Chapter 0 - Preparations

Before the course (Optional)

Course authors (Git file)

Section 1

Description

Description

Before the course starts, you can take some optional preparations regarding the open-source EDA software tools that will be used during the course. Either you can install them locally on your own computer or get access to a preconfigured server from IHP.

Here comes a short description of the three 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 neccessary 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.

Option C: IHP server with installed tools

- Working on the IHP server is the more convenient approach and does not require to install anything on your computer.
- The tools are ready to use installed on the IHP server.
- A permanent connnection to the server is needed (reliable internet connection).
- This option will work on various computers and operating systems (Linux/Win/Mac).

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.

```
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:

```
cd OpenROAD-flow-scripts
```

```
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:

```
source ./env.sh
```

```
klayout -v
```

```
yosys --version
```

```
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 avaiable 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
```

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

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

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

```
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]
        https://download.docker.com/linux/ubuntu \
3   $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
4   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

```
sudo groupadd docker
```

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

```
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:

```
...
Hello from Docker!
This message shows that your installation appears to be
working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from
    the Docker Hub. (amd64)
 3. The Docker daemon created a new container from that
    image which runs the executable that produces the
    output you are currently reading.
 4. The Docker daemon streamed that output to the Docker
    client, which sent it to your terminal.
...
```

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:

```
sudo apt install git
```

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

```
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

```
./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.

Section 4

Install Option C: Guide to use an IHP server with installed EDA tools

Install Option C: Guide to use an IHP server with installed EDA tools

- To be written (t.b.w)
- Waiting on IHP information about their docker / server install.

Ask at IHP for a login

t.b.w.

Getting started on the server

t.b.w.

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 **T3:** Training	**L5:** PDK **T5:** Training	**L7:** OpenROAD Flow scripts **T7:** Training	**L8:** Tapeout **Feedback**
L2: OpenROAD tools	**L4:** OpenROAD first run	**L6:** OpenROAD GUI	**L7:** OpenROAD Flow scripts 2	Spare time and Wrap-Up
T2: Training	**T4:** Training	**T6:** Training	**T7:** Training	

Chapter 1 - Introduction and overview

Course authors (Git file)

1. Welcome
2. Course overview
3. Course components
4. The Training sessions
5. Open-source EDA for digital designs
6. AMA (Ask me anything)

Section 1

Welcome

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

- 01 Introduction
- 02 Workflow
- 03 Design and example pick
- 04 OpenROAD first run
- 05 PDK Examination
- 06 Data in OpenROAD
- 07 LVS and DRC
- 08 Simulation and PPA
- 09 Scripting
- 10 GDS and Tapeout

Chapter names

- 01 Introduction
- 02 Workflow
- 03 Design and example pick
- 04 OpenROAD first run
- 05 PDK Examination

Day 1 - 2

- 06 Data in OpenROAD
- 07 LVS and DRC
- 08 Simulation and PPA
- 09 Scripting
- 10 GDS and Tapeout

Day 3 - 5

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 **T3:** Training	**L5:** PDK **T5:** Training	**L7:** OpenROAD Flow scripts **T7:** Training	**L8:** Tapeout **Feedback**
L2: OpenROAD tools	**L4:** OpenROAD first run	**L6:** OpenROAD GUI	**L7:** OpenROAD Flow scripts 2	Spare time and Wrap-Up
T2: Training	**T4:** Training	**T6:** Training	**T7:** Training	

L : Lectures

T : Training and Hands-On

Q : Questions

Section 3

Course components

Lectures

Lectures:

- All the chapters start with a lecture slide deck.
- The trainer will walk you trough 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 convered.

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.

Cheat Sheets

Some things are really hard to remember:

- Abbreviations
- Complex relations and graphics
- EDA tools workflow
- Schedule of the week
- Mathematics (joking, wer’re not doing math here)
- …

- That is why we have Cheat Sheets.
- They’re made for cheating the hard parts.
- Cheatsheets work best when printed as handouts.
- You should have them nearby the computer during the course.

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 a test nor a challenge.
    - Think of this as a helpfull recap of yesterdays content.
  - If no answer is found, the trainer helps with the answer.

Section 4

The Training sessions

Login at IHP

Now:

- Onboarding to the computers for everyone

Levels

- Success points inbetween lectures
- This is too fast
- This is too slow

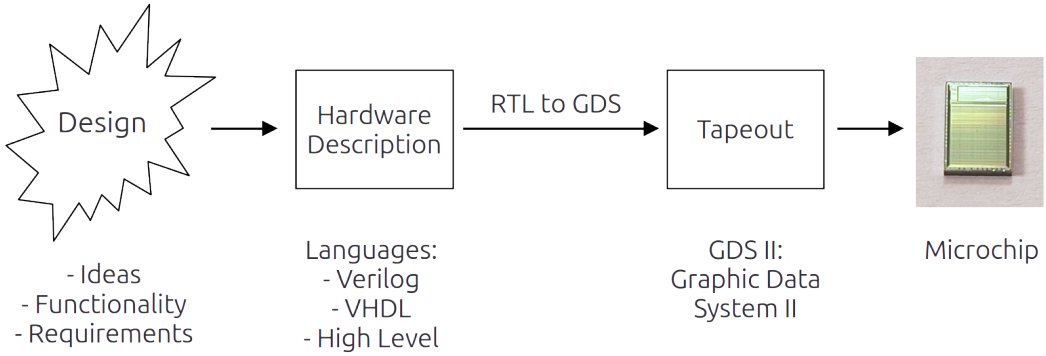
Availablitily GitHub PDF Downloads

- Follow in your own tempo. Get all the data here:
- Link / QR to the course materials

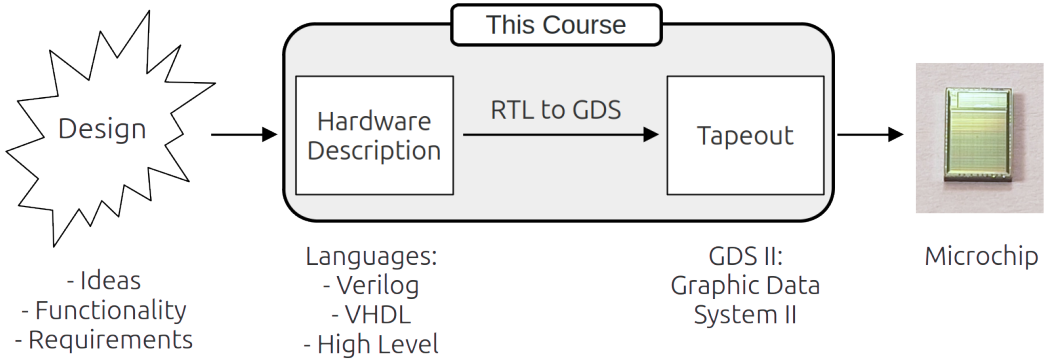
Section 5

Open-source EDA for digital designs

From Design to Microchip



RTL to GDS - Workflow



The cheatsheet

First usage of the cheatsheet:

- EDA
- RTL
- GDS II
- ....

