FOSS ASIC DESIGN FLOW - TUTORIAL

Simon Dorrer, JKU ISP

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

1.) Free and open-source software (FOSS)

- **GHDL:** VHDL analyzer, compiler, simulator and synthesizer
- **YOSYS:** Verilog synthesis tool (with GHDL plugin for VHDL synthesis)
- **Magic:** layout editor with DRC and PEX
- **Klayout:** layout viewer and editor for GDS and OASIS
- **OpenRAM:** Python library for RAM block generation
- **Xschem:** Schematic editor for analog circuit design
- **Openlane:** Digital RTL2GDS 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

2.) Installed open-source PDKs

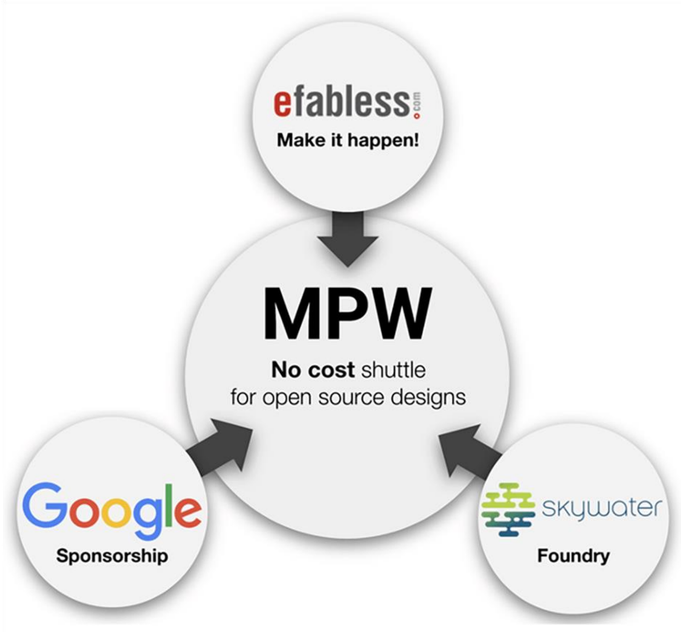
INSTITUTE OF SIGNAL PROCESSING

- PDK: process-development kits
  - https://en.wikipedia.org/wiki/Process_design_kit
  - a set of files used within the semiconductor industry to model a fabrication process for the design tools used to design an integrated circuit
  - Symbols, Device Parameters
  - Design Rule Check (DRC), Layout vs. Schematic (LVS), Electrical Rule Check (ERC)
  - Simulation models for components (e.g. spice)
- Skywater 130nm Technology: https://www.skywatertechnology.com/
- Global Foundries 180nm Technology: https://gf.com/de/
- IHP Microelectronics 130nm Technology: https://www.ihp-microelectronics.com/de

3.) Where to tape out?

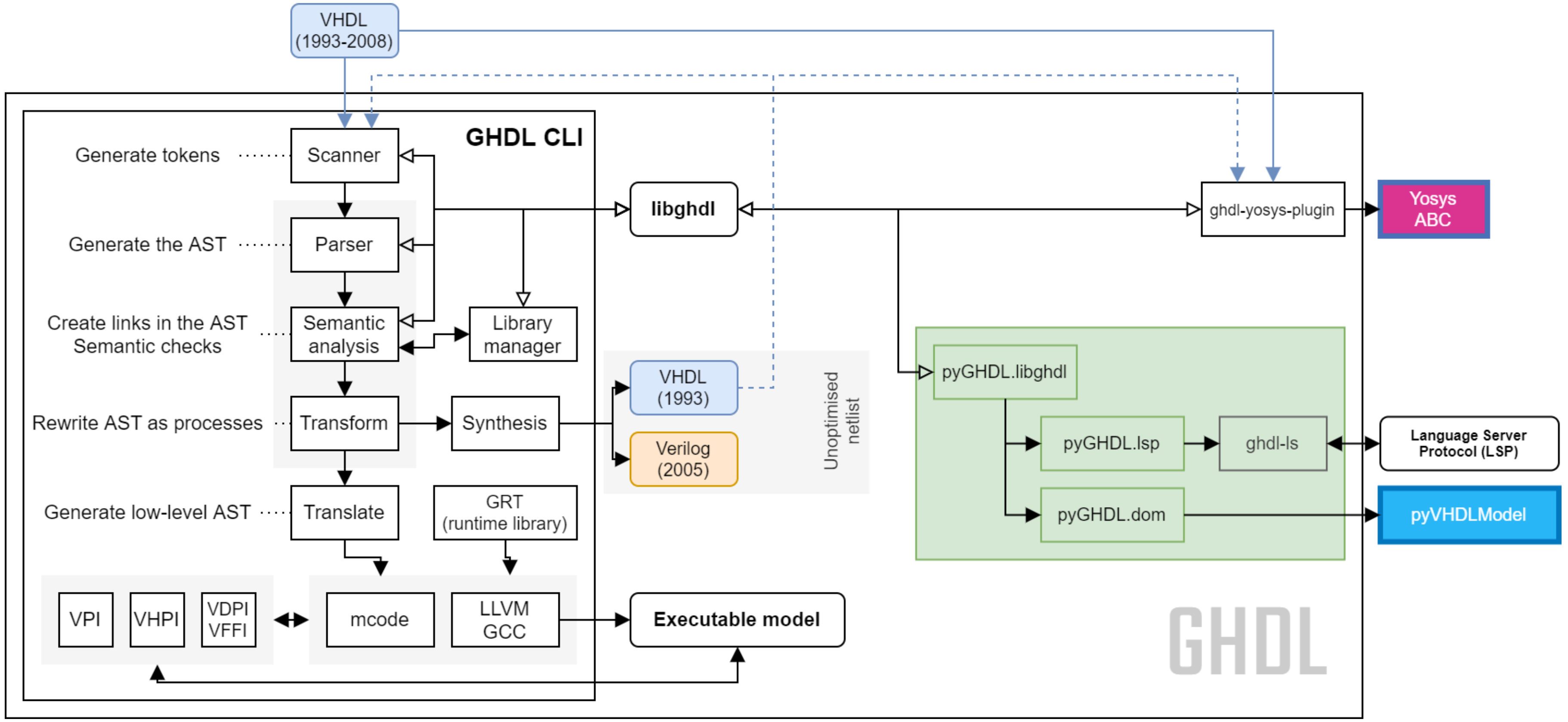
- Efabless Open MPW Program:
  - https://efabless.com/open_shuttle_program
  - PDK from skywater
  - Chip manufactured by Efabless
  - Google coveres costs for fabrication, packaging, evaluation boards and shipping

- Tiny Tapeout 3:
  - https://tinytapeout.com/
  - PDK from skywater
  - Chip manufactured by Efabless
  - Extremely cheap, since more projects are put on one chip



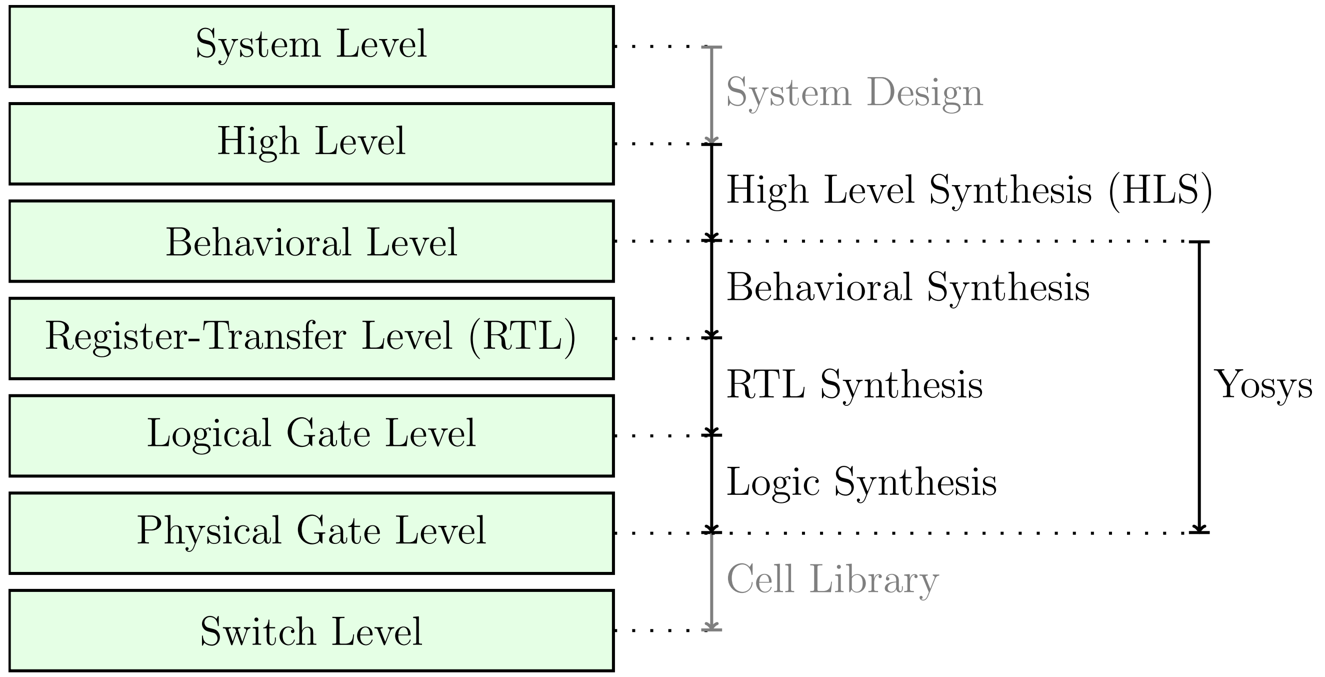
4.) GHDL

- Github: https://github.com/ghdl/ghdl
- Documentation: https://ghdl.github.io/ghdl/about.html



