

# Introduction to Open-Source Tools



# Free and Open-Source Software (FOSS)

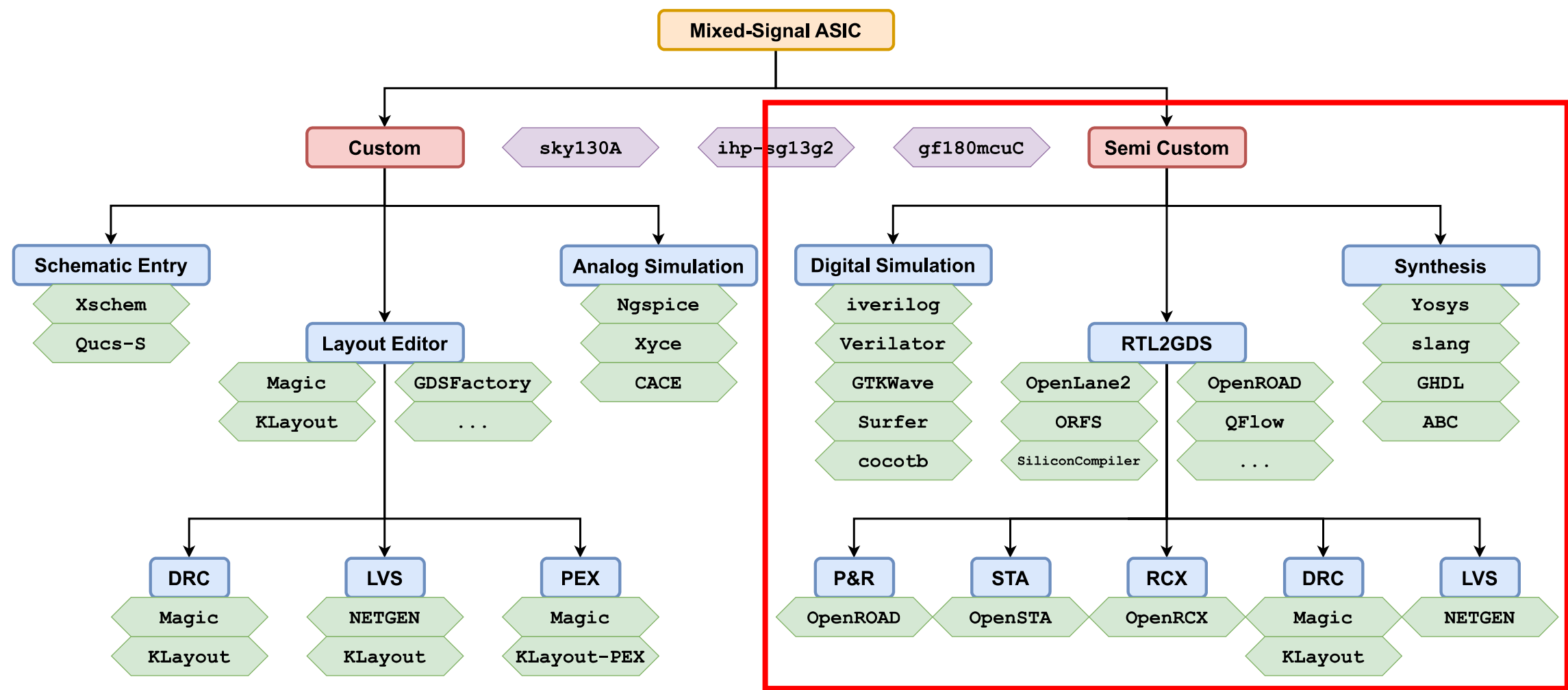
- **Verilator:** Verilog code linter
- **Icarus Verilog (iVerilog):** Verilog compiler and simulator
- **GKTWave / Surfer:** Waveform viewer
- **GHDL:** VHDL analyzer, compiler, simulator and synthesizer
- **YOSYS / ABC:** Verilog synthesis tool
- **Magic / KLayout:** Layout editor with DRC and PEX
- **NextPNR:** Portable FPGA place and route tool
- **LibreLane:** Digital RTL-to-GDS flow
- **IIC-OSIC-Tools:**

