



**Universität Bremen**

**FACULTY OF PHYSICS AND ELECTRICAL ENGINEERING**

Institute for Theoretical Electrical Engineering and Microelectronics

Digital System Laboratory

**Lab Report 2: Exercise on Place & Route**

**Submitted By,**

Md Shazzad Hossain

5228229

Supervised by  
Prof. Dr. Alberto Garcia-Ortiz

Assisted by  
Dr. -Ing. Amir Najafi

## Contents

Chapter 01: Introduction.....	2
1.1 Purpose of this report .....	2
1.2 Creating directories for Data organization and tool setup.....	3
Chapter 02: Importing Design .....	5
Chapter 03: Floor Planning .....	11
3.1 Identification of the floorplan .....	11
3.2 Hard macro positioning.....	14
3.3 Adding Power Rings .....	15
3.4 Adding Pin Placement .....	15
Chapter 04: Placement .....	17
4.1 Standard cell Placement.....	17
4.2 Positioning of spare cells.....	18
Chapter 05: Clock Tree Synthesis .....	19
Chapter 06: Route the design .....	21
Chapter 07: Cell Positioning for Filler.....	23
Chapter 08: Verify and record the outcome.....	25
Figure 1: Making directories.....	3
Figure 2: Creating sourceme.csh file. ....	3
Figure 3: Innovus window.....	4
Figure 4: Import design window. ....	6
Figure 5: Adding Timing Library window. ....	7
Figure 6: Adding Delay Corner window. ....	8
Figure 7: Adding constraint mode.....	9
Figure 8: Adding analysis view. ....	9
Figure 9: Adding Setup and Hold Analysis.....	10
Figure 10: Floor Plan Specification. ....	12
Figure 11: Adding core to IO boundary. ....	13
Figure 12: Hard macro placement.....	14
Figure 13: Adding Power Rings.....	15
Figure 14: Pin Assignment.....	16
Figure 15: Standard cell positioning. ....	17
Figure 16: Adding standard cell positioning mode. ....	17
Figure 17: Adding Spare Cell.....	18
Figure 18: Route Design .....	21
Figure 19: Routing Design .....	22
Figure 20: Adding Filler Cells for cell positioning. ....	23
Figure 21: Routing after adding Filler Cells. ....	24
Figure 22: Violation Check result. ....	25
Figure 23: GDS/OASIS export. ....	26

# Chapter 01: Introduction

The process of converting a circuit description into a physical layout, which specifies the locations of cells and the paths used for their connections, is known as physical design.

The primary goal of the physical design of VLSI chips is to develop a layout with a small area; in addition, the total length of the wires must be kept to a minimum. There are strict limits on the maximum wire length for some important nets.

## 1.1 Purpose of this report

First of all, Physical engineering entails the logical connectivity of cells in netlist (.v) was translated into physical connectivity in GDSII form (layout form).

All design elements are instantiated with their geometric representations during physical design. To put it another way, all macros, cells, gates, transistors, etc. with specified forms and sizes per fabrication layer are given certain spatial places (placement), and the necessary routing connections (routing) are finished in metal layers.

Circuit performance, area, reliability, power, and manufacturing yield are all directly impacted by physical design.

This report seeks to provide a more thorough explanation of the physical design. It involves utilizing CAD tools to place and route an IC. The following steps are involved in an IC's physical design:

1. Floor planning: It establishes the positions of external ports, IP blocks, and sub-circuit or module shapes and arrangements.
2. Placement: It discovers the coordinates of each block's cell in space.
3. Synthesis of a Clock Tree: It decides how to buffer, gate (for power management, for example), and route the clock signal to satisfy specified skew and latency constraints.
4. Routing: It assigns routing resources, such as the channel and switch box's routing tracks, which are used for connections. Moreover, it assigns routes inside the global routing resources to certain metal layers and routing tracks.
5. Verification: It ensures proper electrical and logical performance after the physical design is finished; the layout must be thoroughly validated.

## 1.2 Creating directories for Data organization and tool setup.

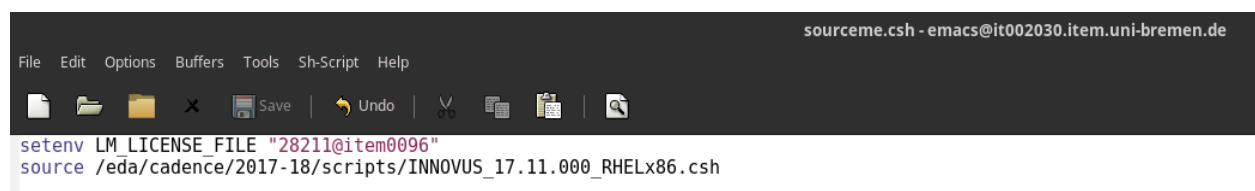
Create the following directories in the `riscv_core` directory:

```
[l_ids303@it002030 exercise2]$ mkdir riscv_core
[l_ids303@it002030 exercise2]$ mkdir do_pr
mkdir: das Verzeichnis „do_pr“ kann nicht angelegt werden: Die Datei existiert bereits
[l_ids303@it002030 exercise2]$ cd riscv_core
[l_ids303@it002030 riscv_core]$ mkdir do_pr
[l_ids303@it002030 riscv_core]$ cd do_pr
[l_ids303@it002030 do_pr]$ pwd
/usrf06/home/agids/lab/l_ids303/2_Exercises/exercise2/riscv_core/do_pr
[l_ids303@it002030 do_pr]$ mkdir results
[l_ids303@it002030 do_pr]$ mkdir reports
[l_ids303@it002030 do_pr]$ mkdir cmd
[l_ids303@it002030 do_pr]$ mkdir log
[l_ids303@it002030 do_pr]$ mkdir tool
[l_ids303@it002030 do_pr]$ ls
cmd  log  reports  results  tool
```

Figure 1: Making directories.

We used the Cadence INNOVUS tools for the lab. To complete the setup, we created a [sourceme.csh](#) file and copied the following commands into it.

```
setenv LM_LICENSE_FILE "28211@item0096"
source /eda/cadence/2017-18/scripts/INNOVUS_17.11.000_RHELx86.csh
```

The image shows a screenshot of an Emacs editor window. The title bar at the top reads "sourceme.csh - emacs@it002030.item.uni-bremen.de". Below the title bar is a menu bar with "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". Underneath the menu bar is a toolbar with icons for file operations like "Save", "Undo", "Cut", "Copy", "Paste", and "Find". The main text area of the editor contains the following two lines of code:

```
setenv LM_LICENSE_FILE "28211@item0096"
source /eda/cadence/2017-18/scripts/INNOVUS_17.11.000_RHELx86.csh
```

Figure 2: Creating `sourceme.csh` file.

Next, the file can now be sourced by me using the following command.

```
source sourceme.csh
```

After that, we can launch the tool using this command.

```
innovus -log log/
```

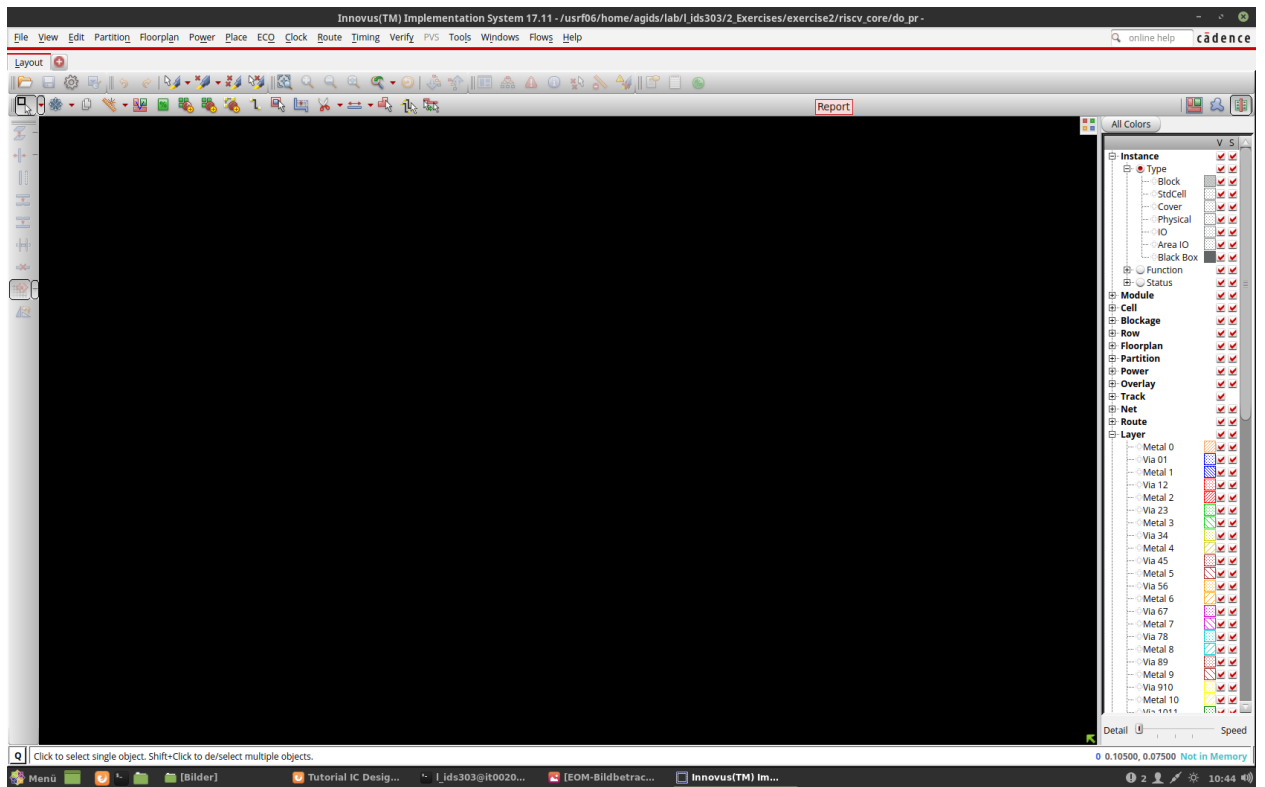


Figure 3: Innovus window.

