Introduction to Open-Source Tools

Simon Dorrer, IICQC, JKU

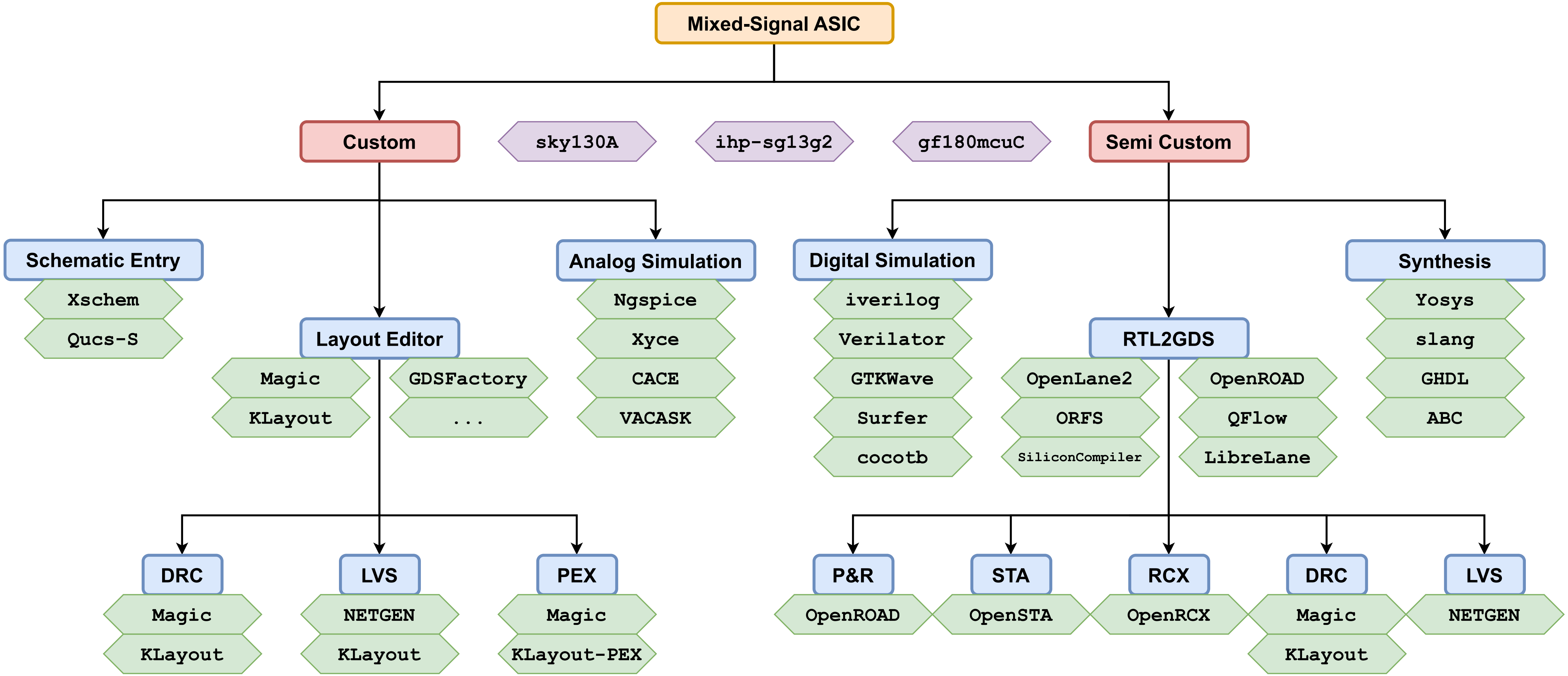
Institute for
Integrated Circuits and
Quantum Computing

**JOHANNES KEPLER
UNIVERSITÄT LINZ**
Altenberger Straße 69
4040 Linz, Österreich
jku.at

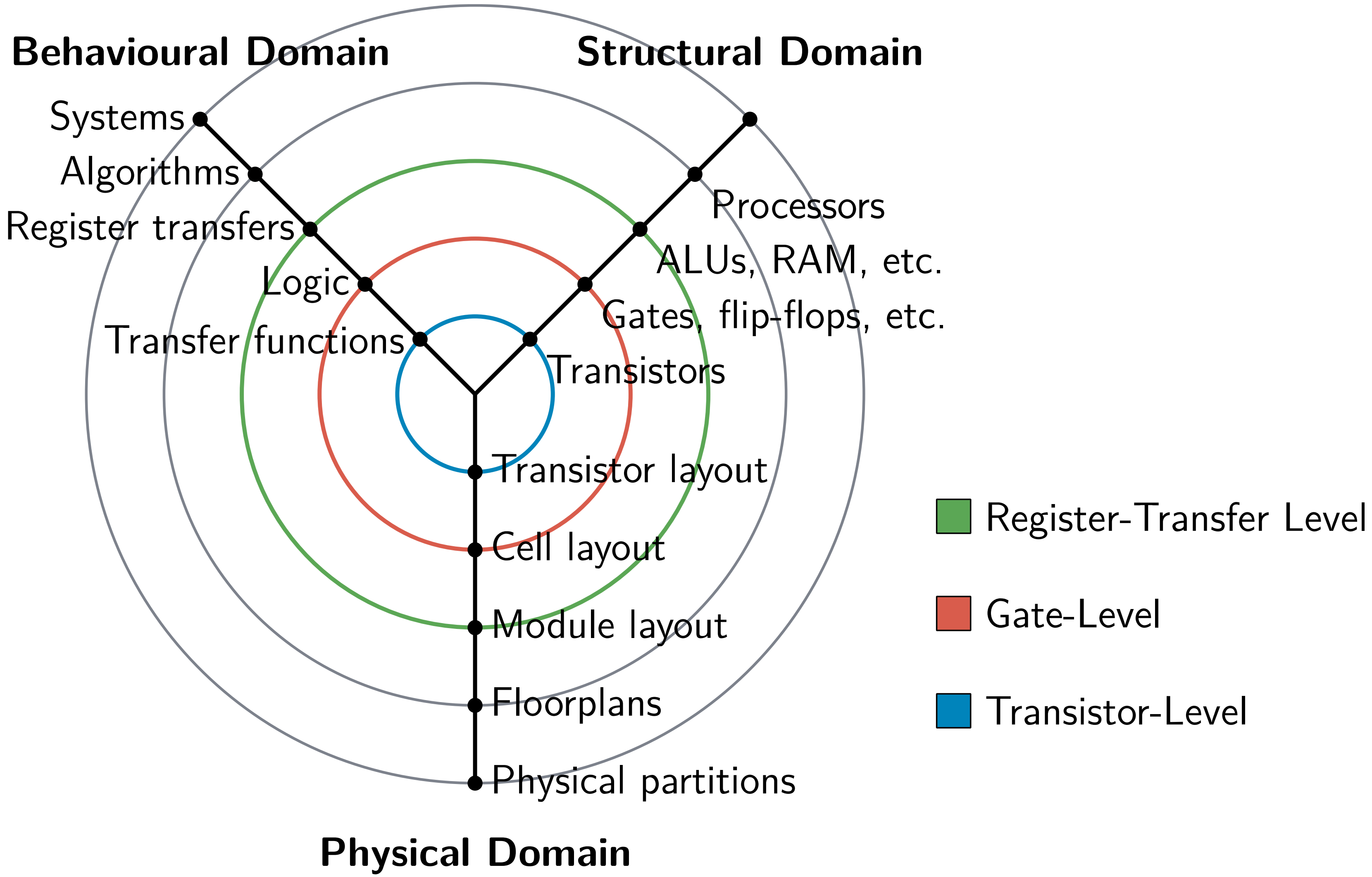
Free and Open-Source Software (FOSS)

- **Verilator:** Verilog code linter
- **Icarus Verilog (iVerilog):** Verilog compiler and simulator
- **GTKWave / Surfer:** Waveform viewer
- **GHDL:** VHDL analyzer, compiler, simulator and synthesizer
- **YOSYS / ABC:** Verilog synthesis tool
- **Magic / KLayout:** Layout editor with DRC and PEX
- **NextPNR:** Portable FPGA place and route tool
- **LibreLane:** Digital RTL-to-GDS flow
- **IIC-OSIC-Tools:**
  - https://github.com/iic-jku/iic-osic-tools
  - All-in-one Docker container for FOSS based IC designs for analog and digital circuit flows

IIC-OSIC-TOOLS



Y-Diagramm



Behavioural Domain

Systems

Algorithms

Register transfers

Logic

Transfer functions

Structural Domain

Processors

ALUs, RAM, etc.

Gates, flip-flops, etc.

Transistors

Physical Domain

Transistor layout

Cell layout

Module layout

Floorplans

Physical partitions

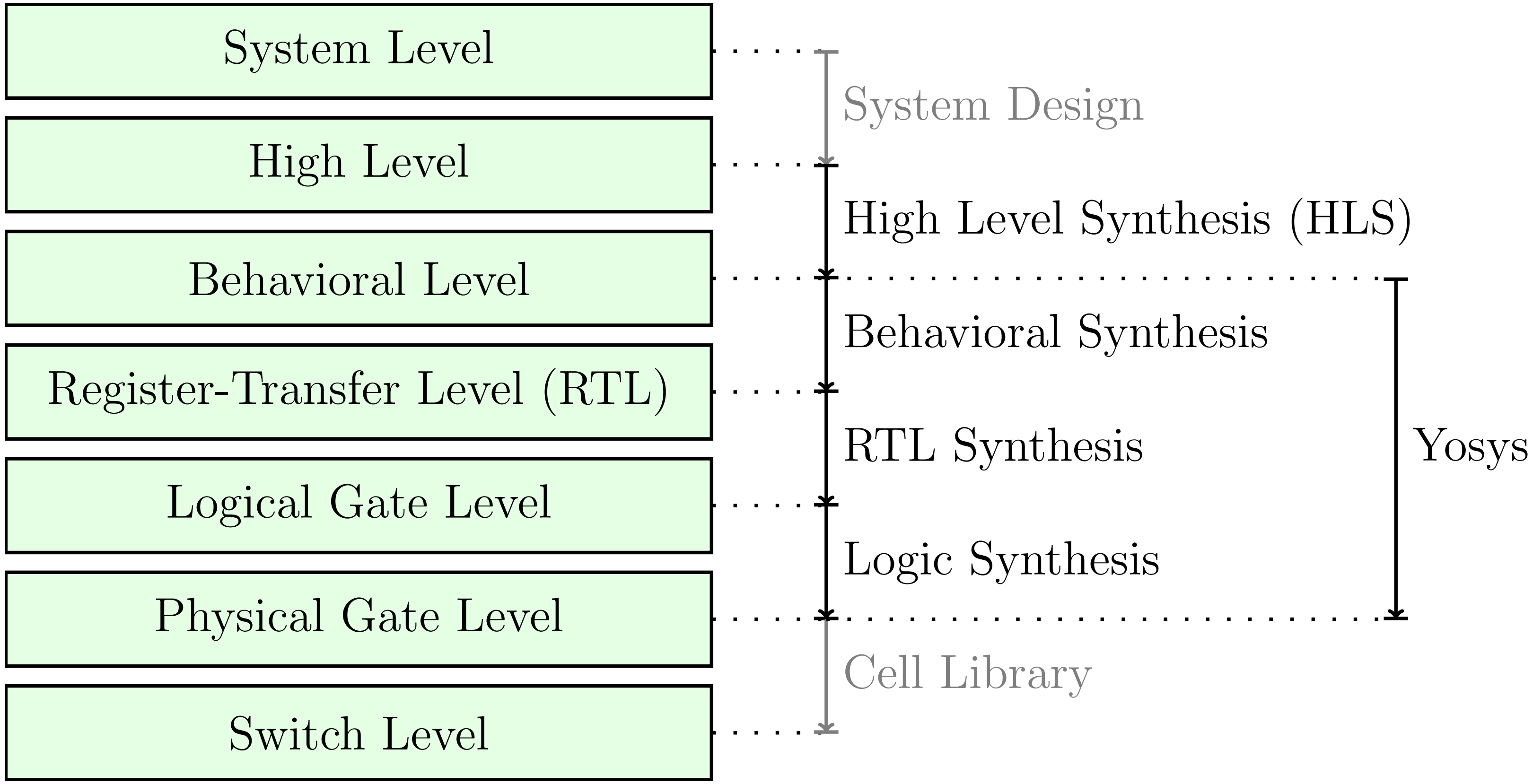
🟩 Register-Transfer Level

🟥 Gate-Level

🟦 Transistor-Level

Yosys

- Github: https://github.com/YosysHQ/yosys
- Documentation: https://yosyshq.readthedocs.io/projects/yosys/en/latest/

