed.

The last process is for the pause button. For the first 500 ns the pause is at zero. Then it is set to one and held for 50 ns. During this time the count should hold at whatever number it is currently on. After this time, pause is set back to zero, and the count continues for another 1000 ns.

What we are looking for is a standard clock waveform with RESET holding COUNT_OUT at zero for 22 ns. After that time COUNT_OUT increments by one on a rising edge of the clock for 478 ns.

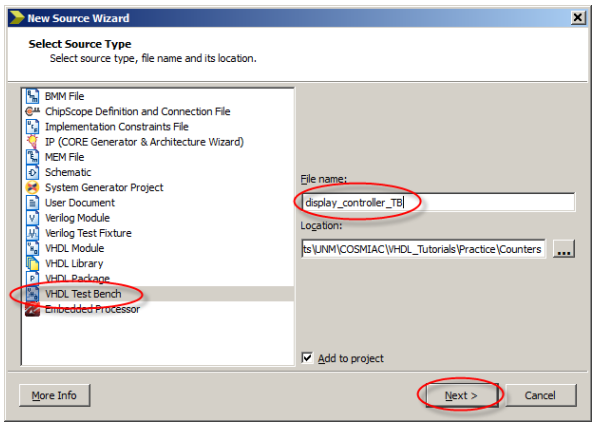
At 500 ns (478 + 22 = 500) the PAUSE is activated for 50 ns and COUNT_OUT will hold at the current count for 50 ns, or until 550 ns. After this the count will increment again until 1022 ns when RESET is again set to one for 18 ns and COUNT_OUT starts again at zero.

NOTE: In order to see the entire test, you probably will need to enter the run command in the ISim console window. Type **run 2000 ns** in the console window at the **ISim>** prompt. (You can choose any time period you want.)

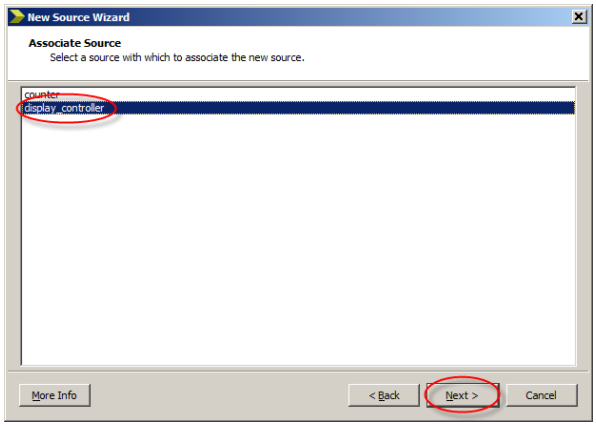
The times used are not rigid; they are merely a starting place to begin the design test. This is a bare minimum test. In reality, a variety of different inputs would be chosen in an effort to fail the design. It is always better for a design to fail in simulation rather than after implementation to the hardware, when a customer is depending on it.