Further topics

- What is the new thing with this course?
- Advantages of open-source in EDA
- The actual state of open-source EDA
- Goals of this course.
- How to participate and interact with this course.
- Producing chips at IHP with the open PDK

Section 6

AMA (Ask me anything)

AMA (Ask me anything)

- Opportunity to ask questions about everything (chapter 1 ?).

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

Course authors (Git file)

1. Server and user credentials
2. First lookaround in Gnome
3. Load the course data
4. Linux shell

Server

- Which software to use on the Workstation
- How to connect to the server in IHP

Login

- User login on the server

Search and start a program

Tabbing between programs

Browser and Course Git Repository

Download and unpack

Look around in the course data

Short introduction to the shell

Chapter 2 - OpenROAD workflow

Course authors (Git file)

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

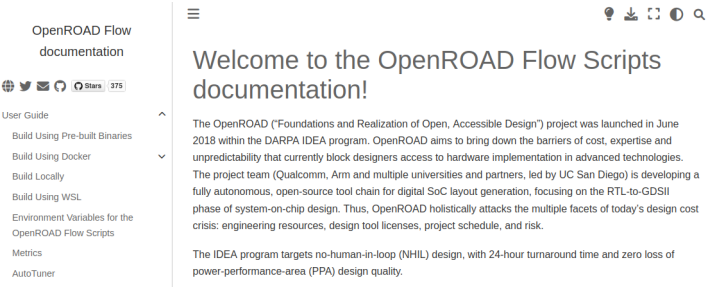


Figure 1: Darpa IDEA [1]

[1] Source: Screenshot of the webpage.

Darpa IDEA

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

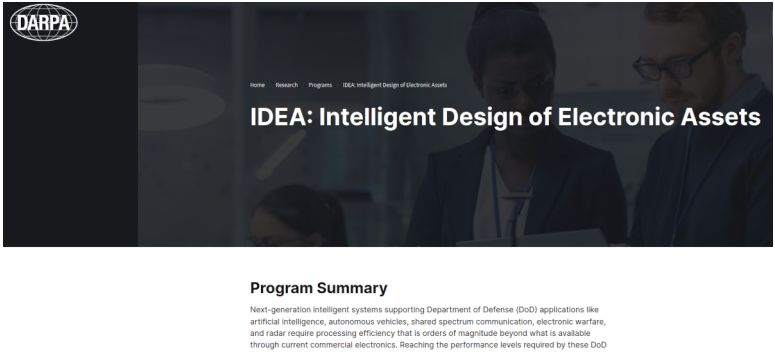


Figure 2: Darpa IDEA [2]

[2] Source: 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!

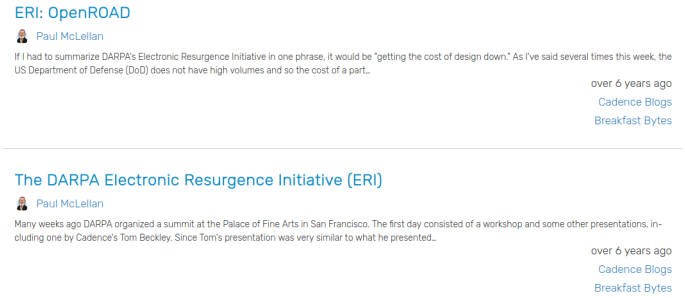
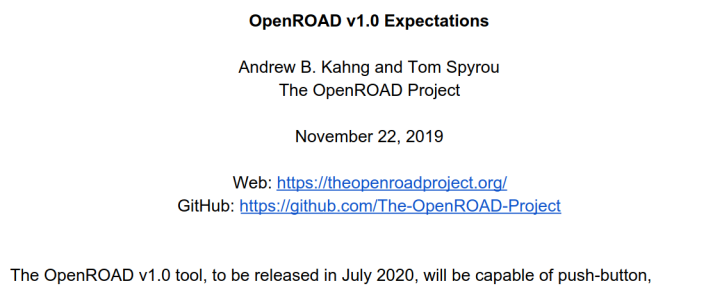


Figure 3: Darpa ERI [3]

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 v1.0 [4]

V1.0 Roadmap

How to deal with these expectations:

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

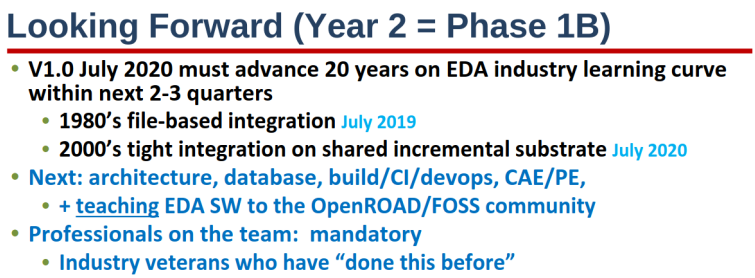


Figure 5: Looking forward [5]

[5] Source: Screenshot of the webpage.

Courses for OpenROAD

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

https://theopenroadproject.org/Courses/

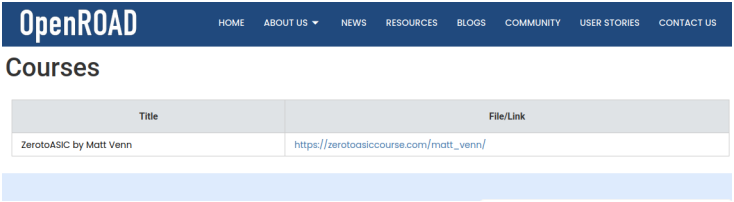


Figure 6: Courses [6]

[6] Source: Screenshot of the webpage.

Help is on the way

Kudos to this course:

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

Figure 7: This course [7]

[7] Source: Screenshot of the webpage.