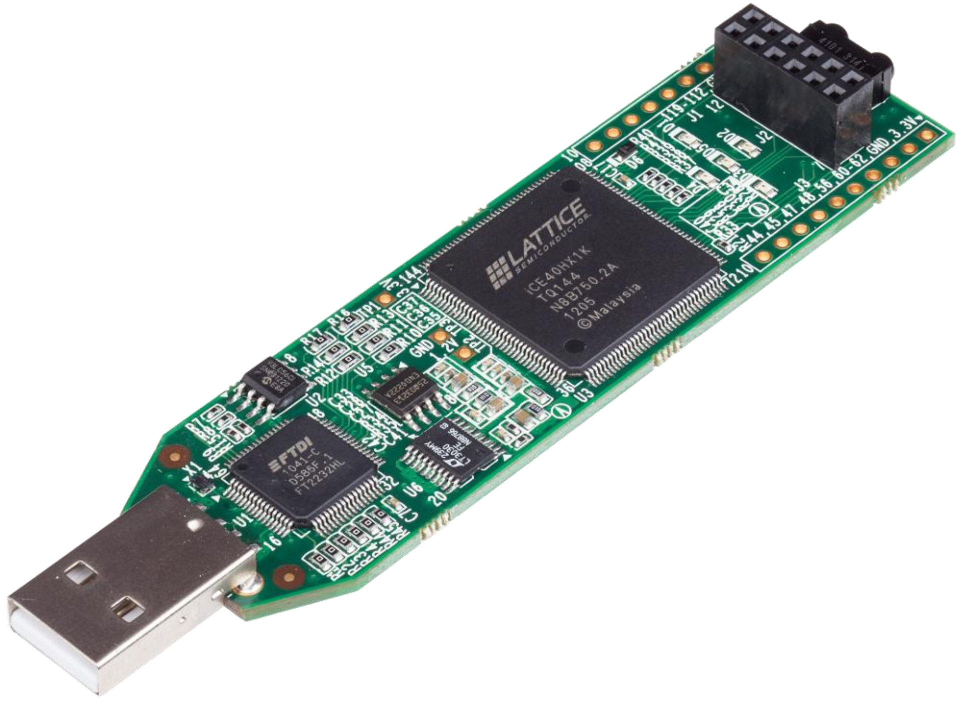


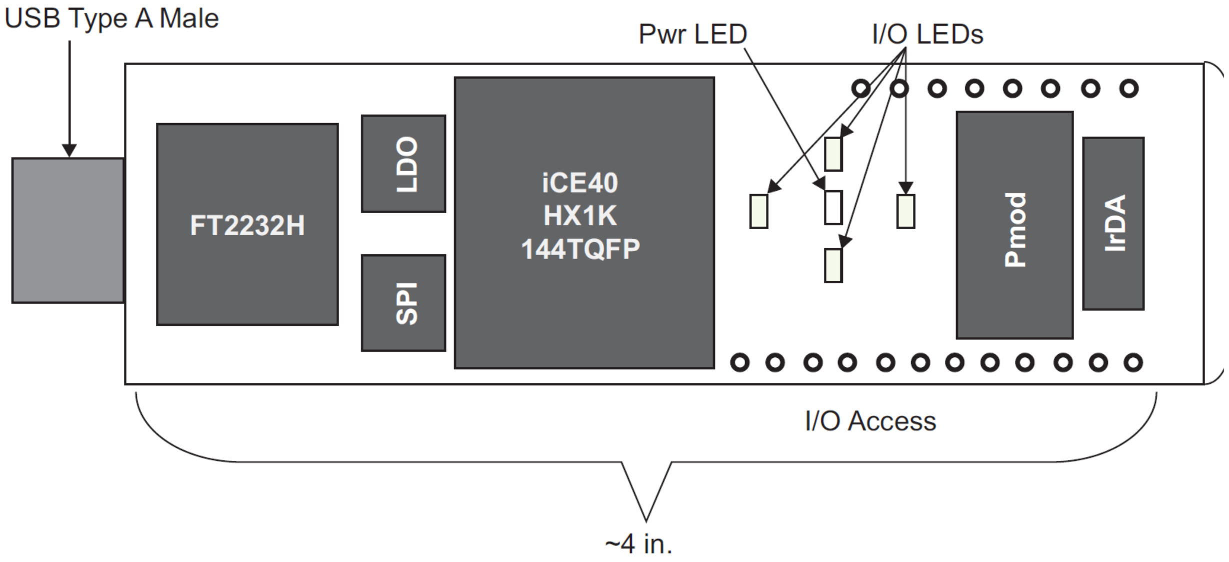
NextPNR & Lattice iCE40 FPGA Board

- Github: https://github.com/YosysHQ/nextpnr
- Lattice iCE40 FPGA Board: https://www.latticesemi.com/icestick

