

Chapter 0 - Preparations

(Optional)

Course authors (Git file)

- 1 Description
- 2 Install Option A: OpenROAD Flow Scripts (ORFS) on your computer
- 3 Install Option B: IIC-OSIC-TOOLS in a docker container on your computer

Section 1

Description

Description

About chapter 0 - Preparations:

- This chapter is not needed when you participate in an on-site course at IHP.
- The tools and computers at IHP are ready-to-use.

This chapter can help you if:

- You want to do the course on your own.
- You want to give this course as a trainer.
- Want to have everything running in your own environment.

Here comes a short description of the options, followed by their detailed guides:

Option A: OpenROAD Flow Scripts (ORFS) on your computer

- A plain installation of OpenROAD, Yosys, Klayout and some flow scripts into your system.
- This option puts everything directly under your control and only installs the minimum toolset necessary for the course.
- It requires the permissions to install software on your computer.
- The guide makes use of Ubuntu Linux.

Option B: IIC-OSIC-TOOLS in a docker container on your computer

- This docker container is like a swiss knife for EDA tools. It can be configured in many ways and contains a lot of useful tools.
- All the tools for the course are in it.
- It requires the permissions to install software on your computer.
- The guide makes use of Ubuntu Linux.

Section 2

Install Option A: OpenROAD Flow Scripts (ORFS) on your computer

Install Option A: OpenROAD Flow Scripts (ORFS) on your computer

- This guide is a list of shell commands with some short explanations and weblinks.
- This was tested on a freshly installed Ubuntu LTS 24.04.1.
- The order of the commands is crucial and must not be skipped.
- For more explanations look into the documentations and README files of the tools. The weblinks are given.

Prerequisites:

- Ubuntu LTS 24.04.1 (should work on other Linux too, see weblink)
- Permission to install software (sudo rights)
- Reliable internet connection
- git installed: `sudo apt install git`

Weblink for detailed information:

<https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/blob/master/docs/user/BuildLocally.md>

Your install folder

Navigate to a folder where you want the installation to reside in. The install will need some Gigabytes space.

```
1 | cd <INSERT PATH TO YOUR INSTALL FOLDER HERE>
```

Clone the ORFS repo

Clone the repository to your computer:

```
1 | git clone --recursive https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts
```

Run the setup script

Run the setup script to install the dependecies:

```
1 | cd OpenROAD-flow-scripts
```

```
1 | sudo ./setup.sh
```

Build the tools

Build all tools. This will take a while, depending on the computer:

```
1 | ./ build_openroad.sh --local
```

Verify the builds

Verify that the tools are available. You should get version informations of the tools with the following commands:

```
1 | source ./env.sh
```

```
1 | klayout -v
```

```
1 | yosys --version
```

```
1 | openroad -version
```

Section 3

Install Option B: IIC-OSIC-TOOLS in a docker container on your computer

Install Option B: IIC-OSIC-TOOLS in a docker container on your computer

- This guide is a list of shell commands with some short explanations and weblinks.
- This was tested on a freshly installed Ubuntu LTS 24.04.1.
- The order of the commands is crucial and must not be skipped.
- For more explanations look into the documentations and README files of the tools. The weblinks are given.

Prerequisites:

- Ubuntu LTS 24.04.1
- Permission to install software (sudo rights)
- Reliable internet connection

The IIC-OSIC-TOOLS docker container:

With the following steps a preconfigured docker gets installed. The docker is created and maintained by: Institute for Integrated Circuits (IIC) at the Johannes Kepler University Linz (JKU) and is available in their Github with more detailed installation instructions:
<https://github.com/iic-jku/IIC-OSIC-TOOLS>

Step 1: Install docker with apt:

Weblink for detailed informations:

<https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository>

Add Docker's official GPG key:

```
1 | sudo apt-get update
```

```
1 | sudo apt-get install ca-certificates curl
```

```
1 | sudo install -m 0755 -d /etc/apt/keyrings
```

```
1 | sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
```

```
1 | sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Add the repository to apt sources:

```
1 echo \  
2   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]  
3     https://download.docker.com/linux/ubuntu \  
4     $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \  
5     sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
1 sudo apt-get update
```

Install the latest version of docker:

```
1 | sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Step 2: Manage docker as a non-root user

Weblink for detailed informations:

<https://docs.docker.com/engine/install/linux-postinstall/#manage-docker-as-a-non-root-user>

```
1 | sudo groupadd docker
```

```
1 | sudo usermod -aG docker $USER
```

```
1 | newgrp docker
```

Step 3: Run the hello-world docker:

Run the hello-world example docker (without the need of sudo user):

```
1 | docker run hello-world
```

No errors should be displayed in running the hello-world example. The output in the shell should contain this message:

```
1 ...
2 Hello from Docker!
3 This message shows that your installation appears to be
4 working correctly.
5
6 To generate this message, Docker took the following steps:
7 1. The Docker client contacted the Docker daemon.
8 2. The Docker daemon pulled the "hello-world" image from
9    the Docker Hub. (amd64)
10 3. The Docker daemon created a new container from that
11    image which runs the executable that produces the
12    output you are currently reading.
13 4. The Docker daemon streamed that output to the Docker
14    client, which sent it to your terminal.
15 ...
```

Step 4: Clone the IIC-OSIC-TOOLS git repository to your computer:

Weblink for detailed informations about the steps 4 - 5:

<https://github.com/iic-jku/IIC-OSIC-TOOLS/blob/main/README.md>

Install git:

```
1 | sudo apt install git
```

Navigate to a folder where you want the repository to be in:

```
1 | cd <INSERT PATH TO YOUR FOLDER HERE>
```

Clone the IIC-OSIC-TOOLS:

```
1 git clone --depth=1 https://github.com/iic-jku/iic-osic-tools.git
```

Step 5: Start the docker

```
1 | ./start_x.sh
```

A shell window pops up, in which the docker runs.

Step 6: Get the OpenROAD flow scripts

- To be written
- This should be matching to option C (IHP server)
- Waiting on IHP information about their docker / server install.



Mon

L1:
Introduction

T1:
Training

Tue

Q1, Q2:
Recap
Feedback

L3:
Verilog

T3:
Training

Wed

Q3, Q4:
Recap
Feedback

L5:
PDK

T5:
Training

Thu

Q5, Q6:
Recap
Feedback

L7:
OpenROAD
Flow scripts

T7:
Training

Fri

Q7:
Recap

L8:
Tapeout

Feedback



L2:
OpenROAD
tools

T2:
Training

L4:
OpenROAD
first run

T4:
Training

L6:
OpenROAD
GUI

T6:
Training

L7:
OpenROAD
Flow scripts 2

T7:
Training

Spare time
and
Wrap-Up

Chapter 1 - Introduction and overview

Course authors (Git file)



- 1 Welcome
- 2 Course overview
- 3 Course components
- 4 Feedback and Cheat Sheets
- 5 The Training sessions
- 6 Certificate
- 7 Open-source EDA for digital designs



Section 1

Welcome



The next 5 days

- A few words to start with.



Trainer profile

Me:

Name, Company / Uni

Why i'm here. My motivation.

What i've done before.

What interests me most.



Participants backgrounds and motivations

You:

Name, Company / Uni

Why i'm here. My motivation.

What i've done before.

What interests me most.



Section 2

Course overview



Chapter names

- 1 Introduction
- 2 OpenROAD tools
- 3 Verilog
- 4 OpenROAD first run
- 5 PDK
- 6 OpenROAD GUI
- 7 OpenROAD flow scripts
- 8 Tapeout



Schedule for the course

Mon	Tue	Wed	Thu	Fri
L1: Introduction	Q1, Q2: Recap Feedback	Q3, Q4: Recap Feedback	Q5, Q6: Recap Feedback	Q7: Recap
T1: Training	L3: Verilog	L5: PDK	L7: OpenROAD Flow scripts T7: Training	L8: Tapeout Feedback
				
L2: OpenROAD tools	L4: OpenROAD first run	L6: OpenROAD GUI	L7: OpenROAD Flow scripts 2 T7: Training	Spare time and Wrap-Up
T2: Training	T4: Training	T6: Training		

L : Lectures

T : Training and
Hands-On

Q : Questions

Section 3

Course components



Get the course materials here:

Course materials (Release):

<https://github.com/OS-EDA/Course/releases>

- Download the latest release
- Unpack into a directory
- There might be daily updates during the course week!



Additional course related links:

OS-EDA Github organization:

<https://github.com/OS-EDA>

Course Github repository:

<https://github.com/OS-EDA/Course>



Duplicated content versus internet links

The course slides

- contain Links to the Internet for a lot of topics.
- do not contain duplicated content (or as less as possible).

This means:

- Follow the links and read there. It is important content for the course.
- The links are carefully curated. It's not spamming.
- Don't expect all the content being duplicated into the course slides.

A brief discussion about pros and cons of this.



Lectures



Lectures:

- All the chapters start with a lecture slide deck.
- The trainer will walk you through the content of the lectures.
- Whenever you have a question inbetween: ask directly.
- The lectures contain the base knowledge of the course.



Trainings



Common training tasks:

Every training sessions starts with the common part. The tasks of the common part are sufficient to follow along the content of the course. If you're a beginner, these trainings should be your goal to reach.



Advanced training tasks:

The advanced training sessions are for those With pre knowledge. If the common training was finished fast or was just to easy, the advanced sessions get you covered.



Bonus training tasks:

Still time left to do some tasks? Want something to take with you as homework? Please enjoy the bonus rounds of the training sessions.

Questions



Questions:

- The questions are for re-visiting and remembering a previous chapter.
- They guide an interactive session between the trainer and the room:
 - Trainer: Asks the questions.
 - Room: Answers the questions.
 - Skipping a question is fine.
 - Not knowing the answer is fine.
 - This is not an exam, not a test and not a challenge.
 - It is meant as a helpfull and hoepfully enjoyable way to recap yesterdays content.
 - If no answer is found, the trainer helps with the answer.



Section 4

Feedback and Cheat Sheets



Feedback and Cheat Sheets

- We please you to give us feedback for the course.
- There is a short timeframe each day reserved for feedback.

We have two ideas about this:

- ① Developing Cheatsheets together
- ② Collecting general feedback



Cheatsheets



Some things are really hard to remember:

- Abbreviations
- Complex relations and graphics
- EDA tools workflow
- Schedule of the week
- Mathematics (joking, we're not doing math here)
- ...

- That is why we would like to develop Cheatsheets with you.
- They're made for cheating the hard parts.
- Cheatsheets work best when printed as handouts.
- One can have them nearby the computer while learning.



Cheatsheet example

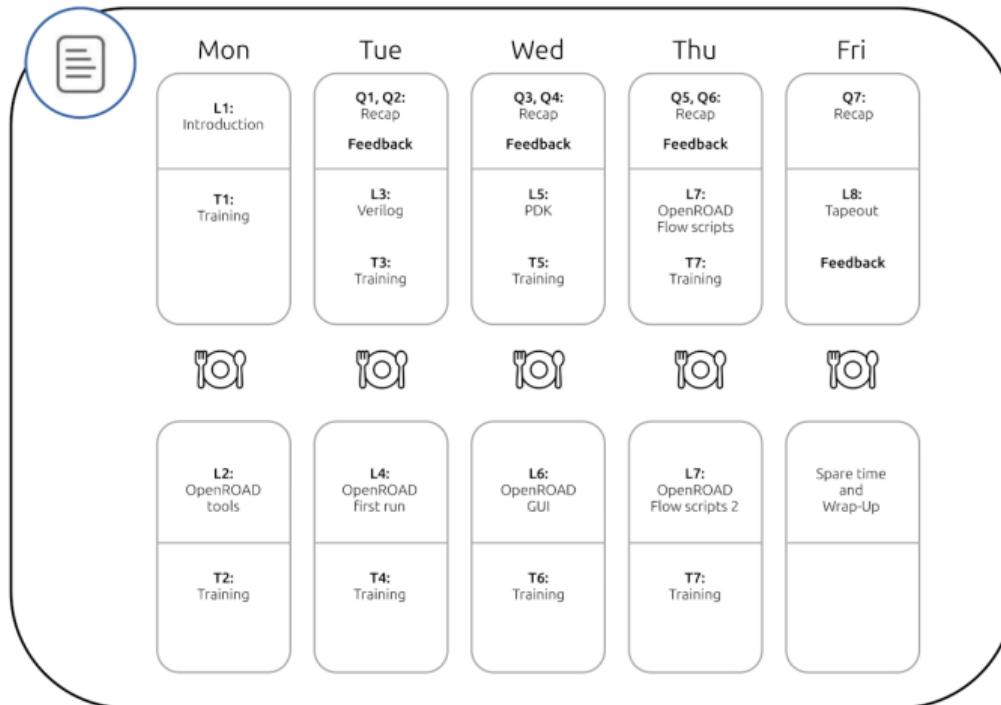


Figure 1: Cheatsheet Chapter 1



Cheatsheet example



**Cheatsheet - Chapter 5
PDK Examination**

KLayout:

.lyp	Layer properties file
.lyt	Technologies file for Layout ↔ Technology mapping

XSchema:

.sym	Schematics file

Abbreviations:

LVS	Layout versus Schematic
CDL	Circuit design language
GDS II	Graphic data system (II)
LEF	Library exchange format
techLEF	Additional info about the technology
.lib	Liberty timing file: ASCII descriptions of timing / power of cells.

Hardware Description Languages (HDL):

Verilog
VHDL

Figure 2: Cheatsheet Chapter 5



Empty Cheatsheet printversion

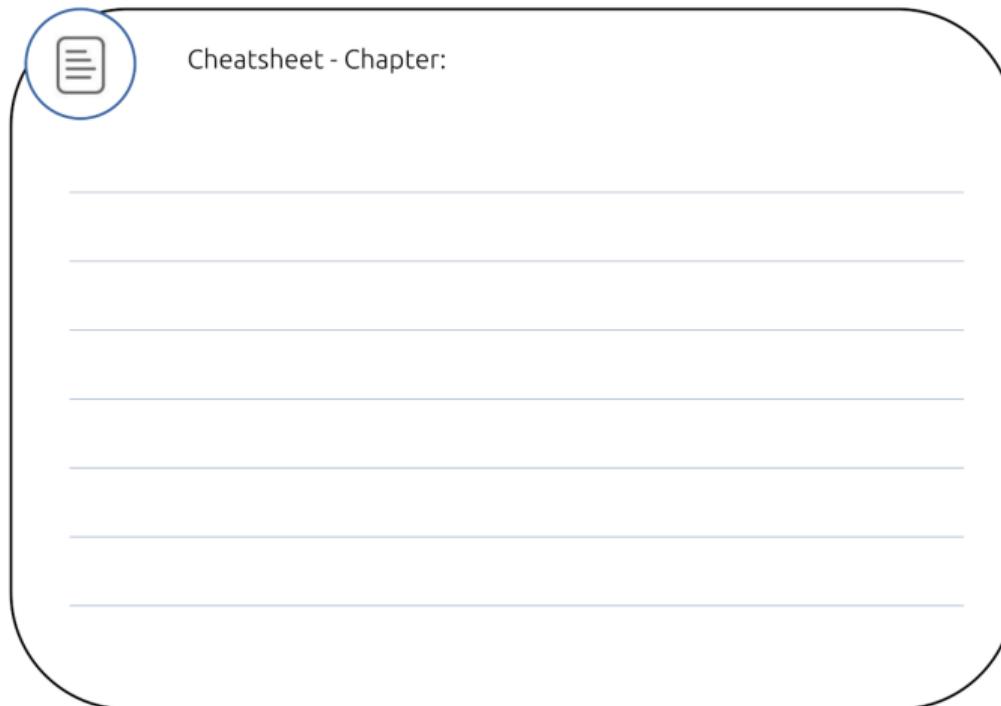


Figure 3: Cheatsheet Chapter 5

Empty Cheatsheet document link

Empty Cheatsheet OpenOffice doc:

https://github.com/OS-EDA/Course/tree/main/Chapter_01_Introduction/pics_lecture



General feedback

General feedback:

- The general feedback will be collected verbally in the room.
- Everyone has the opportunity to give feedback
- We will write down the feedback, without your name.



What will happen with your feedback?

- We will put the feedback into Github issues.
- Right now, think of Github issues as some sort of tracker- or ticket system.
- Your feedback will not be connected to you (Anonymous).
- You can join a public discussion in the Github issues, if you want to.

Weblink to the issues of the course:

<https://github.com/OS-EDA/Course/issues?q=is%3Aissue>



Section 5

The Training sessions



Login at IHP

- Onboarding to the computers for everyone



Levels

- The Trainings should create success points inbetween lectures
- This is going too slow for me:
 - Try the advanced and bonus trainings (even from past chapters)
- This keeps me busy every minnute:
 - Stick to the common trainings. You will be able to follow all chapters then.



Section 6

Certificate



Certificate

We will explain live about the certificate of the course.



Section 7

Open-source EDA for digital designs



Digital designs

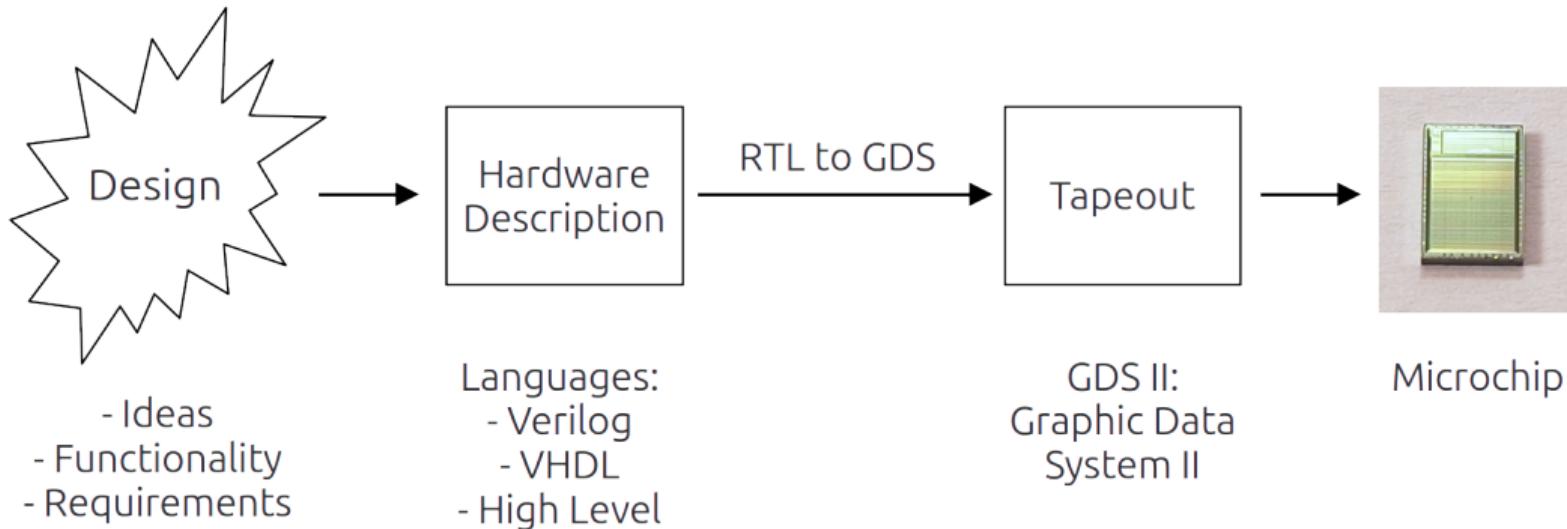
There are:

- **Digital designs** This course!
- Analog designs (Upcoming course)
- Mixed signal designs
- Artwork designs (i.e. Minimal Fab Contest)
 - <https://github.com/mineda-support/Semicon2023-MinimalFab-Design-Contest>
- Your fancy design?

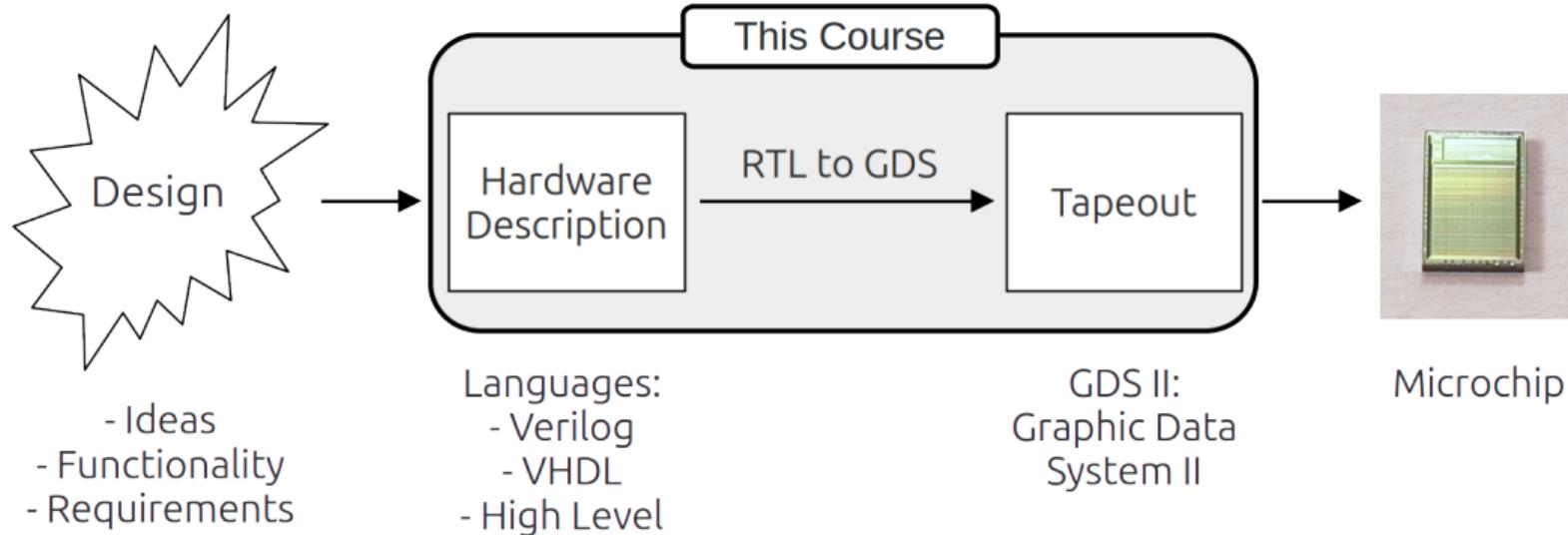
From now on: This course means digital design, even if not mentioned everywhere again



From design to microchip



RTL to GDS toolchain



RTL: Register Transfer Level

In [digital circuit design](#), **register-transfer level (RTL)** is a design abstraction which models a [synchronous digital circuit](#) in terms of the flow of digital signals ([data](#)) between [hardware registers](#), and the [logical operations](#) performed on those signals.

Register-transfer-level abstraction is used in [hardware description languages](#) (HDLs) like [Verilog](#) and [VHDL](#) to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived. Design at the RTL level is typical practice in modern digital design.^[1]

Unlike in software compiler design, where the register-transfer level is an intermediate representation and at the lowest level, the RTL level is the usual input that circuit designers operate on. In fact, in circuit synthesis, an intermediate language between the input register transfer level representation and the target [netlist](#) is sometimes used. Unlike in netlist, constructs such as cells, functions, and multi-bit registers are available.^[2] Examples include FIRRTL and RTLIL.

[Transaction-level modeling](#) is a higher level of [electronic system design](#).

RTL description [\[edit\]](#)

A synchronous circuit consists of two kinds of elements: registers (sequential logic) and [combinational logic](#). Registers (usually implemented as [D flip-flops](#)) synchronize the circuit's operation to the edges of the clock signal, and are the only elements in the circuit that have memory properties. Combinational logic performs all the logical functions in the circuit and it typically consists of [logic gates](#).

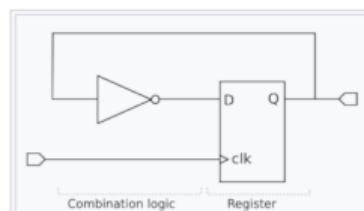


Figure 4: RTL (Screenshot from Wikipedia¹)

¹https://en.wikipedia.org/wiki/Register-transfer_level



GDS: Graphic Data System (II)

GDSII

⋮ A 7 languages ▾

[Article](#) [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▾

From Wikipedia, the free encyclopedia

GDSII stream format (GDSII), is a binary [database file format](#) which is the de facto industry standard for [electronic design automation \(EDA\)](#) data exchange of [integrated circuit \(IC\)](#) or [IC layout artwork](#).^[1] It is a [binary file](#) format representing planar geometric shapes, text labels, and other information about the layout in hierarchical form (two-dimensional/2D CAD file format). The data can be used to reconstruct all or part of the artwork to be used in sharing layouts, transferring artwork between different tools, or creating [photomasks](#).

GDSII

Filename extension	.gds
Developed by	Calma
Initial release	1989; 36 years ago
Type of format	binary ^[1]
Free format?	no

History [edit]

GDS = Graphic Design System (see [GDS78])

Initially, GDSII was designed as a stream format used to control integrated circuit photomask plotting. Despite its limited set of features and low data density, it became the industry conventional stream format for transfer of IC layout data between design tools of different vendors, all of which operated with proprietary data formats.

It was originally developed by [Calma](#) for its layout design system, "Graphic Design System" ("GDS") and "GDSII".

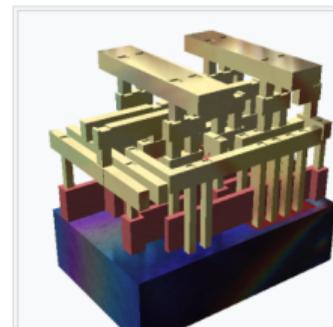


Figure 5: GDS (Screenshot from Wikipedia²)

²<https://en.wikipedia.org/wiki/GDSII>



The GDS II Format (Specifications)

Here are two links about the structure, format and elements of GDS II. The links are for reference reasons. It is not strictly necessary to read or learn the GDS II format for this course. But it might help understanding.

<https://boolean.klaasholwerda.nl/interface/bnf/gdsformat.html>

<https://www.rulabinsky.com/cavd/text/chapc.html>



Example of a GDS in KLayout

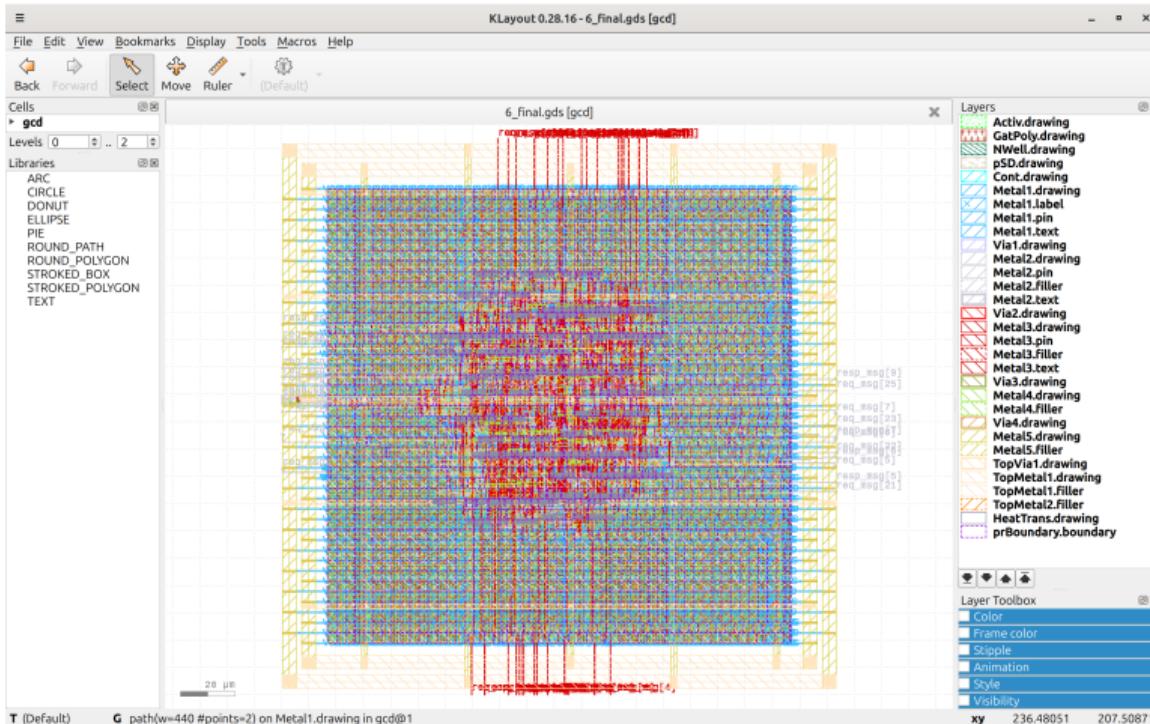


Figure 6: GDS in KLayout: gcd design with ihp130-sg13g2

Naming of RTL-to-GDS tools:

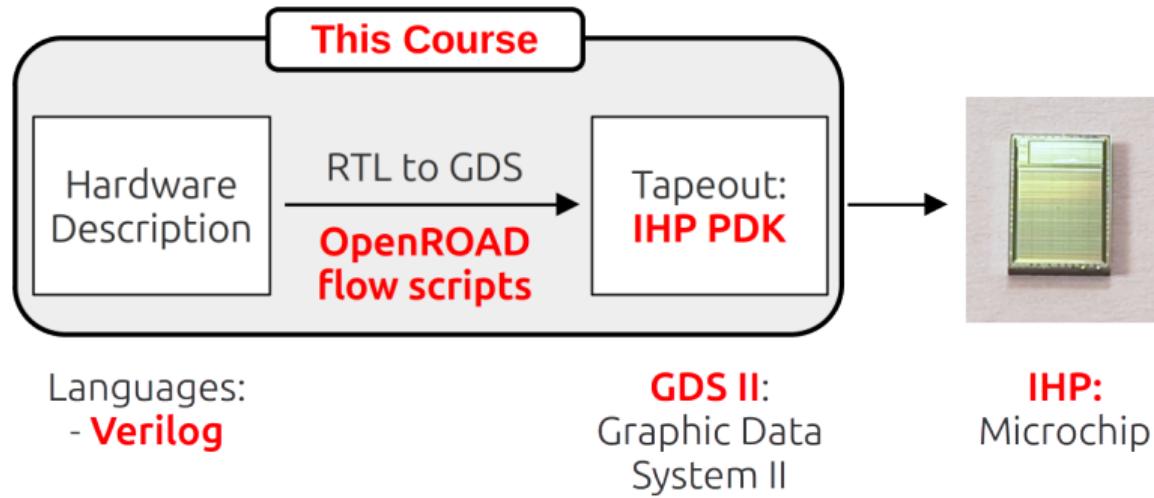
The naming of the tools is confusing:

- RTL-to-GDS
- = RTL-2-GDS
- = End-to-End-ASIC tools
- = End-to-End EDA toolchain

They all mean the same.



In this course: ORFS - OpenROAD flow scripts



Many open-Source RTL-to-GDS toolchains

Used with IHP PDK and in this course:

- **OpenROAD flow scripts**

which is based on

- **OpenROAD**

Most known other RTL-to-GDS toolchains:

- **OpenLANE** (based on OpenROAD)
- **OpenLANE 2** (based on OpenROAD)
- **Silicon Compiler** (works with many toolchains, close and open-source)
- **Coriolis** (developed at University Sorbonne, LIP-6)



A toolchain based on scripts and configuration files

OpenROAD flow scripts are

- based on scripts (obvious in the name)
- based on configuration files

Want most developers know from the commercial tools is:

- Graphical GUIs, used with a mouse and keyboard (shortcuts).
- Configuration through graphical masks, windows, forms.

This might feel uncomfortable at the beginning. But it still has some advantages.



Section 8

About open-source EDA



Advantages of open-source in EDA

- A word by Andrew Kahng (head of OpenROAD) about the relevance of open-source EDA

Andrews slides from the keynote speech at the Chipdesign Network June 2024. As the ucsd server is down, this is a link to the wayback machine:

<https://web.archive.org/web/20240609214401/https://vlsicad.ucsd.edu/NEWS24/InnovationKeynote-v6-ACTUAL-DISTRIBUTED.pptx>

Slides to have a eye on:

- Slide 06: EDA is an optimization problem
- Slide 10: Open-source EDA and disruptive Innovations
- Slide 13: Open-source accelerates EDA
- The complete chapter “Optimization and Virtuous Cycles” starting at slide 25
- Slides 39-43 about AI and proxies.

Some aspects of open-source EDA

- Three well known PDKs are open-source and production-ready.
- Some other open-source PDKs are not that visible or prominent (MiniFab, Pragmatic(soon?), ...)
- More than one RTL-to-GDS toolchain is production tested.
- Academia starts teaching a lot with open-source EDA.
- Building microchips with open-source became easy and affordable.
- No NDAs, No licence costs, Start with a laptop and internet.