5.) Yosys

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



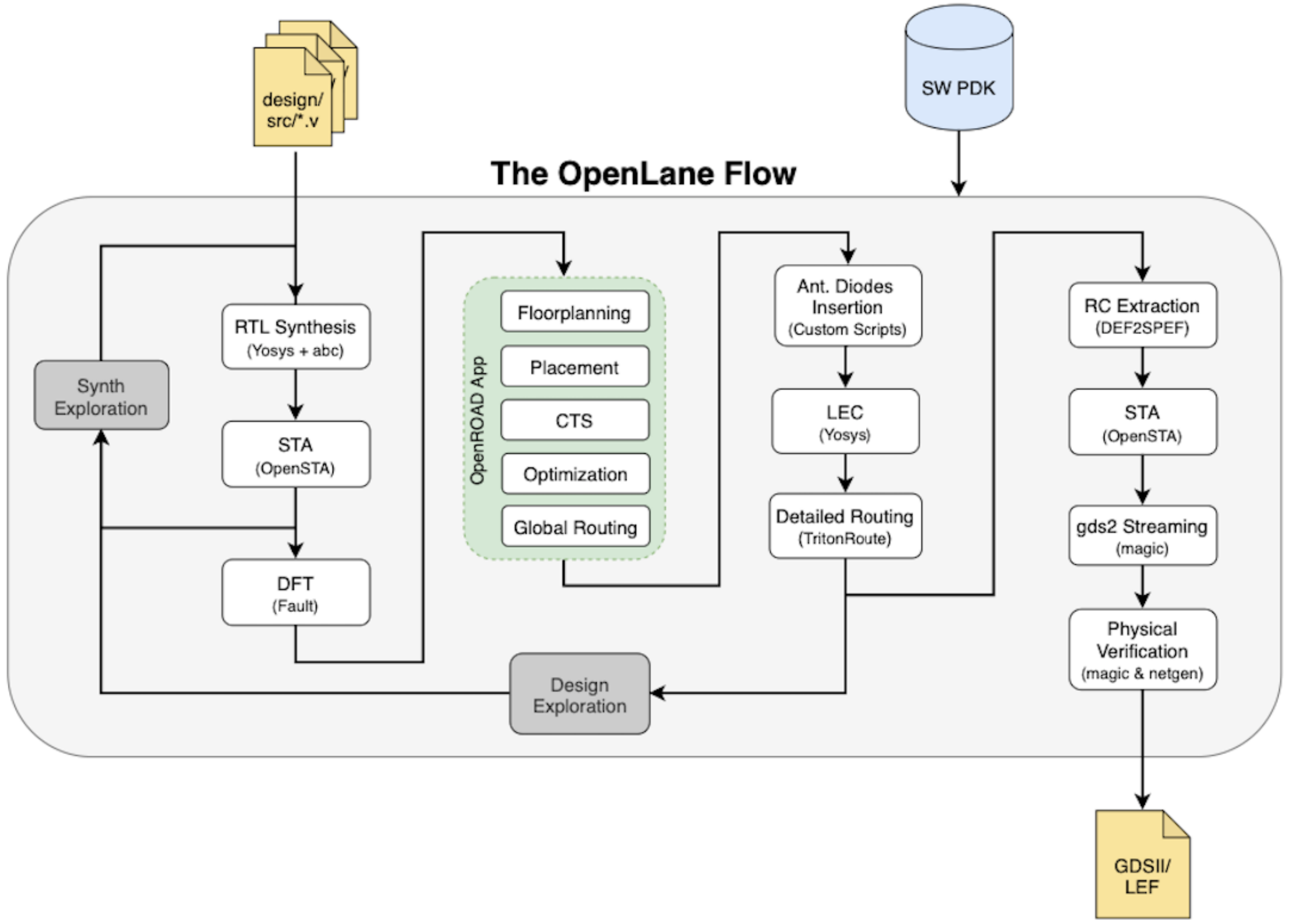
6.) OpenLane – Intro

INSTITUTE OF SIGNAL PROCESSING

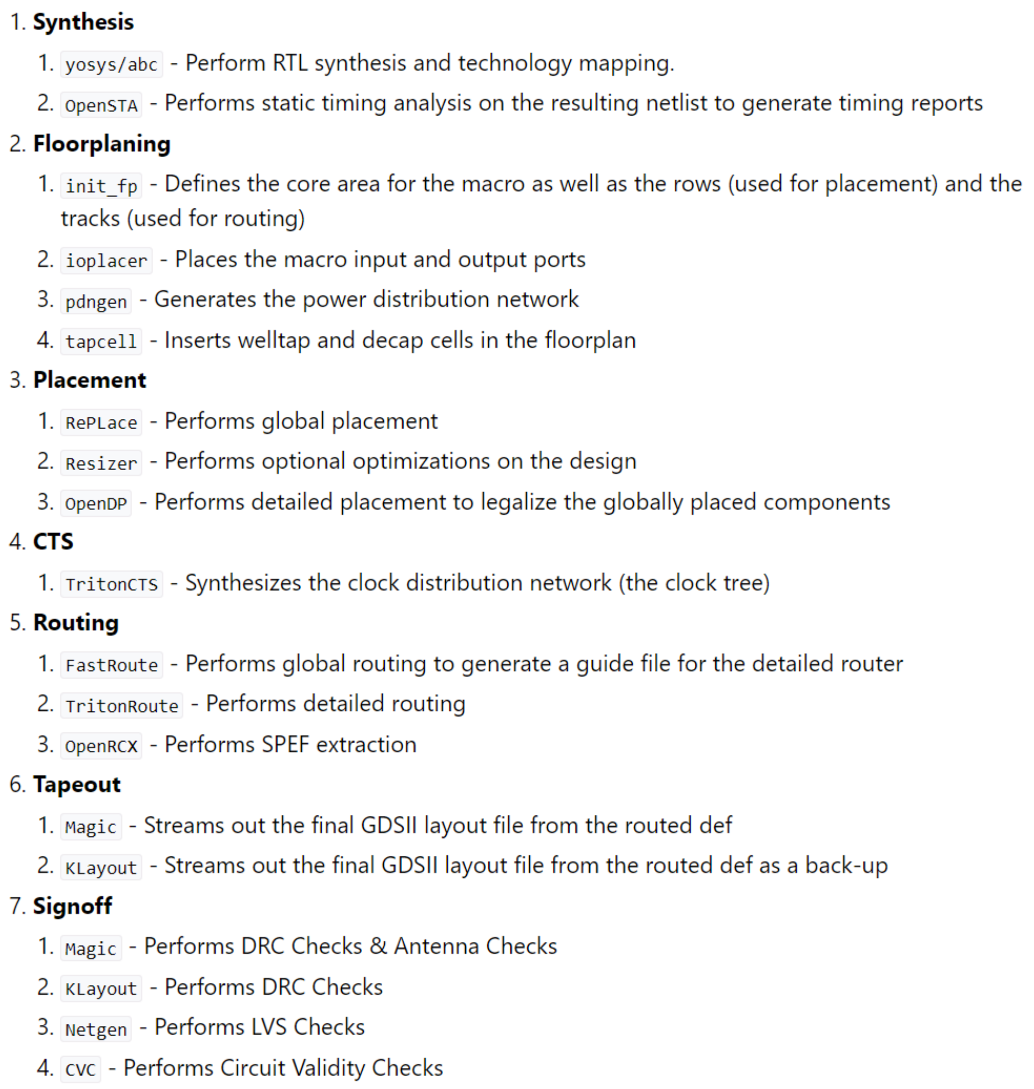
- Github: https://github.com/The-OpenROAD-Project/OpenLane
- Documentation: https://ghdl.github.io/ghdl/about.html
- Yosys, magic, and so on are used by OpenLane
- OpenROAD is one of the tools used by the OpenLane flow
- The OpenLane flow needs a Verilog code as input → convert VHDL to Verilog with GHDL