Step 2: Create a Testbench for the Display_Controller

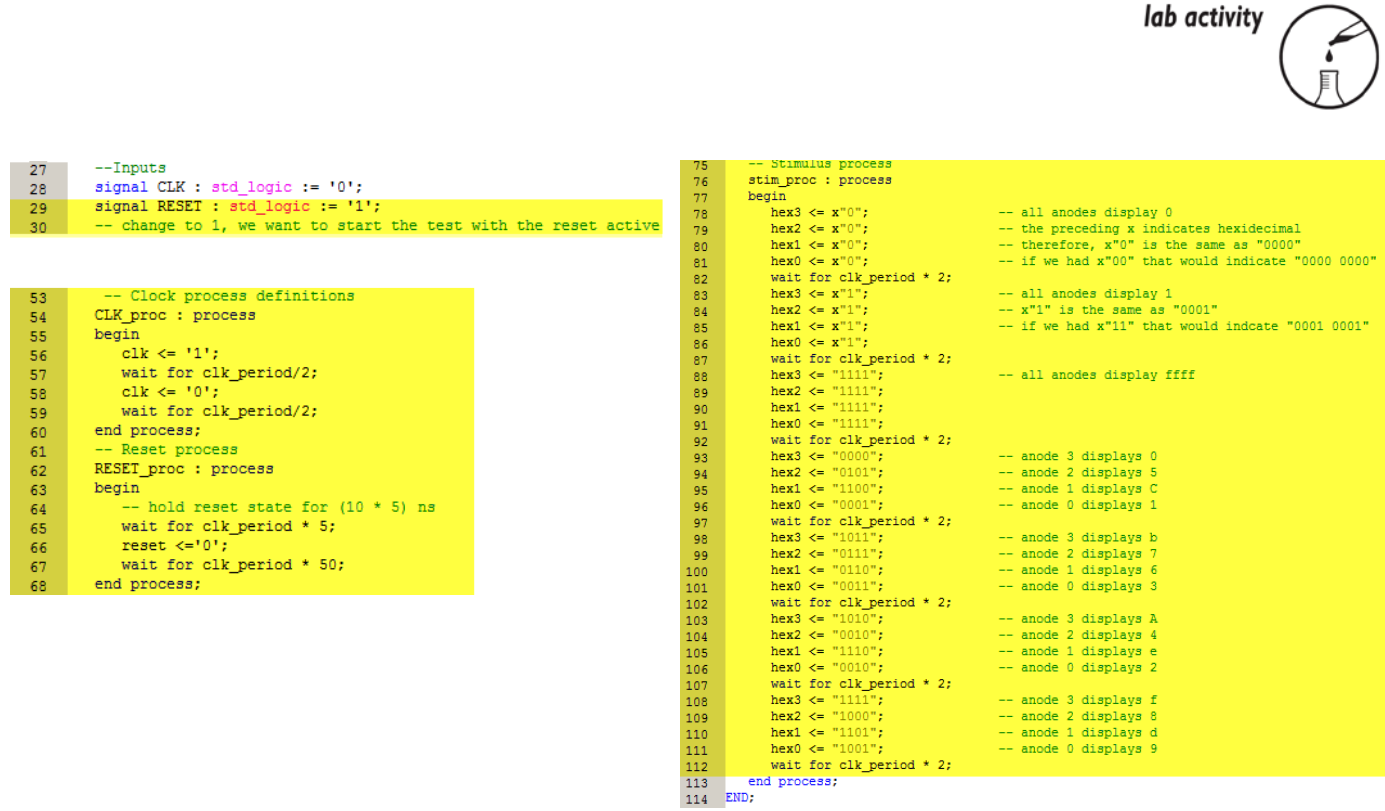
1. We will now create the testbench for the 2nd module of this project, the display_controller.
2. **Right-click** in the Hierarchy Pane and select **New Source** from the short cut menu.
3. Select **VHDL Testbench** as the source type.
4. Enter **display_controller_TB** as the file name.



5. Click the **Next** button. The Associate Source dialog box opens. Select **display_controller** as the source.
6. Click the **Next** button and then the **Finish** button when the New Source Summary dialog box displays.



7. Just like with the counter_TB file, ISE creates a testbench file that we will modify.
8. Make the following changes (highlighted) to the display_counter_TB.vhd file:



```
27    --Inputs
28    signal CLK : std_logic := '0';
29    signal RESET : std_logic := '1';
30    -- change to 1, we want to start the test with the reset active
```

```
53     -- Clock process definitions
54    CLK_proc : process
55    begin
56       clk <= '1';
57       wait for clk_period/2;
58       clk <= '0';
59       wait for clk_period/2;
60    end process;
61    -- Reset process
62    RESET_proc : process
63    begin
64       -- hold reset state for (10 * 5) ns
65       wait for clk_period * 5;
66       reset <='0';
67       wait for clk_period * 50;
68    end process;
```

```
75    -- Stimulus process
76    stim_proc : process
77    begin
78       hex3 <= x"0";              -- all anodes display 0
79       hex2 <= x"0";              -- the preceding x indicates hexidecimal
80       hex1 <= x"0";              -- therefore, x"0" is the same as "0000"
81       hex0 <= x"0";              -- if we had x"00" that would indicate "0000 0000"
82       wait for clk_period * 2;
83       hex3 <= x"1";              -- all anodes display 1
84       hex2 <= x"1";              -- x"1" is the same as "0001"
85       hex1 <= x"1";              -- if we had x"11" that would indcate "0001 0001"
86       hex0 <= x"1";
87       wait for clk_period * 2;
88       hex3 <= "1111";            -- all anodes display ffff
89       hex2 <= "1111";
90       hex1 <= "1111";
91       hex0 <= "1111";
92       wait for clk_period * 2;
93       hex3 <= "0000";            -- anode 3 displays 0
94       hex2 <= "0101";            -- anode 2 displays 5
95       hex1 <= "1100";            -- anode 1 displays C
96       hex0 <= "0001";            -- anode 0 displays 1
97       wait for clk_period * 2;
98       hex3 <= "1011";            -- anode 3 displays b
99       hex2 <= "0111";            -- anode 2 displays 7
100      hex1 <= "0110";            -- anode 1 displays 6
101      hex0 <= "0011";            -- anode 0 displays 3
102      wait for clk_period * 2;
103      hex3 <= "1010";            -- anode 3 displays A
104      hex2 <= "0010";            -- anode 2 displays 4
105      hex1 <= "1110";            -- anode 1 displays e
106      hex0 <= "0010";            -- anode 0 displays 2
107      wait for clk_period * 2;
108      hex3 <= "1111";            -- anode 3 displays f
109      hex2 <= "1000";            -- anode 2 displays 8
110      hex1 <= "1101";            -- anode 1 displays d
111      hex0 <= "1001";            -- anode 0 displays 9
112      wait for clk_period * 2;
113   end process;
114   END;
```

