Xilinx VHDL Test Bench Tutorial

Billy Hnath (bhnath@wpi.edu)
Department of Electrical and Computer Engineering
Worcester Polytechnic Institute
Revision 2.0

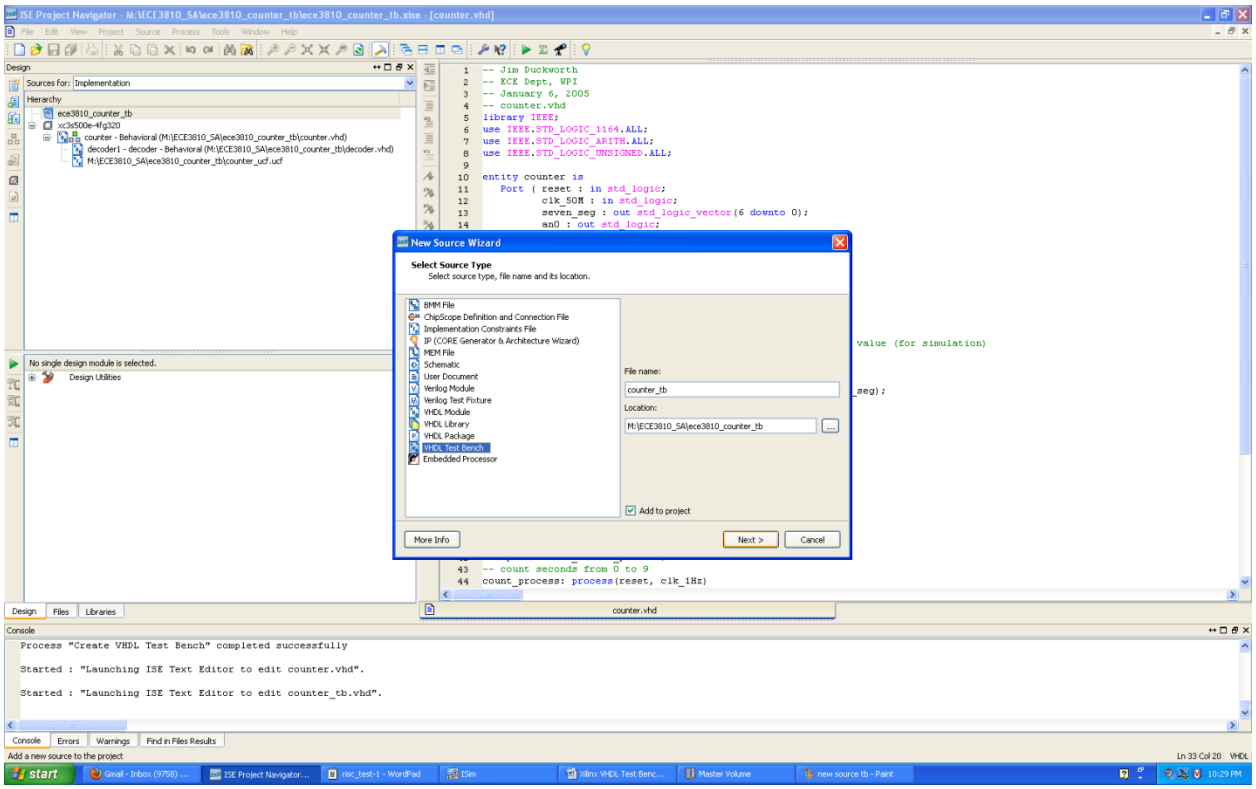
Introduction

This tutorial will guide you through the process of creating a test bench for your VHDL designs, which will aid you in debugging your design before or in addition going to the FPGA for execution. For the sake of simplicity, we will revisit the counter tutorial available at Professor Duckworth's website: http://ece.wpi.edu/~rjduck/Nexys2%20ISE%2010_1%20Counter%20Tutorial.pdf. We will recreate the sample counter and decoder and then create a VHDL test bench for the counter to show what it looks like in the new Xilinx software. This process will be helpful to you in the later labs in the course, as you will be able to see what your signals are doing as well as allow you to check to see if the values coming out are correct or not. Without further ado – let us continue to the counter example.

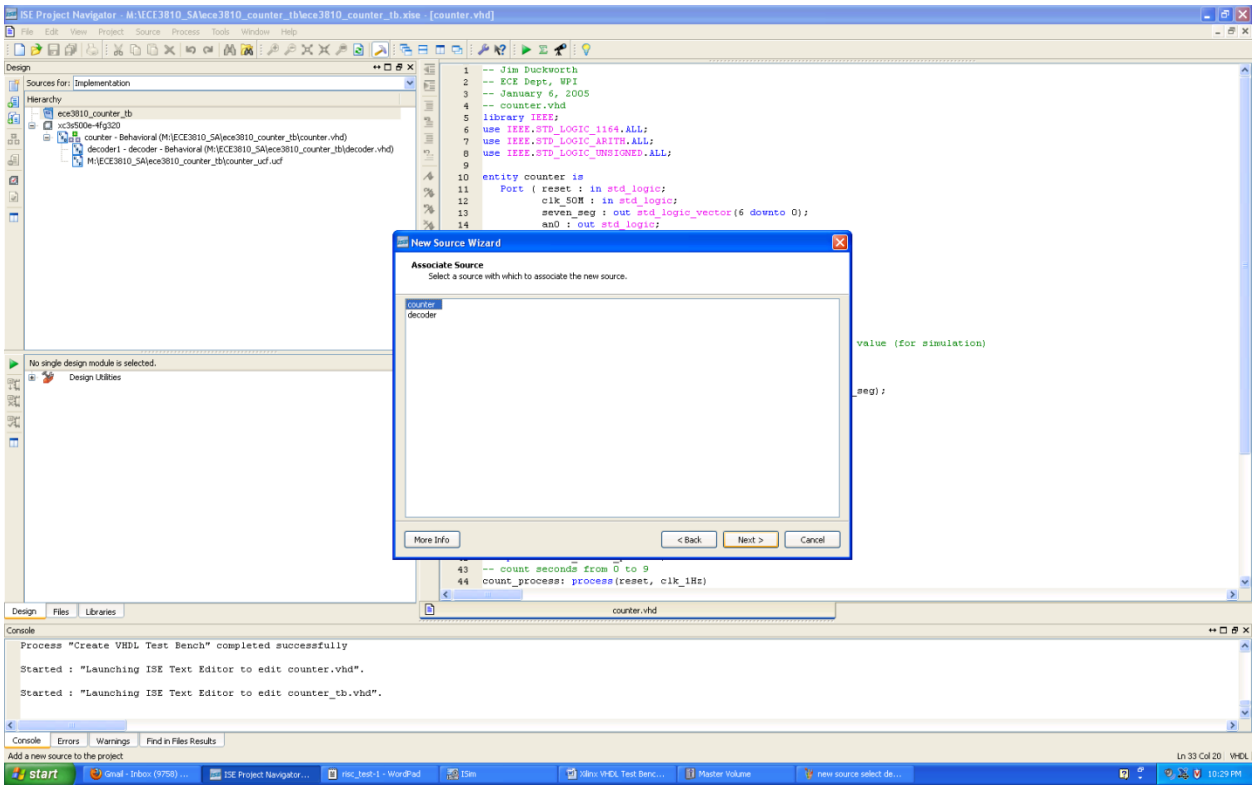
The Simple 4-bit Counter Revisited

If you have already created a project for the counter tutorial, feel free to use that as a base for this tutorial. If you have not, please follow the tutorial at the link given in the introduction to complete the counter design.

