

ISRAEL INSTITUTE OF TECHNOLOGY - TECHNION
ANDREW AND ERNA VITERBI FACULTY OF ELECTRICAL ENGINEERING

Introduction to VLSI

Computer Exercise 2

Behavioral Synthesis and Back-end Design Flow

Last update: November 20, 2018

Submission in pairs only until 24/1/2019

If you have questions please contact
Nicolás Wainstein - nicolasw@campus.technion.ac.il



Contents

1	Introduction	1
1.1	Submission	1
1.2	Design Flow	1
1.2.1	Behavioral Synthesis	2
1.2.2	Physical Synthesis	3
2	Behavioral Synthesis	5
2.1	Setting up Lab 2	5
2.2	MIPS System Verilog	5
2.3	Setting up Design Vision	5
2.4	Import files, Analyze and Elaborate	6
2.4.1	Analyze	6
2.4.2	Elaborate	6
2.4.3	Check Design	7
2.5	Compiling	7
2.5.1	Compilation	7
2.5.2	Constraints	8
2.5.3	Reports and output netlist	9
2.6	First synthesis	10
2.7	Optimization	11
2.8	Fixing maximum delay problems	12
2.9	Fixing minimum delay problems	12
2.10	Final synthesis	13
3	Physical Synthesis	13
3.1	Setting up Innovus	13
3.2	Adding I/O pads to the netlist	14
3.3	Starting Innovus	15
3.4	Floorplan	17
3.5	Power Grid	18
3.5.1	Rings	18
3.5.2	Stripes	19
3.6	Placement	19
3.7	Static Timing Analysis I	20
3.8	Clock Tree Synthesis (CTS)	21
3.9	Static Timing Analysis II	22
3.10	Fill spaces	22
3.11	Routing	22
3.11.1	Power Routing—Connect Stripes to Rings	22
3.11.2	Signal Routing	22
3.12	Power Analysis	22
3.13	Timing Analysis III	23
3.14	Bonus Competition	23
3.15	Rest of the flow	24

1 Introduction

In this lab we will implement complete design flow of an 8-bit MIPS microprocessor using a given RTL description of it. Starting from behavioral synthesis we will follow the whole design flow up to a final layout, covering delay/power/area optimization, floorplaning, partitioning, place and route and design verification.

1.1 Submission

You are intended to submit electronically (Moodle) a zip file named with both students ID number (*i.g.*, ID1_ID2.zip) including:

- A **PDF report** that includes all the figures/simulations/questions that are asked in each section. There's no need to extend in your answers. Be precise and concise. Try to include print screens that clearly explain the issue, for example zoom in to the relevant section.
- The **output report** of Design Vision (see sec. 2.5.3) for initial case and final case.
- **Final** *constraints.tcl* file (see sec. 2.5.2).
- Power and timing reports from Synopsys IC Compiler.

Please, in the first page of your report write down your names, IDs, your lab account and the amount of hours you invested in this work.

1.2 Design Flow

This is background material of the design flow taken from David Harris CMOS VLSI 4th Ed.

A general design flow is shown in Fig. 1. Design starts at the behavioral level and then proceeds to the structural level (gates and registers). This step is called behavioral or Register Transfer Level (RTL) synthesis because the designs are captured at the RTL (memory elements and logic) level in a Hardware Description Language (HDL). The description is then transformed to a physical description suitable for chip fabrication. This step is called physical synthesis (or layout generation). Normally, the synthesis steps are automated, even if guided by human judgment. The verification steps are also shown.

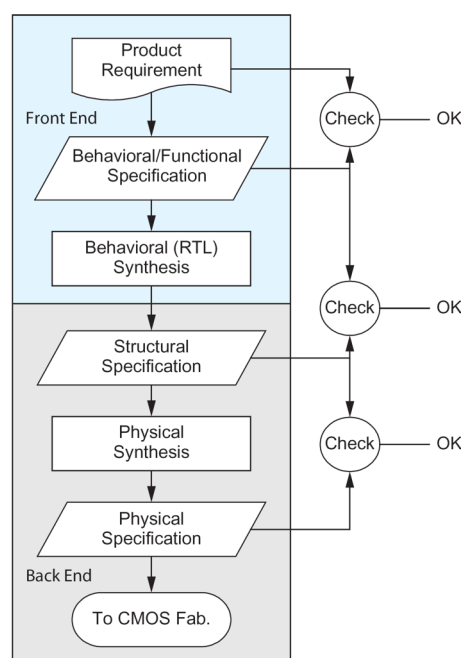


Figure 1: Generalized Design Flow

In Figure 1, the design has been partitioned into the **front end** stage at the behavioral level and the **back end** at the structural and physical levels. This is important because it illustrates a partitioning that is used to build Application Specific Integrated Circuits (ASICs). In an ASIC, the design can be developed at the HDL level and then passed to a company that completes the transition to an actual chip. In this way, the original design company does not have to invest the personnel or tools required to translate an HDL specification into a physical chip.

While it works for moderately complex designs, the interaction between logic and layout is so important in more demanding circuits that such a flow becomes a schedule risk. Primarily, this occurs because the iteration time between logic design and physical placement takes too long when spread over two organizations. Multiple iterations are necessary because the *prelayout* timing estimates available to the HDL designer correlate poorly with the true *postlayout* timing because wire lengths are unpredictable before layout.

1.2.1 Behavioral Synthesis

Design and Verification

The design starts with a specification, which might be a text description or a description in a system specification language. The designer(s) convert this to an RTL behavioral description in an HDL such as Verilog, or VHDL. A set of test benches are then constructed and the HDL is simulated to verify the correct behavior as defined by the specification and product requirements.

RTL Synthesis

Synthesis involves converting the RTL to generic gates and registers, optimizing the logic to improve speed and area, and mapping the generic gates to a standard cell library. Other steps involved at this stage are state machine decomposition, datapath optimization, and power optimization.

Library Mapping

Library mapping takes a generic HDL gate-level description and translates it to a netlist that specifies particular gates in the target library (Tower TSL 018 in our case). This stage also maps predefined blocks such as memories to their appropriate descriptions.

Formal Verification

The next step is prove that the structural netlist performs the same function as the original behavioral HDL. Ideally, the netlist would be correct-by-construction, but ambiguities in HDLs sometimes cause the synthesizer to produce incorrect netlists from poorly written behavioral code. One verification strategy is to rerun the logic test benches and check that they produce exactly the same output for the behavioral and structural descriptions.

Static Timing Analysis (STA)

At the behavioral level, clock cycle time is an abstract notion, but at the structural level, an actual cycle time has to be met by a particular set of gates. A *timing analyzer* is used to verify the timing. This is a critical analytical tool in the arsenal of the modern CMOS digital designer.

Static timing analysis runs quickly and exhaustively evaluates all timing paths. The inputs to the timing analyzer are derived from the basic timing of the library gates due to intrinsic gate delays and routing loads that can be either estimated statistically or derived from *floorplanning* data. Timing analyzers check for both max-delay (will all FFs meet their setup time at the required cycle time?) and min-delay (will any FF violate its hold time?).

Power Analysis

Power consumption depends on the activity factors of the gates, which in turn depends on the inputs the chip receives. Power analysis can be performed for a particular set of test vectors by running a simulator and evaluating the total capacitance switched at each clock transition at each node. At this stage, if the power is too high, the design must return to the architectural level to rethink the solution.

1.2.2 Physical Synthesis

Layout generation is the last step in the process of turning a design into a manufacturable database. It transforms a design from the structural to the physical domain.