OPENLANE

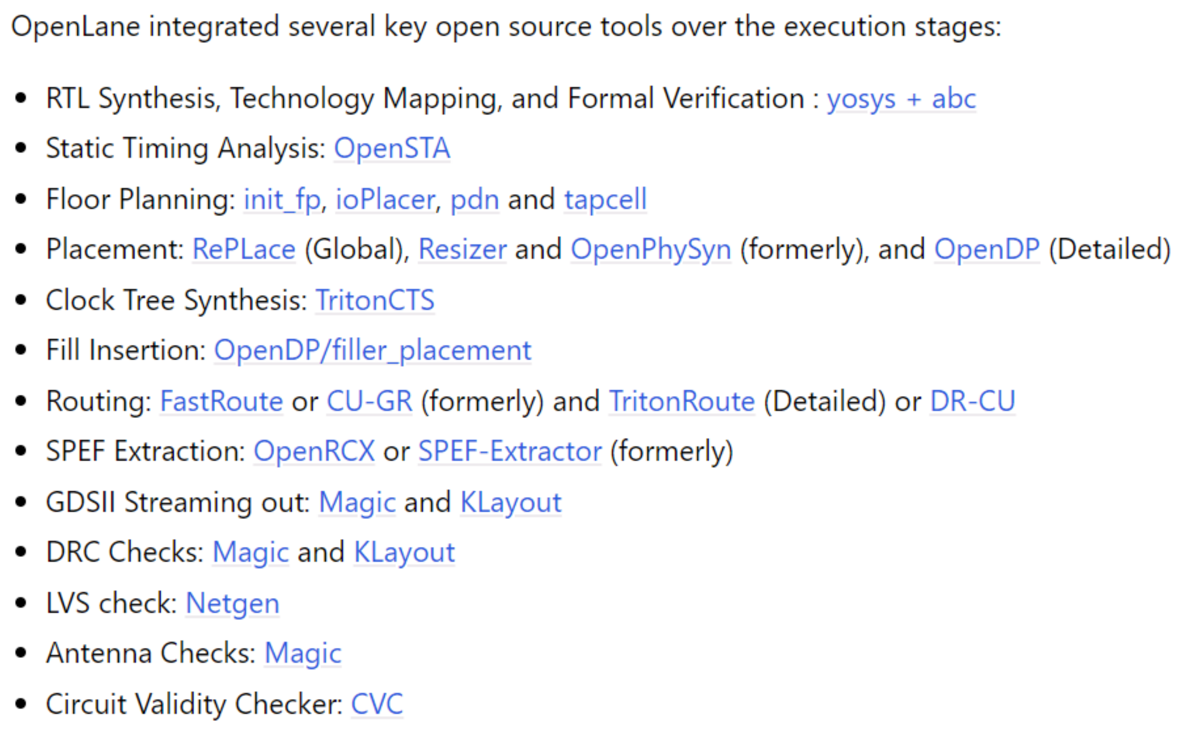
6.) OpenLane – Flow Diagram



6.) OpenLane – 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

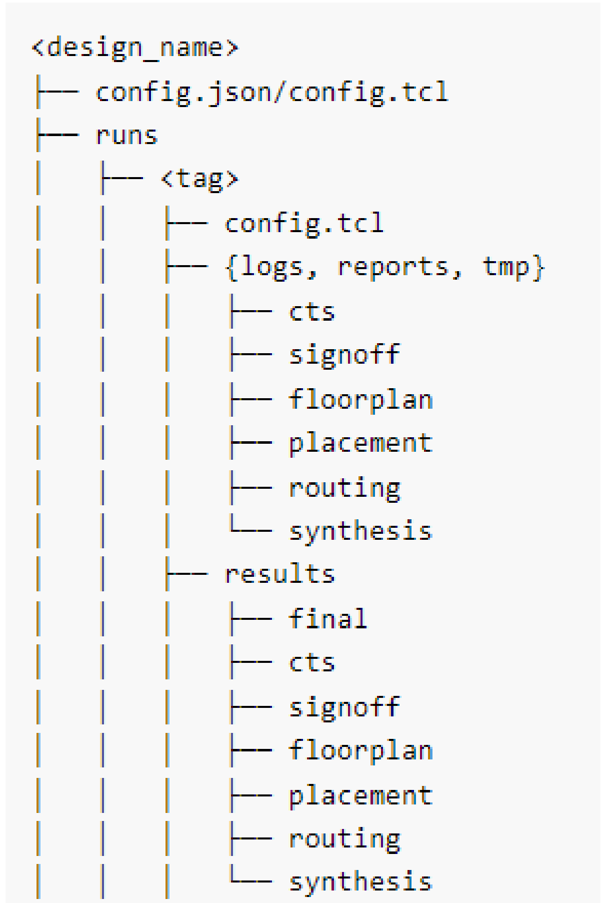


OpenLane integrated several key open source tools over the execution stages:

- 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

6.) OpenLane – Output

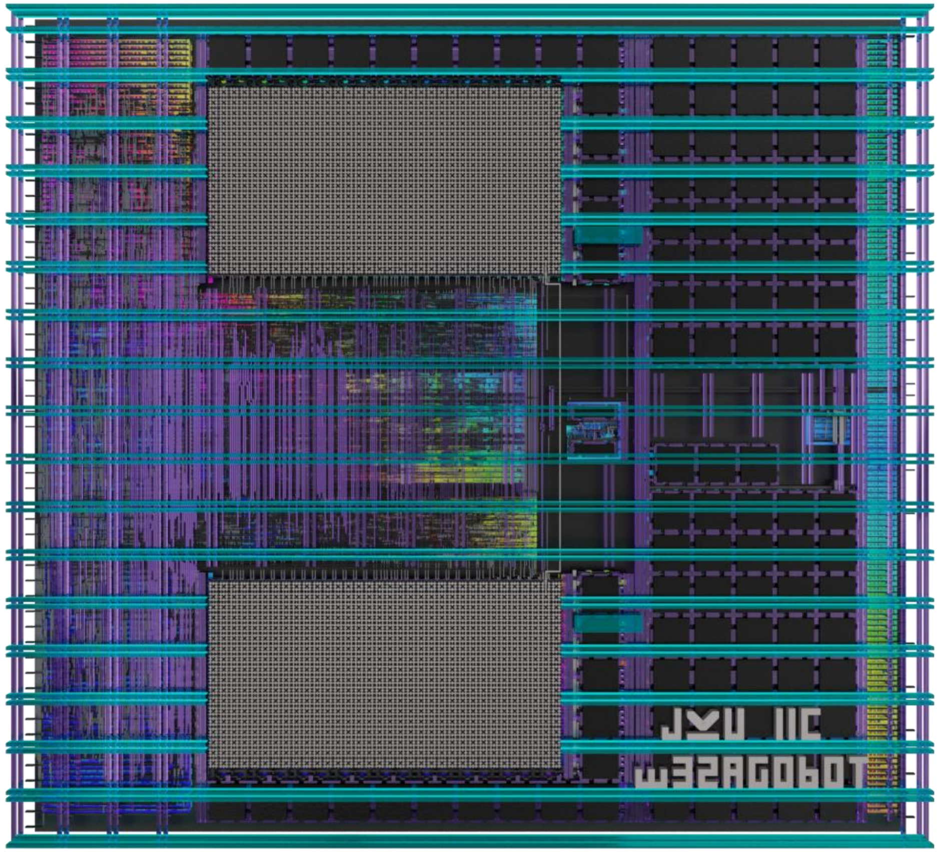
- All output run data is placed by default under *“./designs/design_name/runs”*
- Each flow cycle will output a timestamp-marked folder containing the following file structure:



```
<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
│   │   │
```

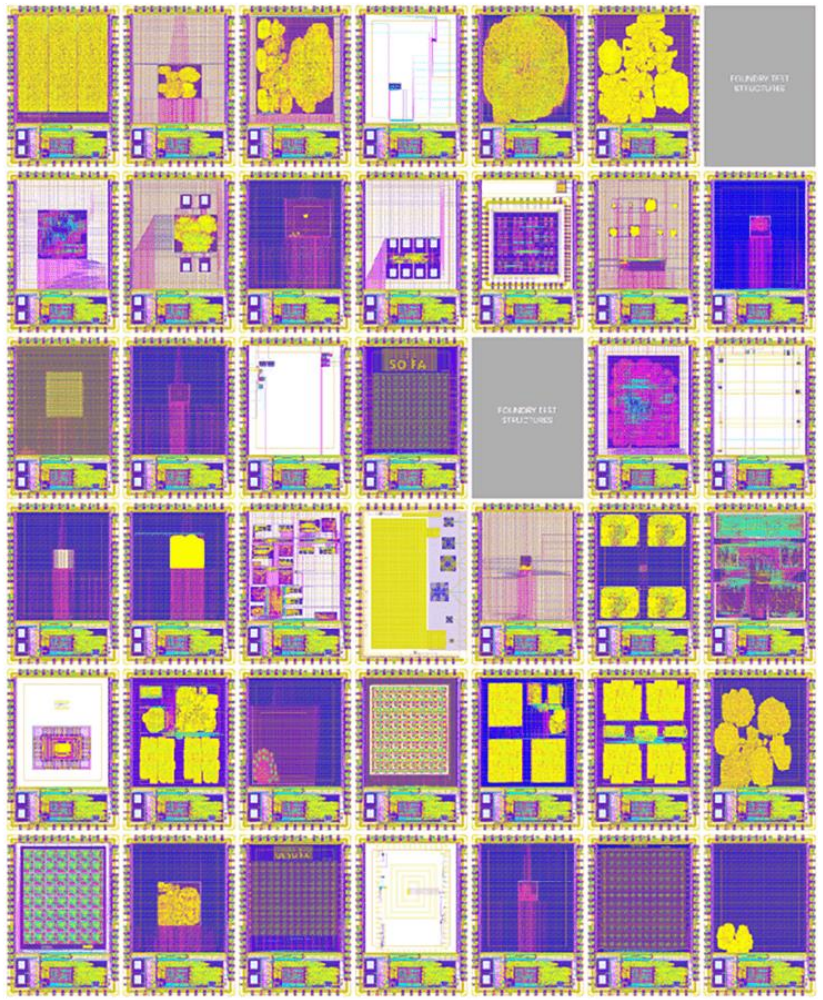
7.) Examples – IIC

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



7.) Examples - Efabless Open MPW Program

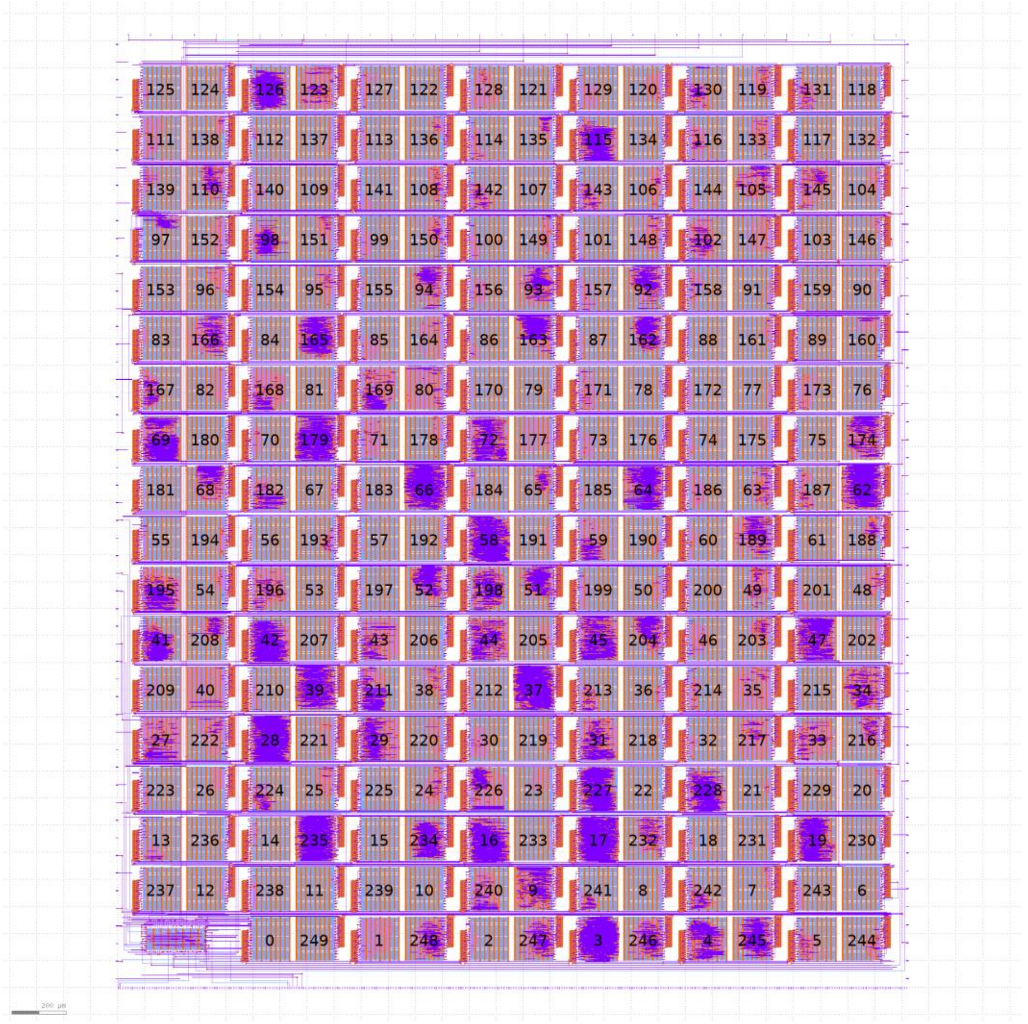
- MPW-8 projects: https://platform.efabless.com/projects/shuttle/14