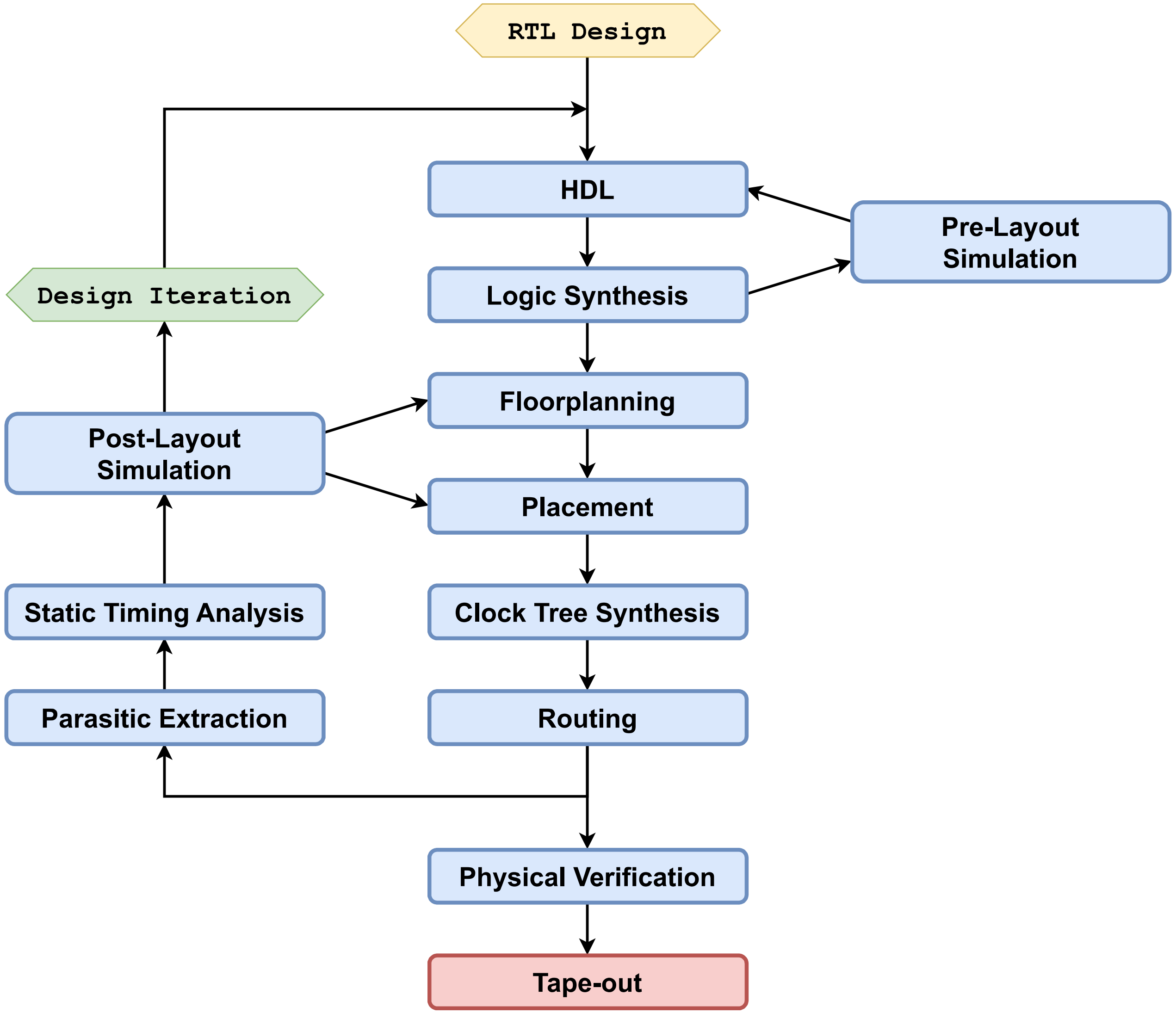
YosysHQ

LATTICE SEMICONDUCTOR



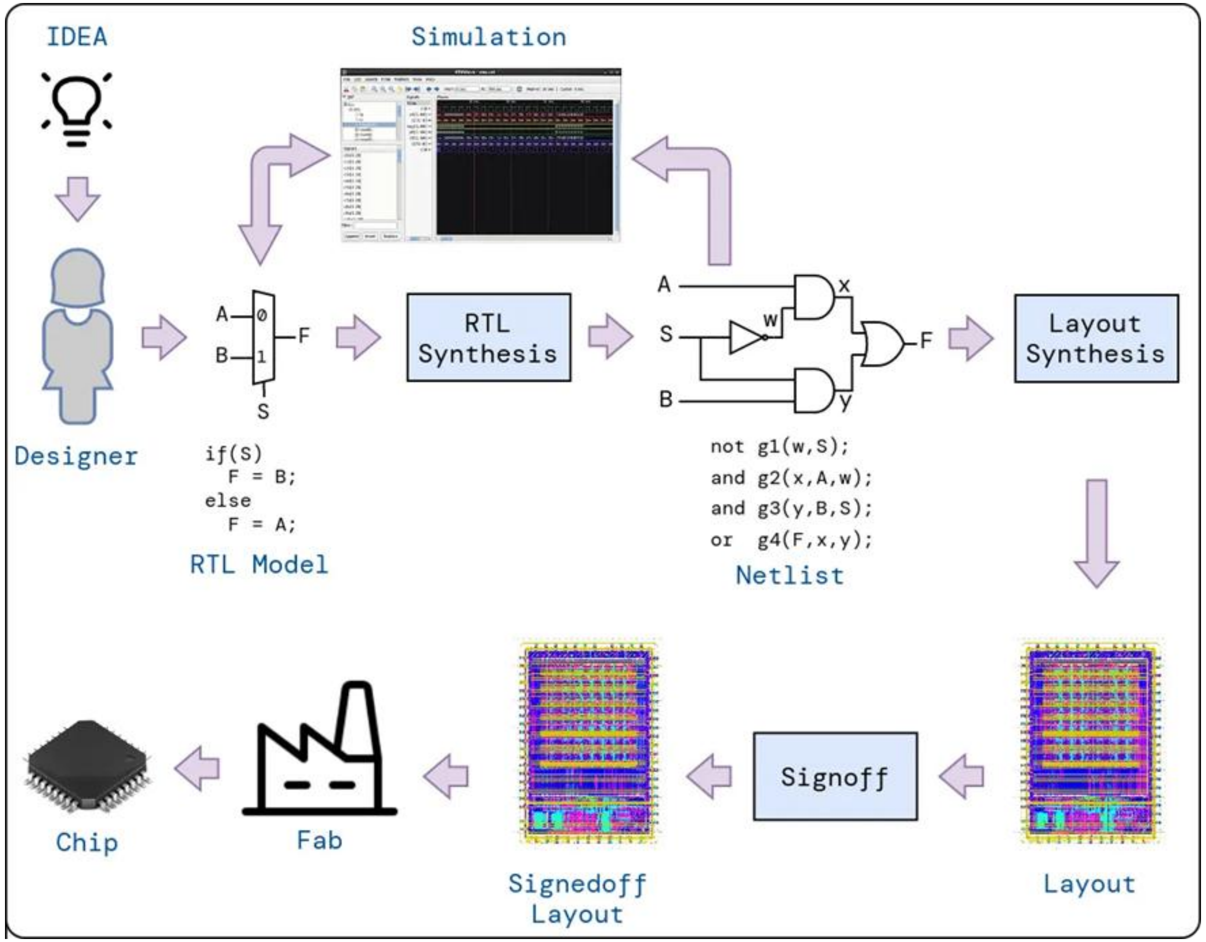


RTL to GDS Flow



LibreLane – Intro

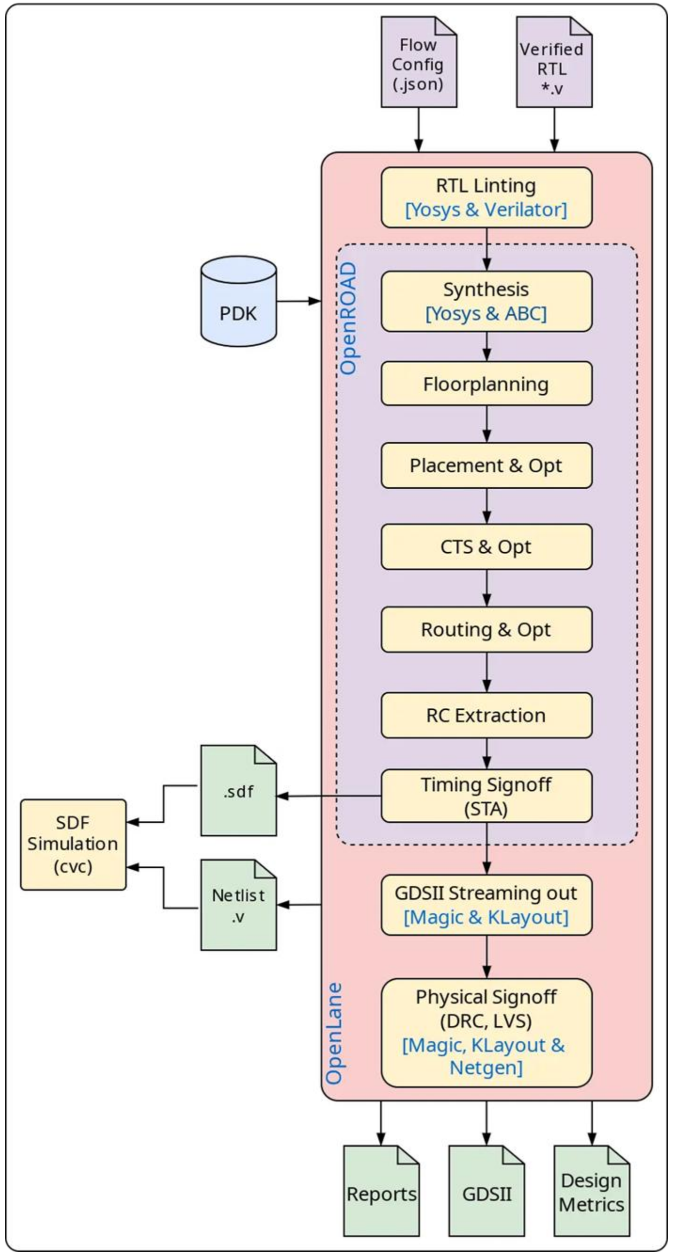
- Github: https://github.com/librelane/librelane
- Documentation: https://librelane.readthedocs.io/en/latest/



```
if(S)
  F = B;
else
  F = A;
```

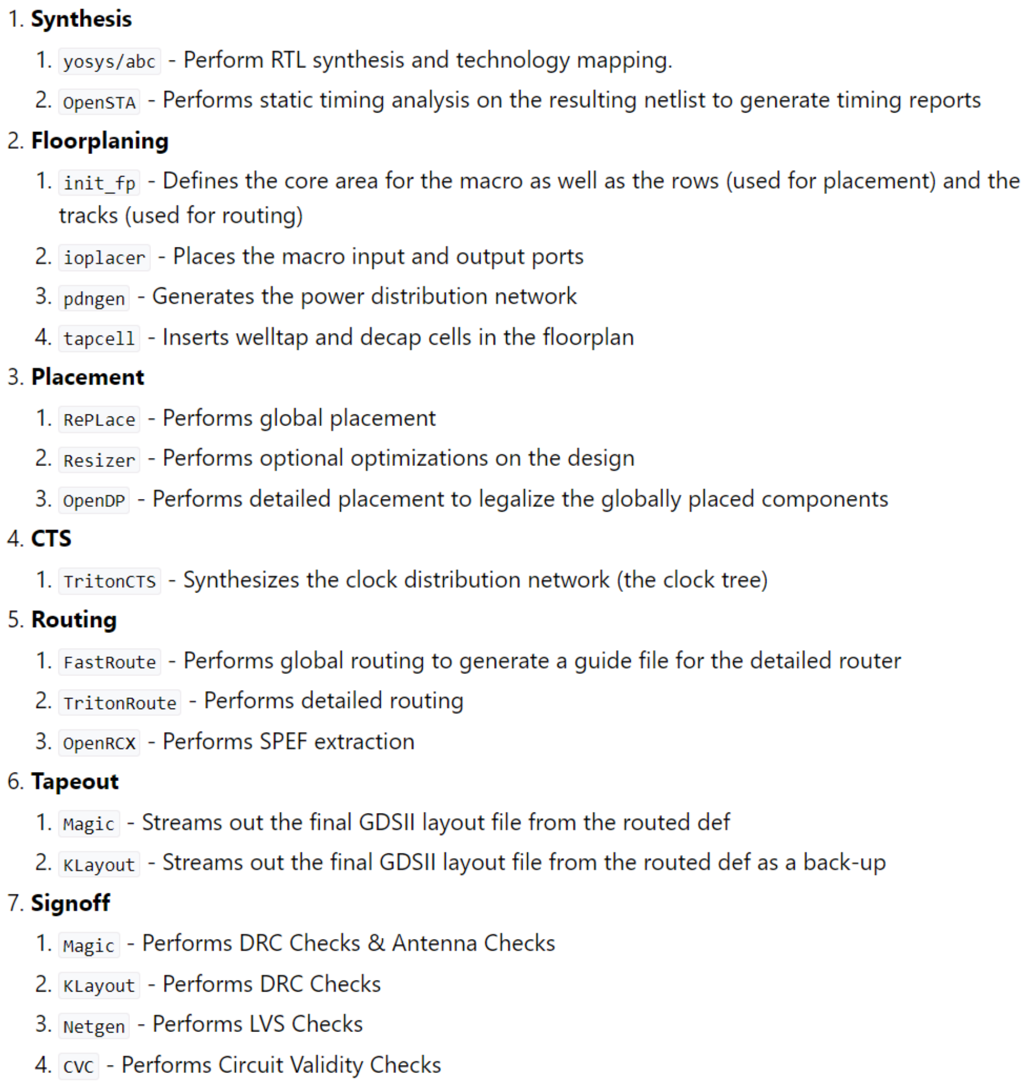
```
not g1(w,S);
and g2(x,A,w);
and g3(y,B,S);
or  g4(F,x,y);
```

LibreLane – Flow Diagram

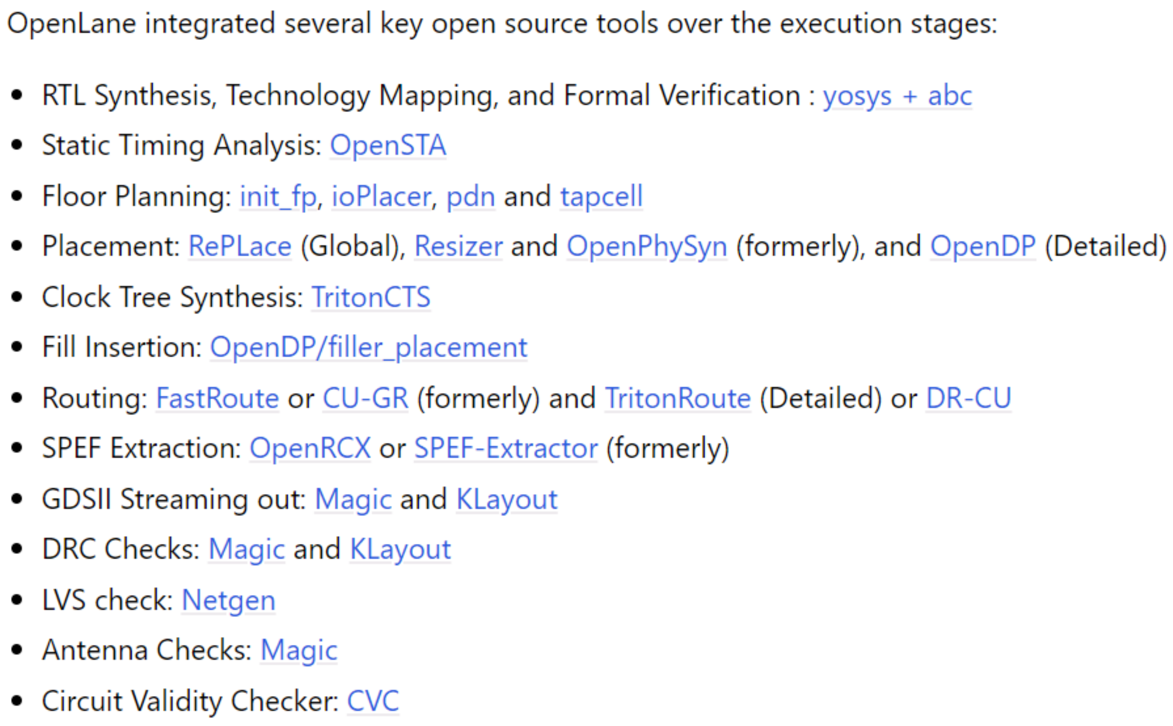


JOHANNES KEPLER UNIVERSITÄT LINZ

LibreLane – Design Stages



1. **Synthesis**
   1. `yosys/abc` - Perform RTL synthesis and technology mapping.
   2. `OpenSTA` - Performs static timing analysis on the resulting netlist to generate timing reports
2. **Floorplaning**
   1. `init_fp` - Defines the core area for the macro as well as the rows (used for placement) and the tracks (used for routing)
   2. `ioplacer` - Places the macro input and output ports
   3. `pdngen` - Generates the power distribution network
   4. `tapcell` - Inserts welltap and decap cells in the floorplan
3. **Placement**
   1. `RePLace` - Performs global placement
   2. `Resizer` - Performs optional optimizations on the design
   3. `OpenDP` - Performs detailed placement to legalize the globally placed components
4. **CTS**
   1. `TritonCTS` - Synthesizes the clock distribution network (the clock tree)
5. **Routing**
   1. `FastRoute` - Performs global routing to generate a guide file for the detailed router
   2. `TritonRoute` - Performs detailed routing
   3. `OpenRCX` - Performs SPEF extraction
6. **Tapeout**
   1. `Magic` - Streams out the final GDSII layout file from the routed def
   2. `KLayout` - Streams out the final GDSII layout file from the routed def as a back-up
7. **Signoff**
   1. `Magic` - Performs DRC Checks & Antenna Checks
   2. `KLayout` - Performs DRC Checks
   3. `Netgen` - Performs LVS Checks
   4. `cvc` - Performs Circuit Validity Checks