Section 2

OpenROAD Flow Scripts

Flow steps

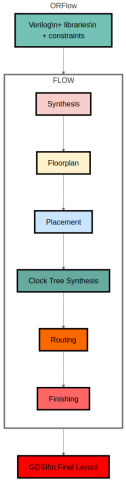


Figure 8: OpenROAD flow steps [8]

Flow components

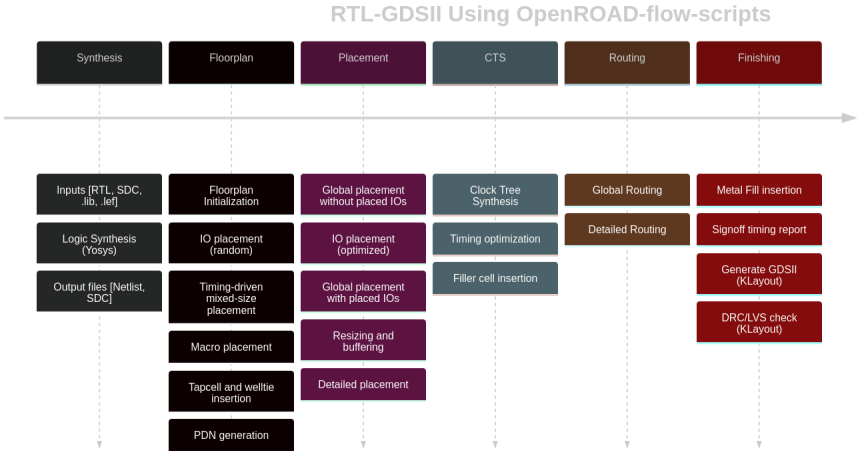
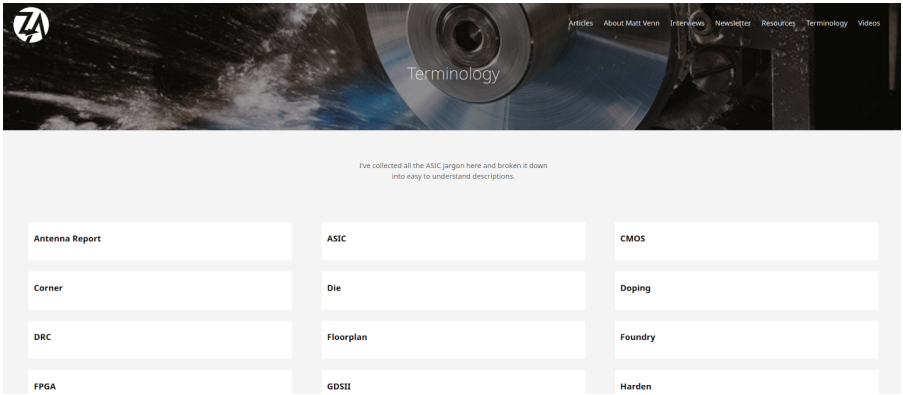


Figure 9: OpenROAD flow components [9]

Help with the terminology

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

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



Chapter 02 - Terminology - TRAINING - Common

Course authors (Git file)

1. Searching

Browse the resources

What is missing?

Chapter 3 - Verilog crash course

Course authors (Git file)

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 synthesible language constructs in Verilog.

The aim of this selection is not commercial tools, but open-source development tools such as OpenRoad [1] 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 VerilogRTLsynthesis. It currently has extensive Verilog-2005 support and provides a basic set of synthesis algorithms for various application do mains. 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  Assignmente
9  Module instanciations
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 shiftregister

```
1  module LSFR (
2
3  input  wire   load,
4  input  wire   loadIt,
5  input  wire   enable,
6  output wire   newBit,
7  input  wire   clk,
8  input  wire   reset);
9
10 wire       [17:0]   fsRegN;
11 reg        [17:0]   fsReg;
12 wire                taps_0, taps_1;
13 reg                 genBit;
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
```

```
1  assign newBit = fsReg[0];
2  assign fsRegN = {genBit,fsReg[17 : 1]};
3
4  always @(posedge clk) begin
5    if(reset) begin
6      fsReg <= 18'h0;
7    end else begin
8      if(enable) begin
9        fsReg <= fsRegN;
10     end
11   end
12 end
13
14 endmodule
```

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

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:

- Hexadepimal 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{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 <= b, und a >= 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    reg q;
6
7    always @ ( posedge clk or
8               posedge reset)
9      begin
10       if ( rst )
11         q <= 1'b0;
12       else
13         q <= d;
14     end
15 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:

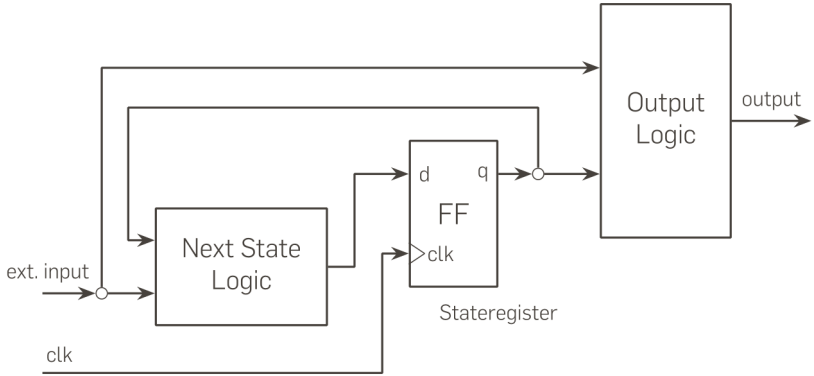


Figure 1: Idealized diagram of a synchronous circuit

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

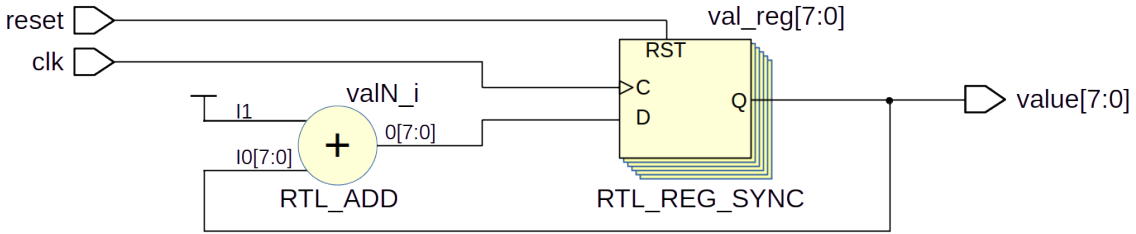


Figure 2: Synthesis diagram of the binary counter

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* instructions 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;
```

```
1    always @(posedge clk) begin
2
3      if (reset) begin // Synchron reset
4        val <= {WIDTH{1'b0}};
5      end else begin
6        val <= valN;
7      end
8
9    end
10
11   // Nextstate logic
12   assign valN = (val < MODULO) ? val + 1 : 0;
13
14   // Output logic
15   assign value = val;
16   assign sync = ((val >= hsMin) && (val <= hsMax)) ? 1 : 0;
17
18 endmodule
```

Synthesis result of the modulo counter

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

TODO: Picture here

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

Synthesis result of the register file

The synthesis result is then a little more confusing:

TODO: Picture here

Section 5

Selected feature: Parameterized counter

Selected feature: Parameterized counter

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

