ETH zürich

Institut für Integrierte Systeme
Integrated Systems Laboratory

Department of Information Technology and Electrical Engineering

EFCL Winter School 2026

Sample Solution Exercise 01

Navigating OpenROAD

F. K. Gürkaynak
Prof. L. Benini

Last Changed: 2026.02.09

1 Overview

This exercise will allow you to get a first impression of the design we will use throughout the exercises and become familiar with the leading open-source back-end tool OpenROAD. It is meant as a first introduction to OpenROAD where you learn to navigate the GUI, a good understanding of the CAD tools is a requirement for every good engineer, and these kills will make it easier to complete the later exercises. The individual steps of the front-end and back-end design flow are covered in further exercises in more detail. Exercise 02 will cover RTL simulation using Verilator. In Exercise 03 you will go through the synthesis process using Yosys and generate timing reports using OpenSTA. After the frontend is complete, we return to OpenROAD for floorplanning and the power grid in Exercise 04. Then, in Exercise 05 you will go through the place and route process to arrive at the same design that you will see today. Finally, in Exercise 06 we will show you the last steps to finish a chip for fabrication.

1.1 About the Style

We will use a number of different styles to identify different types of actions as shown below:

Student Task: Parts of the text that have a gray background, like the current paragraph, indicate steps required to complete the Exercise.

Actions that require you to select a specific menu will be shown as follows:
`menu→ sub-menu→ sub-sub-menu`

Whenever there is an option or a tab that can be found in the current view/menu we will use a BUTTON to indicate such an option.

Throughout the exercise, you will be asked to enter certain commands using the command-line[1].

```
sh>  command to be entered on the Linux command line
```

[1] There are many reasons for using a command-line, some functionality can not not be accessed through GUI commands, and in some cases, using the command-line will be much faster. Most importantly, things you enter on the command line can be converted into a script and executed repeatedly.

2 VLSI2 and Croc

First a bit of history for the interested. After the successful tapeout of Basilisk in May 2024[2], our Linux-capable end-to-end open-source SoC, we were sure that the open-source tools are good enough to teach beginners everything taught in the already existing VLSI2 lecture. Additionally, we wanted to move to a new modality with a project as the main focus of the lecture instead of the previous exam.

For this reason, we decided to immediately start working on a simplified tool flow derived from Basilisk[3] (which itself is derived from the OpenROAD-flow-scripts[4]). For the design we decided to go for a simple microcontroller as a basis for student projects. The SoC was derived from PULPs safety-island[5] (written and maintained by Michael Rogenmoser). As a first testrun of this reworked VLSI2, we made a first version ready for the summer school 2024. The exercises you are solving today are an updated version of the very same.

After a very successful summer school we continued to refine the flow and then moved from a simple safety-island derivative to the more standalone Croc SoC[6]. In the fall semester 2024 two students went through the exercises, added their own designs to the Croc SoC and taped out the first Croc derived chip called MLEM[7] in November 2024 (less than six months after we started this project). This chip works and you can see it on Wednesday in the IC tester lab.

Then, for the first edition of the new VLSI2 course in the spring semester 2025, we had many PhD students work on the expanded exercises now published on the VLSI wiki[8]. The new VLSI2 course was a huge success, the student feedback was very positive, and the results we got from the graded projects were great. Five chip designs were selected and taped (Pjon-on-Croc, River, Chiffre, Crocodilo and Fluffy; you can find them all on our chip gallery[9]).

Since then, we have been working on further refining and documenting the Croc SoC, updating the exercises for the ever evolving tools and preparing this winter school.

