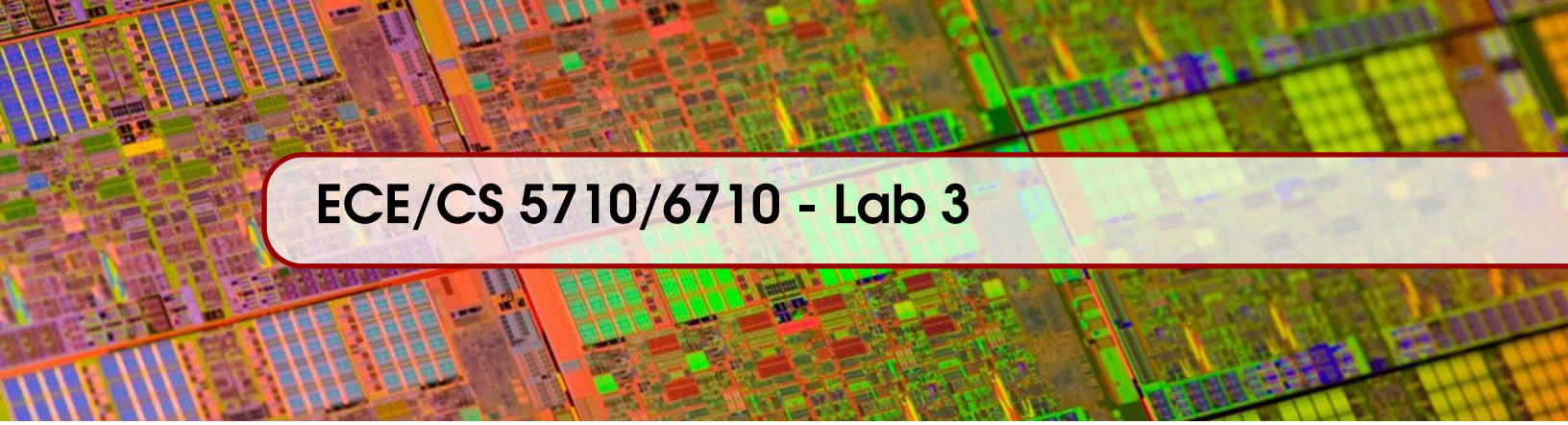


ECE/CS 5710/6710 - Lab 3

Synthesis of a MIPS Processor

Pre-lab Assignment: Check for the date on Canvas.
Lab Report: Check for the date on Canvas.

1 Objective

This lab presents the main steps for performing the RTL/logic synthesis of an 8-bit MIPS microprocessor from the *CMOS VLSI Design* book, using the Skywater 130 *nm* design kit. This lab details the typical steps of the logic synthesis in VLSI design. The tool you will use is Synopsys Design Compiler®, which is widely used in industry and academia.

2 Pre-lab Assignment

Answer the following questions and submit a **.pdf** file through Canvas:

1. What RTL stands for? Explain briefly the different levels of abstraction in VLSI design.
2. Explain briefly what is logic synthesis in VLSI design.
3. What is a timing library and what does it contain? Why do you need it for the synthesis flow?

3 Directory Organization

For the logic synthesis, it will be simplest to work from a *design_compiler* folder. This folder has been added to the git repo. If it does not exist, the repo can be updated with:

```
git pull
```

This folder contains different subdirectories as follows:

- *DDC*: design database, where you will save your design after the different steps of the synthesis.
- *RPT*: report files (area, gate etc.) created by the tool.
- *SDC*: system design constraint file created by the tool.
- *SCRIPTS*: contains a TCL script to automatize the synthesis step (to be used at the end of the lab).
- *SDF*: Timing files created by the tool for functional verification.

4 Lab Assignment

4.1 Starting Synopsys Design Compiler®

Launch the Synopsys Design Compiler® GUI by going in your *design_compiler* folder and doing:

```
design_vision
```

The *design_compiler* directory contains an hidden file named *.synopsys_dc.setup*. This file is read every time Synopsys DC starts. It specify what is the target library (the library that defines the area/timing/power characteristics of the physical logic gates) so in our case the standard cells from Skywater 130 *nm*).