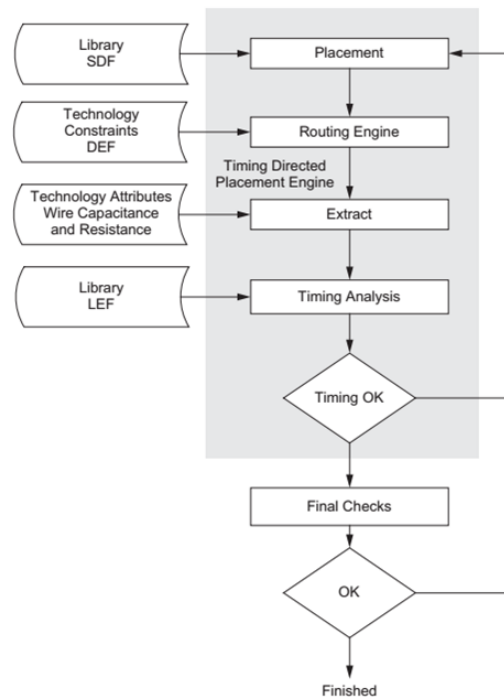


Figure 2: Placement Flow

Floorplanning

Increasingly a manual floorplanning step is required in the placement process. Rather than place a design “flat” (i.e., all cells at the same level of the hierarchy), modules are clustered in areas that are dictated by the need to communicate with other modules.

Placement

The key to automation of standard cell layouts is the use of constant-height, variable-width standard cells that are arrayed in rows across a chip. The objective of a simple placement algorithm is to minimize the length of wires. In *timing-driven placement*, the cost of wires is weighted to meet timing constraints. At the end of the placement phase, the cells have been fixed in position in the overall array.

Clock-Tree Routing

Central to modern high-speed designs is the clock distribution strategy. To minimize skew, it is often best to preroute the clock and its buffers before the main logic placement and routing is completed. This task is performed with a clock tree router.

Routing

After placement of cells, the signal nets in the circuit need to be routed. Routing is normally divided into two steps: *global* and *detailed* routing.

A global router abstracts the routing problem to a notional set of abutting channels that cover the chip surface through which wires are routed. Routes are added to channels according to a cost function. The detailed router places the actual geometry required to complete signal connections.

Parasitic Extraction

The parasitic extractor generates a file that describes the R's and C's associated with all nets in the layout. The extractor uses another technology file defining the interlayer capacitances and layer resistances. The capacitance extractor can be a 2D, 2.5D, or 3D extractor.

Timing Analysis

Static timing analysis is now rerun with the actual routing loads placed on the gates. **This is usually the bottleneck in the design process** as the full reality of a physical realization is apparent. Multiple iterations of synthesis and placement & routing are usually necessary to converge on timing requirements.

Noise, V_{DD} Drop, and Electromigration Analysis

Analyses are now run to check noise, IR drop in supply lines, and electromigration limits. Noise analysis is run to evaluate crosstalk due to interlayer routing capacitance.

Power Analysis

Power estimation can be repeated for the extracted design now that real wire capacitance are available. Similar techniques to those used during RTL synthesis are used.

===== End of introductory information =====

2 Behavioral Synthesis

In this section, you will use Synopsys Design Vision (DV) to synthesize the MIPS module into a gate-level netlist. This is the industry-standard logic synthesis tool. The tool synthesizes VHDL or Verilog hardware descriptions, creating a gate-level description of the design. Although the tool has a graphic interface, every command we execute in the GUI will appear at the Terminal window (in fact we can manage the tool by using only the Terminal).

2.1 Setting up Lab 2

To prepare the Lab 2 environment, download the *lab2* folder from the course website. Create a folder named *lab2* in your home directory. Copy the entire folder in your own *lab2* folder. The folder contains setup files, the MIPS Verilog code, and other scripts. Inside your *lab2* directory you should have the following files:

- `mips.sv`: The MIPS SystemVerilog code.
- `constraints.tcl`: Constraint file for behavioral synthesis.
- `cts.src`: Clock tree synthesis script.

2.2 MIPS System Verilog

Open the 8-bit MIPS microprocessor System Verilog code ¹ (`mips.sv`). A full description of the MIPS μ P can be found in Chapter 1 of the CMOS VLSI Design book. Go through the code, observe that there are two clearly differentiated modules: the *datapath* (`dp`) and the *controller* (`cont`). Each of them is composed of other several submodules. Make sure you understand the implementation and compare it to the FSM shown in the CMOS VLSI Design Book.

(*Warm up questions*) Go through the code, observe and **answer**:

1. Which are the two big differentiated modules? Which are the inputs and outputs of each module?
2. How many different types of FF are defined?

2.3 Setting up Design Vision

To prepare the synthesizer environment you will create a folder inside *lab2* called *design_syn*. The folder will contain setup files², the MIPS Verilog code, and other scripts. Every time you work with the synthesizer, **you must work on this folder (i.e. run the program from inside this folder)**. Also, create a folder named WORK inside *design_syn*, the compiler will save the design files inside that folder. Copy the setup files to *design_syn*, in a Terminal write the following commands,

```
cp /users/iit/synopsys/ts1018_20034_syn2/.syn* design_syn
mkdir design_syn/WORK/
```

Copy the MIPS code to this folder:

```
cp mips.sv ~/lab2/design_syn/
```

Go to `design_syn` folder and type the command `design_vision` to open the tool. Alternatively, you can type the command `dc_shell -gui`. A window as in Fig. 3 will open. In the lower part of the window we can appreciate a *console* in which we can type commands and observe their outputs.

¹Code provided by David Harris (cmosvlsi.com) and modified at Stanford.

²Design Vision uses TYPICAL library. This means that (in our case) we cannot expect timing, area and power estimates of the two tools to match 100%. In real life, exercise caution!

As mentioned before, everything that appears at the console will also appear at the Terminal.

In the left side of the main window we have the hierarchy navigation panel. After loading our HDL file you will observe the hierarchical structure in this panel and you will be able to navigate through the design by clicking on the different blocks and cells.

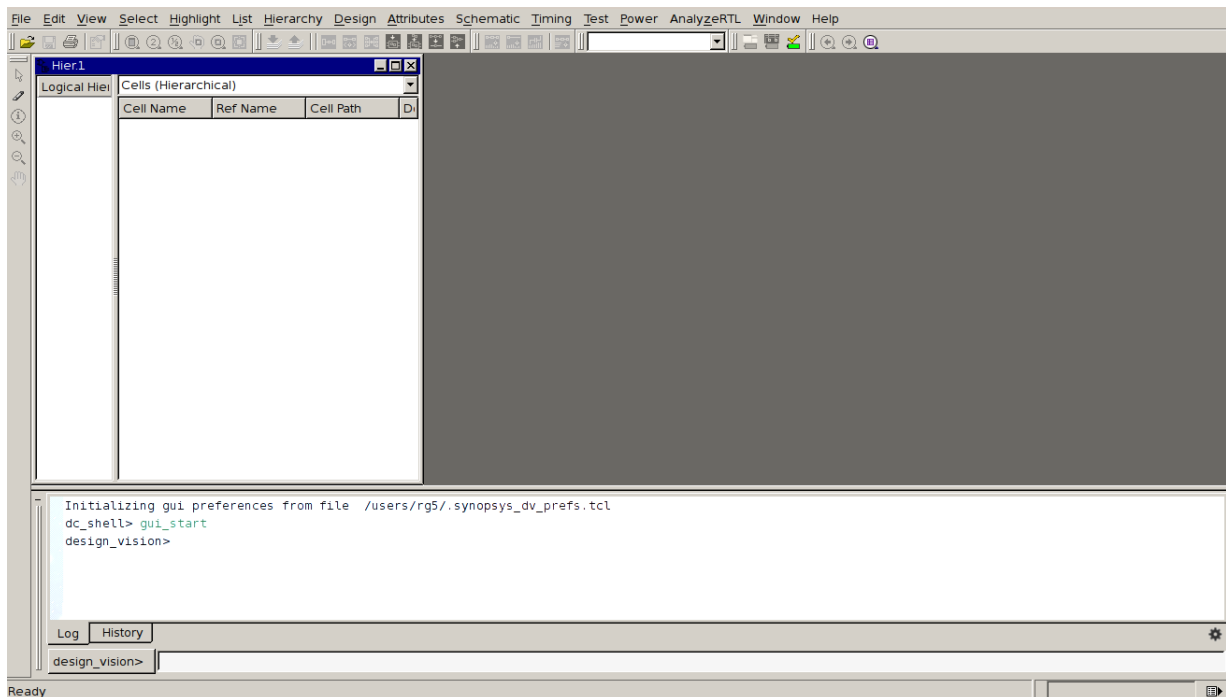


Figure 3: Design Vision main window

2.4 Import files, Analyze and Elaborate

2.4.1 Analyze

The analysis command checks your HDL design for proper syntax and synthesizable logic, and then translates this design into an intermediate format inside the specified WORK directory. In the Design Vision window go to *File* → *Analyze...*. In the window that opens, click *Add...* and select the file *mips.sv*. In the field *Format* select *AUTO* and click the check box below (*Create new library if it does not exist*). Click *OK*.

