VHDL
Introduction to
tools

TSTE12

Datorteknik

Kent Palmkvist, Thomas Johansson

Version 0.8

30. Aug. 2020

1 Introduction

This handbook is your guide to the VHDL simulation tool set used in the course. You will have use for this to complete the homework and in future use of the simulator during the project. This document explains a summary of methods useful for the laboratory work and throughout the project. References to other documents are given for more details.

1.1 General

The purpose of this exercise is to introduce tools for implementation of designs using VHDL. The tools introduced in the tutorial are also used in the project part of this course. The tools used are Modelsim from Mentor Graphics for VHDL simulation, and HDL Designer, which also come from Mentor Graphics, for design and architectural/graphical design approach.

It is assumed that the reader has a basic knowledge to VHDL. This tutorial is basically focused on the tools, not the exercise of writing code.

This document is divided into several small parts. In small parts we supply the basics for several vital parts in the design path. In this chapter we introduce the example designs. Chapter 2 gives a brief introduction to VHDL simulation with Modelsim as a stand alone application. The third chapter explains the basics of graphical design entry for VHDL. In some additional parts we also broaden the views for test and procedures for handling this inside Modelsim.

The tutorial starts in chapter 2, “Modelsim Introduction“. Before that, we need to define the environment and the design examples used throughout the tutorial.

1.2 Example Design

The example used in these tutorials is a 4-bit ripple carry adder with registers at the outputs, see Figure 1. We will use this circuit to demonstrate different kinds of coding styles and ways to increasingly refine a model by performing hierarchical design. As we continuously refine the model, we also demonstrate the tools including simulator and compiler.

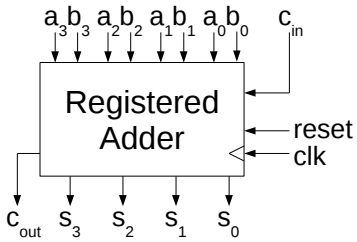

Figure 1: 4-bit registered ripple carry adder example design behavioral model

We want to model a 4-bit adder with carry-in and carry-out. The output from the adder should be registered, i.e. latched into a D-flipflop. This is done because we want to hold the result for one extra clock cycle after the summation has taken place. We start with the first model, Figure 1, “4-bit registered ripple carry adder example design behavioral model“. The corresponding VHDL code of this design is shown in the text box below. The code is behavioral, and does not clearly indicate how this adder will be implemented.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY example_design IS
  PORT (
    a     : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    b     : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    cin   : IN  STD_LOGIC;
    clk   : IN  STD_LOGIC;
    reset : IN  STD_LOGIC;
    cout  : OUT STD_LOGIC;
    s     : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END example_design;


LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ARCHITECTURE behav OF example_design IS
BEGIN
  PROCESS(clk, reset)
    VARIABLE tmp : UNSIGNED(4 DOWNTO 0);
    VARIABLE tmp_cin : UNSIGNED(1 DOWNTO 0);
  BEGIN
    IF (reset = '1') THEN
      tmp := (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
      tmp_cin := '0' & cin;
      tmp := UNSIGNED('0'&a)+UNSIGNED('0'&b)+tmp_cin;
    END IF;
    s    <= std_logic_vector(tmp(3 DOWNTO 0));
    cout <= tmp(4);
  END PROCESS;
END behav;
```

The behavioral model is then refined into a structural model where simpler functions are combined in an hierarchical fasion to implement the function, as shown in Figure 2.

The design is now described as a system of one flip-flop, one 4-bit register, and a 4-bit adder that peforms the same function as the behavior model from Figure 1.

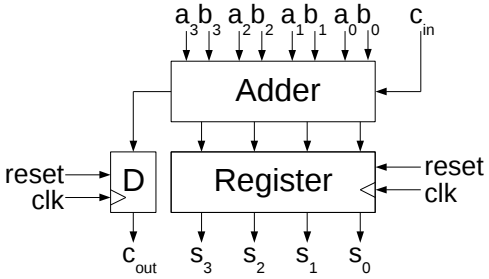

Figure 2: 4-bit registered adder, first hierarchical model

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ARCHITECTURE structural OF example_design IS

  COMPONENT adder
    PORT (
      cin   : IN  STD_LOGIC;
      a     : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
      b     : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
      carry : OUT STD_LOGIC;
      sum   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
  END COMPONENT;

  COMPONENT DFF
    PORT (
      d     : IN  STD_LOGIC;
      clk   : IN  STD_LOGIC;
      reset : IN  STD_LOGIC;
      q     : OUT STD_LOGIC);
  END COMPONENT;

  COMPONENT dff_4bits
    PORT (
      clk     : IN  STD_LOGIC;
      reset   : IN  STD_LOGIC;
      d_4bits : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
      q_4bits : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
  END COMPONENT;

  SIGNAL sum_pipe  : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL cout_pipe : STD_LOGIC;

BEGIN
  xadd : adder PORT MAP (cin, a, b, cout_pipe, sum_pipe);
  xDFF : DFF PORT MAP (cout_pipe, clk, reset, cout);
  xreg : dff_4bits PORT MAP (clk, reset, sum_pipe, s);
END structural;
```

The adder subsystem has an interface as shown in Figure 3. It adds the two 4-bit input vectors a and b together with the carry input bit cin. The result consists of the 4-bit sum and an output carry bit. The corresponding VHDL code is shown in the box below.

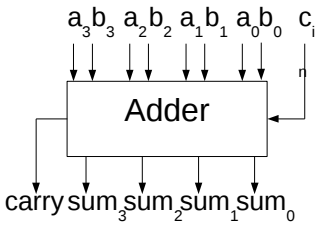

Figure 3: 4-bit addition subcircuit

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY adder IS
  PORT (
    cin   : IN  STD_LOGIC;
    a     : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    b     : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    carry : OUT STD_LOGIC;
    sum   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END adder;

ARCHITECTURE behav OF adder IS
  SIGNAL tmp : UNSIGNED(4 DOWNTO 0);
  SIGNAL tmp_cin : STD_LOGIC_VECTOR(1 DOWNTO 0)
BEGIN
  tmp_cin <= '0' & cin;
  tmp   <= UNSIGNED('0' & a) + UNSIGNED('0' & b) + UNSIGNED(tmp_cin);
  sum   <= STD_LOGIC_VECTOR(tmp(3 DOWNTO 0));
  carry <= tmp(4);
END behav;
```

A basic sequential component is the D-flip-flop. Here a positive edge trigged D- flip-flop with an asynchronous reset is shown. We normally do not draw the clk and reset signal in schematics. They are implicitly defined in the figures to each piece of code. The corresponding VHDL code is shown in the text box below

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY DFF IS

  PORT (
    d     : IN  STD_LOGIC;
    clk   : IN  STD_LOGIC;
    reset : IN  STD_LOGIC;
    q     : OUT STD_LOGIC);
END DFF;

ARCHITECTURE behav OF DFF IS
BEGIN
  PROCESS(clk, reset)
  BEGIN
    IF (reset = '1') THEN
      q <= '0';
    ELSIF rising_edge(clk) THEN
      q <= d;
    END IF;
  END PROCESS;
END behav;
```

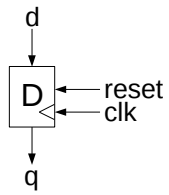

Figure 4: Basic D flipflop symbol

We also here introduce a 4-bit wide register based on the dff code, as shown below