What people have done with open-source EDA

- The following slides contain some works that were made with open-source EDA tools and open-source PDKs.
- All the pictures represent GDS data in different ways.
- A few slides back a picture of GDS in KLayout was shown. The following GDS pictures were created with various other open-source tools (Blender, 3d prints, STL files, ...).
- Most of these works would not have been possible to integrate here in closed source (because of NDAs and licenses)
- Open-source EDA drives people to experiment and play with the technology.





Figure 7: 3d cell parts³

³Picture by T.Knoll under Creative commons



Figure 8: 3d cell inverter⁴

⁴Picture by T.Knoll under Creative commons



Figure 9: 3d cell AND4_1⁵

⁵Picture by T.Knoll under Creative commons

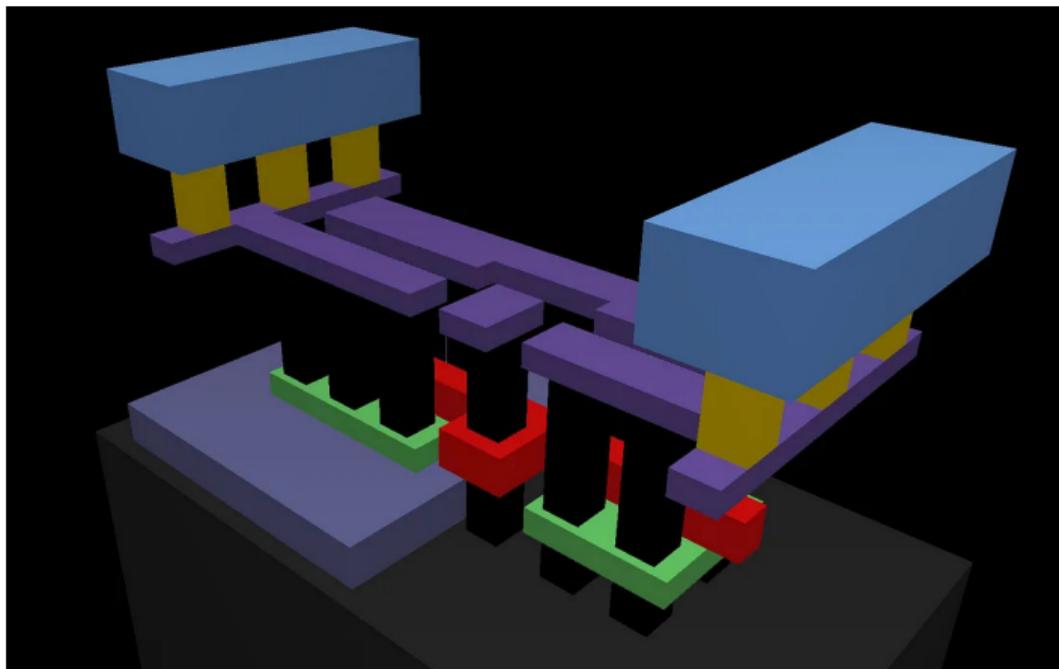


Figure 10: inverter⁶

⁶Picture by T.Knoll under Creative commons

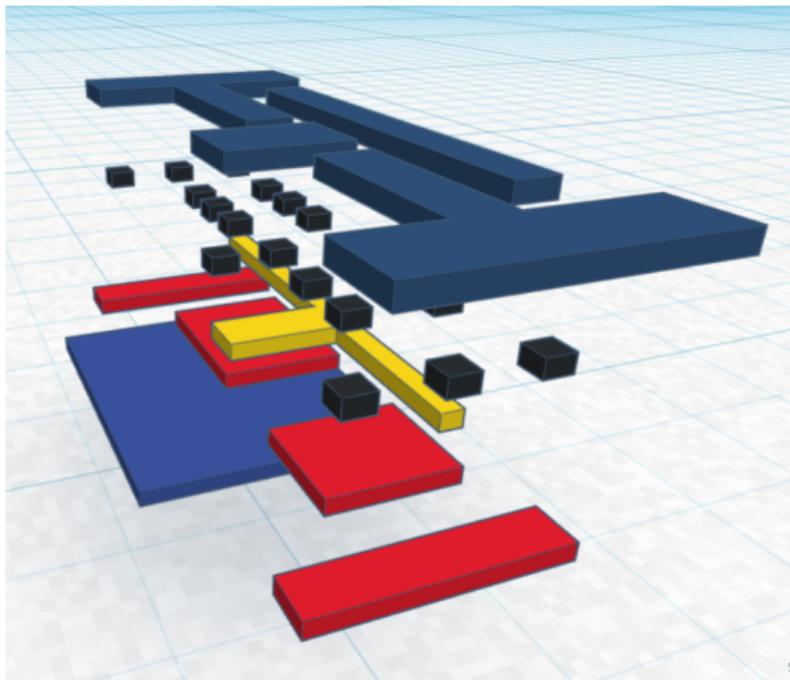


Figure 11: inverter⁷

⁷Picture by T.Knoll under Creative commons

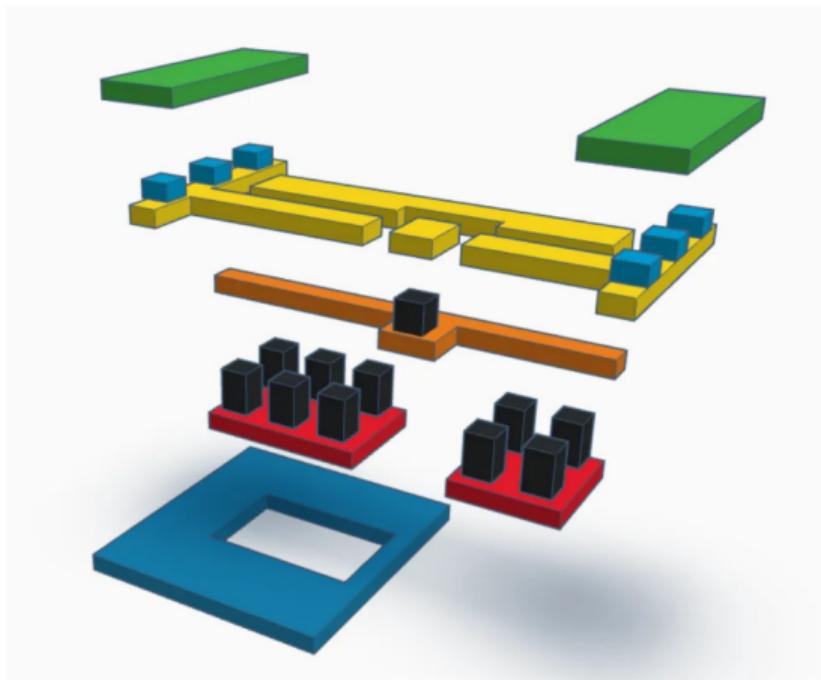


Figure 12: inverter⁸

⁸Picture by T.Knoll under Creative commons

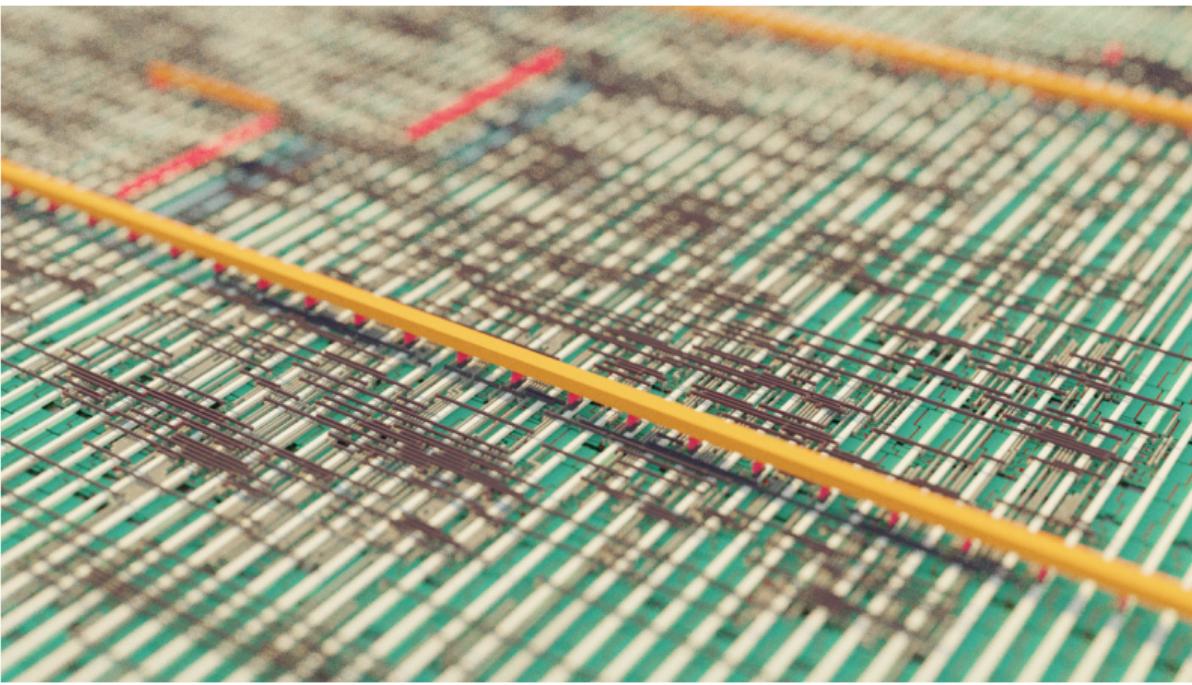


Figure 13: Maximo Borga Rendering 1⁹

⁹Picture by Maximo B. under Creative commons

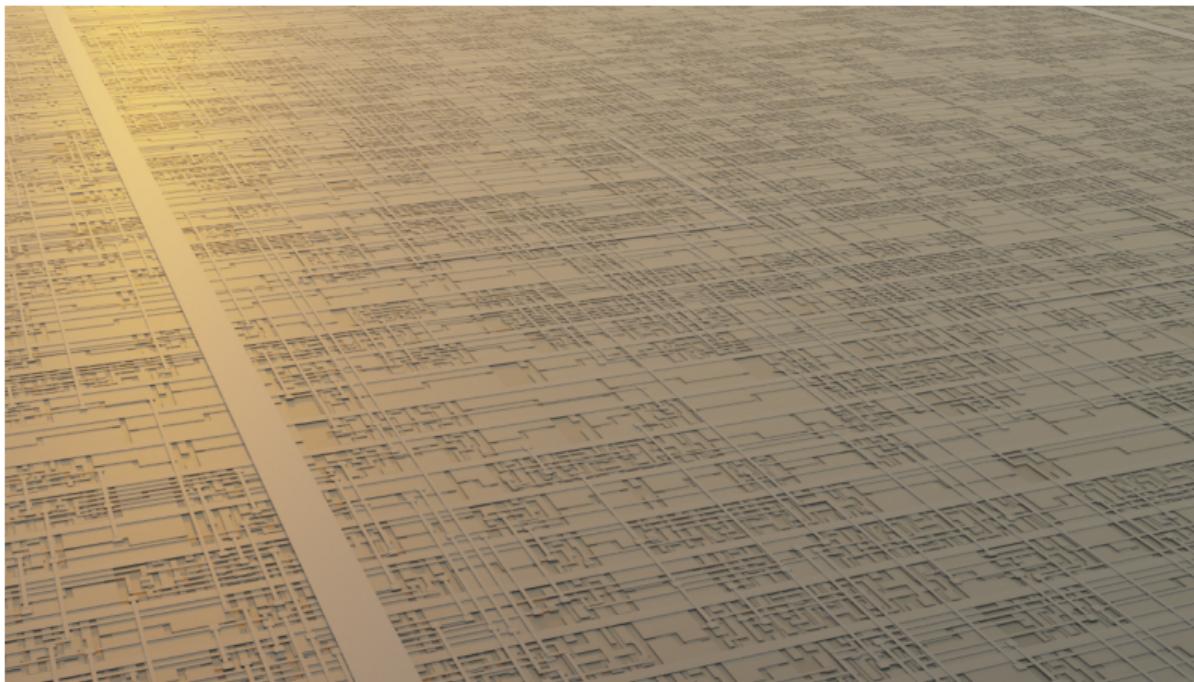


Figure 14: Maximo Borga Rendering 2¹⁰

¹⁰Picture by Maximo B. under Creative commons

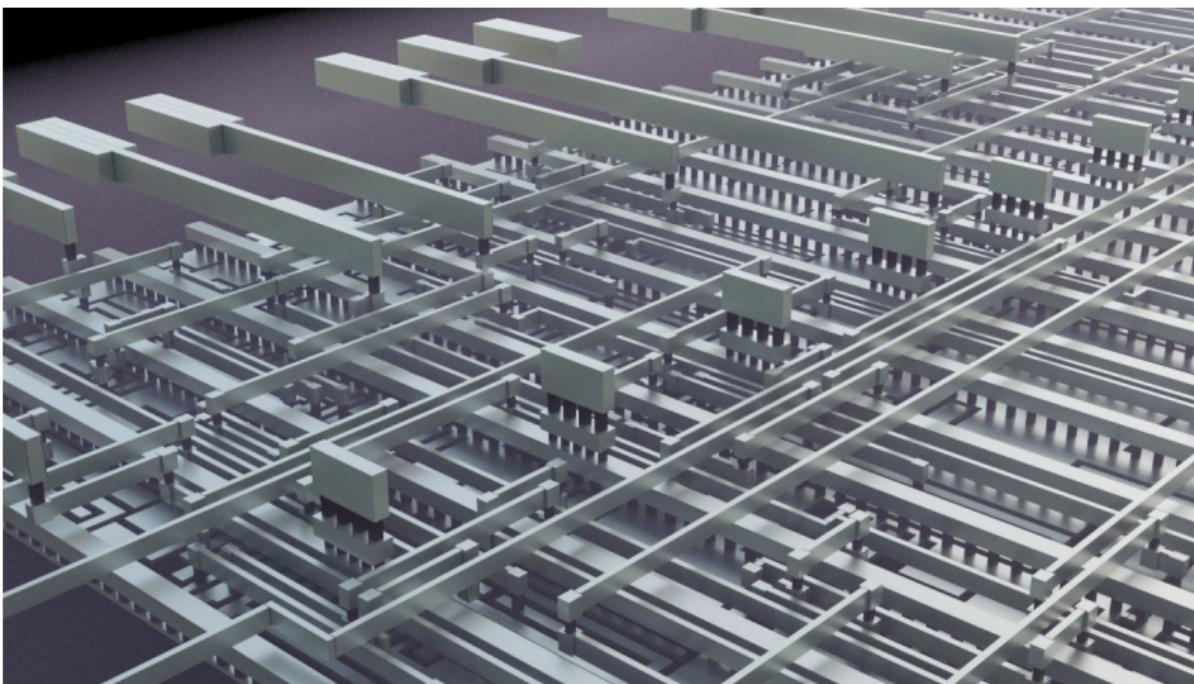


Figure 15: Maximo Borga Rendering 3¹¹

¹¹Picture by Maximo B. under Creative commons

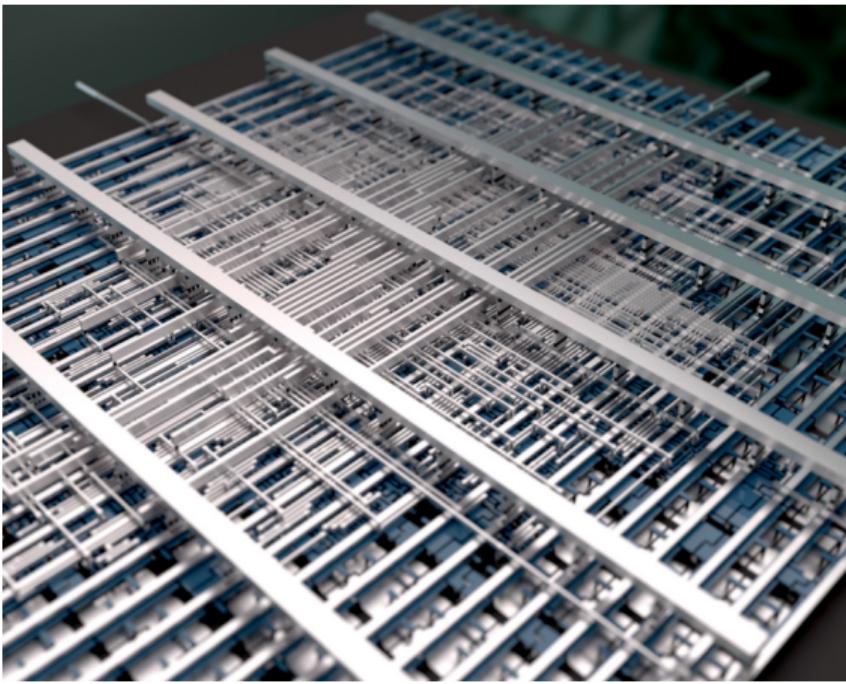


Figure 16: Maximo Borga Rendering 4¹²

¹²Picture by Maximo B. under Creative commons

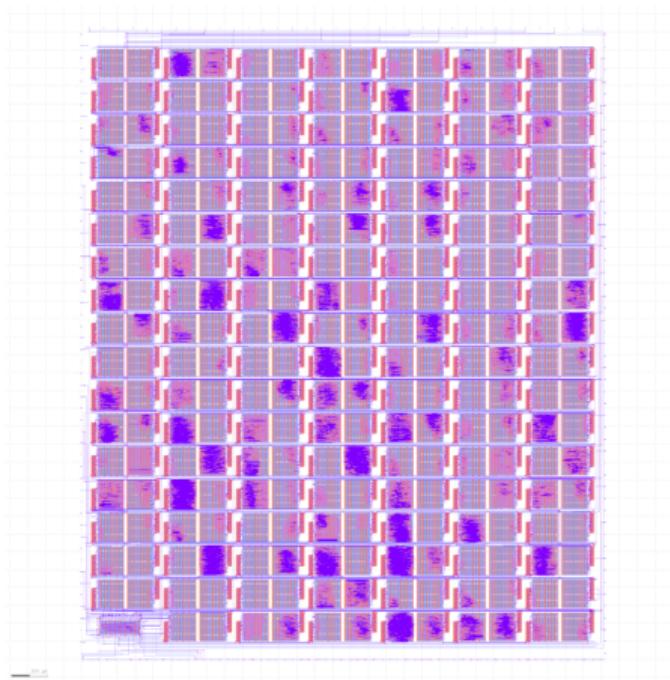


Figure 17: Tinytapeout GDS¹³

¹³Picture by T.Knoll under Creative commons

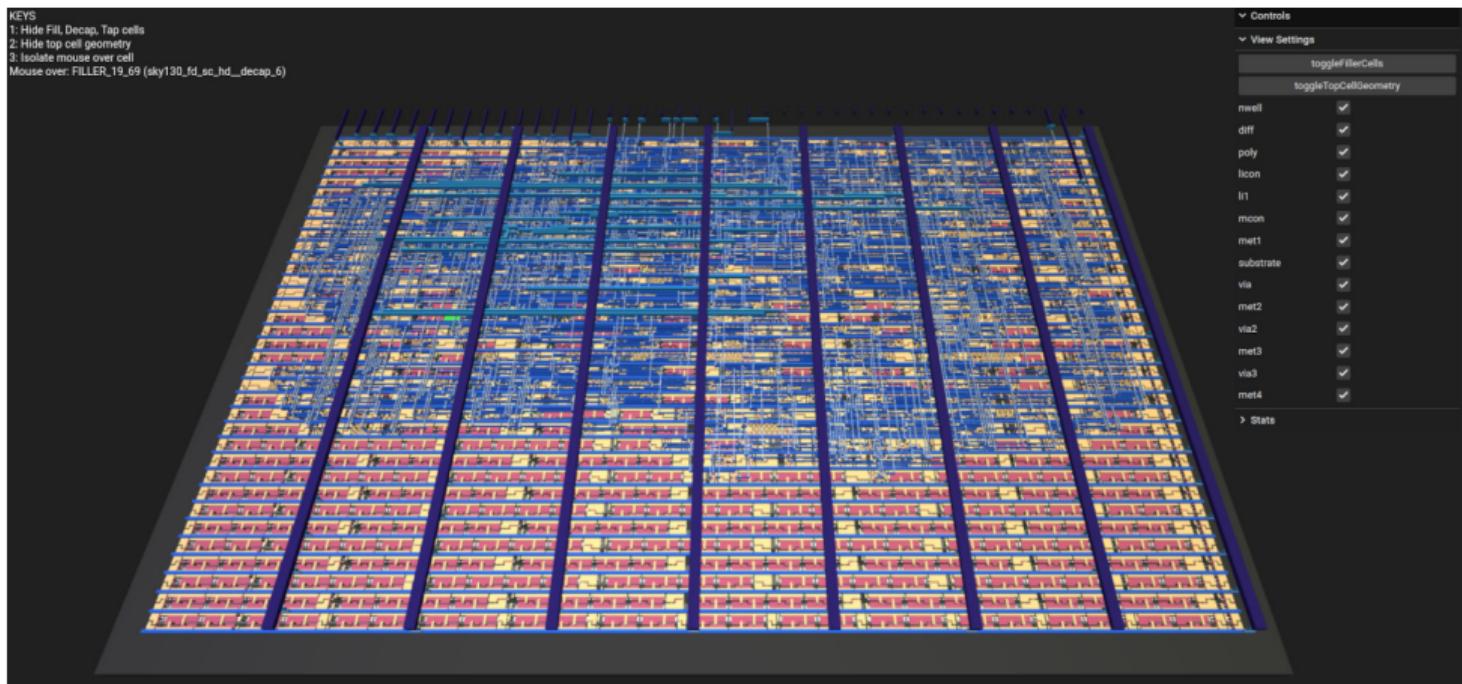


Figure 18: Webviewer Tinytapeout zoomed out¹⁴

¹⁴Picture by T.Knoll under Creative commons



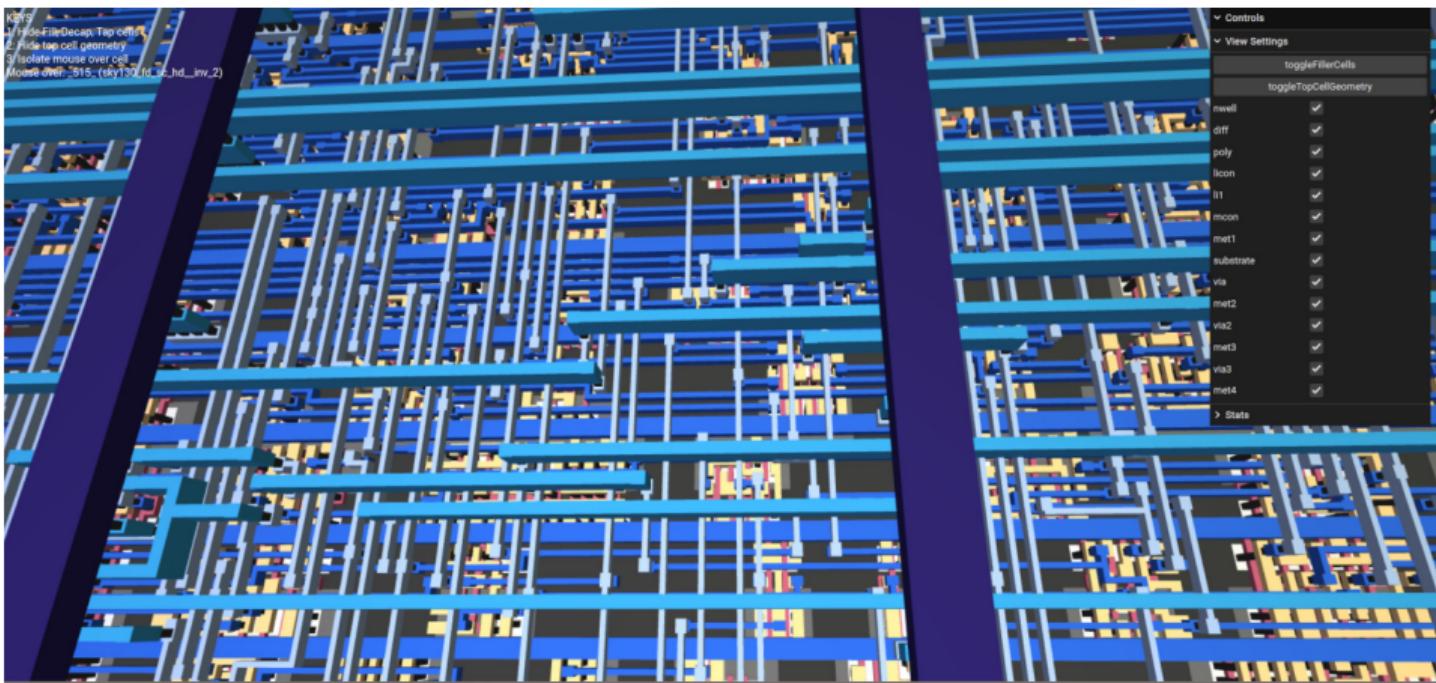


Figure 19: Webviewer Tinytapeout zoomed in¹⁵

¹⁵Picture by T.Knoll under Creative commons



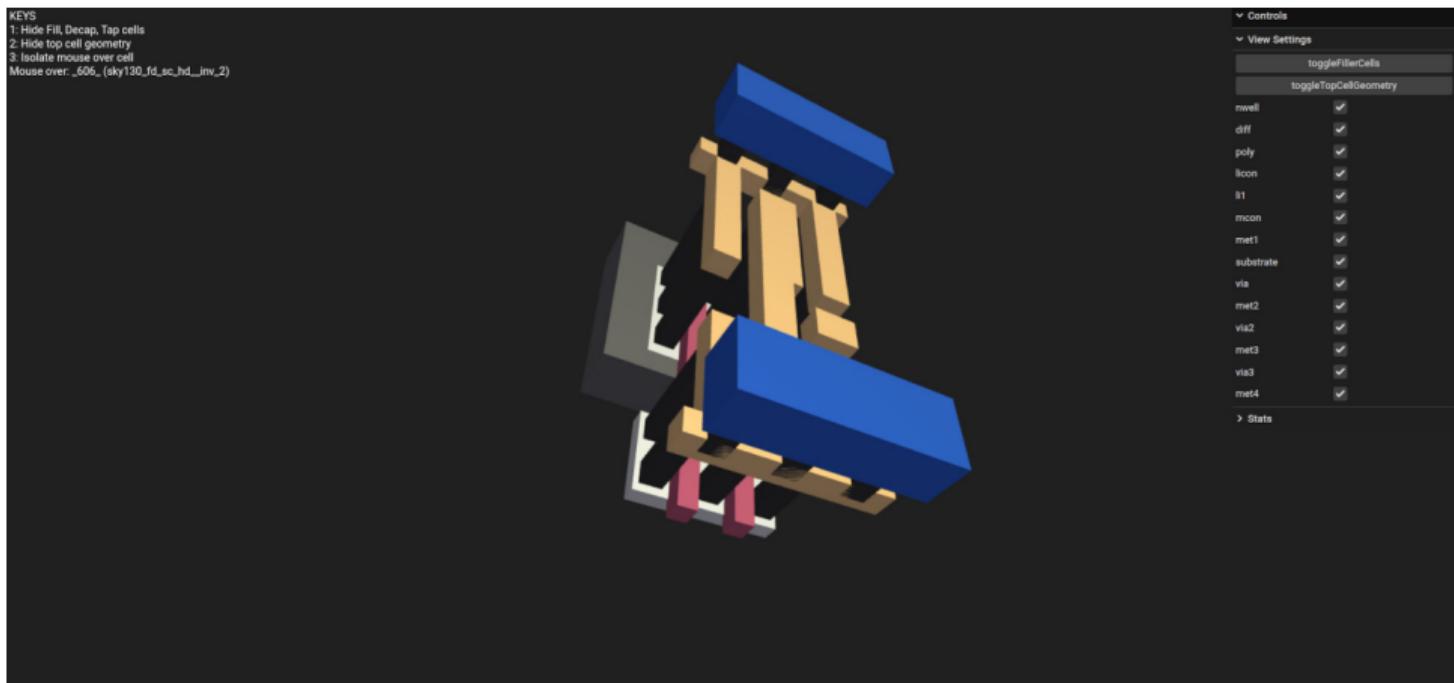


Figure 20: Webviewer Tinytapeout single cell¹⁶

¹⁶Picture by T.Knoll under Creative commons



Webviewer of Tinytapout designs with the IHP PDK

See a TinyTapeout design (GVA clock by Matt Venn) in a 3D viewer:

https://tinytapeout.com/runs/ttihp0p2/tt_um_vga_clock

It is made with the IHP open-source PDK.



Siliwiz - How do semiconductors work?

Play with Siliwiz or take the lessons:

<https://app.siliwiz.com/>



A few words about open-source in general

Wikipedia: Open-source

https://en.wikipedia.org/wiki/Open_source

Wikipedia: Open-source software

https://en.wikipedia.org/wiki/Open-source_software

Wikipedia: Open-source hardware

https://en.wikipedia.org/wiki/Open-source_hardware

Questions:

- Do you see the differences in the terminologies?
- Where should the article about the content of this course be in?





Chapter 1 -
Introduction
and overview -
QUESTIONS

Course authors
(Git file)

Chapter 1 - Introduction and overview - QUESTIONS

Course authors (Git file)



Chapter 1 -
Introduction
and overview -
QUESTIONS

Course authors
(Git file)

Recap questions for Chapter 1



Questions

Chapter 1 -
Introduction
and overview -
QUESTIONS

Course authors
(Git file)

- What are the features of open-source software?
- RTL is an abbreviation for ... ?
- Explain RTL in your own words.
- GDS (II) is an abbreviation for ... ?
- Explaining GDS (II) in your own words.
- RTL-to-GDS (RTL2GDS) stands for? Name some bulletpoints.
- What is OpenROAD?
- How old is OpenROAD?
- Is there more OS-EDA software than OpenROAD?
- What are these? Name some?
- Since when can open-source EDA software be used to build physical microchips?

Chapter 01 - Server, Login, Shell - TRAINING - Advenced

Course authors (Git file)



1

Advanced shell training

Advanced shell training

Learn some more usage of the linux terminal shell with this advanced training.



Rename the textfile

You remember your textfile from earlier?

Task: Rename the textfile

- Do this in a shell terminal (not in GNOME, means: no mouse, no file explorer!)
- Navigate to your created directory (with `cd` and `cd ..`)
- Rename your file to another filename with `mv`
- If you don't know how to use `mv`, read the man page first with `man mv` and learn the usage.



Change the content in the textfile

Task: Change the content in the textfile

- Do this in a shell terminal (not in GNOME, means: no mouse, no file explorer!)
- Navigate to your created directory (with `cd` and `cd ..`)
- Open the textfile with `nano <your filename>`
- Examine the two lines at the bottom. These are keyboard commands for the nano editor.
Find the key shortcuts for save and exit. What are they?
- Change some text, save and exit your textfile.



Chapter 01 - Server, Login, Shell - TRAINING - Bonus

Course authors (Git file)



- 1 Two ideas
- 2 Do the siliwiz lessons
- 3 Look for a tinytapeout design



Two ideas

- For this Bonus training there are two ideas of what can be done.
- These tasks can be re-visited verytime during the course when there is free time. They might take longer then the course week itself.
- Taking these bonus ideas back to home as a starting point for your own EDA designs is intended.



Do the siliwiz lessons

- Doing the siliwiz lessons helps a lot to learn more about semiconductors in general.
- In the course chapter about the open-source PDK the knowledge from Siliwiz will definitely be helpful for deeper understanding.
- If you want to go into analog circuit design, this might be a good start too.

Task: Start doing the SiliWiz lessons

- Here you go (Link to lesson is upper right corner):
- <https://app.siliwiz.com/>
- Come back to the lessons whenever there is free time in the course.



Look for a tinytapeout design

- In this course we have pre-configured and tested examples for the chip designs,
- But you could try to build an own designs. The course trainer might not be able to guide you fully. You're mostly on your own.



What to expect

This idea is for:

- If you feel like you want to do an own design, but don't want to write a Verilog project from the start.
- Look into all the Tinytapeout shuttle runs. The designs are open-source.
- Open-source: You are allowed to review, modify, use the designs.
- So you can use them for creating your own example for this course.



Where and how to start

Task: Find a design from TinyTapeout

- <https://tinytapeout.com/runs/>
- Browse the designs from the TinyTapeout shuttle runs.
- Look a design that looks fitting for you
- Only take designs with good documentation!!!
- Find the Github repository of the design.
- Review the documentation and the Verilog code.



Next steps (roughly)

After chapter 3 + 4 (Verilog and First run):

- Take a pre-configured example design (gcd) as template
- Copy the template as a new design (see ORFS tutorial)
- Add the Verilog from the TT design

After chapter 5 + 6 +7 (PDK and OpenROAD GUI):

- Try to modify the rest of the new design:
 - config files
 - constraints
 - Makefile
- Give it a try: run the design



Chapter 01 - Server, Login, Shell - TRAINING - Common

Course authors (Git file)



- 1 Server and user credentials
- 2 GNOME desktop: First lookaround
- 3 The course data
- 4 Linux shell
- 5 IMPORTANT Tasks
- 6 Flow directory
- 7 Online shell resources



Server and user credentials

The PC environment for the course will be provided by IHP.