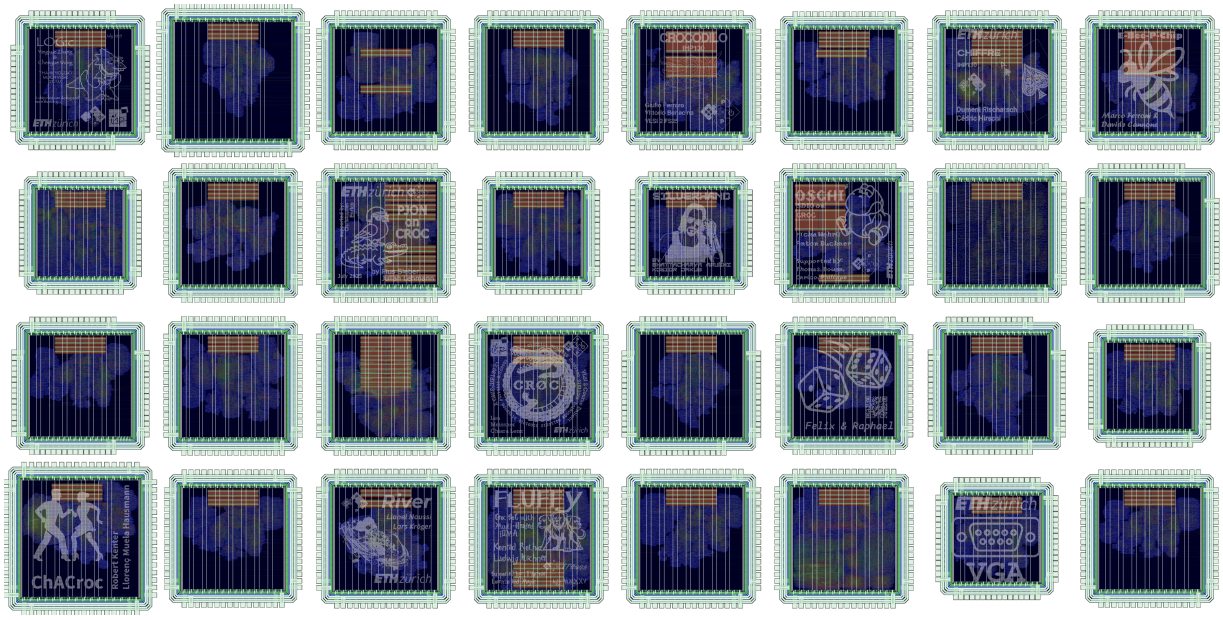

Figure 1: Student designs submitted as part of the VLSI2 2025 final project.

[2] http://asic.ethz.ch/2024/Basilisk.html
[3] https://github.com/pulp-platform/cheshire-ihp130-o/
[4] https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts
[5] https://github.com/pulp-platform/safety_island
[6] https://github.com/pulp-platform/croc/
[7] http://asic.ee.ethz.ch/2024/MLEM.html
[8] https://vlsi.ethz.ch
[9] http://asic.ethz.ch/all/imagemap.html

3 Technology, Cells, Tools and Design

In all exercises today, you will use IHPs open-source 130nm technology[10]. This defines how our geometries have to look like to be manufacturable, the characteristics of the fabricated transistors as well as the SRAMs and IO cells provided by them. The standard cells we will use are not the ones from IHP, instead we will be using the the in-house designed *EZ-cells* developed during the VLSI-5 course (led by Oscar Castañeda and Prof. Christoph Studer). The standard cell library will be released publicly soon; we ask you not to copy or distribute any of the technical details (liberty, lef, and GDS files) until then. We already wanted to use this cell library to show you one thing: an open-source PDK allows *anyone* to help improve libraries, designs and tool integrations. We hope to see more such efforts in the future with different libraries for multiple open-source PDKs specialized for any possible use-case.

To make it easy to maintain and use the tools, we use the IIC-OSIC-TOOLS[11] docker container (internally called `oseda`). It is maintained by Harald Pretl and includes everything necessary for digital, analog or mixed signal IC design using open-source tools. Further, Harald Pretl also maintains scripts to make it easy to use the container on Linux, Windows and Mac. Without this container, we would not be able to offer a complete solution for our students to run the entire VLSI exercises on their personal laptops if they whish to do so.

The design you will see today is the Croc SoC developed as part of the PULP project, a joint effort between ETH Zurich and the University of Bologna. It is a small SoC oriented towards education. Croc is built around OpenHWGroups CVE2[12] (forked from lowRISCs Ibex[13]) RISC-V core and the OBI interconnect[14]. The public repository aims to provide the design and a complete design flow from RTL to GDS, with the primary use-case being our VLSI2 course. Everything is provided under a Apache 2.0 compatible license.