```
module cnt
  #(parameter N = 8,
    parameter DOWN = 0)

   (input clk,
    input resetN,
    input enable,
    output reg [N-1:0] out);

    always @ (posedge clk) begin

     if (!resetN) begin // Synchron
       out <= 0;
     end else begin
     if (enable)
       if (DOWN)
         out <= out - 1;
       else
         out <= out + 1;
     else
       out <= out;
     end

    end

endmodule
```

```
module doubleSum
  #(parameter N = 8)
   (input clk,
    input resetN,
    input enable,
    output [N  :  0] sum);

  wire [N - 1  :  0] val0;
  wire [N - 1  :  0] val1;

  // Counter 0
  cnt #(.N(N), .DOWN(0)) c0 (.clk(clk),
                             .resetN(resetN),
                             .enable(enable),
                             .out(val0));

  // Counter 1
  cnt #(.N(N), .DOWN(1)) c1 (.clk(clk),
                             .resetN(resetN),
                             .enable(enable),
                             .out(val1));

  assign sum = val0 + val1;

endmodule
```

Synthesis result of the parameterized counter

TODO: Picture here

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));
```

```
1    // Counter 1
2    defparam c1.N = N;
3    defparam c1.DOWN = 1;
4    cnt c1 (.clk(clk),
5            .resetN(resetN),
6            .enable(enable),
7            .out(val1));
8
9    assign sum = val0 + val1;
10
11 endmodule
```

This variant leads to the same synthesis result.

Section 6

Selected feature: Preprocessor

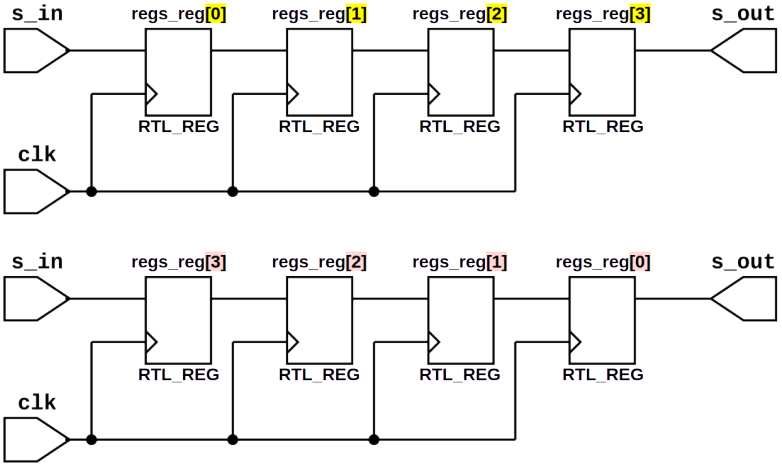
Selected feature: Preprocessor

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 meaningfully 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
```

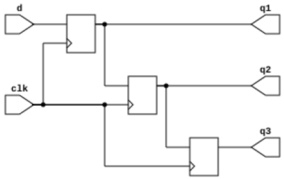
Blocking and Non-blocking assignments in always_ff

Caution with false assignments in *always_ff*:

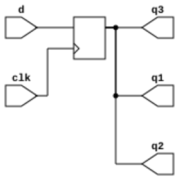
```
1  module demoOk (input clk,
2                 input d,
3                 output q1,
4                 output q2,
5                 output q3);
6    always_ff @(posedge clk) begin
7      q1 <= d;
8      q2 <= q1;
9      q3 <= q2;
10   end
11 endmodule
```

```
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
```

Okay:



Wrong:



Chapter 3 - Verilog - TRAINING - Common

Course authors (Git file)

Analyse parts of the Verilog source

Find combinational and synchronous parts in

- Code
- Schematic

Generate schematic and JSON

Generate * Schematic * JSON

yosys file

Read the .ys file

Chapter 04 - OpenROAD first run

Course authors (Git file)

Section 1

Openroad GUI

Openroad GUI

```
openroad -gui
```

Section 2

Example pick

Example: gcd (from the OpenROAD-flow-script examples)

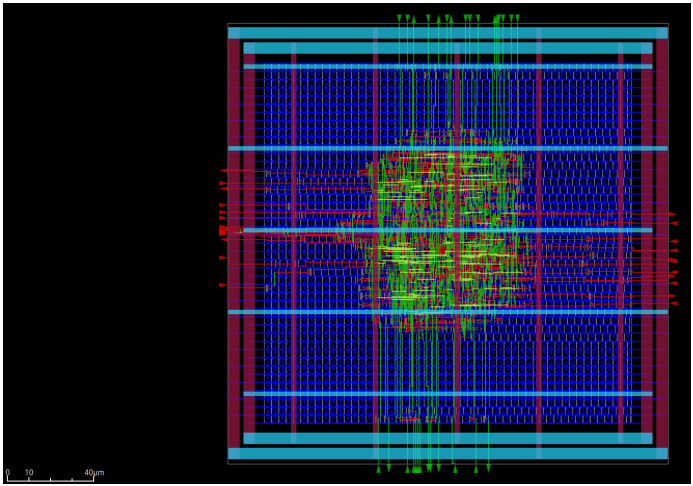
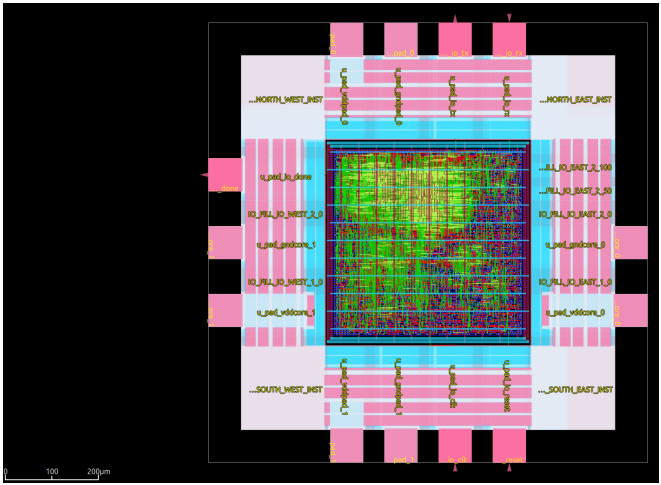


Figure 1: GDS gcd

Example: masked_aes (from the Sign:HEP research project)

HEP Alliance - Masked AES



Chapter 04 - OpenROAD first run - TRAINING - Advanced

Course authors (Git file)

1. Build an external example

EDA tools

- One of the options of chapter 0 (preparations) is needed.
- Navigate to the /flow folder

Clone the design “masked AES” from the HEP Alliance

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
```

Enable the design in the Makefile

- Edit the Makefile:
  - Uncomment the line with your choosen 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. For example the gcd on SKY130 design:

```
1 # DESIGN_CONFIG=./designs/asap7/gcd/config.mk
```

  - 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 the flowscript

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

Success

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

```
1 Log                    Elapsed seconds  Peak Memory/MB
```

CONGRATS! Your design got build to a GDS!