- Your PC in front of you must be connected to a server
- You will then work in a Ubuntu 24 Linux system
- The desktop GUI will be Gnome



Connect to the IHP server

Follow these steps:

- ① Open ThinLinc on the host PC (Windows?)
- ② Go to the Options and set “Fullscreen on this monitor”
- ③ Connect with the login data, given to you by IHP
- ④ Ubuntu with Gnome desktop should start in ThinLinc

For Log out:

- ① Just “Log out”. Don’t Power Off.
- ② This will take you back to the host system.

Notice:

- Please ask if there is questions!
- We’ll try to do this for all participants first, before proceeding to the next steps of the training.

Search, start and close programs

Create and save a textfile:

- ① Search for a texteditor (gedit)
- ② Open gedit
- ③ Write a litte (your name or anything else)
- ④ Save the textfile with a name and the suffix .txt
- ⑤ To a new directory: Documents/myfiles
- ⑥ Close the texteditor (gedit)



Tab-switching between programs

Tab-switch between opened programs:

- ① Search and open at least three different programs (office, gedit, document viewer?)
- ② Tab between the programs (with ALT+TAB on the keyboard)
- ③ It is a circle. After three Tabs you should be at the first again.
- ④ Close all programs



Download and unpack

Get the course data

- Get the latest release download package:

<https://github.com/OS-EDA/Course/releases>

- Create a directory for the course slides.
- In Linux the ~/Documents is a good place to create the directory. Maybe create the directory Documents/course
- Unpack the course into this directory.



Look around in the course data

The screenshot shows a GitHub repository page for the "open-source EDA course". The repository has 161 commits and 1 tag. The main branch is "main". The repository is public and has an Apache-2.0 license. It has 27 stars and 9 watchers. There are 7 forks. The repository is reportable. It has 1 release and no packages published. The repository uses Shell and has 100% coverage.

Code

About
Home of the open-source EDA course.

Readme

Apache-2.0 license

Activity

Custom properties

27 stars

9 watching

7 forks

Report repository

Releases

1 tags

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

Languages

Shell 100.0%

File/Folder	Description	Last Commit
ThorKn README slide update	switch to pandoc:extra	3 months ago
.github/workflows		3 months ago
Chapter_00_Preparations	c01 components icons	3 months ago
Chapter_01_Introduction	q8 links	3 days ago
Chapter_02_OpenROAD_tools	q8 links	3 days ago
Chapter_03_Verilog	q8 links	3 days ago
Chapter_04_OpenROAD_first_run	q8 links	3 days ago
Chapter_05_PDK	q8 links	3 days ago
Chapter_06_OpenROAD_gui	q6 training+	12 hours ago
Chapter_07_OpenROAD_flow_scripts	q6 lecture ready	12 hours ago
Chapter_08_Tapeout	q8 ready	18 hours ago
build	q6 training+	12 hours ago
icons	pandoc pic resize fixed	3 months ago
pandoc/templates	structure and content update	3 days ago
pics	structure and content update	3 days ago
LICENSE	Initial commit	last year
README.md	README slide update	1 hour ago
authors.md	structure and content update	3 days ago
generate_slides.sh	q6 lecture ready	12 hours ago

Figure 1: Course directory structure

The chapters and slides

You will find the slides in the build directory.

- If sorted alphabetical, it makes most sense.
- C0X is the chapter number.
- In each chapter:
 - Start with the lecture
 - Cheatsheet (if available) is a single slide
 - Trainings in the order common, advanced and bonus
 - Questions are for the next day (mornings)

Task:

- Examine the build directory.
- Open and close some of the pdf files.
- Become comfortable with the course structure.

Links from the slides

This might be outdated for the course, please try yourself.

- An issue with the linux document viewer and snap browsers.
- Links from slides don't open in the browser.

Possible workaround:

- Right click the link and copy
- Open browser
- Paste link to URL line



Workspace arrangement

Suggestions or Options:

- Arrange windows next to each other
- ALT+TAB between programs (tasks)
- Close unwanted windows after the completion of a chapter



Short command list of a linux shell

- ls (list content of directory)
- ls -al (list with option for more information)
- cd directoryname (change to directory)
- cd .. (change to upper directory)
- mkdir (make directory)
- touch (make file)
- mv (move)
- cp (copy)
- nano (opens the nano file editor)



Man pages

If unsure how to use a command, read the man-page:

```
1 | man <command>
```

Task:

- Open the man pages of all the commands in the above list (last slide)
- Find the syntax for the commands (it is given on the top of the man page)
- What are OPTION, SOURCE and DEST ?
- Find the definition of the option -al for the command ls



IMPORTANT Tasks

To get ready for working with the open-source EDA tools, you must run (execute) a shell script in your shell terminal.

This is absolutely necessary for the rest of the course

There are two options to do this:

- ① Every time you open a shell and want to work with the tools
- ② Once in the bashrc



Option 1: Source every time in the shell

Task: Setup the linux tool environment

- Open a shell terminal
- Run source /eda/or2/env.sh
- Do this every time you open a new shell.
- It doesn't hurt if you do it multiple times. Nothing breaks.
- It will lead to strange behaviour and no results if it is not done.

You should get a message like copied flow directory or flow directory already there



Option 2: Source in the bashrc

Task: Modify .bashrc with source command

- Find the file .bashrc:
 - Location: go to the users home directory cd ~
 - Hidden files have a point as pre-fix in their name. List the files with ls -al
- Open the file .bashrc in a texteditor. nano .bashrc
- At the bottom line, add the command source /eda/or2/env.sh
- Save the file and close the editor
- In a shell terminal: reload bashrc with the command . ~/.bashrc

You should get a message like copied flow directory or flow directory already there



Check the environment

- Check if the setup was successfull.
- **Don't continue if the following checks are errornous, but ask for help from the trainer**

Task: Check the environment

- Open a shell terminal
- Run source /eda/or2/env.sh
- See if the tools are sourced (available) with getting their versions:
 - openroad –version should give you the version number
 - yosys --version should give you the version number
 - klayout -v should give you the version number



Congratulations!

Congratulations:

You have successfully run some of the open-source EDA tools on a linux server.

- You did run openroad, yosys and klayout.
- They did not do much, but giving you their version numbers.
- But it is the needed base for the rest of the course.

You are on a good way



Linux Power settings

By default the screen goes black after 5 minutes of idle. Additionally the screen locks and you have to enter your password again.

This is good and intended. But might be a little too annoying for the course.

Task: Disable screen black out

- Find the linux “power settings”
- Disable the “screen black after 5 mins” or higher the value.



Flow directory

The last step (the important one) created a directory in your linux home directory:

- The flow directory

This is the most important directory for the course!



Examine the flow directory

Task: Examine the flow directory

- Open a shell terminal
- Run source /eda/or2/env.sh
- Navigate to the flow directory as you learned before
- List the content with ls -al
- Find:
 - The Makefile
 - The /flow/designs/src directory. What is inside?
 - The flow/designs/ihp-sg13g2 directory. What is inside?



The Makefile

Task: Open the Makefile

- Open the Makefile in a texteditor (with gedit)
- What is inside the Makefile?
- What could be the meaning of
 - DESIGN_CONFIG=....
 - #DESIGN_CONFIG=====
- Close texteditor without changes in the Makefile



Online shell tutorial

To learn some Linux shell, you should find a tutorial that matches good to you.

I found this one simple and good to follow for me while learning Linux shell:

<https://community.linuxmint.com/tutorial/view/100>



Makefile

OpenROAD flow scripts use Makefiles. So you might want to learn some basics about Makefiles. Again here is the tutorial that has helped me most:

<https://makefiletutorial.com/>



Tips and tricks

- shell: TAB for autocomplete
- shell: 2xTAB for all choices of autocomplete
- Open a new shell terminal with the mouse:
 - Right click on an empty space in the directory window
 - Choose “open in Terminal”



Chapter 2 - OpenROAD workflow

Course authors (Git file)



1 History of OpenROAD

2 OpenROAD Flow Scripts

3 Fill insertion

4 Resources



Section 1

History of OpenROAD



Foundations and Realization of Open, Accessible Design (OpenROAD)

At the top of the documentation:

<https://openroad-flow-scripts.readthedocs.io>

The screenshot shows the homepage of the OpenROAD Flow Scripts documentation. The left sidebar contains a navigation menu with links to "User Guide", "Build Using Pre-built Binaries", "Build Using Docker", "Build Locally", "Build Using WSL", "Environment Variables for the OpenROAD Flow Scripts", "Metrics", and "AutoTuner". Above this menu are social sharing icons for GitHub, Twitter, LinkedIn, and Stars (375). The main content area features a large title "Welcome to the OpenROAD Flow Scripts documentation!" followed by a detailed description of the project's mission and scope.

The OpenROAD ("Foundations and Realization of Open, Accessible Design") project was launched in June 2018 within the DARPA IDEA program. OpenROAD aims to bring down the barriers of cost, expertise and unpredictability that currently block designers access to hardware implementation in advanced technologies. The project team (Qualcomm, Arm and multiple universities and partners, led by UC San Diego) is developing a fully autonomous, open-source tool chain for digital SoC layout generation, focusing on the RTL-to-GDSII phase of system-on-chip design. Thus, OpenROAD holistically attacks the multiple facets of today's design cost crisis: engineering resources, design tool licenses, project schedule, and risk.

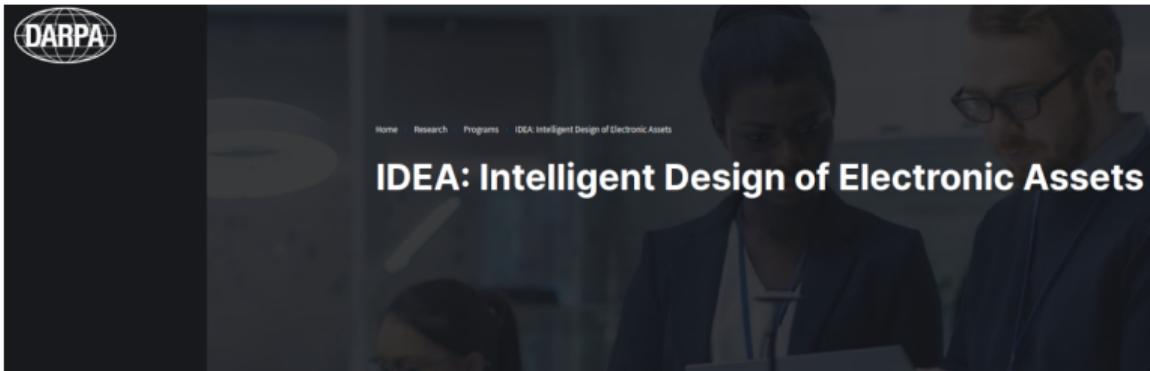
The IDEA program targets no-human-in-loop (NHIL) design, with 24-hour turnaround time and zero loss of power-performance-area (PPA) design quality.

Figure 1: Darpa IDEA¹

¹Source: Screenshot of the webpage.

Darpa IDEA

<https://www.darpa.mil/research/programs/intelligent-design-of-electronic-assets>



Program Summary

Next-generation intelligent systems supporting Department of Defense (DoD) applications like artificial intelligence, autonomous vehicles, shared spectrum communication, electronic warfare, and radar require processing efficiency that is orders of magnitude beyond what is available through current commercial electronics. Reaching the performance levels required by these DoD

Figure 2: Darpa IDEA ²

²Source: Screenshot of the webpage.

Darpa ERI

2018/2019 Darpa ERI, cadence and the people

<https://community.cadence.com/tags/openroad>

Start reading from the bottom!

ERI: OpenROAD

 Paul McLellan

If I had to summarize DARPA's Electronic Resurgence Initiative in one phrase, it would be "getting the cost of design down." As I've said several times this week, the US Department of Defense (DoD) does not have high volumes and so the cost of a part...

over 6 years ago

Cadence Blogs

Breakfast Bytes

The DARPA Electronic Resurgence Initiative (ERI)

 Paul McLellan

Many weeks ago DARPA organized a summit at the Palace of Fine Arts in San Francisco. The first day consisted of a workshop and some other presentations, including one by Cadence's Tom Beckley. Since Tom's presentation was very similar to what he presented...

over 6 years ago

Cadence Blogs

Breakfast Bytes



Figure 3: Darpa ERI ³

OpenROAD V1.0

Document with OpenROAD V1.0 Expectations:

<https://vlsicad.ucsd.edu/NEWS19/OpenROAD%20RTL-to-GDS%20v1.0%20Expectations.pdf>

OpenROAD v1.0 Expectations

Andrew B. Kahng and Tom Spyrou
The OpenROAD Project

November 22, 2019

Web: <https://theopenroadproject.org/>
GitHub: <https://github.com/The-OpenROAD-Project>

The OpenROAD v1.0 tool, to be released in July 2020, will be capable of push-button, DRC-clean RTL-to-GDS layout generation in a commercial FinFET process node. The tool is currently visible [here](#). In its v1.0 form, it will be integrated on an incremental substrate provided by [the OpenDB database](#) and [the OpenSTA static timing engine](#). It will also offer users and

Figure 4: OpenROAD v1.0⁴

V1.0 Roadmap

How to deal with these expectations:

<https://eri-summit.darpa.mil/docs/ERIsummit2019/posh/08IDEA%20UCSD%20Website.pdf>

Looking Forward (Year 2 = Phase 1B)

- V1.0 July 2020 must advance 20 years on EDA industry learning curve within next 2-3 quarters
 - 1980's file-based integration **July 2019**
 - 2000's tight integration on shared incremental substrate **July 2020**
- Next: architecture, database, build/CI/devops, CAE/PE,
 - + teaching EDA SW to the OpenROAD/FOSS community
- Professionals on the team: mandatory
 - Industry veterans who have "done this before"

Figure 5: Looking forward⁵

⁵Source: Screenshot of the webpage.

Courses for OpenROAD

A single excellent project (some of us might know):

<https://theopenroadproject.org/Courses/>

The screenshot shows the OpenROAD website with a dark blue header bar. The header contains the OpenROAD logo on the left and navigation links for HOME, ABOUT US, NEWS, RESOURCES, BLOGS, COMMUNITY, USER STORIES, and CONTACT US on the right. Below the header, the word "Courses" is centered above a table. The table has two columns: "Title" and "File/Link". There is one row in the table with the title "ZeroToASIC by Matt Venn" and the link https://zerotoasiccourse.com/matt_venn/.

Title	File/Link
ZeroToASIC by Matt Venn	https://zerotoasiccourse.com/matt_venn/

Figure 6: Courses⁶

⁶Source: Screenshot of the webpage.

Help is on the way

Kudos to this course:

<https://theopenroadproject.org/news/6455/>



The screenshot shows the OpenROAD website's header. The logo 'OpenROAD' is on the left, followed by a horizontal navigation bar with links: HOME, ABOUT US ▾, NEWS, RESOURCES, BLOGS, COMMUNITY, USER STORIES, and CONTACT US.

Kudos to Christian Wittke (IHP) and Thorsten Knoll (HSRM) on their development of a Digital EDA Course using IHP-SG13G2 and OpenROAD!

See the video of Christian's ORConf 2024 talk at https://www.youtube.com/watch?v=Ozd_yXoExLo !

Figure 7: This course⁷

⁷Source: Screenshot of the webpage.



Section 2

OpenROAD Flow Scripts



Flow steps

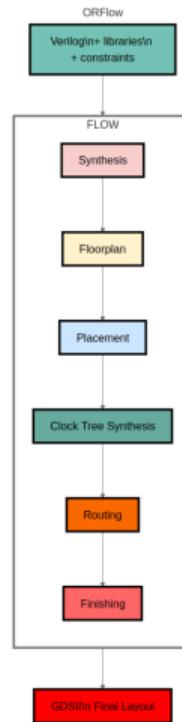


Figure 8: OpenROAD flow steps⁸

Flow components

RTL-GDSII Using OpenROAD-flow-scripts

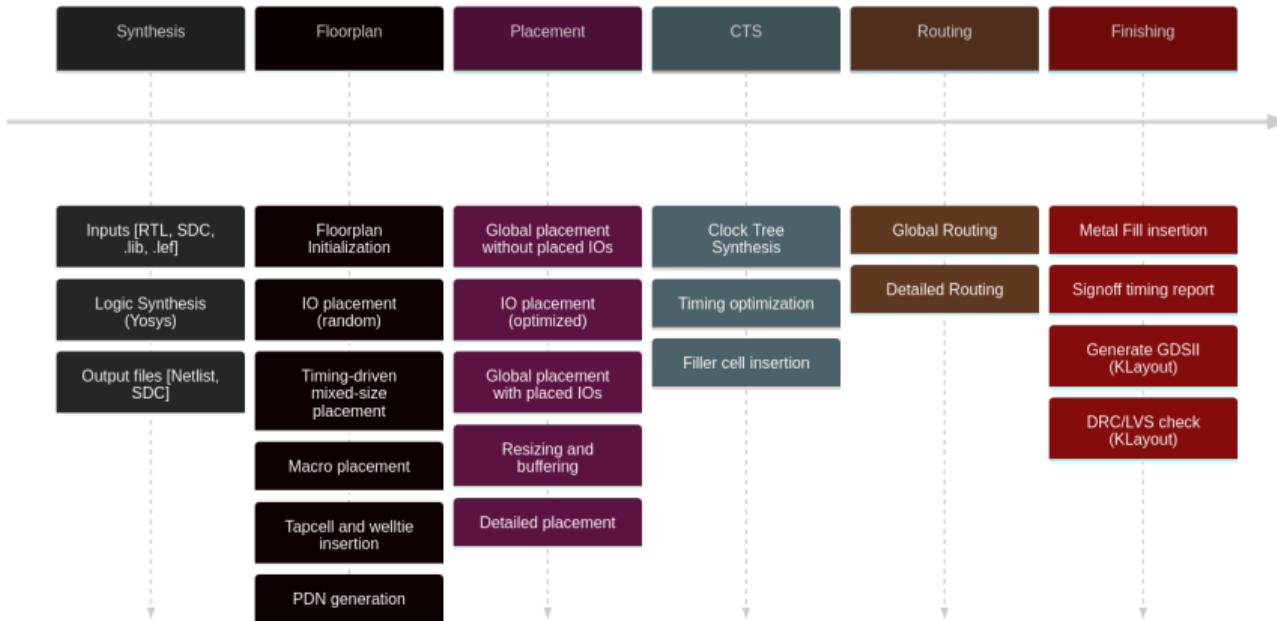


Figure 9: OpenROAD flow components⁹

Section 3

Fill insertion



Fill insertion

There are two flow components that take care of filling area gaps in the design:

- Filler cell insertion
- Metal fill insertion

Both will be explained in the following section.



Filler cells: Reasons

Why filling the design area with more cells?

- After floorplaning and placement there will always be gaps between the standard cells
- These gaps must be filled to keep the continuity of
 - n-well and p-well spacing
 - railing
 - implants
- The filling with cells helps against the WPE (well proximity effect)



Filler cells: Properties

Filler cells

- have no logic function
- have no inputs and no outputs
- are not in the netlist of the design
- are available in different widths



Filler cells: GDS

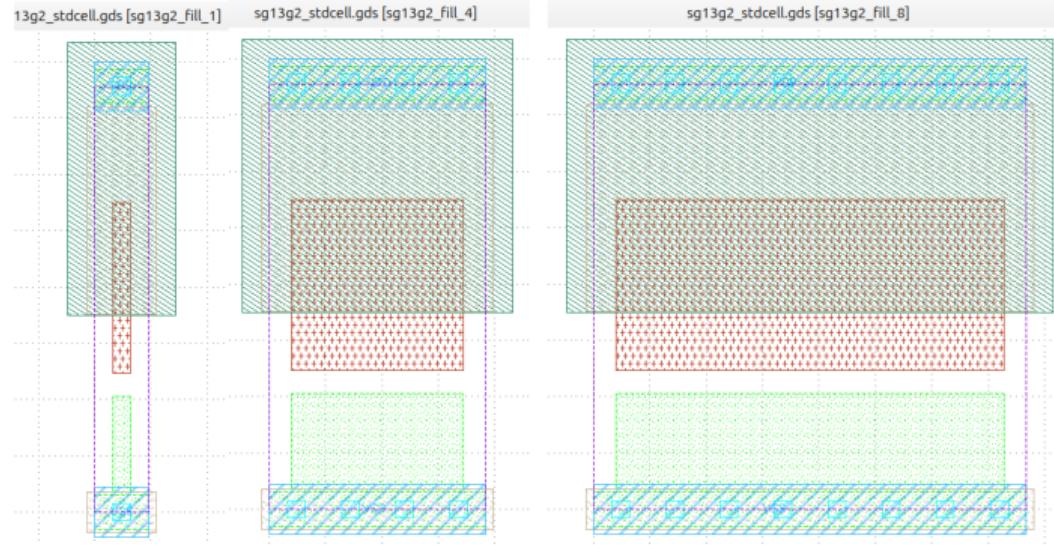


Figure 10: Filler cells in different widths (1, 4, 8)



Metal fill

What is metal filling?

- Metal filling adds polygons and shapes to the metal layer(s).
- These fillings have no logic functionality.
- The purpose is creating a more even or uniform distribution of the metal area.
- Physical reasons are planarity and reducing thickness variations (CMP).
- Metal filling can affect timing and signal integrity.
- Metal filling is done algorithmical (mostly with scripts).

More to read:

https://semiengineering.com/knowledge_centers/materials/fill/



Section 4

Resources



Help with the terminology

Searching the terms with a standard search engine might not bring useful results every time.
Matt Venn created a page for EDA terminology:

<https://www.zerotoasiccourse.com/terminology/>

I've collected all the ASIC jargon here and broken it down into easy to understand descriptions.

Antenna Report	ASIC	CMOS
Corner	Die	Doping
DRC	Floorplan	Foundry
FPGA	GDSII	Harden

The list of open-source tools

The list of flow steps and flow components in ORFS contains the information about the original open-source tools.

We will dive a little deeper into this list with the course training:

<https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#running-the-automated-rtl-to-gds-flow>



Additional links

Awesome open-source ASIC resources:

<https://github.com/mattvenn/awesome-opensource-asic-resources>

Zero-to-ASIC resources list:

<https://www.zerotoasiccourse.com/resources/>

AIC 2025 lectures by Carsten Wullf:

<https://github.com/wulffern/aic2025>

<https://analogicus.com/aic2025/>



Meet the developers

There is a slack community with most of the open-source silicon projects and people in it:

<https://open-source-silicon.dev>

This is the perfect place, if you

- search for people you heard of in open-source EDA.
- want to reach out to tool developers.
- have all sorts of questions in open-source EDA.
- look for a specific problem and want to see if others already are in a discussion about it.
- want to share your experience and help others.



Most of the tools in open-source EDA have their own channel in this slack. Just try search and add for the channel names.

The screenshot shows the 'Get started' page of the open-source-silicon.dev Slack workspace. The left sidebar lists various sections: Home, DMs, Activity, More, and a list of channels. The channels section includes External Connections (#chipignite, #openlane, #sky130), Channels (#announcements, #caravel, #general, #ihp-sg13g2, #openlane-2), Direct messages, Moss Mostly you, and Invite people. The main content area displays a list of channels to join:

- #analog-design 10+ RECENT POSTS
Channel for discussion about designing mixed-signal integrated circuits, including noise-shaping techniques, SAR ADCs and sigma-delta... 2492
efabless staff can view this channel • Last post 2 days ago • 1,821 files
- #xschem 10+ RECENT POSTS
xschem (VLSI hierarchical schematic editor) specific Q&A, feature requests/bug reports, etc. 776
Last post 1 day ago • 1,885 files
- #magic 10+ RECENT POSTS
Last post 1 day ago • 1,562 files 876
- #openlane-development 10+ RECENT POSTS
This channel is dedicated to tracking the openlane commits on all branches. 135
Last post 4 minutes ago • 3 files
- #klayout
The KLayout layout viewer and editor 480
Last post 1 day ago • 220 files
- #openroad 10+ RECENT POSTS
Discussion channel for using the OpenROAD open-source digital place and route flow (<https://theopenroadproject.org/> and <https://gitlab.com/openroad-project/>) 563
Last post 19 hours ago • 177 files

Figure 12: Channels in the open-source-silicon.dev slack



Chapter 2 -
OpenROAD
workflow -
QUESTIONS

Course authors
(Git file)

Chapter 2 - OpenROAD workflow - QUESTIONS

Course authors (Git file)



Chapter 2 -
OpenROAD
workflow -
QUESTIONS

Course authors
(Git file)

Recap questions for Chapter 2



Questions

Chapter 2 -
OpenROAD
workflow -
QUESTIONS

Course authors
(Git file)

- Name some of the internet resources linked in chapter 2.
- Where are the origins of OpenROAD?
- Is OpenROAD a fully from-the-scratch written toolchain?
- Can you name some of the included (used) software projects?
- How many contributors has
 - OpenROAD (OR)?
 - OpenROAD flow scripts (ORFS)?
- How many close (open) issues (pull requests) has ORFS?
- Name the flow steps in ORFS (round robin in the room)
- Name the flow components in ORFS (round robin in the room)
 - (Bonus) Can you describe what the component does?

Chapter 02 - Terminology - TRAINING - Advanced

Course authors (Git file)



1

Preparation for chapter 03 + 04 Bonus Training



Another option for building your own design can be started.

Task: Learn about LFSR

- Read the Wikipedia about Linear Feedback Shift Registers.
- https://en.wikipedia.org/wiki/Linear-feedback_shift_register

Task: Design your own LSFR

- Try drawing a small LSFR
- 4 to 8 bits length maximum
- 1 or 2 feedback lines with XOR

Task: Simulate the LSFR with a Bit table

- Simulate your LSFR with a Bit table
- How should the table look?
- Clock, Register content, Feedback Bits, Input, Output

Chapter 02 - Terminology - TRAINING - Common

Course authors (Git file)



1

Searching

Browse the resources

Task: Review the flow terminology

- Try to formulate the meaning and purpose of all the following search results in your own words.
- Search for each of the flow steps
 - Synthesis
 - Floorplan
 - Placement
 - Clock Tree Synthesis
 - Routing
 - Finishing
- Search for as much flow components as possible (see the components names in the diagram in the lecture slides)
- If no result is found, try a search engine (or chatGPT).

Matt Venns list:

<https://www.zerotoasiccourse.com/terminology/>

Getting the words in place

In the beginning it can be hard to learn all the new words and give them meaningful connections immediately.

Let's try to create a timeline (or taskline) for "How to make microchips?" in a joint effort.

Task: Create a diagram "How to make a microchip?"

- Everyone can give a keyword of the topic "How to make a microchip?"
- We try to figure out the meaning and the position in a timeline / taskline?
- Flipchart, Whiteboard, Texteditor?



List the open-source tools in OpenROAD

OpenROAD uses a lot of open-source tools, that were available even before OpenROAD.

Task: Create a list of tools

- Use this link list as the knowledge base:
 - <https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#running-the-automated-rtl-to-gds-flow>
- Find the names of the open-source tools that are used in ORFS.
- Create a list with the columns:
 - Name of the open-source tool
 - Flow step and flow component in ORFS where the tool is used.
 - Description of the function of the tool
 - Bonus: What was modified or contributed to this tool by the OpenROAD team?

Example row

Example row in the list:

- Yosys
- Synthesis
- Verilog and constraints to netlists
- Most of the initial development was coded by Claire Wolf (not by ORFS)



Chapter 3 - Verilog crash course

Course authors (Git file)



- 1 Introduction
- 2 Verilog elements
- 3 Simple circuits: Combinational
- 4 Simple circuits: Sequential
- 5 Selected feature: Parameterized counter
- 6 Selected feature: Preprocessor
- 7 Selected feature: Yosys and Systemverilog



Section 1

Introduction



Introduction

Verilog was initially developed as a simulation language in 1983/1984, bought up by Cadence and freely released in 1990.

The first standardization took place in 1995 by the IEEE (Verilog 95). A newer version is IEEE Standard 1364-2001 (Verilog 2001).

- Syntax comparable to C (VHDL was started on ADA / Pascal) with compact code
- Spread in North America and Japan (less in Europe)
- Can also be used as the language for netlists
- Support from open source tools
- The majority of the ASICs are developed in Verilog.
- Less expressive than VHDL (curse and blessing)



The proximity to C and Java may lead to confusion. In Verilog, too, lines that describe a combinatorial circuit can also be replaced.

**** Verilog is a hardware description language (HDL)****

This crash course is limited to a subset of synthesizable language constructs in Verilog.

The aim of this selection is not commercial tools, but open-source development tools such as OpenRoad¹ or Toolchains for FPGAs, i.e. we also use some language constructs from Systemverilog, which are supported by the Yosys synthesis tool.

¹<https://theopenroadproject.org/>



Literature

- Donald E. Thomas, Philip R. Moorby, *The Verilog Hardware Description Language*, Kluwer Academic Publishers, 2002, ISBN 978-1475775891
- Blaine Readler, *Verilog by example*, Full Arc Press, 2011, ISBN 978-0983497301



Contributions, mentions and license

- This course is a translated, modified and ‘markdownized’ version of a Verilog crash course from Steffen Reith, original in german language.

<https://github.com/SteffenReith>

- The initial rework (translate, modify and markdownize) was done by:

<https://github.com/ThorKn>

- The build of the PDF slides is done with pandoc:

<https://pandoc.org/>

- Pandoc is wrapped within this project:

<https://github.com/alexeygumirov/pandoc-beamer-how-to>

- License:

GPLv3



Synthesis tool: Yosys

One should also deal with the peculiarities of the synthesis tool. The well-known open source synthesis tool Yosys writes about this

Yosys is a framework for Verilog RTL synthesis. It currently has extensive Verilog-2005 support and provides a basic set of synthesis algorithms for various application domains. Selected features and typical applications:

- Process almost any synthesizable Verilog-2005 design
- Converting Verilog to BLIF / EDIF / BTOR / SMT-LIB / simple RTL Verilog / etc.
- ...



Section 2

Verilog elements



Structure of a verilog module

```
1  module module_name (port_list);
2  // Definition of the interface
3  Port declaration
4  Parameter declaration
5
6  // Description of the circuit
7  Variables declaration
8  Assignments
9  Module instantiations
10
11 always-blocks
12
13 endmodule
```

Port list and port declaration can be brought together in modern verilog.

// introduces a comment.



Example: A linear feedback shiftregister

```

1 module LFSR (
2   input wire load,
3   input wire loadIt,
4   input wire enable,
5   output wire newBit,
6   input wire clk,
7   input wire reset);
8
9   wire      [17:0]  fsRegN;
10  reg       [17:0]  fsReg;
11  wire      taps_0, taps_1;
12  reg       genBit;
13
14
15  assign taps_0 = fsReg[0];
16  assign taps_1 = fsReg[11];
17
18  always @(*) begin
19    genBit = (taps_0 ^ taps_1);
20    if(loadIt) begin
21      genBit = load;
22    end
23  end

```

```

24   assign newBit = fsReg[0];
25   assign fsRegN = {genBit,fsReg[17 : 1]};
26
27   always @ (posedge clk) begin
28     if(reset) begin
29       fsReg <= 18'h0;
30     end else begin
31       if(enable) begin
32         fsReg <= fsRegN;
33       end
34     end
35   end
36
37 endmodule

```

input and **output** define the directions of the ports.



This part of the code

```
18 | always @(*) begin
19 |   genBit = (taps_0 ^ taps_1);
20 |   if(loadIt) begin
21 |     genBit = load;
22 |   end
23 | end
```

becomes this combinational circuit:

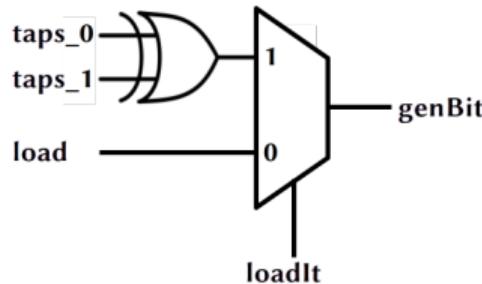


Figure 1: combinational part



And this part of the code

```
18 always @(posedge clk) begin
19   if(reset) begin
20     fsReg <= 18'h0;
21   end else begin
22     if(enable) begin
23       fsReg <= fsRegN;
24     end
25   end
26 end
```

becomes a sequential with memory (flip flops) in it. The flip flops will look something like this:

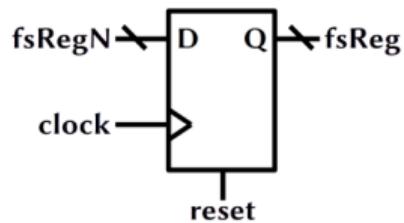


Figure 2: sequential part



Constants and Operators

There are four values available for constants and signals:

- 0 / 1
- X or x (unknown)
- Z or Z (high impedance)

One can specify the width of constants:

- Hexadecimal constant with 32 bit: 32'hDEADBEEF
- Binary constant with 4 bit: 4'b1011
- For better readability you can also use underscores: 12'B1010_1111_0001

To specify the number base use

- b (binary)
- h (hexadecimal),
- o (octal)
- d (decimal)

The default is decimal (d) and the bit width is optional, i.e. 4711 is a valid (decimal) constant.