4 Using OpenROAD

First, a warning: OpenROAD is the state-of-the-art open-source backend design tool. As such, it contains many features that are not necessary for these exercises. Additionally, many features require you to know or look up commands; there are no GUI options for them (to some extent, this is also the case for commercial tools).

Before we get more familiar with OpenROAD, let us first look at a completed design.

Student Task 1:

- Start by navigating to your home and then the exercise directory

```
sh>  cd ~/ex01
```

- Enter the `oseda` containers shell

```
sh>  oseda bash
```

  The **`openroad`** subdirectory called contains all OpenROAD scripts, save points and reports. Start the OpenROAD GUI from the **`openroad`** directory with the following command.

```
sh>  cd openroad/
sh>  openroad -gui scripts/startup.tcl
```

Note 1: If you ever open a new terminal window or tab, don't forget to re-enter the oseda containers shell, otherwise you won't be able to start the open-source tools.

[10] https://github.com/IHP-GmbH/IHP-Open-PDK
[11] https://github.com/iic-jku/IIC-OSIC-TOOLS
[12] https://github.com/openhwgroup/cve2
[13] https://github.com/lowRISC/ibex
[14] https://github.com/openhwgroup/obi/blob/main/OBI-v1.6.0.pdf

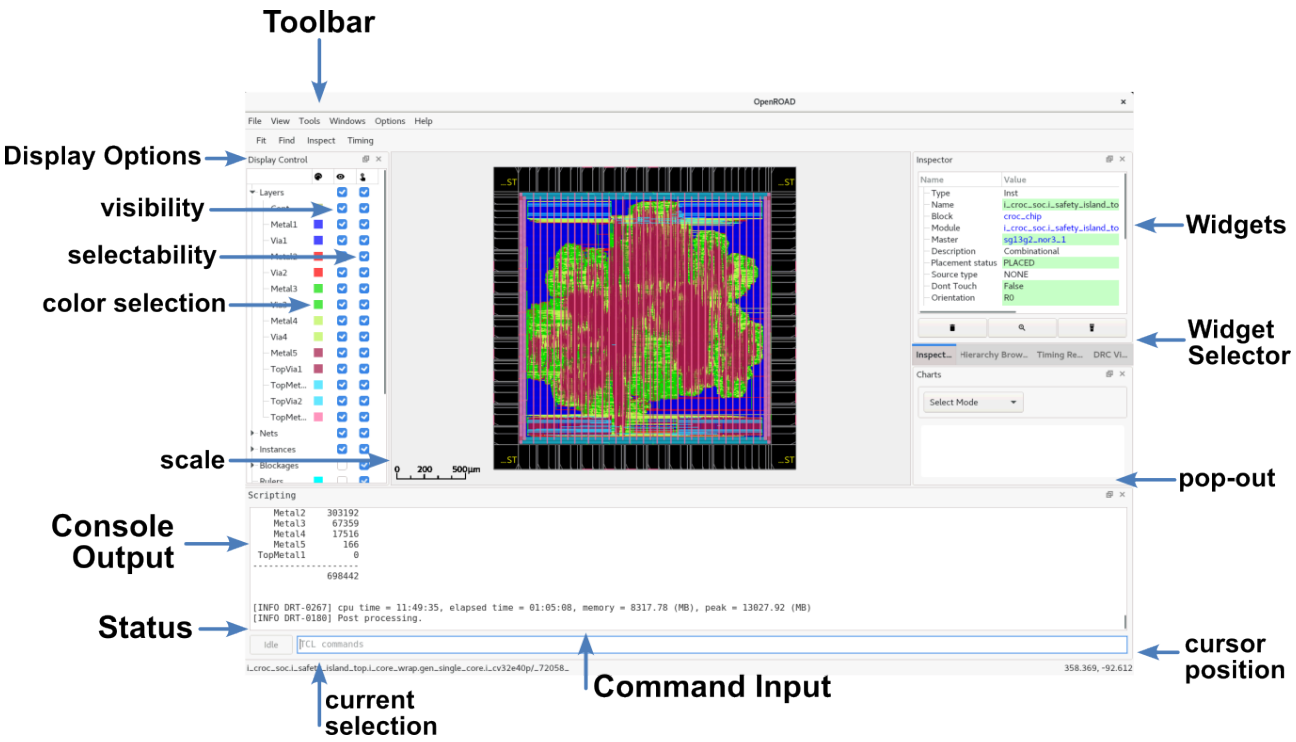

Figure 2: Important elements of an OpenROAD window.