OpenLane integrated several key open source tools over the execution stages:

- RTL Synthesis, Technology Mapping, and Formal Verification : yosys + abc
- Static Timing Analysis: OpenSTA
- Floor Planning: init_fp, ioPlacer, pdn and tapcell
- Placement: RePLace (Global), Resizer and OpenPhySyn (formerly), and OpenDP (Detailed)
- Clock Tree Synthesis: TritonCTS
- Fill Insertion: OpenDP/filler_placement
- Routing: FastRoute or CU-GR (formerly) and TritonRoute (Detailed) or DR-CU
- SPEF Extraction: OpenRCX or SPEF-Extractor (formerly)
- GDSII Streaming out: Magic and KLayout
- DRC Checks: Magic and KLayout
- LVS check: Netgen
- Antenna Checks: Magic
- Circuit Validity Checker: CVC

Installed Open-Source PDKs

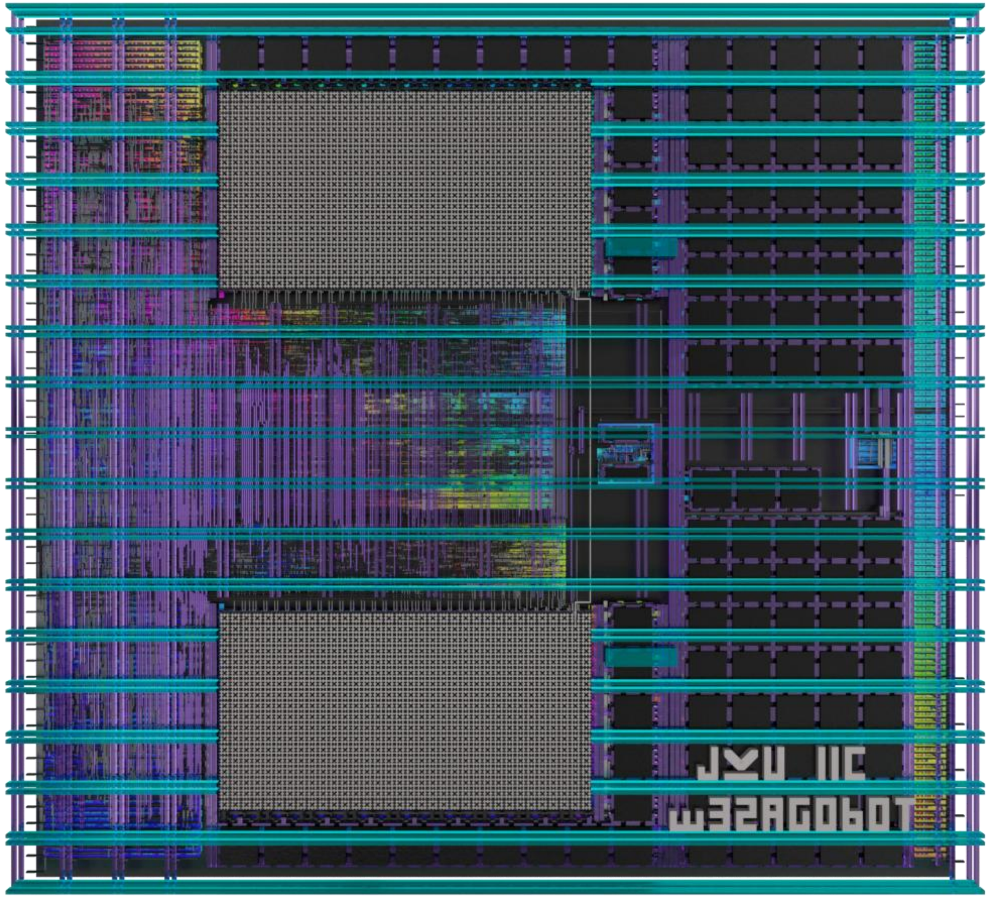
- **PDK:** Process-Design Kit
  - https://en.wikipedia.org/wiki/Process_design_kit
  - Set of files used within the semiconductor industry to model a fabrication process for the design tools used to design an IC
  - Symbols, Device Parameters
  - Simulation models for components (e.g. BSIM, PSP, …)
  - Design Rule Check (DRC), Layout vs. Schematic (LVS), Electrical Rule Check (ERC)
- IHP 130nm BiCMOS Technology (IHP130): https://www.ihp-microelectronics.com/de
- Skywater 130nm CMOS Technology (SKY130): https://www.skywatertechnology.com/
- Global Foundries 180nm CMOS Technology (GF180): https://gf.com/de/

Where to Tape out?

- **Tiny Tapeout:**
  - https://tinytapeout.com/
  - PDKs: SKY130 & IHP130
  - Multi-Project Chip (MPC), Multi-Project Wafer (MPW)
    - Cheap, since more projects are put on one chip
  - Development board
- **ChipFoundry:**
  - https://chipfoundry.io/
- **Europractice:**
  - https://europractice-ic.com/services/fabrication/
  - Supports also commercial PDKs (e.g. TSMC)

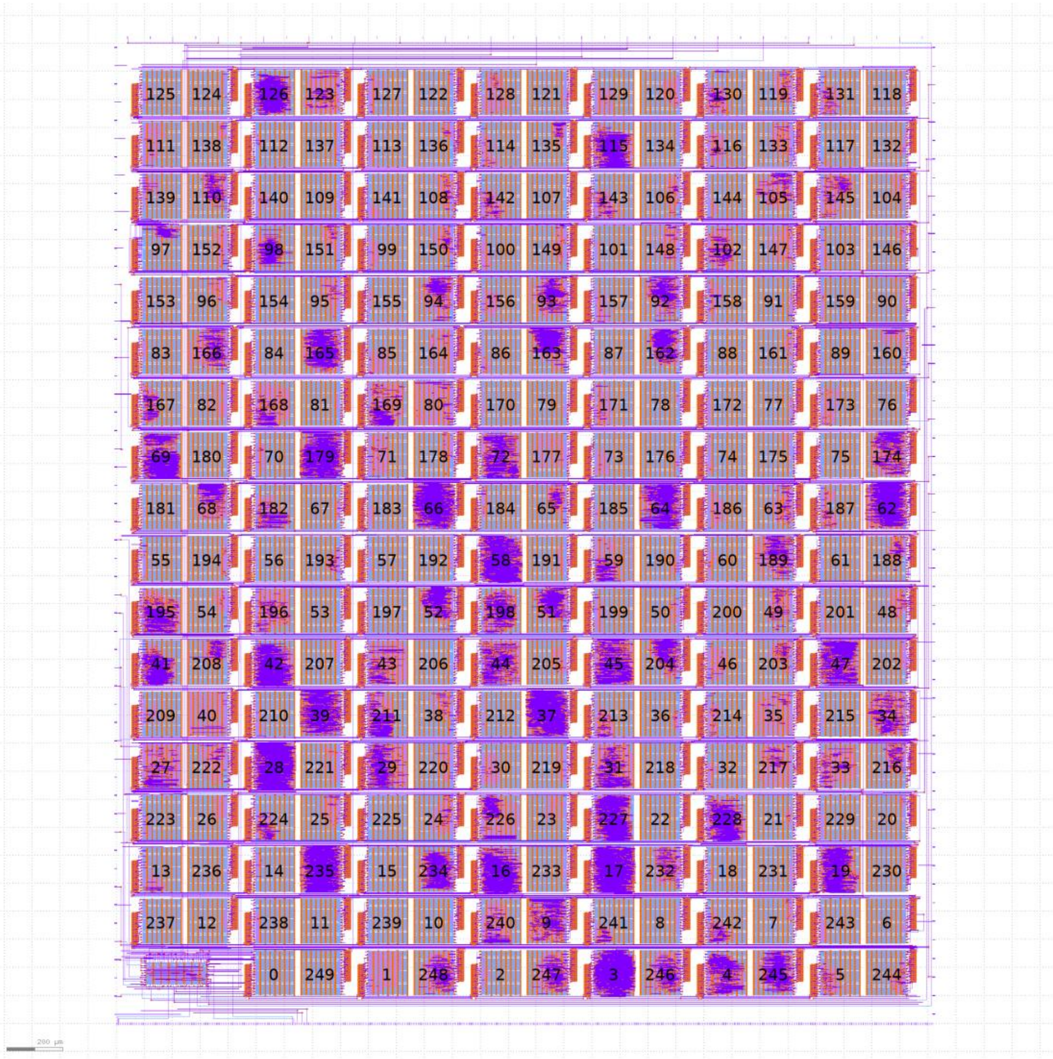
Examples – IICQC

- SAR-ADC: https://github.com/iic-jku/SKY130_SAR-ADC1
- Synthesizable Digital Temperature Sensor: https://github.com/iic-jku/tt03-tempsensor



Examples – Tiny Tapeout

- Tiny Tapeout 3 projects: https://tinytapeout.com/runs/tt03/

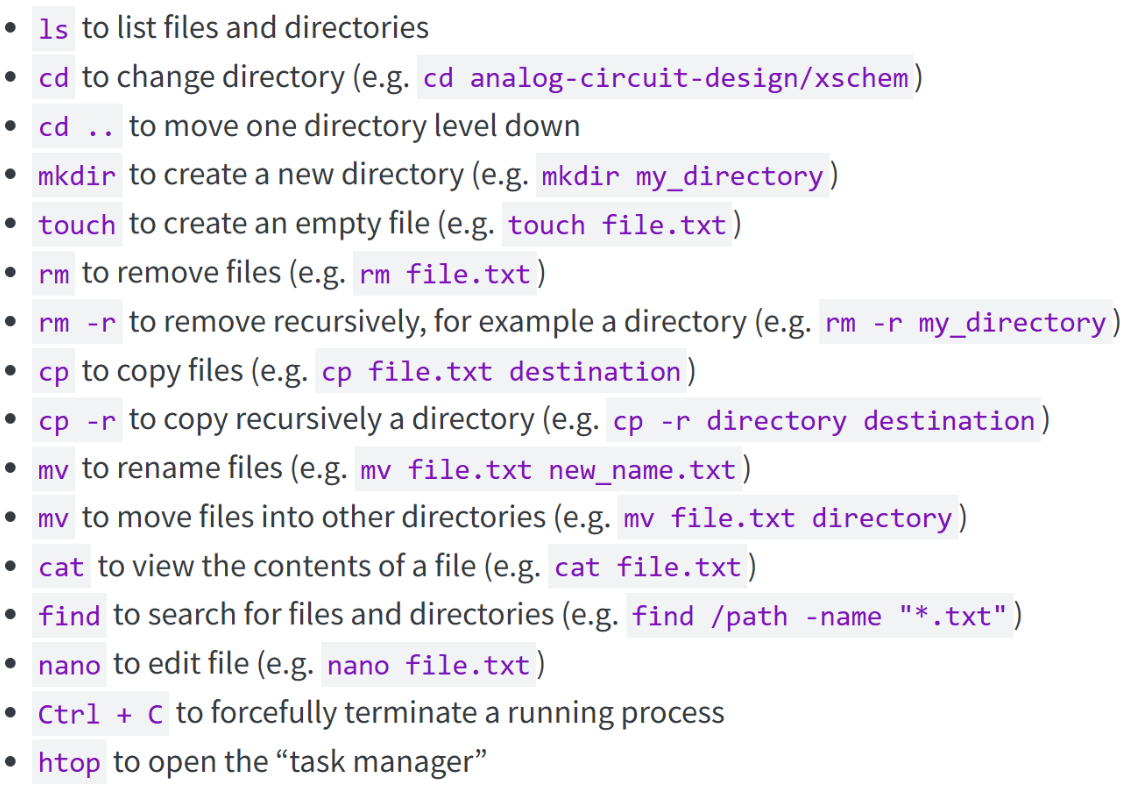


Tutorial – Setting up Docker container (WIN11)