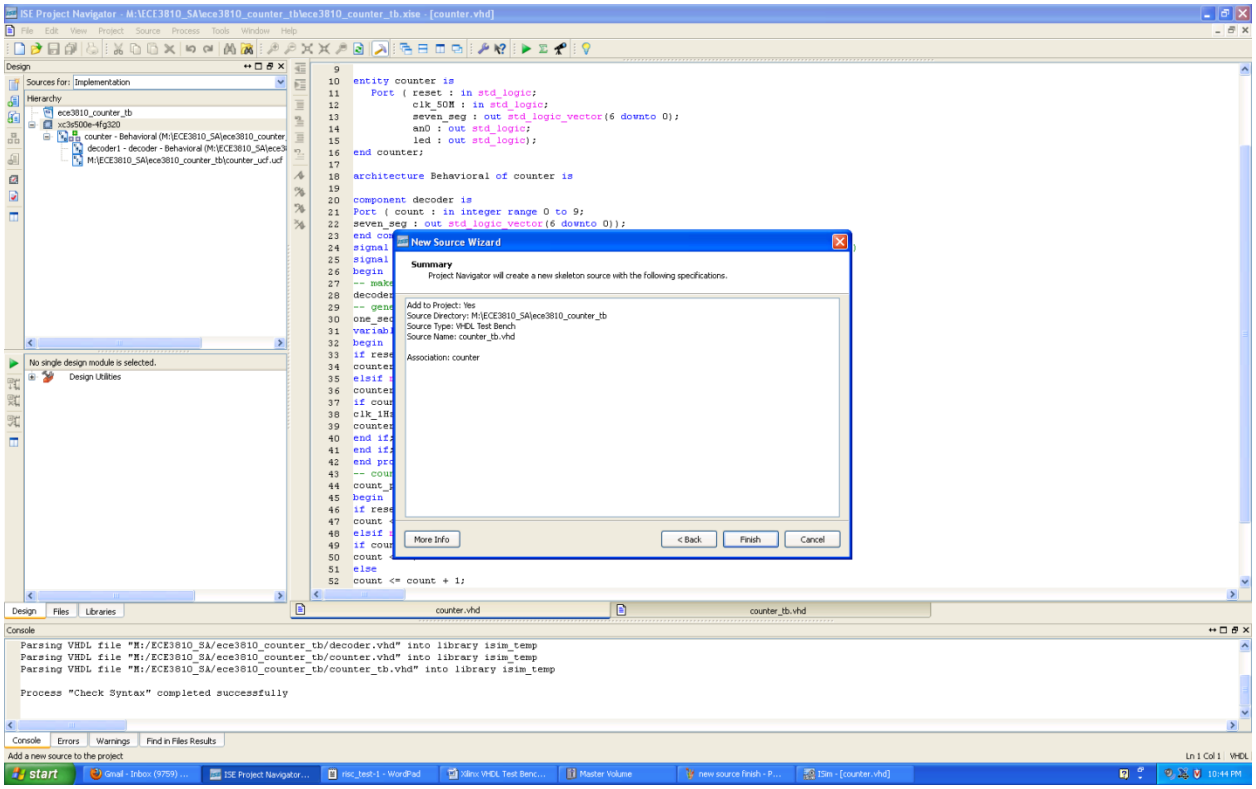
With the counter and decoder designs created and synthesized, select from the Xilinx tool bar **Project** -> **New Source**. Name the source something that signifies it's a test bench, such as "**counter_tb**" in this example. Click "**Next**".



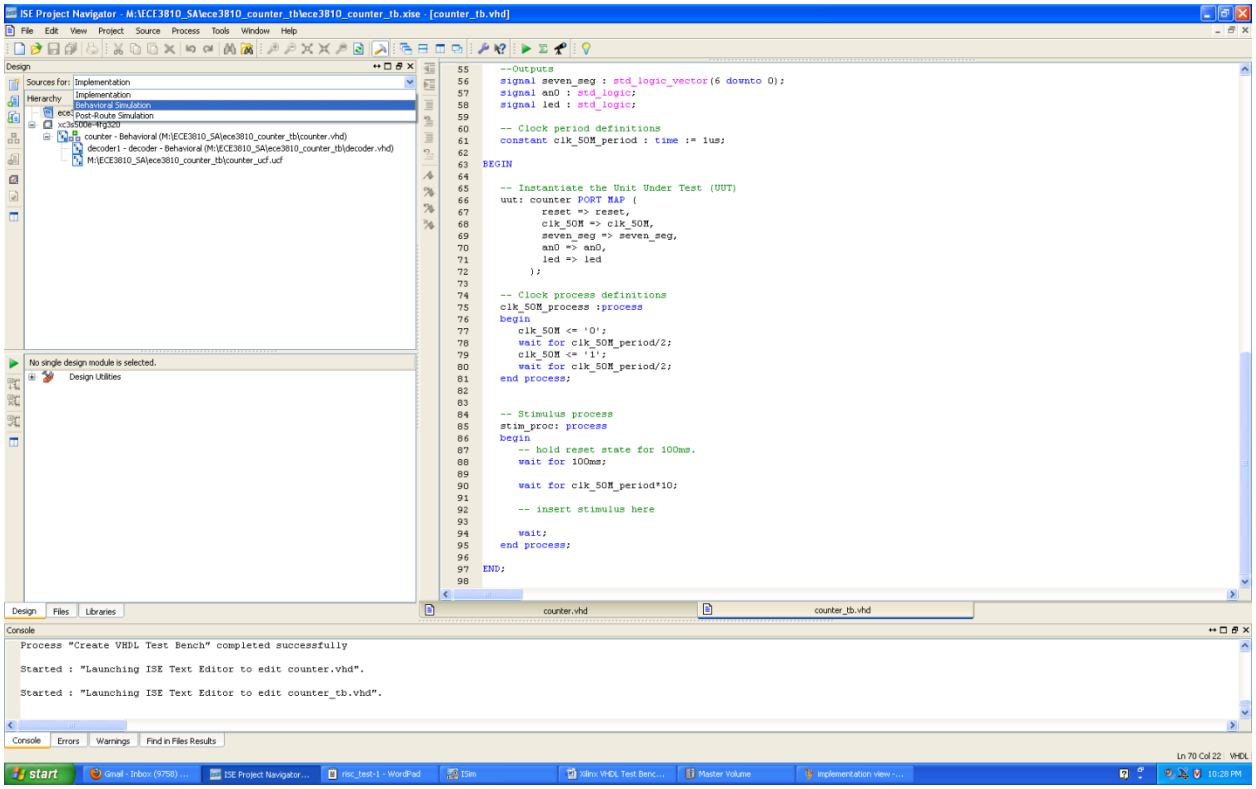
From here, select which design you want to create a test bench for. Here we will select the counter design as it is at the top-level for this project. Keep in mind that you may make test benches for individual components of your designs as well as the top-level design which ties it all together (analogous to testing individual functions versus your main in a programming language). Click "**Next**".