2.4.2 Elaborate

Elaboration then takes this intermediate representation and begins the task of turning your RTL description into actual hardware. This includes replacing HDL arithmetic operators with synthesized operators, determining correct bus size, and inferring the presence of state elements such as latches and flip-flops. In the Design Vision window go to *File* → *Elaborate...*. Select WORK in the *Library* field, *mips(verilog)* in the *Design* field³ and click the check box *Reanalyze out-of-date libraries*. Click *OK*.

Observe how the hierarchy navigation panel shows the MIPS design. We can appreciate the two top hierarchy modules: dp and cont. Beneath the *Logical Hierarchy* panel, right-click on *mips* and select *Schematic View*. A new window will open with a schematic description of the MIPS as in Fig. 4. You can navigate through the schematic by double clicking over the desired module/block. Moreover, you can zoom in and out with the mouse scroll and navigate through hierarchies with the buttons

³*mips* is the top level module name in your code.


 at the Toolbar. **Observe that the gates are from a generic library and not from a particular technology.**



Figure 4: Schematic View of the design

2.4.3 Check Design

The check design command checks the internal representation of your design and corrects certain design problems. Some of the errors you may encounter when running check design include unconnected ports, constant-valued ports, cells with no input or output pins, mismatches between a cell and its reference, multiple driver nets, etc. WARNINGS are not an issue, but you will not be able to synthesize your design while having ERRORS. As a best practice you should always minimize as many WARNINGS as possible, and understand any remaining WARNINGS⁴. To check your design go to *Design* → *Check Design*.

To save the design state for the first time go to *File* → *Save As...*, give a name and click *Save*. Then you can save by clicking the diskette button or *File* → *Save*. To exit Design Vision go to *File* → *Exit* or type *quit* in the console.

2.5 Compiling

Optimizing (compiling) is the step in the synthesis process that attempts to implement a combination of library cells that meets the functional, speed, and area requirements of your design. Optimization transforms a design into a technology-specific circuit based on the **attributes** and **constraints** you place on the design. During optimization, the Design Vision tool from Synopsys attempts to meet the constraints you have set on the design. **This is the main step in the behavioral synthesis process.**

2.5.1 Compilation

The compilation process modifies the logic in a netlist. Compilation uses cells from the technology library in an attempt to meet specified constraints. Design Vision performs the following types of optimizations:

I. Combinational optimization,

- (a) Technology-independent optimization, which operates at the logic level. Represents the gates as a set of Boolean logic equations.

⁴Ignore all LINT warnings

- (b) Mapping, during which it selects components from the technology library to implement the logic structure. Tries different logic combinations, using only components that approach the defined speed and area goals.
- (c) Technology-specific optimization, which operates at the gate-level, attempts to meet your timing and area constraints.

II. **Sequential optimization.** Maps sequential cells to cells in the library, adds information about the delay through the combinational logic and I/O pads. Optimizes timing-critical sequential cells (cells on the critical path).

III. **Local optimizations.** Involves making local changes, where it makes incremental modifications to the design to adjust for timing or area.

2.5.2 Constraints

The synthesizer makes a best effort to synthesize your design while still meeting two types of constraints: *user specified constraints* and *design rule constraints*. User specified constraints can be used to constrain the clock period as well as the arrival of certain input signals, the drive strength of the input signals, and the capacitive load on the output signals (fanout). Design rule constraints are fixed constraints which are specified by the standard cell library.

Some useful and most used constraints are:

- `create_clock -period #period -waveform {rise fall} clkname`. This creates a clock with a `#period` in nanoseconds. Replace `#period` by your desired number and the waveform's `rise` and `fall` to be compliant with the `#period` and **use a 50% duty cycle**. The name `clkname` must be the same as in the Verilog description. This clock will be used to constrain the data paths between flip flops (or latches) in your design. **Important: Whenever you change the period, change the values of the waveform correspondingly.**
- `set_input_delay #delay -clock clkname`. This is the external delay at the input from the clock edge (`#delay` in ns) (*i.e.*, when the input is valid at the input port).
- `set_output_delay #delay -clock clkname`. Idem for output ports.
- `set_max_area #area`. Sets the maximum target area.
- `set_driving_cell -library libraryname -lib_cell cellname`. Sets the driving gate at the input to be `cellname` from `libraryname` library.
- `set_load -library libraryname -lib_cell cellname`. Sets the output load cell from a certain library. Instead of a cell, a defined capacitance can be defined to be the load.
- `set_max_delay #delay [-rise | -fall] [-from list] [-to list]`. Sets the maximum delay from a certain net to another specific net.

The clock period constraint is one of the most important constraints you must consider because it must be set carefully. If the period is unrealistically small, then the tools will spend forever trying to meet timing and ultimately fail. If the period is too large, then the tools will have no trouble but you will get a very conservative implementation.

The constraints can be loaded from a `tcl` script (which is also provided in the lab folder). Open the file `constraints.tcl` with a text editor (Notepad++ or gedit or another one of your preference) and observe its structure. Parameters are defined at the beginning, constraints at the center and compilation commands, database and reports files at the end.

2.5.3 Reports and output netlist

The script generates several report files and a Verilog netlist:

- `mips.sdc`, contains timing constraints generated as defined.
- `mips.rep`, contains timing and area reports.
- `mips.pow`, contains power reports.
- `mips.v`, Verilog netlist containing the technology dependent optimized gate-level description of our 8-bits MIPS.

In the timing and area reports you will find a list of the critical(s) path(s) and their slack. Observe Fig. 5, at the top there is a detailed description of the report, then the path is described by its starting point and endpoint. The path type describes whether the report is for max/min delay. Below, there is detailed list of every point in this specific path, the increment this point adds to the path (**Incr** column) and the total accumulated delay (under **Path** column).

```
*****
Report : timing
        -path full
        -delay min
        -max_paths 1
Design : mips
Version: L-2016.03-SP5-5
Date   : Tue Aug  8 14:25:12 2017
*****

Operating Conditions: TYPICAL   Library: saed90nm_typ
Wire Load Model Mode: enclosed

Startpoint: dp/ir1/q_reg_1_
            (rising edge-triggered flip-flop clocked by clk)
Endpoint:  dp/ir1/q_reg_1_
            (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: min

Des/Clust/Port      Wire Load Model      Library
-----
mips                16000                saed90nm_typ
flopen_WIDTH8_3     8000                saed90nm_typ

Point              Incr      Path
-----
clock clk (rise edge)          0.00      0.00
clock network delay (ideal)    0.00      0.00
dp/ir1/q_reg_1_/CLK (DFFX1)    0.00      0.00 r
dp/ir1/q_reg_1_/Q (DFFX1)     0.17      0.17 f
dp/ir1/U3/Q (A022X1)          0.11      0.28 f
dp/ir1/q_reg_1_/D (DFFX1)     0.04      0.32 f
data arrival time                                0.32

clock clk (rise edge)          0.00      0.00
clock network delay (ideal)    0.00      0.00
dp/ir1/q_reg_1_/CLK (DFFX1)    0.00      0.00 r
library hold time             -0.01     -0.01
data required time                                -0.01

data required time                                -0.01
data arrival time                                -0.32
-----
slack (MET)                      0.32
```

Figure 5: Example of time report

Moreover, you can use Design Vision to examine various timing data. Go to *Timing* → *Paths Slack*. This option will create a histogram of the worst case timing paths in your design. In the window that opens select *Nworst paths* = 10 and *Max Paths* = 50. In *Delay type* select *min* or *max* to obtain the minimum and maximum delay histograms respectively. Then when selecting a bar in the histogram a list appears at its side. You can right click a specific path and choose *Path Schematic*, a window like Fig. 6 will open, which shows the path in a schematic.