JOHANNES KEPLER UNIVERSITÄT LINZ

7.) Examples – Tiny Tapeout

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



8.) Tutorial – Setting up Docker container

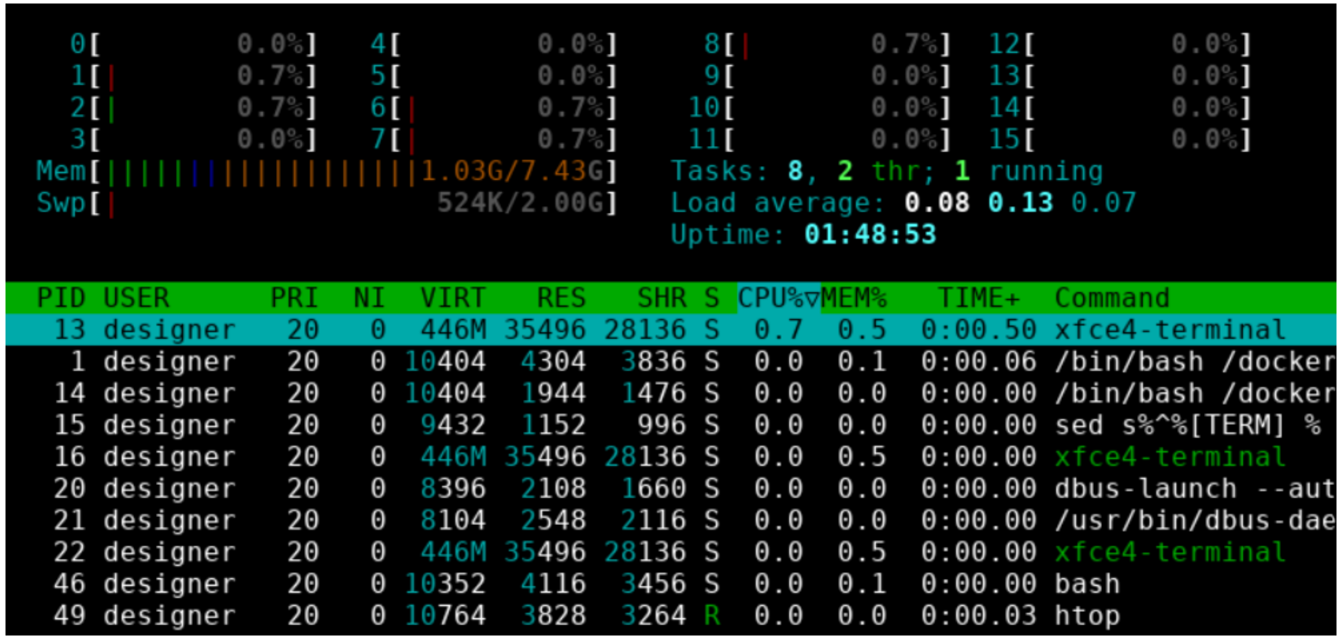
- Install Docker: https://docs.docker.com/desktop/install/windows-install/
- Install Xming X-server: https://sourceforge.net/projects/xming/
- Download IIC-OSIC-TOOLS: https://github.com/iic-jku/iic-osic-tools
- Set up the environment
  - Please, do the step-by-step instruction shown in “EIS-KV_Docker-Tutorial.pdf”
  - X-server or XVNC (WebApp, PW: abc123)

8.) Tutorial – Recommended Workflow

- 1.) Write VHDL Design
- 2.) Simulate VHDL Design in ModelSim
- 3.) Synthesize VHDL Design in Quartus
- 4.) Test VHDL Design on FPGA Board
- 5.) Convert VHDL Design to Verilog Design with GHDL
- 6.) Simulate Verilog Design in ModelSim
- 7.) Synthesize Verilog Design in Quartus
- 8.) Compare hardware resources of VHDL and Verilog Quartus Projects
- 9.) Test Verilog Design on FPGA Board
- 10.) Use Yosys to read the Verilog File and output stats and .svg files
- 11.) Use OpenLane for open-source ASIC Design
- 12.) Take Skywater 130nm Verilog netlist and simulate the design again (Post-Layout Simulation)

8.) Tutorial – Basic Commands

- Enter the next folder with *"cd folder_name"* and go back with *"cd .."*.
- After some minutes of inactivity, the X-server closes itself. Just open the task manager with *"htop"* in a new tab to prevent this.
- If an application (e.g. simulation, yosys) should be closed, just enter STRG+C.

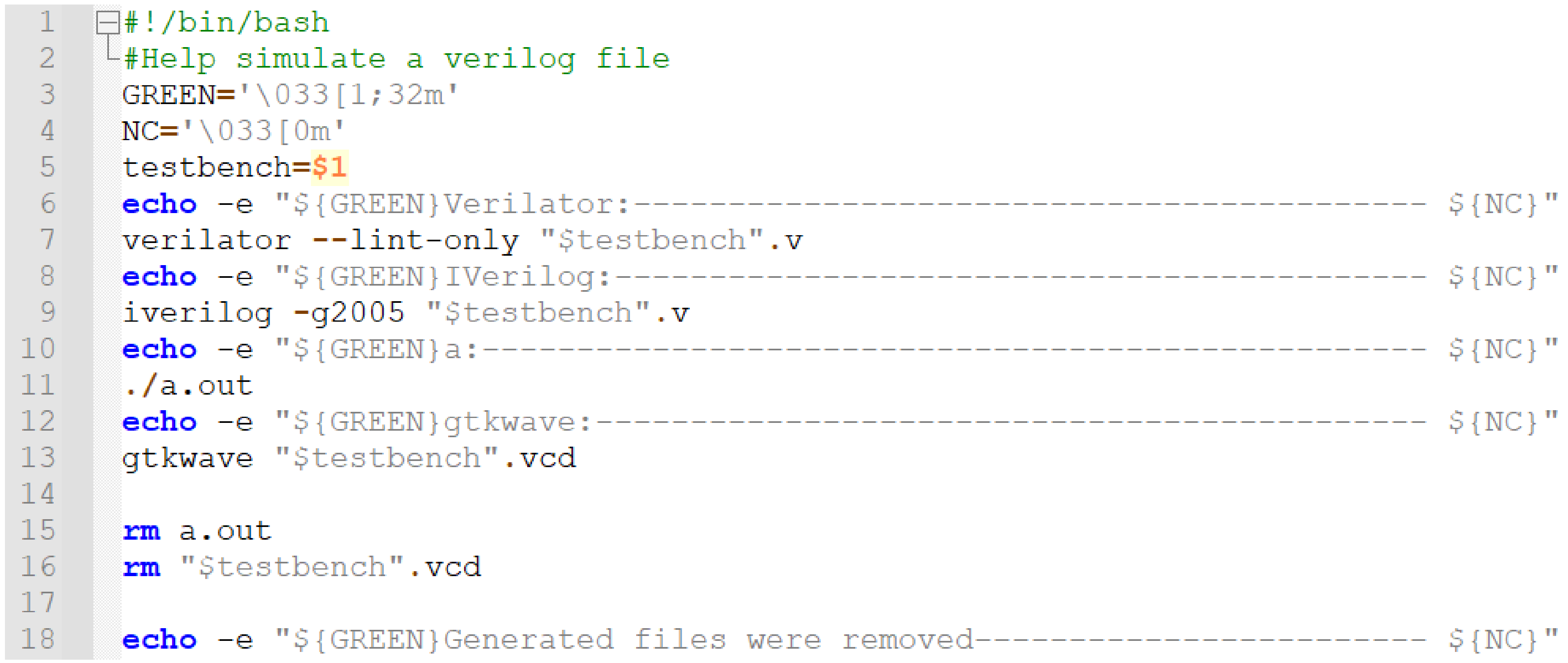


```
    0[         0.0%]   4[           0.0%]     8[|         0.7%]  12[          0.0%]
    1[|        0.7%]   5[           0.0%]     9[          0.0%]  13[          0.0%]
    2[|        0.7%]   6[|          0.7%]    10[          0.0%]  14[          0.0%]
    3[         0.0%]   7[|          0.7%]    11[          0.0%]  15[          0.0%]
  Mem[||||||||||||||||||1.03G/7.43G]   Tasks: 8, 2 thr; 1 running
  Swp[|                 524K/2.00G]   Load average: 0.08 0.13 0.07
                                      Uptime: 01:48:53

  PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
   13 designer   20   0  446M 35496 28136 S  0.7  0.5  0:00.50 xfce4-terminal
    1 designer   20   0 10404  4304  3836 S  0.0  0.1  0:00.06 /bin/bash /docker
   14 designer   20   0 10404  1944  1476 S  0.0  0.0  0:00.00 /bin/bash /docker
   15 designer   20   0  9432  1152   996 S  0.0  0.0  0:00.00 sed s%^%[TERM] %
   16 designer   20   0  446M 35496 28136 S  0.0  0.5  0:00.00 xfce4-terminal
   20 designer   20   0  8396  2108  1660 S  0.0  0.0  0:00.00 dbus-launch --aut
   21 designer   20   0  8104  2548  2116 S  0.0  0.0  0:00.00 /usr/bin/dbus-dae
   22 designer   20   0  446M 35496 28136 S  0.0  0.5  0:00.00 xfce4-terminal
   46 designer   20   0 10352  4116  3456 S  0.0  0.1  0:00.00 bash
   49 designer   20   0 10764  3828  3264 R  0.0  0.0  0:00.03 htop
```

8.) Tutorial – Compilation / 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.
- With the self-written shell, this can be done very easily, just call *“./simulate.sh <file>“*.
- After a simulation is set up it can be saved with STRG+S and re-loaded with STRG+O again.
- A simulation file is saved as „wave.gtkw“ by default