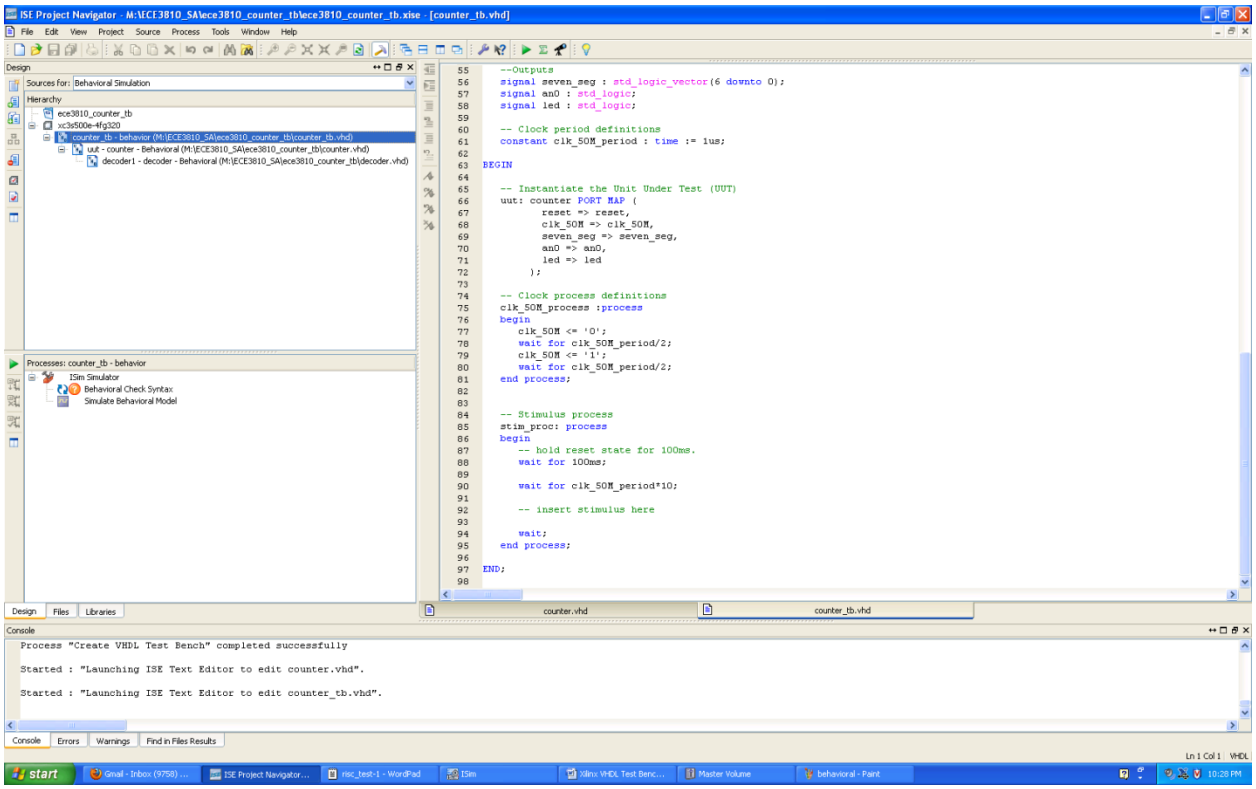
Select "**Finish**".



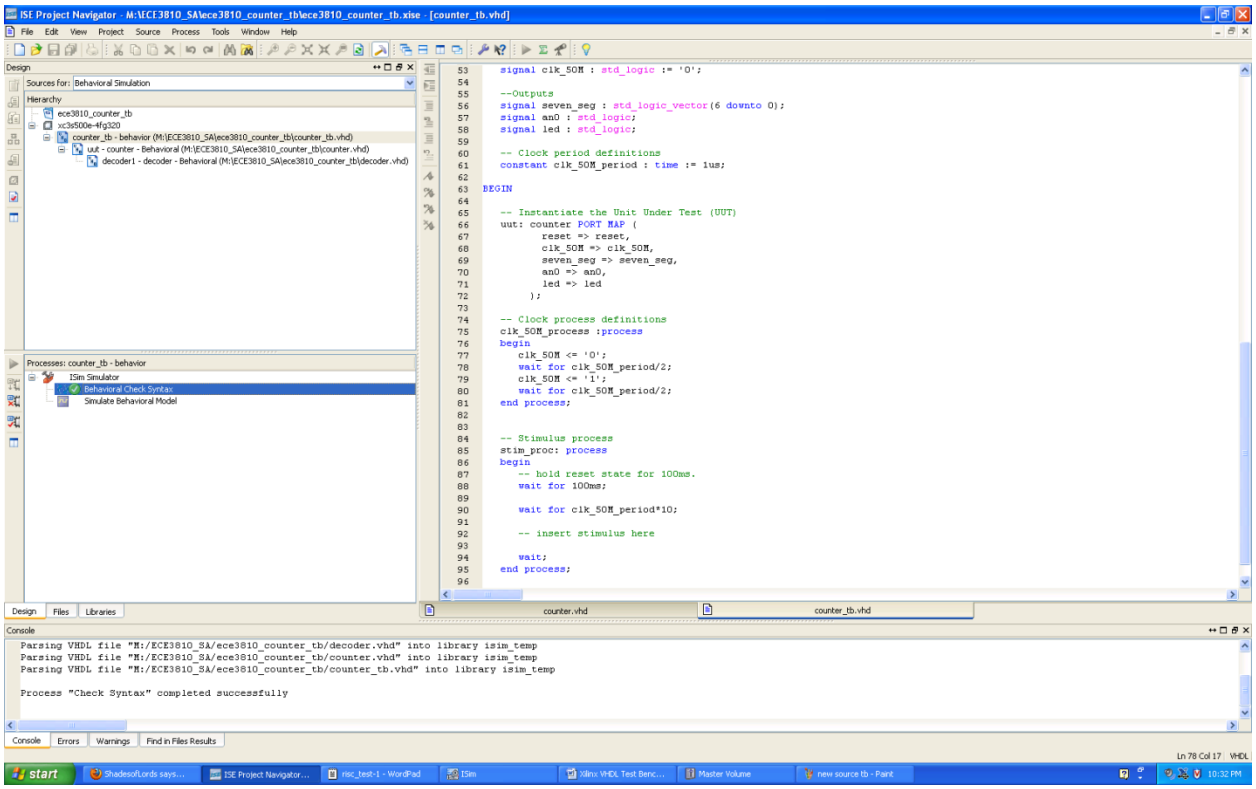
At this point, you will now be able to view your test bench file that Xilinx has generated in the main programming window. You will notice that it does however, not show up in the side panel where your other VHDL files are. This is because Xilinx has separate views for Implementation and Test files. To view your Test Bench file on the panel, navigate to the upper left hand corner of the Xilinx ISE and click the arrow where "**Implementation**" is currently being displayed.



Once you have clicked the arrow, you will see two additional views for the panel. The one you will want to select is titled "**Behavioral Simulation**". Click this option and the testing panel will be available for you to view and select for testing the VHDL designs.



You will now see new options available for you to select, much like when you are implementing your designs on the FPGA (Synthesize, Map, Program, etc). Double Clicking "**Behavioral Check Syntax**" will check to see if the test bench syntax is correct and "**Simulate Behavioral Model**" will execute your test bench in the third party program ISim. Double click "**Behavioral Check Syntax**" to make sure that the default test bench is okay.



VHDL Test Bench – Dissected

Now is an excellent time to go over the parts of the VHDL test bench. A test bench in VHDL consists of same two main parts of a normal VHDL design; an entity and architecture. The entity is left blank because we are simply supplying inputs and observing the outputs to the design in test. The architecture of the test bench will consist of the design we are testing as a component, internal signals for input and output, a port map of the component for the UUT (unit under test), a process to run the clock and finally a stimulus process, which will be responsible for running the tests you write to test the design. Refer to the test bench file that Xilinx generates and go over the components to make sure you have an understanding of what is going on.

Now that we have gone over what the different portions of the generated VHDL test bench file do, let's add in some stimulus code to see how it all works together. First, we need to modify the clock that Xilinx has generated for us to work well with the counter design.