- <https://github.com/iic-jku/iic-osic-tools>

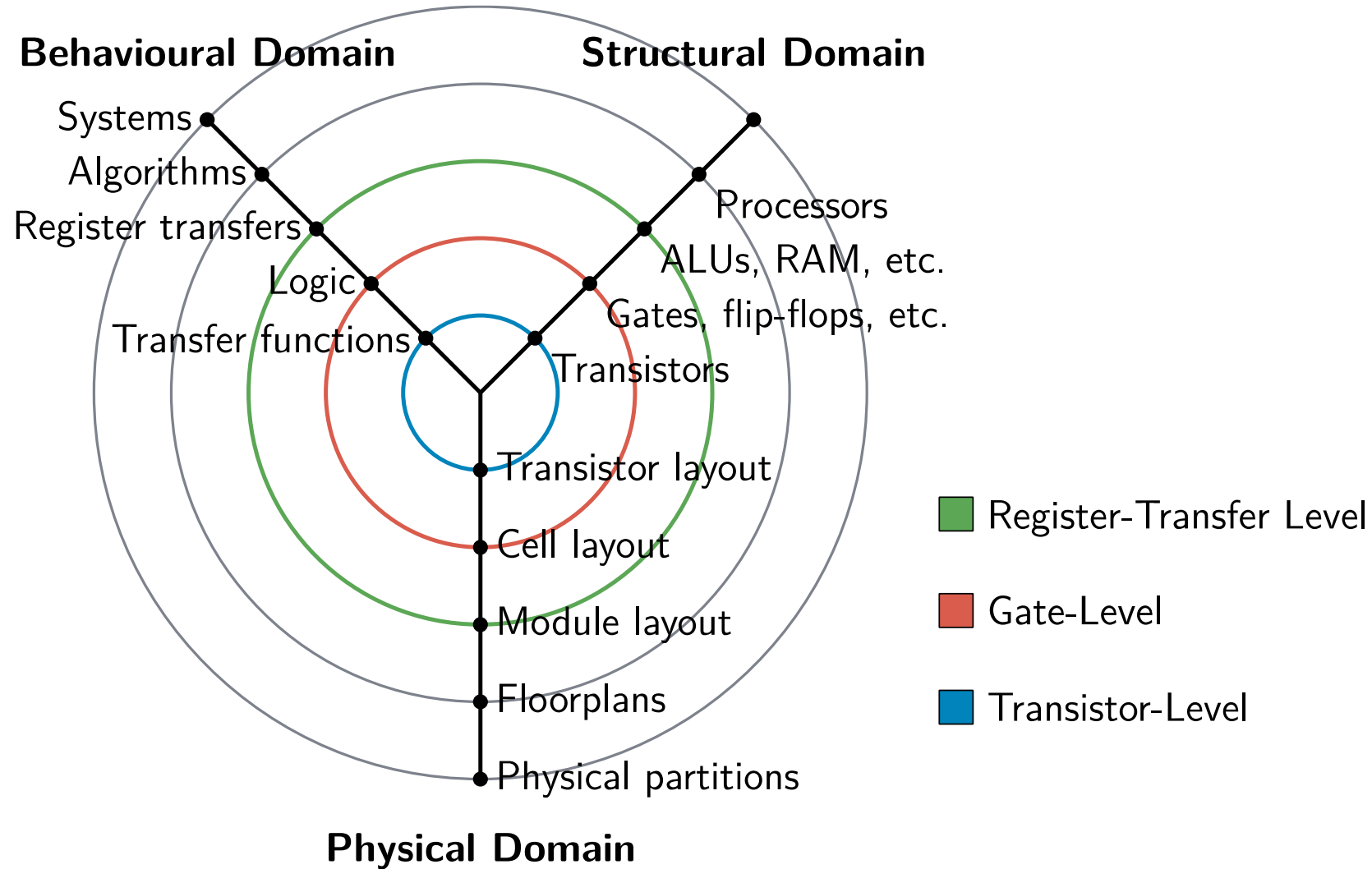
- All-in-one Docker container for FOSS based IC designs for analog and digital circuit flows