9. Click the **Save** button and save your work.

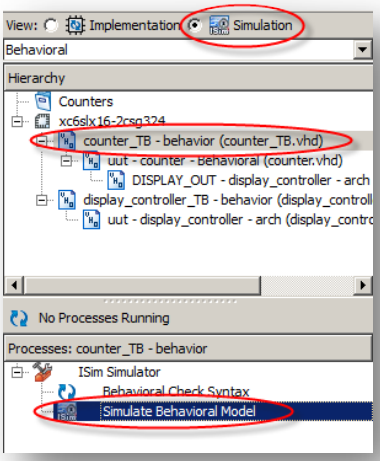
The display_controller_TB has six inputs, CLK, RESET, HEX3, HEX2, HEX1, and HEX0. We set up a stimulus process for each of these inputs. The CLK and RESET processes are the same as the counter_TB.

The HEX3-0 would normally get their input from the counter.vhd main module. However, this testbench is only testing the display_controller sub-module. We will have to feed the input to the HEX3-0. The example code shows seven possible inputs for HEX3-0. You could set up any combination of HEX3-0 for the testbench that you wanted.

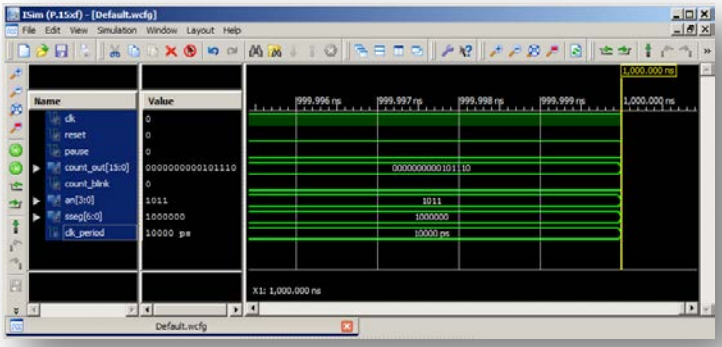
NOTE: In a real-world situation each option would be tested. In other words, for this design, there would be 65,000+ options to test. For complex designs, the verification process can become more complex and time consuming than the initial design. As the testing gets more complex, the person doing the verification uses a set of procedures and functions in the testbench. These procedures and functions run the program and let the designer know if the code is working as specified.

Step 3: Run the Simulation of the *counter_TB* and *display_controller_TB*

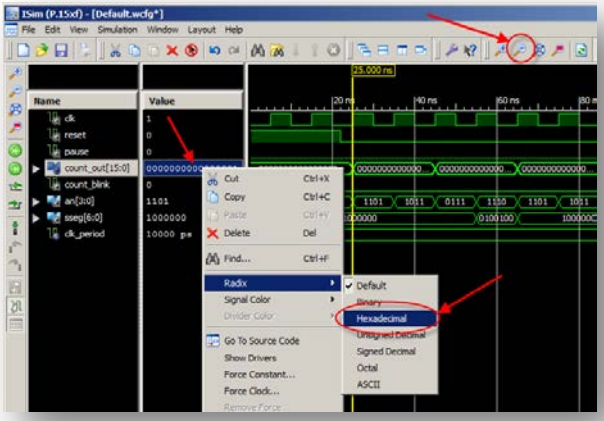
1. Select **Simulation** from the **Design Pane** to see the list of simulation files that ISE has created for this testbench.
2. In the Hierarchy pane select the counter_TB file.
3. In the Processes pane expand the ISim Simulator option if needed.
4. Double-click Simulate Behavioral Model.