When Synopsys DC is opened, you will obtain a window as shown in Fig 1. The **Console terminal** is used for entering Design Compiler (DC) commands, to source Tcl scripts or to run Linux commands. Those things can also be done through the **command line** through a script. The lab will show you how to perform the tasks with the GUI by hand, but it can be automatized by running command lines. In addition, not all commands are available in the GUI. For more complex and repetitive synthesis tasks, it is highly recommended to use scripts. When you will perform actions on the GUI, the equivalent commands will be recorded and stored in the *command.log* file in the *design_compiler* directory. The equivalent commands will also be given in this tutorial. To obtain some help about a command, you can run in the console terminal or the command line:

```
man your_command
```

The console terminal and the *Log* view contain the trace of the execution of the command. The *History* tab contains the history of all executed commands in the session.

4.2 Analyzing the Verilog RTL Files

The analysis phase will compile the Verilog or VHDL source files and ensure that they are synthesizable (that means that they only contains statement that have meaning for synthesis, i.e. that can be mapped into a hardware circuit). To analyze the files, do:

1. *File -> Analyze...*
2. Click on the *Add...* button, as shown in Fig 2 to select the Verilog file of the 8-bit microprocessor (*mips.sv*). It is located in the *HDL/RTL* folder.
3. Click on OK.

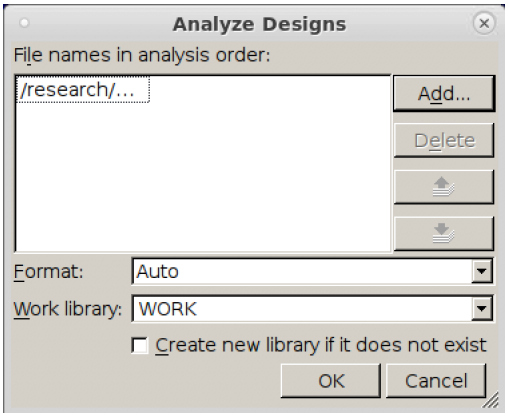


Figure 2: Analyzing the Design

Equivalent DC command:

```
analyze -library WORK -format sverilog {.../HDL/RTL/mips.v}
```