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY dff_4bits IS
  PORT (
    clk     : IN  STD_LOGIC;
    reset   : IN  STD_LOGIC;
    d_4bits : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    q_4bits : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END dff_4bits;

ARCHITECTURE behav OF dff_4bits IS

BEGIN
  PROCESS(clk, reset)
  BEGIN
    IF (reset = '1') THEN
      q_4bits <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
      q_4bits <= d_4bits;
    END IF;
  END PROCESS;
END behav;
```

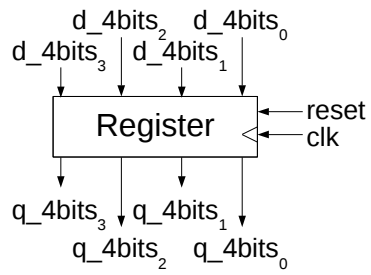

Figure 5: 4-bit register symbol

1.3 Further Reading

This chapter was just a short introduction to VHDL to help you through the exercises. For further studies of VHDL a wide range of material is available on the internet.

At the listed addresses you may find some very useful tips, hints and tricks that might improve your skills in VHDL.

Some useful VHDL links

http://www.vhdl-online.de/	Good reference material in VHDL. The site include links, VHDL reference and much more. Very useful.
http://www.altera.com	FPGA vendor.
http://www.xilinx.com	FPGA vendor.

2 Modelsim Introduction

This chapter describes how to use the Modelsim VHDL simulation environment. The goal of these exercises is to learn how to use the Modelsim simulation environment on your own to compile and simulate VHDL code.

The tutorial is formed as a sequence of numbered steps to follow. Each time a system shell command or simulator command is given it has a special format. First an explanation is written then the command sequence. The example below illustrates how to list all files in the current directory. The line starting with the number 1 is the explanation of what the command does, and "ls -al" is the unix command to type in the terminal window. The commands are always formatted as bold italics.

1. List all files in the current directory

   ls -al

First we describe how to set up a new environment. The next step will be to compile the demonstration files and simulate them.

2.1 Setting up the Tutorial

1. Open up a terminal
2. Load the course module to set up system environment

   module load courses/TSTE12
3. Start the special system shell, mentorskal

   mentorskal

NOTE! All tools must be started in the Mentor-skal window or using special scripts whos name start with TSTE12 to get the correct group permissions on all files.

4. If the labgroup number is unknown check this with

   id -a

   Find the entry for the group tste12lab, e.g. tste12lab99, which would mean group number 99. This is your group number.

   NOTE! If you are not a member of a laboratory group, please talk with the teacher.
5. Set your working directory to the labgroup home directory. Change 99 in the command to your group number

   cd /courses/TSTE12/labs/labgrp99

   NOTE! Do NOT work in your own home directory (e.g., abrli987, djikh999, winch945) unless you are working on the handin tasks. Use the directory '/courses/TSTE12/labs/labgrp<nn>' for tutorial and lab tasks.
6. Copy all necessary files from the course material directory into your directory. Please take notice of the dot in the command

   cp -R /courses/TSTE12/material/VHDL-tutorial .
7. Enter the tutorial directory with

   cd VHDL-tutorial

NOTE!! All the following text assumes that your current working directory is the `VHDL-tutorial` directory unless otherwise noted!!!!

2.2 Compiling and simulating the VHDL files

In VHDL we need a connection between the file systems reference and the logical library used inside VHDL. This is provided with library map file. Logical libraries provide a very easy way to make portable code. Only the library map file describe the physical connection to the file system. We here use the default logical library name work.

8. Create the compile library

   vlib work

9. Create a library map file

   vmap work ./work

You are now ready to compile vhdl code.

10. Compile the example design with the behavioral architecture by entering the command:

    vcom -work work src/example_design_entity.vhdl

    vcom -work work src/example_design_behav.vhdl

    We can specify all design units to compile in the same line. If no library name is specified it is assumed the name is "work", hence we may also write

    vcom src/example_design_entity.vhdl src/example_design_behav.vhdl

11. Start the simulator

    vsim -lib work

12. Look at the simulator start window in Figure 6.

13. Locate the library work. Expand this by clicking on the "+" to the left of the name. Select the unit example_design, right-click and select "Simulate without optimization".

14. The window now change to look like Figure 7, "Modelsim simulation window".

15. Bring up the simulation windows necessary for this simulation by entering menu View, and check that the windows Objects, Locals, and Files are checked.