5. ISim will open and you should see something similar to the image on the right.



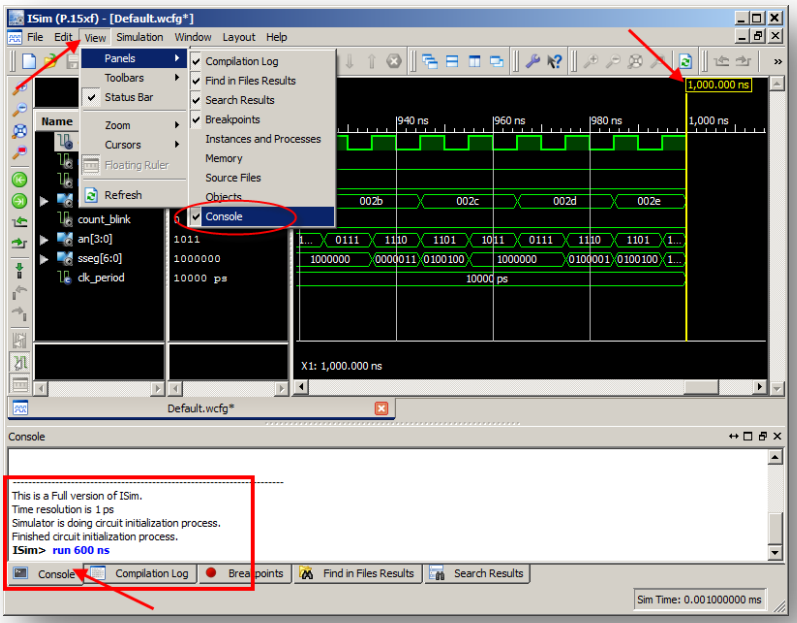
6. Use the Zoom Out button to zoom out of the waveform until you can see the clock cycle.
7. Right-click the count_out value, select Radix then Hexadecimal. This will change the display value of the count_out signal to hexadecimal instead of binary values.



NOTE: ISim initially runs the simulation for 1000ns. In order to run the simulation for the full length of the test (1550 ns), or longer, we will use the run command.

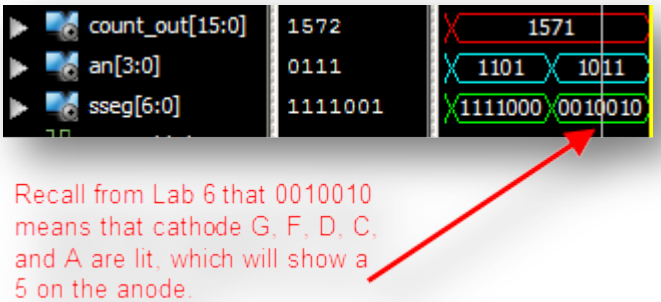
8. If the console is not displayed, select **View → Panels → Console** from the menu.

9. In the Console Panel, at the **ISim>** prompt type **run 600 ns** and press enter.



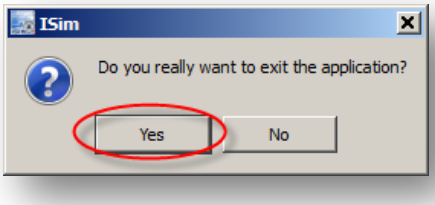
NOTE: In order to run the simulation all the way through ffff restarting at 0000, the simulation must run for a total of ~1,355,460 ns.

NOTE: The clock is driving the AN (anode) output. Recall that the 7-segment cathodes can only display on one anode at a time. The SSEG output is driven by the COUNT_OUT character in the zero spot of AN. For example, if AN equals 1011, and COUNT_OUT's value is 1571, then SSEG's output will be 5.

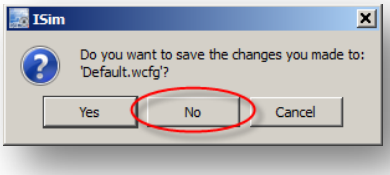


10. If you are satisfied with the simulation results, close ISim by clicking the close button in the upper right corner.

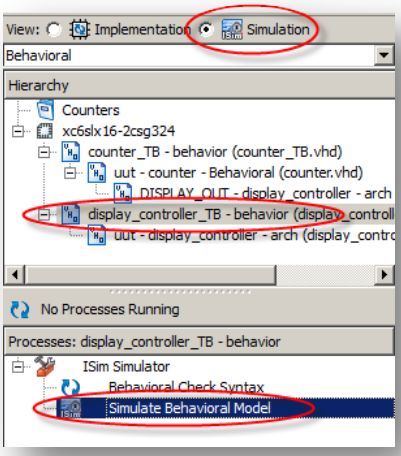
11. Click the Yes button to exit the simulator.



