Columbia University
Department of Electrical Engineering
EECS E4340. Tools tutorial.
VHDL design entry and simulation.

There are a few additional set-ups that you will need in your UNIX environment to make sure you are set up in the CISE (Columbia Integrated Systems Lab) design environment.

1. Please add the line:

    . /usr/tools/init/kshrc

    to your .profile file. This will ensure that you pick up the default environment set up for the class.

2. Please create a file called .cdsinit. In this file, please add the line:

    load("/usr/tools/cds/setup/cdsinit");

3. Please copy /u2/ee4340/public/cds.lib to cds.lib in your home directory.

4. Please copy the entire directory

    /usr/cad/cds/tools/dfII/samples/tutorials/composer/tutorial

    to your home directory:

    cp -pr /usr/cad/cds/tools/dfII/samples/tutorials/composer/tutorial .

Type cdsdoc at the UNIX prompt to bring up the Cadence on-line documentation. Choose the ” Composer” option from the main page. Then select ”Virtuoso Schematic Composer Tutorial.”

Please do the Cadence Composer tutorial, Chapters 2 - 5. Please use icfb& to bring up the Cadence software. It will take several hours to work through the tutorial, but it is worth it!. It is very important that you are familiar with Composer, so please don’t cut any corners here.
In the remainder of this tutorial, you will learn how to perform VHDL simulation from the Composer interface to Cadence's NC-VHDL simulator.

In this tutorial, we will enter and simulate a simple logic design in Cadence.

First, create a library to hold your test design. To do this from the CIW, choose **File → New → Library**. You will get a “New Library” form. In this form, enter the name “tutorial2” into the Name field. On the technology file options, choose “Don’t need a techfile.” Since we are only doing front-end logic design, we do not need techfile information (which contains detailed on a specific integrated-circuit technology). The Design Manager option should remain “No DM.”

In this tutorial, we create a very simple binary counter using a combination of schematics and text VHDL. First create a symbol called “register16.” From the symbol editor, choose **Design → Create Cellview → From Pin List**. In the resulting “Cellview from Pin List” form, enter the input pins \texttt{reg\_in<0:15>} and \texttt{clock} and the output pin \texttt{reg\_out<0:15>}. Choose View Name of “symbol” from the bottom cyclic field. Enter “register16” as the cell name. Hit ”OK.” “Yes” the hey-box indicating that it is Okay to overwrite the symbol. You can then Okay the “Symbol Generation” form. You may edit the register symbol is you want. Next choose **Edit → Properties → VHDL.** Under “Scalar Data Type,” enter \texttt{std\_ulogic}. Under “Vector Data Type,” enter \texttt{std\_ulogic\_vector}. Under “Use Clause,” enter \texttt{ieee\_std\_logic\_1164\_all} and \texttt{ieee\_std\_logic\_arith\_all}. Under “User Comment,” you can add “16-bit register.” You can now create the VHDL entity from the symbol. Choose **Design → Create CellView → From CellView.** “OK” the form. The entity that is automatically created now appears in Emacs for you to edit if necessary (actually, this should not be necessary). Emacs is running with some special templates to help the VHDL novice. For more details in this VHDL language-sensitive editor, please see the on-line documentation in “cdsdoc.”

Now we need to create the architecture for this 16-bit register. Create a cellview for this cell called “behavior.” “behavior” will be the name used for the VHDL architecture. Any name can be used here. On the “Create New File” form, specify “VHDL-Editor” as the “Tool” field. “OK” the form, which brings up Emacs again. We now need to enter the VHDL architecture for this component:
architecture behavior of register16 is begin
  SYNCH: process(clock)
  begin
    if (clock = '1' and not(clock'stable)) then
      reg_out(0 to 15) <= reg_in(0 to 15);
    end if;
  end process;
end behavior;

Save the design and exit Emacs.