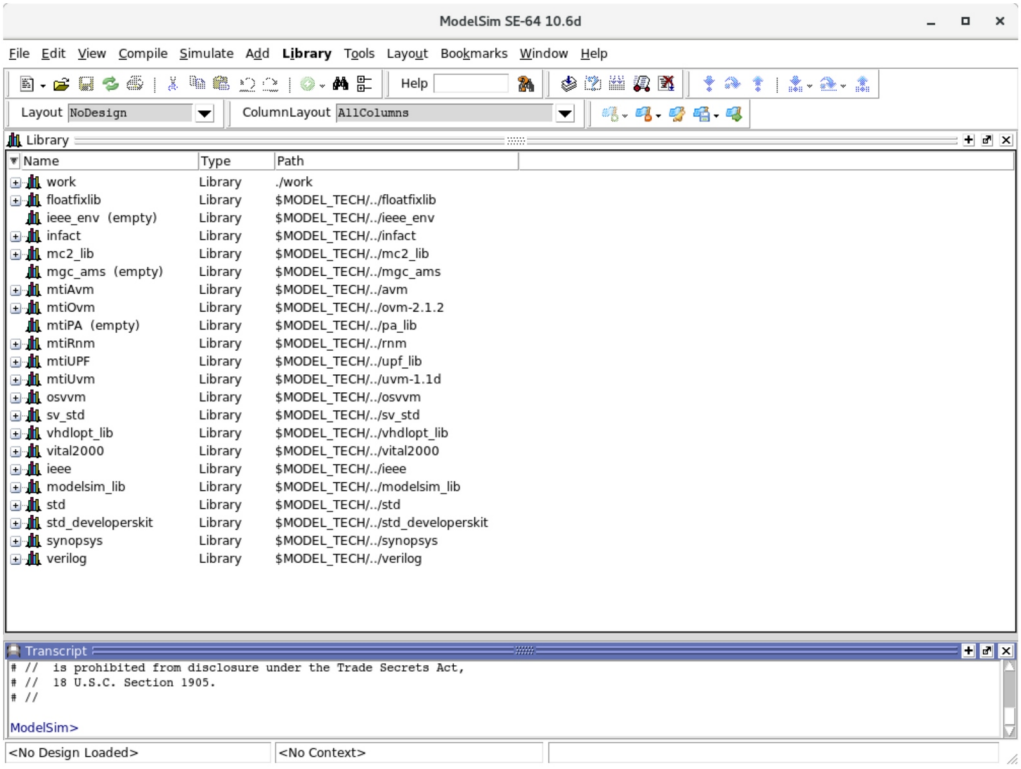

Figure 6 Modelsim simulator startup window

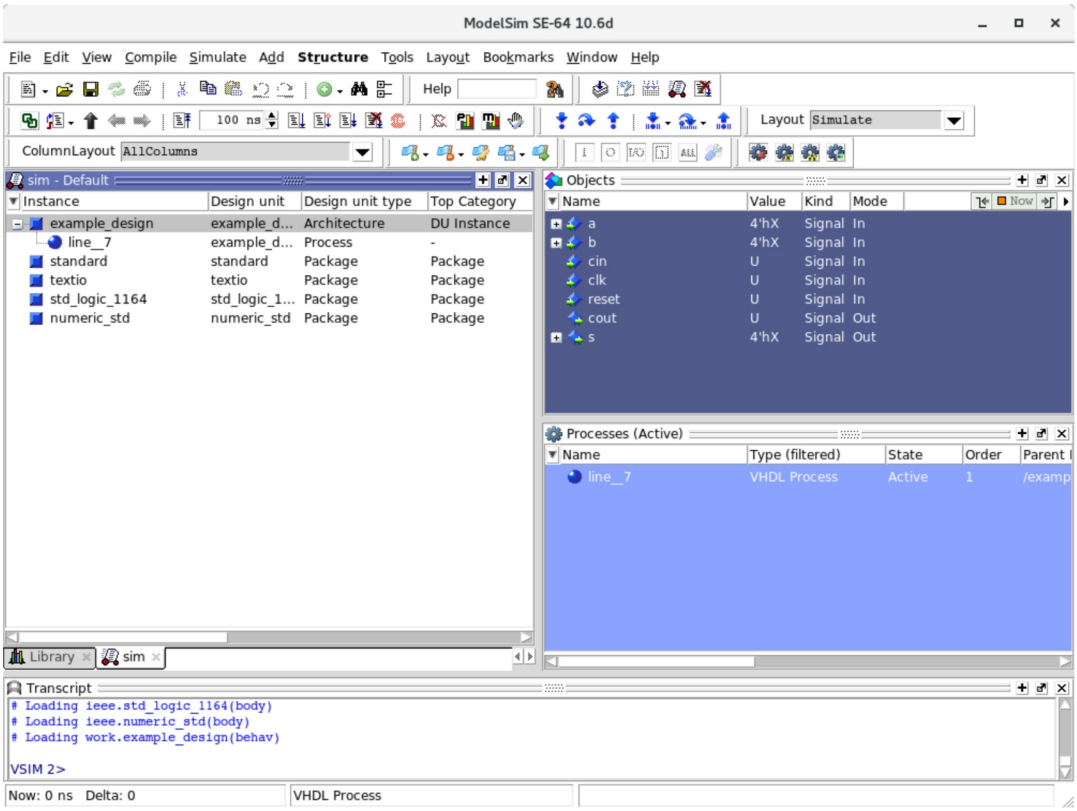

Figure 7 Modelsim simulation window

16. Select the Files tab in the subwindow to the left. View the files included in the current simulation by clicking on the "+" to the left of "sim". Select example_design_behav.vhdl, right-click and select "view source".
17. To add top-level signals to the Wave window, click in the Objects window, then select the menu Add. Select "To wave" and "Signals in region" from the menu.
18. The values shown for the a, b and s vectors are using a different representation, e.g., 4'h3. This corresponds to a 4 element vector that has the value 3.
19. Apply stimulus to the clock input by moving the pointer to the Objects window. Right click on the clk signal, and select "Modify" and then "Apply Clock...". Enter Duty=50, period=100 ns, first edge=falling and then press Ok.

    In the transcipt subwindow should now have appeared

    force -freeze sim:/example_design/clk 0 0, 1 {50 ns} -r {100 ns}

    It is interpreted by the simulator to mean:

    Force clk to the value 0 at the current time

    Then force clk to 1 at 50 ns after the current time

    Repeat this cycle every 100 ns

You will see the effects of this force command as soon as you tell the simulator to run. First we have to apply stimulus to the other inputs.

20. Apply stimulus to the reset input by moving the pointer to the Objects window. Select the reset signal. Enter the menu Objects and select Force. Change value to 1 and press Ok.
21. In the main simulator window select Simulate/ Run / Run 100. This causes the simulation to run and then stop after 100 ns.
22. Run another 100 ns and inspect the wave forms in the wave window.
23. Change reset to the value 0 and simulate another 200 ns.

24. Set a few values of a, b, cin on your own and simulate to verify the functionality of the design. Use force to apply new signal values on the signals a, b, cin.

Reading binary values can be difficult. You will now change the radix in which the signal value is displayed.

25. Add five copies of the signal s to the wave window. Do this by moving the pointer to the objects window. Select signal s. Select menu Add / To wave / "Selected Signals". Repeat this five times.
26. Move the pointer to the wave window and select one of the s signals.
27. Select menu Wave / Format / Radix / **unsigned**. Repeat this on the other sum signals and apply radix **binary**, **decimal**, **hexadecimal** and **octal**.
28. Run for 200 ns, Change the value on the signal a, b and cin and simulate again.
29. To repeatedly press run 100 ns is time consuming. Change the default runtime by moving the pointer to the main window. Press the left mouse button over the field with 100 in the left half of the toolbar buttons. Change the value to 200 and press the icon to the right of the value field. One can also enter the command "**run**" in the transcript window.

The Locals window is still empty. This is because variables are temporary and may only be observed during simulation halt inside a process. To obtain this we will insert a breakpoint in the VHDL code. Next, you will set a breakpoint in the process on line 11.

30. Move the pointer to the example_design_behav.vhdl subwindow. If necessary use the vertical scrollbar, scroll until line 11 is visible. Click at or left of line number 11 to set the breakpoint. You should see a red dot to the right of to the line number when the breakpoint is set. This breakpoint can be toggled on and off by clicking on it.
31. Simulate again.
32. The simulator will hit the breakpoint, as shown by a blue arrow in the Source window and by a message in the ModelSim transcript window. Also note that the parameters and variables within the function are displayed in the Locals window.
33. Click **Step into** to single-step through the simulation. Step into is the button containing a blue arrow pointing down onto a blue dot. Keep the mouse pointer over the buttons for a few seconds and a help text appear to guide which button to press.

    Notice that the values change in the Locals window. You can keep clicking Step into if you wish.
34. In main window select menu Simulate / Run / Continue
35. Notice the simulation continues until the next breakpoint.
36. Remove the breakpoint.

Next is a demonstration of cursors to probe for values and measure time intervals. We also experiment with scrolling and zooming.

37. In the Wave window, click on the green plus sign under the signal names. Placing the curson on top of this button should give the help text “insert cursor”.
38. A yellow vertical line appears. Press left mouse button over the vertical line and move the mouse pointer, and notice how the wave signal values change in the table on the left.