There is an array notation:

- wire [7:0] serDat;
- reg [0:32] shiftReg;
- Arrays can be sliced to Bits:
 - serDat[3 : 0] (low-nibble)
 - serDat[7] (MSB).
- {serDat[7:6], serDat[1:0]} notes the concatenation.
- Bits can be replicated and converted into an array, i.e $\{8\{\text{serData}[7 : 4]\}\}$ contains eight copies of the high-nibble from serDat and has a width of 32.

Arithmetic operations, relations, equivalences and negation:

- $a + b$, $a - b$, $a * b$, a / b und $a \% b$
- $a > b$, $a \leq b$, und $a \geq b$
- $a == b$ und $a != b$,
- $!(a = b)$



** Attention: ** If x or z do occur, the simulator determines false in a comparison. If you want to avoid this, the operators === and !== exist. So the following applies:

```
1 if (4'b110z === 4'b110z)  
2 // not taken  
3 then_statement;
```

```
1 if (4'b110z == 4'b110z)  
2 // not taken  
3 then_statement;
```

Boolean operations exist as usual:

bitwise operators: & (AND), | (OR), ~ (NOT), ^ (XOR) und auch ~^ (XNOR)

logic operators: && (AND), || (OR) und ! (NOT)

Shiftoperations: a « b (shift a for b positions to the left) und a » b (shift a for b positions to the right). A negative number b is not permitted, empty spots are filled with 0.



Parameters (old style)

In order to be able to adapt designs easier, Verilog offers the use of parameters.

```
1 module mux (
2     in1 , in2 ,
3     sel ,
4     out);
5
6     parameter WIDTH = 8; // Number of bits
7
8     input [WIDTH - 1 : 0] in1 , in2 ;
9     input sel;
10    output [WIDTH - 1 : 0] out;
11
12    assign out = sel ? in1 : in2;
13
14 endmodule
```



Instances and structural descriptions

If you describe a circuit through its (internal) structure or if a partial circuit is to be reused, an instance is generated and wired.

```
1 module xor2 (
2   input wire a,
3   input wire b,
4   output wire e);
5   assign e = a ^ b;
6 endmodule
```

```
1 module xor3 (
2   input wire a,
3   input wire b,
4   input wire c,
5   output wire e);
6
7   wire tmp;
8   xor2 xor2_1 // Instance 1
9   (
10     .a(a),
11     .b(b),
12     .e(tmp)
13   );
14   xor2 xor2_2 // Instance 2
15   (
16     .a(c),
17     .b(tmp),
18     .e(e)
19   );
20
21 endmodule
```



Code for sequential circuits

A flip-flop takes over the input of the rising or falling edges of the clock. For this, the block entry is used with the *@-symbol* and *always* blocks:

```
1 module FF (input  clk ,
2             input  rst ,
3             input  d,
4             output q);
5
6   reg q;
7
8   always @ ( posedge clk or
9             posedge reset)
10  begin
11    if ( rst )
12      q <= 1'b0;
13    else
14      q <= d;
15  end
endmodule
```

The list of signals after the *@-symbol* means sensitivity list. The reset is synchronized when you remove or *posedge reset*.



Section 3

Simple circuits: Combinational



Combinational circuits correspond to pure boolean functions and therefore do not contain the key word *reg*. No memory (flip-flops) gets generated and assignments are done with *assign*.

```
1 module mux4to1 (in1, in2, in3, in4, sel, out);
2
3 parameter WIDTH = 8;
4
5 input [WIDTH - 1 : 0] in1, in2, in3, in4;
6 input [1:0] sel;
7 output [WIDTH - 1 : 0] out;
8
9 assign out = (sel == 2'b00) ? in1 :
10      (sel == 2'b01) ? in2 :
11      (sel == 2'b10) ? in3 :
12      in4;
13 endmodule
```



Priority encoder

Similarly to the VHDL version, we describe the priority encoder as follows:

```
1 module prienc (input wire [4 : 1] req,
2                 output wire [2 : 0] idx);
3
4     assign idx = (req[4] == 1'b1) ? 3'b100 :
5                 (req[3] == 1'b1) ? 3'b011 :
6                 (req[2] == 1'b1) ? 3'b010 :
7                 (req[1] == 1'b1) ? 3'b001 :
8                 3'b000;
9
10    endmodule
```



Priority encoder (alternative version)

For a priority encoder you can use the *don't care* feature from Verilog.

```
1 module prienc (input [4:1] req,
2                 output reg [2:0] idx);
3
4     always @(*) begin
5         casez (req) // casez allows don't-care
6             4'b1????: idx = 3'b100; // Also: idx = 4;
7             4'b01???: idx = 3'b011;
8             4'b001?: idx = 3'b010;
9             4'b0001: idx = 3'b001;
10            default: idx = 3'b000;
11        endcase
12    end
13
14 endmodule
```



Section 4

Simple circuits: Sequential



Synchronous design

Contrary to combinational circuits, sequential circuits use internal memory, i.e. the output not only depends on the input.

In the synchronous method, all memory elements are checked / synchronized by a global clock. All calculations are carried out on the rising (and/or) falling edge of the clock.

The synchronous design enables the draft, test and the synthesis of large circuits with market tools. For this reason, it is advisable to remember this design principle.

Furthermore, there should be no (combinational) logic in the clock path, as this can lead to problems with the distribution times of the clock signals.



Synchronous circuits

The structure of synchronous circuits is idealized as follows:

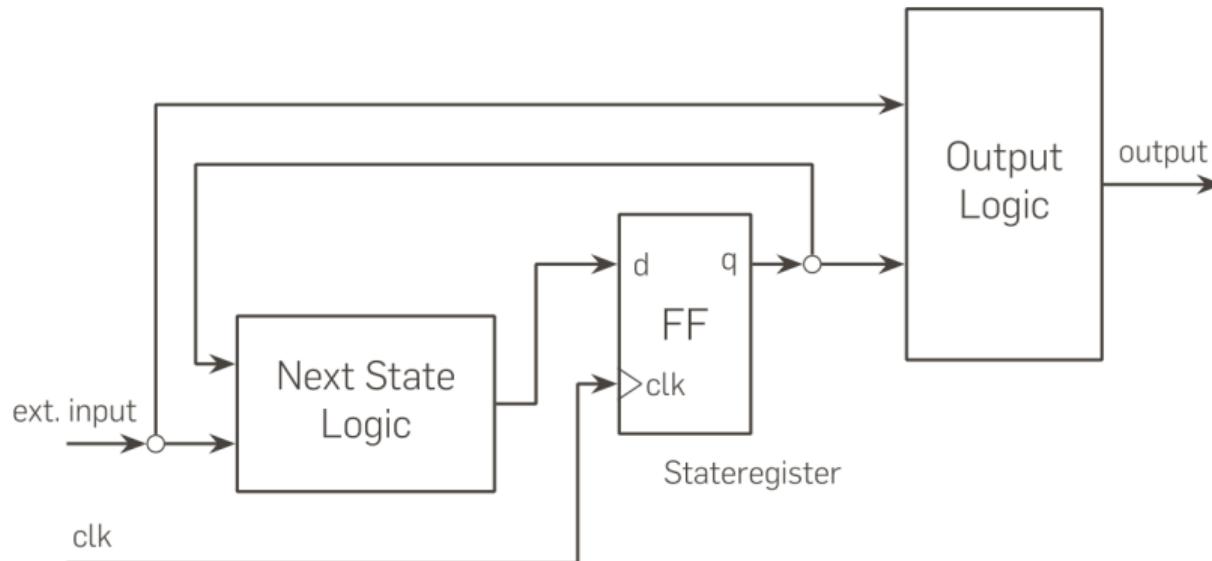


Figure 3: Idealized diagram of a synchronous circuit



A binary counter

According to the synchronous design, a free running binary counter can be realized:

```
1 module freecnt (value , clk , reset);
2
3     parameter WIDTH = 8;
4
5     input  wire  clk;
6     input  wire  reset;
7     output wire [WIDTH - 1 : 0] value;
8
9     wire [WIDTH - 1 : 0] valN;
10    reg   [WIDTH - 1 : 0] val;
11
12    always @ (posedge clk) begin
13
14        if (reset) begin // Synchron reset
15            val <= {WIDTH{1'b0}};
16        end else begin
17            val <= valN;
18        end
19
20    end
21
22    assign valN = val + 1; // Nextstate logic
23    assign value = val; // Output logic
24 endmodule
```



Synthesis result of the binary counter

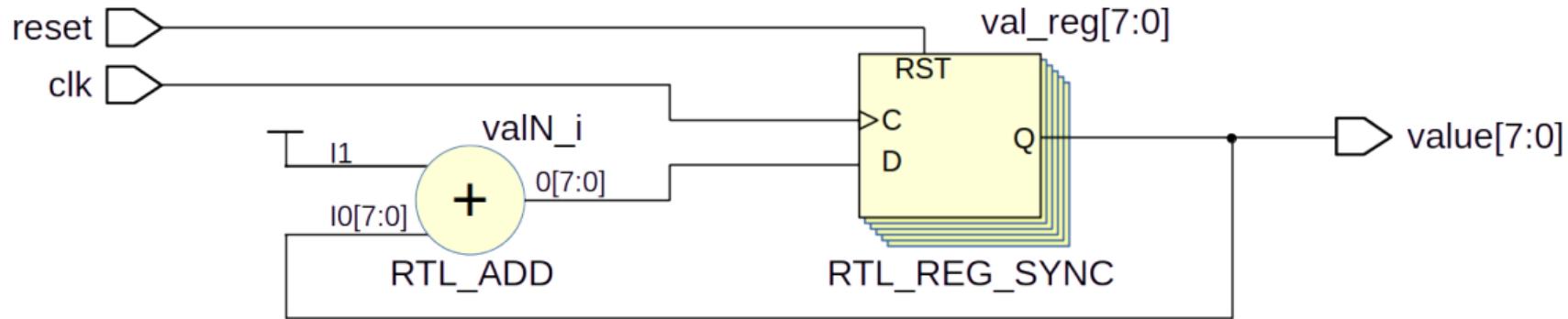


Figure 4: Synthesis diagram of the binary counter

At this point you can see that the result follows the diagram of the synchronous design.

RTL_REG_SYNC corresponds to the stateregister and *RTL_ADD* corresponds to the next state logic.



Some remarks

So far we use three assignment operators:

- assign signal0 = value
- signal2 <= value
- signal1 = value

The *assign* instruction is known as the continuous assignment and corresponds (roughly) to an ever active wire connection. It is used for signals of the type *wire* and is not permitted for *reg* (register).

The operator *<=* means non-blocking assignment. This assignment is used for synthesized registers, i.e. in *always-blocks* with *posedge clk* in the sensitivity list.

The variant *=* is called blocking assignment and is used for combinational *always-blocks*. Attention: Not allowed for signals of the type *wire*. So use the type *reg*.



A modulo counter

According to the synchronous design, a freely running modulo counter can be realized:

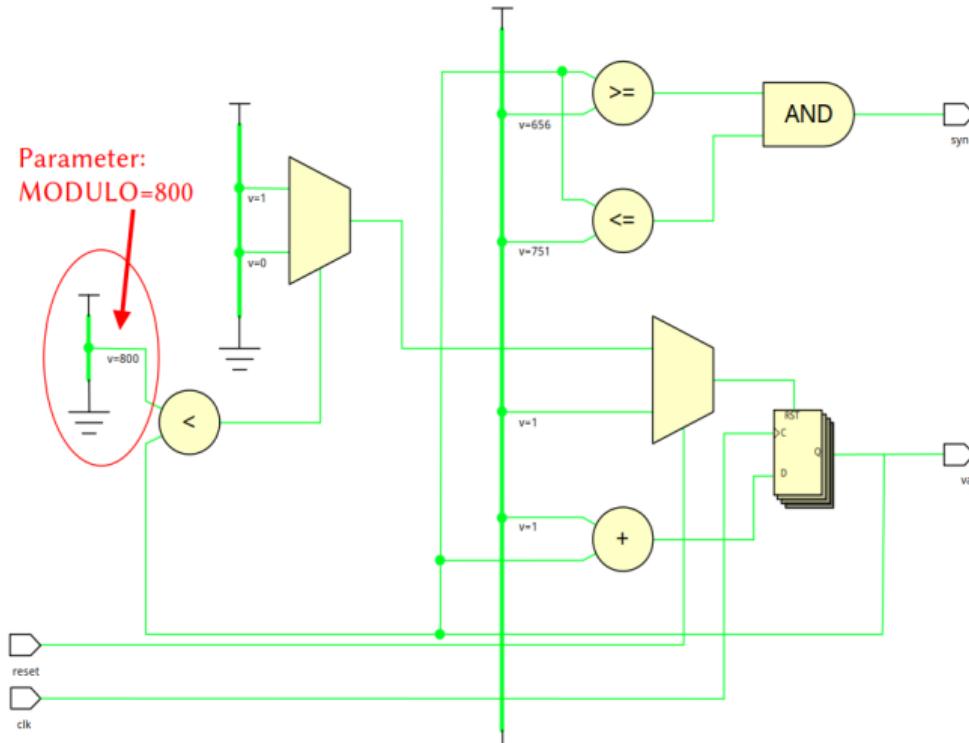
```
1 module modcnt (value, clk, reset, sync);
2
3     parameter WIDTH = 10,
4             MODULO = 800,
5             hsMin = 656,
6             hsMax = 751;
7
8     input wire clk;
9     input wire reset;
10    output wire [WIDTH - 1 : 0] value;
11    output wire sync;
12
13    wire [WIDTH - 1 : 0] valN;
14    reg [WIDTH - 1 : 0] val;
```

```
15    always @ (posedge clk) begin
16
17        if (reset) begin // Synchron reset
18            val <= {WIDTH{1'b0}};
19        end else begin
20            val <= valN;
21        end
22
23    end
24
25    // Nextstate logic
26    assign valN = (val < MODULO) ? val + 1 : 0;
27
28    // Output logic
29    assign value = val;
30    assign sync = ((val >= hsMin) && (val <= hsMax)) ? 1 : 0;
31
32 endmodule
```



Synthesis result of the modulo counter

In this case, next state logic and output logic are of course more complex:



A register file

RISC-V processors have a register file with a special zero register. Reading always provides 0 and writing operations are ignored.

```
1 module regfile (input clk,
2                  input [4:0] writeAddr, input [31 : 0] dataIn,
3                  input wrEn,
4                  input [4:0] readAddrA, output reg [31:0] dataOutA,
5                  input [4:0] readAddrB, output reg [31:0] dataOutB);
6
7 reg [31 : 0] memory [1 : 31];
8
9 always @ (posedge clk) begin
10
11   if ((wrEn) && (writeAddr != 0)) begin
12
13     memory[writeAddr] <= dataIn;
14
15   end
16
17   dataOutA <= (readAddrA == 0) ? 0 : memory[readAddrA];
18   dataOutB <= (readAddrB == 0) ? 0 : memory[readAddrB];
19
20 end
21
22 endmodule
```



Section 5

Selected feature: Parameterized counter



Selected feature: Parameterized counter

The newer variants of Verilog offer an improved version of the parameter feature:

```

1 module cnt
2 #(parameter N = 8,
3   parameter DOWN = 0)
4
5   (input clk,
6    input resetN,
7    input enable,
8    output reg [N-1:0] out);
9
10  always @ (posedge clk) begin
11
12    if (!resetN) begin // Synchron
13      out <= 0;
14    end else begin
15      if (enable)
16        if (DOWN)
17          out <= out - 1;
18      else
19        out <= out + 1;
20      else
21        out <= out;
22    end
23
24  end
25
26 endmodule

```

```

1 module doubleSum
2 #(parameter N = 8)
3   (input clk,
4    input resetN,
5    input enable,
6    output [N : 0] sum);
7
8   wire [N - 1 : 0] val0;
9   wire [N - 1 : 0] val1;
10
11 // Counter 0
12 cnt #(.N(N), .DOWN(0)) c0 (.clk(clk),
13 .resetN(resetN),
14 .enable(enable),
15 .out(val0));
16
17 // Counter 1
18 cnt #(.N(N), .DOWN(1)) c1 (.clk(clk),
19 .resetN(resetN),
20 .enable(enable),
21 .out(val1));
22
23 assign sum = val0 + val1;
24
25 endmodule

```

Synthesis result of the parameterized counter

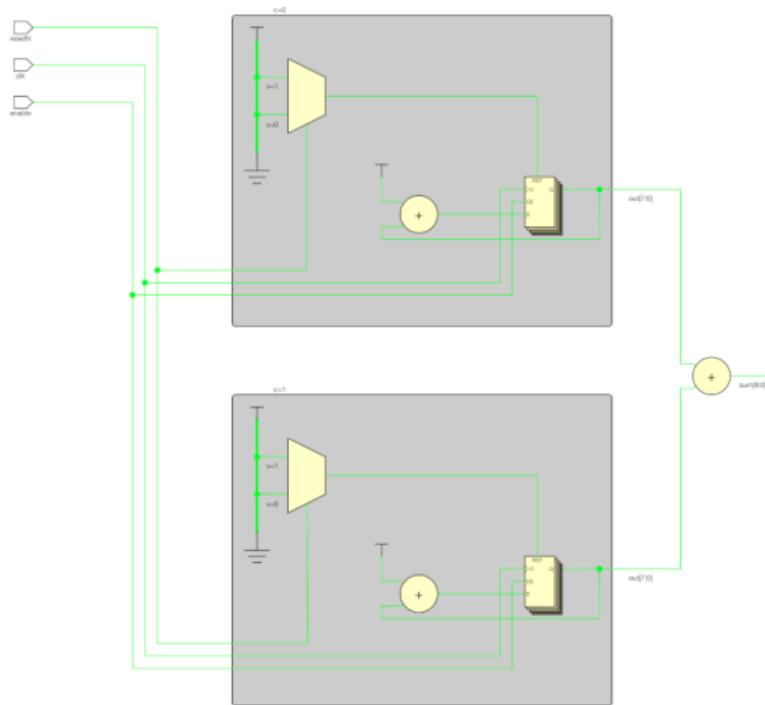


Figure 6: Parameterized counter



An alternative version

Verilog still offers a (older) possibility for the parameterization of a design:

```

1 module double
2   #(parameter N = 8)
3   (input clk,
4    input resetN,
5    input enable,
6    output [N : 0] sum);
7
8   wire [N - 1 : 0] val0;
9   wire [N - 1 : 0] val1;
10
11  // Counter 0
12  defparam c0.N = N;
13  defparam c0.DOWN = 0;
14  cnt c0 (.clk(clk),
15           .resetN(resetN),
16           .enable(enable),
17           .out(val0));

```

```

18 // Counter 1
19 defparam c1.N = N;
20 defparam c1.DOWN = 1;
21 cnt c1 (.clk(clk),
22           .resetN(resetN),
23           .enable(enable),
24           .out(val1));
25
26 assign sum = val0 + val1;
27
28 endmodule

```

This variant leads to the same synthesis result.



Section 6

Selected feature: Preprocessor



Selected feature: Preprocessor

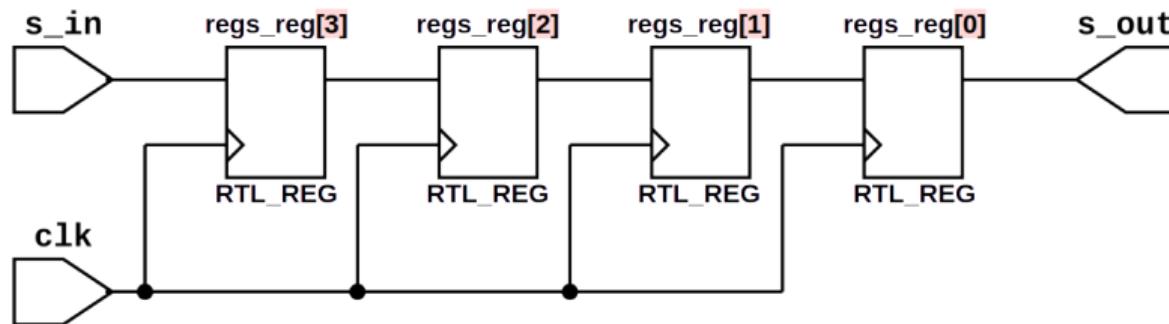
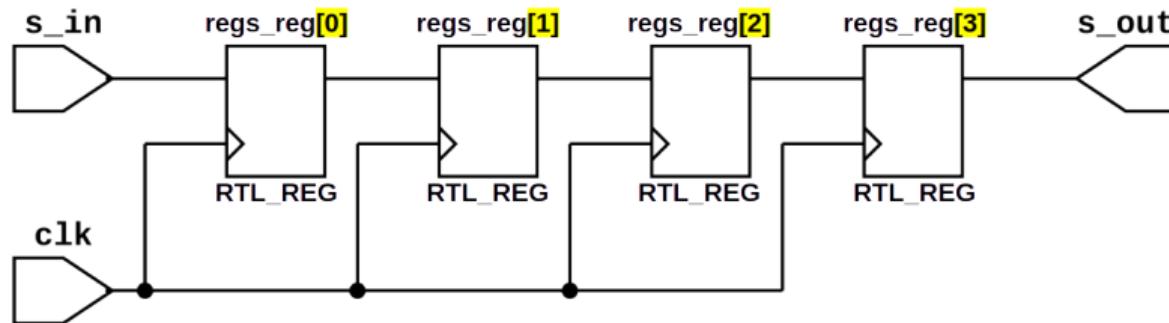
Verilog knows a preprocessor (cf. C/C++) with ‘define’, ‘include’ and ‘ifdef’. A *parameter* defines a constant and ‘define’ a text substitution.

```
1  `define SHIFT_RIGHT
2  module defineDemo (input clk, s_in,
3  output s_out);
4
5  reg [3:0] regs;
6
7  always @ (posedge clk) begin // next state logic in always-block
8    `ifdef SHIFT_RIGHT
9      regs <= {s_in, regs[3:1]};
10     `else
11       regs <= {regs[2:0], s_in};
12     `endif
13   end
14
15  `ifdef SHIFT_RIGHT
16    assign s_out = regs[0];
17  `else
18    assign s_out = regs[3];
19  `endif
20
21 endmodule
```



Two results of the synthesis

The conditional synthesis gives you two different shift registers:



Modularisation

Comparable to the include mechanism of C/C++, Verilog offers the possibility of primitive modularization with ‘include’.

The tick symbol ‘ is again the marker for a preprocessor command, comparable to # at C/C++.

With ‘include headers_def.h, for example, configuration settings from the file headers_def.h can be included. Since a pure text replacement is carried out, the file extension is basically arbitrary. It is meaningful to use .h analogous to C.

If a ‘define is arranged in front of a ‘include, the text replacement is also carried out in the included header file, i.e. a ‘define applies globally from the definition on. However, this can happen comparable to C unintentionally.



Section 7

Selected feature: Yosys and Systemverilog



Selected feature: Yosys and Systemverilog

The open source synthesetool Yosys provides some selected extensions from SystemVerilog.

- The logic datatype is particularly interesting, which simplifies allocations with `reg` and `wire`. With *logic signed* you declare signed numbers.
- The special block `always_ff` was introduced for sequential logic. Only non-blocking assignments (`<=`) are used for assignments.
- For combinatorial logic, `always_comb` replaces the construct `always @()`. Only blocking assignments (`=`) are used in `always_comb` blocks.



Another counter

Now the free running counter is to be re-implemented:

```
1 module freecnt2
2
3 #(parameter WIDTH = 8)
4   (input logic clk,
5    input logic reset,
6    output logic [WIDTH - 1 : 0] value);
7
8   logic [WIDTH - 1 : 0] valN;
9   logic [WIDTH - 1 : 0] val;
10
11  always_ff @(posedge clk) begin
12
13    if (reset) begin // Synchron reset
14      val <= {WIDTH{1'b0}};
15    end else begin
16      val <= valN;
17    end
18
19  end
20
21  always_comb begin
22
23    valN = val + 1; // Nextstate logic
24    value = val; // Output logic
25
26  end
27
endmodule
```

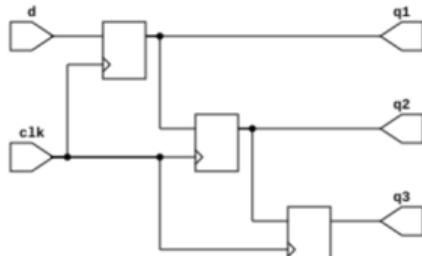
Blocking and Non-blocking assignments in always_ff

Caution with false assignments in *always_ff*:

```

1 module demoOk (input clk,
2   input d,
3   output q1,
4   output q2,
5   output q3);
6   always_ff @(posedge clk) begin
7     q1 <= d;
8     q2 <= q1;
9     q3 <= q2;
10    end
11 endmodule

```

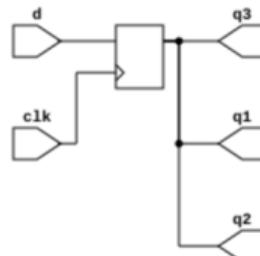


Okay:

```

1 module demoWrong (input clk,
2   input d,
3   output q1,
4   output q2,
5   output q3);
6   always_ff @(posedge clk) begin
7     q1 = d;
8     q2 = q1;
9     q3 = q2;
10    end
11 endmodule

```



Wrong:





Chapter 3 -
Verilog crash
course -
QUESTIONS

Course authors
(Git file)

Chapter 3 - Verilog crash course - QUESTIONS

Course authors (Git file)



Chapter 3 -
Verilog crash
course -
QUESTIONS

Course authors
(Git file)

Recap questions for Chapter 3



Questions

Chapter 3 -
Verilog crash
course -
QUESTIONS

Course authors
(Git file)

- What are HDL?
- What is Verilog?
- Are there other HDL?
- What is the main difference between HDL and programming languages?
- List and describe the structure of a simple Verilog file.
- Yosys is ... ?
- How to define ports in Verilog?
- Describe the concept of module instantiation in Verilog.
- What is the difference between combinational and sequential circuits?

Chapter 3 - Verilog - TRAINING - Advanced

Course authors (Git file)



1 LFSR - Linear Feedback Shift Register

LFSR - Linear Feedback Shift Register

This Training makes use of the Verilog code of the Linear Feedback Shift Register (LFSR) from the lecture slides.



LFSR as example

Task: Create directory and Verilog file

- Create a new directory for the LFSR example (in your Documents dir?)
- Create a new file `lfsr.v` inside that directory
- Copy the Verilog code from the lecture slides into the file `lfsr.v`



Analyse parts of the Verilog source

Task: Identify parts in the code

Find combinational and synchronous parts of the LFSR in

- the Verilog code
- the Schematic drawing (from the lecture slides)



Using yosys

Task: Learn to use yosys basics

Start using the tool `yosys`. You can get a basic help list with `yosys --help`. And for the commands it is

```
yosys --help <command>
```

- Learn how to synthesize a Verilog file to a JSON netlist.
- Learn to save the JSON netlist to a file

Tip: Use ChatGPT or a search engine to get information about how to use yosys.

Task: Netlist

Generate

- JSON Netlist

from the LFSR Verilog code

- Open the JSON netlist. Can you read it?

Chapter 3 - Verilog - TRAINING - Common

Course authors (Git file)



1

ORFS examples

ORFS examples

The Trainings for chapter 3 (Verilog) start with two of the given examples inside ORFS:

- The gcd (greatest common denominator) example
- The ibex (RISC-V) example

For this Training no Verilog code must be written. The Tasks are only about reading and understanding the existing Verilog code.



GDC example

Task: Find the gcd example

- Find the Verilog file of the gcd example in ORSF
- Open the Verilog file with an editor

Task: Questions about the code

- What are the port definitions
- How many inputs and outputs will the chip have?
- Can you identify other parts of the code?
- What would be the list of pins of a microchip with this design?



IBEX example

Task: Find the ibex example

- Find the top level Verilog file of the ibex example in ORSF
- Open the top level Verilog file with an editor

Task: Questions about the code

- What are the port definitions?
- How many inputs and outputs has the top level module?
- Can you identify other parts of the code?
- What would be the list of pins of a microchip with this design?



Chapter 04 - OpenROAD first run

Course authors (Git file)



1 Doing this chapter

2 The Makefile

3 The designs to run



Section 1

Doing this chapter



Doing this chapter

- This chapter is mostly a training.
- We will try to get our first results (GDS files).
- There are example designs available.
- You can start your own design.



Section 2

The Makefile



The Makefile

Let's have a look into the Makefile first.



The flow steps

The Makfile (in the `/flow` directory) contains all the flow steps in the same order we already have seen:

```
# =====
# SYNTHESIS
# =====
.PHONY: sv

# =====
# FLOORPLAN
# =====
.PHONY
# =====
# PLACE
# =====
.PHONY
# =====
# ROUTING
# =====
.PHONY: route
# =====
```

Figure 1: The flow steps in the Makefile



DESIGN_CONFIG

The Makefile starts with the selection of the design to run.

Of interest for this course are the lines regarding to the IHP PDK:

```
1 #DESIGN_CONFIG=./designs/ihp-sg13g2/aes/config.mk
2 #DESIGN_CONFIG=./designs/ihp-sg13g2/ibex/config.mk
3 DESIGN_CONFIG=./designs/ihp-sg13g2/gcd/config.mk
4 #DESIGN_CONFIG=./designs/ihp-sg13g2/spi/config.mk
5 #DESIGN_CONFIG=./designs/ihp-sg13g2/riscv32i/config.mk
6 #DESIGN_CONFIG=./designs/ihp-sg13g2/masked_aes/config.mk
```

The gcd example is selected for the next run.



Section 3

The designs to run



gcd (greatest common denominator)

- The gcd design is included with the OpenROAD-flow-script examples.
- It consists of only a single Verilog file, easy to read.
- Should run on the course server in a few minutes.



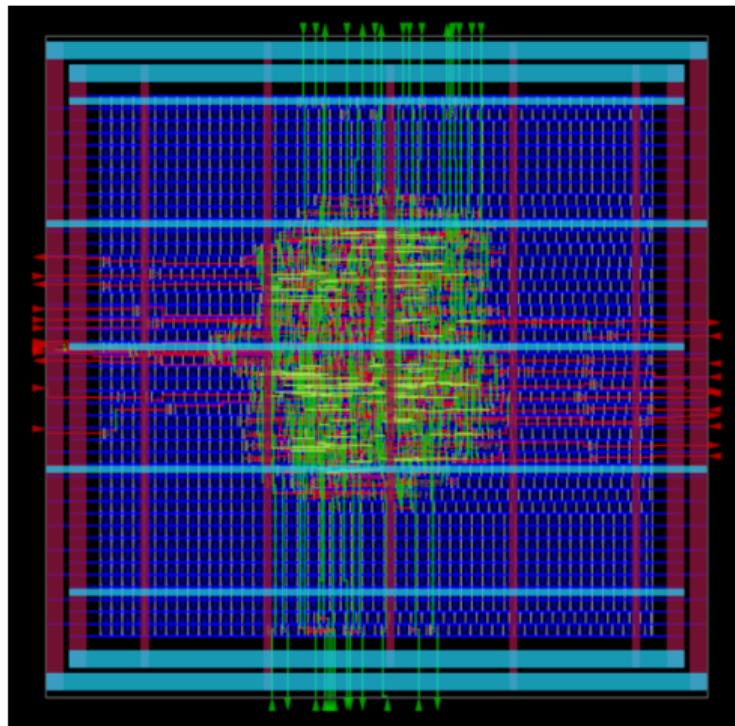


Figure 2: GDS gcd



gcd files:

```
1 | /src/gcd$ ls  
2 | gcd.v  
3 | README.md
```

```
1 | /ihp-sg13g2/gcd$ ls  
2 | autotuner.json  
3 | config.mk          (important)  
4 | constraint.sdc    (important)  
5 | metadata-base-ok.json  
6 | rules-base.json
```



ibex: RISC-V core

- The ibex design is included with the OpenROAD-flow-script examples.
- It consists of many Verilog files, not that easy to read.
- A single run might take more than 30 minutes on the course server.