Chapter 04 - OpenROAD first run - TRAINING - Bonus

Course authors (Git file)

1. Integrate a new design into OpenROAD flowscripts

Integrate a new design into OpenROAD flowscripts

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

- LSFR from earlier?
- Pre-selected TinyTapeOut designs?

Run the flowscript

Todo.

Success

Todo. * The choosen design should finish after a while and a lot of console output with a table (time/memory) like this:

```
1 Log                         Elapsed seconds  Peak Memory/MB
```

CONGRATS! Your first(?) design got build to a GDS!

Chapter 04 - OpenROAD first run - TRAINING - Common

Course authors (Git file)

1. Start the first run

EDA tools

- One of the options of chapter 0 (preparations) is needed.
- Navigate to the /flow folder

Enable the design in the Makefile

- Edit the Makefile:
  - Uncomment the line with your choosen DESIGN_CONFIG from ihp-sg13g2. For example the gcd design:

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

  - Re-comment the previous uncommented line with DESIGN_CONFIG. For example the gcd on SKY130 design:

```
1 # DESIGN_CONFIG=./designs/asap7/gcd/config.mk
```

  - 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 the flowscript

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

Success

- The choosen 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_1_yosys	0	24
1_1_yosys_canonicalize	0	17
1_1_yosys_hier_report	0	12
2_1_floorplan	0	110
2_2_floorplan_io	0	106
2_3_floorplan_tdms	0	98
2_4_floorplan_macro	0	106
2_5_floorplan_tapcell	0	105
2_6_floorplan_pdn	0	108
3_1_place_gp_skip_io	0	108
3_2_place_iop	0	107
3_3_place_gp	0	320
3_4_place_resized	0	289
3_5_place_dp	0	112
4_1_cts	1	379
5_1_grt	0	340
5_2_route	93	899
5_3_fillcell	0	111
6_1_fill	0	113
6_1_merge	1	368
6_report	1	292

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) Examination

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. 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

ToDo: Image of the border between development and production

In the context of this course

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 openss

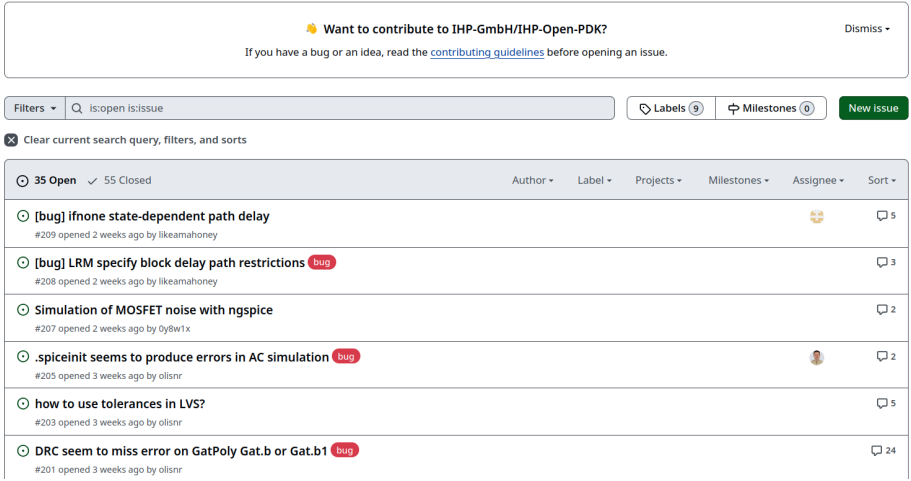


Figure 1: Issues open

Issues closed

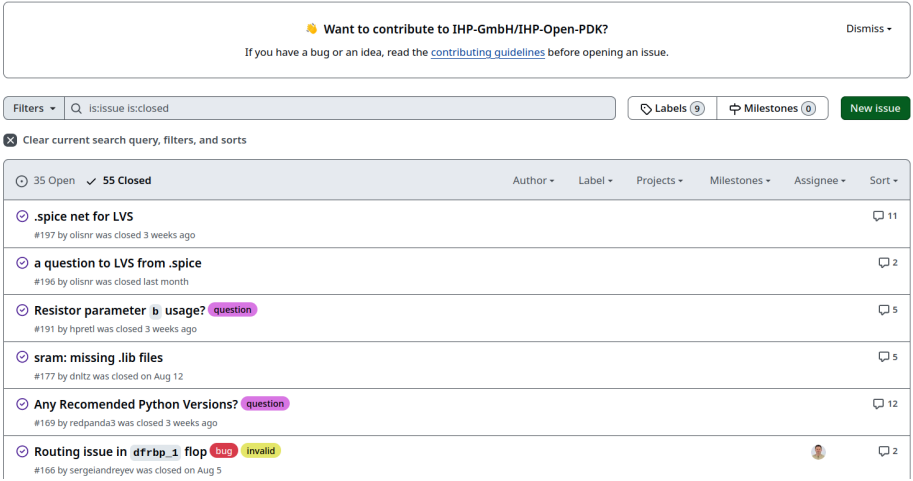


Figure 2: Issues clsoed

Pull requests closed

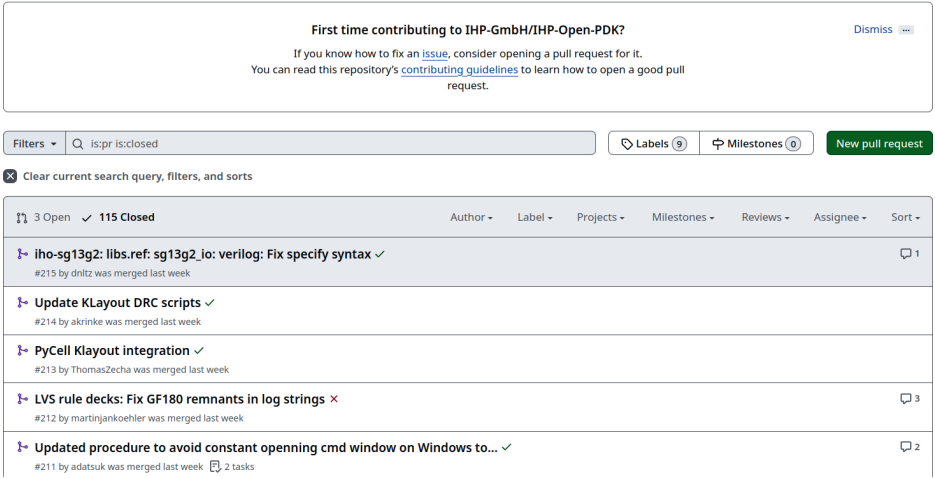


Figure 3: Pull requests closed

Resources for you

Contributing?