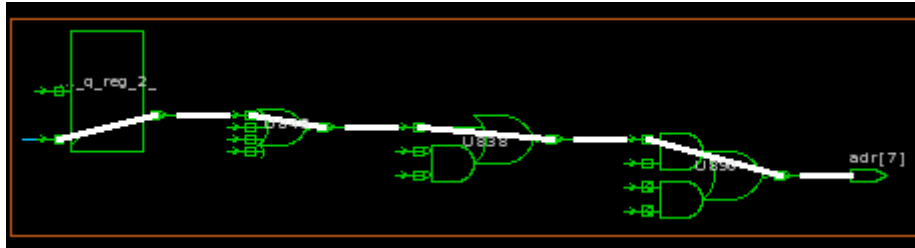


Figure 6: Schematic View of a specific path for maximum delay slack characterization

2.6 First synthesis

Run the script (*without modifying it*) by going to *File* → *Execute Script...* In the explorer window select the file `constraints.tcl` and click *Open*. The constraints will be set and the compilation will begin. After the process finishes, go through the Terminal output to see if there are any violations. *NOTE: The default constraint file is very ambitious—it is designed to create problems that you will first understand and then fix. In the end, you will get a reasonably working chip.*

NOTE II: If after compiling you do not see the logical hierarchy go to *File* → *Read* and click on the *mips.ddc* file.

Go through the report files, observe the results and **answer**:

3. Maximum delay:

- What is the maximum delay? Is there a problem of maximum delay? How much is the *slack*?⁵ How many gates are involved in this path?
- What are the clock-to-out time $t_{clk \rightarrow out}$ and the setup time t_{setup} of the technology?
- Report the critical path.⁶ Which top-level blocks of the design are part of this path? Include a print-screen of the schematic view with this path highlighted.

4. Minimum delay:

- What is the minimum delay? What is the slack?
- What is the hold time t_{hold} of the technology?
- How many gates are involved in the min-delay? What's the particularity of this path?
- Report the critical path for min-delay. Include a print-screen of the schematic view with this path highlighted.

5. Area:

- What is the obtained area? Does it meet the constraint?
- What is the slack? Which constraints influence the area?

⁵Difference between the desired and the obtained

⁶The critical path is the slowest logic path between any two registers, and therefore is the limiting factor preventing a designer from decreasing the clock period constraint (increasing clock speed). In some cases, the critical path may be found between an input and a register, or between a register and an output—does either case apply in our MIPS? Note that the timing report is generated from a purely static worst-case timing analysis

6. Power:

Note that power is split between dynamic and static (leakage), and that dynamic power is further split into “internal” and “switching”. Ignore the latter split—internal and switching numbers are added to produce total dynamic power.

- (a) What is the ratio between dynamic and leakage power? Is this a reasonable result?

7. Submit the initial reports in the zip file.

2.7 Optimization

In this step you will play with the different parameters in order to achieve a balanced design that complies with the following requirements:

- Area < 1850. **Important:** Area is given in terms of a NAND cell. You can approximate its area to $10 \mu m^2$ if needed.
- Critical path slack > 5 ns.

First, *Elaborate* the job as you did in section 2.4.2. Then, change the parameters in the script as described below (**respecting the order, perform one change at a time, incrementally**). Observe carefully the results and how each change affect the output. *Read the questions before performing the changes and the new synthesis.*

- Change the fanout to 4.
- Change period to 10 ns. Change the waveform corresponding to maintain a 50% duty cycle.
- Change area constraint to 2000.
- Change parameter `useUltra` to 1.

IMPORTANT: If your design gets stuck, start from *Elaboration* again. Moreover, if the minimum slack requirement is not achieved with 10 ns change it until you achieve the requirement.

Answer and submit:

- How does each of the aforementioned changes affect the results? Relate to period, area and power.
- Choose a specific logical path between two registers (use *Path Slack* tool). Explain how and why the synthesis tool changes the gates used to implement this path when we change the constraints. Present an example of cells that were changed while you modified the constraint file.
- What can you say about the hierarchical organization while compiling with Ultra and without it? Why does it improve the results?
- Minimum delay:
 - What is the worst slack? Include a print-screen of the schematic view with this path highlighted.
 - What is the best slack? Include a print-screen of the schematic view with this path highlighted.
 - Is it the same critical path as the beginning? Why?
- Maximum delay:
 - What is the worst maximum delay slack? Include a print-screen of the schematic view with this path highlighted.

(b) Why is this the critical path? What's the functionality of this path?

13. **Submit the obtained reports.**

2.8 Fixing maximum delay problems

In this part we discuss how to fix maximum delay problems. *Elaborate* the job as you did in section 2.4.2. Now change the period parameter in the script to be 4 ns, parameter `useUltra` to 1 and fanout to be 4. Execute the script.

After the compilation has finished, in the main window go to *Timing* → *Report Timing Path...* and search for the path (this path is an example, it may or may not constitute a critical path; does it in our MIPS?):

From: `cont_statelog_state_reg_3_/CP`
To: `dp_pcreg_q_reg_7_/D`

Select *Delay Type* to be *max*. Click *OK*.

Observe the obtained report. It shows the timing of a specific path in the schematic. Click in the first point of the list (blue link), a schematic view opens. **Identify the path on the schematic by looking at the list and submit a print-screen of this path.**

Consider that the same path is working under SSSSS corner conditions so that the delay is increased by 20%⁷. Furthermore, the clock may get to the second FF 0.8 ns before it gets to the first FF (potential clock skew that may effectively shorten the clock period), and there is a clock jitter of ± 0.2 ns. *For this part consider that the clocks of both registers are generated by the same source, with an inverter between them that introduces the aforementioned skew.*

Answer:

14. Calculate the maximum delay of this path in the SSSSS corner. Would the circuit (of this path) work under these circumstances?
15. If it does not work, how would you fix the problem? Relate to frequency, skew, changes in the μ Arch, adding FFs.
16. Will these techniques solve the maximum delay problem in all critical paths (worst maximum delay path) in our MIPS? If not, what else may be required?

2.9 Fixing minimum delay problems

As we did in the previous part, search the timing report for the following path:

From: `dp_resreg_q_reg_0_/CP`
To: `dp_pcreg_q_reg_0_/D`

Identify the path on the schematic by looking at the list and submit a print-screen of this path.

Consider that the same path is working under FFFFFF corner conditions so that the delay is decreased by 20%. Furthermore, in the worst case of clock skew, the clock gets to the first FF 0.8 ns earlier than to the second FF, and there is also a clock jitter of ± 0.2 ns. *For this part consider that the*

⁷As noted above, Design Vision uses (in our case) the TYPICAL library. In the worst case (for which the SLOW library would have been more appropriate) the path delay may be at least 20% slower (and sometimes much slower).

clocks of both registers are generated by the same source, with an inverter between them that introduces the aforementioned skew.

Answer:

17. Calculate the minimum delay for this path. Would the circuit work under these circumstances?
18. If not, how would you fix the problem? Relate to frequency, skew, adding buffers in the path, changes in the μArch , adding FFs.

2.10 Final synthesis

This will be your last behavioral synthesis. Once again *Elaborate* your job. Change the `constraints` file with the following parameters:

- Period using Eq. 1. where ID_1 and ID_2 are the last digit of your ID numbers⁸. Change the waveform correspondingly.
- Fanout 4.
- `useUltra` 1.
- Area 1800.

$$period = \left(15 - \frac{ID_1 + ID_2}{2}\right) ns \quad (1)$$

Compile again, go through the reports, **answer and submit**:

19. Go through the final netlist (`mips.v`) and answer:
 - (a) Give an example of different types of inverters that are used (at least 3).
 - (b) What is the difference between those inverter gates?
20. Submit your final `constraints` file. Name it `constraints_rg##.tcl` (`##` is your account number).

3 Physical Synthesis

In this section you will use Cadence Innovus to perform the *Physical Synthesis* of the 8-bits MIPS μP . This tool allows us to design and implement the physical layers of a VLSI chip automatically. Moreover, Cadence Innovus also performs various important analyses and checks such as: timing, power, IR drop, crosstalk, DRC and LVS, among others.

