Development of an open-source Place and Route highly scalable tool: *scalepnr*

https://github.com/mirekez/scalepnr

First version of the place-and-route algorithm

7315 total unique lines of code delivered, not including lib/ folder

Mike Reznikov
Tivat
31.07.2025

Table of contents

General considerations and notes

Source code links and details

The development is going in the following repository:

https://github.com/mirekez/scalepnr

Testing OS and conditions, projects were used in testing, other notes

1. The development is targeted at two OS types: Win64 and Linux.
2. All experiments were run under Virtual Machine with OS Linux Ubuntu 22.04 Server using 16 CPU cores and 32GB RAM.
3. None of compiled designs was tested in hardware yet. Third party tool was used for quality comparison.
4. All considered toolchains use *Yosys* open source synthesis tool in *synth_xilinx* mode and objectively *Yosys* plays a significant role in Place and Route process.
5. GTP transceivers, IOSERDES, DSP and BRAM blocks were not touched yet in this work.

The projects were used in testing:

https://github.com/openXC7/demo-projects

https://github.com/chili-chips-ba/openXC7-TetriSaraj

https://github.com/chili-chips-ba/openeye-CamSI

https://github.com/ZipCPU/kimos

Auxiliary projects and tools used during testing and evaluation:

Own sources developed during this work:

https://github.com/mirekez/pnr_tests

https://github.com/mirekez/scalepnr

https://github.com/f4pga/prjxray

https://github.com/f4pga/prjxray-db

Development process

Requirements to version 1.0

First step of *scalepnr* development process contains several tasks, important in terms of architecture and expected required project qualities formation. While main Place and Routing algorithms are developed in brief, the whole project integrity together with libraries and tools is delivered. The number of auxiliary engines also delivered to provide work with RTL, work with FPGA components, databases, global optimization, images production, tcl scripting, etc, to satisfy the following agreed formula for current development step:

"We will design a simplified version of the place-and-route algorithm where the emphasis will be given to the full-feature functionality in terms of the formalized problem statement from task 1, but not necessarily with the focus on the quality of the solutions, scalability, and the computational speed."

Main requirements to development of first version of the product are listed in **Table 1**.

Number	*Requirement*	*Notes*
1	Propose and implement the placing algorithm	Intense work was done in direction of global optimization while basic packing algorithms was used
2	Propose and implement the routing algorithm	Basic routing algorithm was used
3	The place-and-route algorithm implementation and the technical report with the benchmarking results	Current document

Table 1. *Requirements for the version 1.0*

Additional requirements

Some additional requirements were introduced from architecture point of view to allow open and incremental development process. Considering

The following list of additional requirements was prepared:

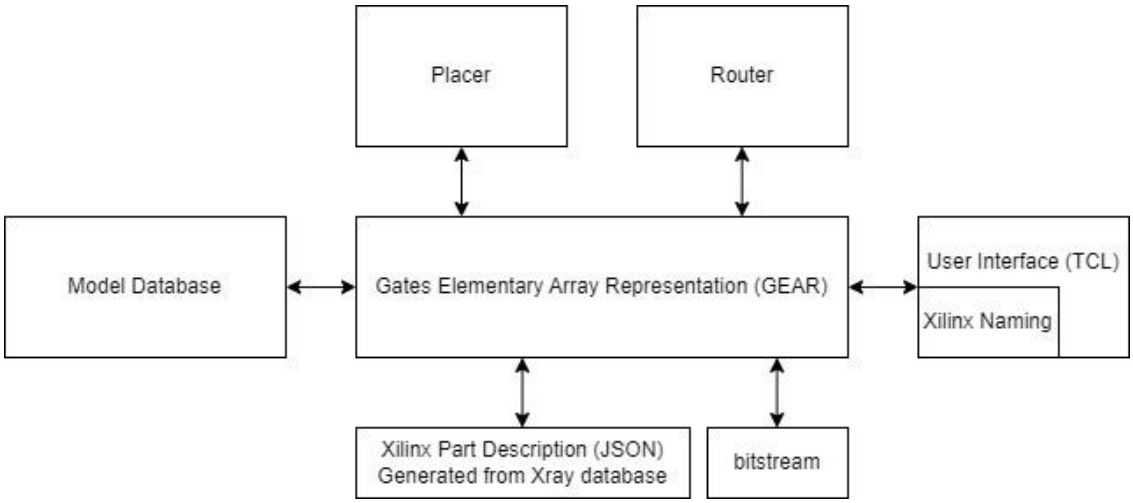
1. Code should be maximally structured and partitioned to small details. Clear hierarchy should be used in each file path in the project.
2. Project should be opened for future development of other vendors hardware support.
3. Project should compile in 2-3 steps on any supported Operation System connected to Internet network.
4. It should be possible to implement scripting as done in modern IDEs
5. Project should use binary representation of objects and should not use textual search during placing and routing procedures.
6. Main algorithms should be developed with clear idea allowing future work on optimization.