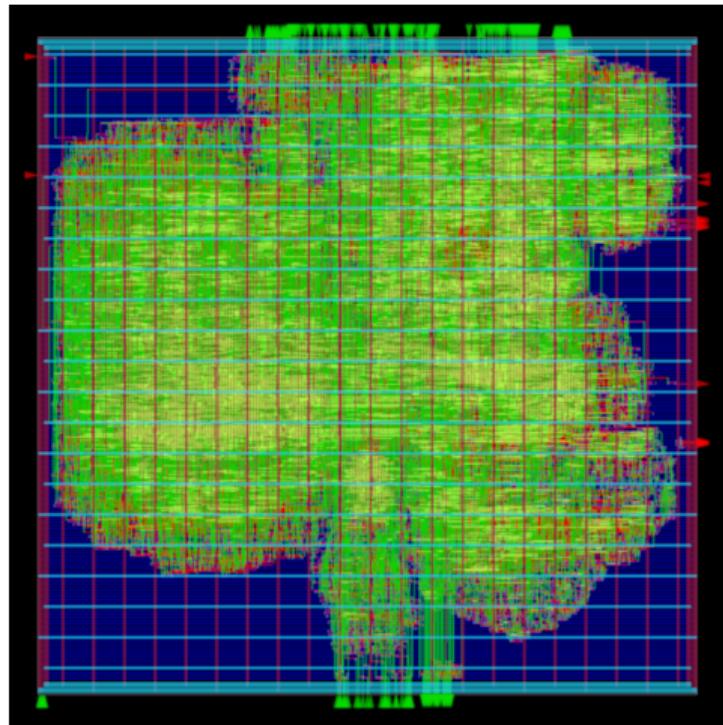


Figure 3: GDS ibex



ibex files:

```
1 src/ibex$ ls
2 ibex_alu.v           ibex_ex_block.v       ibex_register_file_ff.v   prim_ram_1p.v
3 ibex_branch_predict.v ibex_fetch_fifo.v     ibex_register_file_fpga.v prim_secded_28_22_dec.v
4 ibex_compressed_decoder.v ibex_icache.v      ibex_register_file_latch.v prim_secded_28_22_enc.v
5 ibex_controller.v     ibex_id_stage.v       ibex_wb_stage.v        prim_secded_39_32_dec.v
6 ibex_core.v          ibex_if_stage.v       LICENSE                prim_secded_39_32_enc.v
7 ibex_counter.v        ibex_load_store_unit.v prim_badbit_ram_1p.v  prim_secded_72_64_dec.v
8 ibex_cs_registers.v  ibex_multdiv_fast.v    prim_clock_gating.v   prim_secded_72_64_enc.v
9 ibex_csr.v            ibex_multdiv_slow.v   prim_generic_clock_gating.v prim_xilinx_clock_gating.v
10 ibex_decoder.v       ibex_pmp.v           prim_generic_ram_1p.v README.md
11 ibex_dummy_instr.v   ibex_prefetch_buffer.v prim_lfsr.v
```



```
1 ihp-sg13g2/ibex$ ls
2 autotuner.json
3 config.mk
4 constraint_doe.sdc
5 constraint.sdc
6 metadata-base-ok.json
7 rules-base.json
```



masked_aes

- The masked_aes design is part of a research project and is available on Github:
 - HEP Alliance - Masked AES
- It consists of three Verilog files, one of them >2000 lines of code.
- Should run in a few minutes on the course server.

Special Features:

- Contains I/O Pads and a Padring
- Has a README that contains the how-to of a sealring
- Has a README that links to the Metall filler script



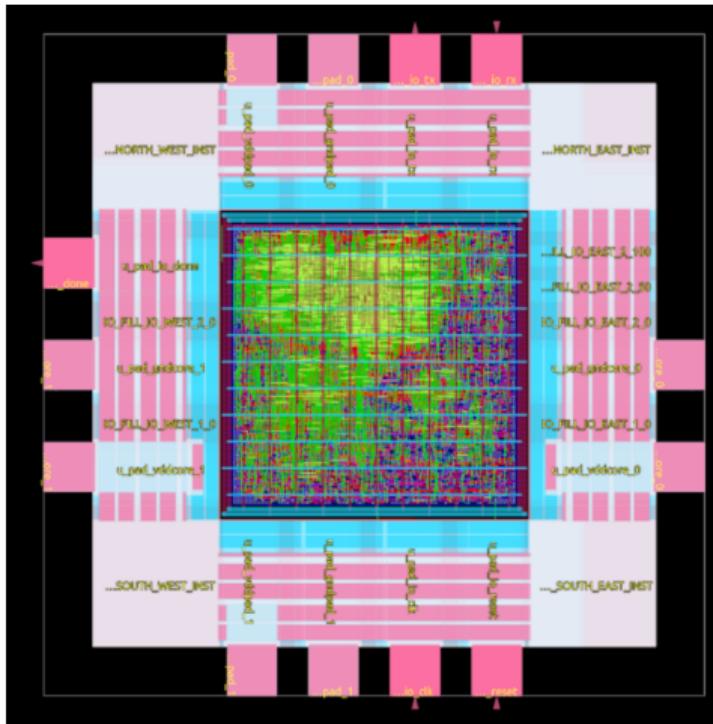


Figure 4: GDS masked_aes

masked_aes files:

```
1 ihp-sg13g2/masked_aes$ ls
2 config.mk
3 constraint.sdc
4 footprint.tcl
5 LICENSE
6 README.md
7 sealring.gds
8 src          (src directory!)
```

```
1 ihp-sg13g2/masked_aes/src$ ls
2 AES_Masked.v
3 AesTb.v
4 MaskedAes.v
```



Ifsr

- The Ifsr design example must be created from the scratch.
- The Verilog code is available in the lecture slides of chapter 3 and should become a single file.
- The structure of other examples must be copied for this.
- The configuration files must be copied and adapted for this.
- A single run should be very short.



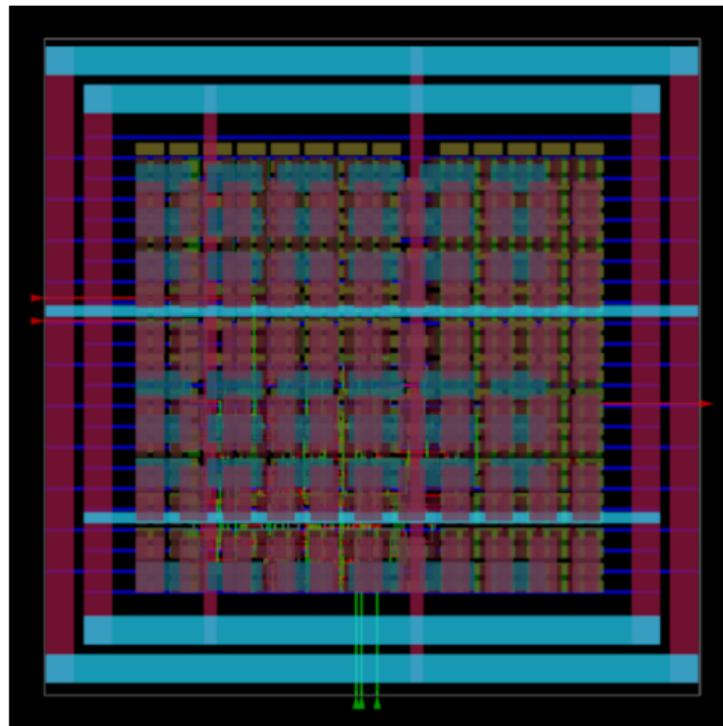


Figure 5: GDS Ifsr

A TinyTapeout design?

- The design example must be created from the scratch.
- The Verilog code is available as open-source via TinyTapeout
- The structure of other examples must be copied for this.
- The configuration files must be copied and adapted for this.
- The run time is unpredicted.

Suggestion:

The VGA clock example from the pictures earlier:

https://tinytapeout.com/runs/ttihp0p2/tt_um_vga_clock



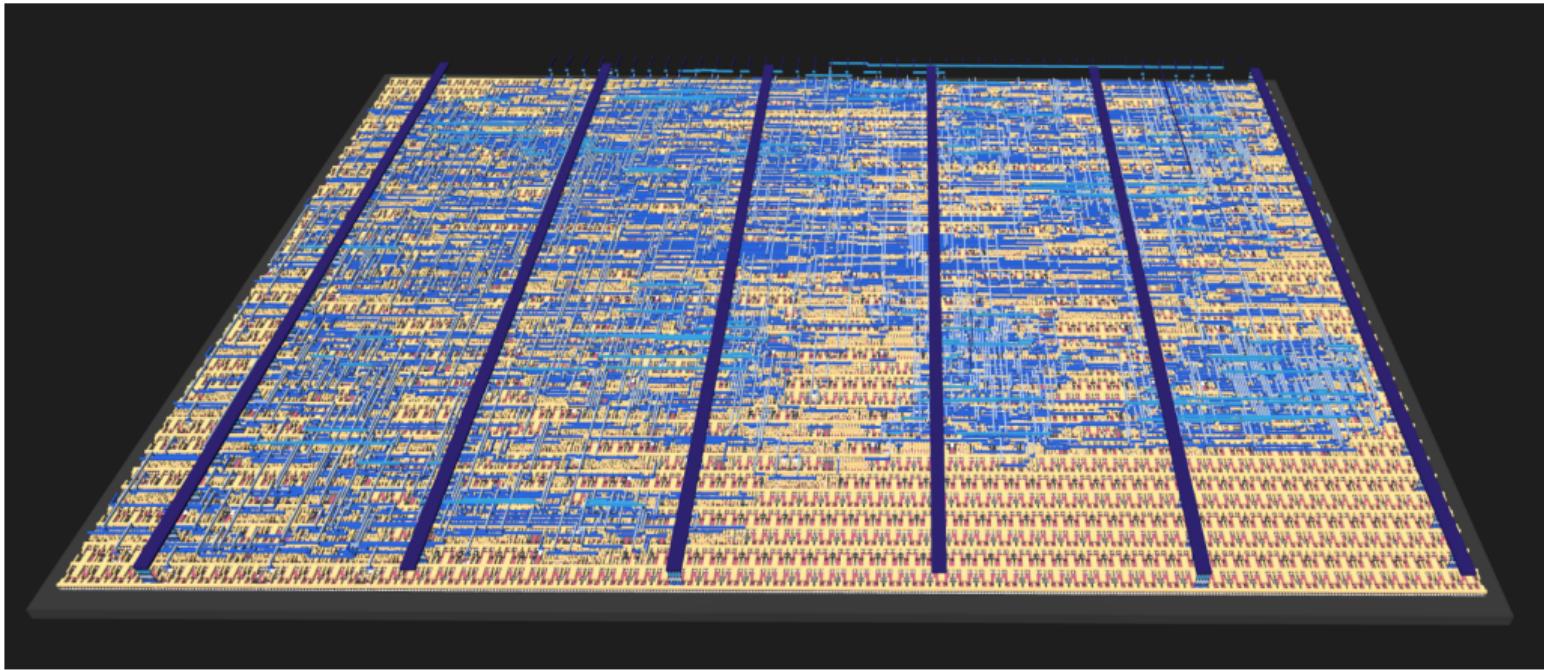


Figure 6: VGA Clock GDS render





Chapter 04 -
OpenROAD first
run -
QUESTIONS

Course authors
(Git file)

Chapter 04 - OpenROAD first run - QUESTIONS

Course authors (Git file)



Chapter 04 -
OpenROAD first
run -
QUESTIONS

Course authors
(Git file)

Recap questions for Chapter 4



Questions

Chapter 04 -
OpenROAD first
run -
QUESTIONS

Course authors
(Git file)

- How to choose the design for a run?
- How to do a run of ORFS?
- What is expected for a succesfull run?
- Name some of the steps in the console output that gets created during the run.
- What is the content of the final table?

Chapter 04 - OpenROAD first run - TRAINING - Advanced

Course authors (Git file)



- 1 Build an external example
- 2 Examine the results



“masked AES” from the HEP Alliance

Task: Clone the design

- Clone the masked AES design from Github.
- Use the tutorial from the HEP Alliance Repository:
<https://github.com/HEP-Alliance/masked-aes-tapeout>
- In a nutshell (clone via https):

```
1 | git clone https://github.com/HEP-Alliance/masked-aes-tapeout.git <ORFS-Root>/flow/designs/ihp-sg13g2/masked_aes
```



Makefile

Task: Enable the design

- Navigate to the `/flow` folder
- Edit the Makefile:
 - Uncomment the line with your chosen DESIGN_CONFIG from ihp-sg13g2. In this case the cloned masked_aes:

```
1 DESIGN_CONFIG=./ designs/ihp-sg13g2/masked_aes/config.mk
```

- Re-comment the previous uncommented line with DESIGN_CONFIG.
- The line with the default design does not need to be commented. This only applies when no previous line with DESIGN_CONFIG is set.



Run

Task: Run ORFS with the design

- Run `make` from inside the `/flow` folder.



Success

- The chosen design should finish after a while and a lot of console output with a table (time/memory) like this:

	Log	Elapsed	seconds	Peak	Memory/MB
1					

CONGRATS! Your design got build to a GDS!



The flow steps

Task: Match the shell output

- Scroll the shell output from the command to the (successfull) end,
- Identify the flow steps in the shell output
- Try to match your findings to the flow steps and flow components from chapter 2
- Can you identify single open-source tools in the output of the flow? Name the ones you identified.



The GDS

Task: Examine the GDS

- See the GDS with the command `make gui_final`

Task: Save an image from the GDS

- In the TCL console at the bottom of the GUI:
 - `save_image <imagename>.png`
 - Find the saved image in your directories.



Chapter 04 - OpenROAD first run - TRAINING - Bonus

Course authors (Git file)



1

Create a new design



Create a new design

In this training session you will integrate a new design for using it with OpenROAD flowscripts.

You can either:

- Have your own design ready.
- Take an opensource design from someone else.



Pick a design to integrate

Task: Choose a design

- LFSR from earlier?
- TinyTapeOut design? (See earlier Bonus Training)
- Write a new Verilog file with your own design idea.



Create a new design in ORFS

Task: Insert a new design

- In the `flow/designs/src` directory: create new design
- Create or copy Verilog to there
- In the `flow/designs/ihp-sg13g2` directory: create a new design(same name as in src)
- Create or copy the `config.mk` and `constraints.sdc` from the gcd example to there
- Modify both files to match the Verilog src file and top module.



Makefile

Task: Enable the design

- Navigate to the `/flow` folder
- Edit the Makefile:
 - Uncomment the line with your chosen DESIGN_CONFIG from ihp-sg13g2. In this case you must create a new line

```
1 DESIGN_CONFIG=./ designs/ihp-sg13g2/<your design>/config.mk
```

- Re-comment the previous uncommented line with DESIGN_CONFIG.
- The line with the default design does not need to be commented. This only applies when no previous line with DESIGN_CONFIG is set.



Run

Task: Run ORFS with the design

- Run `make` from inside the `/flow` folder.



Success

- The chosen design should finish after a while and a lot of console output with a table (time/memory) like this:

	Log	Elapsed	seconds	Peak	Memory/MB
1					

CONGRATS! Your design got build to a GDS!



The GDS

Task: Examine the GDS

- See the GDS with the command `make gui_final`

Task: Save an image from the GDS

- In the TCL console at the bottom of the GUI:
 - `save_image <imagename>.png`
 - Find the saved image in your directories.



Chapter 04 - OpenROAD first run - TRAINING - Common

Course authors (Git file)



1 gcd example

2 ibex: RISC-V



gcd example

Start the OpenROAD flow scripts for the gcd example. ORFS shall create a GDS in this run.



Makefile

Task: Enable the gcd design in the Makefile

- Navigate to the `/flow` folder
- Edit the Makefile:
 - Uncomment the line with `DESIGN_CONFIG` from `ihp-sg13g2` for the gcd example:

```
1 DESIGN_CONFIG=./ designs/ihp-sg13g2/gcd/config.mk
```

- Re-comment the previous uncommented line with `DESIGN_CONFIG`.
- The line with the default design does not need to be commented. This only applies when no previous line with `DESIGN_CONFIG` is set.



Run

Task: Run ORFS with the design

- Run `make` from inside the `/flow` folder.



Success

- The chosen design should finish after a while with a table (time/memory) like this:

	Log	Elapsed seconds	Peak Memory/MB
1	1_1_yosys	0	24
2	1_1_yosys_canonicalize	0	17
3	1_1_yosys_hier_report	0	12
4	2_1_floorplan	0	110
5	2_2_floorplan_io	0	106
6	2_3_floorplan_tdms	0	98
7	2_4_floorplan_macro	0	106
8	2_5_floorplan_tapcell	0	105
9	2_6_floorplan_pdn	0	108
10	3_1_place_gp_skip_io	0	108
11	3_2_place_iop	0	107
12	3_3_place_gp	0	320
13	3_4_place_resized	0	289
14	3_5_place_dp	0	112
15	4_1_cts	1	379
16	5_1_grt	0	340
17	5_2_route	93	899
18	5_3_fillcell	0	111
19	6_1_fill	0	113
20	6_1_merge	1	368
21	6_report	1	292
22	Total	96	899

The flow steps

Task: Match the shell output

- Scroll the shell output from the command to the (successfull) end,
- Identify the flow steps in the shell output
- Try to match your findings to the flow steps and flow components from chapter 2
- Can you identify single open-source tools in the output of the flow? Name the ones you identified.



The GDS

Task: Examine the GDS

- See the GDS with the command `make gui_final`

Task: Save an image from the GDS

- In the TCL console at the bottom of the GUI:
 - `save_image <imagename>.png`
 - Find the saved image in your directories.



Task: Create a GDS of the ibex design

- Do the same as with the gcd example, but now for the ibex example.
- Do the steps from above:
 - Makefile: Enable ibex design
 - Run ORFS with `make`
 - Examine the shell output
 - See the GDS with `make gui_final`
 - Save an image of the GDS.

Be aware: This ORFS run will take more than 30 minutes to finish!





Cheatsheet - Chapter 5 PDK Examination

KLayout:

.lyp	Layer properties file
.lyt	Technologies file for Layout ↔ Technology mapping

xScheme:

.sym	Schematics file

Hardware Description Languages (HDL):

Verilog
VHDL

Abbreviations:

LVS	Layout versus Schematic
CDL	Circuit design language
GDS II	Graphic data system (II)
LEF	Library exchange format
techLEF	Additional info about the technology
.lib	Liberty timing file: ASCII descriptions of timing / power of cells.

Chapter 5 - Process Design Kit (PDK)

Course authors (Git file)



1 What is a PDK?

2 Open-Source PDK and GitHub

3 Content of the PDK ihp-sg13g2

4 File formats

5 Example: Cell AO21

6 Ruleset documents



Section 1

What is a PDK?



Wikipedia definition

A process design kit (PDK) is a set of files used within the semiconductor industry to model a fabrication process for the design tools used to design an integrated circuit. The PDK is created by the foundry defining a certain technology variation for their processes. ...

... The designers use the PDK to design, simulate, draw and verify the design before handing the design back to the foundry to produce chips. The data in the PDK is specific to the foundry's process variation and is chosen early in the design process, influenced by the market requirements for the chip. An accurate PDK will increase the chances of first-pass successful silicon.

Source: https://en.wikipedia.org/wiki/Process_design_kit



Open-source viewpoint

- Semiconductor industry started to integrate open-source.
- Open-source PDKs created by Semiconductor fabs were a “missing link” between:
 - Open-source EDA tools (RTL-to-GDS) and
 - Microchip production (GDS-to-Chip)
- Since there is open-source PDKs, the growth of the open-source ecosystem is measurable.
- Many of the tools have been the classical “one-person maintained” open-source projects. It is getting better.



In the context of this course

The PDK (ihp-sg13g2) integrates seamless (to the user) into the OpenROAD flow scripts toolchain.

We have seen reference points from the tools onto the PDK in:

- the configuration files
- the structure of the design directories
- some Variables



Naming

PDKs sometimes are referred to as:

- Process design kits
- Process node
- Technology node
- Technology



Section 2

Open-Source PDK and GitHub



Difference from closed source

With publishing a PDK under a open-source license, the development from there on becomes a worldwide visible joint effort. The number of contributors and authors of the PDK can only increase from here on.



Collaborative workflow in GitHub

Some of the main principles of open-source are the permissions to use, study, change and re-distribute the published code and data according to the license. This leads to a open collaboration in which everyone can participate.

GitHub enables a workflow that was designed and build with these principles and opportunities in mind. A good starting point to explore the open collaboration in the IHP PDK are

- **Issues (open and closed)**
- **Pull requests (open and closed)**

The topics and discussions that you can read and study there will draw a picture of how the process of open collaboration works for the PDK.



Issues open

👉 Want to contribute to IHP-GmbH/IHP-Open-PDK?
If you have a bug or an idea, read the [contributing guidelines](#) before opening an issue.

Dismiss ▾

Filters ▾ Labels (9) Milestones (0) New issue

× Clear current search query, filters, and sorts

Author ▾	Label ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
35 Open	55 Closed				
<p>① [bug] ifnone state-dependent path delay #209 opened 2 weeks ago by likeamahoney</p>					
<p>① [bug] LRM specify block delay path restrictions bug #208 opened 2 weeks ago by likeamahoney</p>					
<p>① Simulation of MOSFET noise with ngspice #207 opened 2 weeks ago by 0y8w1x</p>					
<p>① .spiceinit seems to produce errors in AC simulation bug #205 opened 3 weeks ago by olsnr</p>					
<p>① how to use tolerances in LVS? #203 opened 3 weeks ago by olsnr</p>					
<p>① DRC seem to miss error on GatPoly Gat.b or Gat.b1 bug #201 opened 3 weeks ago by olsnr</p>					

Figure 1: Issues open

Course authors (Git file)

Chapter 5 - Process Design Kit (PDK)

11/41

Issues closed

💡 Want to contribute to IHP-GmbH/IHP-Open-PDK?
If you have a bug or an idea, read the [contributing guidelines](#) before opening an issue. Dismiss ▾

Filters ▾ Labels (9) Milestones (0) New Issue

Clear current search query, filters, and sorts

Author	Label	Projects	Milestones	Assignee	Sort
olisnr					55 Closed
olisnr					.spice net for LVS
olisnr					a question to LVS from .spice
hpreti	question				Resistor parameter b usage?
dnlitz					sram: missing .lib files
redpanda3	question				Any Recomended Python Versions?
sergeiandreyev	bug invalid				Routing issue in dfrbp_1 flop

User Icon 11 comments

User Icon 2 comments

User Icon 5 comments

User Icon 5 comments

User Icon 12 comments

User Icon 2 comments

Figure 2: Issues closed

Pull requests closed

First time contributing to IHP-GmbH/IHP-Open-PDK?Dismiss 

If you know how to fix an [issue](#), consider opening a pull request for it.
You can read this repository's [contributing guidelines](#) to learn how to open a good pull request.

Filters   Labels (9)  Milestones (0) New pull request

 Clear current search query, filters, and sorts

Author 	Label 	Projects 	Milestones 	Reviews 	Assignee 	Sort 
 3 Open	 115 Closed					
<p> iho-sg13g2: libs.ref: sg13g2_io: verilog: Fix specify syntax ✓  1 #215 by dnltz was merged last week</p> <p> Update KLayout DRC scripts ✓  #214 by akrinke was merged last week</p> <p> PyCell Klayout integration ✓  #213 by ThomasZecha was merged last week</p> <p> LVS rule decks: Fix GF180 remnants in log strings ✗  #212 by martinjankoheler was merged last week</p> <p> Updated procedure to avoid constant opening cmd window on Windows to... ✓  #211 by adatsuk was merged last week  2 tasks</p>						

Figure 3: Pull requests closed

License file

The IHp open-source PDK is published with an Apache 2.0 license:

<https://github.com/IHP-GmbH/IHP-Open-PDK/blob/main/LICENSE>

Apache 2.0 is a permissive open-source license. Read more about different open-source licenses here:

<https://choosealicense.com/licenses/>

Tip:

Know about the permissions, conditions and limitations of the licenses you are using for your projects!



Section 3

Content of the PDK ihp-sg13g2



The README

The Readme file in the PDKs repository is the starting point for information about the content of the PDK.

<https://github.com/IHP-GmbH/IHP-Open-PDK/blob/main/README.md>

The screenshot shows the GitHub repository page for 'IHP Open Source PDK'. The top navigation bar includes links for 'README', 'Code of conduct', and 'Apache-2.0 license'. The main content area features a large title 'IHP Open Source PDK' and a subtitle '130nm BiCMOS Open Source PDK, dedicated for Analog/Digital, Mixed Signal and RF Design'. Below this, a paragraph states the project goal: 'IHP Open Source PDK project goal is to provide a fully open source Process Design Kit and related data, which can be used to create manufacturable designs at IHP's facility.' A note at the bottom indicates the target process node: 'As of March 2023, this repository is targeting the SG13G2 process node.' To the right, there are sections for 'Contributors' (12), showing profile icons for contributors like Natasja, Matheus, and others, and a 'Languages' chart. The chart shows the distribution of code files by language: HTML (66.6%), Python (27.4%), Verilog (3.7%), MATLAB (2.1%), and Makefile (0.2%).

Contributors (12)

HTML 66.6% Python 27.4%

Verilog 3.7% MATLAB 2.1%

Makefile 0.2%

Figure 4: Readme

Project roadmap

A GANTT chart of the roadmap for the open-source PDK is available under this weblink. It shows the projects timeline (2022 - 2026):

https://github.com/IHP-GmbH/IHP-Open-PDK/blob/main/ihp-sg13g2/libs.doc/roadmap/open_PDK_gantt.png

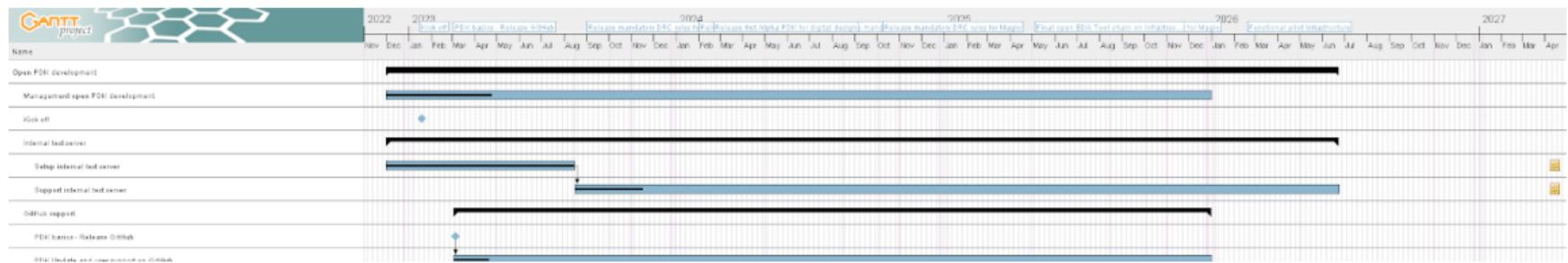


Figure 5: Gantt chart



Tool support

The PDK supports a bunch of tools with the included file formats. A list of the file types and formats follows later in this chapter.

The main tools for this course are

- OpenROAD flow scripts
- OpenROAD
- KLayout



The list of supported tools from the README in the PDK:

Supported EDA Tools

- ngspice
 - Download: <https://ngspice.sourceforge.io/download.html>
 - Source: <https://sourceforge.net/p/ngspice/ngspice/ci/master/tree>
- Xyce
 - Download: <https://xyce.sandia.gov/downloads/executables>
 - Source: <https://github.com/Xyce/Xyce>
- xschem
 - Source: <https://github.com/StefanSchippers/xschem>
- Qucs-S
 - Download: <https://ra3xdh.github.io>
 - Source: https://github.com/ra3xdh/qucs_s
- KLayout
 - Download: <https://www.klayout.de/build.html>
 - Source: <https://github.com/KLayout/klayout>
- OpenEMS
 - Source: <https://github.com/thliebig/openEMS-Project>
- OpenROAD
 - Source: <https://github.com/The-OpenROAD-Project/OpenROAD>
- OpenROAD-flow-scripts
 - Source: <https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts>

Figure 6: Supported tools

Cells in the PDK

There are four different sets of cells (or devices) in the PDK:

- Base cellset with limited set of standard logic cells
 - CDL, GDSII, LEF, Tech LEF
 - Liberty, SPICE Netlist, Verilog
- IO cellset
 - GDSII, LEF, Liberty (dummy), SPICE Netlist
- SRAM cellset
 - CDL, GDSII, LEF, Liberty, Verilog
- Primitive devices
 - GDSII



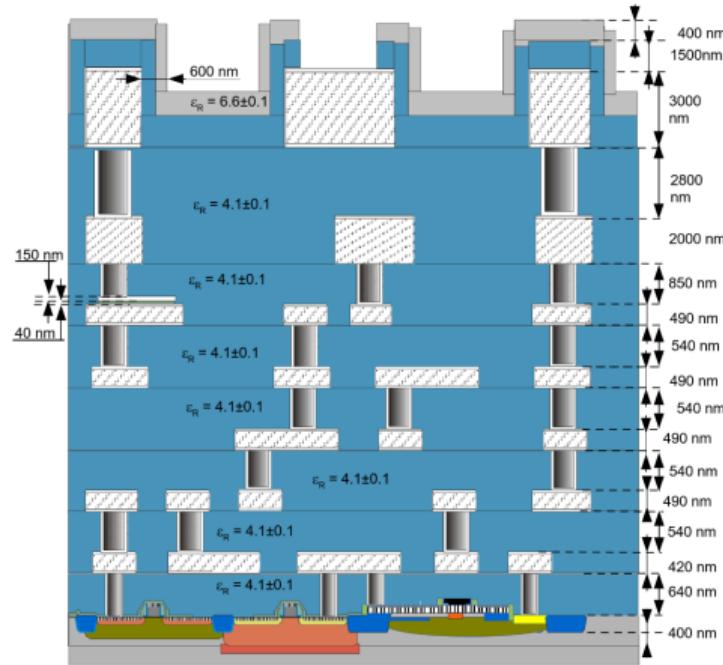
Other data in the PDK

- KLayout tool data:
 - layer property and tech files
 - DRC rules (minimal set)
 - PyCells
 - initial version of the wrapper API
 - sample cells
- Pcells (for reference only) libs.tech/pycell
- MOS/HBT/Passive device models for ngspice/Xyce
- xschem: primitive device symbols, settings and testbenches
- OpenEMS: tutorials, scripts, documentation
- SG13G2 Process specification & Layout Rules
- MOS/HBT Measurements in MDM format
- Project Roadmap Gantt chart



Layer stack

IHP sg13g2 Layers in a picture. [Source PDF](#)



Design rules

- The DRC (design rules check) with ORFS happens in KLayout.
- The data for the minimal and maximal checks is here:

<https://github.com/IHP-GmbH/IHP-Open-PDK/tree/main/ihp-sg13g2/libs.tech/klayout/tech/drc>



Name	Last commit message	Last commit date
 ..		
 MissingRules_maximal.md	Update KLayout DRC scripts	8 months ago
 README.md	Update KLayout DRC scripts	8 months ago
 README_maximal.md	Update KLayout DRC scripts	8 months ago
 README_minimal.md	KLayout DRC: added first version of full DRC rule deck	9 months ago
 sg13g2_maximal.lydrc	Update KLayout DRC scripts	5 months ago
 sg13g2_minimal.lydrc	Update KLayout DRC scripts	5 months ago
README.md		
Minimum Rule Set - README		
Maximum Rule Set - README , MissingRules		

Figure 8: DRC files lydrc for KLayout

Layout versus Schematic

- The LVS check with ORFS happens in KLayout.
- The data for the LVS is here:

<https://github.com/IHP-GmbH/IHP-Open-PDK/tree/main/ihp-sg13g2/libs.tech/klayout/tech/lvs>



Name	Last commit message	Last commit date
..		
images	Updating GUI menus to automatically detect active cell	9 months ago
rule_decks	Merge pull request #212 from martinjankohehler/fix-gf180-remnants	5 months ago
testing	Updating custom writer for LVS runset	7 months ago
README.md	Updating GUI menus to automatically detect active cell	9 months ago
run_lvs.py	Adding some deep/flat tests, Updating FETs/RFFETs derivations, Adding...	9 months ago
sg13g2.lvs	Updating chip derivation, Adding more logs for lvs run, updating lvs ...	9 months ago
sg13g2_full.lylvs	Merge pull request #224 from TinyTapeout/pr-fix-180-ref	4 months ago

Figure 9: LVS files lylvs for KLayout



Section 4

File formats



List of file types and formats

CDL: Circuit design language [Link](#)

LEF: Library Exchange Format [Link](#)

TechLEF: Technology LEF [Link](#)

GDS II: Graphic data system II [Link](#)

lib: Liberty timing and power file [Link](#)

sym: Symbol file (Xschem) [Link](#)

sch: Schematic file (Xschem) [Link](#)

lyp: Layer properties file (KLayout)

lyt: Technology mapping file (KLayout)

lydrc: DRC rules file (KLayout)

llyvs: LVS rule deck (KLayout)