12. Click the No button, because we do not want to save this simulation.

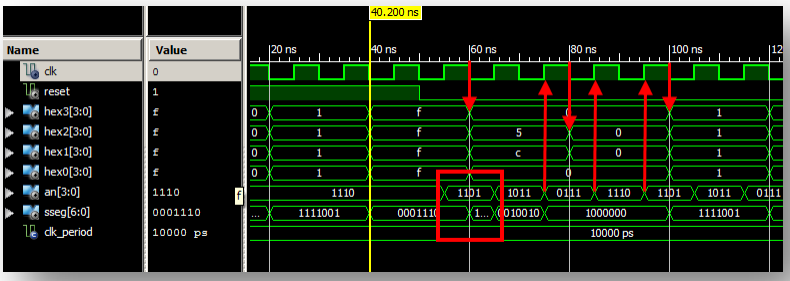


13. Run the display_controller_TB.
14. In the Hierarchy pane select the display_controller_TB file.
15. Double-click Simulate Behavioral Model.
16. ISim starts and shows the display_controller_TB waveform.
17. Select HEX3-0 and change the radix to hexadecimal.



NOTE: Notice that after RESET goes to zero, all changes are happening on the falling edge of the clock. Also notice that around 60 ns the SSEG output is a little off.

We want the count to happen on a rising clock edge and the SSEG output to be a little more in sync with the AN output. We will need to change some of the testbench code.



18. Close ISim by clicking the close button in the upper right corner.
19. Click the Yes button to exit the simulator and the No button when prompted to save the simulation.

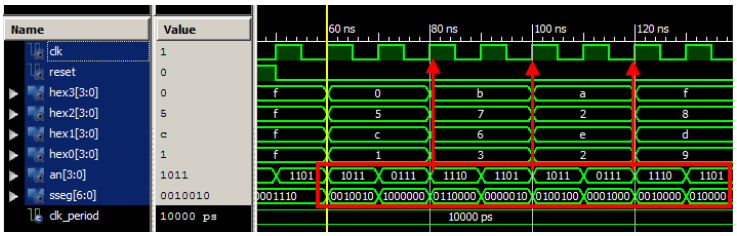
20. Find the clock process in the testbench. It should be around line 53.

21. Change the code so that the clock starts with 1, as shown to the right.

```
53    -- Clock process definitions
54    CLK_proc : process
55    begin
56    -- change the start of the clock to 1 in order
57    -- to drive the count to happen on a rising clock edge
58       clk <= '1';
59       wait for clk_period/2;
60       clk <= '0';
61       wait for clk_period/2;
62    end process;
```

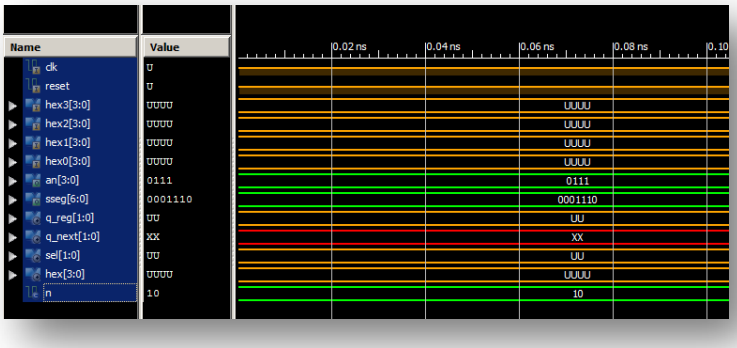
22. Save the file.

23. Run the simulation again. The waveforms should now show the count changing on the rising edge of the clock and the AN and SSEG outputs should be in sync.



NOTE: The clock is driving the AN output. Recall that the 7-segment cathodes can only display on one anode at a time. The SSEG output is driven by the HEX# specified by the AN output, i.e. if AN equals 1011, then HEX2's value is displayed by SSEG.

NOTE: If your inputs show up as red or undefined in the simulation, as shown below, you may have run the simulation with the design file selected instead of the testbench file. Close ISim and make sure you selected the correct testbench file and then run the simulation.



NOTE: It is not possible or necessary to run a test against the UCF file.

Once the testbench is working as expected, you can then compile and download the project to your FPGA board and observe it counting. If it doesn't count check the modules to ensure that you changed the clock settings back to the original values.

Lab Procedure 5: Test your Knowledge

1. Before creating a testbench, what should you do?

2. What additional test inputs could be used for the counter_TB?

3. Why should you name your testbench file with a "_TB" at the end of the file name?

4. What view must be displayed in order to Simulate Behavioral Model?

5. To properly run the simulation, how should you determine the proper time selection for the clock period in your testbench?

6. What is the difference between "00" and x"00"?

7. What does it mean when the waveform simulation displays “UUUU”?

8. From a practical point of view, is it possible to exhaustively test every VHDL synthesizable file?