The OpenROAD GUI should appear. The most important part is the 'Scripting' pane at the bottom. Here you can enter commands and it shows you their outputs (the same output is also in the terminal window you started OpenROAD with). You can pop it out or re-arange it using the symbol with the two windows on the right side of the 'Scripting' pane. You will be using the console more than you think. Make sure that it remains accessible and visible at all times! If you accidentally close it, you can open it via the toolbar at the top by selecting `Windows→ Scripting`. Similarly, if you decide to start OpenROAD without the gui (without the -gui flag), you can open the gui using:

```
OpenROAD > gui::show
```

Note also that while commands are running, the GUI will be frozen. So if you think OpenROAD has crashed, first check the console output and your terminal for activity.

4.1 Terminology

Before we start to explore the example design, some notes about the terminology:

Standard Cell / Cell Building block that implements a logic gate (and, or, flip flop, latch, etc.) and will be placed (aligned in rows) in the core area.

Module Is equivalent to the entities in VHDL or modules in Verilog. It contains a level of design hierarchy.

Instance One specific copy of a building block that is part of your design netlist. Identified by a unique instance name, e.g. 'i_croc_soci_croci_core_wrapi_ibex.cs_registers_imstatus_q_1__reg' or 'clkbuf_0_soc_clk_i'. Note: Also used for Verilog modules used in your design netlist.

Macro-cell / Block Building block that is typically larger and more complex than the standard cells. This could be a sub-design that has been completed previously or a full-custom block like the RAMs used in this example. Typically placed somewhere in the core area.

Pad Building block that is used to connect the core of the chip to the external world. Placed in the IO region around the core.

Master A building block available from a library, e.g. 'RM_IHPSG13_1P_256x48_c2_bm_bist', 'NAND2X1', 'INVX2' etc. Note: Also used for Verilog modules defined in your design netlist.

Row All standard cells have the same height. This allows them to be placed in (horizontal) rows. In back-end terms, a row defines a region where standard cells can be placed.

ITerm A logical connection point (port) of a module or cell, e.g. the inverter 'INVX2' has two terminals ('A' and 'Y').

⟨Inst⟩ Pin A physical connection point/shape of an instance, i.e. where the router contacts the instance.

Net The logical/signal interconnection between instances.

Special Net All instances need to be connected to power and ground. Technically these connections are also nets, but they are treated differently from regular nets. These nets are called special nets.

4.2 Exploring the Design

Now let us restore a completed database. We use a design called 'Croc' you will follow this design through all backend steps in the coming exercises. This design was just recently started and likely still has various problems or room for optimization, we welcome any feedback in this direction. In the simulation exercise you will see what exactly it contains, for the purposes of this exercise, it is just a random design you were handed.

If you are a more advanced user, you may want to use our big open-source Linux capable chip called Basilisk instead. Be warned, it is much larger which may lead to slowdowns when using the GUI.

For our flow we use a script to create checkpoints after all major steps. It writes out all relevant files and zips them together. This allows us to trace down issues we encounter in the flow.

The most important files are the following:

- **Database**: files ending in **`.odb`** save OpenROADs database format of the design. It contains the physical design along with the netlist in its internal format.
- **Netlist**: files ending in **`.v`** are netlists, they are not necessary for reloading a design but they are very convenient to explore what changed compared to the last checkpoint. At any point during this exercise you may want to open the netlist and compare it with what you are seeing in OpenROAD.
- **Constraints**: files ending in **`.sdc`** are Synopsys Design Constraints. They describe timing limitations the design must meet at any point. Most importantly they contain the definition of all clocks used. After loading the database you may want to load the constraints (`read_sdc`) in order to restore timing information.
- **Technology**: Not in the checkpoints are the technology files, these describe the functionality and physical design of all the cells and macros used in the design as well as the geometry of the metal layers used to connect cells together.
  You can reload the technology information at any point using the script **`scripts/init_tech.tcl`**

Student Task 2:

- Select `File→ Open DB` from the menu. The design is stored in OpenROAD's open-database format, it is stored as '**`croc.odb`**' inside the '**`out/`**' directory.