7. Code commenting and other good coding practices should be used to provide readable and sustainable code.

Conclusions

First version development is mostly a strategic step to initialize the process in right way allowing implementation of all further requirements with time. Together with development of the first step the whole idea of possibility of development of this project in “open” way is checked (easy start, easy compile, easy read, easy collaborate).

Architecture of the *scalepnr* project



Picture 1. Principal schema of modules interaction in *scalepnr*

Source code structure

Source code was structured in a number of modules allowing separate modification of some blocks not interfering other blocks for future teamwork (**Table 2**).

Description	*Link to the issue*
db	Code for *prjxray* database import
libs/abseil-cpp	Open source google library for string processing, etc
libs/jsoncpp	Open source library to parse json
libs/re2	Open source library for fast regexp implementation
libs/tcl8.6.14	Open source library for tcl scripting implementation
src/clocks	Clocks storage, algorithms and functions to setup and estimate timing including multiclock

src/common	Things like debugging, verbose, etc
src/format	Formatting functions mostly to read and store textual databases with devices and RTL
src/fpga/xc7	XC7 code which makes all parts move together
src/pnr/estimate	Algorithms and functions to estimate RTL in terms of placing (calculation of groups, distances, etc)
src/pnr/outline	Algorithms and functions for design outlining, i.e. global placing optimization
src/pnr/place	Algorithms and functions for direct packing of RTL into BELs
src/pnr/route	Algorithms and functions for routing implementation
src/rtl	Algorithms and functions to store and process RTL
src/tcl	All work of scalepnr is implemented as a number of TCL functions with parameters allowing scripting
src/tech	Technology-specific functions for devices like XC7
src/utils	Generic utils
.	CMakefiles, conda files, main file, compilation scripts

Table 2. *Source code structure of scalepnr project*

Other important characteristics of the source code database:

1. *Conda* package management system with channel *conda-forge* for *win-64* and *lin-64* is used to easily get particular version of compiler for both operation systems. Generally it should work for all Win10/11 and modern Linux operation systems but it was not tested yet.
2. Very important problem was integration of all libraries keeping only required source code (not taking full projects) allowing fast and smooth compilation, fewer dependencies and ability to fix problems in that libraries.
3. Libpng is used to generate images of intermediate result like outline, placing and routing to allow feedback stage after forward passing of PnR.
4. Yosys JSON format is used to provide RTL database of compiled project sources.

As result of these points successful implementation the whole project looks solid and allows easy start of development from almost any modern setup.

Class hierarchy

Implemented class hierarchy is presented in **Table 3**.

Class/struct name	*Purpose*	*Comments*
Design	The whole RTL	
Module	One module	
Cell	Instantiated module	
Clock	Clock object	Automatically extracted
Port	Port of a Cell	

Net	Alias for connection or bus	Useful for VCC, GND, CLK
Conn	Conn between Ports	Two cells can be connected using a number of Conns
Device	FPGA device	
Pin	FPGA pin	
PinType	In, out, bidir, clk, etc	
Tile	The main object in mesh	
TileType	CLB, RAM, DSP, etc	
Wire	Link between Tiles ports	Many wires can be used to connect two Tiles
WireType	Particular wire type	
Clocks	Storage of clocks	
Timings	Storage for timings calculations	
TimingPath	Path to calculate setups and holds	

Table 2. *Main class hierarchy used in scalepnr*

PnR algorithms used

Place and Route is implemented with emphasis on global optimization in first version of *scalepnr* tool. **Table 3** shows implemented stages and approaches used in first version development.

