Introduction TL & TP

Even if you probably know how to start and run a VHDL simulator under a UNIX environment, it is better if you surf through this chapter and reload into your cache a few informations by reproducing the simplest of the experiments: starting the simulator and simulating a mux in many configurations!

Furthermore (most important) you will learn how to synthesize for ASIC.

0.4 VHDL simulation

0.4.1 Simulating a MUX

Reading the INVERTER netlist you can recall the use of generics. Notice that a library called **WORK.constants** is used: the constant IVDELAY is not defined within the file. It is in **constants.vhd**: have a look of it. [1]

Now we want to use the mux file within a so called "test bench": **tb_mux21.vhd**.

The test bench contains an *instance* named **U1** of the **COMPONENT MUX21** which is first declared (keyword **component**) and then instanced (keyword **Port Map**). This test bech also instances the other three types of MUX, by declaring U2, U3, and U4.

Remember that the assignment

```
inputA1 <= '0' after 1 ns, '1' after 2 ns, '0' after 10 ns;
```

creates a signal waveform for the signal named **inputA1**.

Now let's start with the simulator. We use Modelsim by Mentor Graphics in these labwork. As a first step we must set up the environment variables, so type the command:

```
prompt> setmentor
```

Now type at the prompt the command

```
prompt> vlib work
```

Type again **ls**. You should see that a new directory called "work" has been created. This is a temporary work directory in which the simulator save its results. Now call the simulator:

```
prompt> vsim &
```

A window named *Modelsim SE-64 6.5c* opens. The operations needed for the simulations are:

[1]Remember that if you simply want to read the content of a file without necessarily modify it, you can type on a shell the command: **more filename**, and press the spacebar to go through the file or Ctrl + c to interrupt the reading.

- Go to **Compile→Compile**. Double click on file **constants.vhd**. You'll see that in the command window of the main window a command **vcom** has been executed. This command COMPILES your VHDL file *constants.vhd*. The result of the compilation is written in the directory **work**.
- Now compile the other files in a hyerarchical sequence from bottom to top. You can use either the menu, as in the step before, or type the equivalent command manually in the command window: for example `vcom iv.vhd`. Or better, you can use the shell script:

  ```
  prompt> ./compile
  ```

  What happens?
- When it is finished open the simulation window (Simulate→Start). Choose a simulator *resolution* of 100 ps and *add* the design **work/MUX21TEST** (which is the configuration name in the test bench). Then click **Ok**. Disable the optimization option. The design is now ready for simulation. Note that the equivalent command that has been executed in the main command window is: `vsim -t 100ps work.MUX21TEST`.
- If you want to see the waveforms type the following command

  ```
  add wave *
  ```

  in the command window. A new waveform window opens. You can also play with the menus to decide the waveforms you would like to display. Obviously there's nothing to plot until you don't run the simulation!
- To do so type

  ```
  run 3 ns
  ```

  which means that you start the simulation from 0 to 3 ns. The step for the simulation is 100 ps as you chose before. The time unit defaults is nanoseconds. But you can choose another value. Try for instance to type again

  ```
  run 10000 ps
  ```

  Look at the waveform window. Does the muxes mux?.

You can plot the waveform window to a file: the action is **File→Print postcript**, a print window opens: select the **File name** option and add a file name with extension ".ps". In the directory in which you launched the simulator a new postcript file is now present. If you want to view it type at the prompt:

```
prompt> evince filename.ps &
```

To better visualize the waveform in the sheet, before printing you can select the signals you want to print, and select signal height in pixel by the menu: **Format→Height** (select for example 250).

Here is now a string recommending the most important point you should have at the end of the exercise (this will be given for each exercise).

Remark point

We suppose you already knew how a MUX works. So at this point you are supposed to know how to write and simulate a vhdl circuit. Nothing else is required for this.

0.5 Writing your homebook or your project report

At the end of the course you will be asked to give the teacher a proof of your lab works by means of a detailed report on the labs or on your final project.

You will give the teacher a CDrom with all your VHDL (or others, depending on the lab work to be done) files so that she/he can test your work running the simulations/synthesis.
Furthermore you should document the results of your work using a facility given to you for organizing your simple homebook or project report in a pretty ready document based on latex language. The results should be accurately commented in both cases.

On the course website a file homebook.tar.gz in which three files are given explaining what you have to do: **README**, **homebook.tex** and **master.tex**. You can open them (in the given order) using *emacs* or similar editors. Should you need to generate a picture or a block diagram, you can use **xfig** for this, from the prompt; it is quite user friendly; after you finish the design just save it, and then *export* as encapsulated postcript, or as a jpg if you are going to use pdflatex.