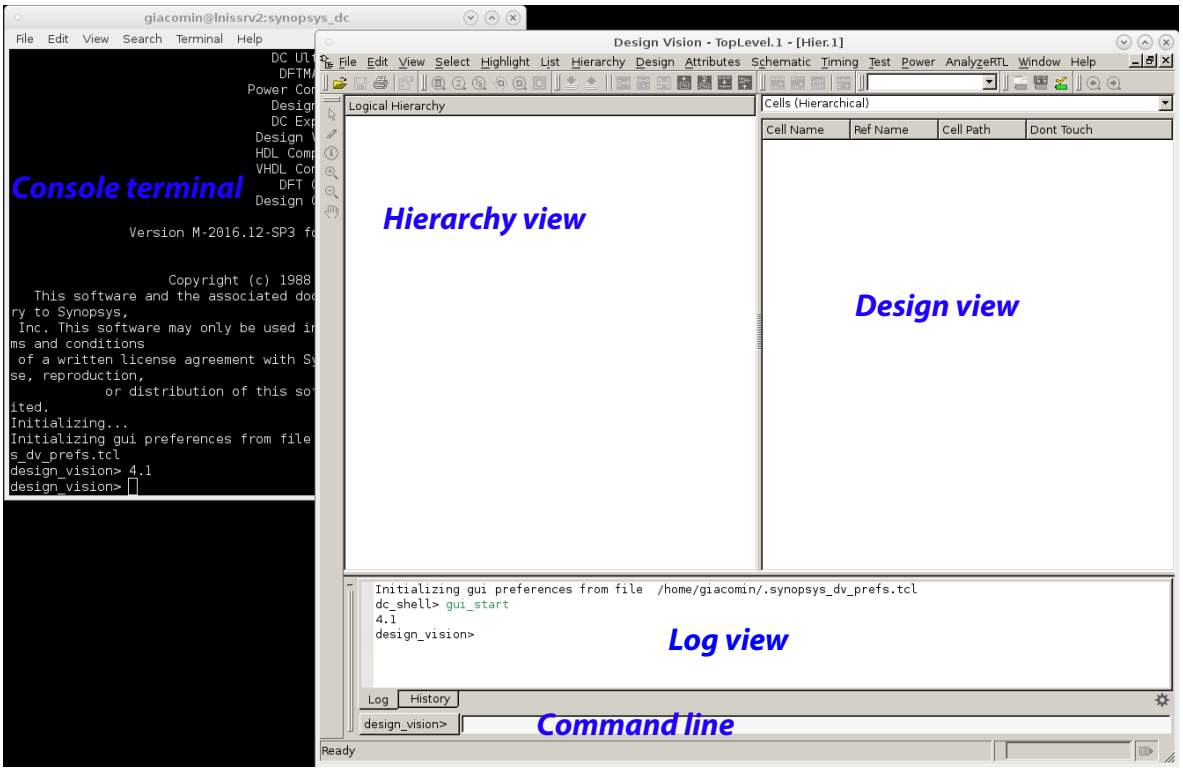


Figure 1: Synopsys DC window organization

4.3 Elaborating the Design

This phase will perform a pre-synthesis of the netlist. It will basically identify the logic functions and the different registers (latches and flip-flops) that will be inferred. To elaborate the design, do:

1. *File -> Elaborate...*
2. Select the top module of the verilog netlist: *Processor*, as shown in Fig 2.
3. Click on OK.

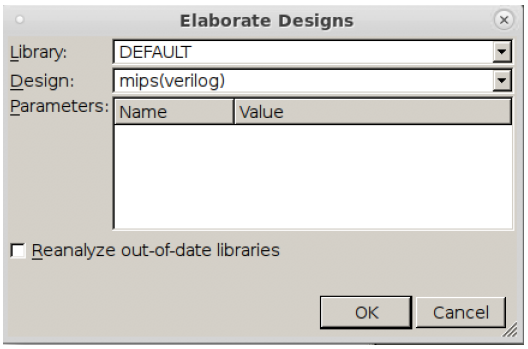


Figure 3: Elaborating the design

Equivalent DC command:

```
elaborate mips -architecture verilog -library DEFAULT
```

On the **Log window**, you can see the inferred registers for the different modules, as shown in Fig 4, their size (width) and their Set/Reset signals (AR/AS: asynchronous reset/set, SR/SS: synchronous reset/set).

```
Inferred memory devices in process
        in routine flopenr_WIDTH8 line 434 in file
                '/research/ece/lnis/USERS/giacomin/tsmc_180nm/labs/HDL/RTL/mips.sv'.
================================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
================================================================================
|       q_reg         | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
================================================================================
```

Figure 4: Some of the inferred registers

On the **Hierarchy view**, you can now see the inferred components and the associated standard logic cells. Select the *andblock* from the *alunit* module, as shown in Fig. 5 and select *Cells (All)* at the top of the design view. It is also possible to display the elaborated schematics. Lets do it for the *andblock*. To do so, make sure that the *andblock* instance is selected in the hierarchy view and click on the *Create Design Schematic* icon, as depicted in Fig. 5. A new window containing the *andblock* schematic now opens.