Now, let’s create a multiplexer, cell name “multiplexer” with pins in0<0:15>, in1<0:15>, and select_in and output mux_out<0:15>. Create the symbol and entity as before. Don’t forget to add the correct VHDL properties on the symbol. Now create an architecture in the “behavior” view for this component:

architecture behavior of multiplexer is begin
  process(select_in, in0, in1)
  begin
    if (select_in = '0') then
      mux_out <= in0;
    elsif (select_in = '1') then
      mux_out <= in1;
    else
      mux_out <= "XXXXXXXXXXXXXXXXXX";
    end if;
  end process;
end behavior;

The final component we need before assembling the top-level schematic is an incremerter, cell name “incremerter” with pins incr_in<0:15> and output incr_out<0:15>. Create the symbol and entity as before. The “behavior” view for this component is as follows:

architecture behavior of incremerter is begin
incr_out <= to_stdulogic(unsigned(incr_in) + '1');
end behavior;

Finally, we need to make the top-level schematic, cell name “top_level.” Instantiate a “multiplexer”, a “register16”, and a “incrementer.” There are three inputs of the top-level schematic, clock, start<0:15>, and load_start. Wire up the schematic so that the multiplexer, feeds the register, which feeds the incrementer. The output of the incrementer then feeds back around to go into the in0 input of the multiplexer. The in1 input of the multiplexer gets the input from start and the select of the multiplexer gets the input from load_start. Check and save your schematic when you have finished the wiring. Choose Edit → Properties → VHDL and add the VHDL properties in a manner similar to how it was done for the symbols.

We know need to set up the VHDL environment, netlist, and simulate the design. From the CIW, choose Tools → VHDL Toolbox. Select Toolbox-Environment to be NC-VHDL.

We will now create a simple testbench to test this design. Enter the library and cell name (“tutorial2” and “top_level”) in the “Top of hierarchy” field. Choose Commands → Create Test Bench. Leave the “Stimulus File Name” field blank and “OK” the form. The cell name “test” with the architecture “stimulus” is created and displayed in Emacs as a template for your testbench. First modify the signal declarations to include initial conditions:

SIGNAL load_start : std_ulogic := '1';
SIGNAL clock : std_ulogic := '0';
SIGNAL start : std_ulogic_vector(0 TO 15) := "0000000000000000";

We will enter a very simple testbench into the architecture body after the “dut” instantiation:

clock <= not(clock) after 10 ns;
load_start <= '0' after 15 ns;

You may also need to add the component binding:

FOR ALL: top_level USE ENTITY tutorial2.top_level(schematic);
Now the “Top of hierarchy” should be “tutorial2”, “test”, “stimulus.”

Now, we need to set up for netlisting. From the VHDL Toolbox, choose **Setup → Hierarchy**. Select **Switch List** and select “The VHDL Architecture” for the location of the component bindings. “OK” the form.

Now choose **Commands → Check Hierarchy** to netlist and syntax check your design, creating a simulatable VHDL model for NC-VHDL. If there are any errors, they are reported and Emacs is automatically brought up with the error highlighted. Correct any syntax errors and repeat **Commands → Check Hierarchy** until the design cleanly analyzes and elaborates.

Now, we are going to simulate the design. Choose **Commands → Simulate**. The NC-VHDL simulator window appears. Choose **Tools → Navigator**, opening a “Navigator” window. Clicking on the top level icon displays the hierarchy under the top level. There should be three components listed: **dut**, **$PROCESS_000**, and **$PROCESS_001**. Highlight **$PROCESS_000** and with the right mouse button select **Set Debug Scope**. In the source window, you’ll see the source displayed for testbench and the line

\[
clock \leq \text{not}(\text{clock}) \text{ after } 10 \text{ ns};
\]

highlighted. The designation **$PROCESS_000** corresponds to this line of concurrent VHDL (you can think of each state of concurrent VHDL as representing a process). Double-clicking on **dut** causes the hierarchy under this level to expand.

Now we are going to set up for a waveform trace and run the simulator. Now from the “Navigator” window, select **dut** and with the right mouse button select **Wave Trace**. This should add all the signals at the top level of the design into the waveform tool (Signalscan), which should automatically come up. Now let’s run the simulator for 1000 ns. From the VHDL simulation window, choose **Set → Breakpoint → Time**. Enter **1000 ns** in the Time field. “OK” the form. Now click the “Run” button on the VHDL Simulator window. Leapfrog should run for 1000 ns and then stop with the waveform recorded into the open Signalscan window.