- Install Docker: https://docs.docker.com/desktop/install/windows-install/
- Download IIC-OSIC-TOOLS: https://github.com/iic-jku/iic-osic-tools
- Start Docker & enter “*.\start_x.bat*” in command line in IIC-OSIC-TOOLS folder
- Set up “*.designinit*” file for used PDK

Tutorial – Linux Cheatsheet

- The most useful commands for the Linux command line are:



  - `ls` to list files and directories
  - `cd` to change directory (e.g. `cd analog-circuit-design/xschem`)
  - `cd ..` to move one directory level down
  - `mkdir` to create a new directory (e.g. `mkdir my_directory`)
  - `touch` to create an empty file (e.g. `touch file.txt`)
  - `rm` to remove files (e.g. `rm file.txt`)
  - `rm -r` to remove recursively, for example a directory (e.g. `rm -r my_directory`)
  - `cp` to copy files (e.g. `cp file.txt destination`)
  - `cp -r` to copy recursively a directory (e.g. `cp -r directory destination`)
  - `mv` to rename files (e.g. `mv file.txt new_name.txt`)
  - `mv` to move files into other directories (e.g. `mv file.txt directory`)
  - `cat` to view the contents of a file (e.g. `cat file.txt`)
  - `find` to search for files and directories (e.g. `find /path -name "*.txt"`)
  - `nano` to edit file (e.g. `nano file.txt`)
  - `Ctrl + C` to forcefully terminate a running process
  - `htop` to open the “task manager”

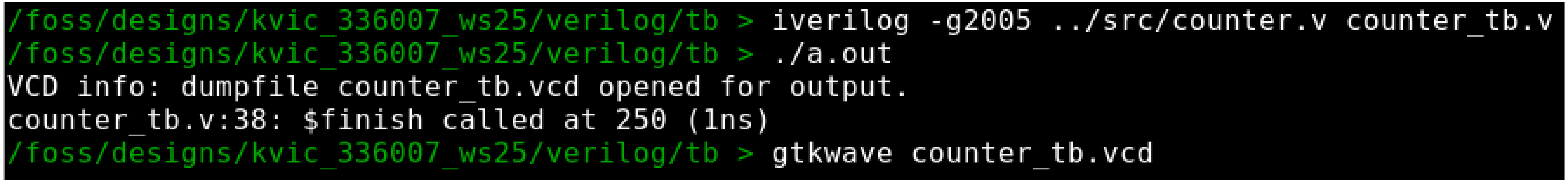
- More advanced commands can be found under https://www.geeksforgeeks.org/linux-commands-cheat-sheet

Tutorial – Recommended Workflow

1. Write Verilog Design
2. Simulate Verilog Design in **GTKWave / Surfer**
3. Synthesize Verilog Design with **Yosys**
4. Test Verilog Design on Lattice iCE40 FPGA board with **nextpnr (optional)**
5. GDS generation with **LibreLane** or **Tiny Tapeout GitHub Actions**
6. Take PDK-dependent Verilog netlist and simulate the design again (**Post-Layout Simulation**)

Tutorial – Compilation / Linting / Simulation

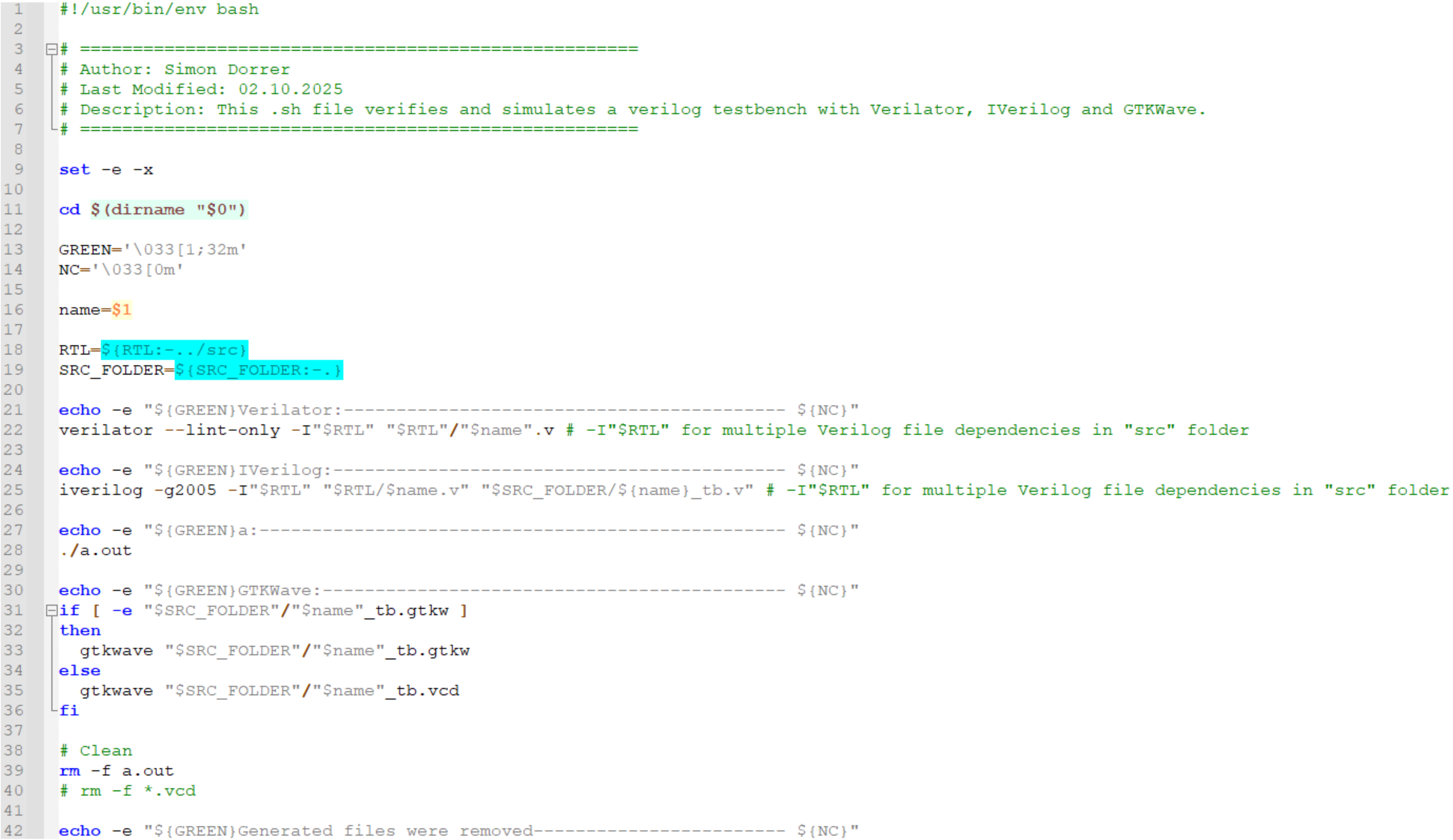
- If the architecture is finished, it should be compiled and checked for **syntax errors**.
- This can be done with the command *“iic-vlint.sh <file.v>“*
- After the architecture is without syntax errors, it should be simulated with its according **testbench**.



```
/foss/designs/kvic_336007_ws25/verilog/tb > iverilog -g2005 ../src/counter.v counter_tb.v
/foss/designs/kvic_336007_ws25/verilog/tb > ./a.out
VCD info: dumpfile counter_tb.vcd opened for output.
counter_tb.v:38: $finish called at 250 (1ns)
/foss/designs/kvic_336007_ws25/verilog/tb > gtkwave counter_tb.vcd
```

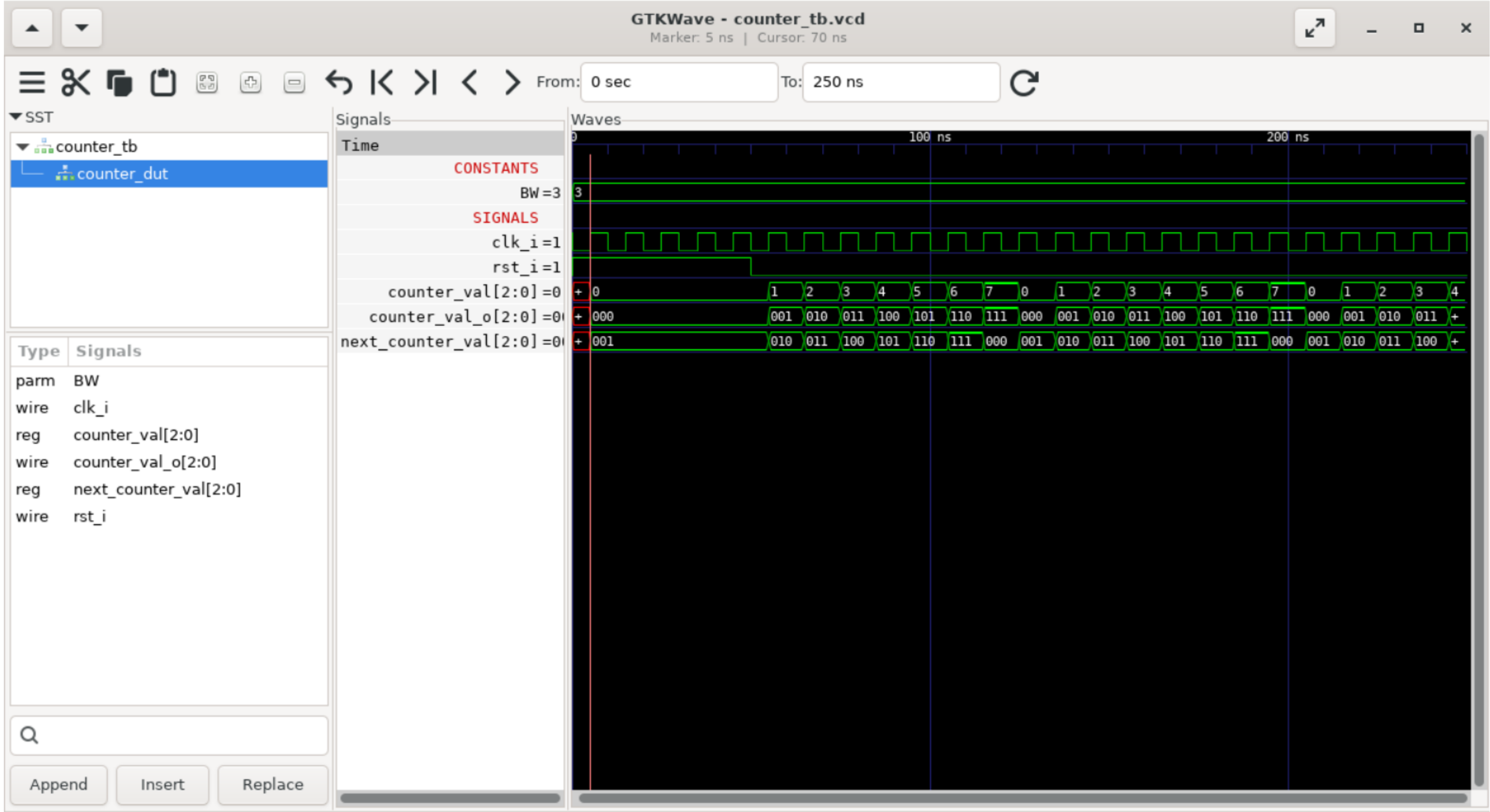
- With the self-written script, this can be done very easily, just call *“./simulate.sh <file>“*.
- After a simulation is set up it can be saved with CTRL+S and re-loaded with CTRL+O again.
- A simulation file is saved as *“<wave>.gtkw*“ by default.
- Alternatively, open Surfer with the “*surfer*” command and load *“<file.vcd>*”.