## Chapter 02: Importing Design

Following several phases, such as specifying the design, design constraints, operational conditions, technology, and libraries, is necessary to transition from logical design to physical design. Here is a description of each action:

1. **Define the Design (.v):** In this step, a hardware description language (HDL) like Verilog (.v) or VHDL (.vhd) is used to describe the design at a logical level.
2. **Define Design Constraints(.sdc):** Design constraints (.sdc) are defined as additional requirements and rules that regulate how the design is physically implemented. They offer data on timing, area, power, and other elements important for proper design synthesis and placement.
3. **Define Operating Conditions (MMMC):** The design will work under certain electrical and climatic circumstances, which are referred to as operating conditions. The abbreviation "MMMC" stands for "Process, Voltage, Temperature, and Modes of operation."
4. **Define Technology and Libraries(.lef,.lib):** Information regarding the target process technology's physical properties and capabilities is available from technology and libraries. Layout Exchange Format (.lef) files, which offer information about the available metal layers, device models, cell dimensions, pin placements, and routing resources, are commonly used to define the technology. The description of libraries, on the other hand, is done using Liberty (.lib) files, which include details regarding the electrical and temporal characteristics of standard cells and other library elements.

Execute on the menu to provide the tool the inputs it needs, [File → Import Design](#)

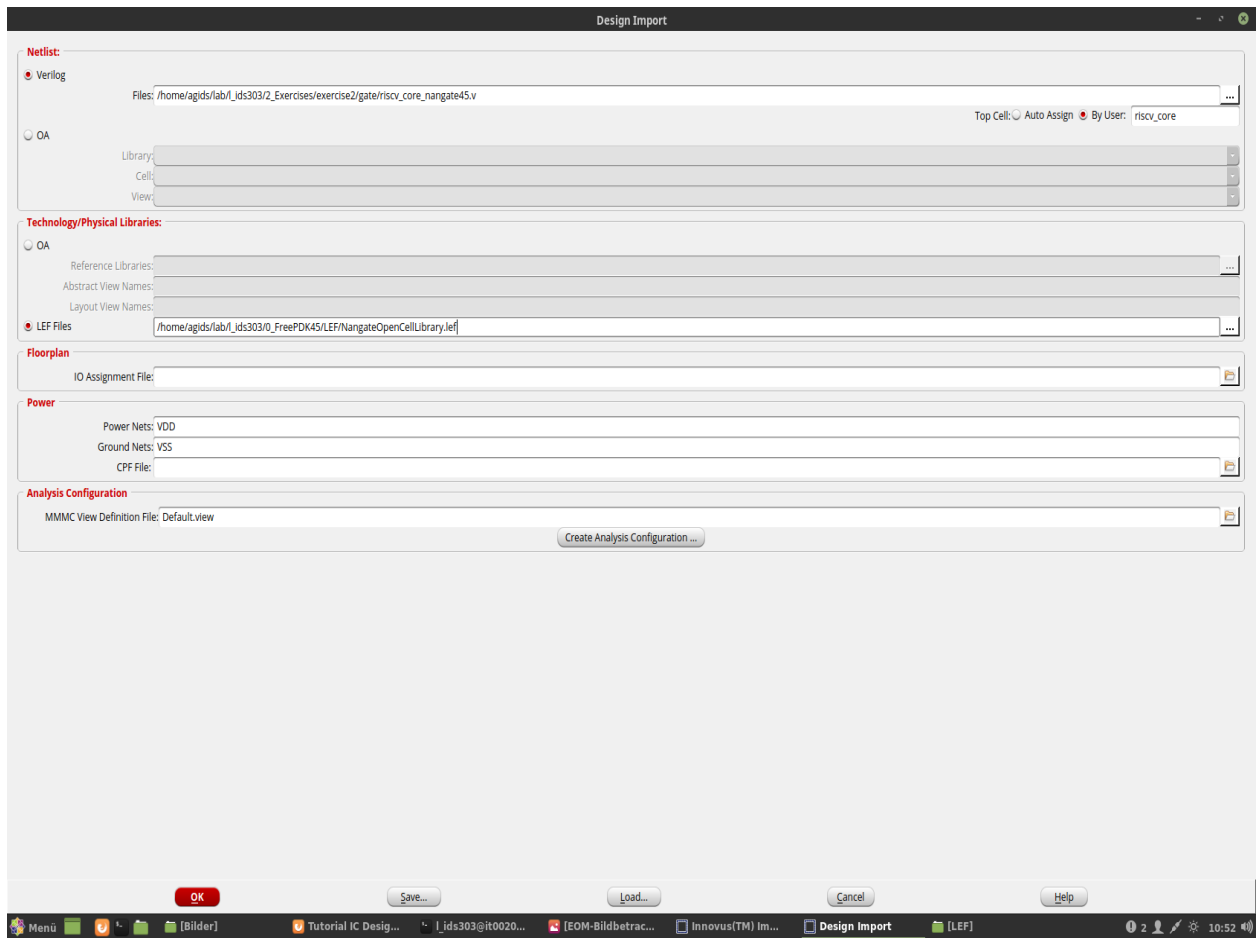


Figure 4: Import design window.

To read the gate-level Verilog file for our design, we must first choose it, after which you must specify the name of the top cell, which is **riscv\_core**. We also need to establish the LEF files for Macros (the memory cell) and the regular cell library. Clicking the "..." button next to the Files: textbox in the Verilog category causes the Verilog file to be added. The Verilog file is added in the Window that will appear by going to the gate directory and double-clicking on the file **riscv\_core\_nangate45.v**. By selecting close at the bottom of the window, this one is then closed.

Similarly, **NangateOpenCellLibrary.lef** is added by browsing to the following directory and selecting **textbfLEF** Files: under the Technology/Physical Libraries: category:

[Lab/0\\_FreePDK45/LEF](#)

The accessible metal layers, via data, design guidelines, and cell geometry are all contained in the LEF file (Library Exchange Format). The power nets of our design, namely VDD and VSS, must then be defined.

By importing an MMC (Multi-Mode Multi-Corner) view file or by selecting Create Analysis Configuration, we may configure the timing libraries and constraints in the Analysis Configuration section.

Select Create Analysis Configuration from the menu. Different working conditions can be set in the MMMC objects section. Provide software with a timing library and use that library to set a delay corner, at the very least:

To add timing libraries, click Library Sets here. Select a name, such as **typical**, and include a timing library for the memory with a **.lib** extension as well as a typical corner library of the standard cells.

The **NangateOpenCellLibrary** typical **ccs.lib** timing library file is added to this window from the location listed below:

[Lab/0\\_FreePDK45/CCS](#)

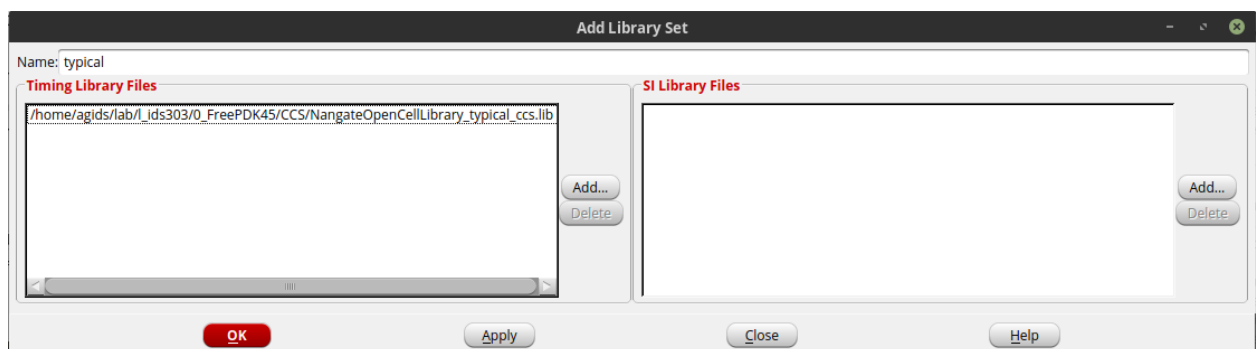


Figure 5: Adding Timing Library window.

To incorporate the delay corner in the double, click the option for delay corners. The Add Delay Corner window will then pop up. The text box next to Name has the name **"typical del corner"** next to the delay corner.

In the Attributes section Selecting **"Library set: typical"** from the drop-down menu saves it by clicking the OK button at the bottom.