First, edit the constant for the clock period definition. We would like this to match the 50 MHz clock that is coming into the test bench to make the timing diagrams match up nicely. To do this, we simply take 1/50 MHz and convert to nanosecond, which comes out to be 20 nanoseconds.

```
-- Clock period definitions
constant clk_50M_period : time := 20ns;
```

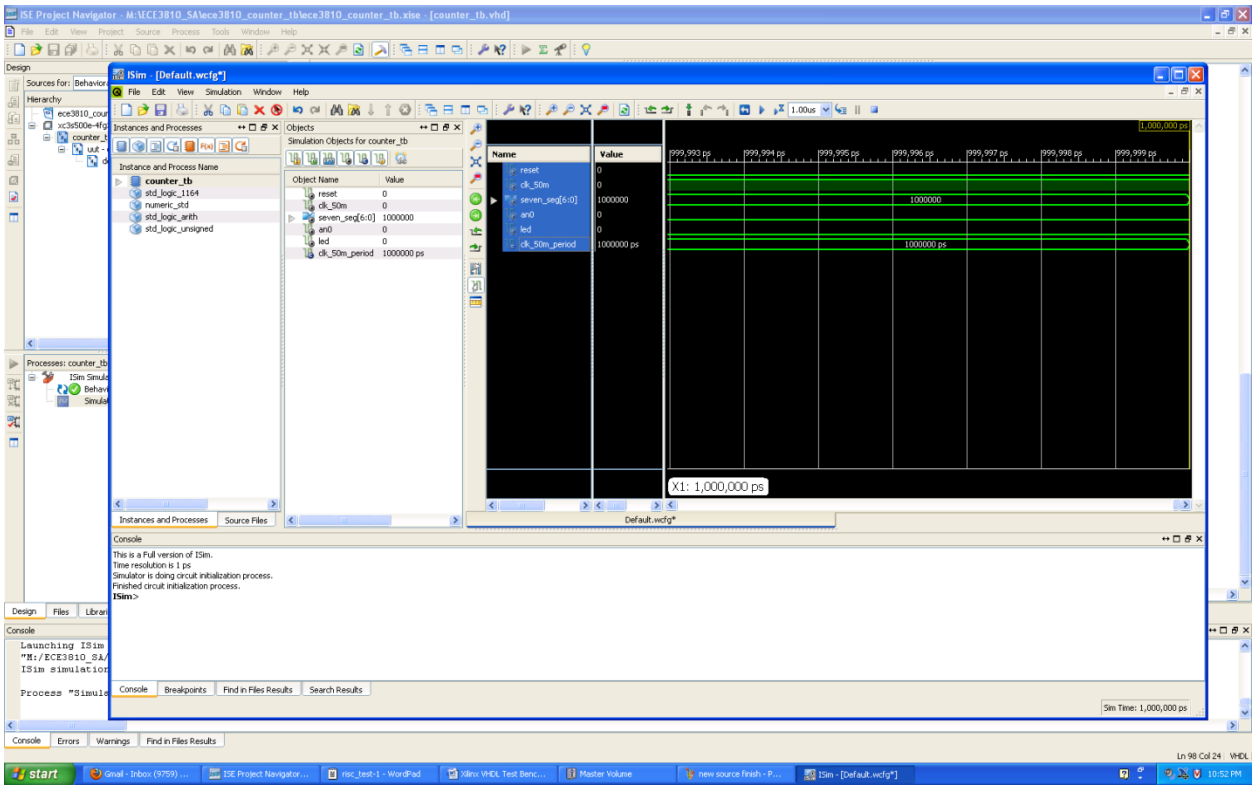
Next, we need to modify the clock process statement that Xilinx generated which divides the clock by two, creating a 25MHz clock. To change this, we just need to remove the dividers in the statement to look like the following.

```
-- Clock process definitions
clk_50M_process :process
begin
                clk_50M <= '0';
                wait for clk_50M_period;
                clk_50M <= '1';
                wait for clk_50M_period;
end process;
```

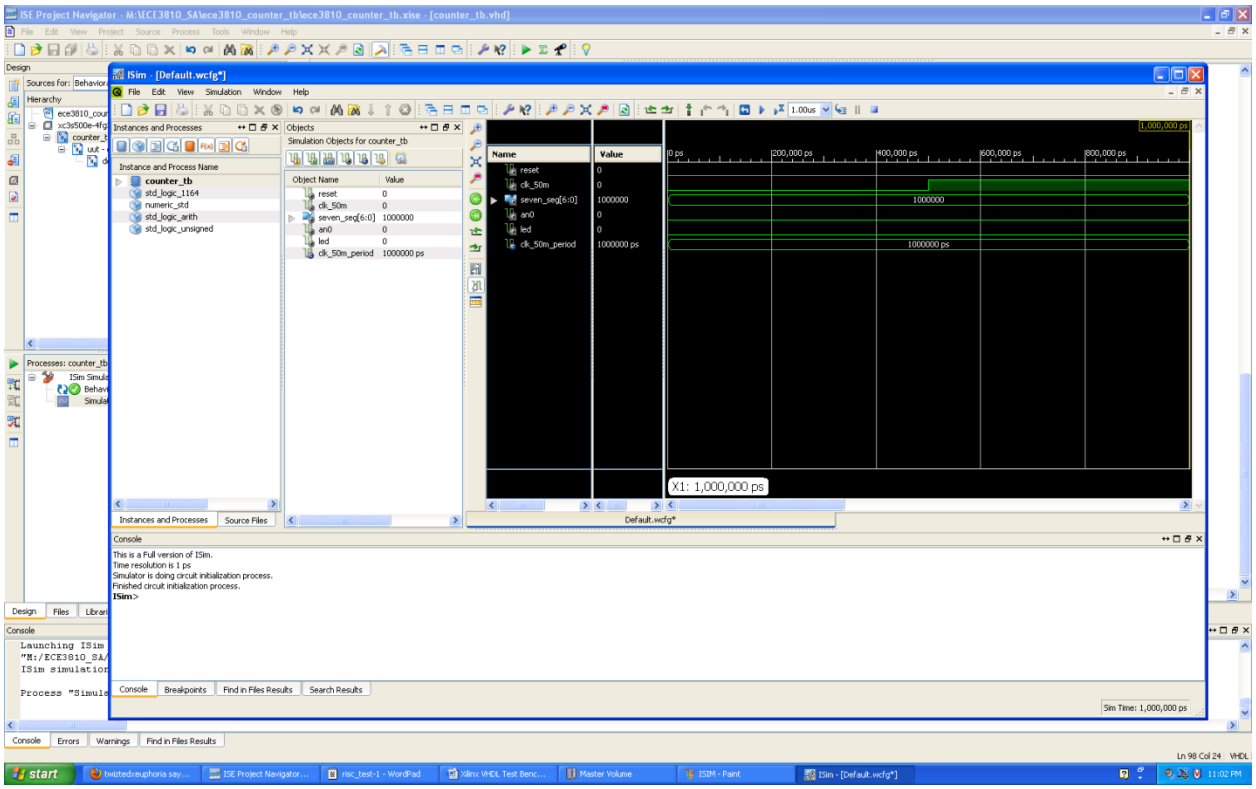
Finally, replace the stimulus process with the following code segment, which will serve as the first example for running a VHDL test bench simulation.

```
-- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100ms.
    wait for 100ms;

                --Sample way of setting inputs - reset used as a redundant example.
                reset <= '1';
                wait for 10ns;
                reset <= '0';
                wait for 10ns;

    wait;
  end process;
```