Wiki

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

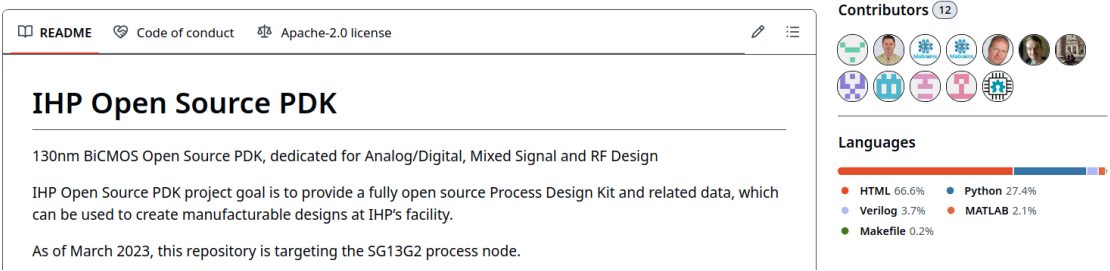


Figure 4: Readme

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

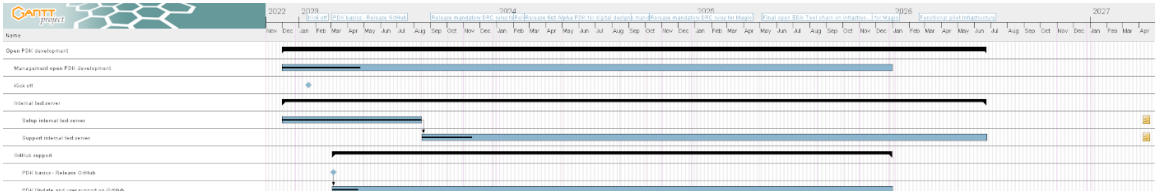


Figure 5: Gantt chart

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

Standard cell library

ToDo:

- Where in the RTl-to-GDS is the cell library needed?
- Design to cells to GDS and Tapeout
- Naming of the cells

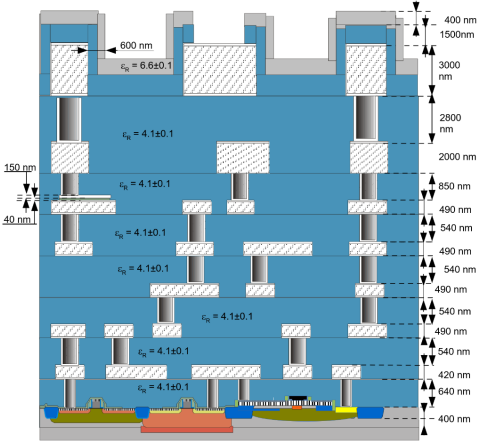
A single cell from the library

ToDo:

- Pick one cell
- Pictures of the layers of the cell
  - Klayout
  - 3D Rederings?
- Schematic of the cell

Layer stack

IHP sg13g2 Layers in a picture.



Section 4

File formats

Cell AO21: VERILOG HDL language

```
// type: AO21
`timescale 1ns/10ps
`celldefine
module sg13g2_a21o_1 (X, A1, A2, B1);
    output X;
    input A1, A2, B1;

    // Function
    wire int_fwire_0;

    and (int_fwire_0, A1, A2);
    or (X, int_fwire_0, B1);

    // Timing
    specify
```

Cell AO21: SPICE Netlist

```
* Library name: sg13g2_stdcell
* Cell name: sg13g2_a21o_1
* View name: schematic
* Inherited view list: spectre cmos_sch cmos.sch schematic veriloga
* pspice dspf
.subckt sg13g2_a21o_1 A1 A2 B1 VDD VSS X
XN0 net1 A1 net2 VSS sg13_lv_nmos w=640.00n l=130.00n ng=1 ad=0 as=0
XN1 net2 A2 VSS VSS sg13_lv_nmos w=640.00n l=130.00n ng=1 ad=0 as=0
XN2 net1 B1 VSS VSS sg13_lv_nmos w=640.00n l=130.00n ng=1 ad=0 as=0
XN3 X net1 VSS VSS sg13_lv_nmos w=740.00n l=130.00n ng=1 ad=0 as=0 p
XP0 net1 B1 net3 VDD sg13_lv_pmos w=1.000u l=130.00n ng=1 ad=0 as=0
XP1 net3 A1 VDD VDD sg13_lv_pmos w=1.000u l=130.00n ng=1 ad=0 as=0 p
XP2 net3 A2 VDD VDD sg13_lv_pmos w=1.000u l=130.00n ng=1 ad=0 as=0 p
XP3 X net1 VDD VDD sg13_lv_pmos w=1.12u l=130.00n ng=1 ad=0 as=0 pd=
.ends
```

Cell AO21: Circuit design language

```
********************************************************************************
* Library Name: sg13g2_stdcell
* Cell Name:    sg13g2_a21o_1
* View Name:    schematic
********************************************************************************

.SUBCKT sg13g2_a21o_1 A1 A2 B1 VDD VSS X
*.PININFO A1:I A2:I B1:I X:O VDD:B VSS:B
MN0 net1 A1 net2 VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
MN1 net2 A2 VSS VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
MN2 net1 B1 VSS VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
MN3 X net1 VSS VSS sg13_lv_nmos m=1 w=740.00n l=130.00n ng=1
MP0 net1 B1 net3 VDD sg13_lv_pmos m=1 w=1.000u l=130.00n ng=1
MP1 net3 A1 VDD VDD sg13_lv_pmos m=1 w=1.000u l=130.00n ng=1
MP2 net3 A2 VDD VDD sg13_lv_pmos m=1 w=1.000u l=130.00n ng=1
```

Section 5

Ruleset documents

Chapter 5 - PDK Examination - TRAINING - Advanced

Course authors (Git file)

1. 1. Open Klayout
2. 2. Matchmaking a cell
3. 3. Examing cell GDS
4. 4. Logic function of the 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

