Microelectronic Systems

Lab 0

First VHDL simulations and syntheses

March 1, 2021

1 Introduction

In these sections you will learn or recall how to start the VHDL simulator, how to write and simulate the basic VHDL structures and how to synthesize them for an ASIC target. This first chapter, given your pre-requisites, shouldn't carry anything difficult to you, but please pay attention to the details, to the VHDL writing styles and constructs, on how to use test benches for a meaningful, portable and easy simulation phase. Probably you are a new-by for what concerns ASIC synthesis, so please pay attention to the first hints on this very important step.

As a rule of thumb you are expected to finish this setup part (vhdl simulating) within the first hour. The last part should be enought for the synthesis session.

If you are not a UNIX expert, you first need some suggestions on how to create directories, handling files, et cetera. So go through the next section as a first step.

2 List of important Bash commands

As the laboratory sessions will require the connection to a remote server and programs will run inside a Linux Virtual Machine, some basic programming skills are needed. In particular, you often will be requested to use the terminal. The most important commands that you will be expected to consider are reported in the following. If necessary, more advanced features will be detailed along the different laboratory experiences.

Bash command	Brief description
ls	Display directory contents
touch <*file_name*>	Create a new file
mkdir <*dir_name*>	Create a new directory
pwd	Display the path of the current working directory
cd <*dir_path*>	Change the working directory[1]
mv <*file_full_source_path*> <*file_full_destination_path*>	Move a file from a source position to a new destination
cp <*file_full_source_path*> <*file_full_destination_path*>	Copy a file from a source position to a new destination
cat <*file_name*>	Display the content of a file
man <*command_name*>	Display the man page of a specific command

Table 1: Main Bash commands

Learning the basics of Bash programming is a very powerful skill. If you are interested in deepen the topic, please refer to the official manual pages[2].

3 Let's start!

At first you need to login on your working area. In the x2go login window, you already configured, type the login and the password that have been assigned to you. In general it is:

ms21.X for groups from 2 to 9

ms21.XX for all the other groups

[1] There are the following conventions: **.** (single dot) refers to the current directory, **..** (double consecutive dots) refer to the parent directory.

[2] Reference: `https://ss64.com/bash/`

where X and XX represent your group number (assigned on the spreadsheet you are supposed to have accessed already.

The default password is set to:

mic!tem21sys

3.1 Change your account password

Once you are logged in, YOU MUST CHANGE THE PASSWORD. You are not allowed to go on until you don't change it. In order to do so:

- First, open a "shell" terminal (the action is: right click over a blank zone of the screen, then click "Konsole")
- Now choose a password of 6 characters, at least one of which should be a special character like **!** **_** #
- Then, digit at prompt the command

  `prompt>` **passwd**

  The system asks for the OLD password first (your login in this case), then it asks twice for the NEW one
- After you completed this operation, COMMUNICATE the new password ONLY to the teachers during the first lab.

3.2 Prepare the work-space directories

The following commands are meant to create directories and copy files. However, as anticipated above, using terminal and manual commands is a much faster and more powerful approach.

You can start now with creating the first directory. At the prompt digit

`prompt>` **mkdir setup**

which creates the directory **setup** inside the current directory (i.e. /home/ms21.XX). If you want to see it, type

`prompt>` **ls**

which is the command that lists the files in the current directory.

Then, enter the directory by typing:

`prompt>` **cd setup**

This is your working directory where you will organize all your results (not only VHDL files).

So, create a new sub-directory in which you will work with your VHDL simulations:

`prompt>` **mkdir vhdlsim**

and another directory in which you will run your synthesis:

```
prompt> mkdir syn
```

Enter the first directory:

```
prompt> cd vhdlsim
```

Suppose you want to copy a file located in the directory

/home/repository/ms/setup/

into your own new directory. In order to do that, type the following command:

```
prompt> cp /home/repository/ms/setup/iv.vhd .
```

Please note the final dot separated by a blank space from the .vhd file, do not forget it.

This action copies the file named **iv.vhd** from the repository's directory to your current directory (**.**).

To see your new file you have already copied, type again:

```
prompt> ls
```

Now copy all the files present in the repository[3]:

```
prompt> cp /home/repository/ms/setup/* .
```

Your directory now contains the VHDL description files of some **small gates**, a **test bench** and a very **simple script** for compiling the VHDL netlist inside ModelSim's environment.