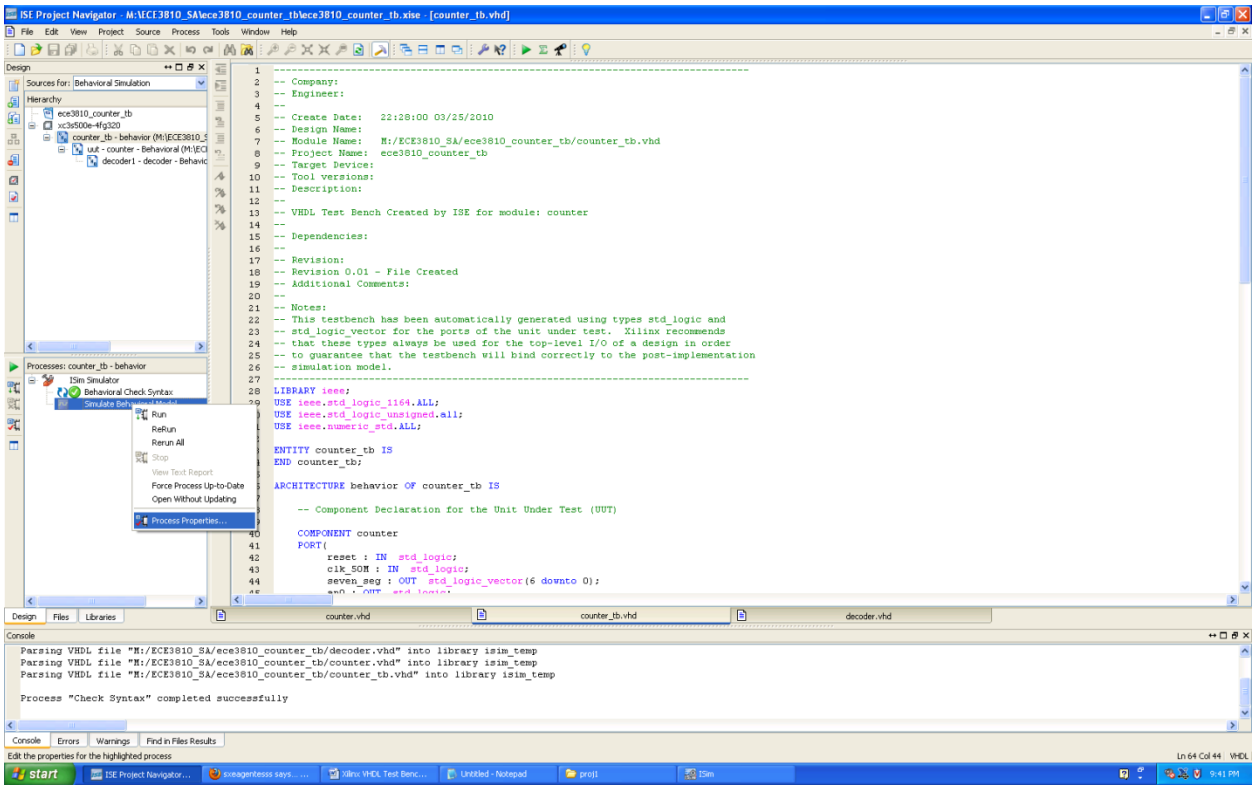
The following code will cycle the reset button and perform a very simple initial test of the design for simulation. To execute the test, double click on "**Simulate Behavioral Model**" and the ISim software will open with your test bench loaded.



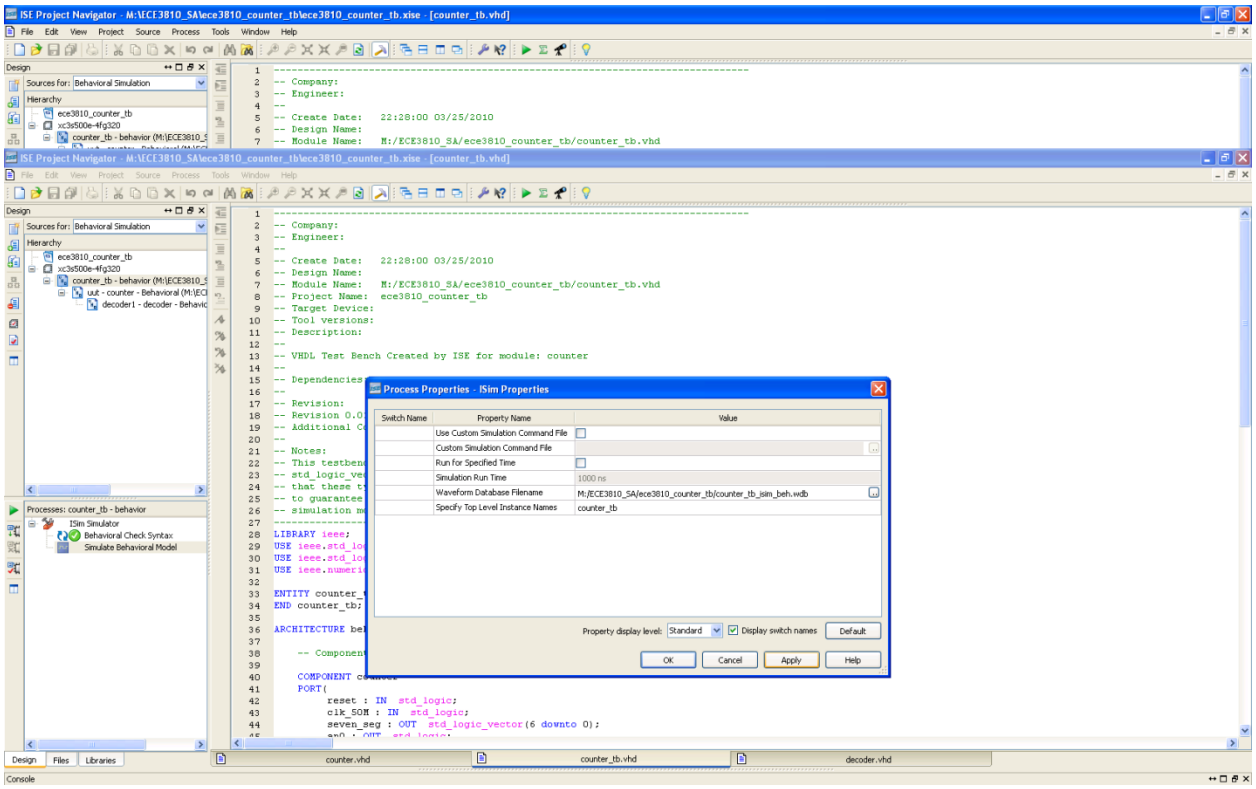
You will now be presented with the ISim program where you will be able to simulate your designs and check for errors. You will notice that the program is very much like a software debugger, in the way which you can step through your VHDL designs and check the states of signals, as well as set the simulation to run for specific amounts of time and view output in the bottom windows. The simulator will open with your simulation executed. You will notice that the resolution for the simulations is set to 1 picosecond. This is due to some Xilinx primitives requiring a 1 picosecond resolution to guarantee that your designs are processed correctly. To get a better view of the simulation, navigate over to **View** -> **Zoom** -> **Full View** (**F6**). This will allow you to get a better view of what your simulation is doing and you may also zoom in and out to get to finer levels of observation.



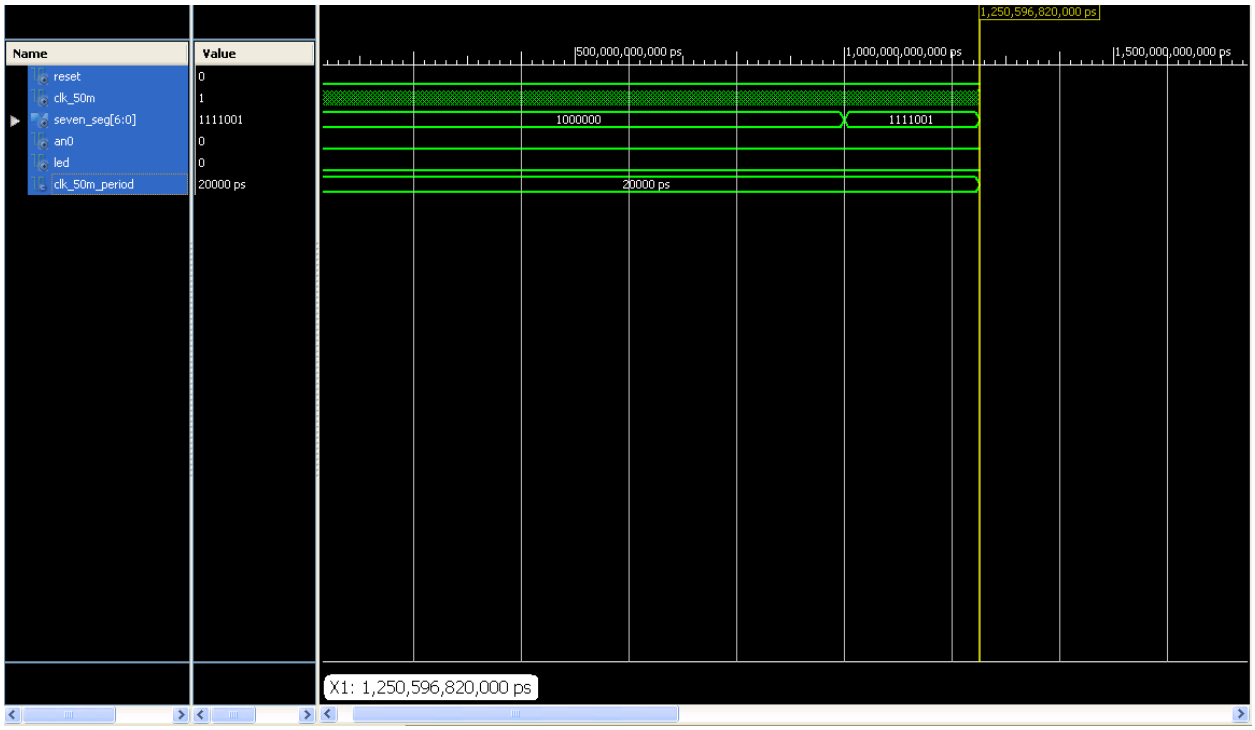
As you can see, when ISim opened it ran your simulation for you at what the parameters were set in the test bench. To avoid this happening and to be able to start with a clean timing diagram at the start of the program, go back to Xilinx and right-click the "**Simulate Behavioral Model**" and select "**Process Properties…**".