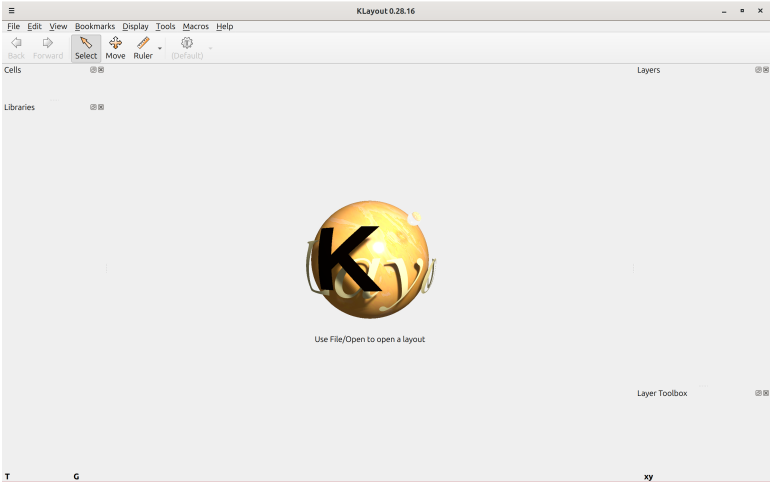


Figure 1: Start Klayout

2. Matchmaking a cell

- Here is a 3D rendering of a standard cell (Insert name? Or find name?)
- ToDo: Insert cell picture
- Can you identify the layers and their order from the cell?

2. Matchmaking a cell

ToDo.

3. Examing cell GDS

Todo. * Search for the cell GDS and examine the layers and their order.

3. Examine cell GDS

Todo. - Picture of

4. Logic function of the cell

- Find the schematic of the cell.

4. Logic function of the cell

- Match the schematic and the logic function of the cell.