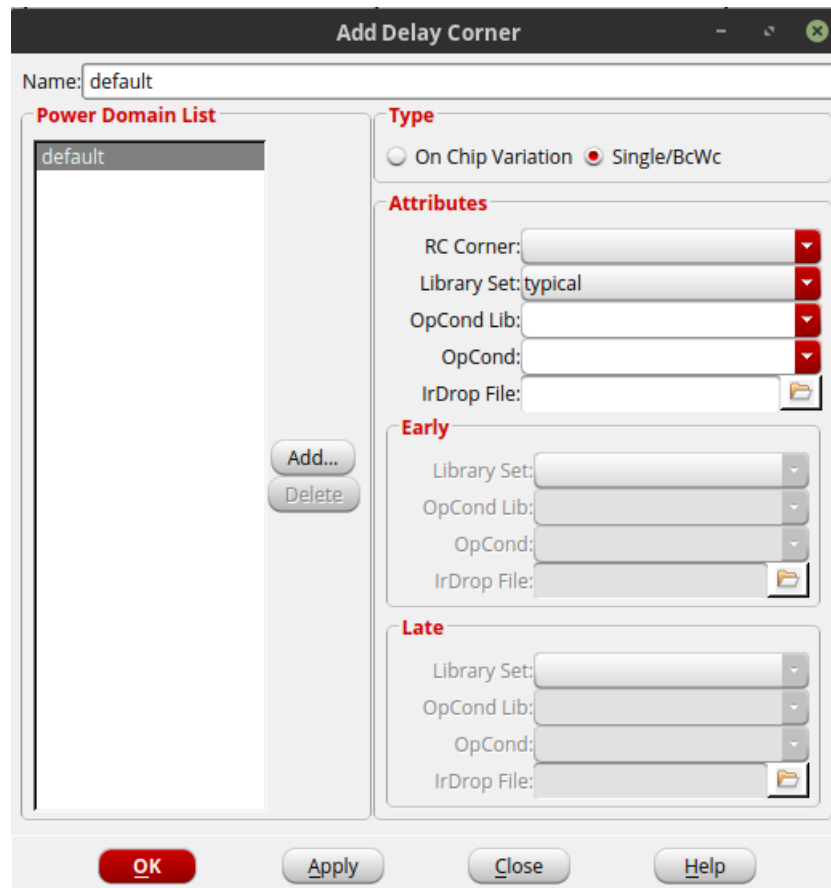


Figure 6: Adding Delay Corner window.

Next, double-click the Constraint Modes option to add the **.sdc** constraint file. The Add Constraint Modes window is then opened. Here, the **.sdc** file from the gate directory is added by clicking the Add button.

By selecting the OK button, the Constraint mode, with the name **my\_sdc**, is saved.

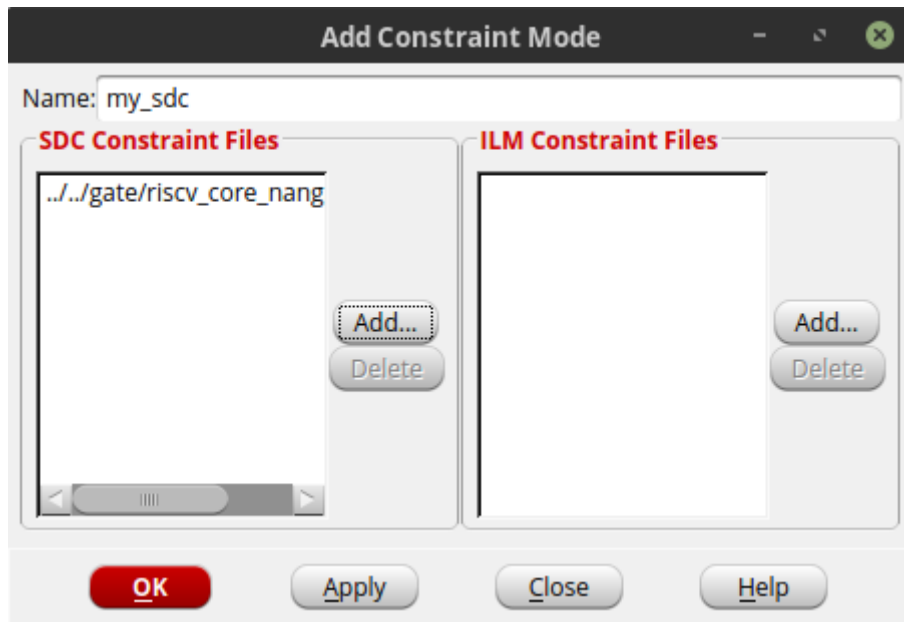


Figure 7: Adding constraint mode.

The Analysis View choice is chosen in the Next step's Analysis View List. The Add Analysis window is then displayed.

By selecting OK, the name "analysis1" is entered and preserved.

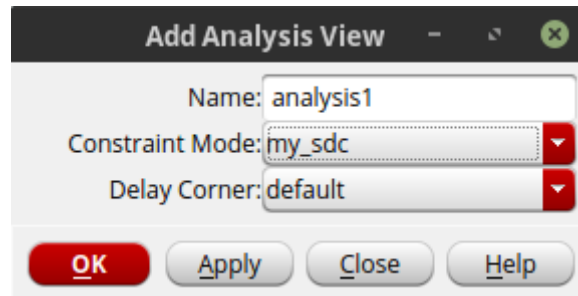


Figure 8: Adding analysis view.

The Setup Analysis Views and Hold Analysis Views are added to the Analysis View List by double-clicking on them. In the dialog that appears, simply click the OK button because the analysis1 Analysis View is already selected.

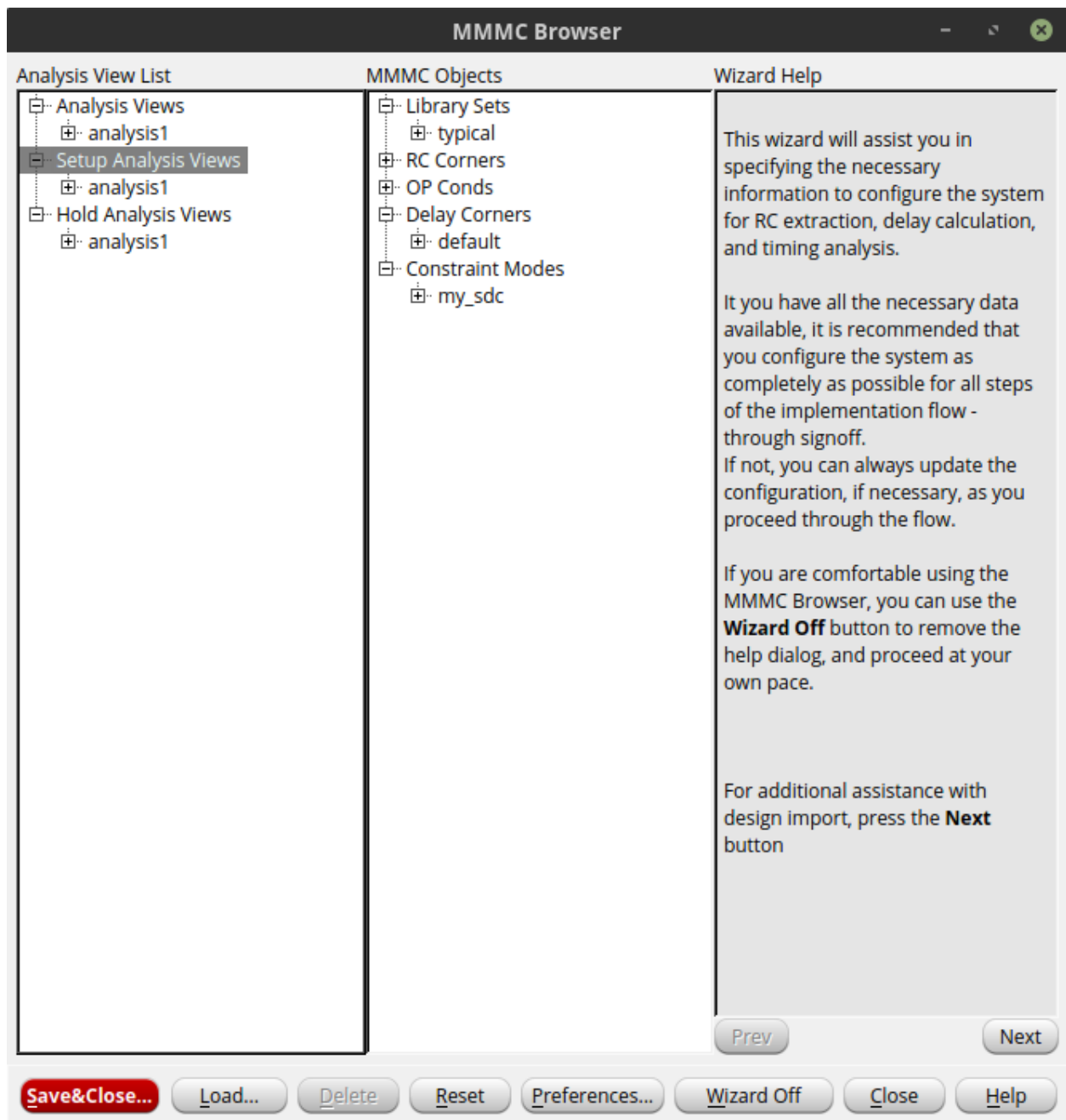


Figure 9: Adding Setup and Hold Analysis.

## Chapter 03: Floor Planning

Floor planning is a crucial component of any physical design. A well-thought-out floor plan results in an ASIC design with increased performance and the ideal amount of space.

Floor layout can be difficult because it involves deciding where to put WE/O pads, macros, power, and ground structure.

To ensure that the inputs utilized for the floor plan are appropriately prepared, we must go through the floor planning process first.

### 3.1 Identification of the floorplan

We should specify the size, aspect ratio, and intended use of our unit in this phase. Choose Floorplan from the menu's "**Specify floorplan**" option.

Changes are made to the following settings:

The ratio (H/W) is set to 1.

Core utilization is set to 0.7.

Core to IO Boundary in all directions is set to 20 microns.

The image shows a 'Specify Floorplan' dialog box with two tabs: 'Basic' and 'Advanced'. The 'Basic' tab is selected. Under the 'Design Dimensions' section, the 'Specify By' options are 'Size' (selected) and 'Die/IO/Core Coordinates'. Under 'Core Size by:', 'Aspect Ratio' is selected with a value of 1. 'Core Utilization' is also selected with a value of .7. 'Cell Utilization' is unselected with a value of 0.699994. 'Dimension' is unselected. Under 'Die Size by:', 'Core Margins by:' is selected with 'Core to IO Boundary' chosen. 'Core to Left', 'Core to Right', 'Core to Top', and 'Core to Bottom' are all set to 20. 'Die Size Calculation Use:' is set to 'Min IO Height'. 'Floorplan Origin at:' is set to 'Lower Left Corner'. The 'Unit' is 'Micron'. At the bottom are 'OK', 'Apply', 'Cancel', and 'Help' buttons.