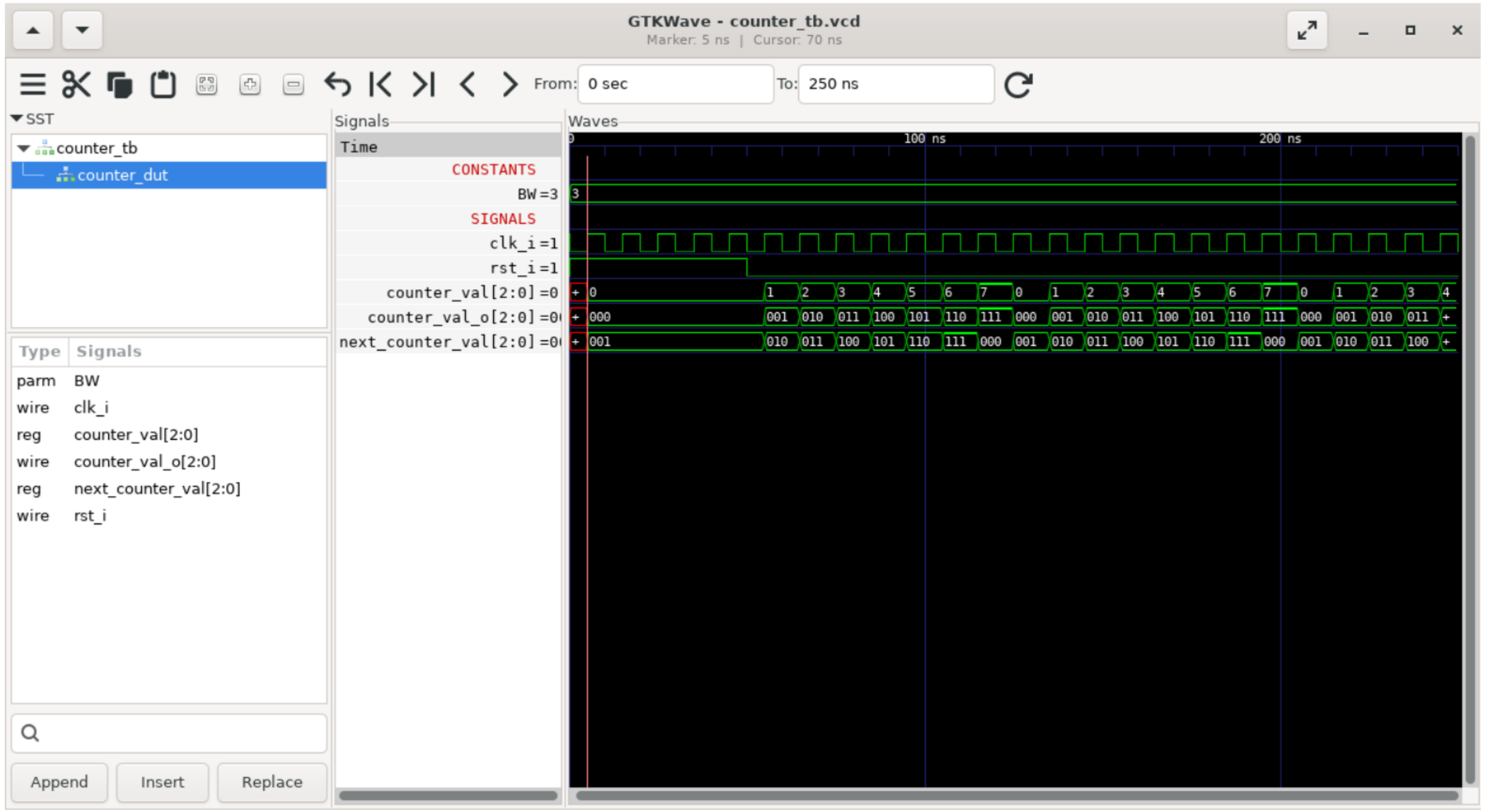
8.) Tutorial – Compilation / Simulation



```bash
#!/bin/bash
#Help simulate a verilog file
GREEN='\033[1;32m'
NC='\033[0m'
testbench=$1
echo -e "${GREEN}Verilator:------------------------------------------ ${NC}"
verilator --lint-only "$testbench".v
echo -e "${GREEN}IVerilog:------------------------------------------- ${NC}"
iverilog -g2005 "$testbench".v
echo -e "${GREEN}a:-------------------------------------------------- ${NC}"
./a.out
echo -e "${GREEN}gtkwave:-------------------------------------------- ${NC}"
gtkwave "$testbench".vcd

rm a.out
rm "$testbench".vcd

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

8.) Tutorial – Compilation / Simulation

- Simulation of counter example



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

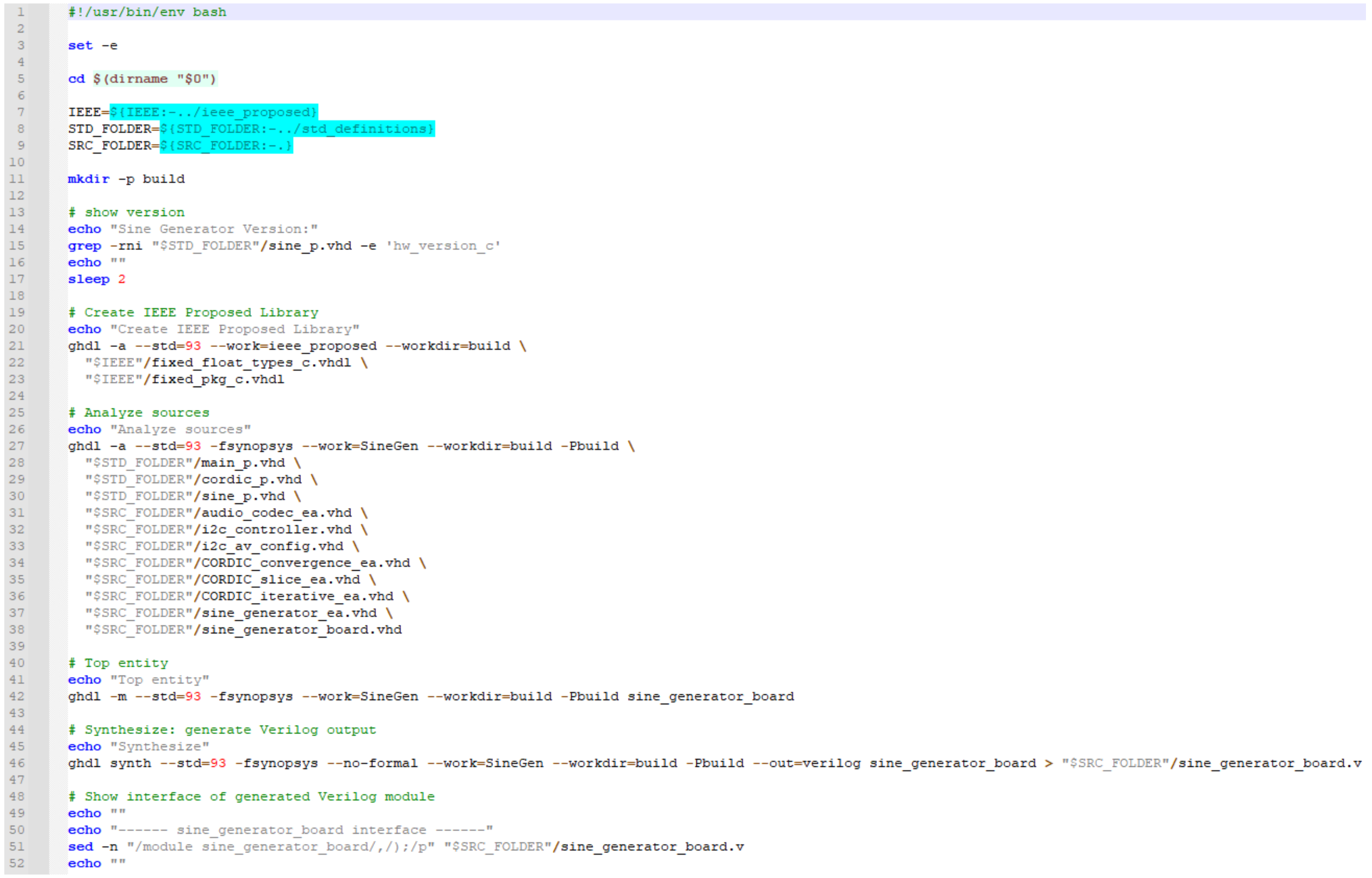
Signals:
```
Time
CONSTANTS
BW=3
SIGNALS
clk_i=1
rst_i=1
counter_val[2:0]=0
counter_val_o[2:0]=0
next_counter_val[2:0]=0
```

8.) Tutorial – Convert VHDL to Verilog

INSTITUTE OF SIGNAL PROCESSING

- In order to convert the VHDL files to one file of Verilog code, a shell script should be written.
- For this tutorial the shell script (*convert.sh*) is already written and can be used as a template for upcoming projects.
- All the commands and parameters can be found in the GHDL manual
- Execute the shell with *“sh convert.sh”*
- For single files: *yosys -m ghdl -p 'ghdl filename.vhd -e entityname; write_verilog filename.v'*
- In order to test the Verilog code, one can synthesize the created code in Quartus and test it on a FPGA board and / or compare the amount of ALMs and registers.

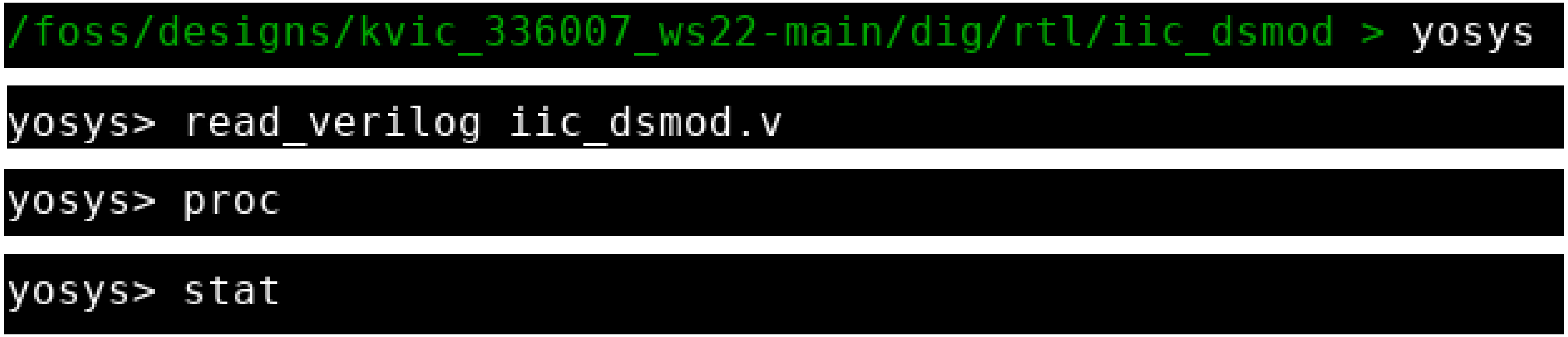
8.) Tutorial – Convert VHDL to Verilog



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

set -e

cd $(dirname "$0")

IEEE=${IEEE:-../ieee_proposed}
STD_FOLDER=${STD_FOLDER:-../std_definitions}
SRC_FOLDER=${SRC_FOLDER:-.}

mkdir -p build

# show version
echo "Sine Generator Version:"
grep -rni "$STD_FOLDER"/sine_p.vhd -e 'hw_version_c'
echo ""
sleep 2

# Create IEEE Proposed Library
echo "Create IEEE Proposed Library"
ghdl -a --std=93 --work=ieee_proposed --workdir=build \
  "$IEEE"/fixed_float_types_c.vhdl \
  "$IEEE"/fixed_pkg_c.vhdl

# Analyze sources
echo "Analyze sources"
ghdl -a --std=93 -fsynopsys --work=SineGen --workdir=build -Pbuild \
  "$STD_FOLDER"/main_p.vhd \
  "$STD_FOLDER"/cordic_p.vhd \
  "$STD_FOLDER"/sine_p.vhd \
  "$SRC_FOLDER"/audio_codec_ea.vhd \
  "$SRC_FOLDER"/i2c_controller.vhd \
  "$SRC_FOLDER"/i2c_av_config.vhd \
  "$SRC_FOLDER"/CORDIC_convergence_ea.vhd \
  "$SRC_FOLDER"/CORDIC_slice_ea.vhd \
  "$SRC_FOLDER"/CORDIC_iterative_ea.vhd \
  "$SRC_FOLDER"/sine_generator_ea.vhd \
  "$SRC_FOLDER"/sine_generator_board.vhd

# Top entity
echo "Top entity"
ghdl -m --std=93 -fsynopsys --work=SineGen --workdir=build -Pbuild sine_generator_board

# Synthesize: generate Verilog output
echo "Synthesize"
ghdl synth --std=93 -fsynopsys --no-formal --work=SineGen --workdir=build -Pbuild --out=verilog sine_generator_board > "$SRC_FOLDER"/sine_generator_board.v

# Show interface of generated Verilog module
echo ""
echo "------ sine_generator_board interface ------"
sed -n "/module sine_generator_board/,/);/p" "$SRC_FOLDER"/sine_generator_board.v
echo ""
```