39. Change input values on a, b and cin for about ten values and simulate 2000 ns for each combination.
40. The wave window will not be able to show all signals values due to long simulation time. Use Wave / Zoom / Zoom Full to adjust this. In the same manner it is possible to zoom in and out.
41. An easier way to achieve zoom is the following. Place the mouse pointer in the wave window. Press and hold (middle) mouse wheel. Make a stroke 45 degrees up left. This is the same as zoom full / view all.
42. Explore the other strokes as well. Zoom out is a keystroke 45 degrees up right. Zoom in is any key stroke below the horizontal line, ie keystroke 45 degrees down left or right.

Next is a demonstration how to trace signal transitions in the wave window.

43. Select a signal by pressing the signal name in the wave window. Make sure the name is high-lighted.
44. Locate the transition toolbar buttons. They are blue arrows pointing left and right towards a green signal transition.
45. Press the arrow point towards right and notice how the cursor change position. This way one may trace changes in signals very easily.

Measure times in the wave window

46. Add an extra cursor in the wave window.
47. Move one of the cursor and watch the time change on the line combining the two cursors.
    This way we have a very easy way to measure time between events. This knowledge will be useful later in the course.

How to stop a running simulation

48. Locate the break button. It is the third button to the left of the default simulation time.
49. Type "**run 1000000000**" in the transcript window.
50. Look at the lower left corner of the main window. Notice how the simulation time is increasing.
51. Wait for a few seconds. Then press the break button. The simulation has now stopped at current simulation time.

End the simulator and the currently loaded simulation.

52. Quit the simulator by using menu File/Quit.
    To quit without the dialogue box and without saving data. Enter in the transcript window.
    quit -force
    This command exits the simulator without saving data.
53. In the shell, view the transcript file
    cat transcript
    All lines the transcript file not starting with a # is a valid modelsim command. This may be useful to create macro files to ease simulation executions. If the transcript file is edited and saved under a suitable name it might be executed from inside modelsim. We will soon demonstrate this with a prepared macro file.

2.3 Simulating the hierarchical model

The first model of the example design has now been verified. If we developed our model, the next step would be to refine the behavioural model into a structural model. This model is already prepared as we described in 1.2, “Example Design” . The next step is to compile, simulate and verify the structural model. This structural version of example_design contains an adder and registers as instantiated components.

54. View the code pieces in the Mentorskal window with
    cat src/dff.vhdl
    cat src/dff_4bits.vhdl
    cat src/adder.vhdl
    cat src/example_design_structural.vhdl
    This is a small design. As the complexity grows the control over the design structure may be hard to manage. In chapter 3 HDL Designer Introduction, we demonstrate how to manage the complex designs with a graphical design entry.
55. Compile the design units
    vcom src/dff.vhdl

vcom src/dff_4bits.vhdl

vcom src/adder.vhdl

vcom src/example_design_structural.vhdl

Take notice of that the design entity is already compiled. We now only add one more architecture to the design. Important is also the order of compilation.

56. Start the simulator

vsim

57. Load the the design example_design and simulate it with the architecture structural. The architecture to use can be selected by clicking on the plus to the left of example_design in the library window, and then right-click on the wanted architecture.

58. Select the menu Tools/Tcl/Execute macro.... In the dialog box open the dofiles directory and select the file example_design.do

Feel free to inspect the file later. It is a text file.

59. Press open and take notice of the actions.

60. Continue to verify the functionality.

61. Place the mouse pointer over the main window. Expand the design structure of example design in the sim window.

62. Double click on one of xdff, xadd, xreg to look at the corresponding source file. Selecting design units this way make it possible to access all sub blocks and place, e.g. breakpoints in the code.

63. You should now have basic skills good enough to complete all necessary simulations during the course. In later chapters there are a few tips and tricks to increase your simulation skills.

64. Quit the simulator.

3 HDL Designer Introduction