Please, do concentrate on this subject only after you have finished this lab-class..

Remark point

We suppose you already knew how a MUX works. So at this point you are supposed to know how to write and simulate a vhdl circuit.

0.6 From VHDL to synthesis

You are going to synthesise the blocks you have simulated in previous sections. You will use a standard cell flow based on one of the most powerful synthesis tool diffused in industry.

Before stepping over please generate a new sub-directory in which you will synthesize:

```
prompt> mkdir syn
```

Copy in there all the VHDL files except tb_mux21.vhd.
Now oper a new shell and go inside such new directory.

0.6.1 VHDL Synthesis of you simple circuits

In this section you will learn how to run a synthesis, given a synthesizable VHDL code and how to analyze a few results of your synthesis. You will be given further informations in the next chapters.

In general the phases you should follow using a synthesis tool are reported below:

1. **Define a library** of gates previously characterized (ex: AND2, input capacitance, delay function of fan-out, power consumption at each output switching, ... in the next chapters you will learn more in detail this point).
2. **Analyze and elaborate the file** you are going to synthesize (in a HDL format): during this step the tool checks if your design is correctly synthesizable. If not, it returns error messages and you can't go through the synthesis until you don't modify the code.
3. **Define constraints** for the synthesis. This is not strictly necessary, but if you don't perform this step you'll obviously get non optimal results. You can set one or more constraints in the same time: the tool builds a multi-objective function and try to figure out the result that assures the minimum distance to all the constraints.

4. **Synthesize the design**. The synthesizer maps the VHDL code on the gates in the library, using their characteristics to choose the gates optimal from the constraints point of view.
5. **Report results**. You can analyze at this point if the constraints are met or not, you can list the critical paths and the power informations for nets and cells, even if the design is a hierarchical one.

In our case we will use as a synthesizer the **Synopsys Design Compiler**, which is the real engine, and its GUI: **Synopsys Design Vision**. Before starting the exercise it is necessary you go through a few steps for making the tool working correctly.

Be sure you have in the directory **syn** the file:

/home/repository/lowpower/setup/.synopsys_dc.setup

This is a hidden file that you can list using the command:

```
prompt> ls -a
```

and that you can normally display using **more** or **emacs**. It gives some information to the tool, such as which is the library you are going to use and where it can be found. Anyway, you don't need to modify it right now.

Now the environment variables needed to launch the Synopsys synthesizer must be defined: this is done in a script file that you can execute by typing the command:

```
prompt> setsynopsys
```

As a last preliminary operation create a directory in which the synthesizer save temporary files during compilation:

```
prompt> mkdir work
```

A final IMPORTANT comment. With the informations you will be given hereinafter, you should be able to synthesize all the VHDL designs you have already completed. Two are the main ways to perform a synthesis: using the graphical user interface and going to and fro around the monitor clicking with you mouse here and there, or using a command window in which you are allowed to execute singular commands or a script grouping a command sequence that you execute only once. The use of the command window and of scripting is obviously the clever way, but it is not for a newbie. So you'll learn here how to click in the right sequence and understand what's going on; sometimes informations and suggestions will be given so that, hopefully, you will be automagically transformed from a newbie to an experienced user.

0.6.2 Synthesis of the MUX

Now launch the synthesizer by typing:

```
prompt> design_vision &
```

If everythink is ok with the licence then a main window opens with title: "Design Vision" (in case this does not happen you get a licence error: GOTO the end of this chapter). Immediately note the bottom command line, from which you can manually set commands. In the space above, the results of your commands are displayed. If you operate using the main window menu, the name of the corresponding command and its results are displayed in the command window as well. For the

moment use this command line only when suggested.
Please note exactly above the command line the menu with the alternative “Log”, which corresponds to the log window above, and “History”: select it. Each time you will type a command in the command line, or give a command by the Design Vision buttons, you will have it in this history window. You will be able to use it for editing, executing or saving such commands, so that you will easily generate scripts for you next sessions.

Now let’s go back to the Log window and let’s use the interface: using its menu perform the following operations:

Analyze file: From menu select: **File→analyze**; a window opens from which you can type “Add”, select the **constants.vhd** file and click OK to confirm your selection. In this phase the code is checked and errors are displayed if it is not synthesizable. Note the corresponding command in the Log window and in the history window.
Now analyze the **iv.vhd** file, the **nd2.vhd**, and the **mux21.vhd** files with the same steps as before. The top entity is the MUX21. Note that in file constants the delay values are defined, but should be commented in the top level: the timing parameters are ignores for the analysis, and must not be included in the further steps.