Tutorial – Compilation / Linting / Simulation



```bash
#!/usr/bin/env bash

# =====================================================
# Author: Simon Dorrer
# Last Modified: 02.10.2025
# Description: This .sh file verifies and simulates a verilog testbench with Verilator, IVerilog and GTKWave.
# =====================================================

set -e -x

cd $(dirname "$0")

GREEN='\033[1;32m'
NC='\033[0m'

name=$1

RTL=${RTL:-../src}
SRC_FOLDER=${SRC_FOLDER:-.}

echo -e "${GREEN}Verilator:------------------------------------------ ${NC}"
verilator --lint-only -I"$RTL" "$RTL"/"$name".v # -I"$RTL" for multiple Verilog file dependencies in "src" folder

echo -e "${GREEN}IVerilog:------------------------------------------- ${NC}"
iverilog -g2005 -I"$RTL" "$RTL/$name.v" "$SRC_FOLDER/${name}_tb.v" # -I"$RTL" for multiple Verilog file dependencies in "src" folder

echo -e "${GREEN}a:-------------------------------------------------- ${NC}"
./a.out

echo -e "${GREEN}GTKWave:-------------------------------------------- ${NC}"
if [ -e "$SRC_FOLDER"/"$name"_tb.gtkw ]
then
  gtkwave "$SRC_FOLDER"/"$name"_tb.gtkw
else
  gtkwave "$SRC_FOLDER"/"$name"_tb.vcd
fi

# Clean
rm -f a.out
# rm -f *.vcd

echo -e "${GREEN}Generated files were removed----------------------- ${NC}"
```

Tutorial – GTKWave



GTKWave - counter_tb.vcd

Marker: 5 ns | Cursor: 70 ns

From: 0 sec To: 250 ns

SST

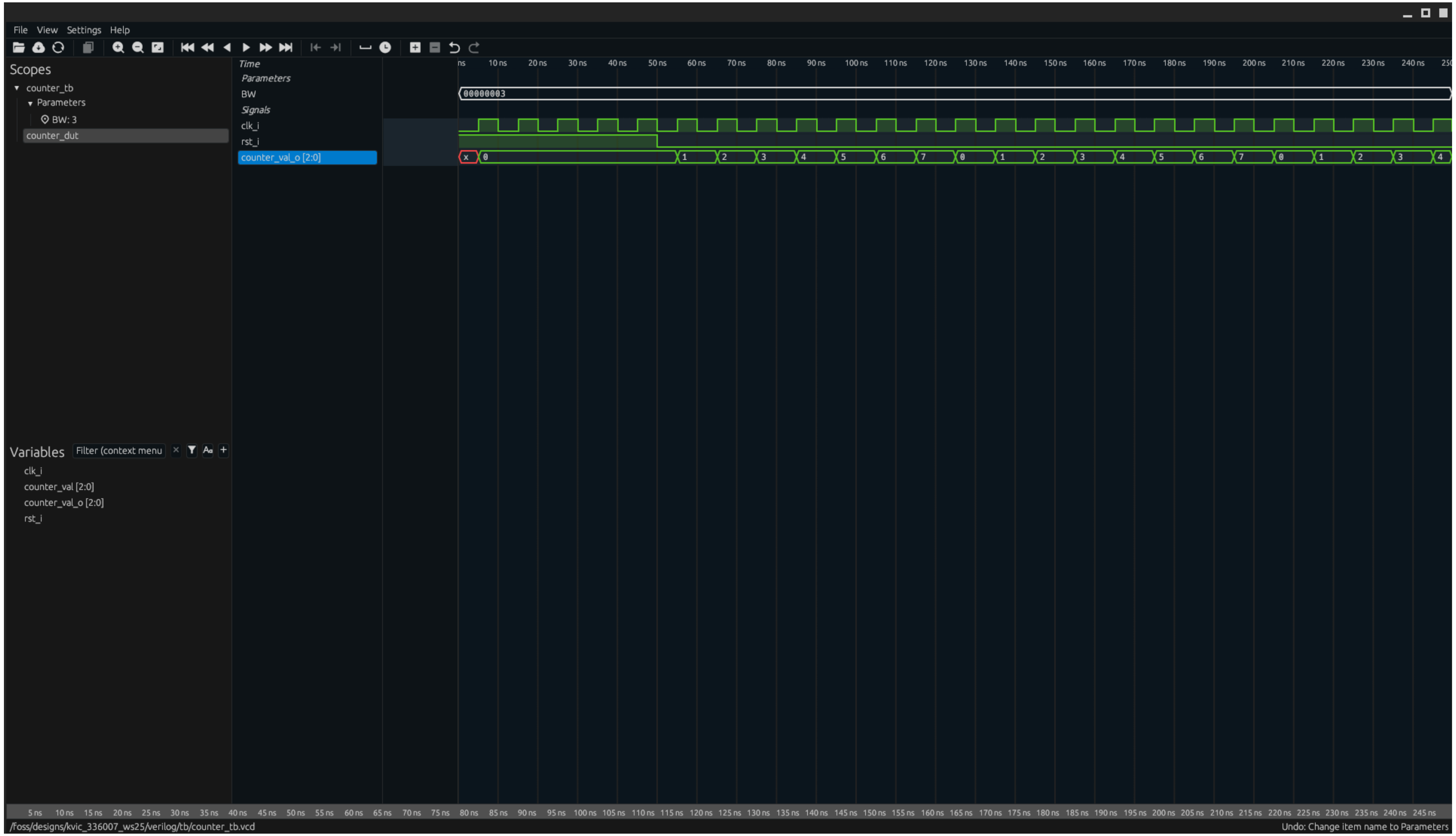
- counter_tb
  - counter_dut

Type	Signals
parm	BW
wire	clk_i
reg	counter_val[2:0]
wire	counter_val_o[2:0]
reg	next_counter_val[2:0]
wire	rst_i

Signals

```
Time
CONSTANTS
BW=3
SIGNALS
clk_i=1
rst_i=1
counter_val[2:0]=0
counter_val_o[2:0]=0(
next_counter_val[2:0]=0(
```

Tutorial – Surfer

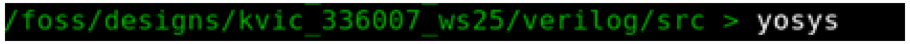


/foss/designs/kvic_336007_ws25/verilog/tb/counter_tb.vcd

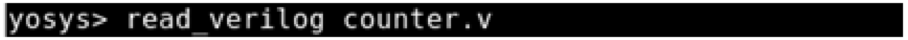
Undo: Change item name to Parameters

Tutorial – Showing Stats

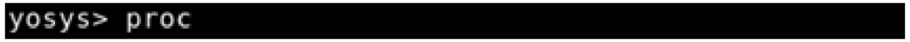
- To show the stats with “*yosys*” the following commands are needed.



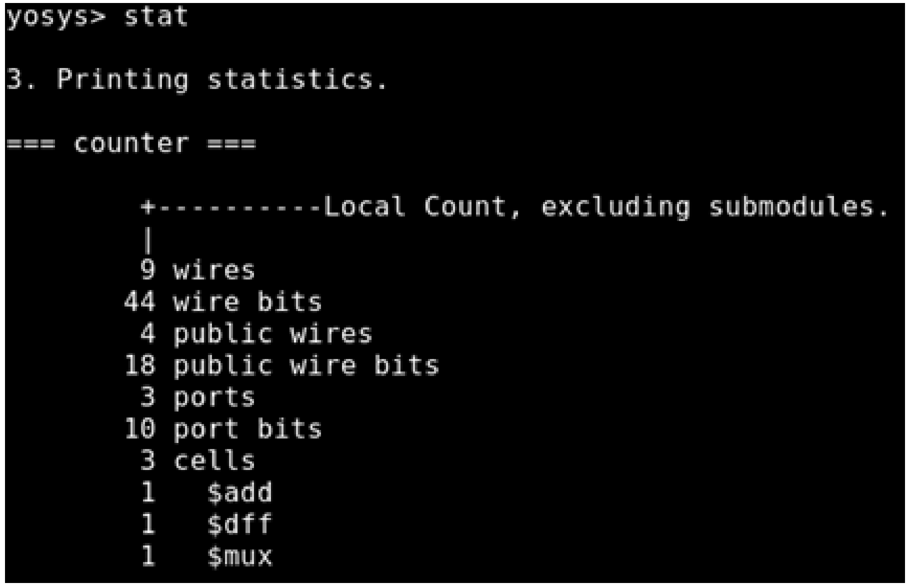
```
/foss/designs/kvic_336007_ws25/verilog/src > yosys
```



```
yosys> read_verilog counter.v
```



```
yosys> proc
```

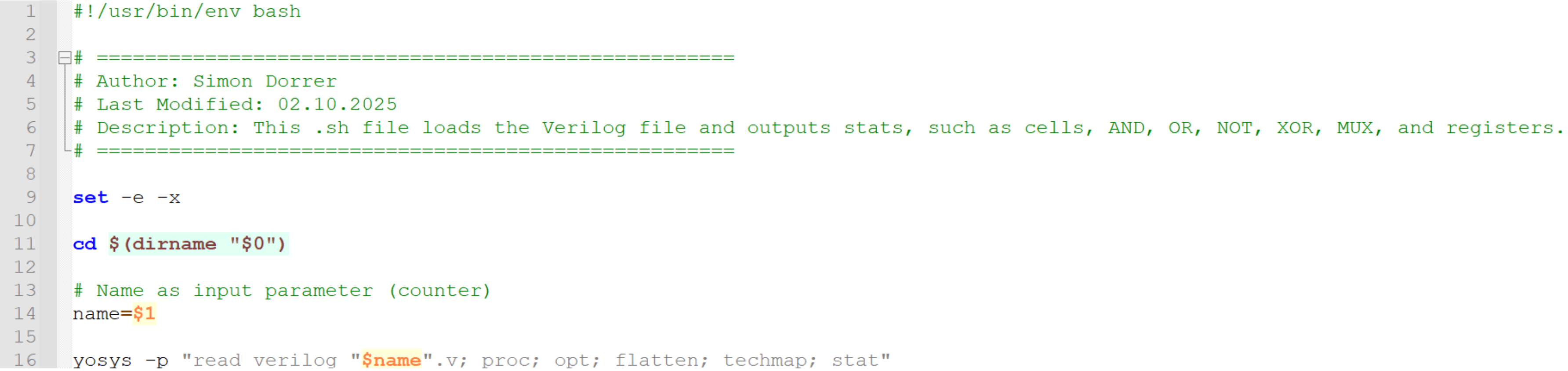


```
yosys> stat

3. Printing statistics.

=== counter ===

        +----------Local Count, excluding submodules.
        |
        9 wires
       44 wire bits
        4 public wires
       18 public wire bits
        3 ports
       10 port bits
        3 cells
        1   $add
        1   $dff
        1   $mux
```