Section 5

Example: Cell AO21



Cell AO21: GDS in KLayout

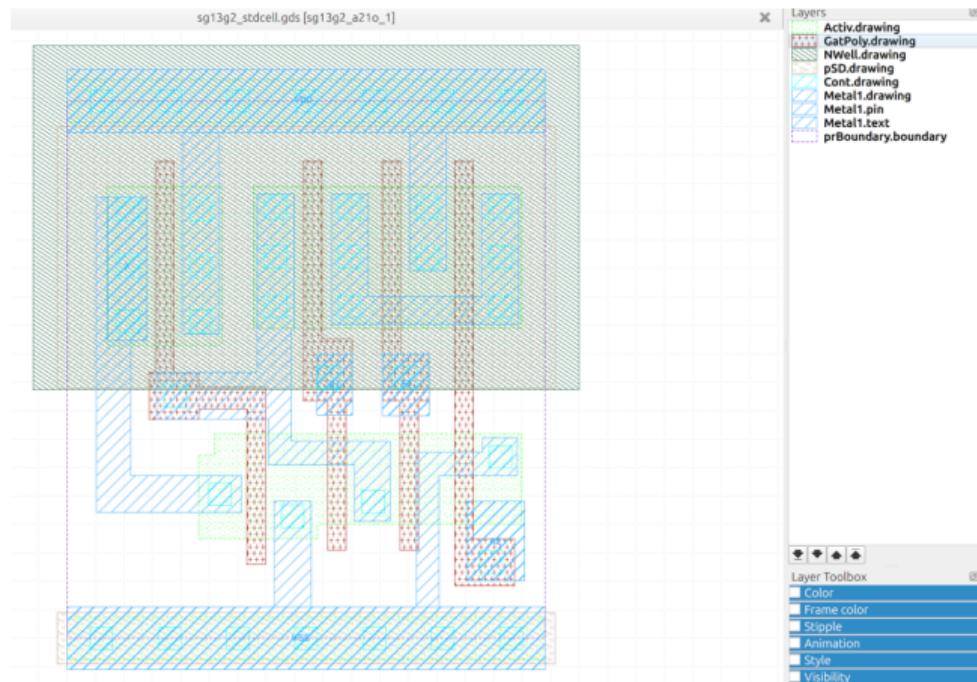


Figure 10: Cell AO21 GDS

Cell AO21: VERILOG HDL language

```
1 // type: AO21
2 `timescale 1ns/10ps
3 `celldefine
4 module sg13g2_a21o_1 (X, A1, A2, B1);
5   output X;
6   input A1, A2, B1;
7
8   // Function
9   wire int_fwire_0;
10
11  and (int_fwire_0 , A1, A2);
12  or (X, int_fwire_0 , B1);
13
14 // Timing
15 specify
16   (A1 => X) = 0;
17   (A2 => X) = 0;
18   if (A1 == 1'b1 & A2 == 1'b0)
19     (B1 => X) = 0;
20   if (A1 == 1'b0 & A2 == 1'b1)
21     (B1 => X) = 0;
22   ifnone (B1 => X) = 0;
23 endspecify
24 endmodule
`endcelldefine
```

Cell AO21: SPICE Netlist

```
1 * Library name: sg13g2_stdcell
2 * Cell name: sg13g2_a21o_1
3 * View name: schematic
4 * Inherited view list: spectre cmos_sch cmos.sch schematic veriloga ahdl
5 * pspice dspf
6 .subckt sg13g2_a21o_1 A1 A2 B1 VDD VSS X
7 XN0 net1 A1 net2 VSS sg13_lv_nmos w=640.00n l=130.00n ng=1 ad=0 as=0 pd=0 ps=0 m=1
8 XN1 net2 A2 VSS VSS sg13_lv_nmos w=640.00n l=130.00n ng=1 ad=0 as=0 pd=0 ps=0 m=1
9 XN2 net1 B1 VSS VSS sg13_lv_nmos w=640.00n l=130.00n ng=1 ad=0 as=0 pd=0 ps=0 m=1
10 XN3 X net1 VSS VSS sg13_lv_nmos w=740.00n l=130.00n ng=1 ad=0 as=0 pd=0 ps=0 m=1
11 XP0 net1 B1 net3 VDD sg13_lv_pmos w=1.000u l=130.00n ng=1 ad=0 as=0 pd=0 ps=0 m=1
12 XP1 net3 A1 VDD VDD sg13_lv_pmos w=1.000u l=130.00n ng=1 ad=0 as=0 pd=0 ps=0 m=1
13 XP2 net3 A2 VDD VDD sg13_lv_pmos w=1.000u l=130.00n ng=1 ad=0 as=0 pd=0 ps=0 m=1
14 XP3 X net1 VDD VDD sg13_lv_pmos w=1.12u l=130.00n ng=1 ad=0 as=0 pd=0 ps=0 m=1
15 .ends
* End of subcircuit definition.
```



Cell AO21: Circuit design language

```
1 ****
2 * Library Name: sg13g2_stdcell
3 * Cell Name:    sg13g2_a21o_1
4 * View Name:   schematic
5 ****
6
7 .SUBCKT sg13g2_a21o_1 A1 A2 B1 VDD VSS X
8 *.PININFO A1:I A2:I B1:I X:O VDD:B VSS:B
9 MN0 net1 A1 net2 VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
10 MN1 net2 A2 VSS VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
11 MN2 net1 B1 VSS VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
12 MN3 X net1 VSS VSS sg13_lv_nmos m=1 w=740.00n l=130.00n ng=1
13 MP0 net1 B1 net3 VDD sg13_lv_pmos m=1 w=1.000u l=130.00n ng=1
14 MP1 net3 A1 VDD VDD sg13_lv_pmos m=1 w=1.000u l=130.00n ng=1
15 MP2 net3 A2 VDD VDD sg13_lv_pmos m=1 w=1.000u l=130.00n ng=1
16 MP3 X net1 VDD VDD sg13_lv_pmos m=1 w=1.12u l=130.00n ng=1
17 .ENDS
```



Cell AO21: LEF

```
1 MACRO sg13g2_a21o_1
2   CLASS CORE ;
3   ORIGIN 0 0 ;
4   FOREIGN sg13g2_a21o_1 0 0 ;
5   SIZE 3.36 BY 3.78 ;
6   SYMMETRY X Y ;
7   SITE CoreSite ;
8   PIN A2
9     DIRECTION INPUT ;
10    USE SIGNAL ;
11    ANTENNAMODEL OXIDE1 ;
12      ANTENNAGATEAREA 0.2132 LAYER Metal1 ;
13      PORT
14        LAYER Metal1 ;
15        RECT 2.81 0.405 3.215 0.965 ;
16      END
17    END A2
18    PIN A1
19      DIRECTION INPUT ;
20      USE SIGNAL ;
21      ANTENNAMODEL OXIDE1 ;
22      ANTENNAGATEAREA 0.2132 LAYER Metal1 ;
23      PORT
24        LAYER Metal1 ;
25        RECT 2.215 1.565 2.545 2 ;
26      END
27 END_A1
```

Cell AO21: Schematic in XScheme

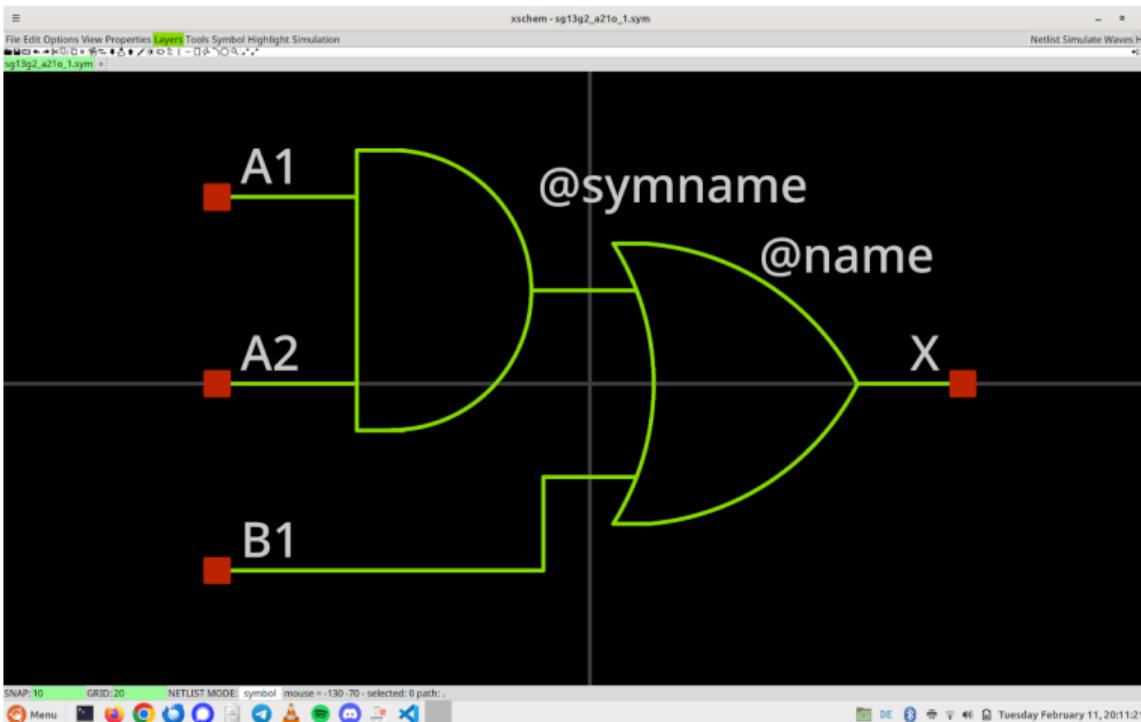


Figure 11: AO21 in XScheme

Liberty files

- Liberty files (.lib) contain information about timing, power and temperature of the cells.
- The ihp130-sg13g2 PDK contains six different Liberty files for the standard cells.
- These six files are categorized by nominal voltages and nominal temperatures:

Liberty file	Voltage	Temperature
sg13g2_stdcell_fast_1p32V_m40C.lib	1.32 V	-40° C
sg13g2_stdcell_fast_1p65V_m40C.lib	1.65 V	-40° C
sg13g2_stdcell_slow_1p08V_125C.lib	1.08 V	125° C
sg13g2_stdcell_slow_1p35V_125C.lib	1.35 V	125° C
sg13g2_stdcell_typ_1p20V_25C.lib	1.20 V	25° C
sg13g2_stdcell_typ_1p50V_25C.lib	1.50 V	25° C

- The PDK contains .lib files for RAM macros and io cells too.



Cell AO21: Liberty description

- The cell AO21 (first cell in the standard cell library list) has 622 lines of data in the lib file.
- The cell description contains:
 - First comes the leakage power for the whole truth table on the inputs.
 - After that each pin gets described with its timing and power characteristics
 - The order of the pins starts with the output. Order: X, A1, A2, B1



Cell AO21: Liberty file extract:

```
1  cell (sg13g2_a21o_1) {  
2      area : 12.7008;  
3      cell_footprint : "AO21";  
4      cell_leakage_power : 158.343;  
5      leakage_power () {  
6          value : 163.606;  
7          when : "!A1&!A2&!B1";  
8      }  
9  
10     ...  
11  
12     pin (X) {  
13         direction : "output";  
14         function : "((A1*A2)+B1)";  
15     ...  
16  
17     pin (B1) {  
18         direction : "input";  
19         max_transition : 2.5074;  
20     ...  
21  
22     ...
```



Section 6

Ruleset documents



Layout rules document

<https://github.com/IHP-GmbH/IHP-Open-PDK/tree/main/ihp-sg13g2/libs/doc/doc>



Figure 12: Layout rules document



Process specification document

<https://github.com/IHP-GmbH/IHP-Open-PDK/tree/main/ihp-sg13g2/libs.doc/doc>



Figure 13: Process specification





Chapter 5 -
Process Design
Kit (PDK) -
QUESTIONS

Course authors
(Git file)

Chapter 5 - Process Design Kit (PDK) - QUESTIONS

Course authors (Git file)



Chapter 5 -
Process Design
Kit (PDK) -
QUESTIONS

Course authors
(Git file)

Recap questions for Chapter 5



Questions

Chapter 5 -
Process Design
Kit (PDK) -
QUESTIONS

Course authors
(Git file)

- What is a PDK?
- Define the border between development (design) and production of a microchip.
- Where is the (actual) border of open- to closed-source in this?
- Name as many parts of the IHP open-source PDK as you can (round robin)
- What are layers?
- What is a layer stack?
- Can you name some layers and their purpose?
- Name some of the important file formats from the PDK. What are they used for? (round robin).
- Describe the content of the design rules document.
- Which tool does the DRC?
- Which tool does the LVS?

Chapter 5 - PDK Examination - TRAINING - Advanced

Course authors (Git file)



- 1 Start with common training
- 2 Run a 2.5d script
- 3 Examine th cell in 2.5d
- 4 Add layers in the script
- 5 View a more complex design



Start with common training

Task: Common training first

- Do the common training from chapter 05 first.
- You should have
 - a standard cell open in KLayout.
 - the PDK lyp file loaded.



Run a 2.5d script

Task: Run 2.5 script

- Open the 2.5d script editor:
 - Tools -> 2.5d View -> New 2.5d Script
- Press the play button (in the middle of the upper tool row)
- The Viewer should pop up and show the cell in 2.5d



Examine th cell in 2.5d

Task: Examine 2.5d view

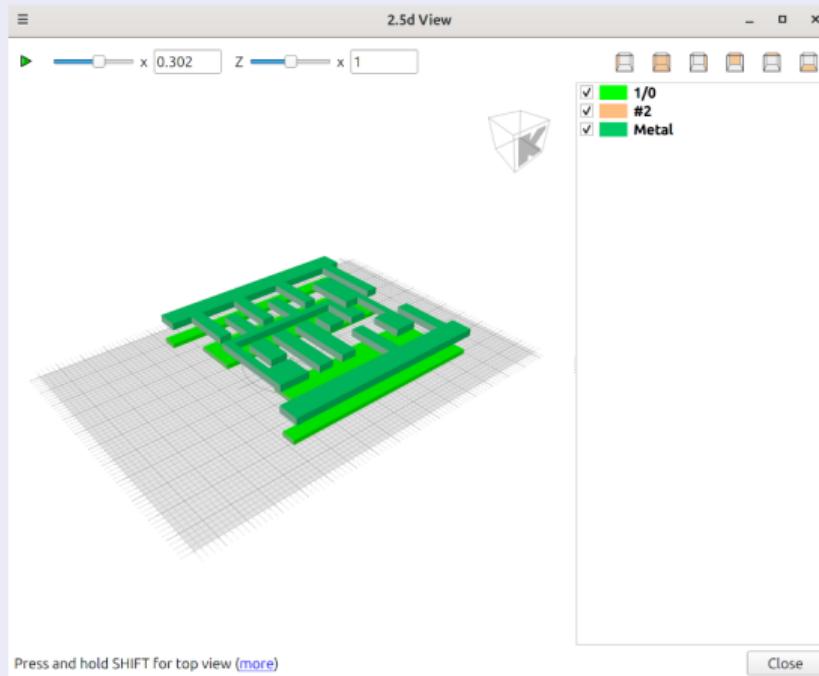


Figure 1: Standard cell in 2.5d viewer

Add layers in the script

- You will see only a little of the standard cells layers.
- That is because the script is very basic.

Task: Enhance the 2.5d script

- Try to add at least one more layer to the 2.5d script
- Run again and see the result.

Tip: Cheat from a pre-made script

Here is a script with some more layers in it:

<https://gist.github.com/rslawson/fecc97d8731da95204ed59b00571582f>



View a more complex design

Task:

- Load a complete design GDS into KLayout,
- Run a 2.5d script and see the viewer.



Chapter 5 - PDK Examination - TRAINING - Bonus

Course authors (Git file)



- 1 1. Transistor count
- 2 Next slide: Solution Spoiler!
- 3 Solution: Transistor count
- 4 Solution: Transistor count in file



1. Transistor count

- Load the gds of the standard cell AND4_1.
- How many transistors are in the cell?
- How to verify this with the use of another file from the PDK?



Next slide: Solution Spoiler!

SPOILER ALERT:

- The next slide contains the solution.
- Only proceed to the next slide if you want to see the solution right now!



Solution: Transistor count

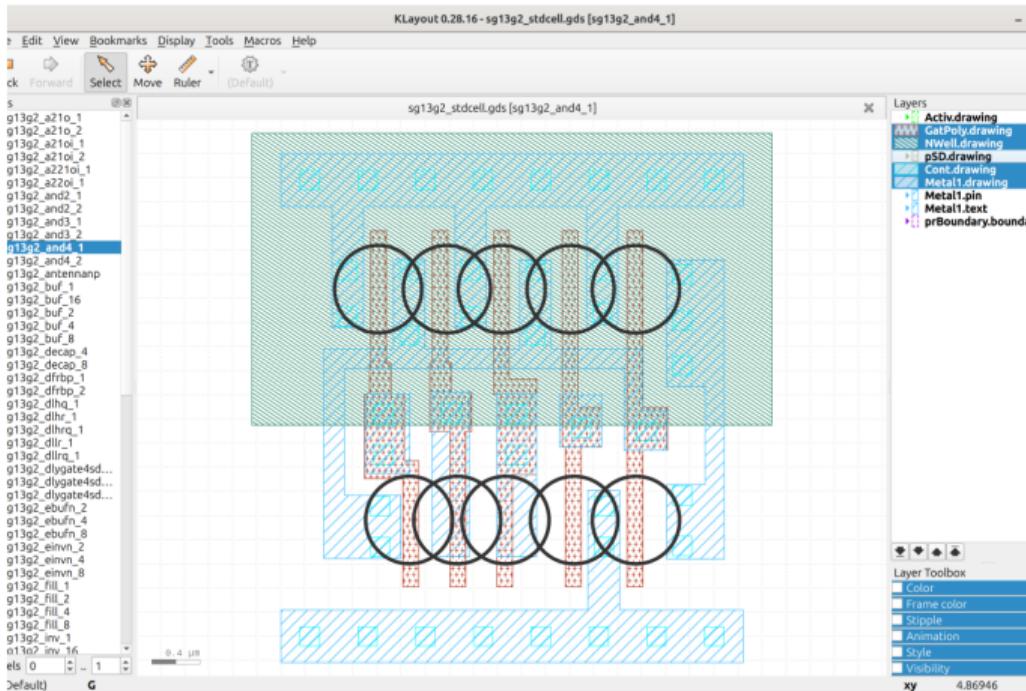


Figure 1: AND4_1

Solution: Transistor count in file

File: sg13g2_sdtcell.cdl

```
1 MN4 net17 D VSS VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
2 MN3 net16 C net17 VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
3 MN2 net15 B net16 VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
4 MN1 net1 A net15 VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
5 MNO X net1 VSS VSS sg13_lv_nmos m=1 w=740.00n l=130.00n ng=1
6 MP0 net1 A VDD VDD sg13_lv_pmos m=1 w=840.00n l=130.00n ng=1
7 MP4 X net1 VDD VDD sg13_lv_pmos m=1 w=1.12u l=130.00n ng=1
8 MP3 net1 D VDD VDD sg13_lv_pmos m=1 w=840.00n l=130.00n ng=1
9 MP2 net1 C VDD VDD sg13_lv_pmos m=1 w=840.00n l=130.00n ng=1
10 MP1 net1 B VDD VDD sg13_lv_pmos m=1 w=840.00n l=130.00n ng=1
```



Chapter 5 - PDK Examination - TRAINING - Common

Course authors (Git file)



1 1. Open Klayout

2 2. Load example GDS

3 3. Use a LYP file

4 4. Navigate Layers and GDS

5 5. Load cell library GDS

6 6. Pick a cell



1. Open Klayout

- Execute `klayout` in console shell.
- Klayout starts in viewer mode.
- Edit mode can be started with `klayout -e` but is not needed for this training.



1. Open KLayout

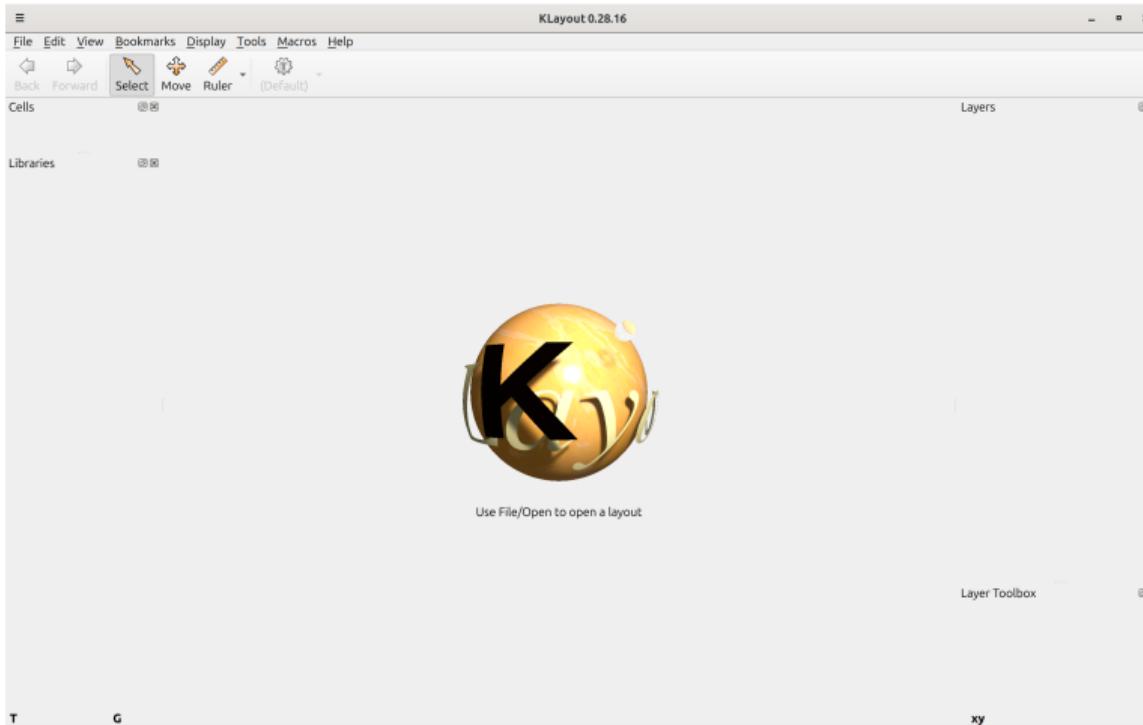


Figure 1: Start KLayout

2. Load example GDS

- Search for the final GDS from your example run and load it into Klayoutv



2. Load example GDS

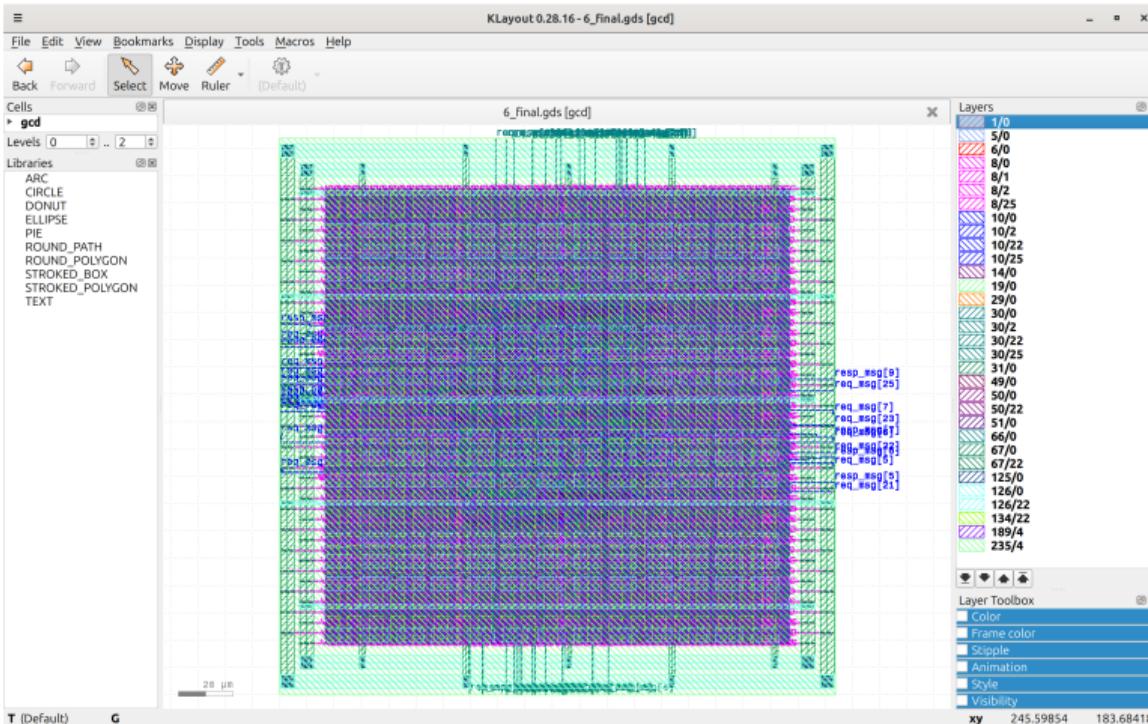


Figure 2: Load GDS

3. Use a LYP file

- Recall: What are layer properties and the .lyp file?
- Search the sg13g2 lyp-file from the PDK and enable it in Klayout.
- Make it the default lyp in Klayout.
- Look for changes in the layer list and the GDS.
- What has changed?



3. Use a LYP file

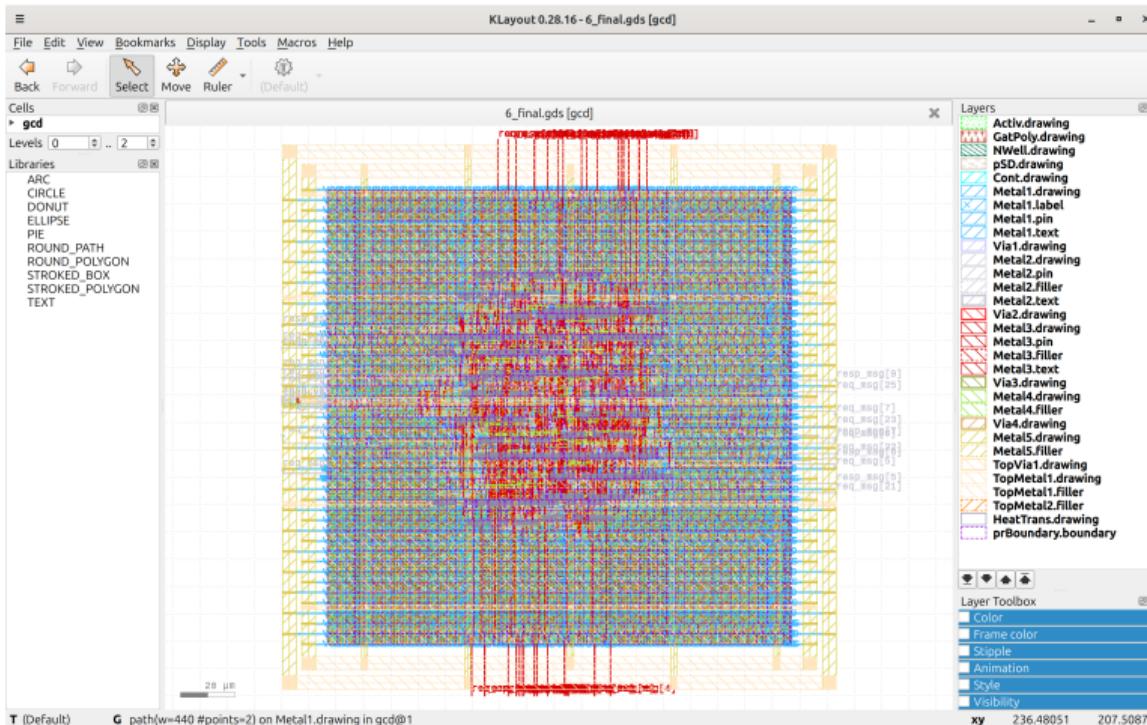


Figure 3: Use LYP file

4. Navigate Layers and GDS

- In the layer list:
 - Enable “Hide empty layers”.
 - Enable “Visibility follows selection”.
 - De-/Select multiple layers and see the changes.
- Zoom out to see the complete GDS and de-/select layers.
- Zoom into details and de-/select layers.
- Move the GDS file while zoomed in.
- Try to find interesting views.
- Discuss with neighbours what might be of interest.



4. Navigate Layers and GDS

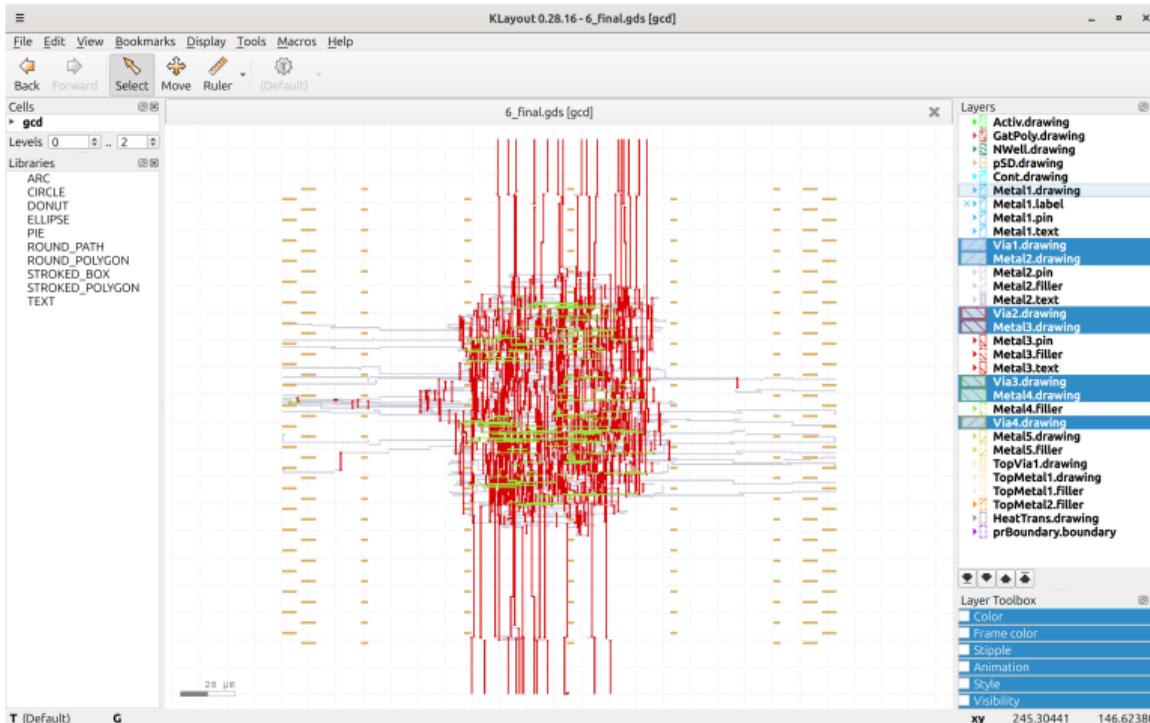


Figure 4: GDS zoomed out

4. Navigate Layers and GDS

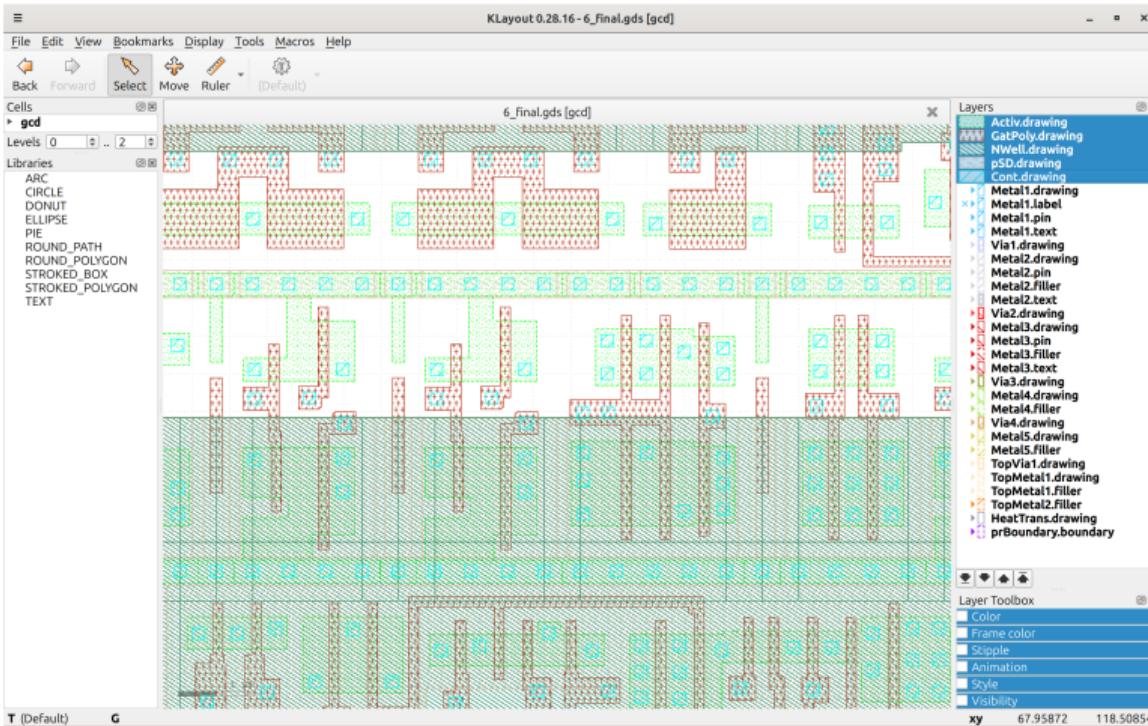


Figure 5: GDS zoomed in

5. Load cell library GDS

- Now open another GDS: the standard cell library from the sg13g2 PDK.
- The standard cell library is a single GDS file. Search and load the library file.



