Laboratory 2
VHDL Simili and Sonata

CO 2206 Computer Organization

Objectives

- To be introduced to VHDL Simili 2.0
- To have some hands-on with VHDL programming

VHDL Simili and Sonata

- VHDL Simili is a collection of tools that facilitate the design, development and verification of hardware models using VHDL
- Sonata, on the other hand, is VHDL Simili's IDE (Integrated Development Environment) tool for designing, verifying and managing HDL projects
  - Sonata is capable of handling complex projects consisting of multiple libraries where each library is built with zero or more HDL files
- Sonata project management tasks can be broken down into the following broad categories:
  - The workspace
  - Working with libraries
  - Compilation
  - Simulation

The workspace

- Each project is represented by a workspace (also called a project)
  - Before Sonata can be used for anything, a workspace must be created
- Workspace functions such as opening, saving, closing and creating workspaces can be accessed using the **File** menu
- A workspace can be created in a directory of your choice
  - all files and libraries will be maintained in this workspace relative to this directory
- Creating a workspace will also create a library with the same name
  - becomes the current working library (which can be changed later)
- Sonata project files are stored in files with the extension **.sws**
  - highly recommended to backup the project files along with source code (i.e. **.vhd** files and associated directories)

Working with Libraries

- The function of libraries can be summarized using the following key points:
  - Every Sonata project is a workspace that manages one or more libraries
  - A library contains a set of design units
    - design units are entities, architectures, configurations, packages and package bodies
  - There is always a concept of the current working library
    - the current working library is displayed in the top left area of the project management window
  - All actions related to project management typically apply to the current working library
  - Each library can be associated with a set of source files (VHDL files)
    - files can be added or detached. Note that detaching a file does not physically remove the file from the file system.

Compilation

- The **Compile** menu provides access to compilation tasks
  - using this menu, you can compile the current file being edited, selected files or all files
- When a VHDL source file is compiled, the results of a successful compilation are placed in a library
  - in other words, the design units contained within a VHDL source file are compiled into a library
- When using a VHDL compiler or simulator, there is always a concept of a current working library
  - if no particular library is specified as the current working library, the current working library is assumed to be "**work**". You can associate the "**work**" library with any other library. When using Sonata, work is always associated (mapped) with an actual library that is registered with the project.

Simulation

- The Simulation menu provides access to simulation tasks
- Basically, this involves identifying the test bench, start the simulation and inspecting the waveform (a graphical representation of a record of the history of changes that have occurred to a given simulation object)

Task 1.1: Adding a Library

- Start up **Sonata**
- You are about to create a library called **lcdf_vhdl** in your project directory
- Download **func_prims.vhd** from *My Moodle* and copy this file into your working directory
- Create a new workspace called "**lcdf_vhdl**"
  - this will create a workspace (with a **.sws** extension) and a similarly named library (with a **.sym** extension)
- Select **Project > Add Files...** and choose **func_prims.vhd**
  - a file called **func_prims.vhd** appears just below the **lcdf_vhdl** library in the *Project Management Pane*
- Double-click **func_prims.vhd** to view the file in the Document Pane
  - you will notice this library consisted of predefined entity such as logic gates
- Compile **func_prims**
  - if you are asked to add the vhd file to the current working directory, click **Yes**
- Note the "Finished compilation session" message in the bottom right Console Pane indicating successful compilation
- Close the workspace

Task 1.2: Creating a Design Entity

- Create a new workspace called "**comb_circ**"
- Select **File > New** to start entering your VHDL code
  - a new file called **Textfile0*** appears in the Document Pane.
  - type the code shown in next slide in **Textfile0***
- Save your file as "**decoder_2_to_4_st.vhd**"
  - don't forget the **.vhd** extension
- Compile your **vhd** file