Parameter	Value
Specify By	Size
Core Size by:	Aspect Ratio
Ratio (H/W)	1
Core Utilization	.7
Cell Utilization	0.699994
Dimension	Width: 262.055, Height: 260.4
Die Size by:	Width: 262.055, Height: 260.4
Core Margins by:	Core to IO Boundary
Core to Left	20
Core to Right	20
Core to Top	20
Core to Bottom	20
Die Size Calculation Use:	Min IO Height
Floorplan Origin at:	Lower Left Corner
Unit	Micron

Figure 10: Floor Plan Specification.

Here, we can specify that the design should have a core usage of 70%, a Core to IO boundary distance of 20  $\mu\text{m}$  in all directions, and an aspect ratio of 1. The ratio of the height to the breadth, or aspect ratio, indicates the size of the chip's core. Core Utilization establishes the core and module sizes based on the density of macros and total standard cells.

By selecting OK, these modifications are stored. Following that, the window below will show up.

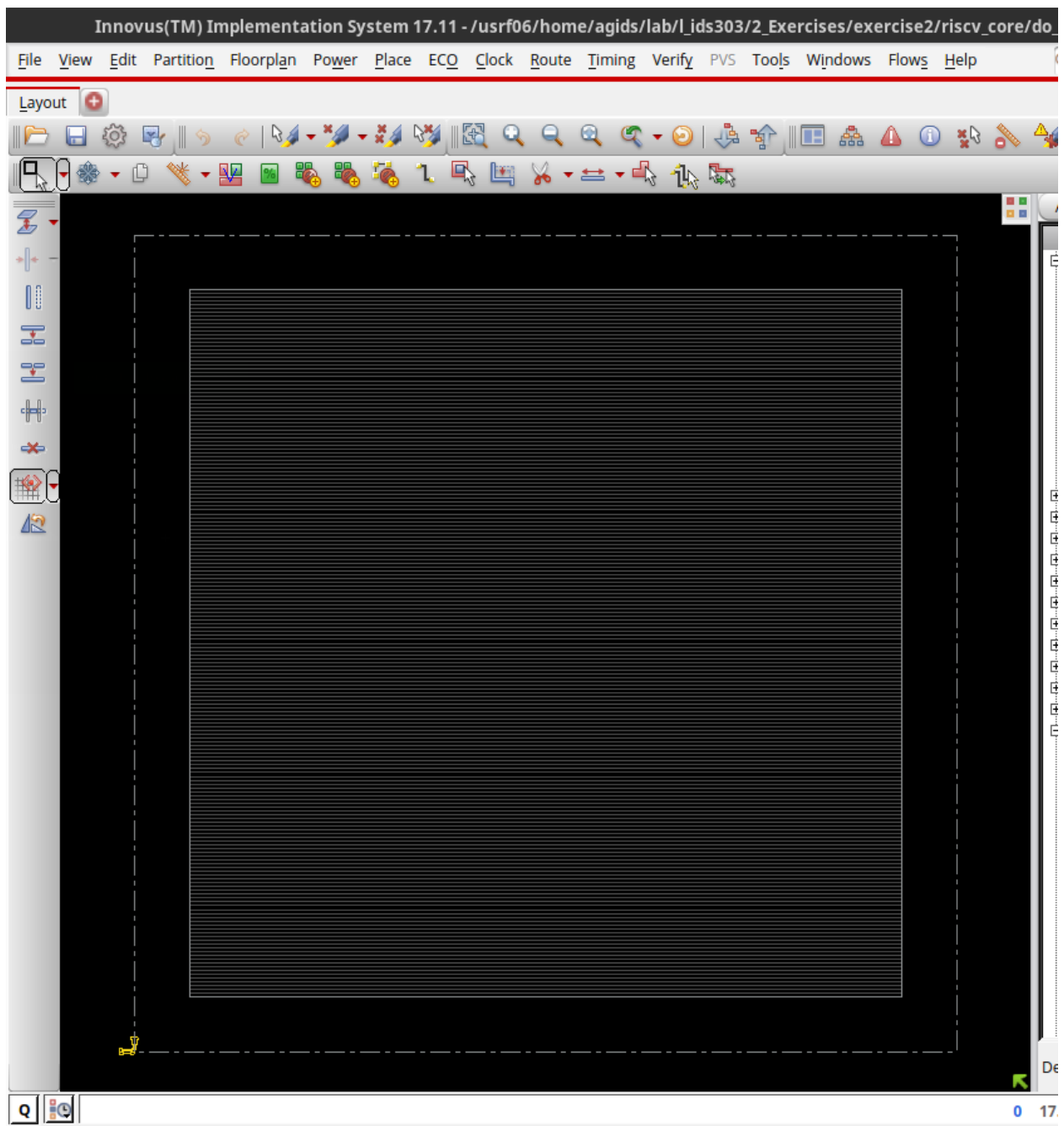


Figure 11: Adding core to IO boundary.

## 3.2 Hard macro positioning

The Floorplan view is chosen from the Top toolbox bar to switch from Physical Design View to Macro View.

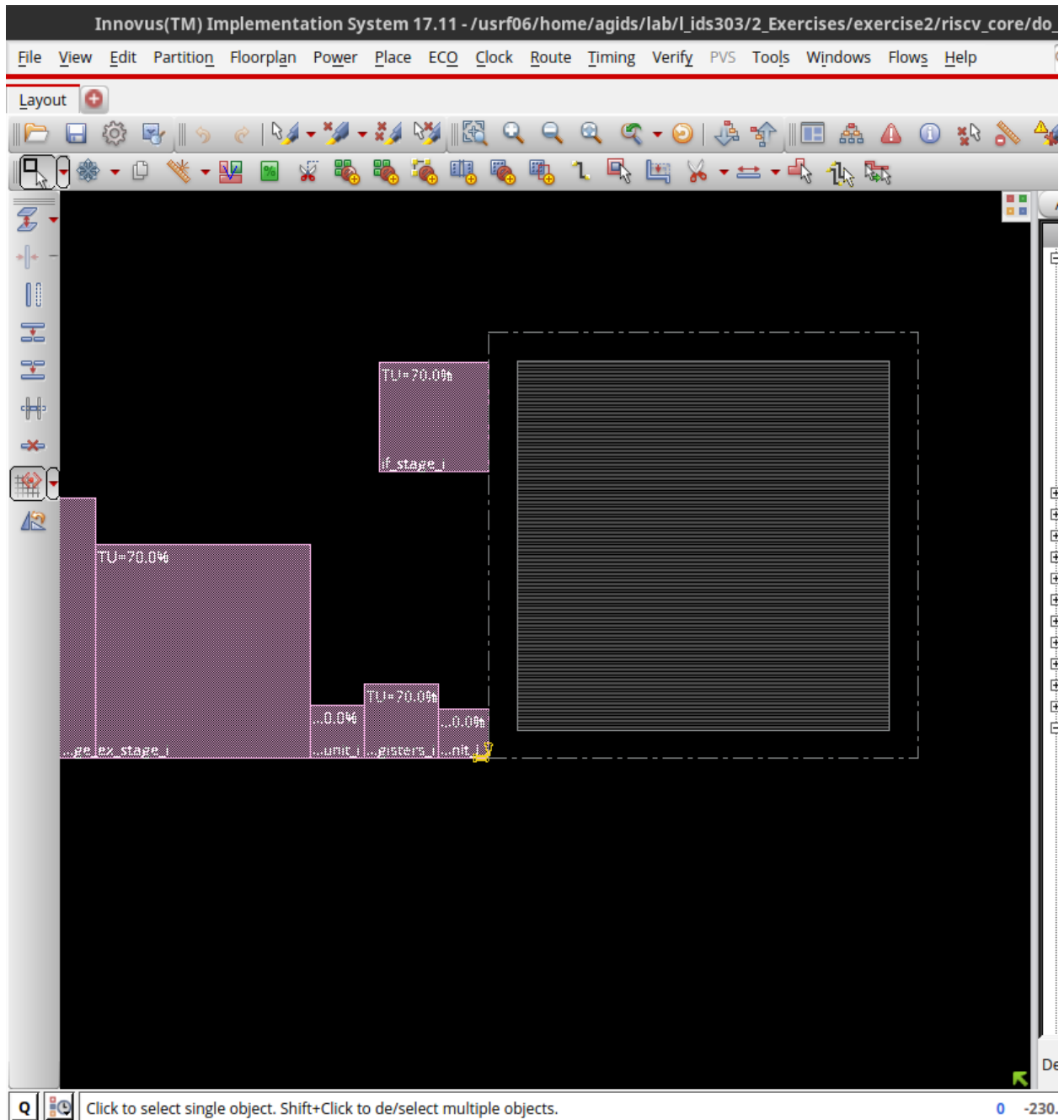


Figure 12: Hard macro placement

Then, choose macros with our mouse and freely position them on our floor plan. When placing huge macros, it would be wise to take effects on routing, time, power, and especially the position of the pins into account. If we typically pushed them to the outside of the floor plan, it might help.

The macro placement should be confirmed by selecting **Floorplan**, then **snap Floorplan** after the macros have been tentatively placed.

### 3.3 Adding Power Rings

VDD and GND rings are frequently positioned around the edges of each hard IP and the chips themselves. The power rings around the core boundary are added in the following stage. From the menu, pick **Power** → **Power Planning** → **Add Ring**.