# IIC-OSIC-TOOLS

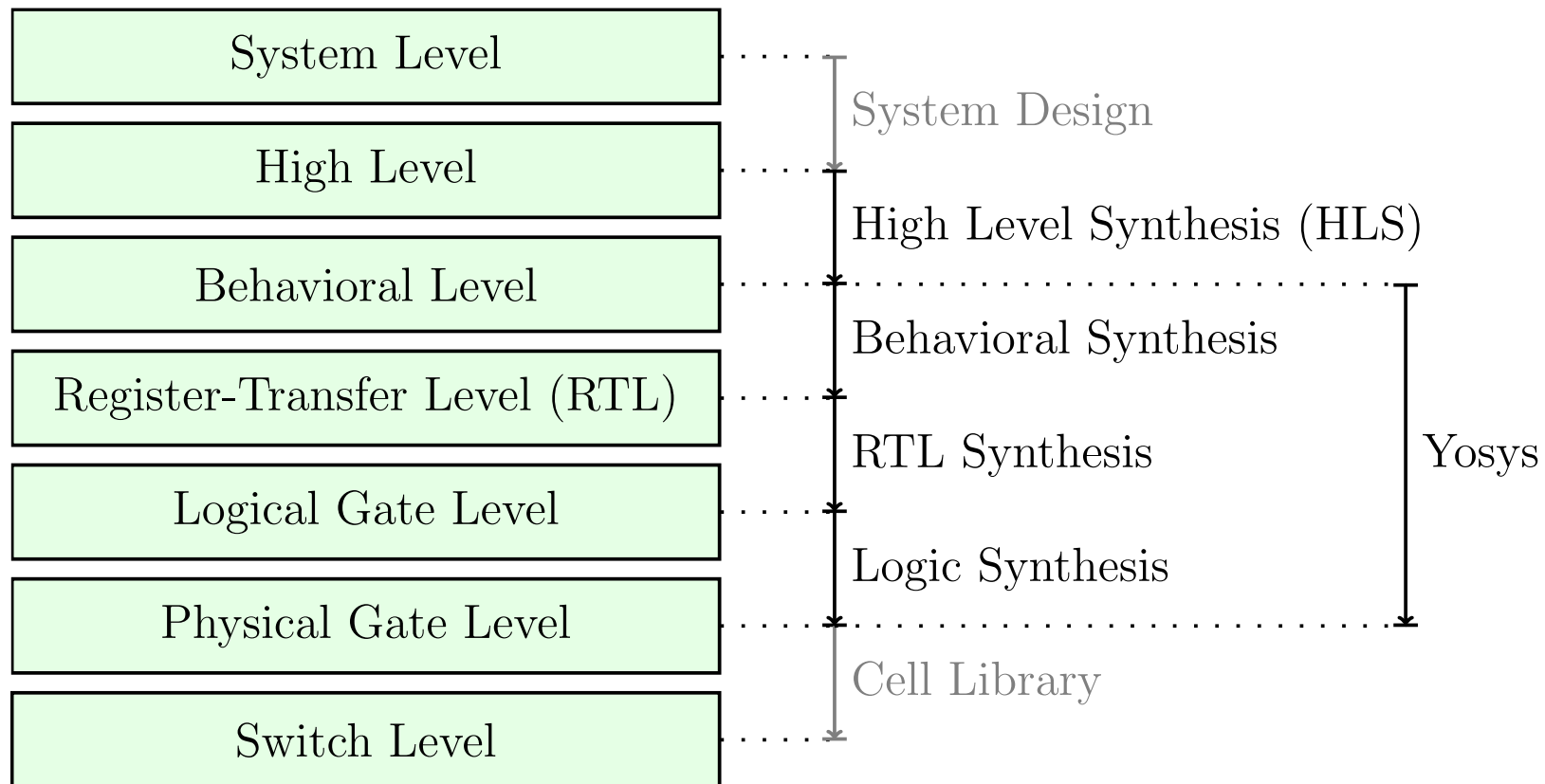


# Y-Diagramm



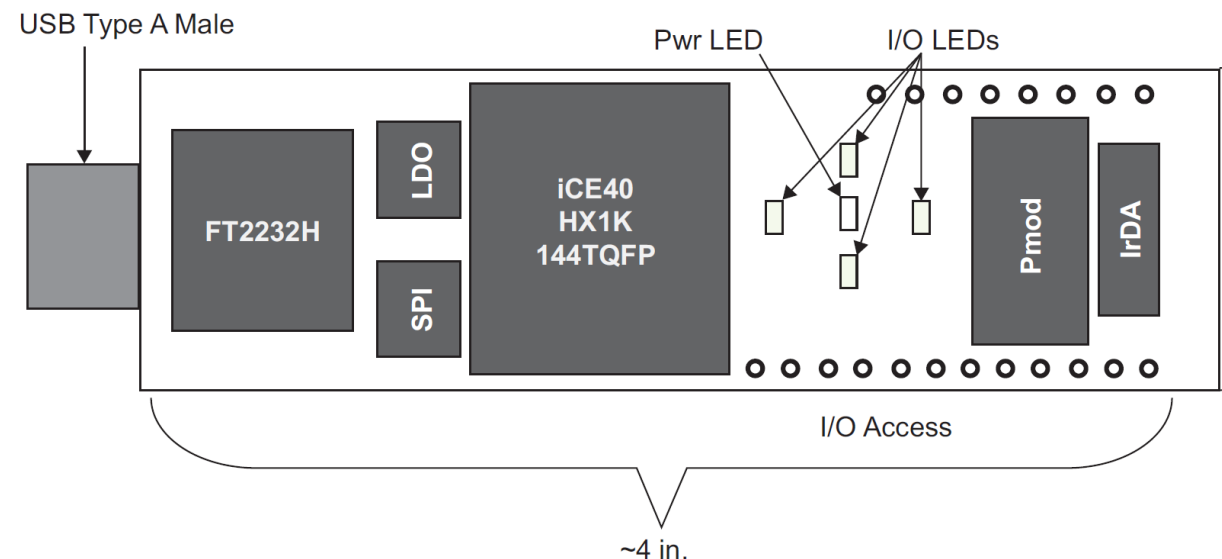
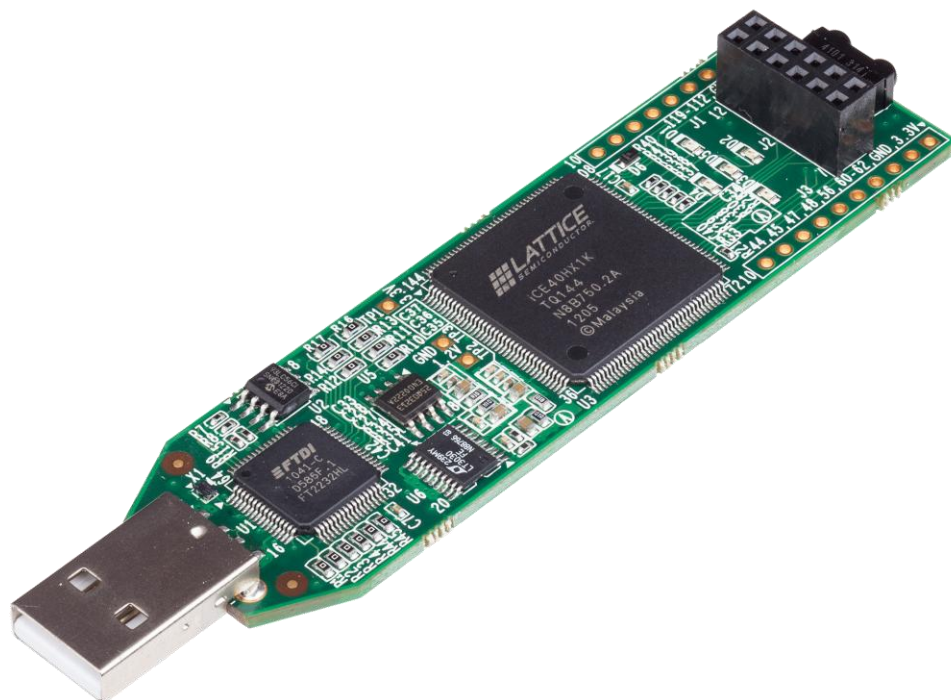
# Yosys