In this window you can modify some various settings of the ISE simulator. For our purposes, uncheck the box marked "**Run for Specified Time**" and hit Apply. Note that if in the future you want to run your simulation for a specified amount of time prior to starting the application that this is the place to do that.



With this setting changed, re-execute the "**Simulate Behavioral Model**" line in the tools section for behavioral simulation. Note that you should close ISim before rerunning this command as it will try to open a new instance of the program and fail. Now you should be able to see a blank timing diagram because nothing has been run yet. Pressing the 'Run' button on the top of the screen (marked with the right pointed triangle) will execute your design until you hit pause (also along the buttons near the run button). You may wish to play with the zoom features of ISim to be able to get a better view of what is happening with the simulation. Once you play with the zoom for a little while, the timing diagram should come out to something like below.



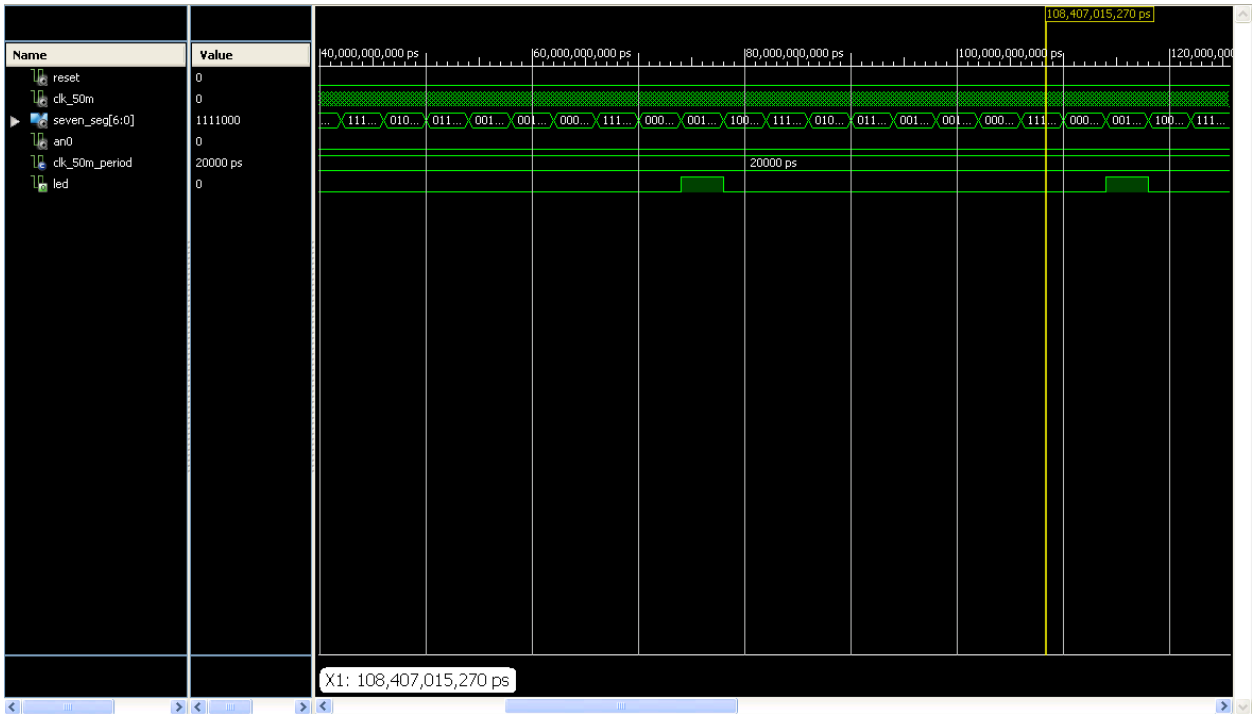
You may also specify points in time for the simulation to run for in the text box located in the same region as the run button. The button to the left of the box will execute the simulation for the time you have specified in the box. Note that it seems that ISim has some problems dealing with measurements such as '**1s**' for one second, so instead try to keep your timings in something such as '**1000ms**'.

Upping the Clock Speed and Internal Signal Watching

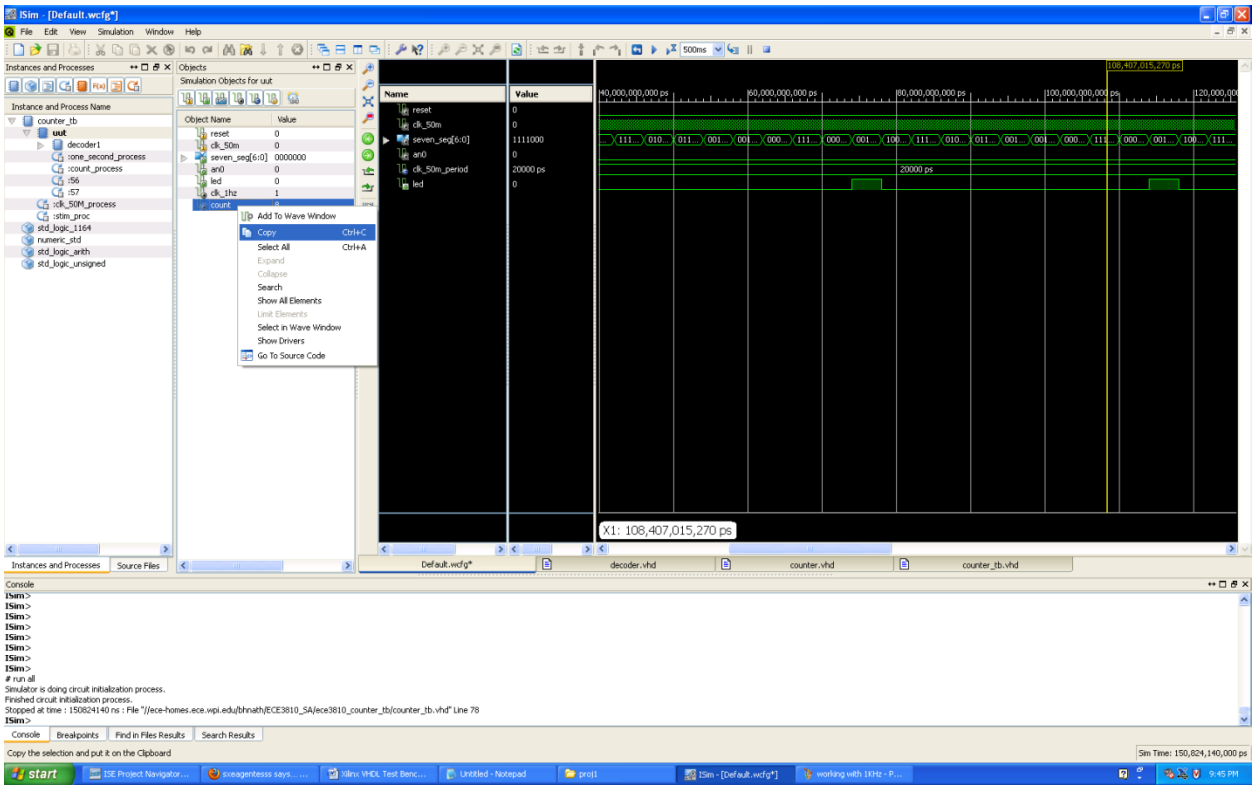
You may notice that the simulation runs rather slow, and you can guess that it's because the counter is running at 1Hz from the clock dividing process located in the counter design. This makes checking to see if your simulation works for this example rather painful, so let's go ahead and up the process from 1Hz to 1KHz. Change the **one_second_process** in the counter.vhd file as so.

```
-- generate a 1 second clock from the 50MHz oscillator
  one_second_process: process(reset, clk_50M)
    variable counter_50M : integer range 0 to 50_000;
  begin
    if reset = '1' then
      counter_50M := 0;
    elsif rising_edge(clk_50M) then
      counter_50M := counter_50M + 1;
      if counter_50M = 50_000 then
        clk_1Hz <= NOT clk_1Hz;
        counter_50M := 0;
      end if;
    end if;
  end process one_second_process;
```