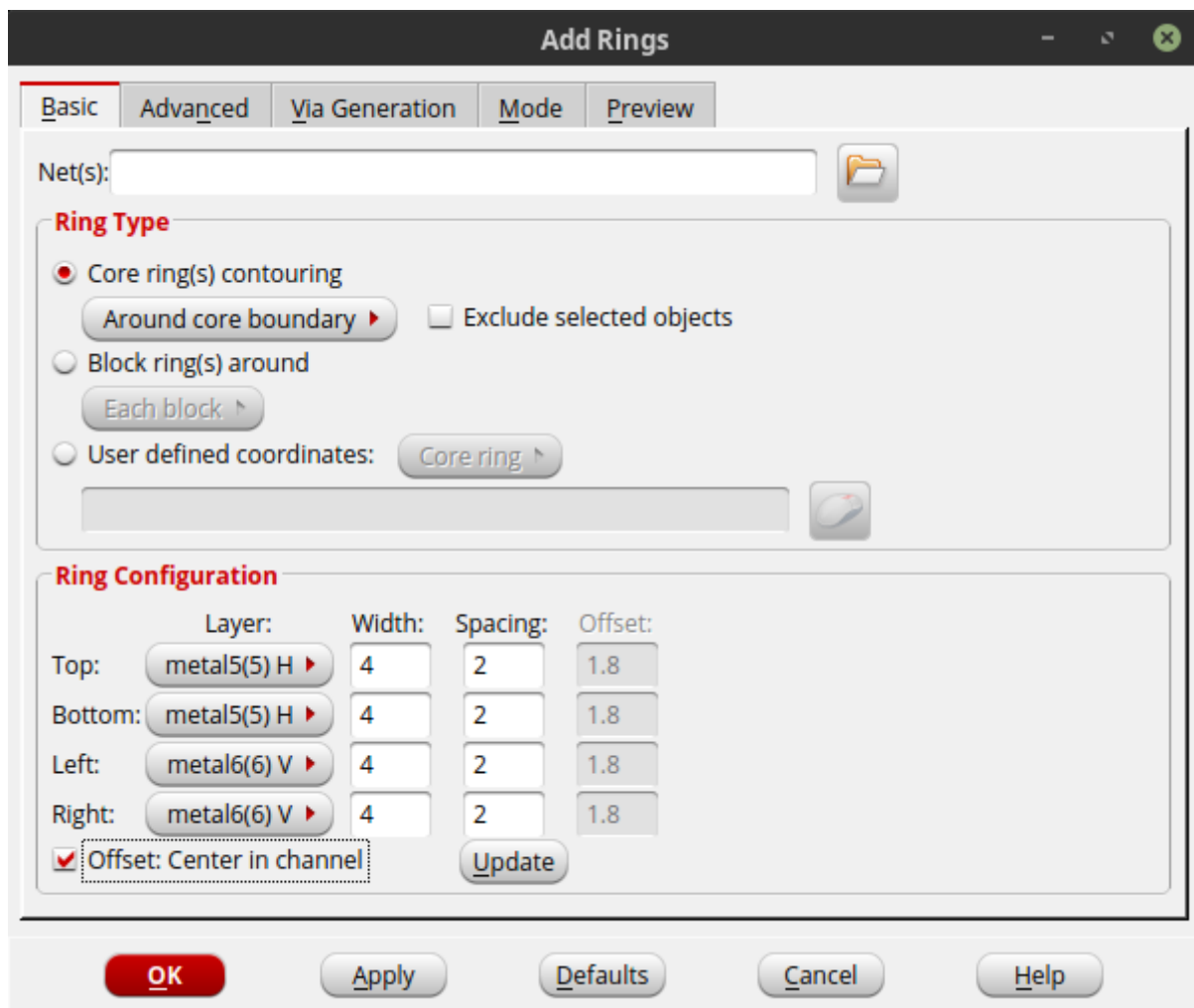


Figure 13: Adding Power Rings.

### 3.4 Adding Pin Placement

Pin locations can be changed using the **Edit**, then **Pin editor**. The pins and the side of the floorplan on which they should be placed are our choices.



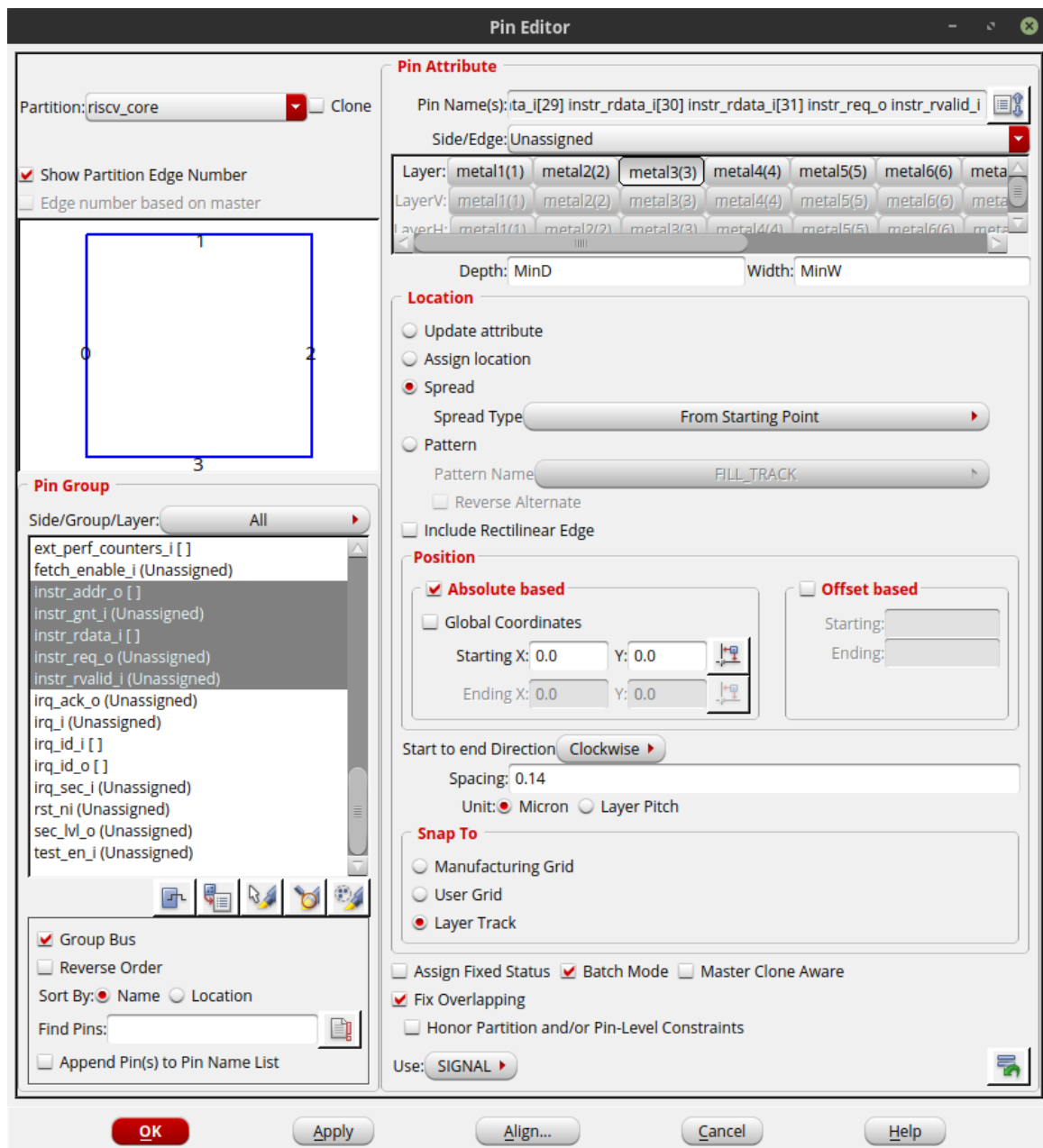


Figure 14: Pin Assignment

Here, we may choose the **instr** pins and set them on metal 3's top side of the floorplan. The recommended distance between pins is  $.14\mu$ .

# Chapter 04: Placement

## 4.1 Standard cell Placement

The cells can now be placed. From the menu, pick [Place](#) → [Place standard cells](#).

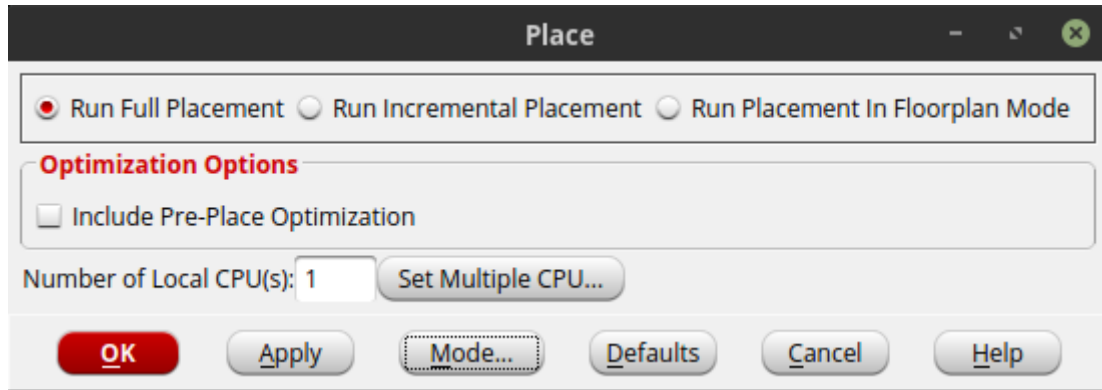


Figure 15: Standard cell positioning.