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

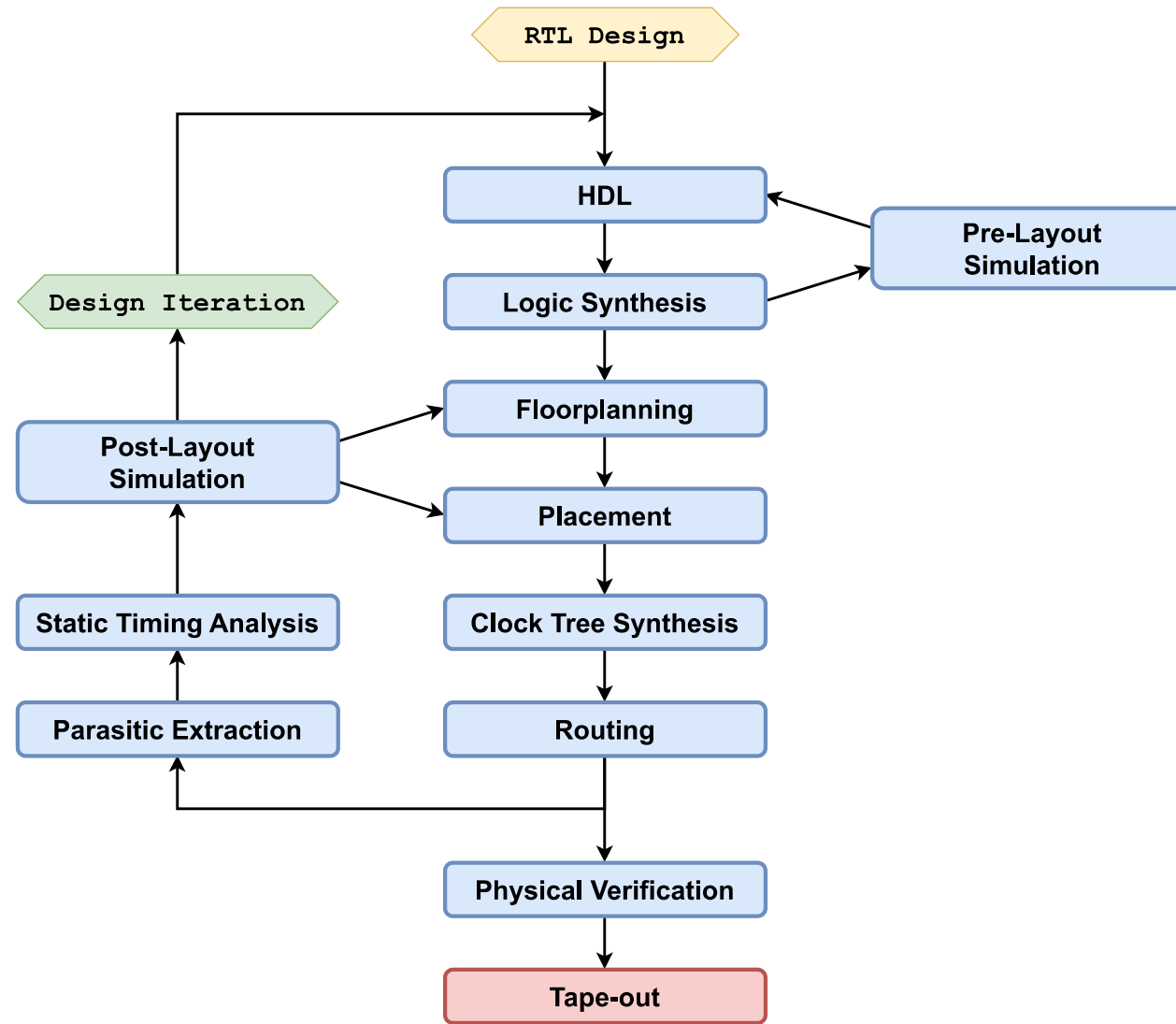


# NextPNR & Lattice iCE40 FPGA Board

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

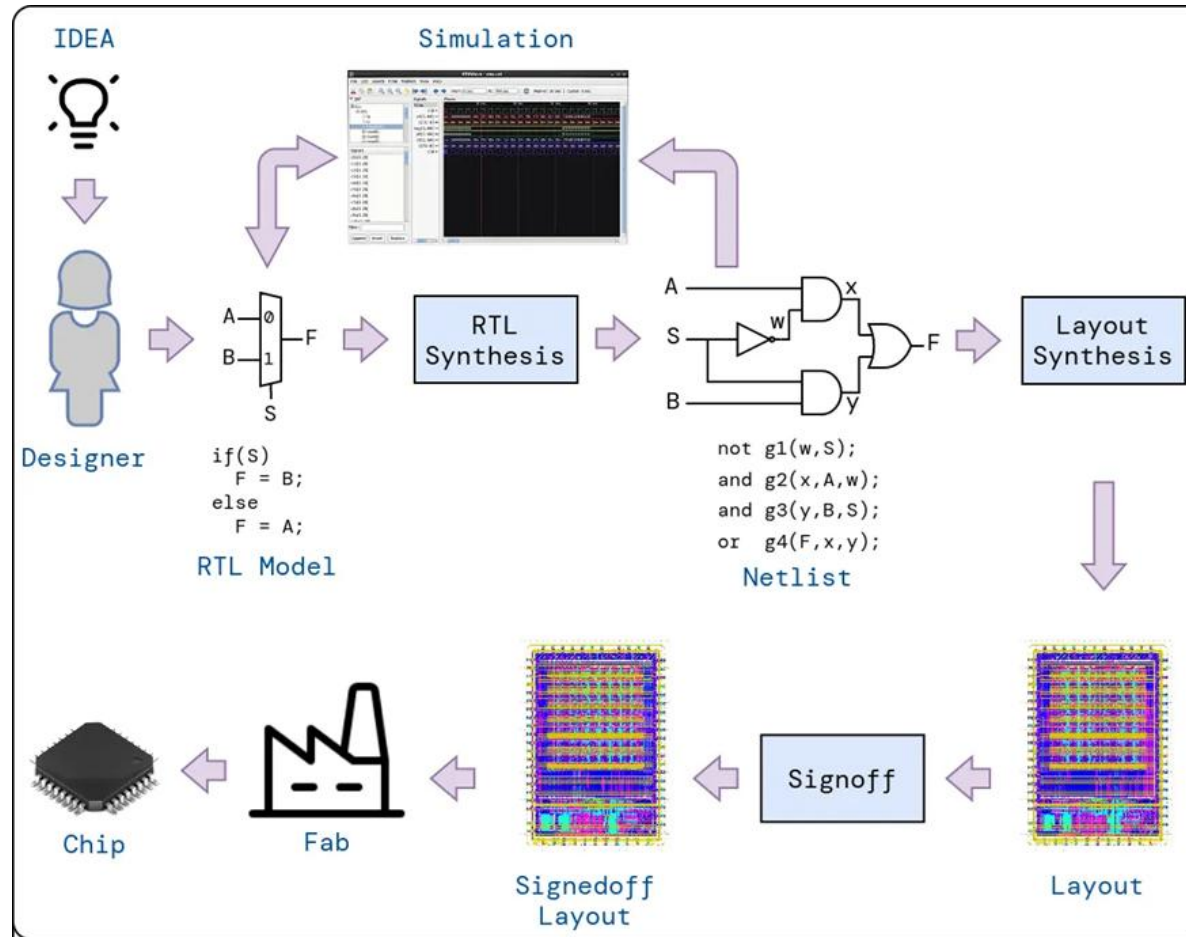


# RTL to GDS Flow



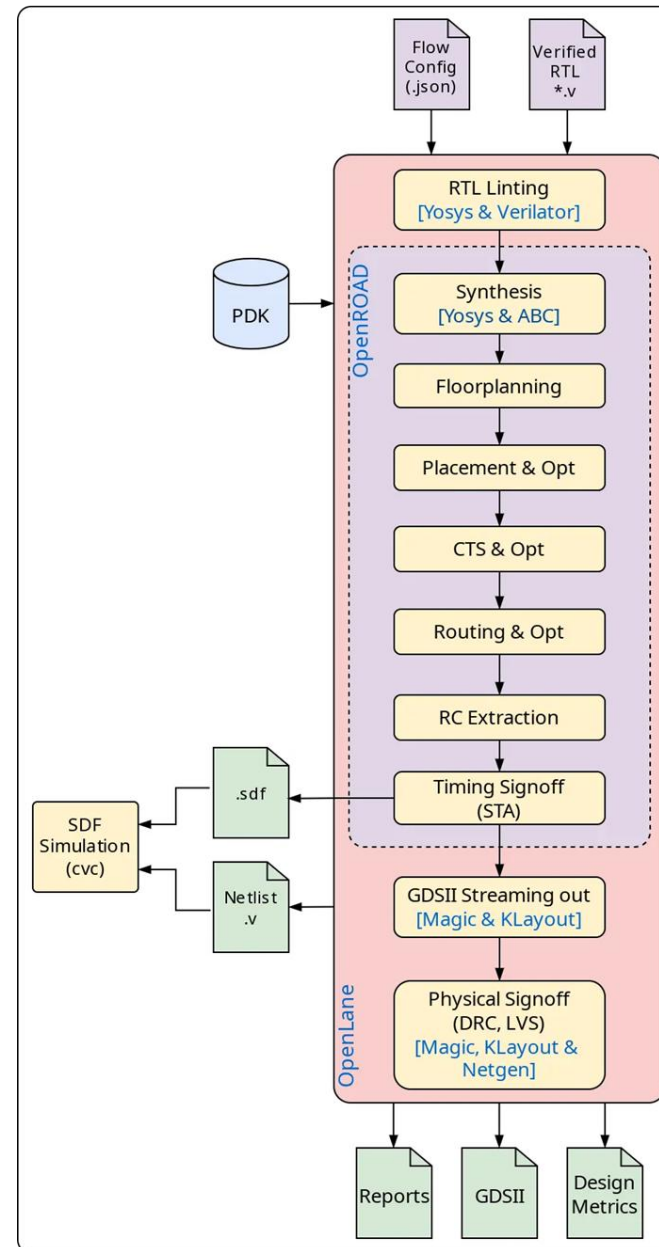
# LibreLane – Intro

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





# LibreLane – Flow Diagram



# 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. `pdnngen` - 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](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/>