3.1 Setting up Innovus

To prepare the environment you will create a new folder called *innovus* in your user home folder and copy some configuration files. Every time you should work with the synthesizer, **you must work on this folder (i.e. run the program from inside this folder)**. Open a Terminal and do the following commands,

⁸If you are alone use $ID_2 = 0$.

```
mkdir innovus
cd innovus
cp /users/iit/cadence/tsl018b_2/env.globals .
cp /users/iit/cadence/tsl018b_2/cds.lib .
cp /users/iit/cadence/tsl018b_2/mmmc.view .
cp /users/iit/cadence/tsl018b_2/top.sdc .
cp ~/design_syn/mips.v .
```

the last command will copy the synthesized MIPS netlist to your environment folder.

3.2 Adding I/O pads to the netlist

The synthesized netlist file does not include the I/O pads needed to connect the chip with the package's pins. Before we begin working with the layout we will add the pads using a script. In the terminal, while being at the `innovus` folder, type the following command,

```
/users/iit/cadence/tsl018b_2/gentop.pl mips.v mips
```

you should get the following output:

```
-----
Automatic Layout Pads File Maker for tower 0.18u
Written by Galina Sagdeev
Input: <input_file_name> <module>
Out: top.v & top.io files
-----
```

Before we continue, we will check that the script worked properly. Open the file `top.v` with `gedit` as we did previously and check that the first lines are:

```
module top();
wire wire_clk;
wire [7:0] wire_memdata;
wire wire_reset;
```

Important: If the first lines are:

```
module top();
wire net1000;
wire net1001;
```

it means the script did not work. If that is the case, try starting this step from the beginning and if you fail again inform the teaching or lab staff. If you succeeded, continue to the next step.

We will now change some definitions in this file, where it says:

```
pv0a PAD_G1 (.VSS0(VSS0));
pv0i PAD_G2 (.VSS(VSS));
pv0c PAD_G3 (.VSSC(VSS));
pvda PAD_I1 (.VDD0(VDD0));
pvdi PAD_I2 (.VDD(VDD));
pvdc PAD_I3 (.VDDC(VDD));
```

change it for:

```
pv0a PAD_G1 ( );
pv0c PAD_G3 (.VSSC(VSS));
pvda PAD_I1 ( );
pvdc PAD_I3 (.VDDC(VDD));
```


By the way, the VSS stands for GND and the suffix C indicates supplies to the Core (as opposed to the entire chip or the I/O).

Now open `top.sdc` and change the line of `create_clock` with the following command (with the same clock parameters you defined in `constraints.tlc`⁹):

```
create_clock I5/CIN -name clk -period #period -waveform {0 #period/2}
```

Change `#period` and `#period/2` with your own numbers and delete the `set_max_delay` command. The parameter defines that the I/O pad I5 corresponds to the clock signal.

3.3 Starting Innovus

In the Terminal, while in the `innovus` folder, type `innovus`. The GUI of the tool will open, as in Fig. 7. At the right side we can appreciate the *Layer Control* palette, where we can change the layers' visibility or selectivity. The three icons over the mentioned palette (next to the *online help* textbox) allow us to switch between different abstraction levels in the design and correspond to *Physical*, *Amoeba* and *Floorplan View*.

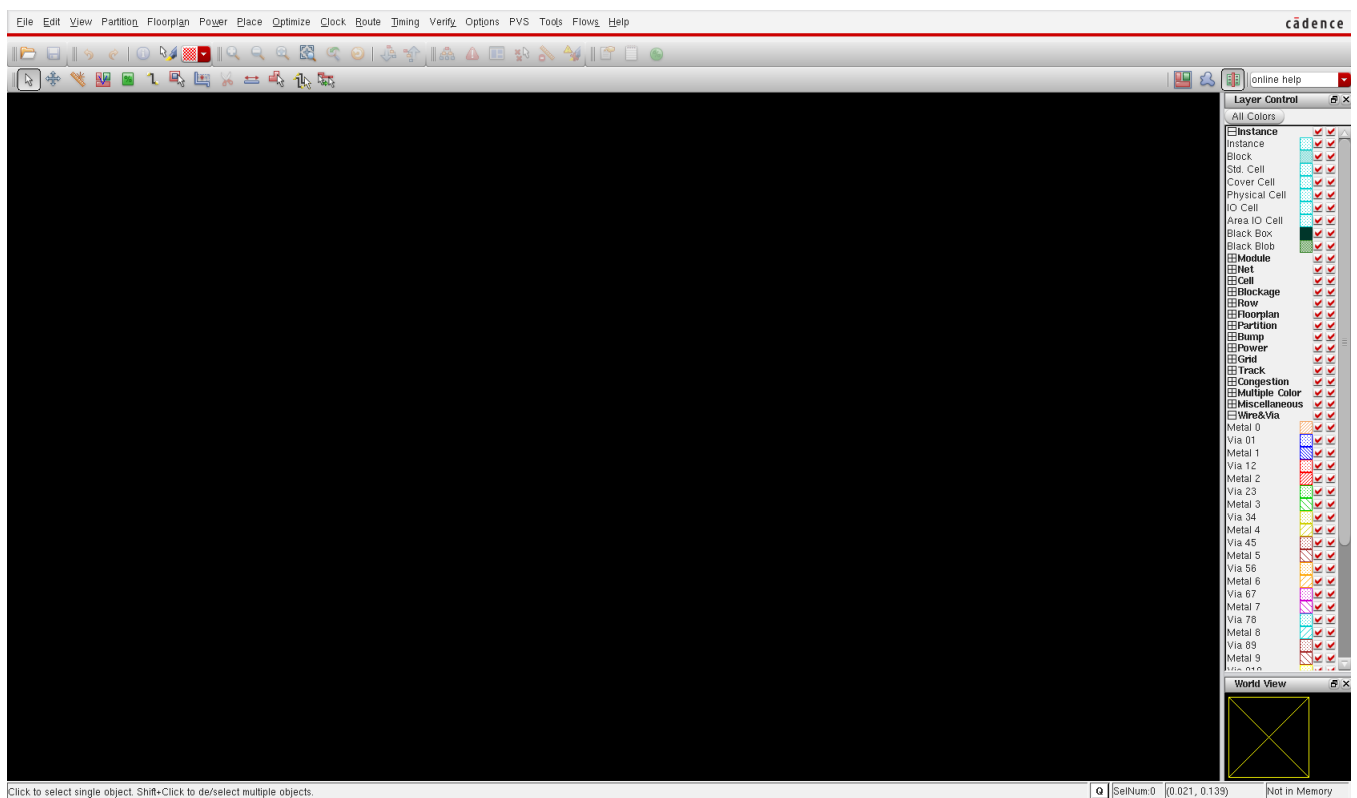


Figure 7: Innovus main window

- Floorplan View: Shows the blocks' placement. This is useful when our VLSI design includes several blocks *e.g.*, memory, several μ P cores, analog-mixed signal blocks, etc.
- Amoeba View: Shows the basic cells in the design.
- Physical View: Shows the whole layout of the design.

You can fit the layout to the screen by pressing 'f' and zoom in/out with the mouse scroll. Also you can zoom in to a specific area by selecting the area with the right click.

⁹This statement assures that the clock is defined properly for all subsequent stages.

To import the design go to *File* → *Import Design*. A window like in Fig. 8 opens. Click on *Load* at the bottom. In the file explorer window that opens select the file `env_globals` and click *Open*. Observe how the different fields in the design import window are filled automatically. Click *OK* and the design will be loaded (it can take a while).