The purpose with this exercise is to introduce tools for implementation of designs using VHDL. The tools introduced in the tutorial is also used in the project part of this course. The tools we will use are Modelsim from Mentor Graphics for VHDL simulation and HDL Designer which also come from Mentor Graphics for design and architectural/graphical design approach.

3.1 Background

We first define the basics of the HDL designer environment and component concepts. The previously simulated example_design will be reimplemented using the design tools.

To demonstrate the possibility of hierarchical design we utilize the ability to create one or more views to each component. We first add the component symbol and a behavioral architecture. The design is then simulated and validated.

The next step is to add hierarchy in the example design and we introduce the block diagram and use this to refine the example design structure.

3.2 Top-down design

We here define a model as a step in the design flow of successive refinement. To each model a number of sub blocks exist, that are achieved as sufficiently defined blocks are obtained. The goals for each model are stated in advance.

In top-down design we develop the system stepwise by synthesizing and validating each level. The design levels are successively partitioned into sub blocks. This process for decomposition is repeated until sufficiently simple blocks are obtained, as illustrated in Figure 8.

The process of decomposition with successive refinement also guarantees that larger and more important issues are resolved before the detailed issues. By only making small modifications between successive models, the managing and validation of models becomes simpler. Furthermore, a correct design becomes more likely.

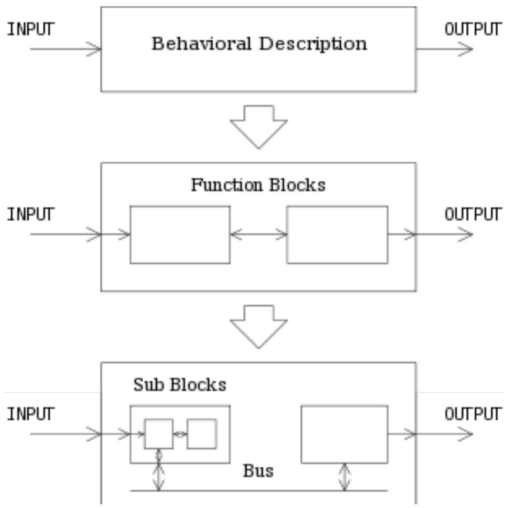

Figure 8 Top-down design flow

Since the only thing that changes is the structure of each level in the design process, the design will at all levels be possible to validate against the top model with the same validation methods. This framework for testing is described later in the course, using testbenches and macro files.

3.3 Getting started

The HDL Designer tool needs to be setup the first time you use it. This setup consists of configuring default simulation tools, synthesis tools, etc. The settings presented in 3.3.2 Configuring HDL Designer the first time is only necessary to perform once.

3.3.1 Starting the tool

It is assumed that the reader has a basic knowledge in VHDL. This tutorial is focused on the tools, not the exercise of writing code. The project module must be added, all system setup is included in the course module. You will then be able to start a special terminal shell called "mentorskal". In case you do not use this environment you have to load a special mentor module. Ask the course administration about the actual module to load.

The next 6 steps are vital before starting any tool used in this tutorial.

1. Open up a terminal
2. Load the course module to set up system environment

   module load courses/TSTE12

NOTE! All tools must be started in the Mentorskal window to get the correct group permissions on all files.

3. Start the special system shell, mentorskal

   mentorskal
4. If the lab group number is unknown check this with

   id -a

   Find the entry for the group tste12lab, eg tste12lab99, which would mean group number 99. This is your group number.

   NOTE! If you are not a member of a laboratory group, please talk with the teacher.
5. Set your working directory to the labgroup home folder. Change 99 in the command to your group number

   cd /courses/TSTE12/labs/labgrp99

   Do NOT work in your own home directory (e.g., abrli987, djikh999, winch945), use the directory ‘/courses/TSTE12/labs/labgrp<nn>’.

There are two ways to start the HDL designer application, TSTE12lab and TSTE12proj. These commands checks the group number assigned to you, and moves automatically to the correct project or lab directory before starting HDL designer.

6. Use the started terminal as your terminal.
7. Now start the HDL designer environment from the mentorskal

   TSTE12lab

   or

   TSTE12proj

There will appear a window looking like Figure 9, “Default HDL Designer window”. This is the initial appearance of HDL designer. The first time you start HDL Designer is a setup assistant

opened, as shown in Figure 10. Follow the step below in 3.3.2 Configuring HDL Designer the first time the first time to configure the tool.

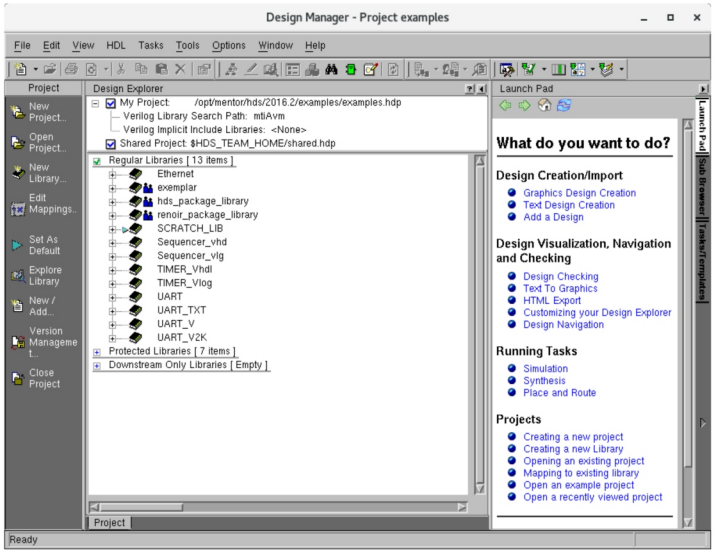

Figure 9 Default HDL Designer window

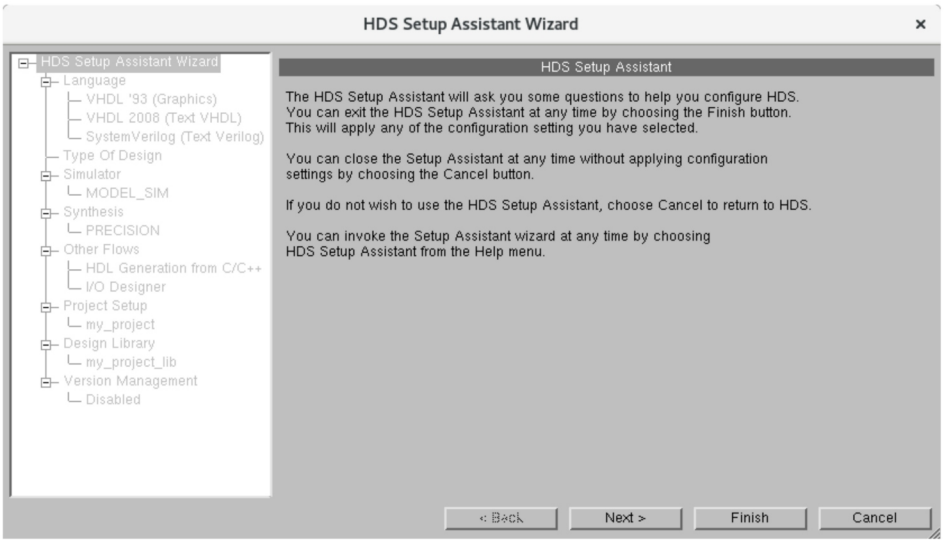

Figure 10 Setup assistant initial window

3.3.2 Configuring HDL Designer the first time

The HDL Designer tool requires some initial configuration to be performed before it may be used. The following step should only be necessary the first time the tool is started. It will create some configuration files in your home directory, and use that configuration for any future usage.