To choose further options, uncheck **Include Pre-Place Optimization** and click on **Mode...** Pick **Place IO Pins** from the list of options in the new window.

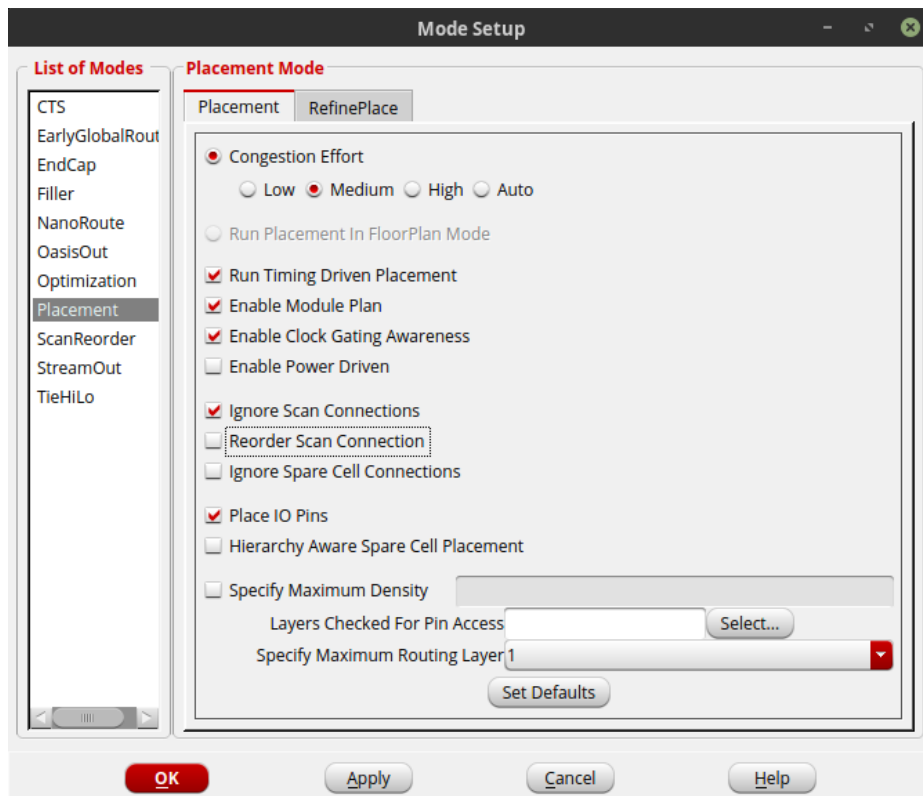


Figure 16: Adding standard cell positioning mode.

## 4.2 Positioning of spare cells

After tape-out, not everything always functions correctly. Spare cells are fundamental parts of the design that do not drive anything. The hope is that they might make a simple correction possible without calling for a complete overhaul. Select a location to add some extra cells. Put the spare cell there.

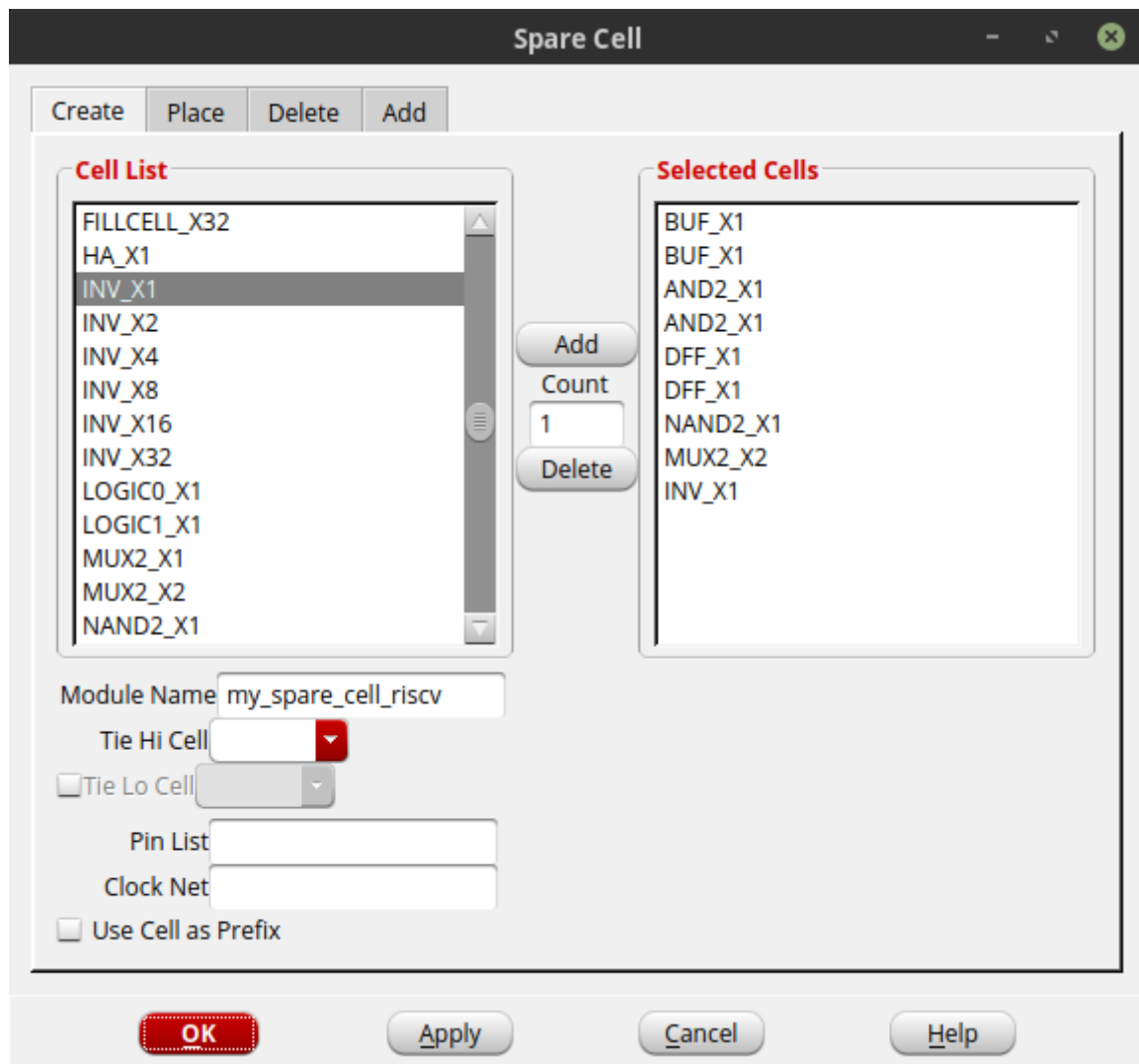


Figure 17: Adding Spare Cell.

## Chapter 05: Clock Tree Synthesis

Before CTS, the clock is not propagated, so after building the clock tree in CTS, we take hold timings into account and attempt to adhere to any hold violations.

The positions of all standard cells and macros are known after placement, and the ideal clock is known during placement (for the sake of simplicity, we will assume that there is just one clock for the entire design). Buffer insertion, gate sizing, and other optimization techniques are exclusively employed for data paths during the placement optimization step; there is no change made to the clock path.

To balance the skew and reduce the insertion latency, CTS involves connecting clocks to every clock pin of sequential circuits using inverters or buffers. One clock source drives all of the clock pins. Meeting all the design requirements requires clock balance.

We have created a floor layout and designated a spot for each gate. But up to this point, We've presumed an ideal clock. As a result, in this phase, we must give a genuine clock signal to every successive element.

Timing, power, area, etc. are all significantly impacted by the clock net. As a result, unlike other nets, the clock net is not sent to all sequential elements.

Clock Tree Synthesis (CTS) may synthesize and balance clock trees according to a specification that is automatically created from a set of multi-mode timing constraints. To improve performance, area, and power, the CCOpt (Concurrent clock optimization) tool expands CTS by concurrently improving the clock and datapath.

**LEF** files often offer the default routing policy for routing. For routing specialized nets like clock nets, we would like to offer wider width and spacing. We employ Non-Default-Rules (NDR) for that. In this situation, clock nets have lower crosstalk and noise levels and higher signal integrity. Therefore, it is advised to add double width and spacing to all clock nets. In general, it is challenging to adhere to the NDR in Metal 1. Designers can define NDR from metal 2 as a result. We should be mindful of the detrimental effects of NDR width and spacing increase on the chip area, though.

To set the NDR, kindly use the Tcl instructions below:

```
add_ndr -name default_2x_space -spacing {metal1 0.38 metal2:metal5 0.42 metal6 0.84}
```

```
create_route_type -name leaf_rule -non_default_rule default_2x_space -top_preferred_layer metal4 -bottom_preferred_layer metal2
```

```
create_route_type -name trunk_rule -non_default_rule default_2x_space -top_preferred_layer metal4 -bottom_preferred_layer metal2 -shield_net VSS -shield_side both_side
```

```
create_route_type -name top_rule -non_default_rule default_2x_space -top_preferred_layer
metal4 -bottom_preferred_layer metal2 -shield_net VSS -shield_side both_side
```

```
set_ccopt_property route_type -net_type leaf leaf_rule
set_ccopt_property route_type -net_type trunk trunk_rule
set_ccopt_property route_type -net_type top top_rule
```

An option for the process technology can be specified using the **setDesignMode** command. Setting the process technology makes the RC extraction more precise.

```
setDesignMode -process 45
```

Then, we set a target maximum transition time and a target skew:

```
set_ccopt_property target_max_trans 0.08
set_ccopt_property target_skew 0.5
```

Additionally, we need to configure the clock tree's buffer and inverter cells, which come from the standard cell library:

```
set_ccopt_property buffer_cells {BUF_X1 BUF_X2 BUF_X4 BUF_X8 BUF_X16
CLKBUF_X1 CLKBUF_X2}
set_ccopt_property inverter_cells {INV_X1 INV_X2 INV_X4 INV_X8 INV_X16}
```

In the end, we construct the clock tree using **create\_ccopt\_clock\_tree\_spec** to generate a clock tree specification using active timing constraints. The following commands automatically route clock networks.

```
create_ccopt_clock_tree_spec -file ./results/ctsspec.tcl
source ./results/ctsspec.tcl
ccopt_design
```

These CCOpt reporting commands can be used to get reports on clock trees and skew groups:

```
report_ccopt_clock_trees -file reports/clock_trees.rpt
report_ccopt_skew_groups -file reports/skew_groups.rpt
```

We give the timing of the design following clock tree optimization:

```
report_timing > reports/timing_report_postCCOpt.rpt
```

## Chapter 06: Route the design

Now, we want to start the routing process of the nets after placement and clock tree synthesis. Choose **Route > Nano route > Route** from the menu.