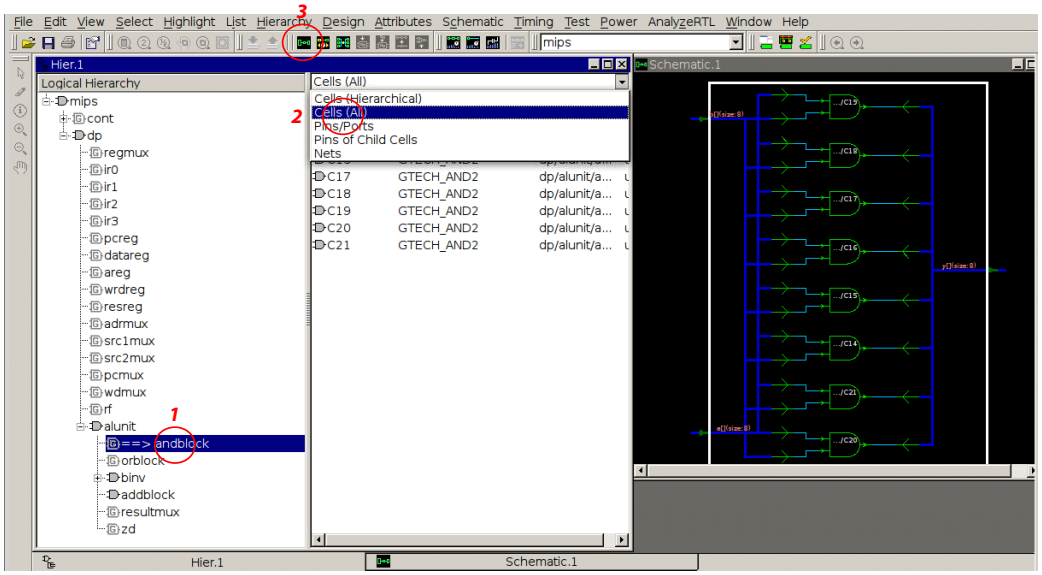


Figure 5: Displaying all cells and creating the schematic of the 8-bit AND.

R The GTECH_XXX elements are denoting generic combinational logic functions which are not mapped to the standard cell library yet. In other modules, you can find names starting with a * (e.g.*ADD, *SUB etc.) which denote more complex generic combinational operators. The reference name **logic0** or **logic1** denotes a wire tied to 0 (G_{ND}) or tied to 1 (V_{DD}), respectively and the reference name **SEQGEN** denotes a flip-flop register.

4.4 Saving and Restoring the Design

At this step, it is recommended to save the elaborated design. To do so:

1. *File -> Save As...*
2. Save the design in the *DDC* folder, as shown in Fig 6 with the name: *mips_elab*.
3. Select the DDC format. This is a binary format for storing the designs.
4. Check the box Save all designs in hierarchy.
5. Click on OK.

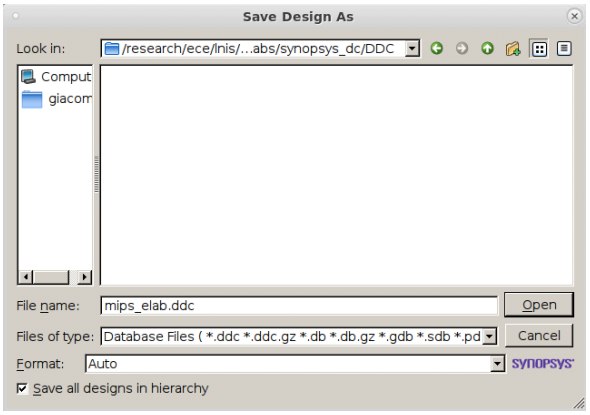


Figure 6: Saving the elaborated design

Equivalent DC command:

```
write -hierarchy -format ddc -output DDC/mips_elab.ddc
```

When starting Synopsys DC again, it will be possible to directly load the elaborated design (if you want to change the constraints for instance or work on your project later) without re doing the previous steps by doing: *File -> Read...* and selecting the appropriate design file in the *DDC* folder.

Equivalent DC command:

```
read_ddc DDC/mips_elab.ddc
or
read_file -format ddc DDC/mips_elab.ddc
```

4.5 Linking the Design

After elaborating the design, you may have noticed some warnings that means that for some components, no Verilog design entity has been defined:

```
Warning: Design 'ExampleRocketSystem' has '47' unresolved references. For more detailed information, use the "link" command. (UID-341)
```

Figure 7: Unsolved references

The link operation defines the link libraries (the libraries that define the area/timing/power characteristics of other *Intellectual Property* (IP) components, such as memories, analog blocks, *etc.*). To do so:

1. *File -> Link Design...*
2. For your convenience, the linking libraries are already loaded.
3. Click on OK.