It is possible to start the HDS setup assistant wizard later by selecting Help->HDS Setup Assistant.

The following entries should be set in the HDS Setup Assistant Wizard once it starts.

1. Select Next
2. Set the default language to VHDL'93 both for graph and text, select next
3. Select FPGA as the type of design to create, and select both Altera and Xilinx FPGA technology, select next

4. Select the default MODEL_SIM simulator, select next
5. Select the default PRECISION synthesis tool, select next
6. Deselect HDL Wrapper Generation from C++ and I/O Designer, select Finish
7. Select cancel when given option to add a design or create a new design contents.

A web browser is necessary to read the software documentation etc. The one to use must be defined.

8. Select Options→HTML browser. Set it to /usr/bin/firefox.

HDL Designer automatically adds some default library definitions to all designs. This have to be modified in our case. The following step sets the default library set to include ieee.numeric_std.

9. Select Options->VHDL in the main windows. Select the Default Packages subwindows. Change the text by replacing the reference to ieee.std_logic_arith with a reference to ieee.numeric_std as shown in Figure 11.

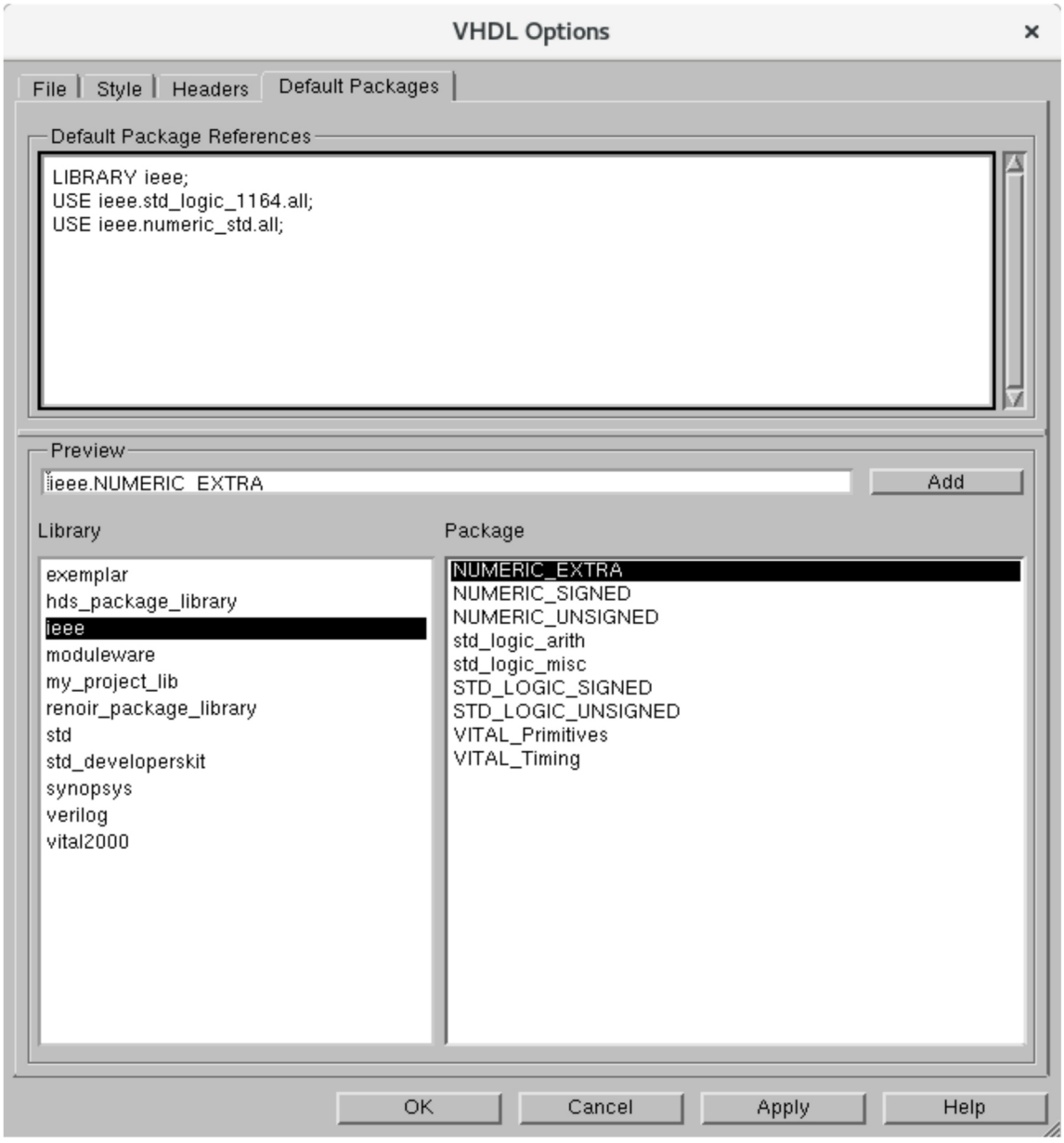

Figure 11: Modified default packages definitions

3.3.3 Creating a new project

A project is a collection of libraries containing the design files. Libraries in a project can be of three types; regular, protected, and downstream only. The design you create should be placed in a regular library.

1. Select File->New->project. The current opened project (if any) must be closed before a new project can be created, select ok if asked to do so.
2. Add a name to your project. In this example we have used my_project. You may also enter a short description of your project. See Figure 12.
3. Make sure the directory in which the project is created is correct. Use the Browse button to select another directory. It should point to the proper lab or project directory. Press the next button to continue to the next step.
4. Select Next. Select “Open the project” and press Finish to end the project setup. You are now ready to enter design data into the project design library.

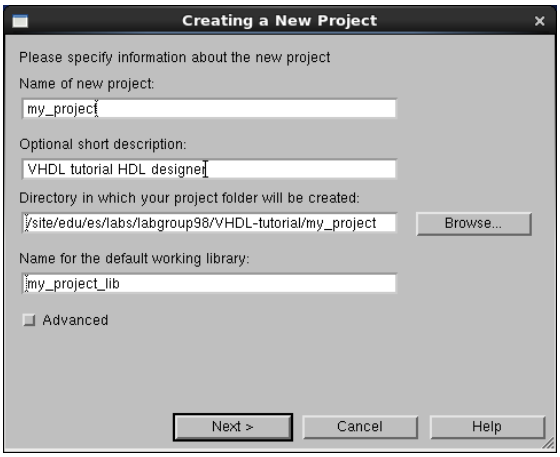

Figure 12 Project creation window

3.3.4 Creating the symbol

We are now ready to enter the component interface. The interface and its corresponding symbol corresponds to the VHDL entity description.

1. To define the symbol select File->New->Design content...
2. Select Interface from the dialogue that is shown in Figure 13, “Create a new symbol using New Design Contents Wizard” and press finish.
3. The symbol editor window appears. The window have a number of possible different views, selected through the Structure Navigator in the upper right corner of the window.
4. Save the symbol using the file menu/save. Save to the library we created and use “example_design” as name of the design unit.
5. Now add ports to the symbol. Select the Symbol tab in the symbol editor window. Useful symbol editing functions are described in Figure 14, “Symbol editor buttons”. Locate them in the symbol window and press to add an input port. Place the cursor pointer next to the symbol and click. The port is now attached to the symbol. Right-click to leave the add port mode.
6. Edit the port definition to suit our definition. Double-click on the port text to edit the name and definition. Figure 15 shows the completed symbol. This can also be done using the textual interface window as shown in Figure 16 Symbol editor window with completed example_design interface.

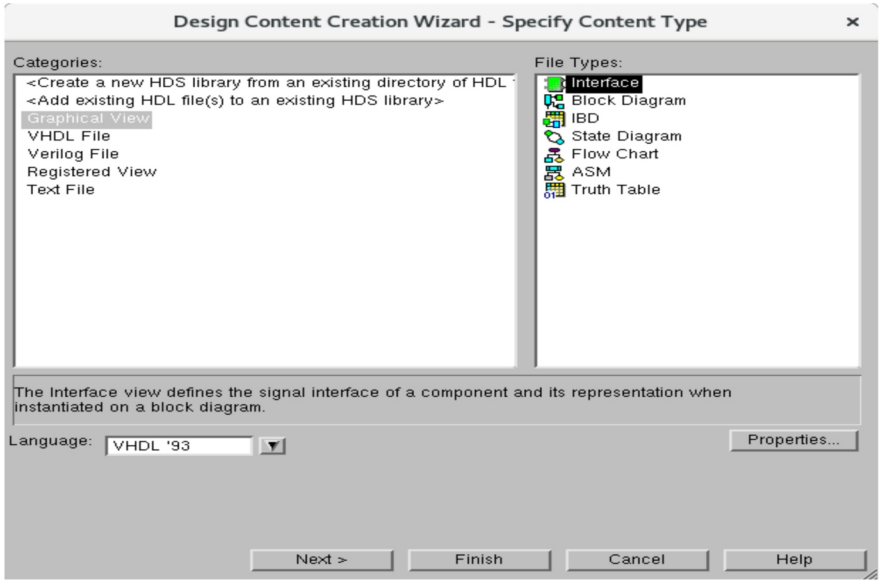

Figure 13: Create a new symbol using New Design Contents Wizard

Button	Description
	Select text or object
	Select text only
	Select object only
	Add or modify comment text
	Pan the window
	Add an input port
	Add an output port
	Add a bidirectional (inout) port
	Add a buffer port (VHDL only)
	Add a panel
	Display the symbol interface as a tabular IO view

Figure 14 Symbol editor buttons

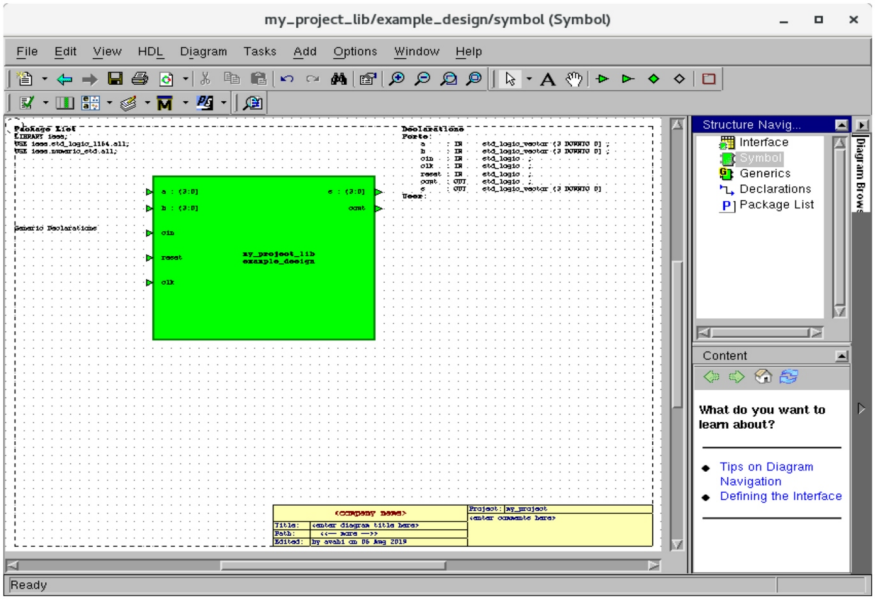

Figure 15 Symbol editor window with example_design symbol

7. Add remaining ports to have a complete symbol as shown in Figure 15, “Symbol editor window with example_design symbol“.
8. Save the symbol.

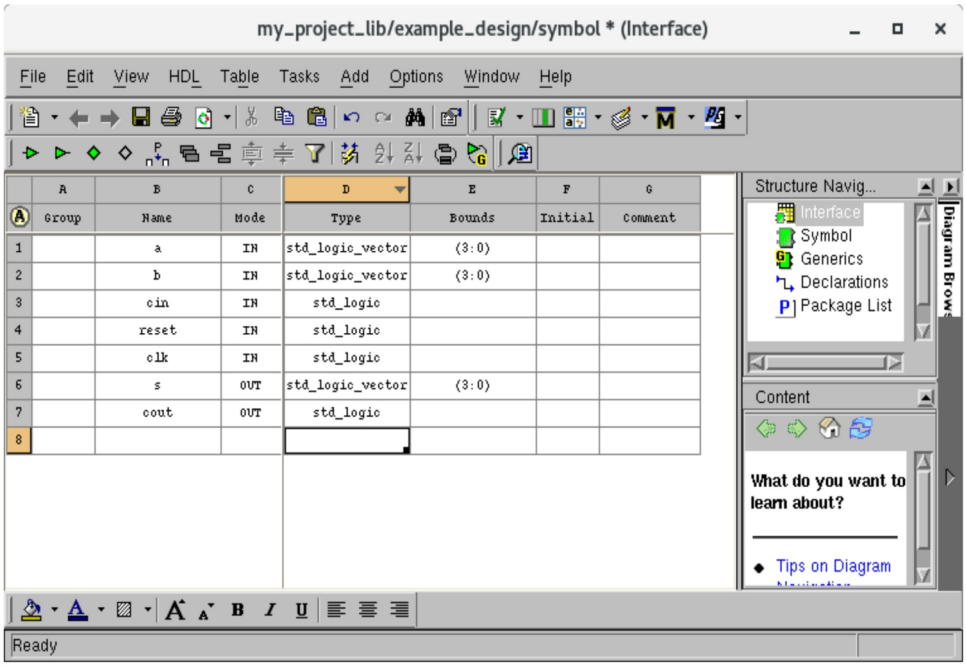

Figure 16 Symbol editor window with completed example_design interface

3.3.5 Add a view to the symbol

The views to each component symbol define the behavior/architecture. There are several different ways to define this. All views are translated to VHDL based on the graphical views. The first we explore is a user written VHDL view.

1. In the symbol window press right mouse button on the symbol and choose "Open / New view"