The NanoRoute dialog box is divided into several sections:

- Routing Phase:**
  - ☒ Global Route
  - ☒ Detail Route    Start Iteration: default    End Iteration: default
  - Post Route Optimization    ☐ Optimize Via    ☐ Optimize Wire
- Concurrent Routing Features:**
  - ☒ Fix Antenna    ☐ Insert Diodes    Diode Cell Name:
  - ☐ Timing Driven    Effort: 5    Congestion:     Timing: S.M.A.R.T.
  - ☐ SI Driven
  - ☐ Post Route SI    SI Victim File:
  - ☐ Litho Driven
  - ☐ Post Route Litho Repair
- Routing Control:**
  - ☐ Selected Nets Only    Bottom Layer: default    Top Layer: default
  - ☐ ECO Route
  - ☐ Area Route    Area:
- Job Control:**
  - ☒ Auto Stop
  - Number of Local CPU(s):
  - Number of CPU(s) per Remote Machine:
  - Number of Remote Machine(s):
  -

At the bottom, there are buttons: **OK**, **Apply**, **Attribute**, **Mode...**, **Save**, **Load**, **Cancel**, and **Help**.

Figure 18: Route Design.

Additionally, we can run the TCL command below.

`routeDesign -globalDetail`

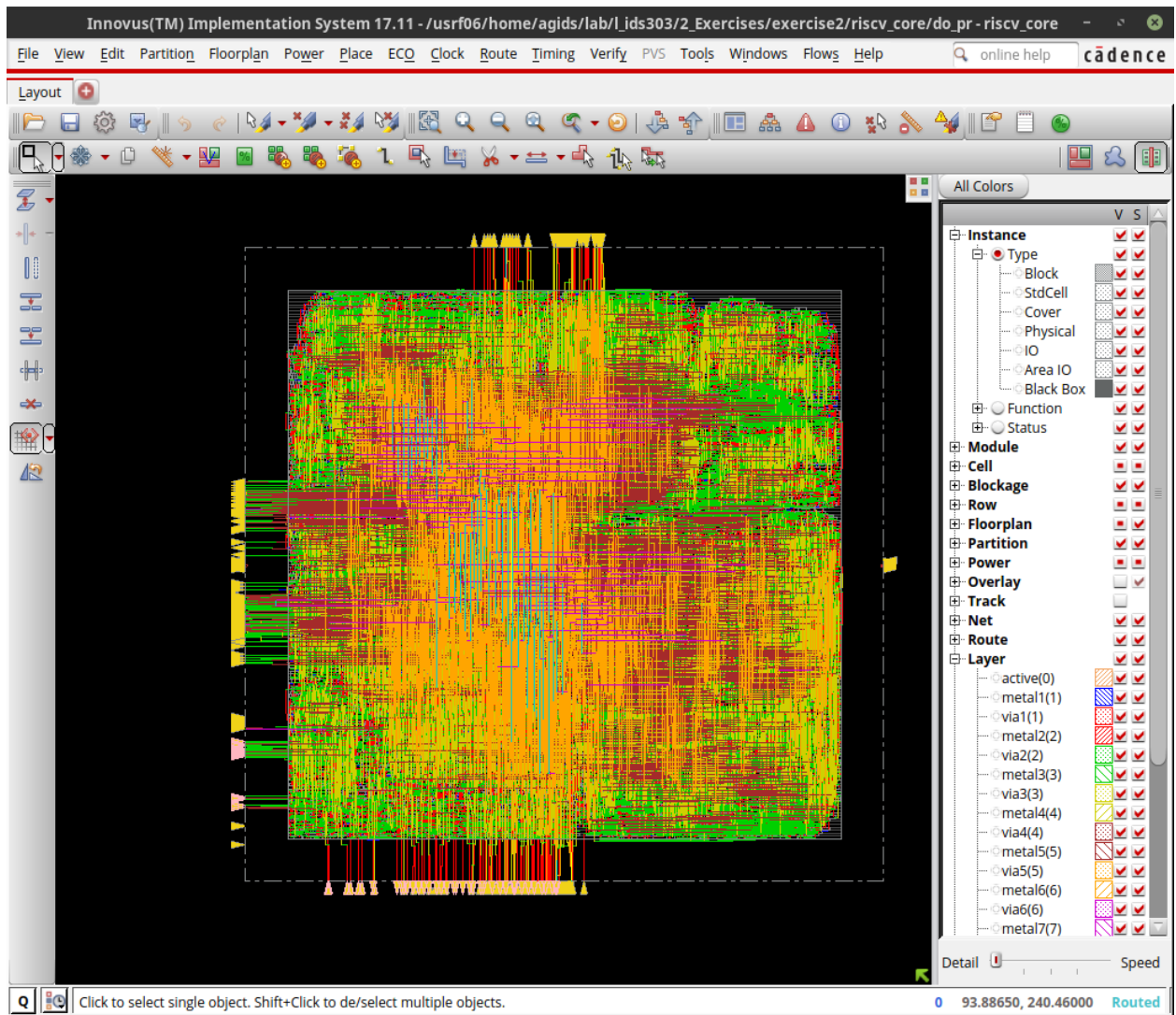


Figure 19: Routing Design.

## Chapter 07: Cell Positioning for Filler

To fill up the gaps in the rows, add filler cells. However, to avoid congestion issues, it should often be done after routing and clock tree synthesis. Choose, first of all, **Place**, then **Physical cells**, and next **Filler cells** from the menu. Select all the filler cells by clicking on **Select**, then click **OK**.

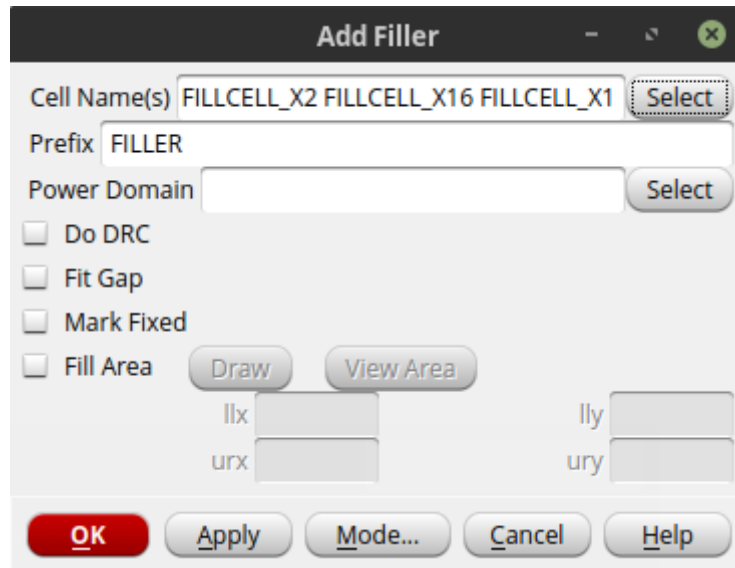


Figure 20: Adding Filler Cells for cell positioning.