Equivalent DC command:

```
link
```

4.6 Specifying the Design Constraints

During synthesis, constraints are typically timing and area. The design here is synchronous so a constraint has to be defined on the clock. The area constraint will ensure to get the smallest area possible. Synopsys DC will always consider the timing constraint in first and then will try to meet the area constraints if there are any.
To define the clock:

1. Select the *mips* instance in the hierarchy view (step 1 in Fig. 8) and open its schematic view with the Create schematic icon (step 2). This will open a new tab with the symbol view of the component.
2. Select the *clk* pin. It should be on the left side, at the top (step 3).
3. Go to: *Attribute -> Specify Clock...* (step 4)
4. Define a clock period of 30ns, as shown in Fig 8 and a pulse width of 15ns (the unit is defined in the *.db* file of your standard library. In our case, the unit is *ns*). The clock waveform should be displayed if the clock specification is correct. The selected *clk* pin is used as the port name.

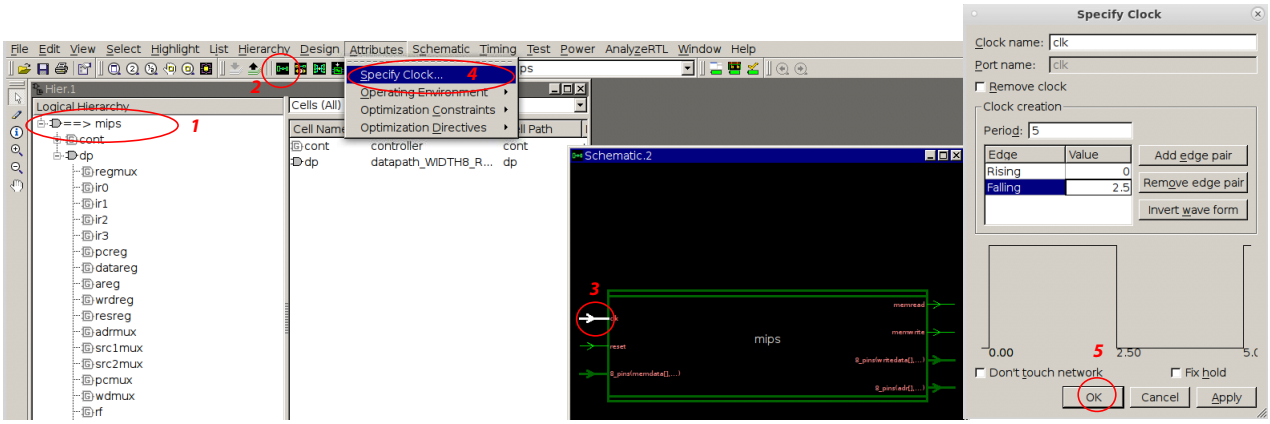


Figure 8: Clock generation

5. Click on OK (step 5).

Equivalent DC command:

```
create_clock -name "clk" -period 30 -waveform { 0 15 } { clk }
```

To define the area constraint:

1. *Attributes -> Optimization Constraints -> Design Constraints...*
2. Specify a Max area of 0 as depicted in Fig 9. This is not realistic but this will tell DC to minimize the area without too much computation effort.
3. Click on OK.

Equivalent DC command:

```
set_max_area 0
```

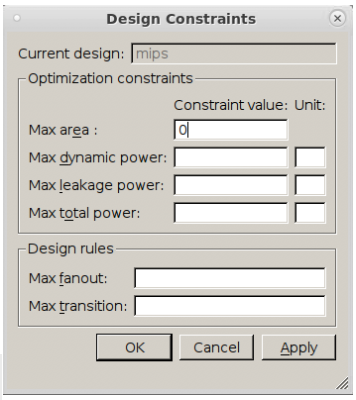


Figure 9: Defining area constraint

4.7 Mapping the Design

This phase (also called compilation phase) is technology dependent. It will assign the logic gates from the standard logic cell library defined in the .db file from the foundry to the generic gates obtained after the elaborated design by meeting the timing and area constraints.

To map the design:

1. *Design -> Compile Design...*
2. Do not change the default settings. The map effort is the amount of CPU time dedicated to select the proper gate from the cell library. A high effort will take more time. The area effort is the amount of CPU time spent during the area recovery phase (when DC try to meet the area constraint without breaking the timing constraint).
3. Click on OK.