Task 1.3: Entity Code

```
-- 2-to-4 Line Decoder with Enable: Structural
    VHDL Description
library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all,
    lcdf_vhdl.func_prims.all;

entity decoder_2_to_4_st is
   port(EN, A0, A1: in std_logic;
        D0, D1, D2, D3: out std_logic);
end decoder_2_to_4_st;

architecture structural_1 of decoder_2_to_4_st is

component NOT1
   port(in1: in std_logic;
        out1: out std_logic);
end component;

component AND2
   port(in1, in2: in std_logic;
        out1: out std_logic);
end component;
```

```
signal A0_n, A1_n, N0, N1, N2, N3: std_logic;

begin
   g0: NOT1 port map (in1 => A0, out1 =>A0_n);
   g1: NOT1 port map (in1 => A1, out1 => A1_n);
   g2: AND2 port map (in1 => A0_n, in2 => A1_n, out1 => N0);
   g3: AND2 port map (in1 => A0, in2 => A1_n, out1 => N1);
   g4: AND2 port map (in1 => A0_n, in2 => A1, out1 => N2);
   g5: AND2 port map (in1 => A0, in2 => A1, out1 => N3);
   g6: AND2 port map (in1 => EN, in2 => N0, out1 => D0);
   g7: AND2 port map (in1 => EN, in2 => N1, out1 => D1);
   g8: AND2 port map (in1 => EN, in2 => N2, out1 => D2);
   g9: AND2 port map (in1 => EN, in2 => N3, out1 => D3);
end structural_1;
```

Task 2.1: Creating Test Bench

- Now it's time to test your design by first creating a test bench:
  - Create a new file just like before and enter the code shown in next slide
  - Save the file as "**decoder_2_to_4_st_tb.vhd**"
  - Compile the **vhd** file

Task 2.2: Test Bench Code

```
library ieee, lcdf_vhdl ;
use ieee.std_logic_1164.all,
    lcdf_vhdl.func_prims.all;

entity decoder_2_to_4_st_tb is
    -- no need for any ports or generics
end decoder_2_to_4_st_tb;

architecture testbench of decoder_2_to_4_st_tb is
    component decoder_2_to_4_st is
     port(EN, A0, A1: in std_logic;
            D0, D1, D2, D3: out std_logic);
    end component;

    signal ENt, A0t, A1t, D0t, D1t, D2t, D3t:
    std_logic;
    constant prop_delay: time:= 10 ns ;

begin
    decoder: decoder_2_to_4_st port map(ENt, A0t,
    A1t, D0t, D1t, D2t, D3t);
    test: process
    begin
     ENt <= '0'; A0t <= '0'; A1t <= '0';
     wait for prop_delay;
     ENt <= '0'; A0t <= '1'; A1t <= '0';
     wait for prop_delay;
     ENt <= '0'; A0t <= '0'; A1t <= '1';
     wait for prop_delay;
     ENt <= '0'; A0t <= '1'; A1t <= '1';
     wait for prop_delay;
     ENt <= '1'; A0t <= '0'; A1t <= '0';
     wait for prop_delay;
     ENt <= '1'; A0t <= '1'; A1t <= '0';
     wait for prop_delay;
     ENt <= '1'; A0t <= '0'; A1t <= '1';
     wait for prop_delay;
     ENt <= '1'; A0t <= '1'; A1t <= '1';
     wait;
    end process;
end testbench;
```

Task 2.3: Simulation

- Once you have completed the code, you are ready to simulate your design:
  - Select **Simulate > Select Toplevel… > decoder_2_to_4_st_tb**
  - a top-level design is the name of an entity or architecture or a configuration that will serve as the root of your design. In most instances, this is actually your test bench. If a top-level design is already established, you will see it listed in the top-left area of the *Hierarchy Pane* (*Workspace pane*)
  - Select **Simulate > Go**, or click the button
  - Waveform **Waveform0.wfs** will be displayed in the *Waveform Pane* (in the *Document Pane*)
    - initially, the waveform window does not have any objects displayed
  - To display waveforms for VHDL signals (or non-subprogram variables), they have to be added to the waveform window
    - in the bottom left *Objects Pane*, right click to **Add all signals to waveform**
  - At the very beginning, simulation is paused just prior to time zero
    - this means that no simulation time has elapsed and advancing the simulation in any way would potentially advance time. Simulation can be advanced in one of two ways:
      - use the run ▶ button to advance simulation
      - running the simulation will display the simulation behavior of the signals being probed in the waveform window

Task 3: 2-to-4 Decoder

- Rewrite **decoder_2_to_4_st.vhd** (simply create a new architecture on top of the previous architecture) using
  - **std_logic_vector** notation instead of **std_logic** notation
  - implicit specification of the component input and output names by their order in package func_prims in library **lcdf_vhdl**
  - for example, `g0: NOT1 port map (A0, A0_n);`
- Compile, simulate and verify the circuit

Task 4: Combinational Circuit

- Using the figure below as a framework, write a structural VHDL description of the circuit
  - Consult package **func_prims** for information on the various gate components
- Compile your VHDL, and simulate your VHDL for all possible input combinations to verify your description's correctness