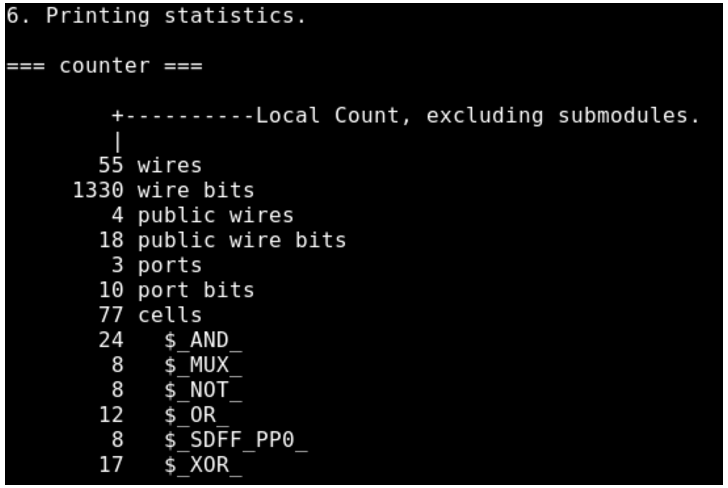
JOHANNES KEPLER
UNIVERSITÄT LINZ

Tutorial – Showing Stats

- To show the stats with “*yosys*” after “techmap” a script is provided.
- Just run <*./yosys_stats.sh counter*>

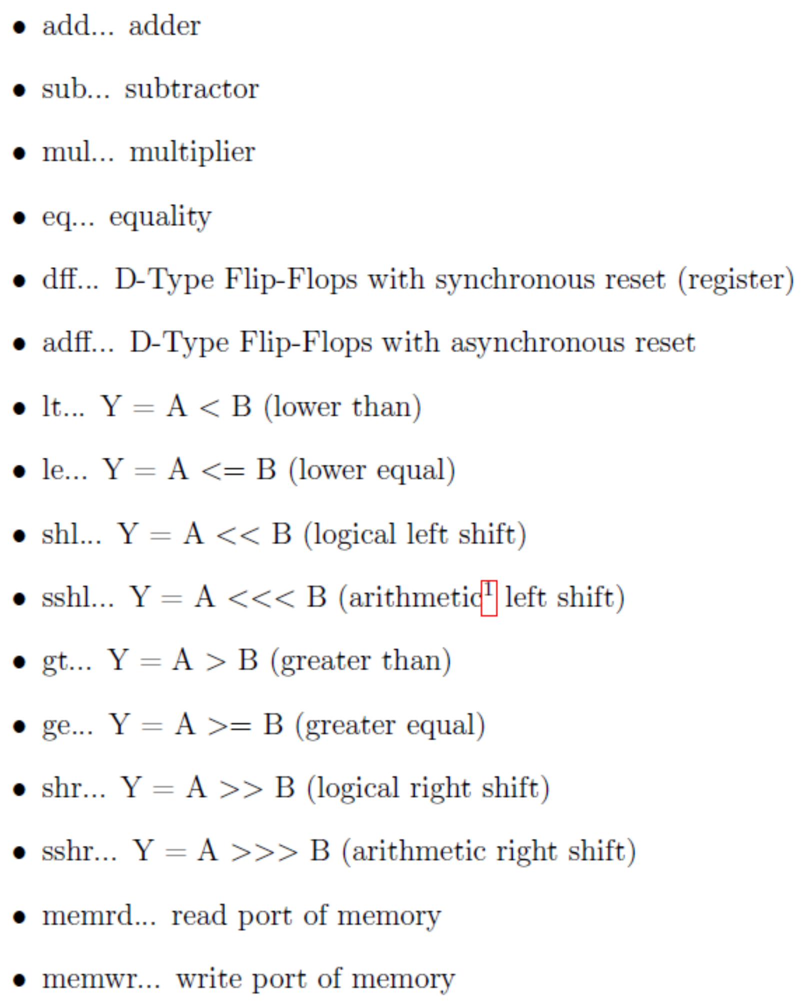


```
#!/usr/bin/env bash

# =====================================================
# Author: Simon Dorrer
# Last Modified: 02.10.2025
# Description: This .sh file loads the Verilog file and outputs stats, such as cells, AND, OR, NOT, XOR, MUX, and registers.
# =====================================================

set -e -x

cd $(dirname "$0")

# Name as input parameter (counter)
name=$1

yosys -p "read_verilog "$name".v; proc; opt; flatten; techmap; stat"
```



```
6. Printing statistics.

=== counter ===

        +----------Local Count, excluding submodules.
        |
       55 wires
     1330 wire bits
        4 public wires
       18 public wire bits
        3 ports
       10 port bits
       77 cells
       24   $_AND_
        8   $_MUX_
        8   $_NOT_
       12   $_OR_
        8   $_SDFF_PP0_
       17   $_XOR_
```

Tutorial – Showing Stats



- add... adder
- sub... subtractor
- mul... multiplier
- eq... equality
- dff... D-Type Flip-Flops with synchronous reset (register)
- adff... D-Type Flip-Flops with asynchronous reset
- lt... $Y = A < B$ (lower than)
- le... $Y = A <= B$ (lower equal)
- shl... $Y = A << B$ (logical left shift)
- sshl... $Y = A <<< B$ (arithmetic[1] left shift)
- gt... $Y = A > B$ (greater than)
- ge... $Y = A >= B$ (greater equal)
- shr... $Y = A >> B$ (logical right shift)
- sshr... $Y = A >>> B$ (arithmetic right shift)
- memrd... read port of memory
- memwr... write port of memory

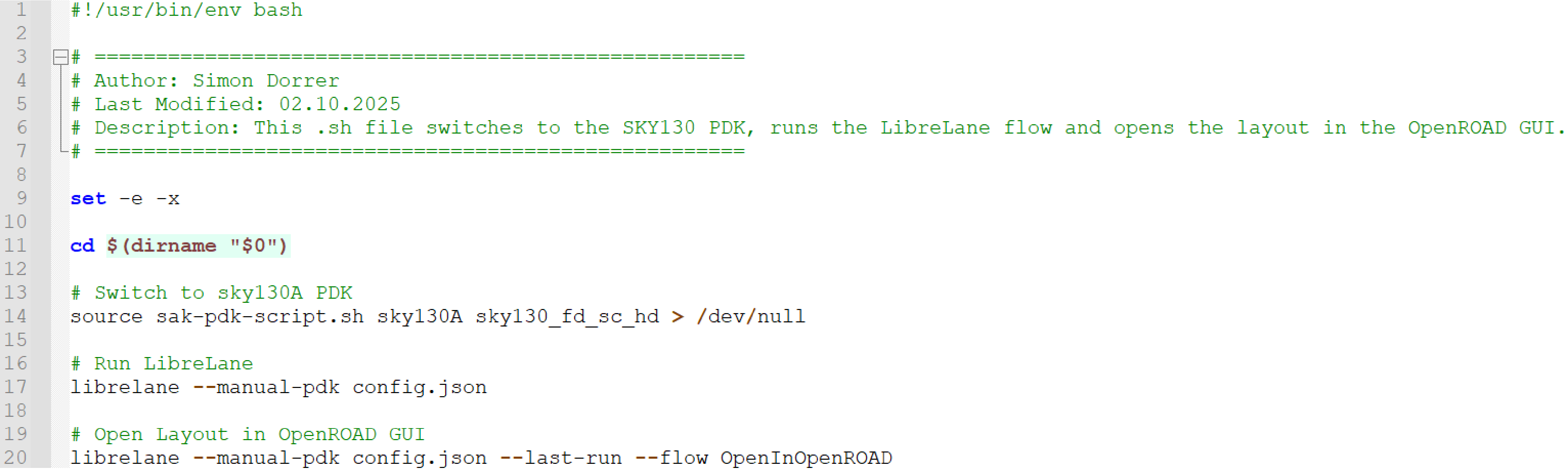
- The $memwr cells have a clock input CLK, an enable input EN (one enable bit for each data bit), an address input ADDR and a data input DATA.
- The $memrd cells have a clock input CLK, an enable input EN, an address input ADDR and a data output DATA.
- More informations can be found in the yosys manual, just search for „$abbreviation_name“.

Tutorial – LibreLane

- In the „config.json“ file the Tiny Tapeout parameters for the digital layout are set.
  - CLOCK_PERIOD, CLOCK_PORT, …
  - DIE_AREA, PL_TARGET_DENSITY_PCT, …
- In order to create the layout, one must navigate to the folder with the config file and execute the command “*librelane --manual-pdk config.json*”.
- After some minutes / hours, the design should be created, and it can be viewed in 2D with the tools „***magic***“ or „*klayout*“ or in 3D with “*GDS3D*” or the **Tiny Tapeout GitHub actions**.
- Alternatively, the built-in **OpenROAD GUI** can be used.
- More information (HeiChips 2025 LibreLane Workshop): https://github.com/FPGA-Research/heichips25-workshop?tab=readme-ov-file