You can edit your inverter file using the available text editors[4], as EMACS, KEDIT, VI or JOE. Most of them recognize the VHDL keywords properly coloring the different words. EMACS is a bit *tricky* but very powerful as it enables an automatic completion of VHDL constructs while you are writing.

To try EMACS run the following command at the prompt:

```
prompt> emacs iv.vhd &
```

A new window opens. The symbol & helps in running the process you started in background: this means that you will still be able to use the terminal in which you are working. This is true for every command you are going to launch. If you forget to type the & character, you can anyway put the process in background by typing in sequence:

```
prompt> CTRL z

prompt> bg
```

The first command suspends the running process, the second resumes it in background.

[3]The same files will be also available on the course website
[4]You can find them in the applications menu

A closer look at EMACS

Just to try EMACS' power: add an entity by typing *'ent'* and press tab to complete the it. Now type space and you can type the name as it appears in the bottom line of the window. Then, type enter and the *'end entity'* appears as well as its *'generic'* structure, if you have one. Suppose you have not and type enter. At this point, the *'port'* definition appears. From these simple tips you can PLAY yourself!

Now you are ready to analyze and simulate a VHDL block.

4 VHDL simulation

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 but you have to refer to **constants.vhd**, take a look of it[5].

Now you are ready to analyze the gate characterized inside the ND2.vhd file together with the various descriptions of the MUX21. Please, take your time and be sure you understand the meaning of all the code you are reading.

At this point, we want to use the mux file within a so called *"test bench"*, implemented inside **tb_mux21.vhd**.
The test bench is a wrapper which uses the circuit you designed; it helps the simulation, but is not part of the architecture. It is extremely useful and avoids forcing signal by hands.

It contains an *instance* named **U1** of the **COMPONENT MUX21** which is first declared (keyword **component**) and then instanced (keyword **Port Map**). This test bench also instances the other three types of MUXes, by declaring U2, U3, and U4. Notice that the entity is always the same. The different architectures are differentiated by the configuration declaration at the bottom.

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 and, in particular for these lab works, we will consider Mentor Graphics' ModelSim.
As a first step we must set up the *environment variables*, so type the command:

```
prompt> setmentor
```

Now type:

```
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 saves its results.

[5]Remember that if you simply want to read the content of a file without necessarily introducing any modifications, you can type inside a shell the command: **more filename** and press the space bar to go through the file or CTRL-C to interrupt the reading.

Now call the simulator:

```
prompt> vsim &
```

A window named *Questa Sim-64 10.7c* opens. The operations needed for the simulations are:

- Go to **Compile→Compile...** . Double click on the file **constants.vhd**. You will see in the command terminal belonging to the main window that the command **vcom** is executed. This command COMPILES the VHDL file you have previously selected. The result of the compilation process is written in the directory **work**, inside your working directory.
- Now, compile the other files in a hierarchical sequence from the ground up. 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`. Moreover, an even better procedure could consider directly a shell script (outside the simulator, in the main terminal), whose content is:

  ```
  prompt> ./compile
  ```

  What happens?

- When the process is finished, open the simulation window (Simulate→Start Simulation...). Choose the simulation *resolution* of 10 ps and *add* the design **work/MUX21TEST** (which is the configuration name in the test bench: be careful, if you have different configurations and you want to simulate them, all that you need is to select the configuration and not just the entity in this point). Then click **load** and disable the optimization option. Now the design is ready for simulation.
  Note that the equivalent command that has been executed in the main command window is: `vsim -t 10ps work.MUX21TEST`.

  REMEMBER in the future to set in all the simulations the RESOLUTION. The smaller is the number you choose, the longer is the simulation. However, the resolution must be lower that the smallest timing value you happened to use in your VHDL description.

- If you want to see the waveforms of the signals beloging to the design, type the following command in the command window:

  ```
  add waves *
  ```

  The new waveform window opens. You can also play with the menus to decide the waveforms you would like to display. However, at this point, there's nothing to plot as you didn't run the simulation.

- To do so type always in the same command window:

  ```
  run 3 ns
  ```

  This means that you start the simulation from 0 to 3 ns. The step for the simulation is 10 ps, corresponding to the resolution you chose before. The default time unit is nanoseconds, but you can choose another value. Try for instance to type again:

  ```
  run 8000 ps
  ```

  Look at the waveform window. Does the muxes mux? Are there differences for the various architectures considered?

You can also plot the waveforms that you display inside a file: you have to click the waveform window, then select **File→Print postscript**. A print window opens: select the **File name** option and define a file name with extension *".ps"*. Inside the directory in which you launched the simulator, a new postscript file is present. If you want to view it, type at the prompt:

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

In case you want to transform it into a *.pdf* file, just use the shell command `prompt> ps2pdf filename.ps` and the equivalent pdf will be generated.

To better visualize the waveform in the .ps file, before you print it you can select the signals you want to save and then select the signal height (expressed in pixel) by the menu: **Wave→Format→Height**. Select for example 250.

In the following, a string is reported recommending the most important points you should have gained at the end of this exercise. The same approach will be considered for all the future exercises.

Summary of what is requested

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.

5 From VHDL to synthesis

In the following, you are going to synthesize the blocks you have previously simulated. You will use a **standard cell flow** (subject of this course and described later), based on one of the most powerful synthesis tool diffused in Industry.

Before stepping over, please move into the sub-directory in which you will proceed with the synthesis:

```
prompt> cd ../syn
```

Inside there, copy all the VHDL files with the exeption of tb_mux21.vhd. In fact, the test bench is considered only for simulations and to synthesize it would represent a serious mistake.

Now, open a new shell inside the same directory you have just created.

5.1 VHDL Synthesis of your 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 deriving from your synthesis. You will be given further information along 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 (e.g. for an AND2, input capacitance, delay as a function of fan-out, power consumption at each output switching, ...). In the next chapters you will learn more in detail about that.
2. **Analyze and elaborate the file** you are going to synthesize (described through a HDL format): during this step the tool checks if your design is correctly synthesizable. If not, it returns proper error messages which prevent you to proceed with the synthesis until you don't resolve them, modifying the code.

3. **Define constraints** for the synthesis. This is not strictly necessary, but if you don't perform this step you will obviously get non optimal results. You can set one or more constraints at once: the tool builds a multi-objective function and tries 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 present inside the library, considering their characteristics in order to choose the optimal gates from the constraints point of view.
5. **Report results**. You can analyze at this point if the constraints are met or not, you can display the list of the critical paths and the power information for both the nets and the cells, even if the design is hierarchically described.

In our case we will use **Synopsys Design Compiler** as synthesizer, together with its Graphical Interface, **Synopsys Design Vision**.

Before starting the exercise it is necessary you go through a few steps to ensure the correct functioning of the tool.

Be sure you have in the working directory the file **.synopsys_dc.setup** by copying it from /**home/repository/ms/setup/**):

cp ../.synopsys_dc.setup .

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

prompt> **ls -a**

You can normally display it using **more** command or **emacs**.
The file provides some information to the tool, such as the library you are going to use and where it is located. Anyway, you don't need to modify it right now, just remember to have it in the directory you are using for the synthesis, whenever you are synthesizing.

Now, the environment variables needed to launch the Synopsys synthesizer must be defined. This is performed through a script that you can execute by simply typing the command:

prompt> **setsynopsys**

As a last preliminary operation, create a directory in which the synthesizer will save all the temporary files generated during the compilation process:

prompt> **mkdir work**

A final IMPORTANT comment must be made. Through the information 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* or using a *command window*, in which you are allowed to execute either singular commands or a script grouping a command sequence that you will execute only once. The use of the command window and scripts 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 information and suggestions will be given to you so that, hopefully, you will be automagically transformed from a newbie to an experienced user.

5.2 Synthesis of the MUX

The aim of this section is twofold and, in particular to let you learn firstly how to use the synthesizer and secondly how the different VHDL descriptions are synthesized.

Now, launch the synthesizer by typing:

prompt> **design_vision &**

A main window named "Design Vision" opens.

Immediately note the bottom command line, from which you can manually set commands. In the space above, the results of your commands will be displayed. If you operate using the main window menu, the name of the corresponding command and its results are displayed there. For the time being, use this command line only when suggested.

Please note that exactly above the command line there is a menu with the alternative options *"Log"*, for accessing the log window, and *"History"*, which collects all the commands previously executed (also if you act through the GUI's menus). If you select the latter, you will be able to learn the most important commands so that you can easily generate scripts for you next laboratory sessions.

Now, let's go back to the Log window and let's use the Graphical Interface to run the synthesis process. Using its menus, perform the following operations:

Analyze file: From the top menu, select **File→analyze**. A window opens from which you can type "Add", select the **constants.vhd** file and then click OK to confirm your selection. In this phase the code is checked and errors are displayed if the design is not synthesizable. Please, note the corresponding command in both the Log and History windows.
Now analyze, in the order, the **iv.vhd**, the **nd2.vhd** and the **mux21.vhd** files with the same steps described before. The top entity corresponds to the MUX21. Note that inside the file constants.vhd the values for the characterization of delais are defined- However, they must be commented in the top level: the timing parameters are ignored during the analysis phase and they must not be included in the subsequent steps. Therefore, open the files **iv.vhd** and **nd2.vhd** inside which the delays are included and comment them using "- -" (you you can recognize the delays from the keyword **after**). Then, save the files and analyze them again.

Elaborate: From the same top menu, select **File→elaborate**. During this step the compiler creates an internal graph representing your circuit and virtually maps to it general gates. You need to elaborate only the **top level entity**.During this step we will work on the fist *behavioral architecture*. So, in the "Elaborate design" window, select the "WORK" library, and the "MUX21(BEHAVIORAL_1)" option.
You can now explore "logical" structure of this MUX. On the left "Logical Hierarchy" window, select the MUX structure. Then, through the the top menu select the "Create Symbol View": the "Create Design Schematic" appears. Click the "Create Symbol View" button: a MUX symbol appears. Now double click inside it: the schematic view appears where you can surf by using the right mouse button (Zoom selections... Press ESC to leave the Zoom option).
If you click on the internal boxes, you will only get the logic representation created using generic basic ports. These are not the cells present inside the library and that we are going to use for the synthesis.

Before stepping over, click on the UP ARROW (on the top menu) until you arrive at the top view and, in case, the arrow is black.

Finally, check that inside the *Log window* the following string is present: **current_design "MUX21"**.

Moreover, take a look for a moment at the *History window* and at the correspondence between your selection and the corresponding 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 you don't need to select anything different from the default values: just click OK. This operation corresponds to execute inside the command window the instruction **compile -exact_map**.

Now, the tool will map your design on the library we are considering ($0.045\mu m$). Once the tool has finished, you can explore the results from the main window exactly as before. Note that the MUX structure is not changed, however, its internal structure, which is behavioral, has been replaced by the true cells present inside the given library.
For example, select the internal leaf. Then, right click with the mouse and select "Properties" to read the real name of the used gate from the library.

You can plot to a file the schematic which is displayed on the main window by selecting **File→Print schematic**, checking the "Print to file" option and then typing the file name.

Again, if you want to go up in the hierarchy of the design you must click on the up arrow present 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 the compiled design just obtained so that, in case we optimize it or change something, we can start again from the saved point, without the need for repeating all the previous steps. From the main menu select **File→Save as** and in the new window define a meaningful name (eg. "mux1.ddc"). Hereinafter you will be able to restore your design by simply reading it from the main menu (**File→Read** and select it.) Please, take a look at the corresponding 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 back-annotated simulation.

Now, exactly as before, from the main menu select **File→Save as** and choose VHDL as extension (VHDL, not VHD). Then, look at the file and analyze the mapped structure: clearly it supposes to have the entity description of each of the components declared.
Please, take a look at the command in the History page and remember it.

List reports: we want to analyze the circuit performance. From the main menu select **Design→ Report Area** and a new window opens. You can now decide 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 operation can be performed in the command line by typing:

report_area > mux_area.txt

Hereinafter, instructions 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 for subsequent comparisons (e.g. with the ones derived constraining the synthesis).

Now, report the timing analyses: from the main menu select **Timing→Report Timing paths** and leave the default settings. Then, 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 information.
The same results could have been obtained simply writing inside the command line (try it!):

report_timing

In this way the worst critical path is displayed.

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

report_power

Look at the different power contributions and analyze them.

Know the command you're using: The report commands offer to the user many options to deeply analyze the designed circuit. This is very important when dealing with complex designs. To know all the options applicable to report_timing and report_power "short" manuals are available via the command line and by typing **man** before the command you want to know. The manual is available for almost every command inside Design Compiler. Try now to type *man report_timing* and *man report_power* in the command line.

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.

Final comments

◊ **REMEMBER: close the x2go session when you are done.**

To save a copy of your work you can:

- Send it by e-mail using the available browser and using your student account
- Use your disk space in your Portale della didattica disk
- Download it on your personal PC using scp from command line or WinScp or FileZilla

◊ **Please note: For this lab nothing is to be submitted, however, in all the next labs you will use these systems and commands, so you are expected to know them very well.**