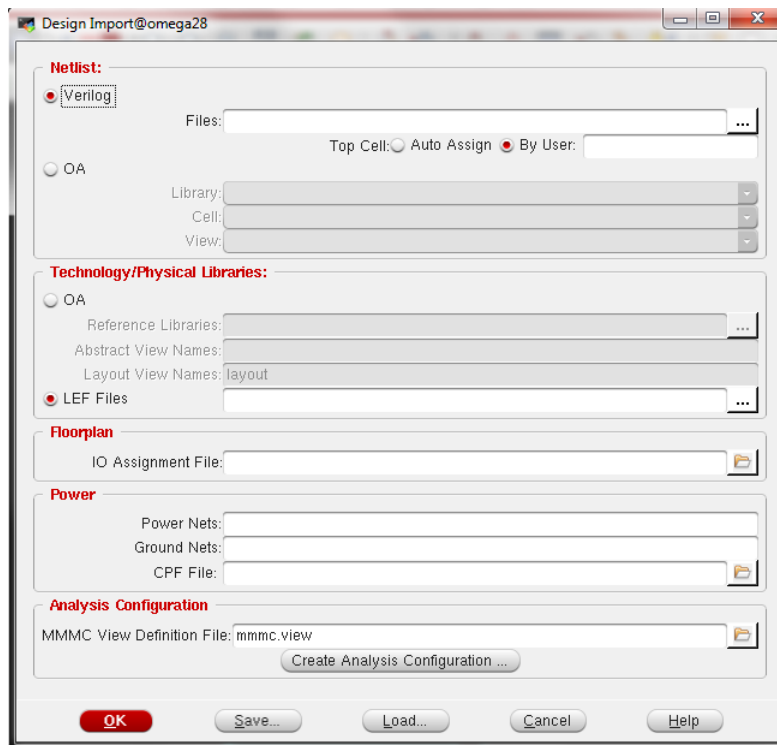


Figure 8: Design import window

After the design is loaded, you should get the layout in Fig. 9. Observe carefully the layout, the outer square are the *I/O pads* and the inner square is called *core* and its where our MIPS will be placed. This is our initial floorplan.

Before we continue, we will save the design. Go to *File* → *Save Design*. In *Data type* select *Innovus* and name the design as `mips.enc`.

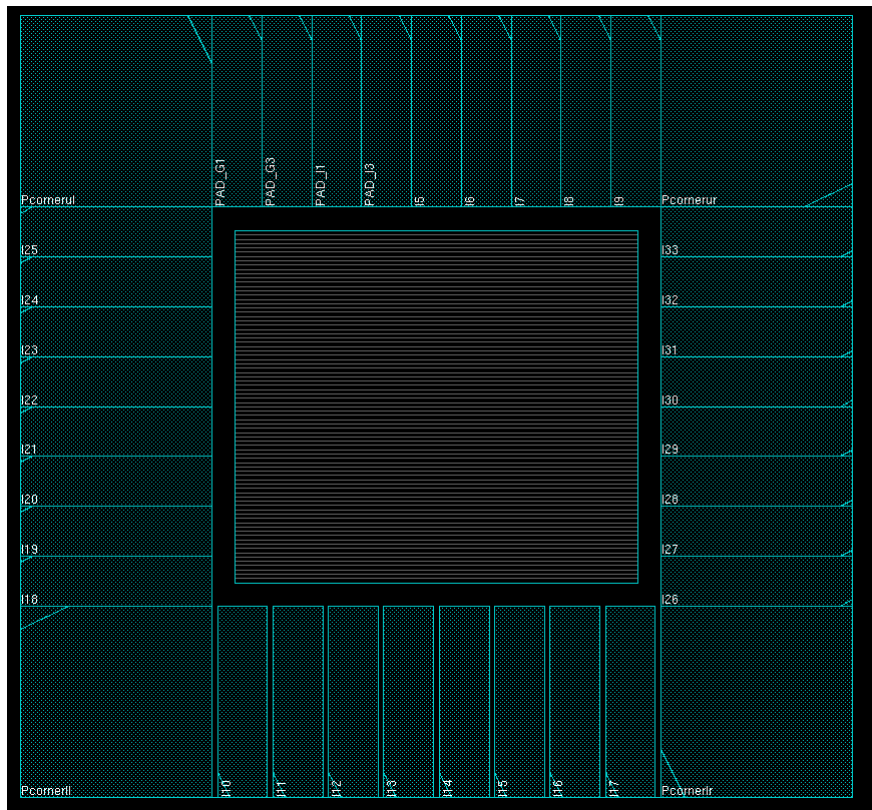


Figure 9: Floorplan

3.4 Floorplan

Firstly we will change the grid, go to *Options* → *Set Preferences*. In the window that opens, select the *Floorplan* tab. Change both *Snap Guides/Regions/Fences to:* and *Snap Macros/Blackboxes to* to *User-defined Grid*. In the center of the same window change *User-defined Grid X* and *Y* to be 0.01.

The floorplan in Fig. 9 is the initial floorplan, we will now change some of its dimensions. Go to *Floorplan* → *Specify Floorplan....* This menu window helps us define the floorplan dimensions. In the *Core Margins* section select *Core to IO Boundary* and change the four options *Core to Left/Right/Top/Bottom* to be 100. Click *Apply*.

Now check that where it says *Die Size by* both *Width* and *Height* are multiples of 0.01 (defined grid). If not, select *Die Size by* and change it to be multiple of 0.01. The final definition should look like Fig. 10 (obviously the dimensions do not have to be equal to your design). Observe how the floorplan is changed, both the distance between the core and the pads and the distance between pads are bigger.

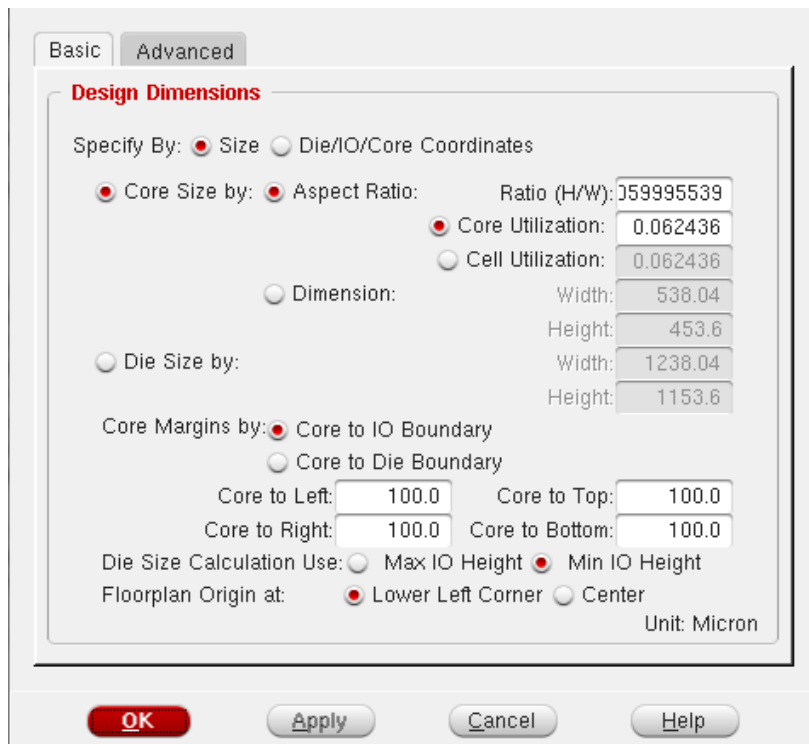


Figure 10: Floorplan specification window

3.5 Power Grid

We must define which pin of each cell is connected to the power source. Go to *Power* → *Connect Global Nets*. In the window that opens you will see *Connection List* at the left (that is now empty). In the *Connect* area select *Pin*, the *Instance Basename* field should have an * (leave it that way). In *Pin Name(s)* field and in the field *To Global Net* write VDD. Click the button *Add to List*, VDD should now appear in the *Connection List*. Repeat this step with VDDC, VSS and VSSC.

Now clean every field (particularly the * in *Instance Basename*) and select *Tie High*. In *To Global Nets* write VDD, click *Add to List* and *Apply*. Then, select *TieLow* (take care *Instance Basename* is clean), write VSS in *To Global Net*, click *Add to List* and *Apply*.

To attach the pads to the grid do the following command in the Terminal window:

```
snapFPlanIO -userGrid
```

and then to add the frames to the I/O pads perform the following command:

```
source /users/iit/cadence/tsl018b_2/iofill.src
```

The power grid consists of Rings and Stripes. Rings surround the core or blocks while the stripes provide power rails across the core or blocks.