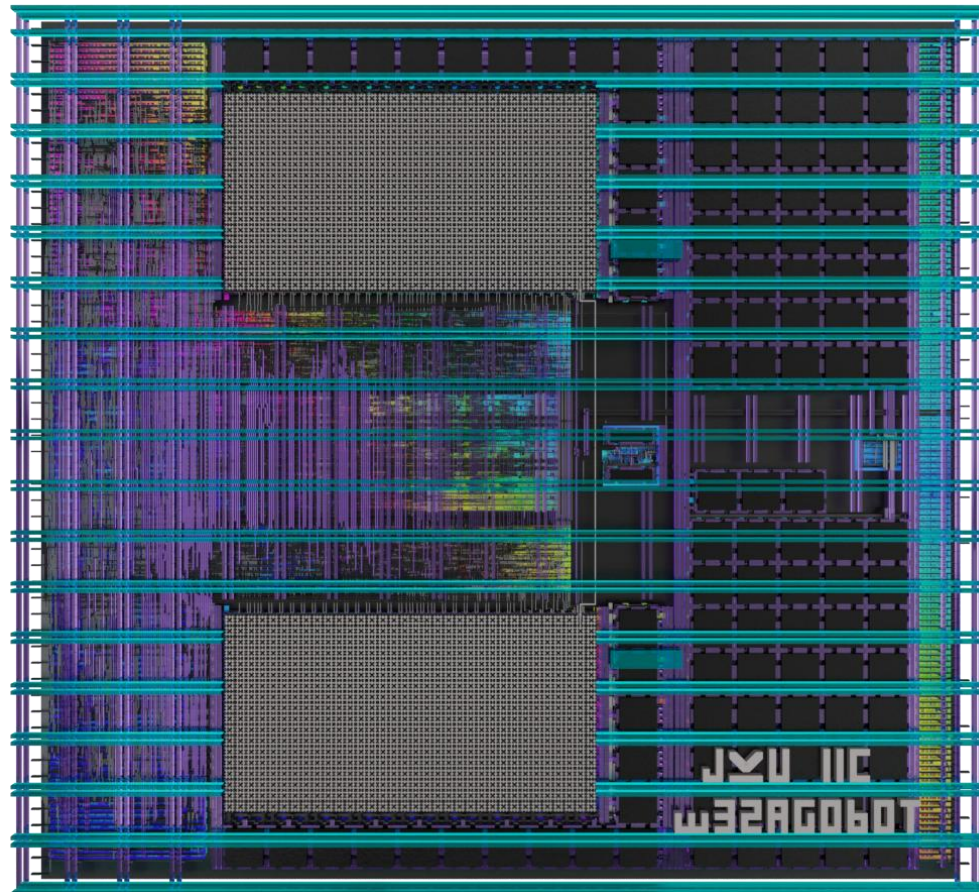
- **Europractice:**

- <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](https://github.com/iic-jku/SKY130_SAR-ADC1)
- Synthesizable Digital Temperature Sensor: <https://github.com/iic-jku/tt03-tempsensor>





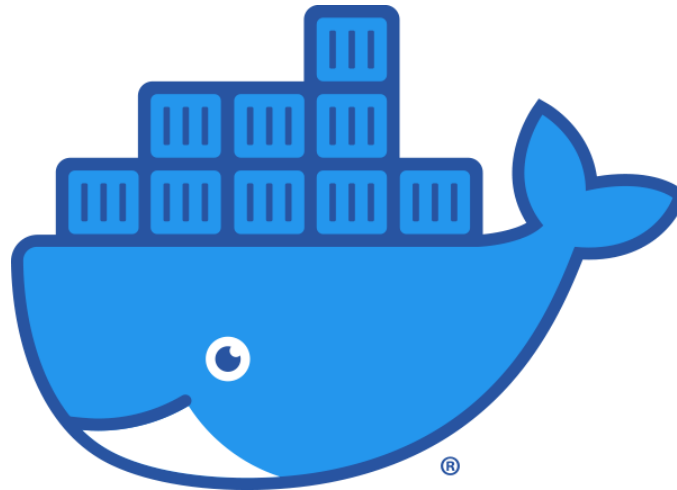
# Examples – Tiny Tapeout

- 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
- Alternatively, access to a virtual machine (VM) can be requested by email.