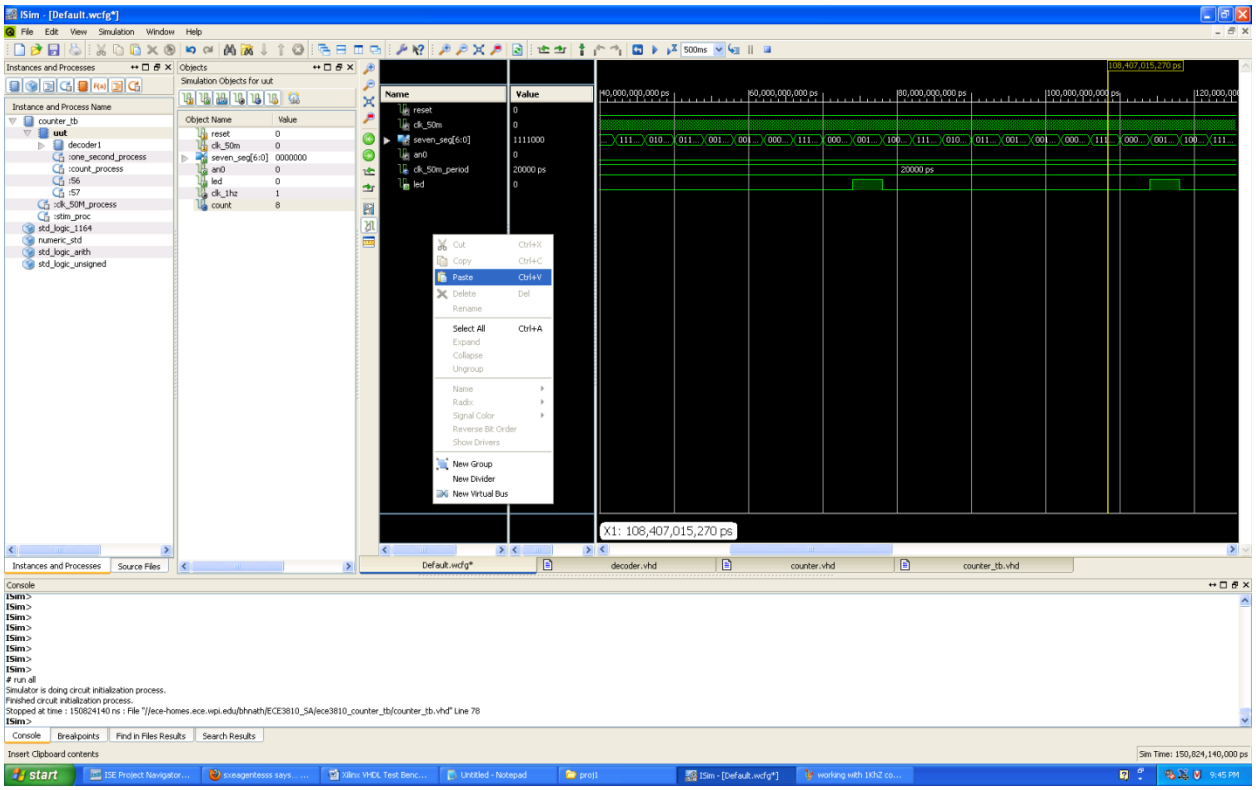
Close the current ISim session and re-run the behavioral simulation as you have done before and when you run the simulation you will now see that it runs much faster. After playing around with the zoom settings a bit, your timing diagram should look close to that displayed below.



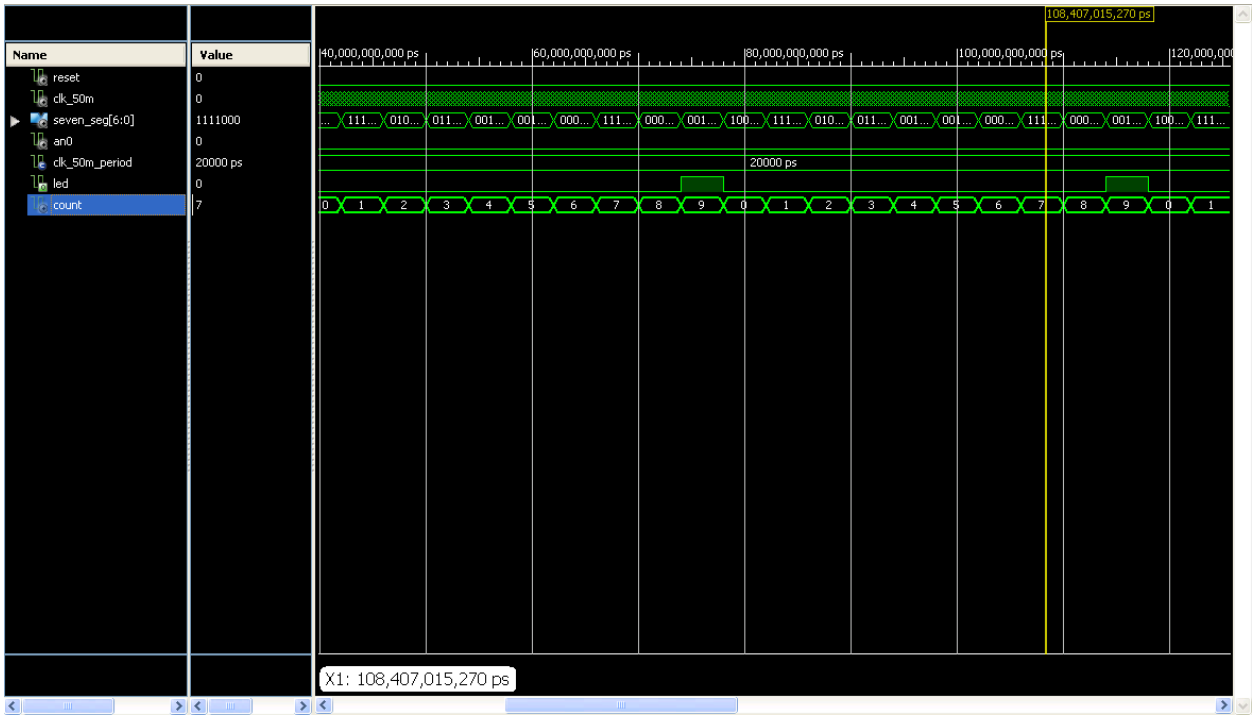
While running simulations on your designs, you will probably find the need to be able to view internal signals and their values as the simulation runs on via the timing diagram. To get this working, draw your attention to the left most panel when ISim opens, called "**Instances and Processes**". Here you can view all of the running instances of designs and processes within each design in your current test bench. To get the items from the decoder to show up in your timing diagram, click the down arrow on "**counter_tb**", followed by the down arrow on "**uut**". This will show you all of the processes running inside of the design as well as internal signals used within. To display an internal signal in the timing diagram, highlight one such as "**count**" and right-click, followed by **Copy**.