Note 2: You can also load *.odb* files in OpenROAD via the `read_db <file>` command.

After loading the database you will see a routed chip similar to Figure 3 on the following page).

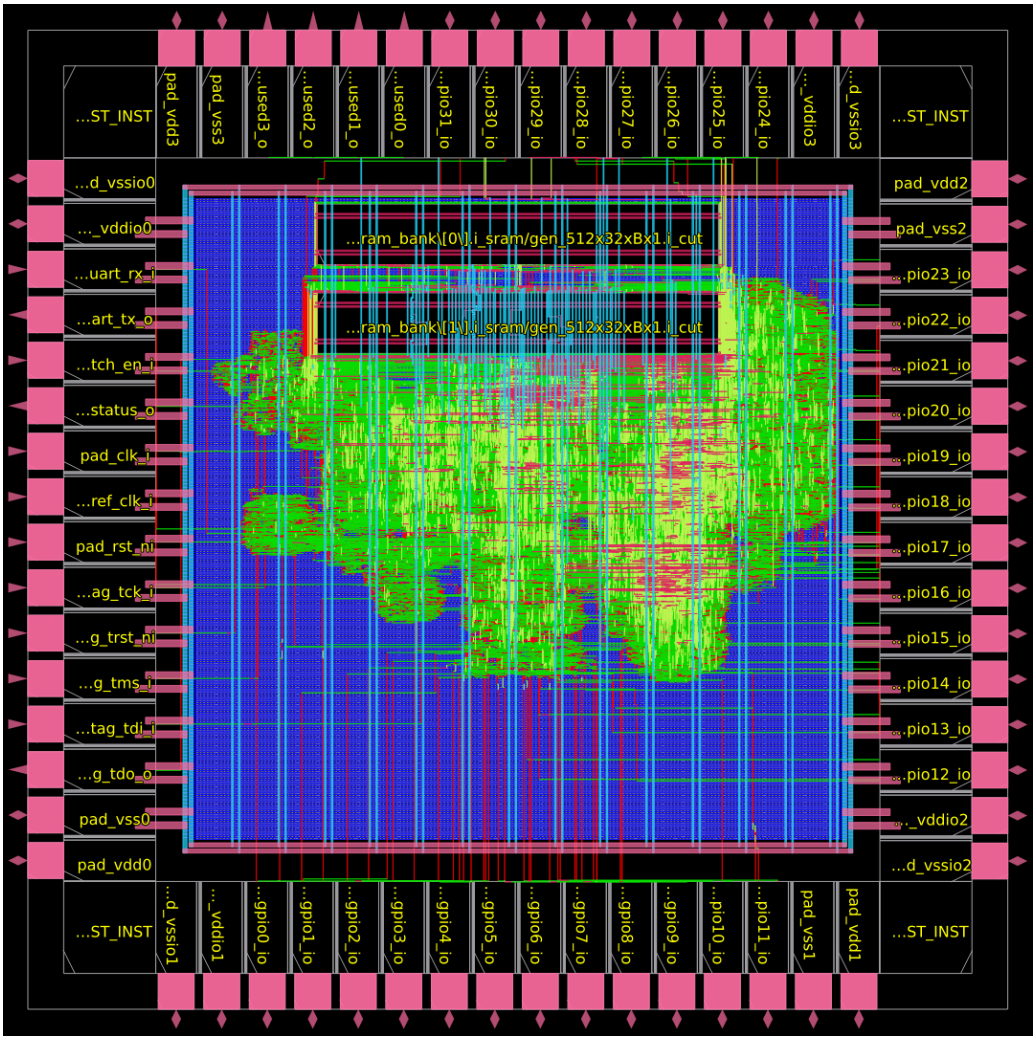

Figure 3: Croc chip loaded in OpenROAD.