# 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**.

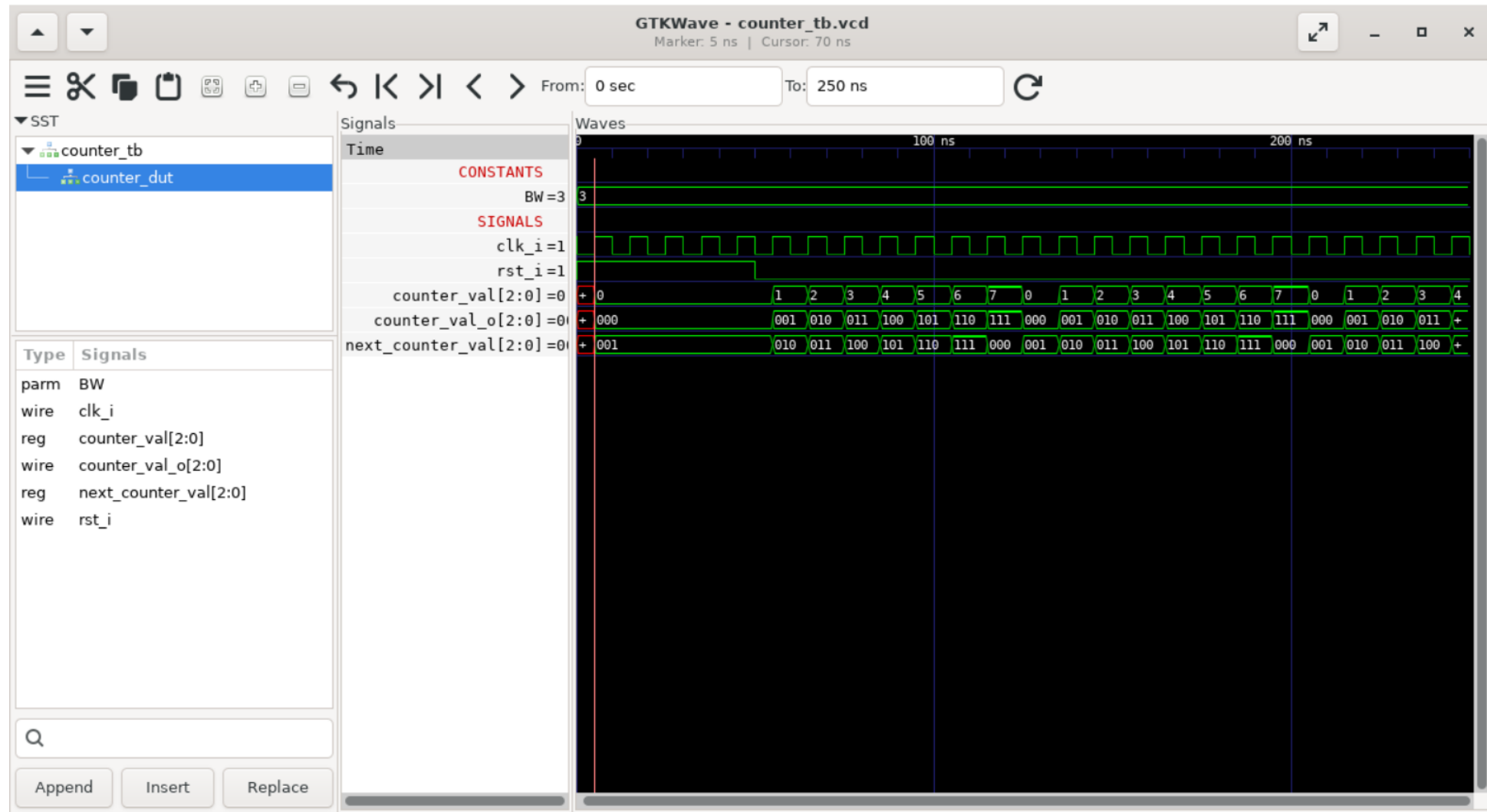
```
/foss/designs/kvic_336007_ws25/verilog/tb > iverilog -g2005 ../src/counter.v counter_tb.v  
/foss/designs/kvic_336007_ws25/verilog/tb > ./a.out  
VCD info: dumpfile counter_tb.vcd opened for output.  
counter_tb.v:38: $finish called at 250 (1ns)  
/foss/designs/kvic_336007_ws25/verilog/tb > gtkwave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

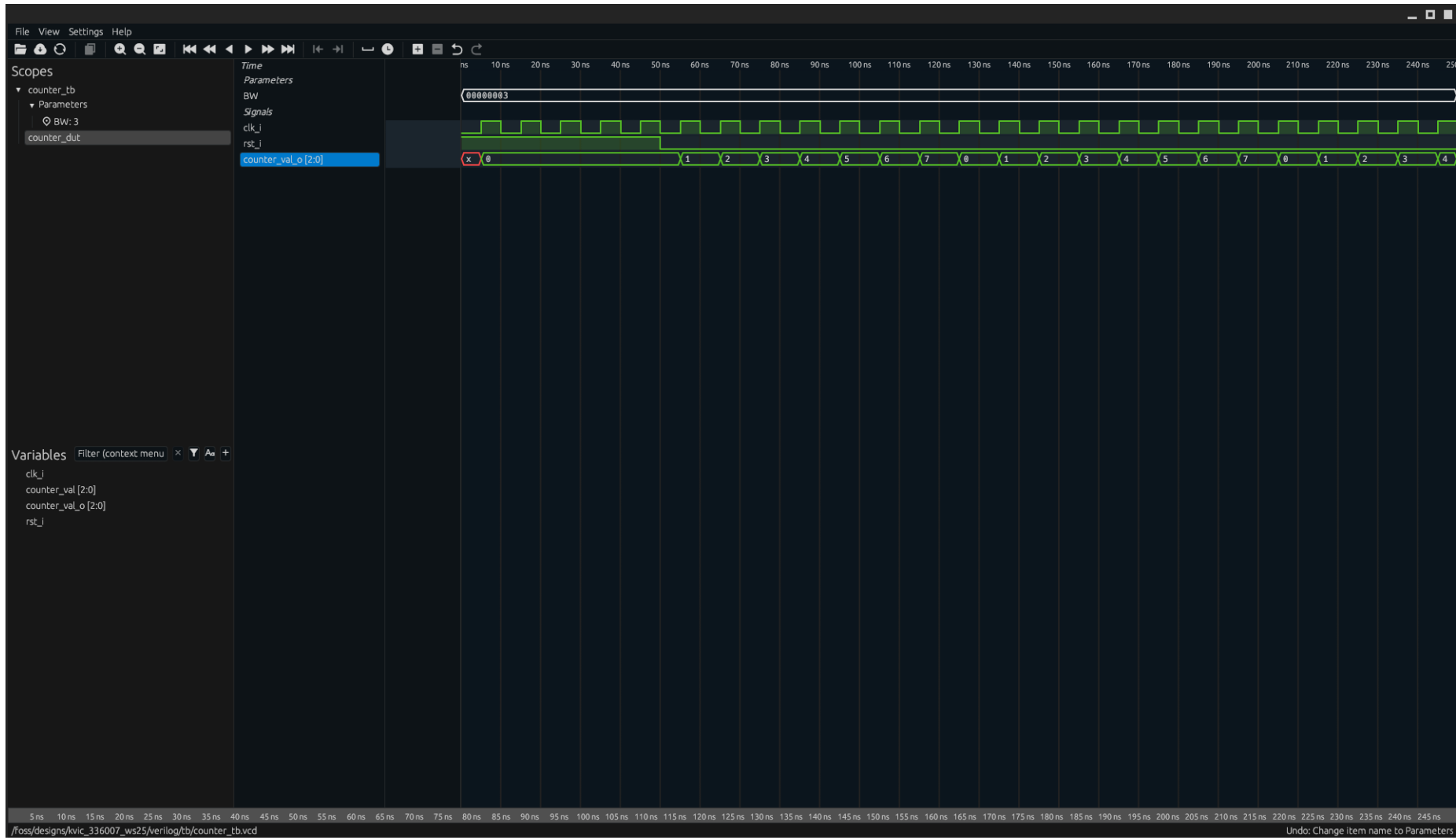
# Tutorial – Compilation / Linting / Simulation

```
1  #!/bin/bash
2
3  # =====
4  # Author: Simon Dorrer
5  # Last Modified: 02.10.2025
6  # Description: This .sh file verifies and simulates a verilog testbench with Verilator, IVerilog and GTKWave.
7  # =====
8
9  set -e -x
10
11  cd "$(dirname "$0")"
12
13  GREEN='\033[1;32m'
14  NC='\033[0m'
15
16  name=$1
17
18  RTL=${RTL:-../src}
19  SRC_FOLDER=${SRC_FOLDER:-.}
20
21  echo -e "${GREEN}Verilator:----- ${NC}"
22  verilator --lint-only "$RTL"/"$name".v
23  echo -e "${GREEN}IVerilog:----- ${NC}"
24  iverilog -g2005 "$RTL"/"$name".v "$SRC_FOLDER"/"$name"_tb.v
25  echo -e "${GREEN}a:----- ${NC}"
26  ./a.out
27  echo -e "${GREEN}GTKWave:----- ${NC}"
28  if [ -e "$SRC_FOLDER"/"$name"_tb.gtkw ]
29  then
30      gtkwave "$SRC_FOLDER"/"$name"_tb.gtkw
31  else
32      gtkwave "$SRC_FOLDER"/"$name"_tb.vcd
33  fi
34
35  # Clean
36  rm -f a.out
37  rm -f *.vcd
38
39  echo -e "${GREEN}Generated files were removed----- ${NC}"
```

# Tutorial – GKTWave



# Tutorial – Surfer



# Tutorial – Showing Stats