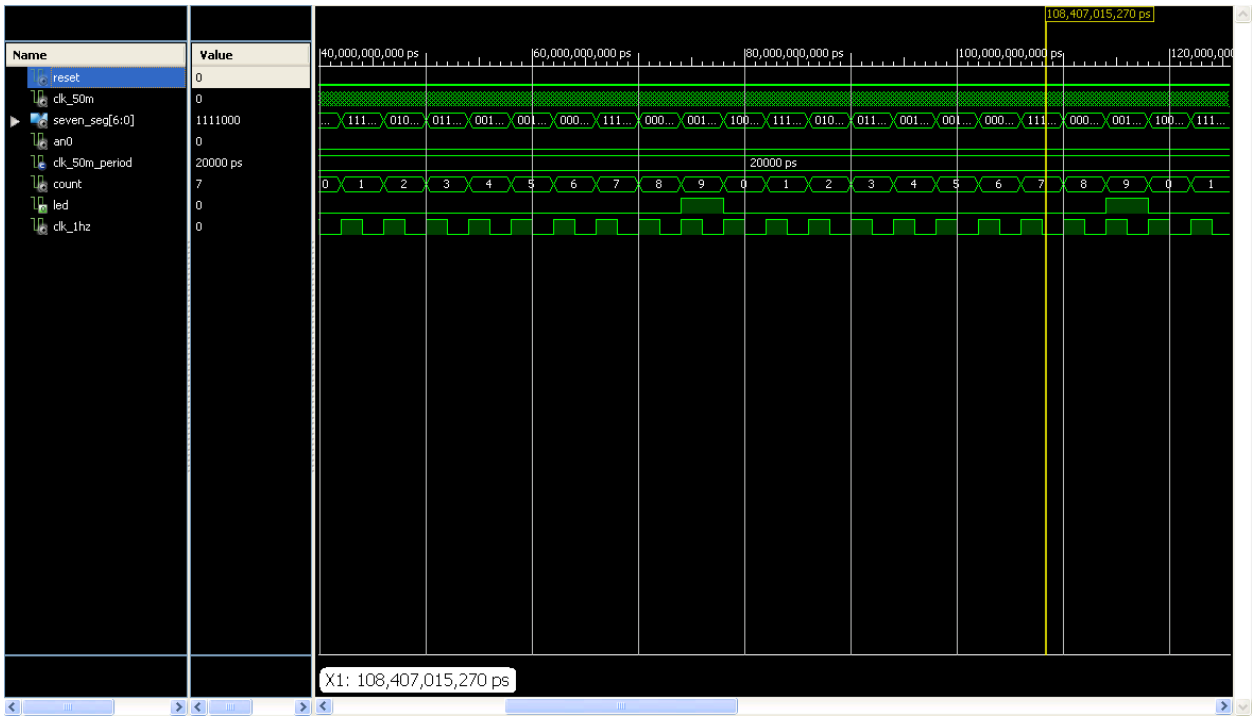
Next, right-click under the "**Name**" field in the timing diagram window under all the current signals being displayed and select Paste.



You should now see the internal counter signal displayed in the timing diagram window under all of the previous top-level signals. After some adjustments to the zoom settings it should look more or less like the following image.

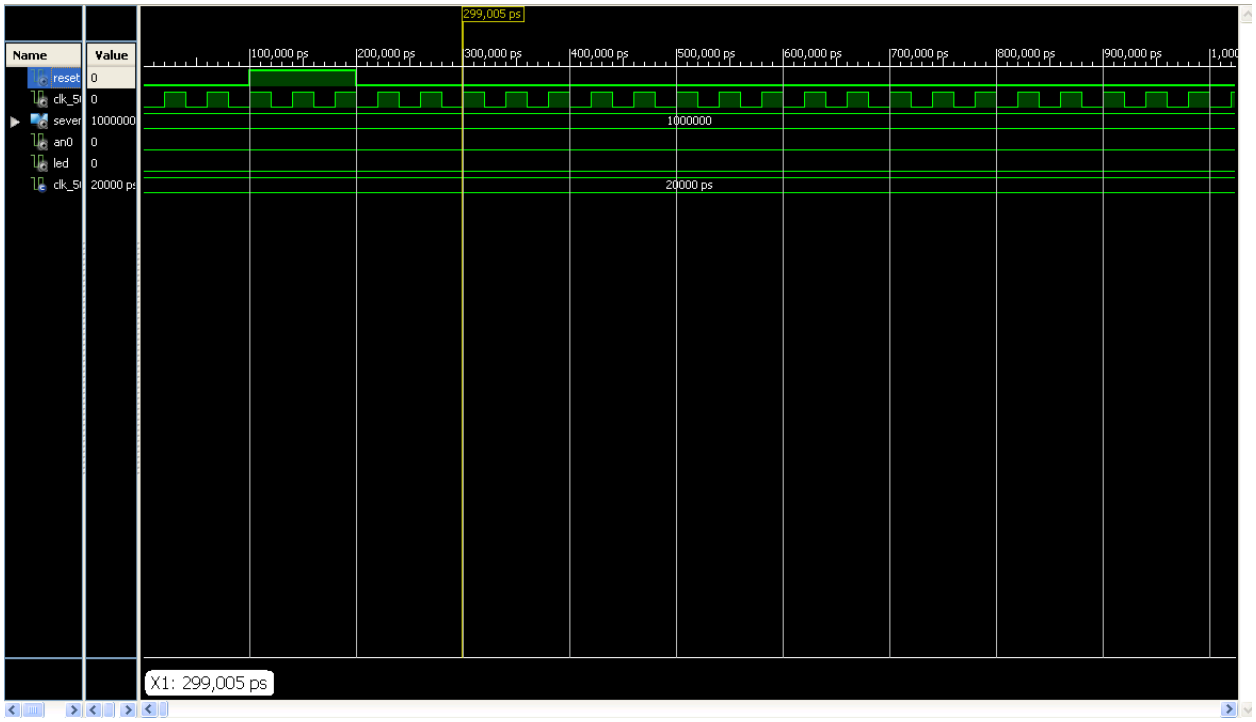


You may repeat the steps above to view the 1 KHz clock process and in the future any other signals you wish to the view. The final result should look approximately the same as below.



Finally, replace the stimulus of the test bench with the following code segment. This will display a reset signal as well as a sample assert statement for checking certain conditions in your test benches. This test is meant to fail to show you the kind of debug output from the console to expect when a statement fails.

```
-- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100ns.
    wait for 100ns;

                --Sample way of setting inputs - reset used as a redundant example.
                reset <= '1';
                wait for 100ns;
                reset <= '0';
                wait for 100ns;

                --Sample assert statement; check if the seven segment display should
                --be displaying '1'. If it is not, report the error to the console.
                assert seven_seg = "1111001"
                        report "Output incorrect after reset"
                        severity note;

    wait;
  end process;
```



You now have a working counter test bench simulation with the ability to change clock rates as well as add internal signals and check assert statements for a fully functional VHDL debugger. For more information on ISim as well as more tips on simulation for VHDL and Xilinx, please refer to the references of this document.

References

1. VHDL Counter Tutorial, http://ece.wpi.edu/~rjduck/Nexys2%20ISE%2010_1%20Counter%20Tutorial.pdf

2. ISE Simulator – In-depth Tutorial, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ug682.pdf