Elaborate From menu select: **File→elaborate**. During this step the compiler creates an internal graph of your circuit and maps it virtually to general gates. You need to elaborate only the top level entity. In this first step we will work on the first behavioral architecture, so, in the “Elaborate design” window select the “WORK” library, and the “MUX21(CFG_MUX21_BEHAVIORAL_1)” option.
You can now explore this “logical” structure of the MUX. In the left “Logical Hierarchy” window select the MUX structure. Now, in the top menu the “Create Symbol View” and the “Create Design Schematic”appear. Click the “Create Symbol View” button: a MUX symbol appears. Now double click inside of it: the schematic view appears. You can surf in it by using the right mouse button (Zoom selections... ESC to leave the Zoom option). If you click in the internal boxes you will only get the logic representation created using generic basic ports. These are not the cells in the library we are going to use for the synthesis.
Before steeping over, click on the drop-down menu selecting ”MUX21”.
Check that in the log window appears: **current_design ”MUX21”**

Look one moment in the history window the correspondence between your selection and the given command.

Synthesize: for the moment we do not use constraints for the synthesis, so go to the main window and select **Design→Compile Design**. A small window opens, in which, for the moment, you don’t need to select anything different from the default: just click OK. This corresponds to giving in the command window the instruction

compile -exact_map

Now the tool is mapping your design on the library we are using ($0.045\mu m$). When it has finished you can explore the results from the main window exactly as before. Note that the MUX structure did not change, but the internal structure, which is behavioral, has been replaced by exact cells present in the given library. For example, select the internal leaf. Click with the right mouse button selecting “Properties”: you will read the real name of the used gate in the library.

You can plot to a file the schematic displays in the main window by selecting **File→Print**, checking the "Print to file" option and then typing the file name.
If you want to go up in hierarchy you must click on the up arrow displayed on the top bar of the main window. Remember that whatever command you are giving, it will be applied to the hierarchical level you are in.

Save the design: We want to save this compiled design so that, in case we optimize it or change something, we can start again from here without reproducing the previous steps. From the main menu select: **File→Save as**, and in the new window put a meaningful name, eg. "mux1.ddc". Hereinafter you will be able to get your design at this point by simply reading it from the main menu (**File→Read** and select it.)
Please look at the command in the history page and remember it.

Generate to VHDL file: We want to see the VHDL netlist of the whole design so that we would be able to perform a backannotaded simulation.

Now, exactly as before, from the main menu select: **File→Save as** and choose vhdl as extension (vhdl, not vhd). Now look at the file end analyze the mapped structure: clearly it supposes to have the entity description of each of the component declared.
Please look at the command in the history page and remember it.

List reports: Now we want to analyze the circuit performance. From the main menu select **Design→Report Area**. A new window opens. You can decide (do decide it now..) to write the report on an external file by checking the "to File" box and inserting the file name, eg. *mux-area.txt*. The same command can be obtained by typing in the command line

report_area > mux_area.txt

Hereinafter instruction for saving the reports will not be given: it's up to you to decide when you need to save the results on a file or to write down the results you will find from these reports so that you can compare them with future results obtained by constraining the synthesis (next chapters).

Now report the timing analyses: from the main menu select **Timing→Report Timing path** and leave the default settings. Save the report on a file. You will see the timing report for the worst critical paths on the report window: try to understand the given informations.
The same results could have been obtained simply writing in the command line (try it) the command:

report_timing

In this way the worst critical path is displayed.

Finally you are for sure interested on how to analyze power. The command to be used is:

report_power

in which three contributions are reported:

Internal power this is due to short circuit current (pmos network and nmos network both partially in conduction), which strongly depends on transistor sizing, and partially depends on the gate delay, and thus on the input transition time and on the output load.

Net switching power this is due to the output load of a gate, thus the higher is the load and the bigger is this power dissipation. Such load is due to the total input capacitances of the following stages and to the net load.

Leakage power due to static power and depends on transistor sizes and threshold voltage.

Look at the different power contributions and analyze them.

Mind: this command must be carefully used: during the next labs you will be warned about this point.

It's your turn: Now you know how to run a synthesis. Repeat the same steps as before for the other MUX architectures and analyze carefully the different results.

Remember to close the NX session when you are done. To save your work you can send it by e-mail using the available browser using your student account, use your disk space in your Portale della didattica disk, use the scp command or FileZilla.