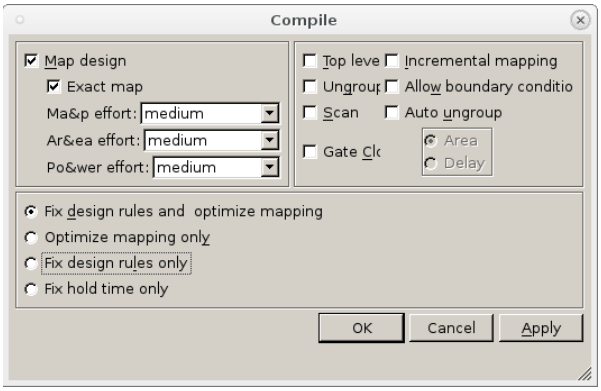


Figure 10: Compiling the design

Equivalent DC command:

```
compile -exact_map -map_effort medium -area_effort medium
```

Once this phase is done, save the mapped design in the *DDC* folder with the name: *mips_mapped*.
Equivalent DC command:

```
write -hierarchy -format ddc -output DDC/mips_mapped.ddc
```

R As you did previously for the *andblock* of the *alunit*, you can display its schematic. This time, it will use the standard cells from the Skywater 130 *nm* library and not the generic combinational logic functions (GTECH_XXX elements) since your design is now mapped.

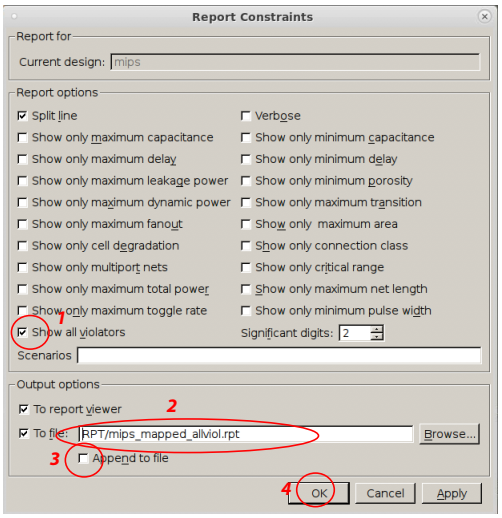
4.8 Generating the Reports

Once the mapping is done, many kind of reports can be generated. The DC command *help report** can give you a list of all the available report commands. Now, we are going to consider only some useful reports.

4.8.1 Reporting all Violated Constraints

This step will report if you design meets the timing and area constraints you specified. To do so:

1. Go to: *Design -> Report Constraints...*
2. As in Fig. 11, check *Show all violators* and *To report viewer* (that means to the console). Uncheck *Append to file*. In that way, if the file was already existing, it would overwritten its content.
3. Check *To file* and specify the path: *RPT/mips_mapped_allviol.rpt*
4. Click on OK.
5. On the console terminal, you should only see an area constraint violation. This is normal since the area has been set (artificially) to 0 and the clock period constraint is large.



Equivalent DC command:

```
report_constraint -nosplit -all_violators > RPT/mips_mapped_allviol.rpt
```

4.8.2 Reporting the Area

1. Go to: *Design -> Report Area...*
2. Select the parameters as in Fig. 12.
3. Check *To file* and specify the path: *RPT/mips_mapped_area.rpt*
4. Click on OK.

Equivalent DC command:

```
report_area -hierarchy -designware >
RPT/mips_mapped_area.rpt
```

In the log view, you can now see the area report with the total area, the combinational area, the noncombinational area (registers and RAM) *etc*. The area is generally provided in square microns (depending on the cell library). The net interconnect area is, for this particular cell library, not defined as the supplied wire load models from the timing library does not provide any area values. This is why the total area is said to be undefined so for this step, consider the total cell area as your design area. The actual net area will be known after the back-end flow in the next lab.