4.3 Basic Navigation

You can zoom in by using Ctrl + or by drawing a rectangle with clicked. If you want to go back to seeing the full chip, either use 'Fit' from the lower toolbar or with the keyboard shortcut F. You may also notice that some letters in the toolbar are underlined (eg V for 'View'), you can press Alt and the highlighted letter to open this toolbar menu, then use up and down keys to navigate the dropdown.

Pan by pushing and moving the cursor, by using (up and down), using Shift ⇑ + (up an down by one screen height) and using .

There is also a menu with some options, most importantly the `Clear` submenu where you can clear highlights, selected objects or the rulers currently displayed. can remove all the rulers by pressing Shift ⇑ + K. To see (or change) keyboard shortcuts go to the menu "`Design` →`Preferences...`"

Student Task 3:

- To get a better impression of the size, let us try to measure the chip dimensions. First we will have to select the 'ruler' tool from the toolbar in `Tools`→ `Ruler` or by using its keyboard shortcut ' K '. The units are in micrometers. How long are the sides of the chip?

  Sample solution:

  $h = 1916\,\mu m$
  $w = 1916\,\mu m$

- When you look at the chip, you realize that a significant part of the area is used for the bondpads and IO-cells. The area in the center is called core area, and in between the core area and the pad ring is a set of power wires called the core power ring. What is the ratio between core area (with power ring) and the total area?

  Sample solution:

  You get the answer by measuring the 'thickness' of the padring and then:
  $A_{total}/A_{core} = 3.7\,mm^2/2.0\,mm^2 = 1.83 \Rightarrow 83\%$ larger
  `Note`: After OpenROAD we add a 'seal ring' around the entire chip bringing the total size to $2000\,\mu m$ making the difference even larger.

- There are two identical SRAM blocks placed in this design. What is the area of such a RAM module? In this example, the RAM holds 256 words of 64 bit each. What is the bit density in this technology, in other words how many bits can you store per mm^2?

  Sample solution:

  $A = 93\,180\,\mu m^2 = 0.09\,mm^2$
  Either from measuring using the ruler, or by clicking on a SRAM and using the bounding box, or by using the command line (take a look at **`scripts/01_floorplan.tcl`**)

  Sample solution:

  $\rho = 172\,Kibit/mm^2$

4.4 Display Controls

The design we have loaded, is already routed and you can see a jungle of wires on the metal layers. You will need to zoom in really close to differentiate individual connections. A modern IC manufacturing technology

provides several metal layers for routing, the technology we use for these exercises has 5+2 (two are TopMetal which are significantly thicker). Each metal layer is separated by an insulating layer. In this way separate signals can be carried on different metal layers. To move a signal from one metal layer to another a special connection named 'via' is used. Each via connects two adjacent metal layers. In this technology multiple vias can also be stacked (placed on top of each other), so that it is possible to move from the top most metal (TopMetal2) to the bottom most metal layer (Metal1) at one point[15].

There are several colored squares in 'Display Control' on the left hand side of the GUI. Each square shows the color of an object category in the design. The object category is written on the left side of the colored square. On the right side of the squares you can see two check boxes. The first check box is used to toggle the visibility of an object category. Let us take the object category 'Nets'. This category contains the various type of connections (power (Power for Vdd and Ground for Vss), clock and data/signals). Click on the first check box of Signal so that the box is not selected. You will see all data interconnections disappear. Only the clock tree and the power distribution remains. Turn off visibility for Power and Ground as well, you can now see the placed standard cells in light-gray with the clock tree on top. The second box determines whether or not you will be able to select objects belonging to that category. Remember that you can only select objects that are visible. We recommend you turn off selectability for Power and Ground, it is very easy to accidentally click on them and select this very large net (turning off Metal1 also helps a lot with this and is a good alternative).

We will not explain all object categories (you can explore it yourself) but we highlight the most important ones. The category 'Instances' will control all instances (Cells, Pads, Macros,...) in your design, and 'Misc/Instances' controls various parts of how instances are shown. 'Misc/Highlight selected' controls highlighted objects and 'Heat Maps' contains various different overlays.

Student Task 4:

- There are 5+2 metal layers in this process. Typically, each metal layer is used predominantly in one direction (either horizontal or vertical). For the metal layers 2 to 5, find out in which direction it is running.