5. Load cell library GDS

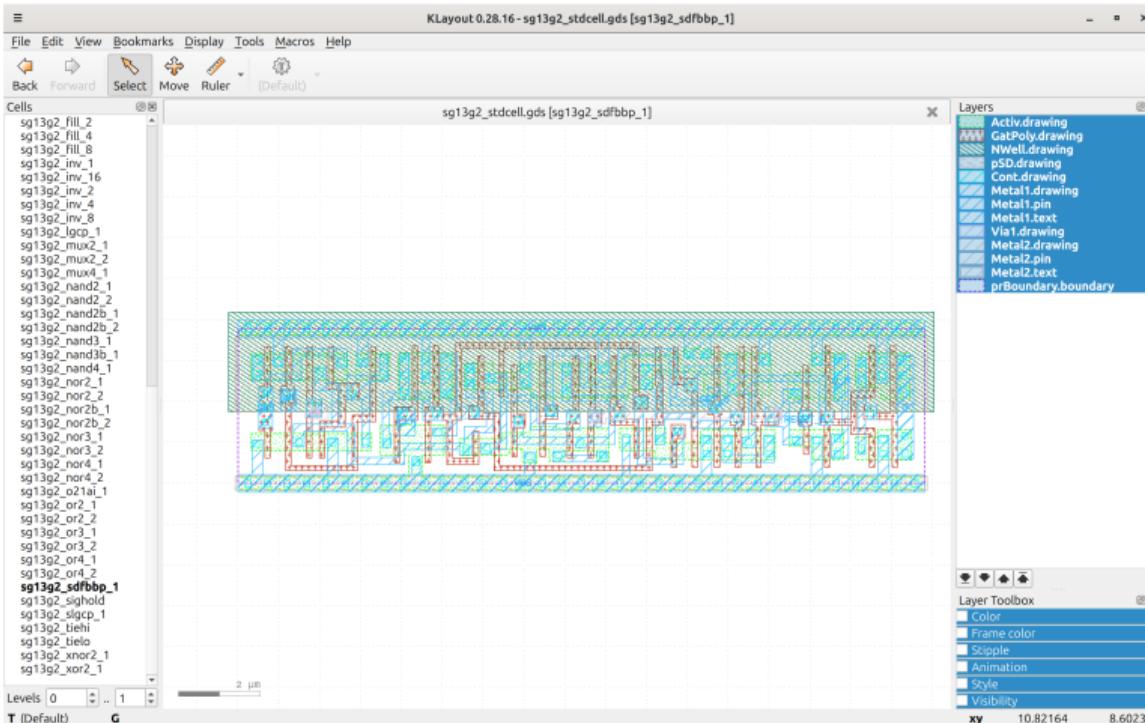


Figure 6: Cell library

6. Pick a cell

- All the cell names are displayed in a window to the left. One cell is selected (displayed in bold). The GDS of this cell is viewed. Try to bring another cell “to the top” to see that cells GDS.
- Repeat the layer navigation like in the eaxmple GDS.



6. Pick a cell

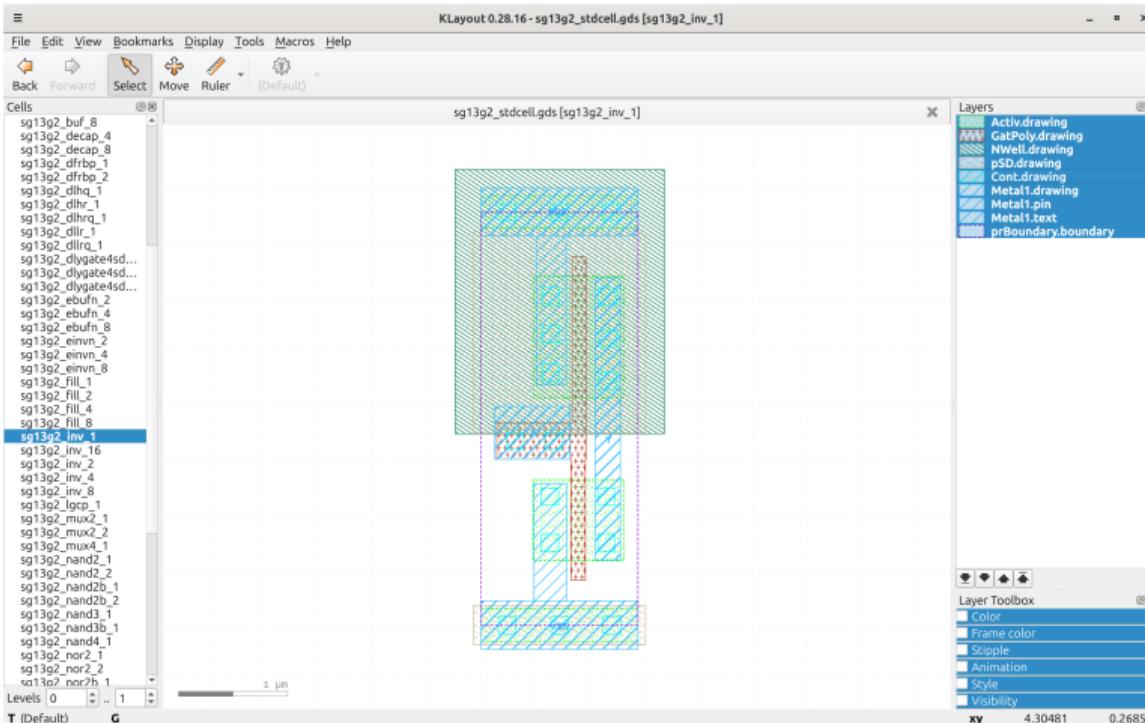


Figure 7: Single cell

Chapter 6 - OpenROAD GUI

Course authors (Git file)



1 Analysing designs with the GUI



Section 1

Analysing designs with the GUI



View layout

Viewing Layout Results

The `make gui_final` command target successively reads and loads the technology `.odb` files and the parasitics and invokes the GUI in these steps:

- Reads and loads `.odb` files.
- Loads `.spf` (parasitics).

The figure below shows the post-routed DEF for the `ibex` design.

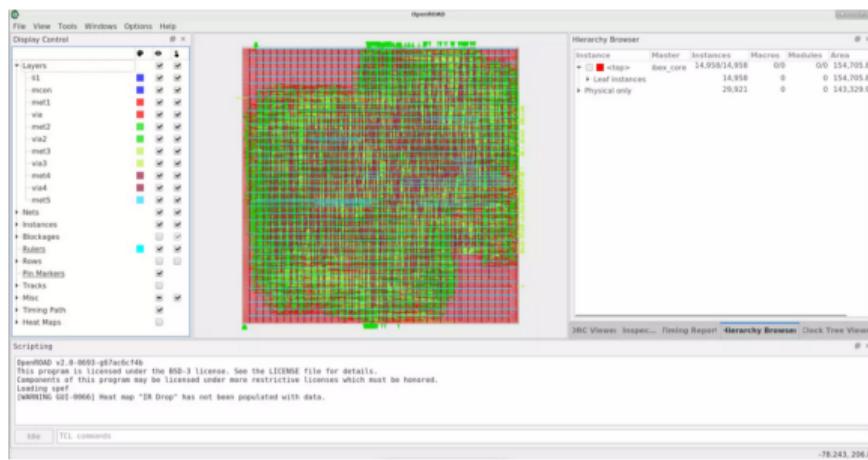


Figure 1: View layout

View objects

Visualizing Design Objects And Connectivity

Note the **Display Control** window on the LHS that shows buttons for color, visibility and selection options for various design objects: Layers, Nets, Instances, Blockages, Heatmaps, etc.

The Inspector window on the RHS allows users to inspect details of selected design objects and the timing report.

Try selectively displaying (show/hide) various design objects through the display control window and observing their impact on the display.

Figure 2: View objects



Clock tree

Tracing The Clock Tree

View the synthesized clock tree for `ibex` design:

- From the top Toolbar Click `Windows` -> `Clock Tree Viewer`

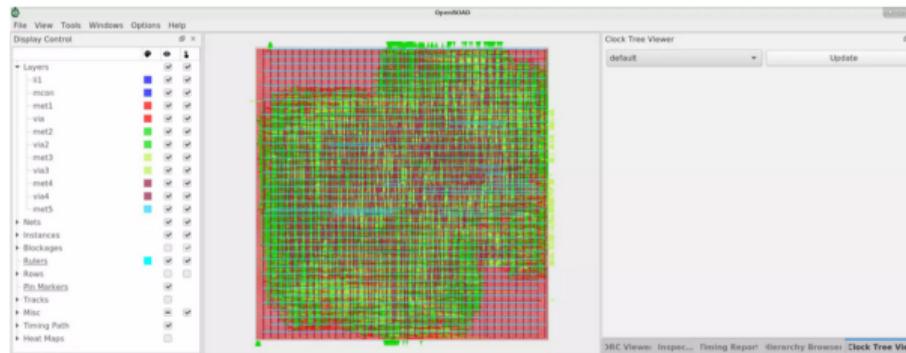


Figure 3: Clock tree



Heat maps

Using Heat Maps

From the Menu Bar, Click on **Tools** -> **Heat Maps** -> **Placement Density** to view congestion selectively on vertical and horizontal layers.

Expand **Heat Maps** -> **Placement Density** from the Display Control window available on LHS of OpenROAD GUI.

View congestion on all layers between 50-100%:

In the **Placement density** setup pop-up window, Select **Minimum** -> **50.00%** **Maximum** -> **100.00%**

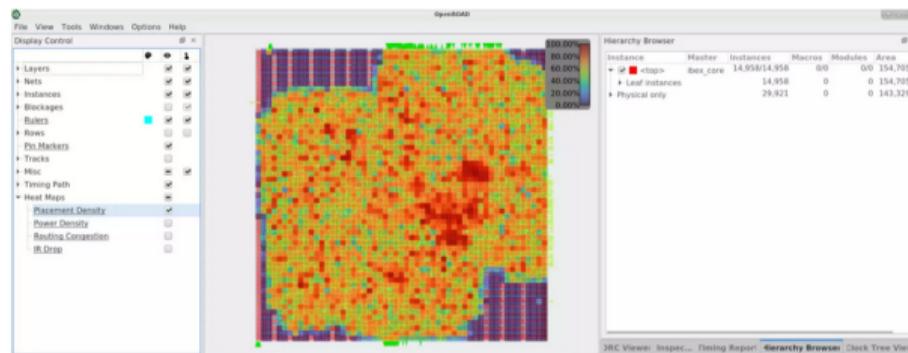


Figure 4: Heat maps

Timing report

Viewing Timing Report

Click **Timing** -> **Options** to view and traverse specific timing paths. From Toolbar, click on the **Timing** icon, View **Timing Report** window added at the right side (RHS) of GUI as shown below.

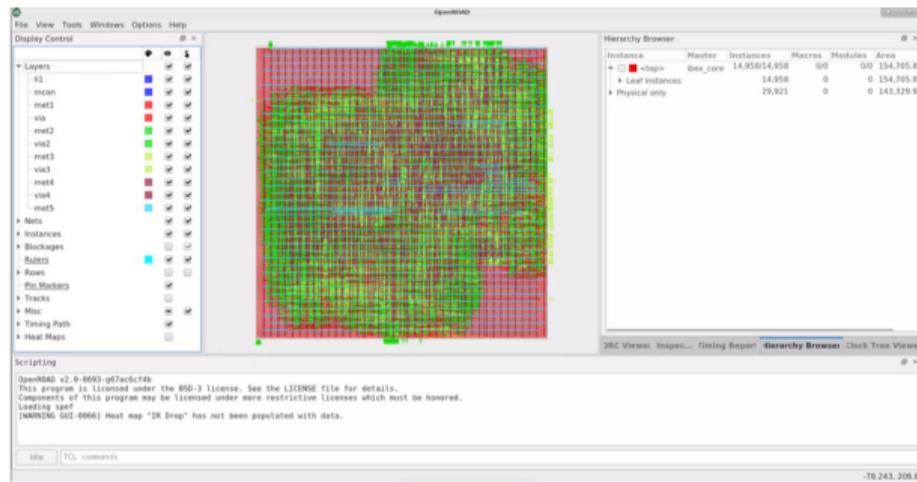


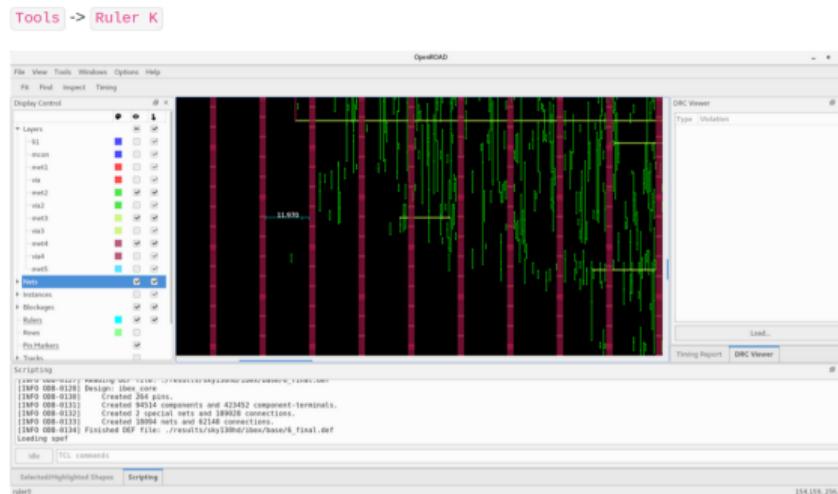
Figure 5: Timing report

Rulers (measure distances)

Using Rulers

A ruler can measure the distance between any two objects in the design or metal layer length and width to be measured, etc.

Example of how to measure the distance between VDD and VSS power grid click on:



Distance between VDD and VSS layer is **11.978**

Figure 6: Rulers

TCL command interface

Tcl Command Interface

Execute OpenROAD-flow-scripts Tcl commands from the GUI. Type `help` to view Tcl Commands available. In OpenROAD GUI, at the bottom, `TCL commands` executable space is available to run the commands. For example

View `design area` in the `Tcl Commands` section of the GUI:

```
report_design_area
```

Try the below timing report commands to view timing results interactively:

```
report_wns  
report_tns  
report_worst_slack
```

Figure 7: TCL interface



Area report

Area

View design area and its core utilization:

```
make gui_final
```

In the [Tcl Commands](#) section:

```
report_design_area
```

View the resulting area as:

```
Design area 191262 u^2 30% utilization.
```

Figure 8: Area report



Timing slack report

Timing

Users can view flow results using the command interface from the shell or the OpenROAD GUI to visualize further and debug. Learn more about the [GUI](#).

```
make gui_final
```

Use the following commands in the [Tcl Commands](#) section of GUI:

```
report_worst_slack  
report_tns  
report_wns
```

Note the worst slack, total negative slack and worst negative slack:

```
worst slack -0.99  
tns -1.29  
wns -0.99
```

Figure 9: Timing slack report



Power report

Power

Use the report command to view individual power components i.e. sequential, combinational, macro and power consumed by I/O pads.

In the [Tcl Commands](#) section:

```
report_power
```

The power output is as follows:

Group	Internal Power	Switching Power	Leakage Power	Total Power
Sequential	5.58e-03	6.12e-04	1.67e-08	6.19e-03 19.0%
Combinational	9.23e-03	1.71e-02	4.90e-08	2.63e-02 81.0%
Macro	0.00e+00	0.00e+00	0.00e+00	0.00e+00 0.0%
Pad	0.00e+00	0.00e+00	0.00e+00	0.00e+00 0.0%
Total	1.48e-02 45.6%	1.77e-02 54.4%	6.57e-08 0.0%	3.25e-02 100.0%

Figure 10: Power report



Chapter 6 -
OpenROAD GUI
- QUESTIONS

Course authors
(Git file)

Chapter 6 - OpenROAD GUI - QUESTIONS

Course authors (Git file)



Chapter 6 -
OpenROAD GUI
- QUESTIONS

Course authors
(Git file)

Recap questions for Chapter 6



Questions

Chapter 6 -
OpenROAD GUI
- QUESTIONS

Course authors
(Git file)

- Is the OR-GUI a HDL code editor?
- Is the OR-GUI a full RTL-2-GDS graphic tool?
- What is OR-GUI instead?
- If you have a ready-build project (GDS), what can you do with it in OR-GUI?
- Explain heatmaps
- How to make better chip designs with the analysis in OR-GUI?
- What is the TCL console in the GUI?

Chapter 6 - OpenROAD GUI - TRAINING - Common

Course authors (Git file)



1

Guess and rebuild

Challenge 1

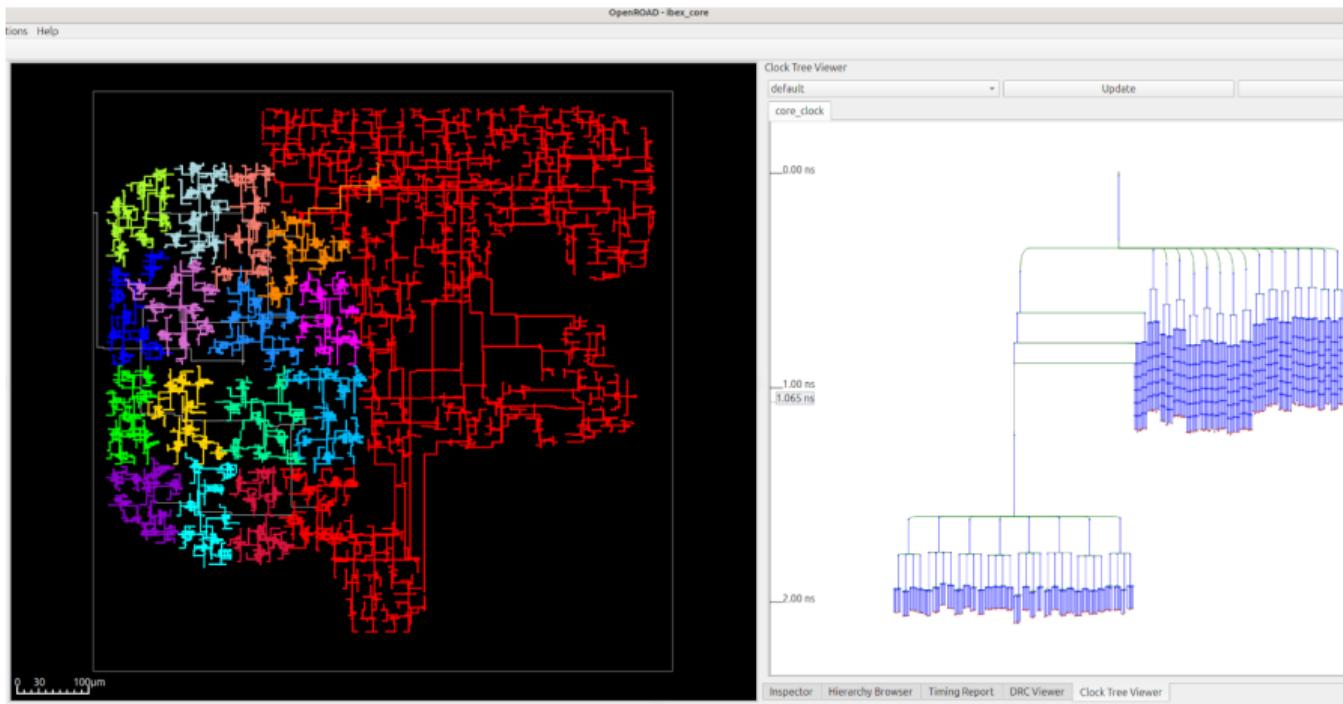


Figure 1: What is this?

Challenge 2

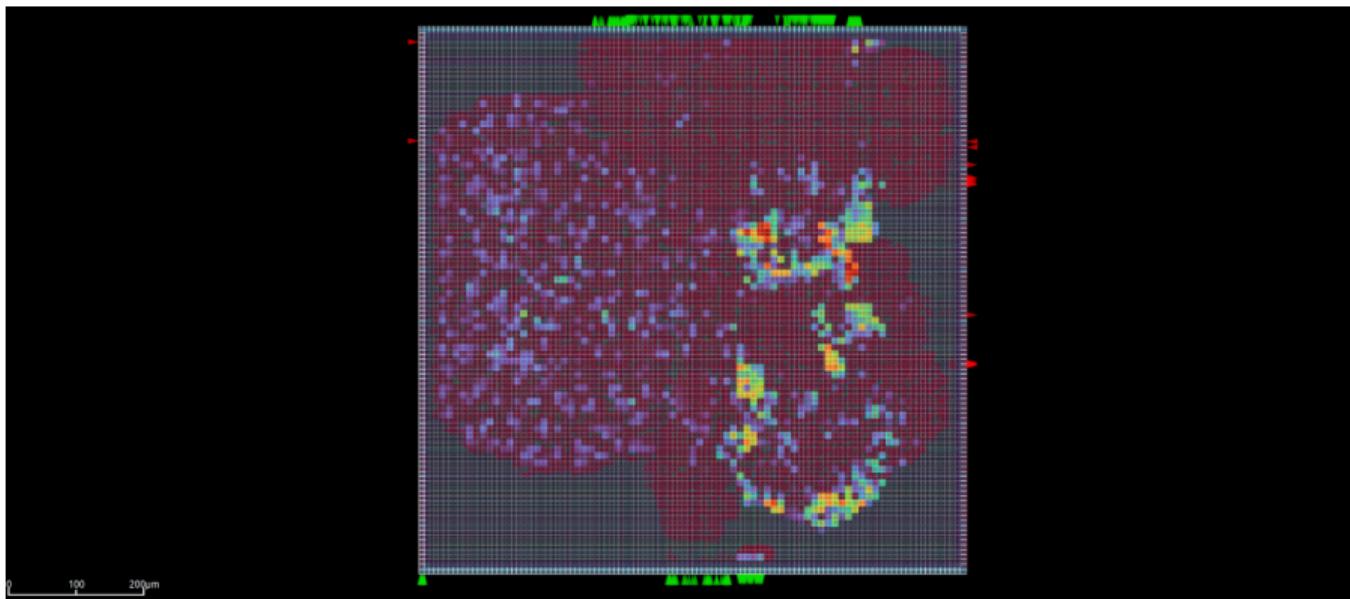


Figure 2: What is this?

Rebuild the picture with any design!

Challenge 3

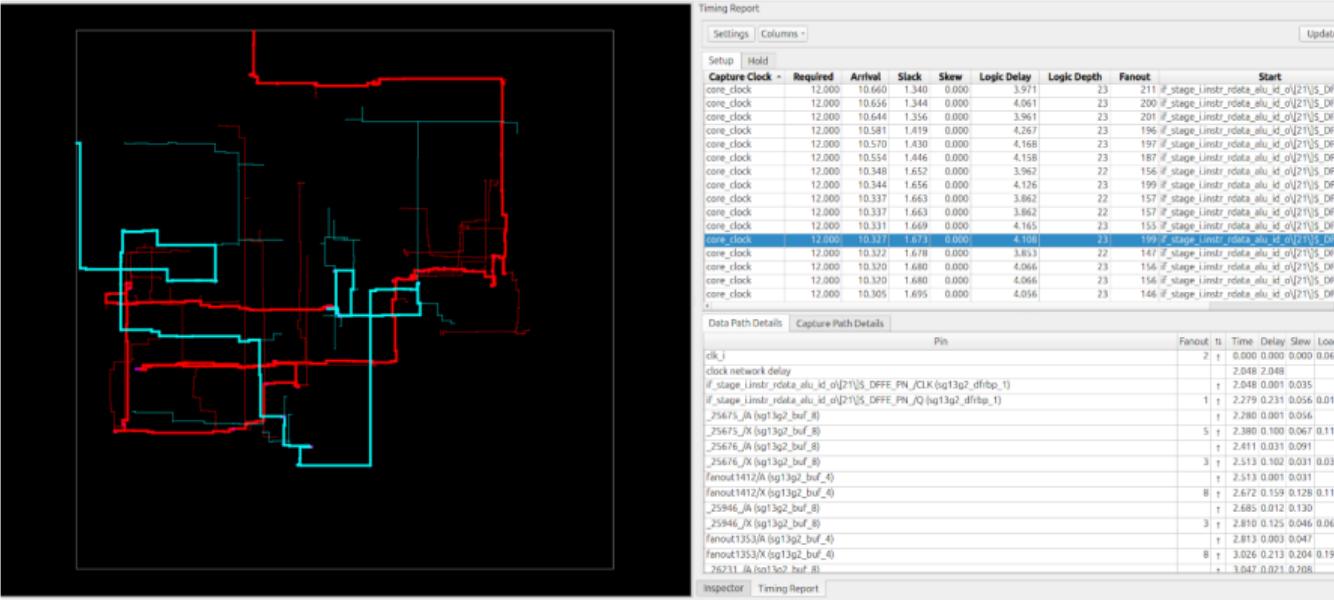


Figure 3: What is this?

Rebuild the picture with any design!

Challenge 4

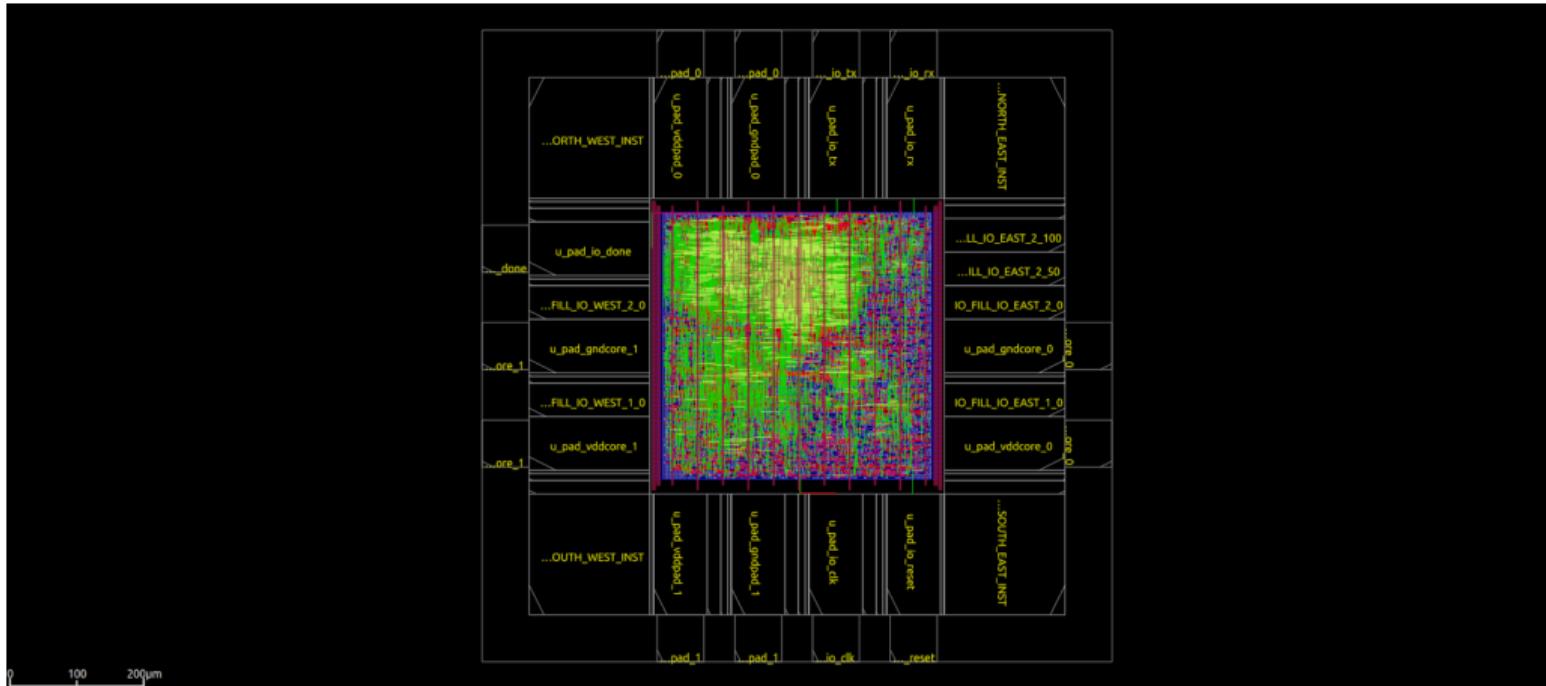


Figure 4: What is this?

Solutions (of the challenges)

- ① Ibex clocktree without most other layers (IHP PDK)
- ② Ibex power density >15 uW (IHP PDK)
- ③ Ibex single clock path with all layers off, but the timing paths (IHP PDK)
- ④ Masked aes without most metals and only pins names (IHP PDK)



Chapter 7 - OpenROAD flow scripts

Course authors (Git file)



1 Introduction

2 ORFS Tutorial

3 Multiple runs

4 Structure of flow directories

5 TCL Console and commands

6 Reports

7 Logs



Section 1

Introduction



Introduction

What happened on the way to here:

- GDS-2-RTL: OpenROAD
- OpenROAD flow scripts (ORFS) overview
- ORFS flow steps and flow components
- First run of the flow scripts
- A Dive into the PDK (Klayout)
- Analysing: Heatmaps and more (ORFS GUI)

NOW:

- One day of using ORFS
- Getting a hands on with important data and features.



Section 2

ORFS Tutorial



ORFS Tutorial

There is a good tutorial about ORFS in the official documentation:

<https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html>

The ORFS online-tutorial was not written for the use with the IHP PDK especially, but we can adopt this easily.





Contents

Introduction

User Guidelines

Getting Started

Configuring The Design

Running The Automated RTL-to-GDS Flow

Viewing Results And Logs

OpenROAD GUI

Understanding and Analyzing

OpenROAD Flow Stages and Results

Troubleshooting Problems

OpenROAD Flow Scripts Tutorial

Introduction

This document describes a tutorial to run the complete OpenROAD flow from RTL-to-GDS using [OpenROAD Flow Scripts](#). It includes examples of useful design and manual usage in key flow stages to help users gain a good understanding of the [OpenROAD](#) application flow, data organization, GUI and commands.

This is intended for:

- Beginners or new users with some understanding of basic VLSI design flow. Users will learn the basics of installation to use OpenROAD-flow-scripts for the complete RTL-to-GDS flow from here.
- Users already familiar with the OpenROAD application and flow but would like to learn more about specific features and commands.

Figure 1: ORFS Online Tutorial

Section 3

Multiple runs



Caveats of multiple runs in ORFS

- ORFS does not handle multiple runs for a single design.
- The design run must be cleared with `make clean_all`, before a new runs can be started.
- !!! The previous data from the previous run will be lost.

Side feature:

- A run can start over where you left it.



Workaround for saving the design data

- ① After a design run:
 - Rename the results directory to something different:
 - flow/results/ihp-sg13g2/designname/base
 - flow/results/ihp-sg13g2/designname/base_old_1
 - Rename the reports directory to something different:
 - flow/reports/ihp-sg13g2/designname/base
 - flow/reports/ihp-sg13g2/designname/base_old_1
- ② With the start of the next run (make):
 - The original directory gets created again.
- ③ Repeat that before every new run of the same design.



Reviewing the older design data

- The command `make gui_final` only works on the enabled design (Makefile, DESIGN_CONFIG)
- To load an older design from a renamed folder, run `openroad -gui`
- This opens an empty GUI and you can load a GDS into it.
- This can be done multiple times in parallel.



Section 4

Structure of flow directories



Structure of flow directories

Inside the flow directory:

```
1 flow$ ls
2
3 Makefile      platforms  test
4 designs       reports    tutorials
5 results        util      logs
6 objects       scripts
```

- **Makefile:** Runs the RTL-2-GDS toolchain with a design
- **platforms:** Technology nodes and PDKs
- **designs:** Source and configuration files of the designs
- **reports:** Generated report files from the design runs
- **results:** Generated result files from the design runs



Section 5

TCL Console and commands



TCL Console and commands

At the bottom of the OpenROAD GUI is the TCL command console.

Type `help` into the console to get a list of the available commands and their syntax.