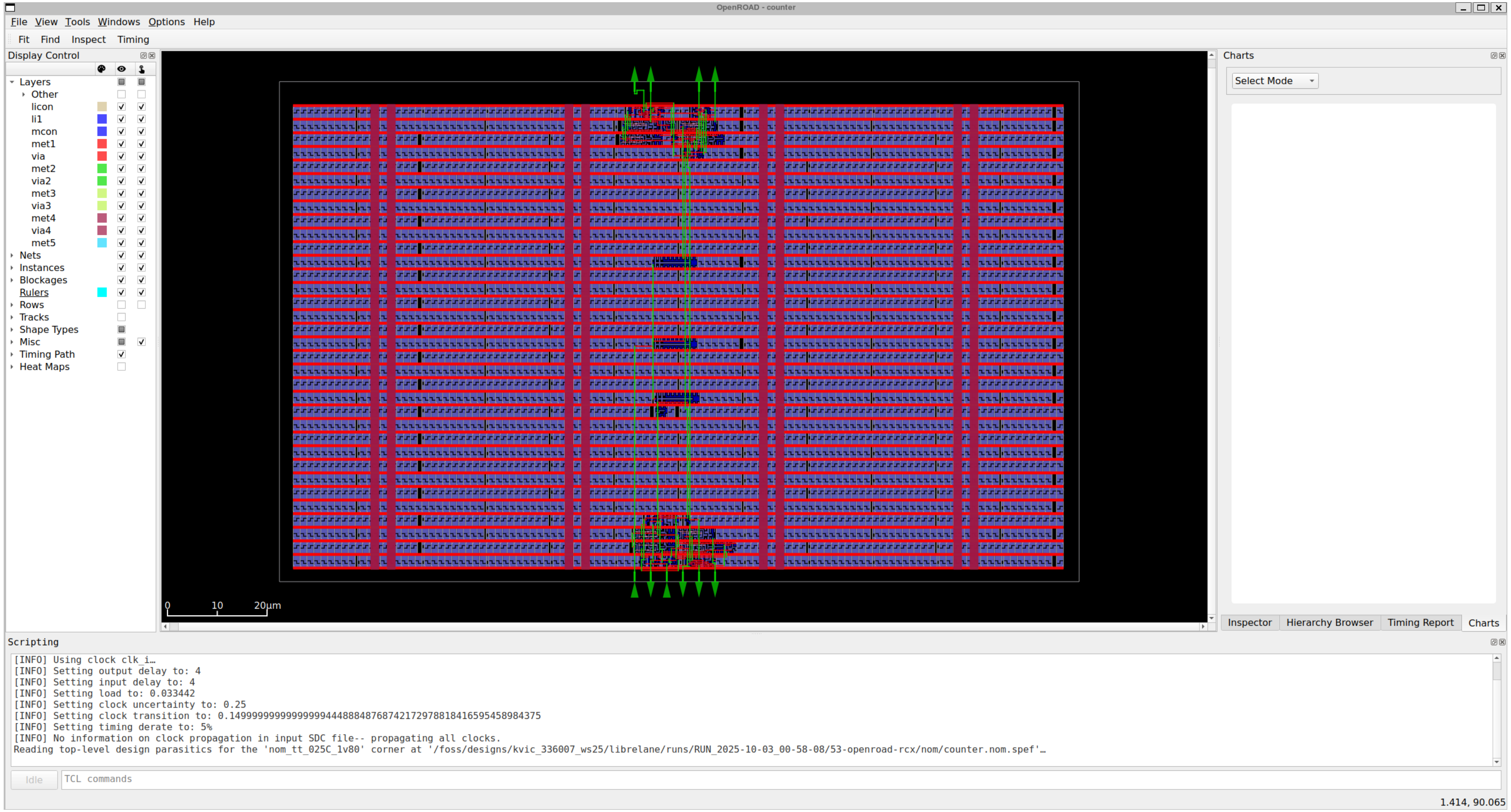
Tutorial – LibreLane

- To switch to the sky130A PDK, run the LibreLane flow and open the layout in the OpenROAD GUI, an additional script is provided.
- Just run *<./run_librelane.sh>*



```bash
#!/usr/bin/env bash

# =====================================================
# Author: Simon Dorrer
# Last Modified: 02.10.2025
# Description: This .sh file switches to the SKY130 PDK, runs the LibreLane flow and opens the layout in the OpenROAD GUI.
# =====================================================

set -e -x

cd $(dirname "$0")

# Switch to sky130A PDK
source sak-pdk-script.sh sky130A sky130_fd_sc_hd > /dev/null

# Run LibreLane
librelane --manual-pdk config.json

# Open Layout in OpenROAD GUI
librelane --manual-pdk config.json --last-run --flow OpenInOpenROAD
```

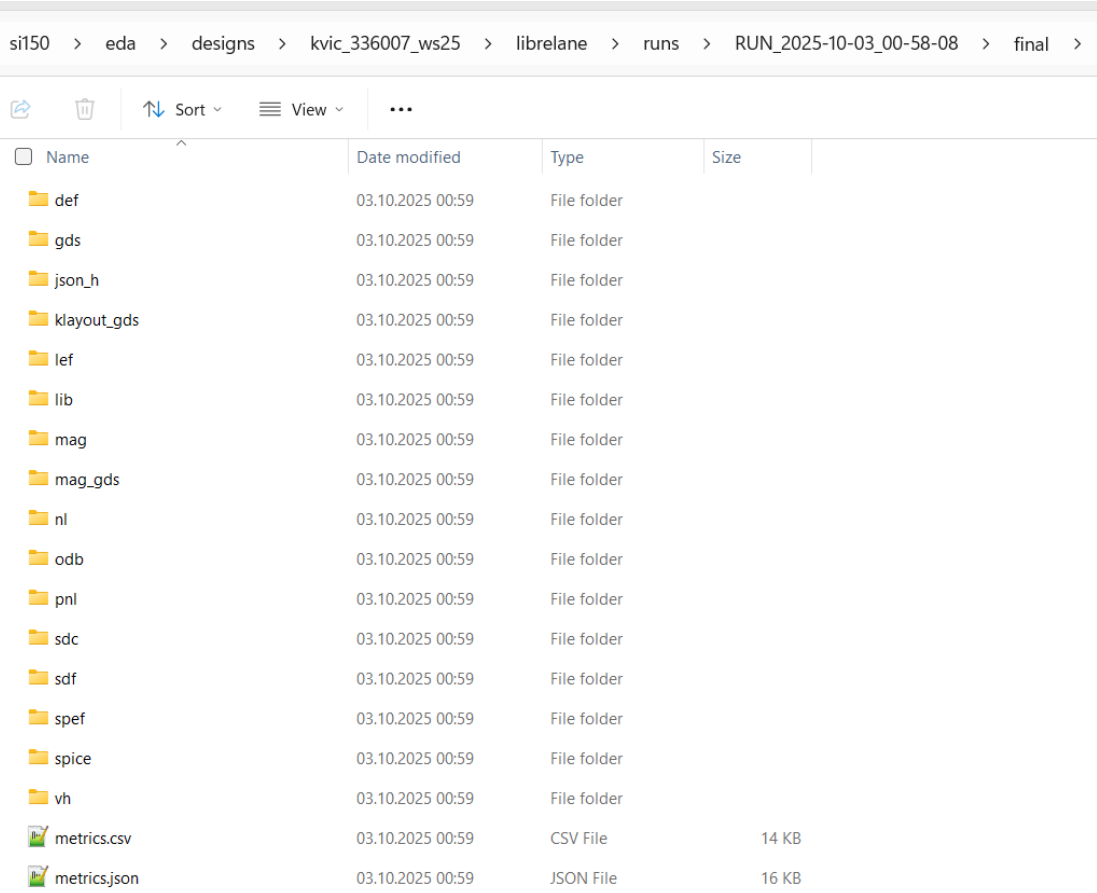
Tutorial – LibreLane – OpenROAD GUI



```
[INFO] Using clock clk_i…
[INFO] Setting output delay to: 4
[INFO] Setting input delay to: 4
[INFO] Setting load to: 0.033442
[INFO] Setting clock uncertainty to: 0.25
[INFO] Setting clock transition to: 0.1499999999999999944488848768742172978818416595458984375
[INFO] Setting timing derate to: 5%
[INFO] No information on clock propagation in input SDC file-- propagating all clocks.
Reading top-level design parasitics for the 'nom_tt_025C_1v80' corner at '/foss/designs/kvic_336007_ws25/librelane/runs/RUN_2025-10-03_00-58-08/53-openroad-rcx/nom/counter.nom.spef'…
```

Tutorial – LibreLane

- All output data is placed in *“librelane/runs/ RUN_XYZ/final”* by default:



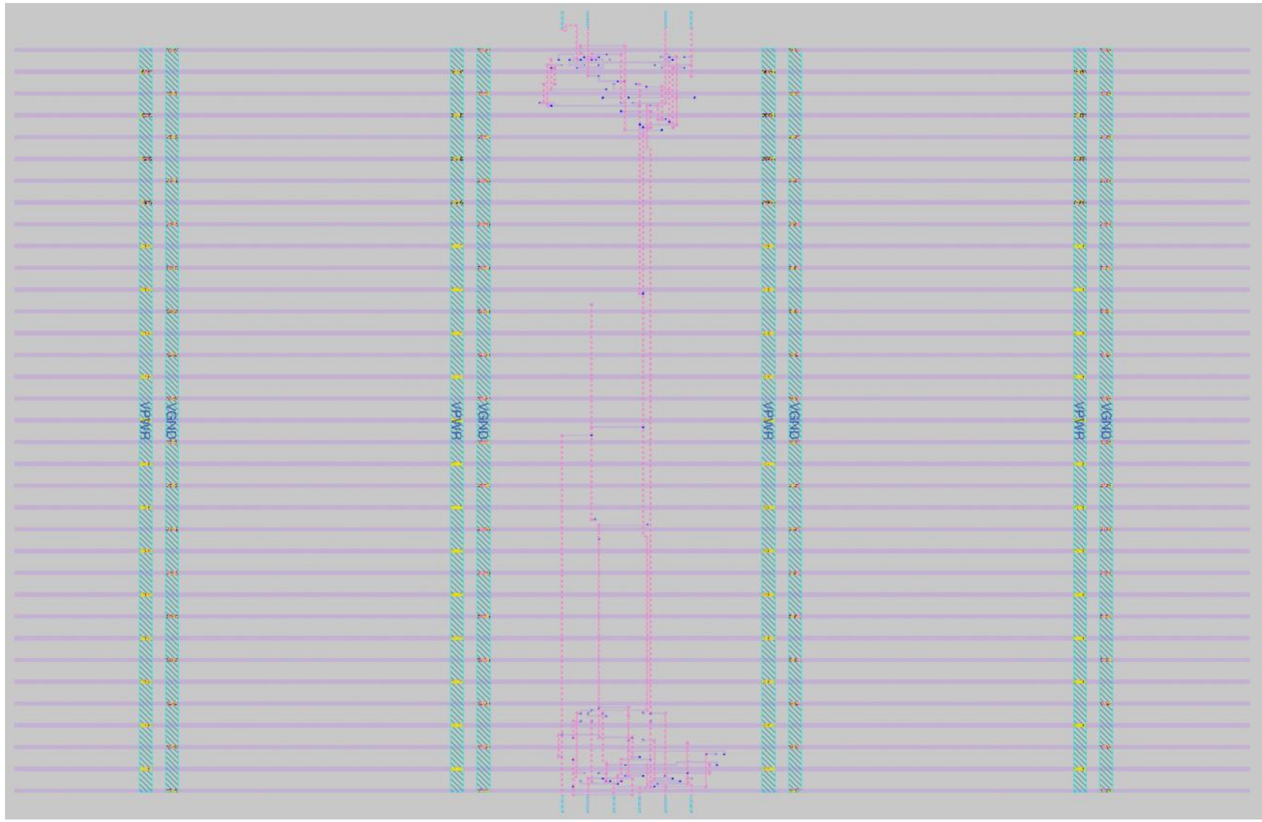
si150 › eda › designs › kvic_336007_ws25 › librelane › runs › RUN_2025-10-03_00-58-08 › final ›

Name	Date modified	Type	Size
def	03.10.2025 00:59	File folder	
gds	03.10.2025 00:59	File folder	
json_h	03.10.2025 00:59	File folder	
klayout_gds	03.10.2025 00:59	File folder	
lef	03.10.2025 00:59	File folder	
lib	03.10.2025 00:59	File folder	
mag	03.10.2025 00:59	File folder	
mag_gds	03.10.2025 00:59	File folder	
nl	03.10.2025 00:59	File folder	
odb	03.10.2025 00:59	File folder	
pnl	03.10.2025 00:59	File folder	
sdc	03.10.2025 00:59	File folder	
sdf	03.10.2025 00:59	File folder	
spef	03.10.2025 00:59	File folder	
spice	03.10.2025 00:59	File folder	
vh	03.10.2025 00:59	File folder	
metrics.csv	03.10.2025 00:59	CSV File	14 KB
metrics.json	03.10.2025 00:59	JSON File	16 KB

- Furthermore, each flow cycle will output a folder in *“librelane/runs/RUN_XYZ”.*

Tutorial – Magic

- Go to “*librelane/runs/RUN_XYZ/results/final/mag”* & enter “*magic counter.mag*”
- Please see the “magic_cheatsheet.pdf” for Magic commands!



- **For a nice 3D view, use the GDS viewer of the Tiny Tapeout GitHub actions!**

JKU

JOHANNES KEPLER
UNIVERSITY LINZ