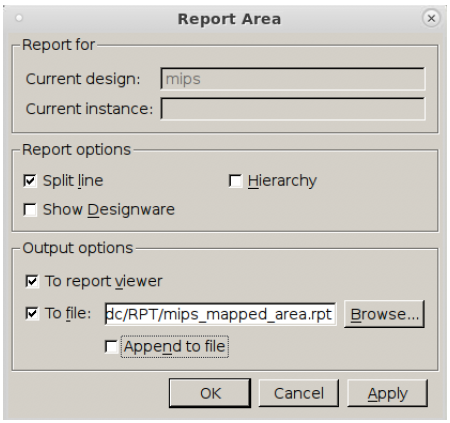


Figure 12: Reporting the design area

4.8.3 Reporting the Timings

This report will give an important timing: the critical path. This is the path in your design which takes the longest time to be propagated.

1. Go to: *Timing -> Report Timing Path...*
2. Select the parameters as in Fig. 13.
3. Check *To file* and specify the path: *RPT/mips_mapped_timing.rpt*
4. Click on OK.

Equivalent DC command:

```
report_timing >
RPT/mips_mapped_timing.rpt
```

A new window should open where you can see the slack of your mapped circuit. The slack is the difference between the required time and the longest arrival time (which is the critical path of your circuit). A negative slack value indicates a timing violation and a positive slack indicates that your design meets the timing constraints.

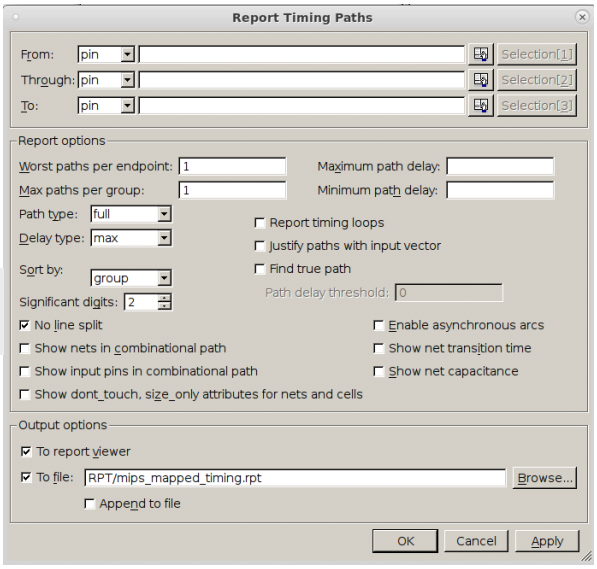


Figure 13: Reporting the critical path

The timing analysis only consider approximate values for the interconnection delays. Accurate values will be only known after the place and route step (in the next lab).

4.8.4 Reporting the Power

This report will give the power consumption of your circuit, which is an important parameter in VLSI design.

1. Go to: *Design -> Report Power...*
2. Select the parameters as in Fig. 14.
3. Check *To file* and specify the path: *RPT/mips_mapped_power.rpt*
4. Click on OK.

Equivalent DC command:

```
report_power -nosplit
-analysis_effort low >
RPT/mips_mapped_power.rpt
```

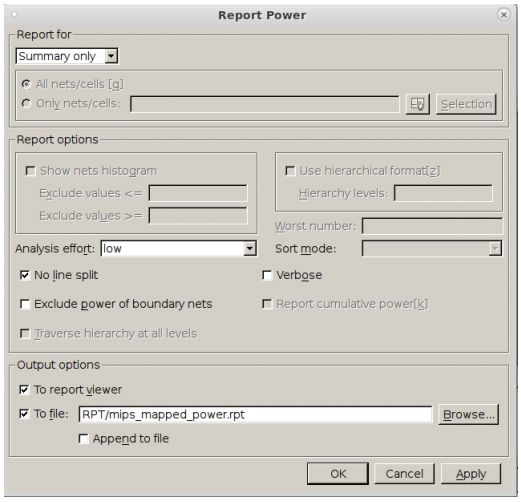


Figure 14: Reporting the power

4.9 Exporting the Synthesized Design

This part will generate all the necessary files for the next step: Place & Route.

4.9.1 Generating the Verilog Gate-Level Netlist

This step will generate the verilog gate-level netlist, i.e, the netlist of your design which only contains standard cell from the targeted technology library and no more behavioral verilog.