8.) Tutorial – Showing Stats

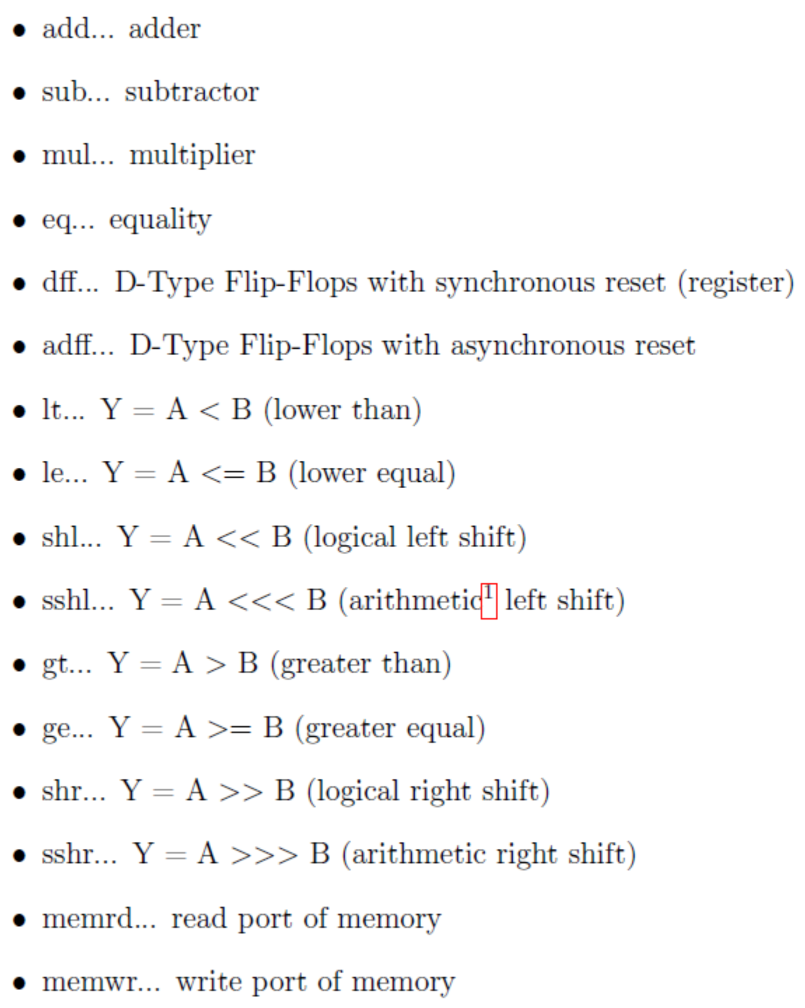
- The following figure shows the sequence of CMD commands to output the needed hardware for the written architecture.



```
/foss/designs/kvic_336007_ws22-main/dig/rtl/iic_dsmod > yosys
yosys> read_verilog iic_dsmod.v
yosys> proc
yosys> stat
```

JOHANNES KEPLER UNIVERSITÄT LINZ

8.) Tutorial – Showing Stats



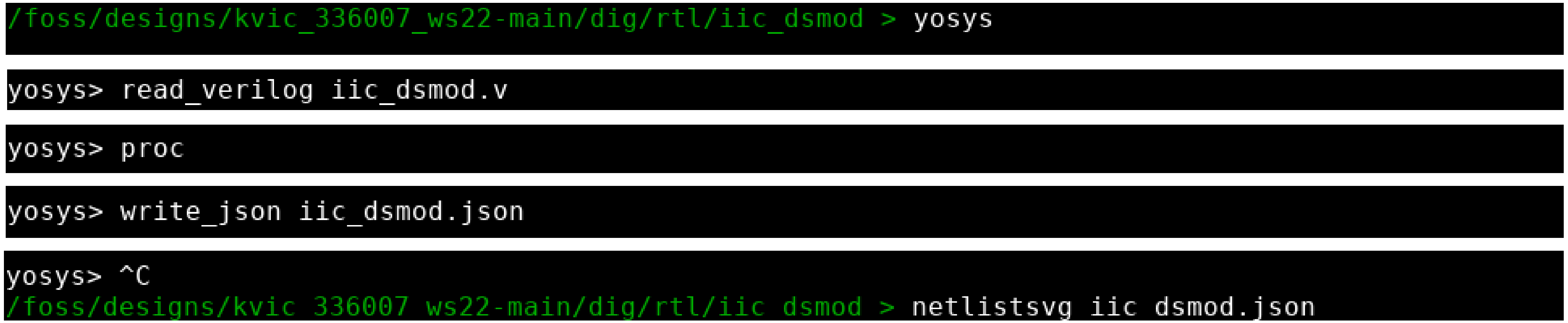
- add... adder
- sub... subtractor
- mul... multiplier
- eq... equality
- dff... D-Type Flip-Flops with synchronous reset (register)
- adff... D-Type Flip-Flops with asynchronous reset
- lt... Y = A < B (lower than)
- le... Y = A <= B (lower equal)
- shl... Y = A << B (logical left shift)
- sshl... Y = 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“.

8.) Tutorial – Drawing RTL View

INSTITUTE OF SIGNAL PROCESSING

- The following figure shows the sequence of CMD commands to create the RTL view in .svg file format.



```
/foss/designs/kvic_336007_ws22-main/dig/rtl/iic_dsmod > yosys
```

```
yosys> read_verilog iic_dsmod.v
```

```
yosys> proc
```

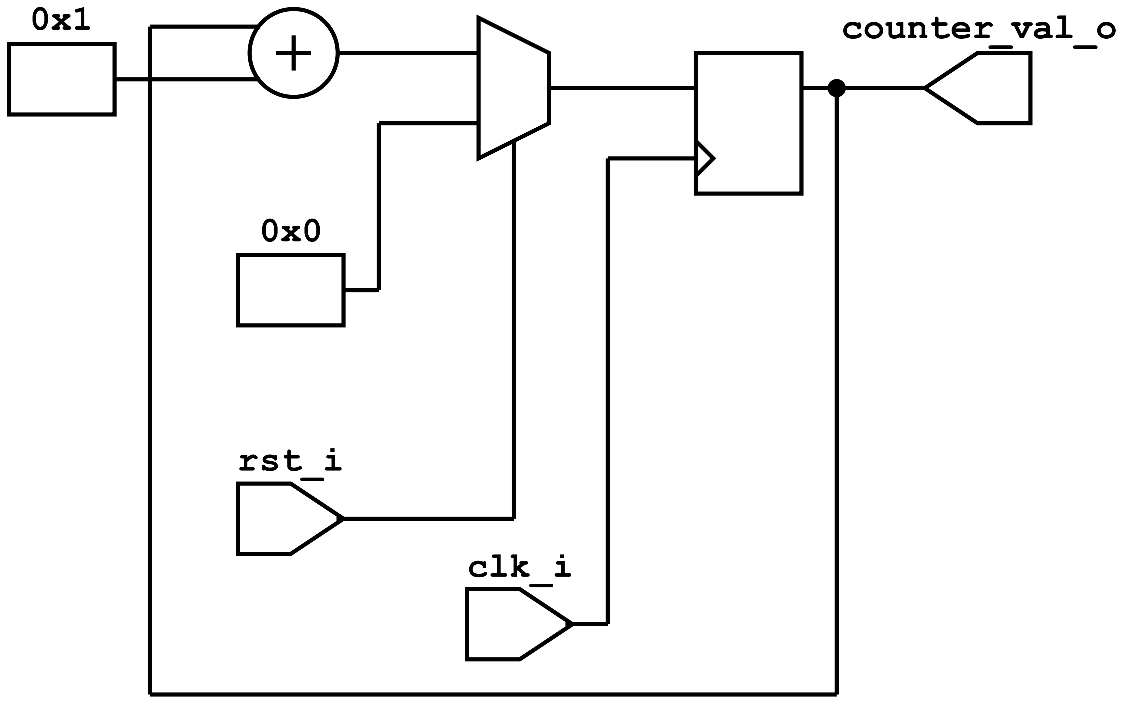
```
yosys> write_json iic_dsmod.json
```

```
yosys> ^C
/foss/designs/kvic 336007 ws22-main/dig/rtl/iic dsmod > netlistsvg iic dsmod.json
```