  You will need the section 'Layers' in the layer control area on the right to switch the physical layers on and off.

  Sample solution:

M5	horizontal
M4	vertical
M3	horizontal
M2	vertical

- When looking at the individual metal layers, do you notice any exceptions in the direction of the wiring? Why or why not?

  Sample solution: Small short-distance adjustments are made in non-typical direction for local wiring; this is called *same-layer jogging*

- Follow the power connections of the SRAM. On which metal layer is the connection, where does it connect to?

  Hint: You can switch off the visibility of the majority of connections ('Nets') to see only power and ground. Additionally, you can hide all instances except Macros ('Instances/Macros') You might also want to toggle the pins of the RAM block ('Misc/Instances/Pins').

  Sample solution: The many, many power pins of the RAM are on Metal4, and they are connected to the power stripes on TM1 and then TM2

[15] Note that there will be 6 separate vias that are placed on top of each other for this to happen. This will block all in-between metal layers (Metal2, Metal3, ..., TopMetal1) at this point as well.

- Switch on a single metal layer in vertical direction. You should notice a very distinctive, regular pattern all across the chip (you need to make all 'Nets' visible again if you have switched it off before). You probably have noticed that the main power distribution grid is on layer TM1 and TM2 with very wide tracks. Yet it clearly also seems to influence the routing on other metal layers. Can you explain this?

  Sample solution: Very little routing directly below the power lines on TM1 since they are blocked by vias going down from TM1 to Metal1.

Don't forget to turn back on all layers with the checkbox next to 'Layers' at the top.

4.5 Overlays

OpenROAD has various different overlays you can activate. Each is designed to give you further insight insight into your design. The first one we will look at it is the `Module view`. It shows you where certain blocks of your design are placed. Typically each module in your RTL performs a smaller part of the overall chips functionality. This means the logic in these modules is closely related to each other, in the chip this often means they form large blobs (commonly called amoeba) of standard cells. This is exactly what the `Module view` shows you.

Student Task 5: Continuing from before, besides the vertical pattern, it seems that there are clear borders between different clusters of routing. If you turn on Metal5 you can also see that its only used in certain locations. To analyze the source of this, in the Widget view select '`Hierachy Browsers`' on the right of the main window. Then click on '`Module view is no enabled`', to enable this view (you can also find it under 'Misc/Module view').

You can select and de-select each module or change their colors. Zoom in to various areas and toggle 'Misc/Module view' on and off.

Can you now explain these clear borders in the routing layers?
ie routing is divided into 'blocks', why?

Sample solution: Areas that form boundaries between blocks (instances) are sparsely routed. Hierarchical designs usually have blocks with very dense routing/logic and relatively few connections between them.

Related to the exploration of wiring is the concept of 'routing congestion'. Same as on the street, congestion is bad especially if it crosses a certain threshold and becomes too big to handle. Routing congestion tells us how many vertical and horizontal routing resources are used in some area. You can display the associated heat map by selecting 'Heat Maps/Routing Congestion'. But first, we recommend you turn off visibility of Power and Ground nets to get a cleaner overlay. To make it a bit easier to read, we can also change the heat map settings under `Tools→ Heat maps→ Routing Congestion`.

I recommend you turn on 'Show numbers' (displays the percentage in each grid-cell when zoomed in) and 'Show legend'. Play with the Minimum and Maximum until you find values that give you a nice picture.

`Note`: You probably want to make sure 'Misc/Module view' is turned off, otherwise the color will bleed through to the overlay.

Student Task 6:

- Compare the heat map with the routing on Metal5, do you notice something?

  Sample solution:

  Metal5 routing is primarily used in areas with higher routing congestion. Generally the router tries to first use the lower layers before going to higher ones.

- OpenROAD also has a congestion estimation algorithm called RUDY, which is typically used after

placement to get a quick estimate of the expected routing congestion without having to perform global routing. Turn on its heat map instead and compare the two, do they match?

Sample solution:

The intensity (exact value) of the congestion may be different but where the congestion will occur is usually fairly accurate, so the overall pattern should look similar.