2. We reach the window in Figure 17, “Open Down Create New View, add a VHDL combined view”. Select VHDL File and Combined file type. Select also VHDL'93. Press next.
3. Name the architecture to behav. Press finish.
4. Inspect the window Figure 18, “New VHDL combined view”. The complete combined view is shown. However, only the architectural is to be changed here. The entity part is generated from the symbol.
5. Add the missing code in the architectural part. The code is described in section 1.2, “Example Design“ as architecture behav. It is the same code we simulated in section 2.2, “Compiling and simulating the VHDL files“.
6. Save the VHDL view.
7. Use Document->Check Syntax to verify that the code syntax is correct. This is also done automatically when the file is saved.
8. If there was a syntax error use the error message to locate the error. Correct the syntax errors and repeat from 6 above until no errors remain.

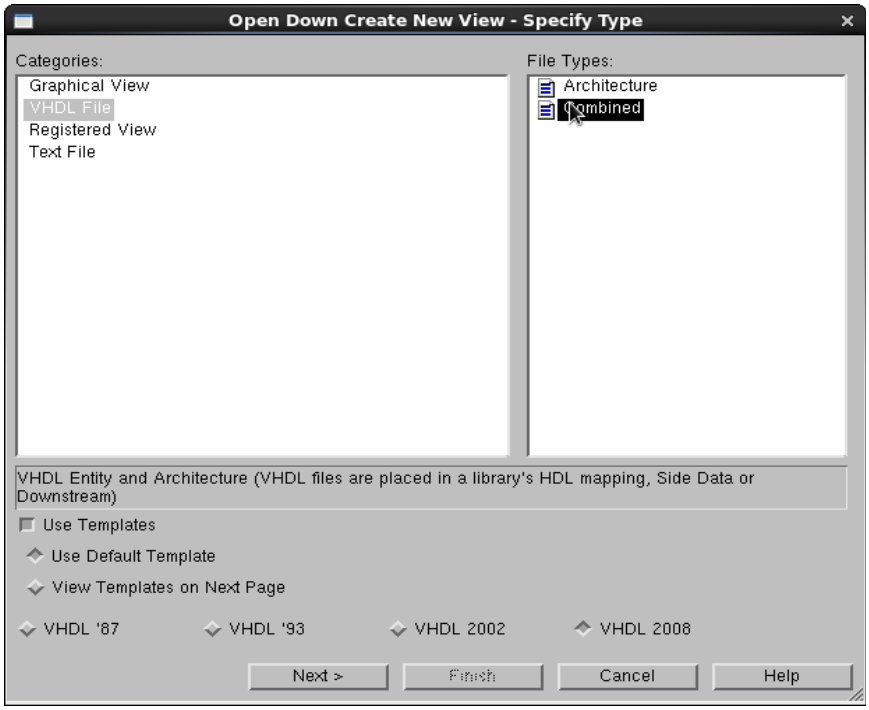

Figure 17 Open Down Create New View, add a VHDL combined view

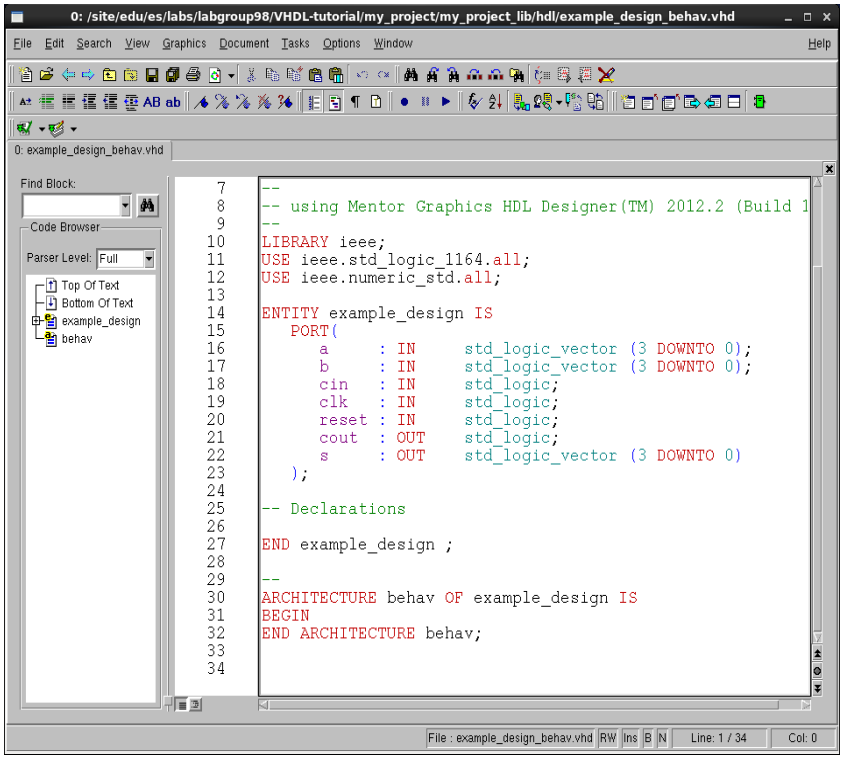

Figure 18 New VHDL combined view

3.4 Validate the design – single level

There is now a complete design. The first step is to generate the VHDL code, compile this to validate a lexically correct code. In the next step a functional validation is performed with simulation

1. First we have to add a toolbar button to the design manager window. Enter the menu Tasks and select "Tasks and templates".
2. A window now appear to the right in the design manager window.

Locate the tasks window tab in the right part of the window.

3. Select the icon "Modelsim Compile" with the left mouse button, Press right mouse button on the icon, in the pop-up menu select "Add to" and then Toolbar.

   We have now installed the compile toolbar button.
4. Select the design unit "example_design" in the design manager window.
5. Locate the Generate toolbar button. Keep the mouse pointer over the buttons and a help text appear as guide. Press the drop menu at the Generate button and select "Perform generation on graphics files (Single)"

The tool now generates interface VHDL code and concatenates it with the behavioral code we entered. Locate and correct any errors that appear in the transcript window. The next step is to compile and simulate.

6. Locate the Modelsim Compile toolbar button. Keep the mouse pointer over the buttons and a help text appear as guide. Press the drop menu at the Compile button and select "Runs Modelsim compilation (Single)".

Locate and correct any errors that appear in the transcript window. It is not possible to simulate the design if it not is compiled without errors.

7. Locate the Modelsim Flow toolbar button. Keep the mouse pointer over the buttons and a help text appear as guide. Press the drop menu at the Compile button and select "Generate and run entire Modelsim flow (Single)".
8. The dialog box to start modelsim simulator appear. Press Ok and wait until the simulator has started.
9. Validate the design with the methods as the pure VHDL model. Here it is possible to save a lot of time with the use of modelsim macro files.

3.5 Additional views

In this part, the concept of successive refinement using hierarchy is demonstrated. We use block diagram to implement a schematic and refine the structure of the design.

3.5.1 Hierarchy

1. Open the symbol editor window.
2. Press and release right mouse button over the symbol. From the popup menu select Open / "New View...".
3. In the "Open Down Create New View" window select the type "Block Diagram". Press button "Next" in the dialog box and then button "Finish" on the next dialog box.
4. Save the schematic. Move the mouse pointer to the menu File and then select Save. Is is also possible to use the keyboard shortcut pressing Control and S simultaneously, i.e. Ctrl-S. Save your changes frequently.

The schematic that appears will once completed describe the same structure as shown in Figure 2, but this far only the port connections have been automatically added. The interface is extracted from the symbol interface. Compare the ports in the schematic to the symbol. The next step is to add blocks and connections.

5. Add three blocks to the schematic. Locate the blue block entry in the toolbar buttons. Press the button and place three blocks in the schematic using the left mouse button. End placing blocks with the right mouse button. Resize and arrange the blocks on the schematic.
6. Connect the ports to the associated blocks. Place the mouse over the red circle. Press and keep down the left mouse button over the circle and drag over the block it should be connected to. Release the mouse button and the wire is now connected to the block.

Next is to add a global connector and illustrate the use of connect by name.