JOHANNES KEPLER
UNIVERSITÄT LINZ

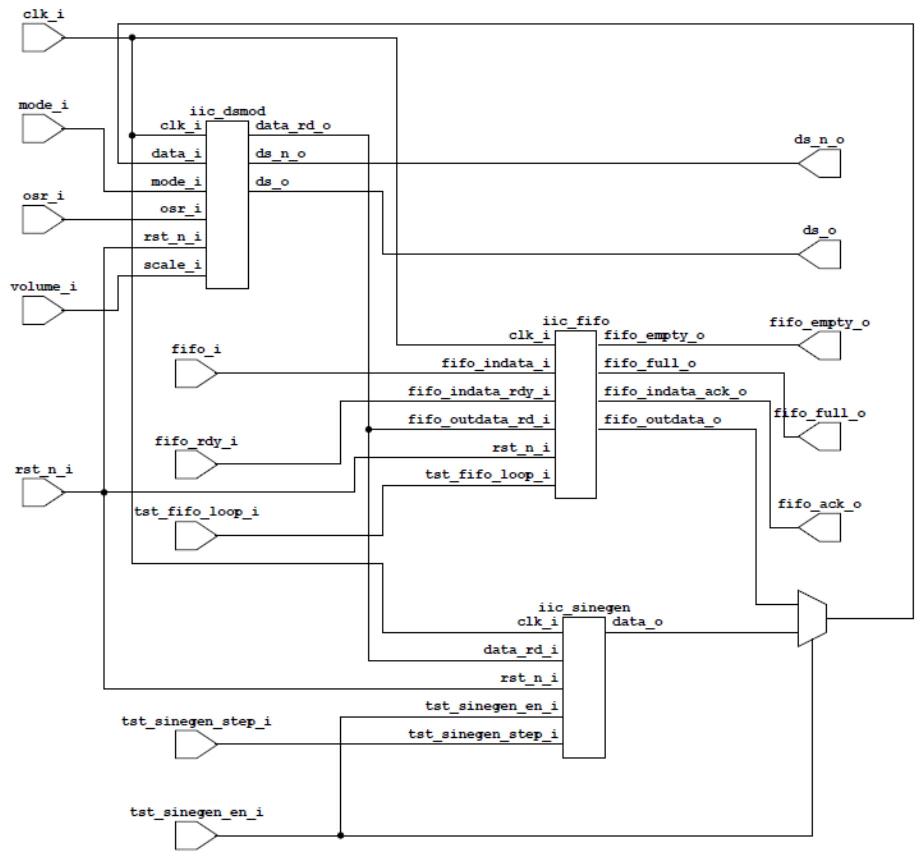
8.) Tutorial – Drawing RTL View

- RTL view of the counter tutorial



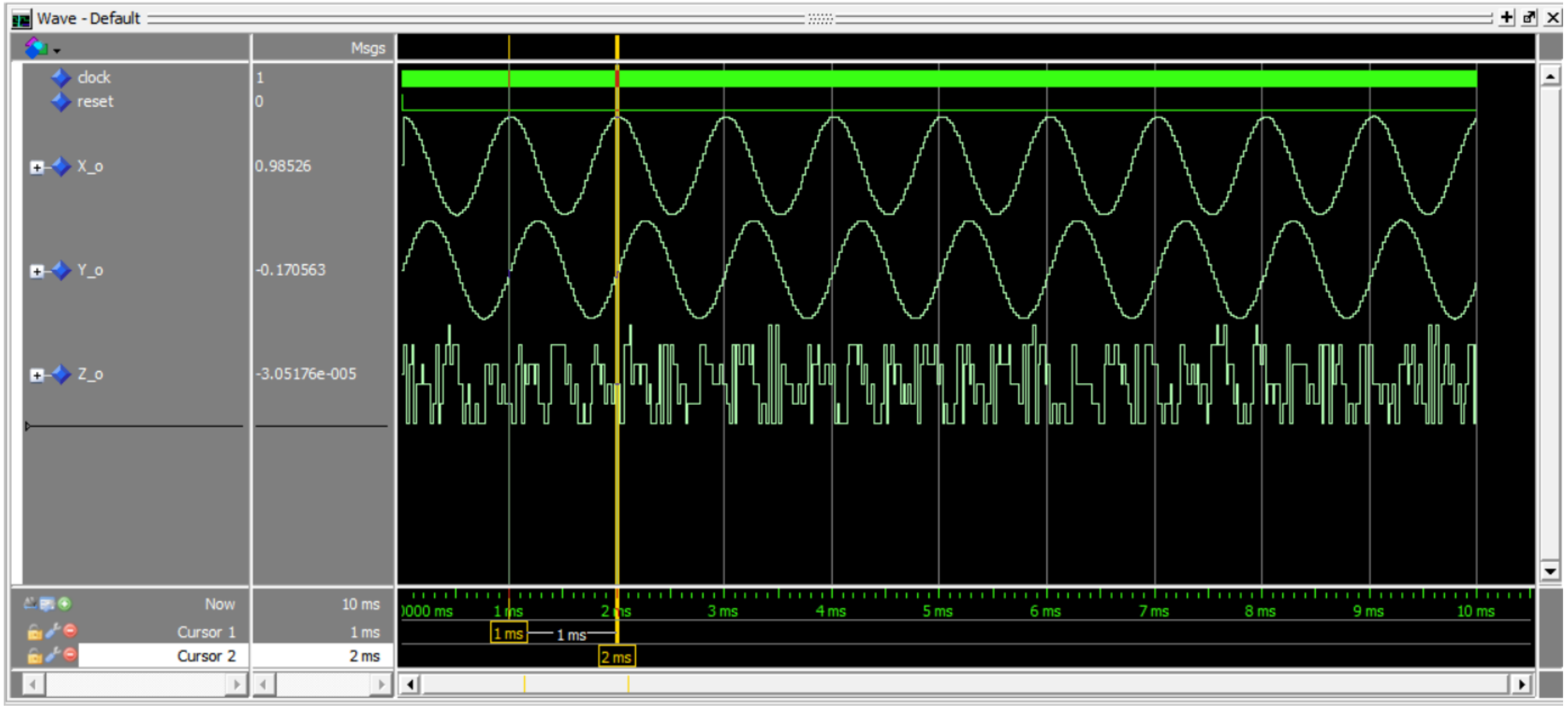
8.) Tutorial – Drawing RTL View

- RTL view of a digital Sigma Delta Modulator with FIFO and Sinegen



8.) Tutorial – Compare VHDL / Verilog

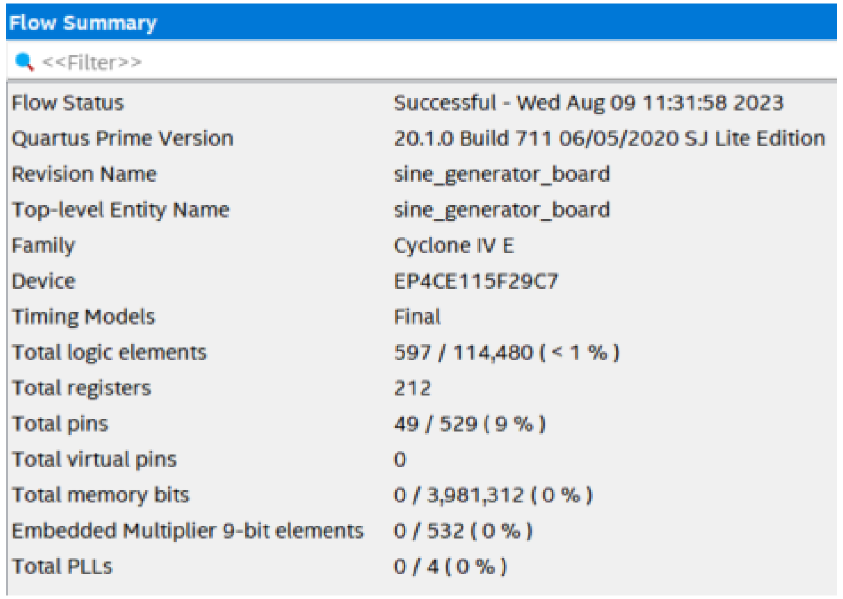
- Mixed VHDL / Verilog Simulation with ModelSim



```
clock     1
reset     0
X_o       0.98526
Y_o       -0.170563
Z_o       -3.05176e-005
Now       10 ms
Cursor 1  1 ms
Cursor 2  2 ms
```