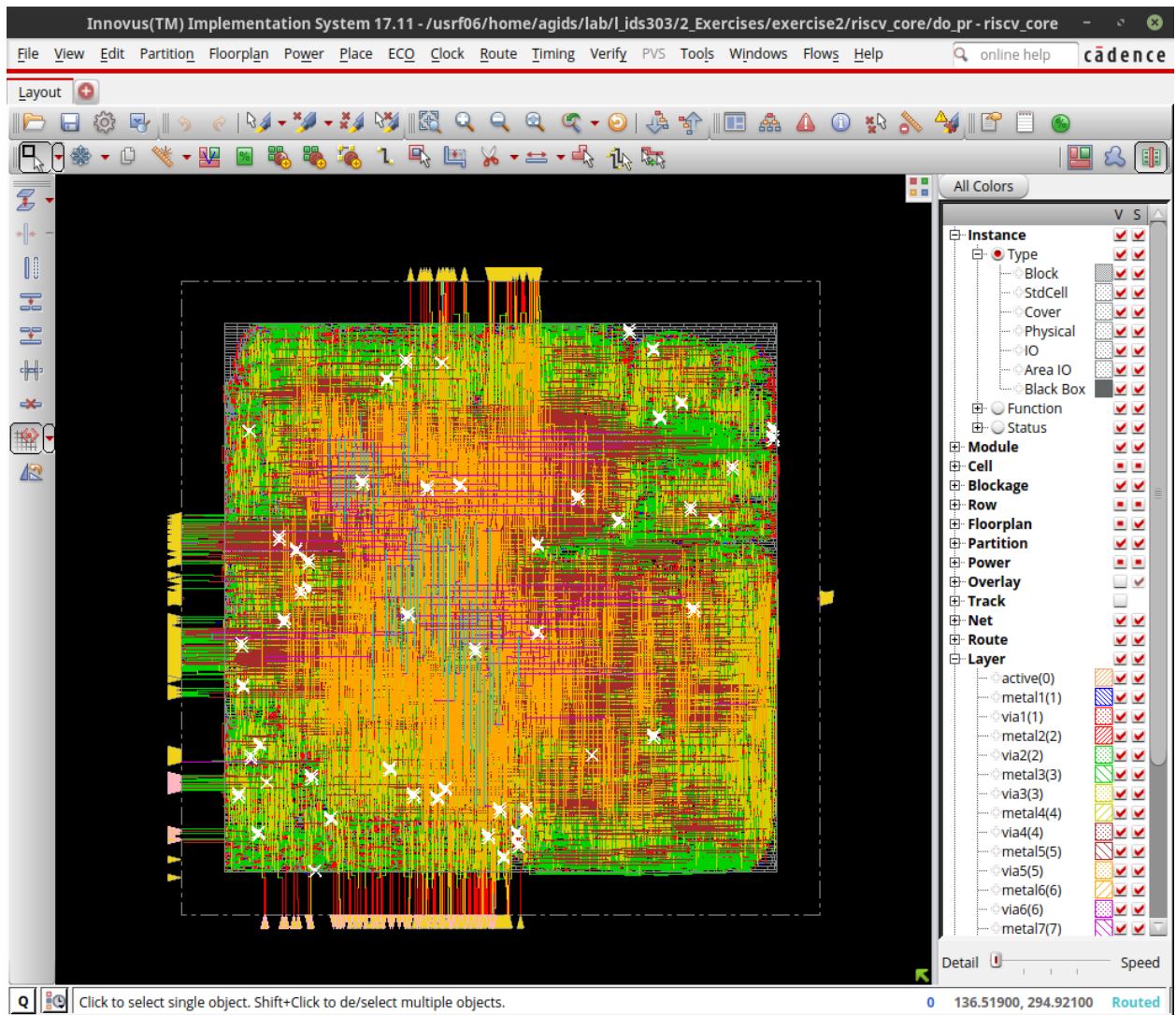


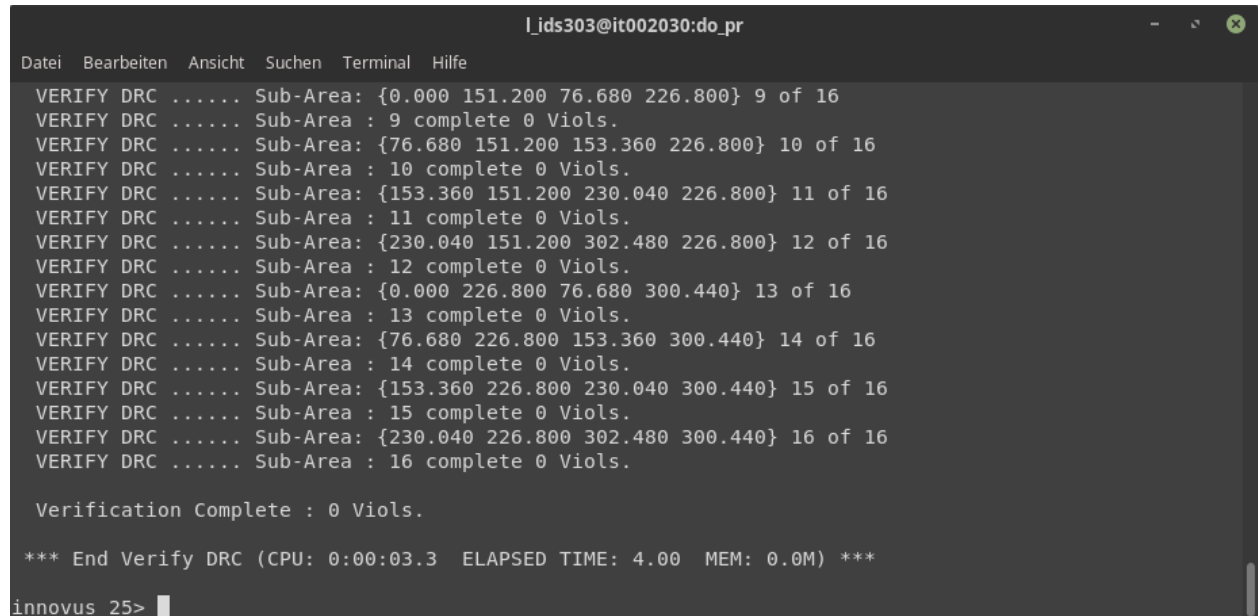
Figure 21: Routing after adding Filler Cells.

## Chapter 08: Verify and record the outcome.

To make sure that our design is error-free. In the tcl terminal, we can write,

```
verify_drc -limit 100000 -report reports/riscv_core.drc
verify_connectivity -report reports/riscv_core.connect
```

Additionally, select **verify DRC** from **verify** in the menu.



```
l_jds303@it002030:do_pr
Datei  Bearbeiten  Ansicht  Suchen  Terminal  Hilfe
VERIFY DRC ..... Sub-Area: {0.000 151.200 76.680 226.800} 9 of 16
VERIFY DRC ..... Sub-Area : 9 complete 0 Viols.
VERIFY DRC ..... Sub-Area: {76.680 151.200 153.360 226.800} 10 of 16
VERIFY DRC ..... Sub-Area : 10 complete 0 Viols.
VERIFY DRC ..... Sub-Area: {153.360 151.200 230.040 226.800} 11 of 16
VERIFY DRC ..... Sub-Area : 11 complete 0 Viols.
VERIFY DRC ..... Sub-Area: {230.040 151.200 302.480 226.800} 12 of 16
VERIFY DRC ..... Sub-Area : 12 complete 0 Viols.
VERIFY DRC ..... Sub-Area: {0.000 226.800 76.680 300.440} 13 of 16
VERIFY DRC ..... Sub-Area : 13 complete 0 Viols.
VERIFY DRC ..... Sub-Area: {76.680 226.800 153.360 300.440} 14 of 16
VERIFY DRC ..... Sub-Area : 14 complete 0 Viols.
VERIFY DRC ..... Sub-Area: {153.360 226.800 230.040 300.440} 15 of 16
VERIFY DRC ..... Sub-Area : 15 complete 0 Viols.
VERIFY DRC ..... Sub-Area: {230.040 226.800 302.480 300.440} 16 of 16
VERIFY DRC ..... Sub-Area : 16 complete 0 Viols.

Verification Complete : 0 Viols.

*** End Verify DRC (CPU: 0:00:03.3 ELAPSED TIME: 4.00 MEM: 0.0M) ***
innovus 25>
```

Figure 22: Violation Check result.

Deleting the routing of the networks with violations and rerouting them is one method for dealing with the routing violation. We can do the following in case of DRC violations.

```
editDeleteViolations
routeDesign
verify_drc -limit 10000
```

To save our design we can use the following command:

```
saveDesign ./results/postRoute.enc
```

We can read our design using the following command:

```
source ./results/postRoute.enc
```

The fastest **postRoute** extraction engine is chosen by the following command. The resistance and capacitance for the interconnects are extracted by the **extractRC** command, and the information is then stored in an RC database. To extract the parasites from the RC database and create an ASCII report using the database data, use the

`extractRC` command followed by the `rcOut` command. RC is extracted, then saved as a file:

```
setExtractRCMode -engine postRoute -effortLevel low
extractRC
rcOut -spef fe_extended.spef
```

At this moment, We can export our design into a **gds** file, **File**→ **Save** → **GDS/Oasis**.

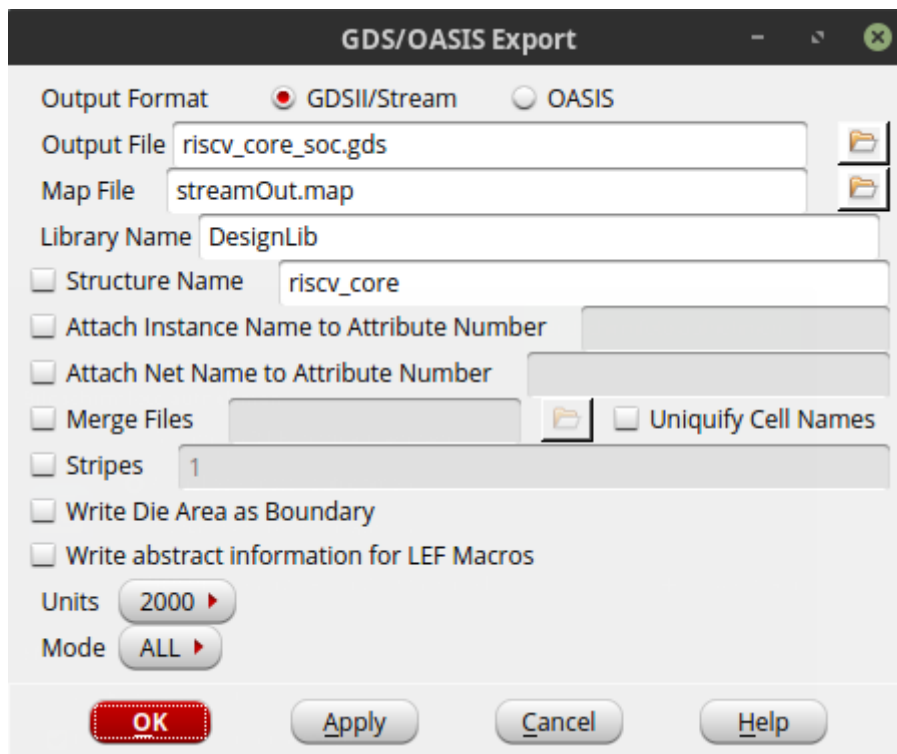


Figure 23: GDS/OASIS export.