7. Add a global connector to the clk port. This way clk will be implicitly declared on all block interfaces in the schematic. Locate the yellow round dot entry in the toolbar buttons. Place the connector close to the red circle of the clk net. Drag and drop the red circle from clk net on the connector to connect them.
8. Connect an input port/net to the dff and dff4 blocks. Locate the "Add signal" toolbar button. Add the reset signal by pressing left mouse button outside the block followed by pressing left mouse button inside the block. The signal will have a name like sig0. We will change this in the next step. Continue adding the second reset signal and end using the right mouse button.
9. Change name from sig0 to reset. Press right mouse button on the sig0 name and select Object properties. Alter the name to reset and press Apply. A warning appear the nets are now connected by name. Press Ok! Repeat this on all reset signals.
10. Next is the addition output connected to the 4-bit register. Locate the "Add bus" toolbar button. Add the bus pressing left mouse button over the block and then move over to the next block and press left mouse button again. The signal will have the name look like dbus0. We change this in the next step.
11. Change name from dbus0 to d_4bits. One way to do this is to press right mouse button on the dbus0 name and select Object properties. Alter the name to d_4bits. If necessary change style to bus, type to std_logic_vector and set bound to 3 downto 0. Press Ok
12. Add the reaming signals and buses.

The next step is to add behavior to the blue blocks.

13. Press and release right mouse button on the block that will be named dff. From the popup menu select Open as / "New View...".
14. We reach the window , “ Open down Create New view - ” . Select VHDL view and combined view. Press next.
15. Replace the entity name <block> with dff and set architecture name to behav, press Ok.
16. Name the architecture to behav. Press finish.
17. Add the missing code in the architectural part. The code is described in 1.2, “Example Design“ as architecture behav in the file example_design_behav.vhdl. It is the same code we simulated in “2.3 Simulating the hierarchical model“.
18. Save the VHDL view.
19. Add behavior to the remaining two blue blocks.
20. Save the schematic.

The block diagram should now be complete. The next step will be generate code, compile and validate the behavior.

3.6 Validate the design - block diagram

There is now a complete design. The first step is to generate the VHDL code, compile this to validate a lexically correct code. In the next step a functional validation is performed with simulation

21. We have to make sure that the struct view is the default one, i.e. the view that will be used in generate and compile.

    Select the struct view in the design manager window. Press right mouse button on the name and release. From the popup menu select "Set Default View". A small blue arrow will now appear next to the view.
22. Select the design unit "example_design" in the design manager window.
23. Locate the Generate toolbar button. Keep the mouse pointer over the buttons and a help text appear as guide. Press the drop menu at the Generate button and select "Perform generation on graphics files (Through blocks)"

The tool now generates interface VHDL code and concatenates it with the behavioral code we entered. Locate and correct any errors that appear in the transcript window. The next step is to compile and simulate.

24. Locate the Compile toolbar button. Keep the mouse pointer over the buttons and a help text appear as guide. Press the drop menu at the Compile button and select "Runs Modelsim compilation (Through blocks)".

Locate and correct any errors that appear in the transcript window. It is not possible to simulate the design if it not is compiled without errors.

25. Locate the Modelsim Flow toolbar button. Keep the mouse pointer over the buttons and a help text appear as guide. Press the drop menu at the Compile button and select "Generate and run entire Modelsim flow (Through blocks)".
26. The dialog box to start modelsim simulator appear. Press Ok and wait until the simulator has started.
27. Validate the design with the methods as the pure VHDL model. It is here possible to save a lot of time with the use of modelsim macro files.

3.7 Tips and tricks

The VHDL generated for a component can be viewed by right-clicking on a component and select “View generated HDL”.

All generated VHDL code can be removed. The schematics and interface definitions are not removed, but the VHDL generated from these descriptions is removed. Right-click on the library name and select “Delete generated HDL from disk”.

If the library you have worked with is not available any more, then create a new library in the project where the location is set to the same as your previous library (e.g. /courses/TSTE12/labs/labgrpXX/lab1/KEYBOARD).