3.5.1 Rings

To add rings go to *Power* → *Power Planning* → *Add Rings*. In the menu window that opens select:

- *Net(s)* field: VDD VSS (you can click the ... button and add from the list).
- In *Ring Configuration* select:
 - Top and Bottom to be layer TOP_M.

- Left and Right to be layer M5.
 - Change all widths to 6.
 - Change all Spacing to 1.8
 - Select *Offset : Center in Channel*.
- Click *Apply* and then *OK*.

Observe how the rings were added to your layout.

3.5.2 Stripes

To add rings go to *Power* → *Power Planning* → *Add Stripes*. In the menu window that opens select:

- In *Set Configuration* select:
 - *Net(s)* field: VDD VSS (you can click the ... button and add from the list).
 - Select Vertical
 - Select Layer: TOP_M.
 - Change Widths to 6 .
 - Change Spacing to 1.8 .
- In *First/Last Stripe* select:
 - Select *Start from*: left
 - *X from left and X from right* to be 110.
- In *Advanced* tab select:
 - Omit stripes inside block rings.
 - Omit stripes over selected blocks/domains.
- Click *Apply* and then *OK*.

Observe how the vertical stripes were added to your layout. **Repeat the same procedure to add horizontal stripes in metal layer M5.**

Answer:

23. What is the purpose of power rings and stripes in a chip? Which phenomena do they prevent/reduce?

3.6 Placement

We will now perform the placement of the Standard Cells. Let's say that we want to assure that the cells are not placed below the power grids (to reduce noise). To achieve this, go to *Place* → *Specify* → *Placement Blockage*. In the window that opens select all metals and click *OK*.

To place the cells go to *Place* → *Standard Cells*, in the window that opens select *OK*. Wait a while and the cells will be placed. If you do not see the cells, click on the *Physical View* button.

After this stage your layout should look like Fig. 11. Observe how the cells do not fully occupy the area. Zoom in and observe how the cell area is aligned to the horizontal grids. If you click any wire you will see the whole connectivity of that net highlighted. Now, in the *Layer Control* pallet

select *Cell Visibility*, observe that now layer M1 inside the cells is shown and power rails are aligned to the horizontal grids.

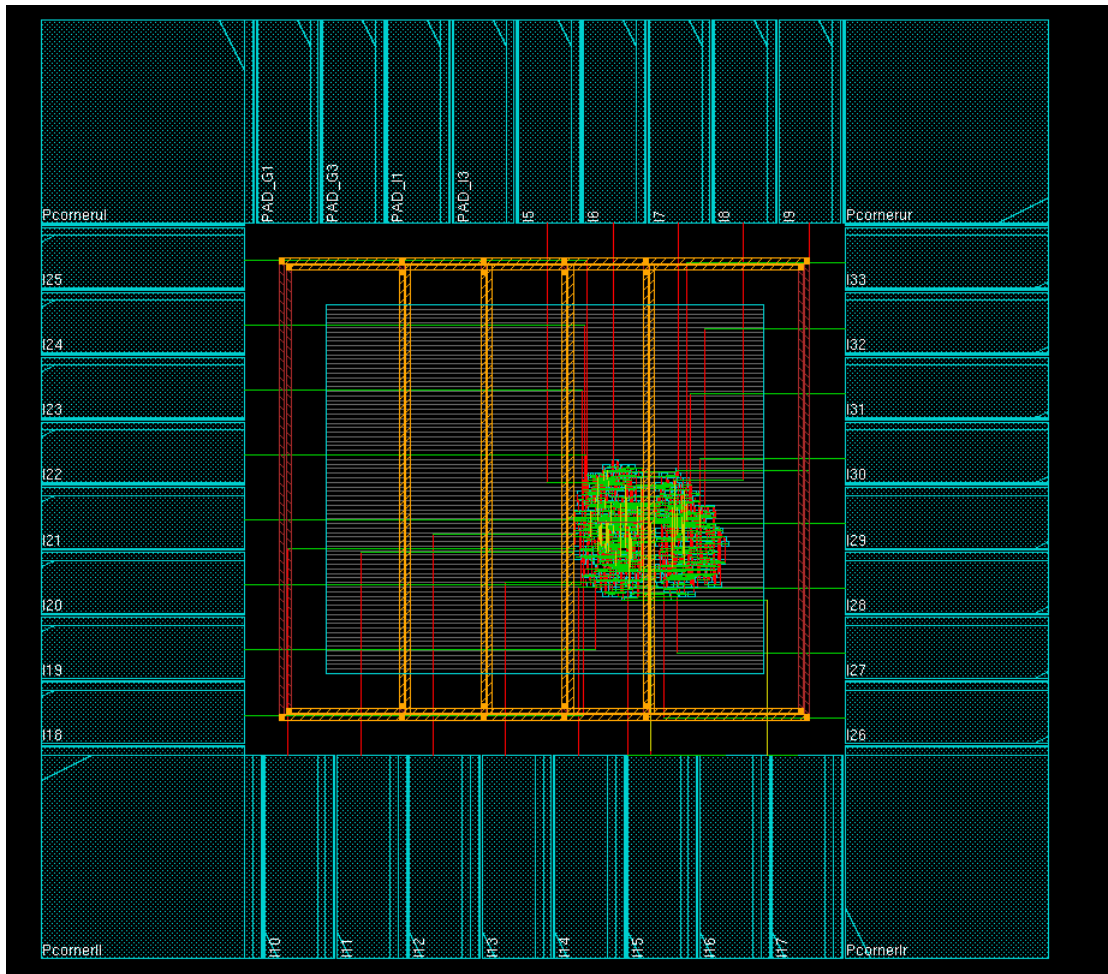


Figure 11: Layout after placement

Observe the output in the Terminal and **answer**:

24. Make a table including: length and number of vias of each metal layer.
25. Can you suggest why the length of M1 is 0 μm ?
26. Can you suggest why the number of vias increases from top layer to bottom layer?

3.7 Static Timing Analysis I

Static Timing Analysis (STA) checks the delay, now taking into account both placement and routing. This stage is critical in the design flow and will determine whether the design complies with the frequency requirement.

Go to *Timinig* \rightarrow *Report Timing....* In the window that opens select *Pre – CTS* (i.e., STA done after CTS) and in *Analysis Type* choose *Setup* (for max delay; the other analysis, *Hold*, checks only for min-delay violations). Observe the output at the Terminal. It should give you a summary and show all timing violations. Moreover, you can go to the directory **timingReports** and read the specific report files for each constraint (e.g., capacitance (.cap), fanout (.fanout), etc).

If the design has some violation. Go to ECO \rightarrow Optimize Design.... In the window that opens select *Design State* to *Pre – CTS* and click OK. Run again STA to see if the problem was solved.

In the terminal type

```
report_timing -nworst 5
```

Answer and submit:

27. What is the critical path slack? Is it the same path than in the behavioral synthesis? If not, explain why.
28. Search for the critical path obtained in the behavioral synthesis, what's the delay? Submit the report of this path. *Hint: Use report_timing -from [starting point] -to [end point].*

3.8 Clock Tree Synthesis (CTS)

The next step is clock tree synthesis. This is a fundamental step in the back-end design. We will use the script `cts.src` from the lab zip file. Copy the file to `innovus/` directory. Remember that the goal is to create a balanced clock with the lowest possible clock skew, insertion delay and power dissipation, but it may be impossible to achieve all minima at the same time...

In the script we have specified a maximum skew of 200 *ps* and a maximum transition time (rise & fall) of 150 *ps*. Run the script by:

```
source cts.src
```

After the CTS finishes open the Clock Tree Debugger. Go to *Clock* → *CCOpt Clock Tree Debugger...* and in the window that opens click *OK*. A window like Fig. 12 will open. This tool will show you in a schematic view how the clock tree is formed. Note that in the left side there's an axis showing the delay from the starting node (clock port). The CTS should have achieved a balanced solution (*i.e.*, the delay from the clock pin to each FF should be similar). Click on any of the elements, it will be highlighted in the layout (at Innovus main window). If you select a net then it will show the whole path in the layout. Moreover, observe that in the lower part of the window there's a browser where you can find values of delay and skew.