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

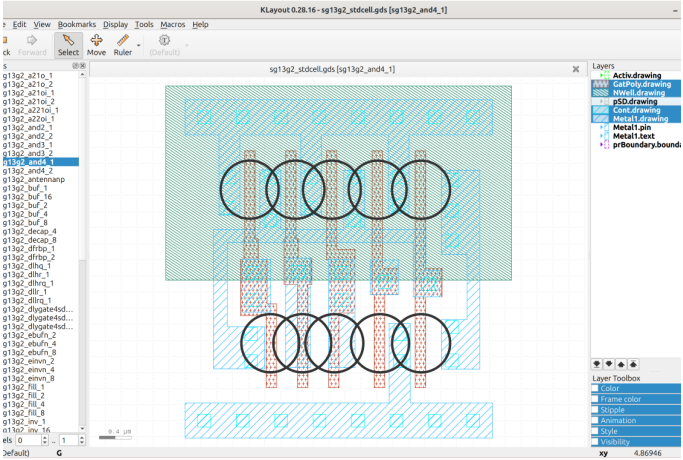


Figure 1: AND4_1

Solution: Transistor count in file

File: sg13g2_sdtcell.cdl

```
MN4 net17 D VSS VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
MN3 net16 C net17 VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
MN2 net15 B net16 VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
MN1 net1 A net15 VSS sg13_lv_nmos m=1 w=640.00n l=130.00n ng=1
MN0 X net1 VSS VSS sg13_lv_nmos m=1 w=740.00n l=130.00n ng=1
MP0 net1 A VDD VDD sg13_lv_pmos m=1 w=840.00n l=130.00n ng=1
MP4 X net1 VDD VDD sg13_lv_pmos m=1 w=1.12u l=130.00n ng=1
MP3 net1 D VDD VDD sg13_lv_pmos m=1 w=840.00n l=130.00n ng=1
MP2 net1 C VDD VDD sg13_lv_pmos m=1 w=840.00n l=130.00n ng=1
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

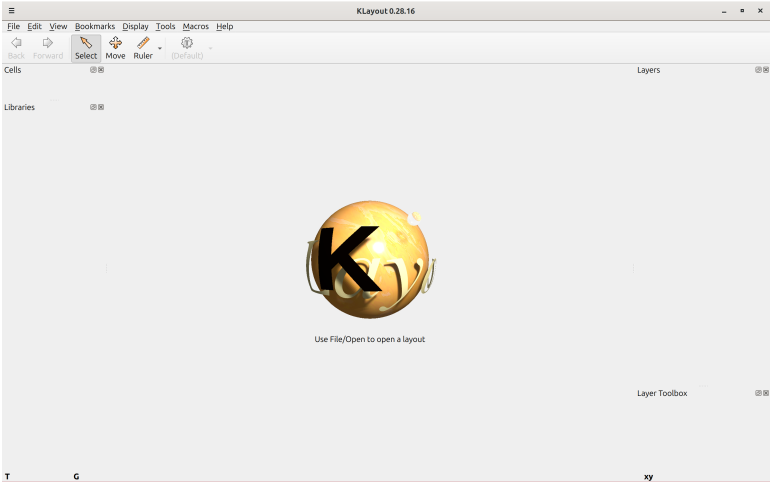


Figure 1: Start Klayout

2. Load example GDS

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

2. Load example GDS

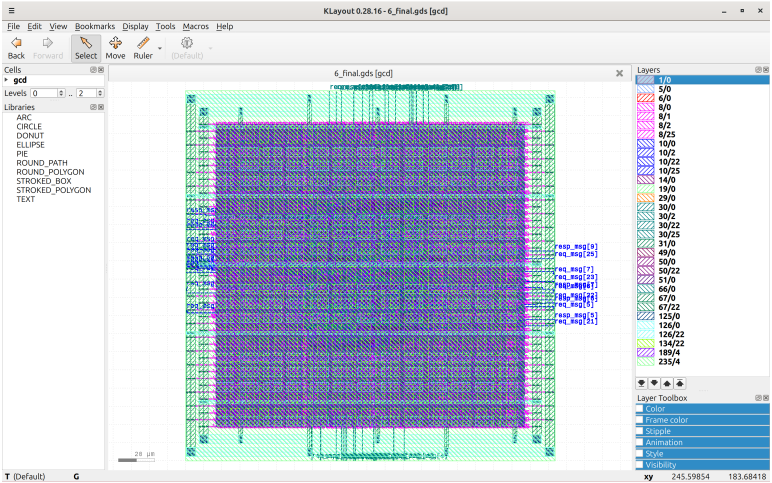


Figure 2: Load GDS

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

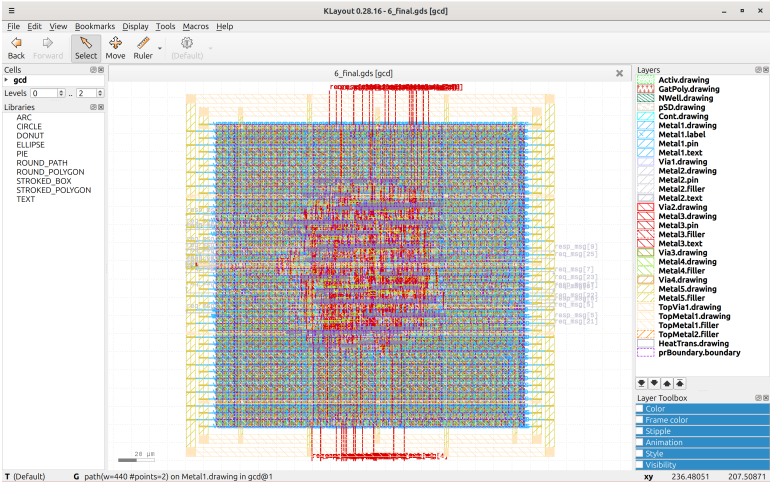


Figure 3: Use LYP file

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

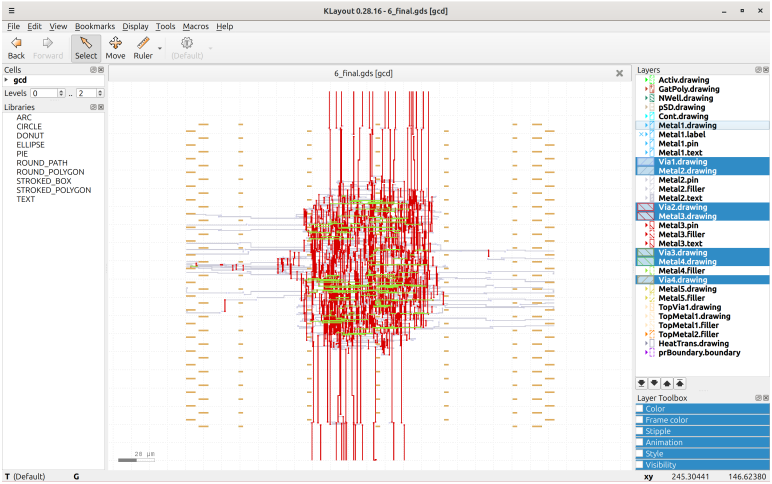


Figure 4: GDS zoomed out

4. Navigate Layers and GDS

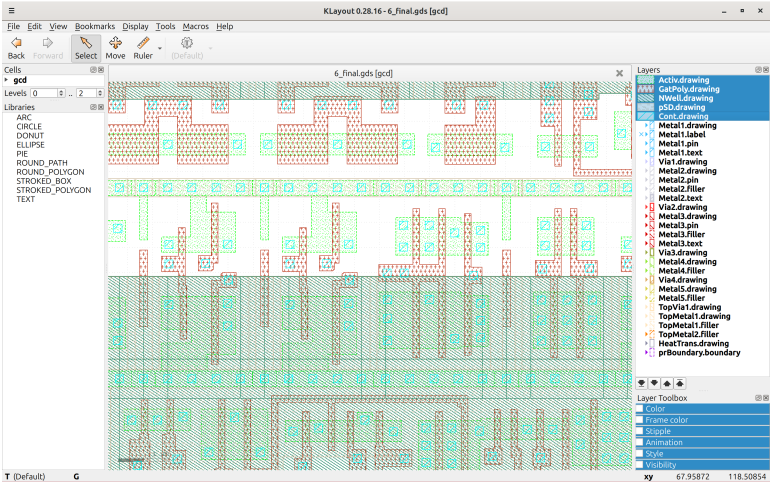


Figure 5: GDS zoomed in

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

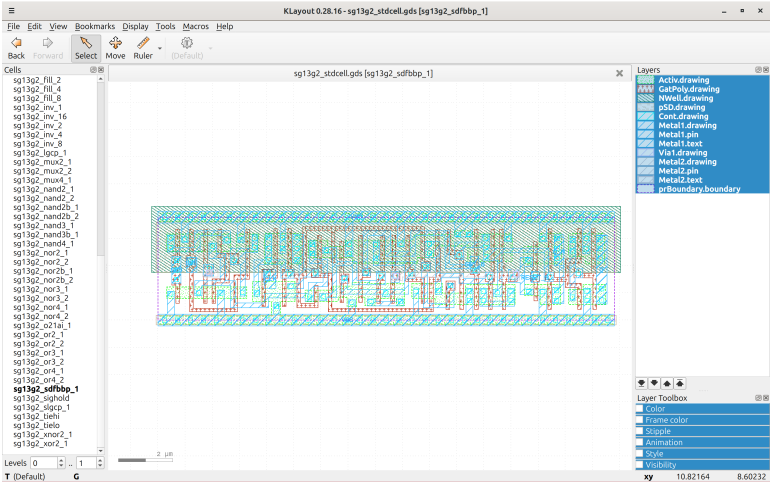


Figure 6: Cell library

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

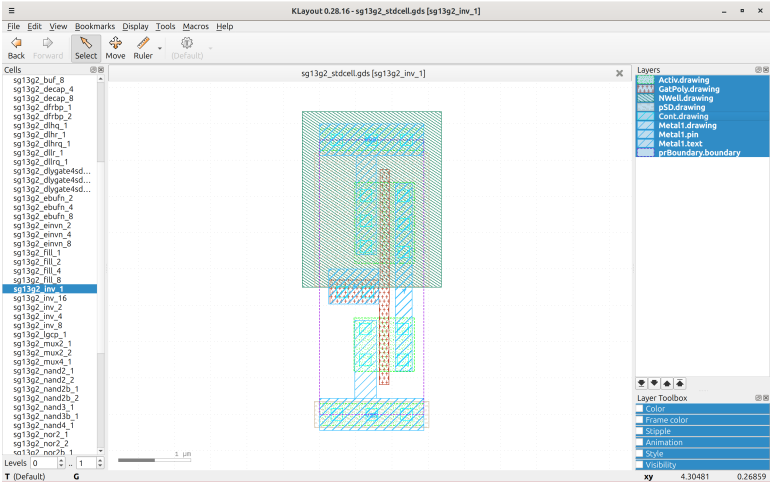


Figure 7: Single cell

Chapter 6 - OpenROAD GUI

Course authors (Git file)

Chapter 6 - OpenROAD GUI - TRAINING - Common

Course authors (Git file)

Chapter 7 - OpenROAD flow scripts

Course authors (Git file)

Chapter 7 - OpenROAD flow scripts - TRAINING - Common

Course authors (Git file)

Chapter 8 - Tapeout

Course authors (Git file)

Chapter 8 - Tapeout - TRAINING - Common

Course authors (Git file)

1. The IHP repos

OpenLib

PDK

TO_XYZ

Read the docs

Github Actions

- Look at some Github Actions in the Tapeout repositories
- What is a yaml workflow?