4.6 Instance Inspector & Search

Using the `Hierarchy Browser`, the `Inspector` (also a widget) and the `Find` function (top-left or `CTRL+F`) you can browse through the logical hierarchy of the design, highlight and select instances and nets. To find objects using the search command it is also possible to use wild cards ('*' and '?': the '`*`' matches any number of characters, a '`?`' can be used to match exactly one character.).

However, often you need to do more than just find instances, instead you need to be able to see their connectivity to the next few components. We examine one option to do so with the next task.

Student Task 7:

- We start at an SRAM, click on one and it will populate the `Inspector` widget on the right. Clicking on the blue text in the Inspector will change the selection to this object.
- Click/hover on the 'Master' cell, this should select all SRAMs
- Click on any instance in the selection shown in the Inspector, we are back at just one SRAM.
- Expand 'ITerms', this shows you all SRAM pins (left) and the connected net (right)
- Click on one of the nets connected to a data pin. The net is now selected (shown in yellow).
- At the bottom of the Inspector you have various buttons. Try them out, what does each one do?
  - How can you zoom in to the net?
  - How can you add it to a highlight group?
  - How can you show only this net?
- Navigate to a pin of some instance (select some instance and navigate to the correct ITerm, or display the Pins of instances, zoom-in and click directly on a pin)
- Display all connections to/from this pin using the button of the Inspector

  Using this newly acquired knowledge, try to figure out how the two 64-bit wide SRAMs are connected to the 32-bit system.
  Hint: Explore the connectivity of a few neighboring data-pins into and out of the SRAM and the logic function of connected cells.

  Sample solution: The two SRAMs share a lot of connections (seen eg by going to a A_DIN pin, then navigating to the driving pin and showing the fanout)

- Somewhere in the design is a cell called 'something-something-`timeout_count_q_0__reg`', find it and figure out in which module it is

  Sample solution: Using the Find function, you can enter '`*timeout_count_q_0__reg`', this will bring you to the cell. In the inspector under 'Module' it tells you in which module it is.

4.7 Clock Tree

Another important part of any design are the clocks. They are designed in a special way to make sure all the flip-flops receive the clocks at the same time. In particular they are implemented using tree structures, hence

the name Clock-Tree.
OpenROAD can give us various views of the clock-tree but first we need to actually tell it where our clocks are. For this we need to load a constraints file where we define the clocks (among other things). However, before we can do so we should first reload the technology files to make sure everything is present and configured properly.

Student Task 8:
So far we have loaded the technology (the **`startup.tcl`** script already loads **`init_tech.tcl`** which performs the technology setup) and the design stored in **`croc.odb`**. The timing information is not stored so we need to recreate it.

- Now load the constraints for this design

```
OpenROAD > read_sdc out/croc.sdc
```

- Finally, open the Clock Tree Viewer widget and click on Update

You should now be looking at a schematic view of the clock tree(s) in the widget as well as having the clock tree highlighted on the chip. The highlighted nets are divided into two colors, they always correspond to the left and right branch shown in the schematic view.

Student Task 9:

- Navigate to different buffers in the schematic view and click on them, what is shown in the chip view?
- Starting from '`timeout_count_q_0__reg`' again, find the clock buffer it is driven by and in the schematic view see where in the clock-tree it is located.

4.8 Further Help

These exercises will only explain the basic functionality of OpenROAD, for additional help please use the 'Help' function in the toolbar. It will lead you to the OpenROAD documentation, where you can find additional information.

Newer versions of OpenROAD also have a `man` command to show you helpful information about commands and there is a online documentation available where the most important commands and interactions are explained.

4.9 A Last Note

Back-end design is an iterative process. The decisions you will make at the beginning of back-end design may have a profound impact on the outcome. However, you will not always be able to predict this. You will have to complete individual stages of back-end design (floorplaning, placement, routing etc) and then assess the quality of the result. If you are not satisfied with this result you will have to repeat the stage or even the entire process. This is normal, be prepared for it. Notice that each time you repeat a certain stage, you will gain more experience and will be able to work faster.

$\mathcal{E}$ **You are done with this Exercise.**
Discuss your experience with assistants and colleagues or continue to explore OpenROAD and the provided design on your own. $\mathcal{E}$