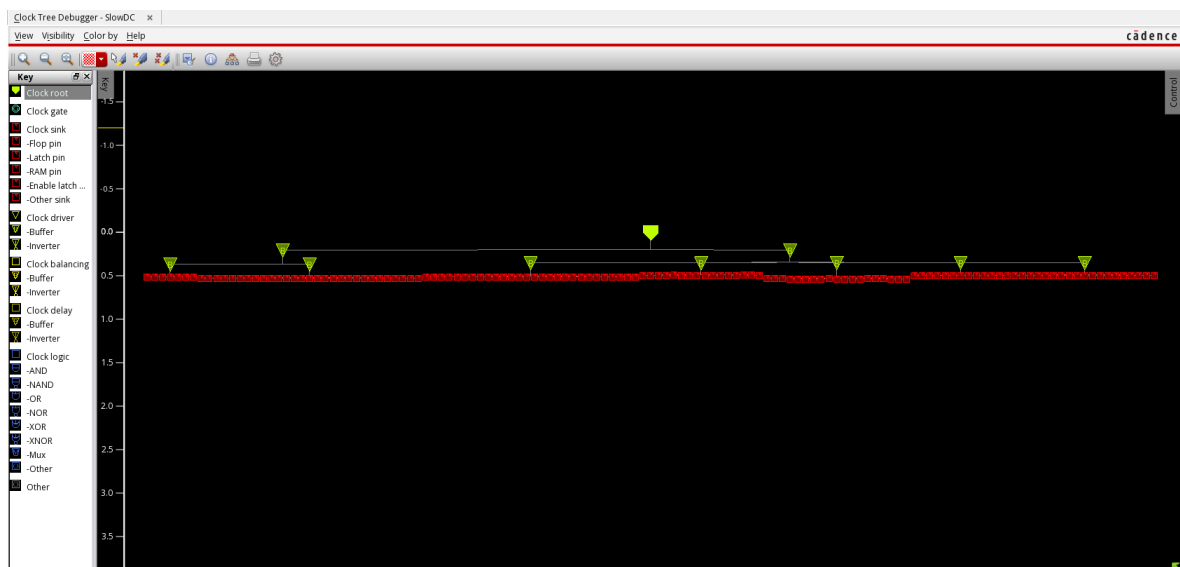


Figure 12: CTS Debugger main window

Explore the CTS debugger and **answer**:

29. What's the average delay from the clock port to the clock pin of the FFs?
30. How many buffers were inserted? What's the purpose of buffers in the CTS?
31. What is the average skew between the FFs? Considering the max/min delay timing analysis, would this skew work? Answer quantitatively.

3.9 Static Timing Analysis II

Check the delays again, now taking into account routing. Go to *Timinig* → *Report Timing....* In the window that opens select *Post – CTS* (i.e., STA done after routing) and in *Analysis Type* choose *Setup* (for max delay; the other analysis, *Hold*, checks only for min-delay violations). Observe the output at the Terminal. It should give you a summary and show all timing violations. Moreover, you can go to the directory **timingReports** and read the specific report files for each constraint (e.g., capacitance (.cap), fanout (.fanout), etc).

If the design has some violation. Go to *Optimize* → *Optimize Design....* In the window that opens select *Design State to Post – CTS* and click *OK*. Run again STA to see if the problem was solved.

3.10 Fill spaces

We usually fill empty spaces by dummy or filler cells that, among other goals, help to meet DRC check of minimum/maximum density of certain layers over the entire silicon. To fill the empty spaces we will use filling cells. These cells come with their own M1 power rails and other layers required by the design rules. **Remember the AA.C DRC error of the Lab 1?**

In the Terminal write the following code:

```
source /users/iit/cadence/tsl018b_2/fill.src
```

Observe the added cells, zoom in and note how M1 rails have filled the entire area.

3.11 Routing

3.11.1 Power Routing–Connect Stripes to Rings

Go to *Route* → *Special Route*. In the window that opens **deselect** the option *Pad Rings*. Click *OK*.

3.11.2 Signal Routing

Go to *Route* → *Nanoroute* → *Route*. In the window that opens select options *Detail Route* and *Global Route*.

Observe and answer:

32. Why are most odd metal layers routed horizontally and most even metal layers routed vertically?
33. Why are VSS and VDD lines placed in an interleaved manner?

3.12 Power Analysis

‘*Dynamic*’ analysis of power is achieved by simulating the circuit over a long time, with a long set of inputs (data and instructions, in the case of MIPS processor), counting actual toggling of each node, and accumulating the energies dissipated as a result. The total accumulated energy is then divided by the total simulated run time, to produce power. Leakage, namely static power, is also computed and accumulated.

‘*Static*’ analysis of power (also named ‘static power analysis’, but take note that it does NOT mean ‘analysis of static power’) means that the circuit is analyzed without simulating it. Rather, activity factors are specified for the inputs, and activity factors for internal and output nodes are computed

as shown in class and as explained in the textbook.

We will now perform a static analysis of power. Go to *Power* → *Power Analysis* → *Setup*. In the window that opens select the Analysis Method to be *Static* and click *OK*. Now go to *Power* → *Power Analysis* → *Run*. In the window select Input Activity to be 0.1 and Dominant Frequency the same as your frequency (defined in Section 2.10 above). In the Terminal you will get the results. Furthermore, you can observe the report `.rpt` file for more information (it should be on `innovus` directory).

Answer:

34. Compare dynamic power obtained here to the dynamic power reported by behavioral synthesis. What can you say about the difference between them? Is it an “apples to apples” comparison even possible?
35. Compare static power (*i.e.*, leakage) obtained here to the static power reported by behavioral synthesis. What can you say about the difference between them?

3.13 Timing Analysis III

Check the delays again, now taking into account routing. Go to *Timinig* → *Report Timing...*. In the window that opens select *Post – Route* (*i.e.*, STA done after routing) and in *Analysis Type* choose *Setup* (for max delay; the other analysis, *Hold*, checks only for min-delay violations). Observe the output at the Terminal. It should give you a summary and show all timing violations. Moreover, you can go to the directory `timingReports` and read the specific report files for each constraint (*e.g.*, capacitance (`.cap`), fanout (`.fanout`), etc).

If the design has some violation. Go to *Optimize* → *Optimize Design...*. In the window that opens select *Design State* to *Post – Route* and click *OK*. Run again STA to see if the problem was solved.

Answer:

34. Compare the critical path reported here to the critical path reported by behavioral synthesis. Are they the same path? Are they similar? If they are not the same path, can you find the path delays for each of the two paths reported by the two different tools, and explain the differences?
35. Compare maximum **achievable** frequency reported by each of the two tools, and explain any differences. By the way, sometimes this is the most critical, most expensive, and most time consuming question asked by VLSI design teams...
36. **Submit the .summary file.** You may need to unzip it, use the command `gunzip #file`.

3.14 Bonus Competition

VLSI chips are typically evaluated on three figures of merit: Small area, low power and high speed. Moreover, we usually cannot optimize for all three goals at the same time...

Let's invent a Figure of Merit (FOM) for our MIPS as follows:

$$FOM = \frac{f}{A \times P} \quad (2)$$

where,

- ***A* is the area of the design after CTS** (without fillers and IO pads). To get the area go to *File* → *Check Design*, in the window that opens click *OK*. Observe the output and use **Total Standard Cell Area** as *A* (again, calculate the area after CTS).

- f is the maximum frequency, as reported at the end of the physical design.
- P is the combination of dynamic and static power, as reported at the end of the physical design while executing in frequency f and using input activity factor of 0.1.

The five teams of students who achieve the highest FOM will win a bonus (for this lab grade) as follows: 5 points of final grade for best result, 4 for second best, 3 for third best, and so on down to 1 for the fifth best FOM.

YOU MUST PROVIDE ALL THE PRINT-SCREENS, REPORTS, ETC. THAT SUPPORT YOUR RESULTS.

3.15 Rest of the flow

Please do NOT perform the reminding design flow steps. This is a description of what should happen next, to get you to a successful tape-out. But we leave this job to those of you who might make a real chip in a Project at the VLSI lab and in the rest of your career as a VLSI designer...

Good Luck

===== End of Lab 2=====