1. Go to: *File -> Save As...*
2. Select the parameters as in Fig. 15. Don't forget to select the appropriate format and the appropriate path.
3. Click on OK.

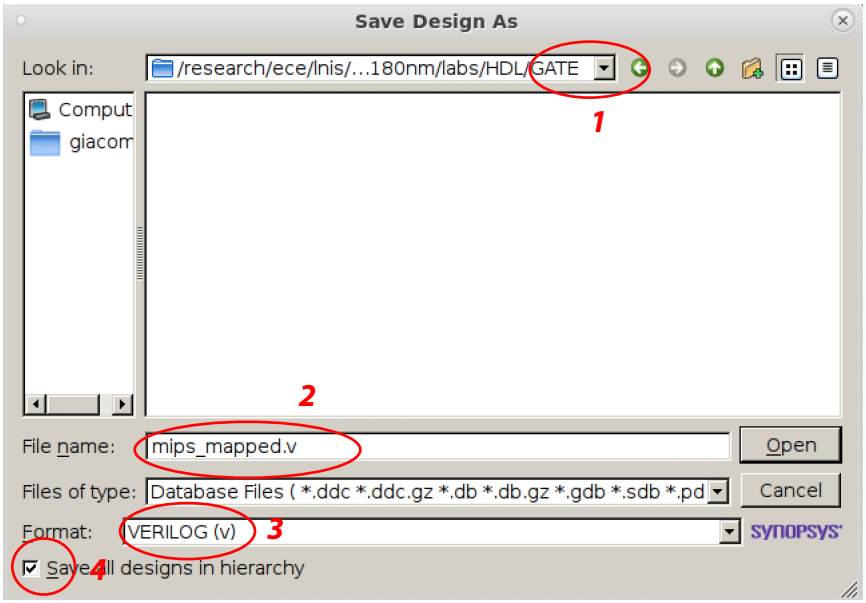


Figure 15: Exporting the verilog gate-level netlist

Equivalent DC command:

```
write -format verilog -hierarchy -output ../HDL/GATE/mips_mapped.v
```

4.9.2 Generating the SDF timing file

This step will generate the *Standard Delay Format* (SDF) file which includes the gate delays. It is then used for post-synthesis functional verification (with Modelsim® for instance). To do so, you need to use the console terminal:

Equivalent DC command:

```
write_sdf -version 2.1 SDF/mips_mapped.sdf
```

4.9.3 Generating the Constraint File

This step will generate the constraints file which specifies the design constraints. The *sdc* format can latter be read by the Place & Route tools (like synopsys PrimeTime® or Cadence Innovus®).

Equivalent DC command:

```
write_sdc -nosplit SDC/mips_mapped.sdc
```

4.9.4 Using a Tcl Script

When designs are more complex or if you want to repeat the previous tasks, it is recommended to use scripts and to run those from the Synopsys DC command line. Synopsys Design Compiler supports the Tcl language for building scripts. All the commands specified in this manuscript can be grouped into a single Tcl script file which is in your folder: *synthesis.tcl*. You can now perform the same steps as before by modifying the clock constraint (and use the script to go faster). To do so:

1. Open the *synthesis.tcl* file in your *design_compiler/SCRIPTS* folder and try to find the line where the clock is specified and modify its value and save the script.
2. In the Synopsys DC terminal, run:

```
source SCRIPTS/synthesis.tcl
```

Assignment

1. Optimize your design specifying a tighter clock constraint (do not forget that your timing slack has to remain positive or equals to 0). To do so, you need modify the *source synthesis.tcl* and run it.
2. Provide the screenshots of the critical path, the area and the power report of your final optimized design. Also report the number of combinational and sequential cells.
3. Report the smallest clock period you achieved while keeping a positive slack.

Checkpoint

Please call an assistant and show him that you generated the *.sdc* file and the verilog gate-level netlist file correctly.

5 Assignment and Checkpoint Summary

Write a report and answer the assignment. Do not forget to validate the checkpoint by an assistant before the end of the lab.