Some commands that were already used in this course:

- `save_image`
- `report_design_area`
- `report_power`
- `report_worst_slack`



Section 6

Reports



Reports

- Reports get generated for each design run.
- The reports are stored in the reports directory.
- These are the report files for the gcd example:

```
1 reports/ihp-sg13g2/gcd/base$ ls
2
3 2_floorplan_final.rpt  6_finish.rpt      final_resizer.webp
4 3_detailed_place.rpt   congestion.rpt    final_routing.webp
5 3_resizer.rpt          cts_core_clock.webp grt_antennas.log
6 4_cts_final.rpt        drt_antennas.log  synth_check.txt
7 5_global_place.rpt    final_clocks.webp  synth_stat.txt
8 5_global_route.rpt    final_ir_drop.webp VDD.rpt
9 5_route_drc.rpt       final_placement.webp VSS.rpt
```



Section 7

Logs



Logs

- Logs get generated for each design run.
- The logs are stored in the logs directory.
- These are the log files for the gcd example:

```
1 logs/ihp-sg13g2/gcd/base$ ls
2
3 1_1_yosys_canonicalize.log  2_6_floorplan_pdn.json      4_1_cts.log
4 1_1_yosys_hier_report.log   2_6_floorplan_pdn.log       5_1_grt.json
5 1_1_yosys.log               3_1_place_gp_skip_io.json  5_1_grt.log
6 2_1_floorplan.json          3_1_place_gp_skip_io.log   5_2_route.json
7 2_1_floorplan.log           3_2_place_iop.json        5_2_route.log
8 2_2_floorplan_io.json       3_2_place_iop.log         5_3_fillcell.json
9 2_2_floorplan_io.log        3_3_place_gp.json        5_3_fillcell.log
10 2_3_floorplan_tdms.json    3_3_place_gp.log         6_1_fill.json
11 2_3_floorplan_tdms.log     3_4_place_resized.json   6_1_fill.log
12 2_4_floorplan_macro.json   3_4_place_resized.log    6_1_merge.log
13 2_4_floorplan_macro.log    3_5_place_dp.json        6_report.json
14 2_5_floorplan_tapcell.json 3_5_place_dp.log        6_report.log
15 2_5_floorplan_tapcell.log  4_1_cts.json
```

Section 8

Results



Results

- Results (mostly odb, GDS) get generated for each design run.
- The results are stored in the results directory.
- These are the result files for the gcd example:

```
1 | flow /results /ihp-sg13g2/gcd/base$ ls
2 |
3 | 1_1_yosys.v          3_3_place_gp.odb      6_1_fill.sdc
4 | 1_synth.rtlil        3_4_place_resized.odb  6_1_fill.v
5 | 1_synth.sdc          3_5_place_dp.odb      6_1_merged.gds
6 | 1_synth.v            3_place.odb          6_final.def
7 | 2_1_floorplan.odb    3_place.sdc          6_final.gds
8 | 2_2_floorplan_io.odb 4_1_cts.odb          6_final.odb
9 | 2_3_floorplan_tdms.odb 4_cts.odb          6_final.sdc
10 | 2_4_floorplan_macro.odb 4_cts.sdc          6_final.spf
11 | 2_5_floorplan_tapcell.odb 5_1_grt.odb      6_final.v
12 | 2_6_floorplan_pdn.odb  5_2_route.odb      clock_period.txt
13 | 2_floorplan.odb       5_3_fillcell.odb    keep_hierarchy.tcl
14 | 2_floorplan.sdc       5_route.odb          mem.json
15 | 3_1_place_gp_skip_io.odb 5_route.sdc      route.guide
16 | 3_2_place_iop.odb     6_1_fill.odb          updated_clks.sdc
```



Section 9

Basic design initialization



Design configuration (config.mk)

config.mk from the ibex example:

<https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/blob/master/flow/designs/sky130hd/ibex/config.mk>

Tutorial about the design configuration;

<https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#design-configuration>



Variable Name	Description
PLATFORM	Specifies Process design kit.
DESIGN_NAME	The name of the top-level module of the design
VERILOG_FILES	The path to the design Verilog files or JSON files providing a description of modules (check <code>yosys -h write_json</code> for more details).
SDC_FILE	The path to design <code>.sdc</code> file
CORE_UTILIZATION	The core utilization percentage.
PLACE_DENSITY	The desired placement density of cells. It reflects how spread the cells would be on the core area. 1 = closely dense. 0 = widely spread

Figure 2: Design config variables



Clock constraints (constraints.sdc)

constraints.sdc from the ibex example:

```
1 current_design ibex_core
2
3 set clk_name core_clock
4 set clk_port_name clk_i
5 set clk_period 10.0
6 set clk_io_pct 0.2
7
8 set clk_port [get_ports $clk_port_name]
9
10 create_clock -name $clk_name -period $clk_period $clk_port
11
12 set non_clock_inputs [lsearch -inline -all -not -exact [all_inputs] $clk_port]
13
14 set_input_delay [expr $clk_period * $clk_io_pct] -clock $clk_name $non_clock_inputs
15 set_output_delay [expr $clk_period * $clk_io_pct] -clock $clk_name [all_outputs]
```

<https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#timing-constraints>



Design Verilog input

<https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#design-input-verilog>

These are the Verilog files of the ibex design example:

```
1 flow/designs/src/ibex$ ls
2
3 ibex_alu.v           ibex_ex_block.v      ibex_register_file_ff.v    prim_ram_1p.v
4 ibex_branch_predict.v ibex_fetch_fifo.v   ibex_register_file_fpga.v prim_secded_28_22_dec.v
5 ibex_compressed_decoder.v ibex_icache.v     ibex_register_file_latch.v prim_secded_28_22_enc.v
6 ibex_controller.v     ibex_id_stage.v     ibex_wb_stage.v          prim_secded_39_32_dec.v
7 ibex_core.v          ibex_if_stage.v     LICENSE                  prim_secded_39_32_enc.v
8 ibex_counter.v        ibex_load_store_unit.v prim_badbit_ram_1p.v    prim_secded_72_64_dec.v
9 ibex_cs_registers.v  ibex_multdiv_fast.v  prim_clock_gating.v     prim_secded_72_64_enc.v
10 ibex_csr.v           ibex_multdiv_slow.v prim_generic_clock_gating.v prim_xilinx_clock_gating.v
11 ibex_decoder.v       ibex_pmp.v          prim_generic_ram_1p.v   README.md
12 ibex_dummy_instr.v   ibex_prefetch_buffer.v prim_lfsr.v
```



Section 10

Design tweaking



Design tweaking

- OpenROAD is build on many different tools
- It does not feel consistent to configure the tools.
- To find and understand the possibilities of improving a design via tweaking one must read the documentation of the tools.
- It might take some time to become comfortable with tweaking.
- Don't give up!

In the following we present

- some easy tweaking possibilities to start with



Synthesis AREA or SPEED

<https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#area-and-timing-optimization>

In a nutshell:

- Set ABC_SPEED=1 or ABC_AREA=1 in the config.mk
- Rerun.



DIE_AREA and CORE_AREA

- Set DIE_AREA and CORE_AREA in the config.mk
- Rerun

This is an example for the two variables, taken from the config.mk in the masked_aes example earlier. The comments contain a list of added spaces around the core area.

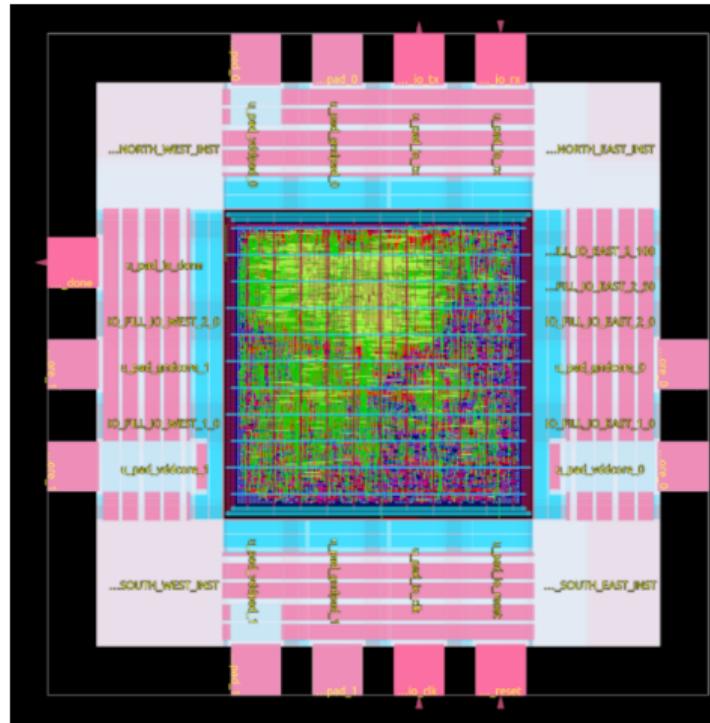
masked_aes config.mk:

```
1 # (Sealring: roughly 60um)
2 # I/O pads: 180um
3 # Bondpads: 70um
4 # Margin for core power ring: 20um
5 # Total margin to core area: 270um
6 export DIE_AREA = 0 0 940 940
7 export CORE_AREA = 270 270 670 670
```



masked_aes areas

The area calculations from the masked_aes config.mk in a GDS:



Density

In a nutshell:

- Change the PLACE_DENSITY value in the config
- Value between 0.2 and 0.95
- Rerun

```
export PLACE_DENSITY ?= 0.88
```



CORE_UTILIZATION

In a nutshell:

- Change the CORE_UTILIZATION value in the config
- Value between 20 and 80
- Rerun

```
export CORE_UTILIZATION = 45
```



Example for Utilization and Density with the ibex design

In the ORFS tutorial is a tweak example with these two variables:

<https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#defineing-placement-density>

Read how this should change the GDS.



Further reading on the topic

- The chapters in the ORFS tutorial starting here:

<https://openroad-flow-scripts.readthedocs.io/en/latest/tutorials/FlowTutorial.html#understanding-and-analyzing-openroad-flow-stages-and-results>

- Synthesis Explorations
- Floorplanning
- Power Planning And Analysis
- Macro or Standard Cell Placement
- Timing Optimizations
- Clock Tree Synthesis
- ...



Section 11

Finishing a design



Footprint for IOPads

- A TCL script is needed to arrange the IOPads around the core design area.
- This TCL script must be referenced in the config.mk:

```
1 | export FOOTPRINT_TCL = $(DESIGN_HOME)/$(PLATFORM)/$(DESIGN_NICKNAME)/footprint.tcl
```

A working example of such a footprint.tcl can be found inside the masked_aes example:

masked_aes footprint.tcl:

<https://github.com/HEP-Alliance/masked-aes-tapeout/blob/main/footprint.tcl>



Sealring

- A sealring GDS must be generated and merged with the design GDS.
- Information about how to create a sealring is available as an example in the masked_aes README:

Sealring

The sealring was generated using a script included with IHP's open PDK.

Clone the PDK and set up the technology in KLayout. The following command creates the sealring:

```
$ klayout -n sg13g2 -zz -r <IHP-repo-root>/ihp-sg13g2/libs.tech/klayout/tech/scripts/sealring.j
```

The generated sealring has to be moved by -60 in both directions, which can be done in KLayout.

Figure 4: Sealring Information masked_aes



<https://github.com/HEP-Alliance/masked-aes-tapeout/tree/main?tab=readme-ov-file#sealring>



Metal fill

- Ongoing issue discussion about the Metall fill:

<https://github.com/IHP-GmbH/IHP-Open-PDK/pull/229>

It is solved and merged to the repo, but the issue is kept open for enhancement reasons.

Metal Fill

Metal fill has to be performed on the output GDS using a KLayout script provided as part of the IHP PDK. The script is currently work-in-progress here: [IHP-GmbH/IHP-Open-PDK#229](#)

Figure 5: Metal fill information masked_aes





Chapter 7 -
OpenROAD
flow scripts -
QUESTIONS

Course authors
(Git file)

Chapter 7 - OpenROAD flow scripts - QUESTIONS

Course authors (Git file)



Chapter 7 -
OpenROAD
flow scripts -
QUESTIONS

Course authors
(Git file)

Recap questions for Chapter 7



Questions

Chapter 7 -
OpenROAD
flow scripts -
QUESTIONS

Course authors
(Git file)

Questions about:

- Power
- Performance
- Area
- How to change?
- Configuration of a design:
 - pdk specific configs
 - design specific configs
- Structure of a design project in ORFS?

Chapter 7 - OpenROAD flow scripts - TRAINING - Bonus

Course authors (Git file)



Show what you found out

Task: Present today at 3pm

- Create 1-3 pictures of something that interests you
- It should be related to the course
- Present those pictures with a few words
- Start is 15:00 (3 pm).



Ideas

- Compare two runs of the same design with different GDS
- Show a 2.5D rendering in fancy colors
- Find the most complex cell in the PDK in KLayout



How to

Task: The pictures

- Create a directory pics in your home directory
- Save your pictures in pics
- The names of the pictures must be:
 - A.png
 - B.png
 - C.png
- Have them ready at 15:00 (3 pm)



Chapter 7 - OpenROAD flow scripts - TRAINING - Common

Course authors (Git file)



- 1 Save the ready examples
- 2 Multiple runs
- 3 Reports
- 4 LFSR: parameterized version
- 5 LFSR (param version) optimize
- 6 LFSR IOPads
- 7 gcd | LFSR | TT: Area change

Save the ready examples

- Some design runs were done in the previous chapters.

Task: Rename them

- To a backup save of all these designs by renaming their directories.
- How to rename them is described in C7-S3.
- Try to find meaningful renames:
 - date, time
 - versioning
 - optimization param



Multiple runs

- The lecture slides of C7-S3 explain how to do multiple runs (one after another) on the same design.

Task: Two runs - Same design

- Make two runs with a different DENSITY for the gdc design
- Compare the runs via their final GDS
- Save both GDS as png images



Reports

The reports for power and area are needed for the upcoming training sessions.

Task: Read reports

- Look up for the TCL commands for different reports, at least power and area.
- Try the commands in the GUI TCL console with any already made design.



LFSR: parameterized version

Task: LFSR width as parameter

- Rewrite the LFSR Verilog with a parameter for the register width
- Maybe parameterize the index of the feedback bits.
- Run to get a GDS of this.



LFSR (param version) optimize

Optimize the LFSR param version in three different ways.

Task: Min clock period

- Fix the length (make it long!)
- Find the minimal clockperiod for your design.

Task: Max length

- Fix the area
- Find the maximum length for the LFSR

Task: Max density

- Fix the length and area
- Find the max density

LFSR IOPads

The LFSR has only very few inputs and outputs. This might be the most simple design for starting with to create a Padring with IOPads.

Task: footprint.tcl

- Take the floorplan.tcl from masked_aes as an example
- Create a floorplan.tcl for the LFSR
- Run to a GDS



gcd | LFSR | TT: Area change

For this Task, use one (or more, one after another) of these designs:

- gcd
- LFSR
- TinyTapeout
- Some other own design

Task: Compare two areas

- Keep the density fixed (Not too high, more relaxed)
- Change the core area and rerun
- Compare the GDS of two different versions



gcd | LFSR | TT: Free area

For this Task, use one (or more, one after another) of these designs:

- gcd
- LFSR
- TinyTapeout
- Some other own design

Task: Free area, changing the density

- No constraint for an area. (Comment it out)
- Experiment with a wide range of density.
- How high/low can the density be?
- Does the power estimation change much?

gcd | LFSR | TT: Uniform distributed heatmap

For this Task, use one (or more, one after another) of these designs:

- gcd
- LFSR
- TinyTapeout
- Some other own design

Task: heatmap optimization

- Choose one of the heatmaps in GUI
- Make a suggestion of how to find a most uniform distributed heatmap design.
- Try to build a GDS of that.
- Compare to the “not optimized” version. Did you achieve your goal?

Chapter 8 - Tapeout

Course authors (Git file)



1 Open-source Tapeout informations from IHP

2 READMEs

3 Read-the-docs

4 The last steps to a successful Tapeout

5 Contribute to open-source



Section 1

Open-source Tapeout informations from IHP



Open-source Tapeout informations from IHP

This chapter provides information about the Tapeout at IHP. Please follow the links to tag along with the chapter.



IHP - Repositories structure

- The designs for the open-source Tapeouts are collected as the “Open DesignLib”.
- During the process of each Tapeout there are dedicated repositories by the names TO_monthyear.
- After Tapeout these designs shall be merged into the Open DesignLib repository.



PDK and Open DesignLib

- Open.source PDK:

<https://github.com/IHP-GmbH/IHP-Open-PDK>

- Open DesignLib:

<https://github.com/IHP-GmbH/IHP-Open-DesignLib>



TapeOut repositories (TO)

- Tapeout May 2024:

https://github.com/IHP-GmbH/TO_May2024

- Tapeout Nov 2024:

https://github.com/IHP-GmbH/TO_Nov2024

- Tapeout Dec 2024:

https://github.com/IHP-GmbH/TO_Dec2024



Section 2

READMEs



README file in TO repositories

Important notes:

- Always look for the most actual and stable information
- Tools and workflows change fast
- Design rules don't change fast (if ever)
- Most parts of the PDK are long term stable



Submission process

Screenshot from the [README TO_DEC2024](#):

Submission process

To submit for our OpenMPW run you have to have a valid github account. Make a fork of this repository and then create a separate directory for your design next to the `ExampleDesign` (you can also make a copy and rename it). Structure your data according to our recommendations, update the documentation and push your files to your fork, then make a pull request.

ⓘ Caution

On each PR a github action will be triggered to run a minimal DRC precheck (rejection test). Please consider it and do not upload many `gds` files.

Once you make a PR a github action will run a minimum set of DRC checks on each `gds` and `gds.zip` file. If the test passes it means that your design is manufacturable at our pilot line not ensuring the reliability. An example of a failure is shown on the following figure

Figure 1: Submission process



Physical constraints

Screenshot from the [README TO_DEC2024](#):

Physical design constraint

1. Please align with the layout design rules which can be found [here](#)
2. The area granted to a community member is 2 mm^2 It includes the sealring.
3. The sealring can be found among KLayout PyCells.

Figure 2: Physical design constraints

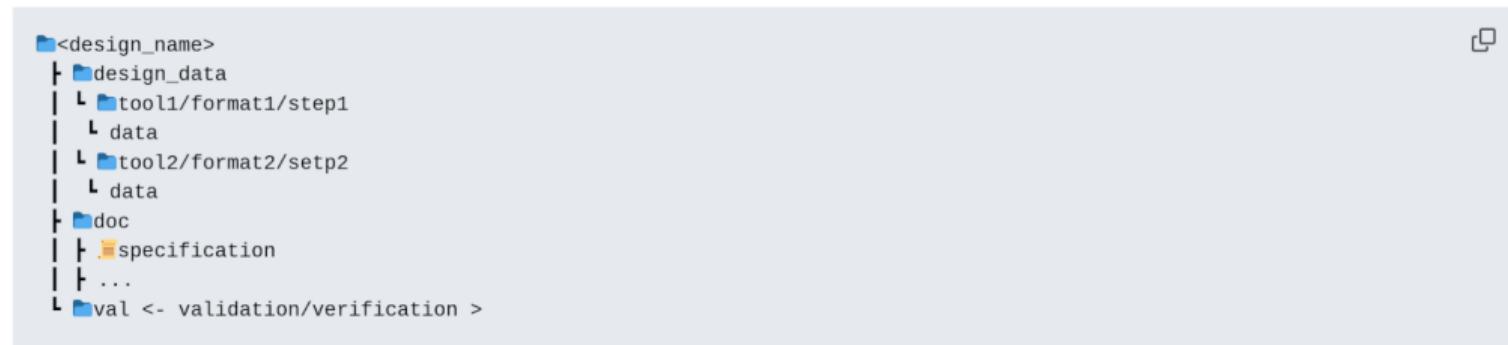


Project structure

Screenshot from the [README TO _DEC2024](#):

Directory structure

If you are a designer, we propose the following directory structure, which we and the community would appreciate you using. Please ensure that the design you submit is reproducible, meaning it should include all the information necessary to replicate the design.



```
<design_name>
├── design_data
│   ├── tool1/format1/step1
│   │   └── data
│   ├── tool2/format2/step2
│   │   └── data
├── doc
│   ├── specification
│   └── ...
└── val <- validation/validation >
```

The first segmentation separates the `design data` from a `documentation` and `verification/validation data`.

Figure 3: Directories in a project

Design data structure

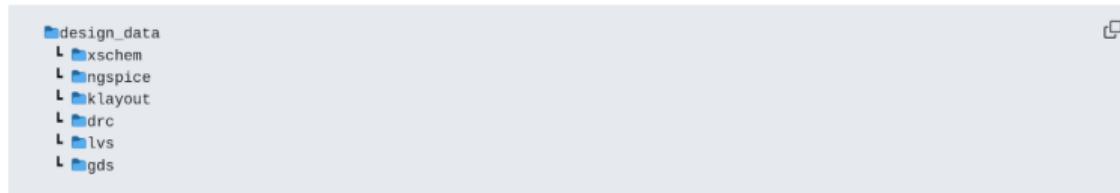
Screenshot from the **README TO _DEC2024:**

Design data directory structure

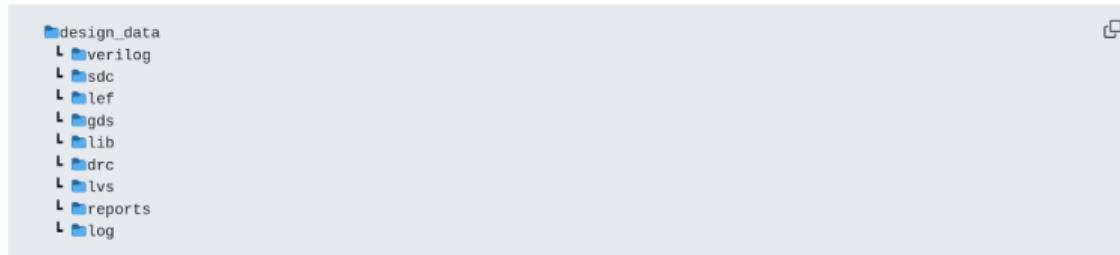
The `design_data` should be structured using tool/format/step specific scheme.

Here you can find some examples:

Example1



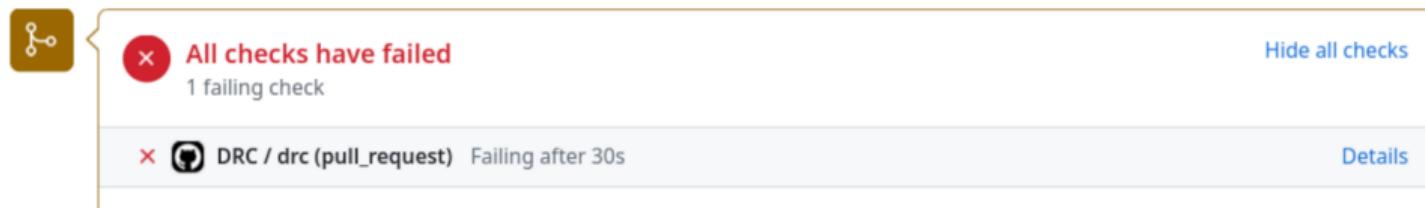
Example2



DRC

During a submission to a Tapeout the design gets checked with a DRC via Github Actions.

Once you make a PR a github action will run a minimum set of DRC checks on each `gds` and `gds.zip` file. If the test passes it means that your design is manufacturable at our pilot line not ensuring the reliability. An example of a failure is shown on the following figure



The detailed report can be downloaded from a link, which can be found at the end of the section `Details->Archive DRC Results` as show on the image:



Figure 5: DRC via Github actions

Section 3

Read-the-docs



Read-the-docs

- Open PDK Docu:

<https://ihp-open-pdk-docs.readthedocs.io/en/latest/index.html>

- Open DesignLib Docu:

<https://ihp-open-ip.readthedocs.io/en/latest/>



README versus Read-the-docs

- The README version is a extract of the full documentation (read-the-docs) for the DesignLib.
- The read-the-docs version contains the Tapeout calender and further info



Tapeout calendar

Welcome to IHP-Open-DesignLib documentation!

IHP-Open-DesignLib is repository, which contains open source IC designs using IHP SG13G2 BiCMOS processs. It is also a central point for design fabrication under the concept of IHP Free MPW runs funded by a public German project [FMD-QNC \(16ME083\)](#). Project funds can be used exclusively to produce chip designs for non-commercial activities, such as university education, research projects, and others. In the project, a continuation for the provision of free area for the open source community is to be worked out.

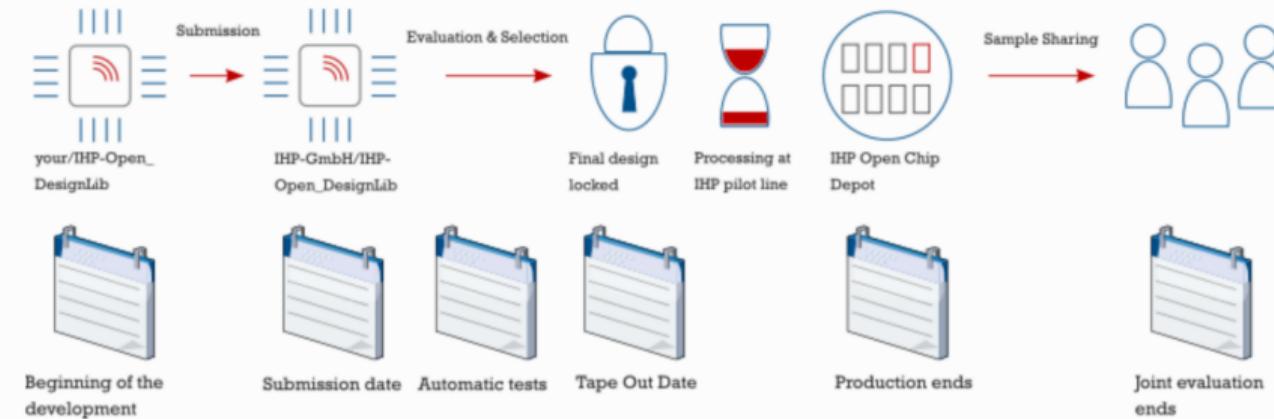
Tape In date	10 May 2024	11 Nov 2024	22 Nov 2024	07 Apr 2025	09 May 2025
Technology	SG13G2	SG13CMOS	SG13G2	SG13G2	SG13G2
Area available [mm ²]	10	220	20	140	30
11 Nov 2024	22 Nov 2024	07 Apr 2025	09 May 2025	18 Jul 2025	15 Sep 2025
SG13CMOS	SG13G2	SG13G2	SG13G2	SG13G2	SG13CMOS
220	20	140	30	30	220

Figure 6: Open DesignLib Tapeout Calendar



More submission information

The overview of the submission process is shown on the following figure.



The submission process contains a few steps, where some of them are mandatory and crucial:

1. Project development phase. At the beginning specifications and criteria will be defined by PDK status, later specifications from sponsors might be possible

Figure 7: Submission process detailed



Section 4

The last steps to a successful Tapeout



The last steps to a successful Tapeout

A few words about the finish of a Tapeout-ready GDS:

- A successfull Tapeout for a producable, correct and working micorchip contains more steps and knowdledge. It is out of scope of the course to explain and tutor these all details about this knowledge.
- Most of these steps will integrate into ORFS soon in a easy-to-use way eventually.



Topic to watch in design

Till then, here is a (not complete) list of important topics:

- Placement and instantiation of the Input-Outputs.
- Metall fills
- Sealring construction
- Automated DRC free



Github Actions

- Tapeout Nov 2024 Actions drc.yml:

https://github.com/IHP-GmbH/TO_Nov2024/blob/main/.github/workflows/drc.yml

- Tapeout Nov 2024 Actions runs:

https://github.com/IHP-GmbH/TO_Nov2024/actions



Active development in ORFS

Pull requests in ORFS with the search IHP in it:

<https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/pulls?q=IHP+>

The screenshot shows a GitHub repository page for 'OpenROAD-flow-scripts'. The 'Pull requests' tab is selected, showing 53 results. A search bar at the top right contains 'Type [] to search'. Below the tabs are filters for 'Labels (44)', 'Milestones (0)', and a 'New pull request' button. A search bar also contains 'IHP'. The main list displays the following pull requests:

- #2744 Reverted AT mods in config.mk's ✓ (merged 3 days ago)
- #2730 update metrics for ppl fix ✓ (merged 4 days ago)
- #2726 ibex autotuner updates for gf180, ihp-sg13g2 and sky130hd ✓ (merged 5 days ago)
- #2723 update Yosys to version 0.49 ✓ (merged 4 days ago)
- #2722 AES AutoTuner updates ✓ (merged 5 days ago)
- #2721 flow: designs: ihp-sg13g2: i2c-gpio-expander: Fix missing IO pwr/gnd ✓ (merged last week)



Issues in ORFS (open and IHP in it):

<https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/issues?q=is%3Aissue%20state%3Aopen%20IHP>

The screenshot shows the GitHub Issues page for the repository 'The-OpenROAD-Project / OpenROAD-flow-scripts'. The search bar at the top contains the query 'is:issue state:open IHP'. Below the search bar, there are filters for 'Issues' (88), 'Pull requests' (53), 'Discussions' (0), 'Actions' (0), 'Projects' (0), 'Security' (0), and 'Insights' (0). A 'Preview' button is visible on the right.

The main list displays eight open issues:

- #2616 - [ERROR] LEF Cell 'spi_DEF_FILL' has no matching GDS/OAS cell. Cell will be empty when running ihp-sg13g2 examples (waiting on op)
- #2574 - Min metal area ignored by ORFS (DRC error) (drt) (waiting on op)
- #2553 - Honor Cell Footprint in sky130hd, sky130hs and ihp (rsz)
- #2536 - Change CI to only run relevant checks (syndace)
- #2473 - make command fails inside the docker during detail routing (vijayank88)
- #1953 - Bug in ihp-sg13g2 jpeg flow (luarss)

Each issue entry includes a link, a number indicating the number of comments, and a small profile picture of the author.

Figure 9: ORFS issues, open and IHP in it

Workarounds (example masked_aes)

The README of the design example from earlier (masked_aes) gives some basic, but sufficient information about the actual state and the workarounds.

<https://github.com/HEP-Alliance/masked-aes-tapeout/blob/main/README.md>

The ASIC

To build the ASIC, set up OpenROAD-flow-scripts, clone this repository as `<ORFS-Root>/flow/designs/ihp-sg13g2/masked_aes` and run the build like any other ORFS design.

Sealring

The sealring was generated using a script included with IHP's open PDK.

Clone the PDK and set up the technology in KLayout. The following command creates the sealring:

```
$ klayout -n sg13g2 -zz -r <IHP-repo-root>/ihp-sg13g2/libs.tech/klayout/tech/scripts/sealring.py -rd width=1065.0 -rc ↗
```

The generated sealring has to be moved by -60 in both directions, which can be done in KLayout.

Caveats

Information on steps that are not fully automated yet as well as known issues.

Metal Fill

Metal fill has to be performed on the output GDS using a KLayout script provided as part of the IHP PDK. The script is currently work-in-progress here: [IHP-GmbH/IHP-Open-PDK#229](#)

DRC Violations



Before your Tapeout

If you plan to do a Tapeout with open-source to a Shuttlerun at IHP:

- Get in touch with the people at IHP before submitting a GDS.
- Inform yourself about the actual changes in the tools, the PDK and the submission repositories.
- Look for recent examples from the CoreExpert Group.



Non public and non open-source designs

IMPORTANT NOTICE:

- Using open-source doesn't mean you have to publish your work in open-source!

With using the open-source EDA tools and the open-source IHP PDK it is possible

- to design and fabricate closed source microchips at IHP.
- to not make a design public.

If you plan to do this: Talk to IHP!



Section 5

Contribute to open-source



Contributing with Git

If you want to start contributing to open-source (maybe to this course?):

- Create a Git account
- Join a discussion about an issue.
- When you discover an error, write an issue yourself.

It is really not much more than a ticket-system via email.



The Feedback to this course

Your feedback will become issues in the course repository. Join in the discussion if you want to contribute:

<https://github.com/OS-EDA/Course/issues?q=is%3Aissue>

A screenshot of a GitHub repository interface. The top navigation bar shows 'OS-EDA / Course'. Below it, the 'Issues' tab is selected, showing a count of 1. Other tabs include 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. A search bar at the top right contains the placeholder 'Type / to search'. Below the tabs, there's a 'New issue' button. The main area displays two issues under the heading 'is:issue':

- Tutorial for the slides** (opened by ThorKin last week)
- Which pandoc version is needed to compile/build slides?** (closed by sergeiandreyev last week)

Filtering options like 'Open', 'Closed', 'Author', 'Labels', 'Projects', 'Milestones', 'Assignees', 'Types', and sorting by 'Newest' are available.

Figure 11: Issue tracking in the course repository



Chapter 8 -
Tapeout -
QUESTIONS

Course authors
(Git file)

Chapter 8 - Tapeout - QUESTIONS

Course authors (Git file)



Chapter 8 -
Tapeout -
QUESTIONS

Course authors
(Git file)

Recap questions for Chapter 8



Questions

Chapter 8 -
Tapeout -
QUESTIONS

Course authors
(Git file)

- Where to find the reports, results and logs?
- How to read them (name two ways)?
- How to create a new design in the ORFS directory structure?
- What is a IO-Pad?
- Name some tuning parameters (round robin)
- What is PPA?
- How to submit a Tapeout at IHP?
- What parts are still in the making of ORFS to come soon?
- Try to explain Sealring, IO-Cells, Filler-Cells, PDN. (Round robin)
- What to do in case you discovered an error in ORFS?

Chapter 8 - Tapeout - TRAINING - Common

Course authors (Git file)