```
/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
```

```
1  #!/usr/bin/env bash
2
3  # =====
4  # Author: Simon Dorrer
5  # Last Modified: 02.10.2025
6  # Description: This .sh file loads the Verilog file and outputs stats, such as cells, AND, OR, NOT, XOR, MUX, and registers.
7  # =====
8
9  set -e -x
10
11  cd $(dirname "$0")
12
13  # Name as input parameter (counter)
14  name=$1
15
16  yosys -p "read verilog \"$name\".v; proc; opt; flatten; techmap; stat"
```

# 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...  $Y = A \leq B$  (lower equal)
  - shl...  $Y = A \ll B$  (logical left shift)
  - sshl...  $Y = A \lll B$  (arithmetic left shift)
  - gt...  $Y = A > B$  (greater than)
  - ge...  $Y = A \geq B$  (greater equal)
  - shr...  $Y = A \gg B$  (logical right shift)
  - sshr...  $Y = A \ggg B$  (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 – Wokwi to Verilog Template

- Rename the existing repo
- Create a new repo from ttsky-Verilog-template with the same name as the original repo
  - <https://github.com/TinyTapeout/ttsky-verilog-template>
- Get docs and GDS green
- Go to <https://app.tinytapeout.com/> and click on your project name, then click on the big blue **“SUBMIT A NEW REVISION”** button



# Tutorial – Wokwi to Verilog Template

- Rename the existing repo
- Create a new repo from ttsky-Verilog-template with the same name as the original repo
  - <https://github.com/TinyTapeout/ttsky-verilog-template>
- Get docs and GDS green
- Go to <https://app.tinytapeout.com/> and click on your project name, then click on the big blue “**SUBMIT A NEW REVISION**” button
- Detailed description: <https://moodle.jku.at/mod/forum/discuss.php?d=151162>

# Tutorial – LibreLane

- In the „config.json“ file the parameters for the digital layout is set.
  - CLOCK\_PERIOD (to achieve a fast enough design)
  - FP\_CORE\_UTIL, FP\_IO\_MIN\_DISTANCE, PLACE\_DENSITY, ...
  - The manual and ChatGPT can help out here 😊
- In order to create the IC design, one must navigate to the folder with the config files and execute the command „flow.tcl -designs .“.
- After some minutes / hours, the design should be created and it can be viewed with the tool „magic“ or „klayout“.
- In „magic“, one can zoom after creating a box by a left click and a right click and pressing STRG+Z. The full view can be established again by pressing the key „f“.
- Please see the “magic\_cheatsheet.pdf” for further commands!

# LibreLane – Output – ToDo

- All output run data is placed by default under “./designs/design\_name/runs”
- Each flow cycle will output a timestamp-marked folder containing the following file structure:

```
<design_name>
├─ config.json/config.tcl
├─ runs
│   └─ <tag>
│       ├── config.tcl
│       ├── {logs, reports, tmp}
│       ├── cts
│       ├── signoff
│       ├── floorplan
│       ├── placement
│       ├── routing
│       └─ synthesis
│   └─ results
│       ├── final
│       ├── cts
│       ├── signoff
│       ├── floorplan
│       ├── placement
│       ├── routing
│       └─ synthesis
```

# Tutorial – LibreLane

```
/foss/designs/kvic_336007_ws22-main/dig/audiodac > flow.tcl -designs .  
/foss/designs/kvic_336007_ws22-main/dig/audiodac > cd runs  
/foss/designs/kvic_336007_ws22-main/dig/audiodac/runs > ll  
total 0  
drwxr-xr-x 1 designer designers 4096 Dec 22 13:47 RUN_2022.12.22_13.47.02  
drwxr-xr-x 1 designer designers 4096 Dec 22 13:48 RUN_2022.12.22_13.47.50  
drwxr-xr-x 1 designer designers 4096 Dec 22 13:54 RUN_2022.12.22_13.50.48  
drwxr-xr-x 1 designer designers 4096 Dec 22 14:08 RUN_2022.12.22_14.04.30  
drwxr-xr-x 1 designer designers 4096 Dec 22 14:27 RUN_2022.12.22_14.22.53  
drwxr-xr-x 1 designer designers 4096 Jan  3 23:40 RUN_2023.01.03_23.36.27  
/foss/designs/kvic_336007_ws22-main/dig/audiodac/runs > cd RUN_2023.01.03_23.36.27/results/final/gds  
/foss/designs/kvic_336007_ws22-main/dig/audiodac/runs/RUN_2023.01.03_23.36.27/results/final/gds > magic audiodac.gds  
/foss/designs/kvic_336007_ws22-main/dig/audiodac/runs/RUN_2023.01.03_23.36.27/results/final/gds >  
/foss/designs/kvic_336007_ws22-main/dig/audiodac/runs/RUN_2023.01.03_23.36.27/results/final/gds > klayout audiodac.gds
```

- Verilog: “./designs/design\_name/runs/RUN\_XYZ/results/final/verilog”
- GDS: “./designs/design\_name/runs/RUN\_XYZ/results/final/gds”
- Reports: “./designs/design\_name/runs/RUN\_XYZ/reports”

# Tutorial – LibreLane

- Final digital layout of the counter