8.) Tutorial – Compare VHDL / Verilog

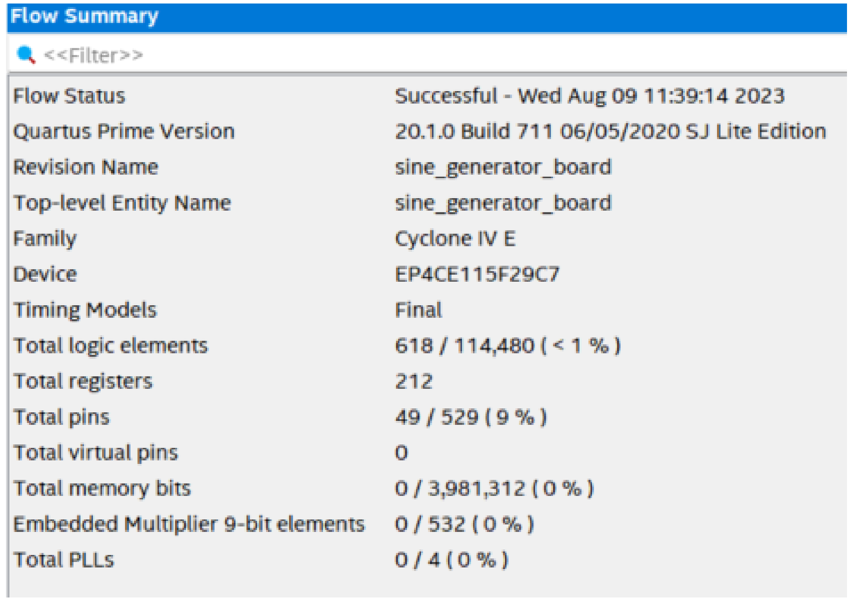
- Left: VHDL
- Right: Verilog



Flow Summary

<<Filter>>

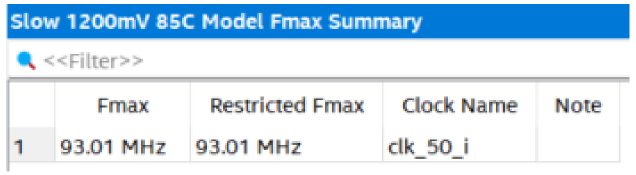
Flow Status	Successful - Wed Aug 09 11:31:58 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	sine_generator_board
Top-level Entity Name	sine_generator_board
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	597 / 114,480 (< 1 %)
Total registers	212
Total pins	49 / 529 (9 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)



Flow Summary

<<Filter>>

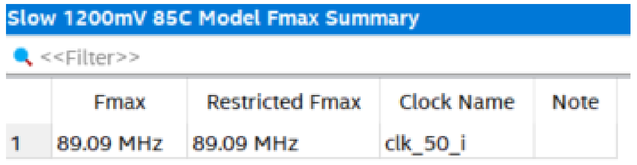
Flow Status	Successful - Wed Aug 09 11:39:14 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	sine_generator_board
Top-level Entity Name	sine_generator_board
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	618 / 114,480 (< 1 %)
Total registers	212
Total pins	49 / 529 (9 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)



Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	93.01 MHz	93.01 MHz	clk_50_i	



Slow 1200mV 85C Model Fmax Summary

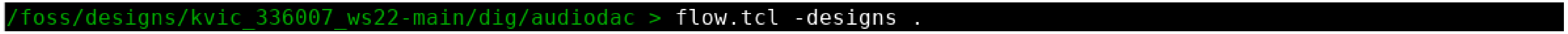
<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	89.09 MHz	89.09 MHz	clk_50_i	

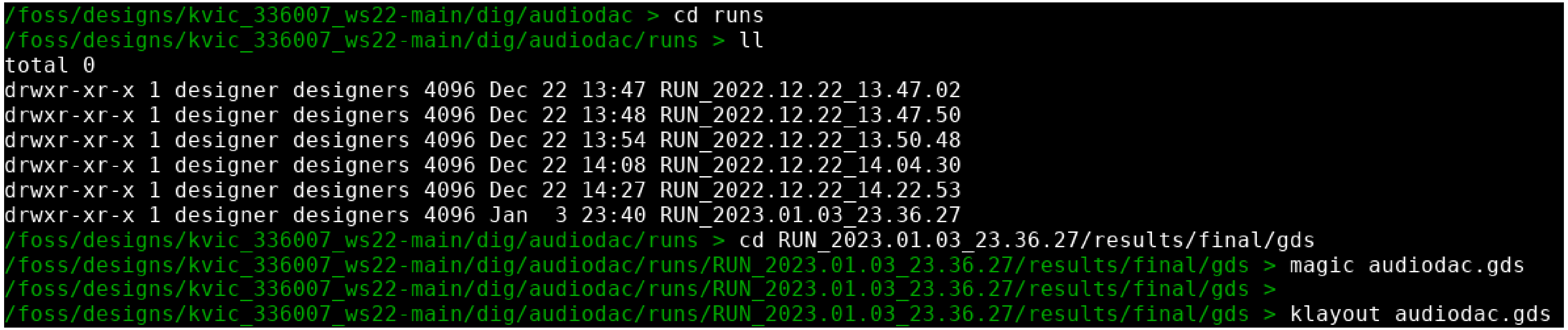
8.) Tutorial – OpenLane

- 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!

8.) Tutorial – OpenLane



```
/foss/designs/kvic_336007_ws22-main/dig/audiodac > flow.tcl -designs .
```

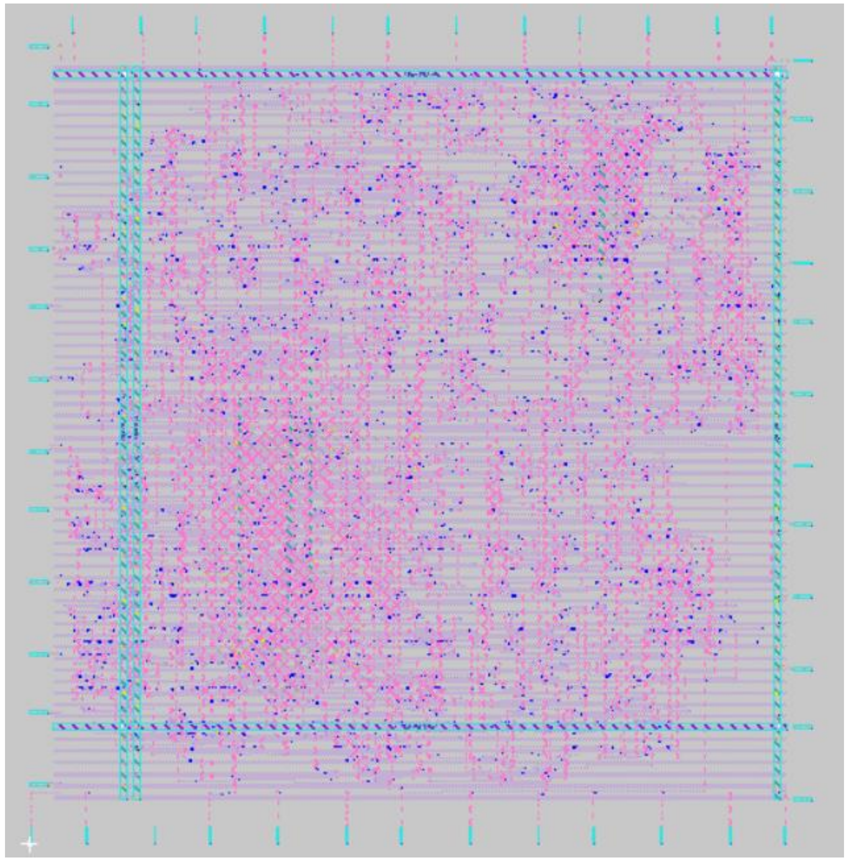


```
/foss/designs/kvic_336007_ws22-main/dig/audiodac > cd runs
/foss/designs/kvic_336007_ws22-main/dig/audiodac/runs > ll
total 0
drwxr-xr-x 1 designer designers 4096 Dec 22 13:47 RUN_2022.12.22_13.47.02
drwxr-xr-x 1 designer designers 4096 Dec 22 13:48 RUN_2022.12.22_13.47.50
drwxr-xr-x 1 designer designers 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
```

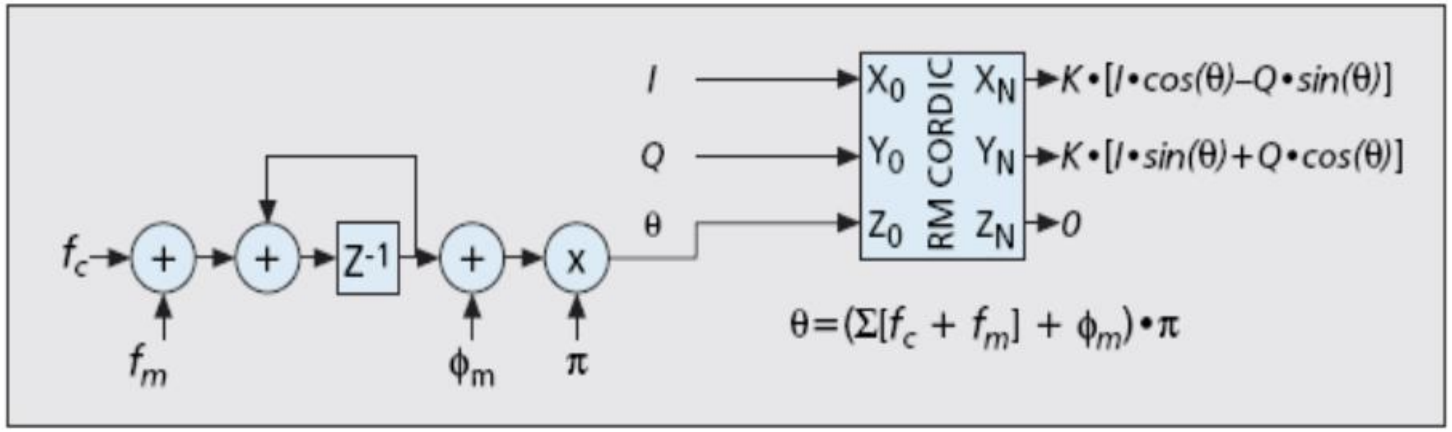
- Verilog: *“./designs/design_name/runs/RUN_XYZ/results/final/verilog”*
- GDS: *“./designs/design_name/runs/RUN_XYZ/results/final/gds”*
- Reports: *“./designs/design_name/runs/RUN_XYZ/reports”*

8.) Tutorial – OpenLane

- Final digital layout of the sine generator using a 16-Bit CORDIC



8.) Tutorial – Sine Generator with CORDIC



I
Q
θ
f_c
f_m
Z^{-1}
ϕ_m
π
RM CORDIC
X_0
Y_0
Z_0
X_N
Y_N
Z_N
$K \cdot [I \cdot \cos(\theta) - Q \cdot \sin(\theta)]$
$K \cdot [I \cdot \sin(\theta) + Q \cdot \cos(\theta)]$
0
$\theta = (\Sigma[f_c + f_m] + \phi_m) \cdot \pi$

- To generate sine and cosine waveforms of a digital frequency fc with the generic scheme above, we can choose:
  - I = 1/K and Q = 0
  - $f_m = 0$, $\phi_m = 0$

$$f = f_c \cdot \frac{f_s}{2}$$