Task	*Approach, algorithm*	*Notes*
Estimation	The purpose of estimation stage is to find the most critical chains of combinational cells. Using timing information and RTL navigation algorithm calculates rating of each part of RTL, sorting cells in a large list where the most critical cells and paths go in the beginning.	Algorithms already looks crucial for reasonable PnR and tradeoff in cases of conflicting blocks present. It already support multi-clock model
Outline	The Force-Directed Graph Drawing (or Spring layout) stepping algorithm is used to optimize global layout. Algorithm uses nets between bunches of RTL as elastic objects to make tradeoff for placement when multiple constraints are applied.	https://en.wikipedia.org/wiki/NetworkX Algorithms already look useful while there is still question regarding performance during scaling to 500K or 1M objects
Placing	Direct nearest-free Tile packing algorithm is used based in global optimization results.	Algorithm should be enhanced during work with larger designs > 100K objects Packing should contain combinatorial optimization of packing in near-zone

Routing	Direct “greedy” algorithm is used for routing	Algorithm should be enhanced during work with larger designs > 100K objects A lot of further work should be done to make routing feasible and transparent for modifications

Table 3. *Main algorithms and approaches used in scalepnr*

PnR algorithms discussion

While it was not possible to organize deep study on the science of placing and routing optimization in the first version, the main priority was given to feasible global optimization. This allows testing of the main classes and objects of the architecture to be efficiently used in algorithmic processing of thousands to million units and allows further work.

Two first stages got higher priority in first version of the program, to highlight the main optimization approach as “divide, (sort) and conquer”. The most attention should be done to the most critical paths and regions of RTL with highest potential timing slack or routing congestion.

The whole design is first split into bunches of combinational logic blocks. Each block’s chains are estimated, finding the most critical considering set up clock constraints. The number of fan-in and fan-outs is counted. The the most critical parts of RTL is put into the beginning of the PnR “todo” list.

Brief output and using guides

User interface

User interface of *scalepnr* application is implemented with TCL command system. Therefore *scalepnt* is an interpreter of TCL commands list or script, containing all work inputs and commands. **Table 4** contains main tcl commands list currently supported.

Command	*Description*	*Notes*
check_timing	Runs timings check and clocks calculations	Currently supports only pre-PnR calculations
create_clock	Creates clock constraint	Multi-clock is possible but not all algorithms are implemented with multi-clock support

get_ports	Finds ports by name or regexp mask	
get_tiles	Finds device tiles by name or regexp mask	
load_design	Loads RTL from Yosys JSON	
open_design	Performs RTL processing and estimation	
place_design	Does design placing	
print_design	Prints design or it's part in ASCII hierarchical format	
set_property	Sets PINs assignments and constraints	
route_design	Does design routing	

Table 4. *TCL commands supported in scalepnr*

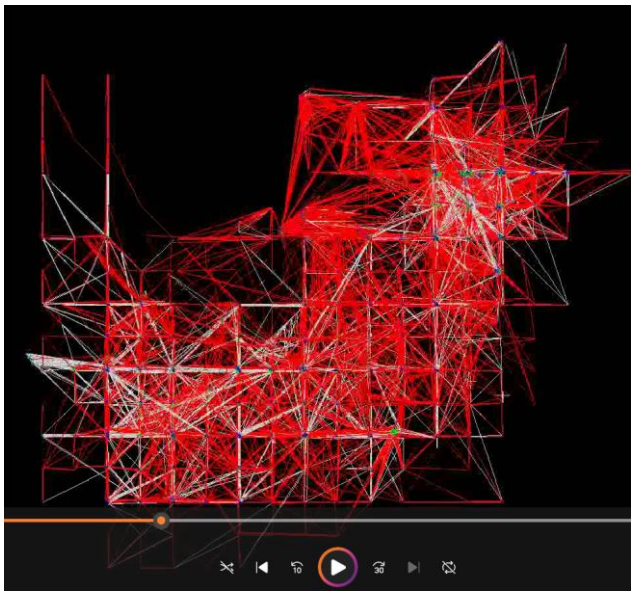
Example TCL script for processing of *TestMesh.json* RTL is presented on **Picture 2.**

```
/home/me/scalepnr/build/commands.tcl   [----]  0 L:[  1+
load_design TestMesh.json TestMesh
create_clock -name clk -period 5.0 [get_ports clock]

set_property IOSTANDARD LVCMOS33 [get_ports clock]
set_property PACKAGE_PIN C9 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN K6 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports in_valid]
set_property PACKAGE_PIN U1 [get_ports in_valid]
set_property IOSTANDARD LVCMOS33 [get_ports in_ready]
set_property PACKAGE_PIN B1 [get_ports in_ready]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[0]]
set_property PACKAGE_PIN N17 [get_ports in_bits[0]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[1]]
set_property PACKAGE_PIN C17 [get_ports in_bits[1]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[2]]
set_property PACKAGE_PIN T16 [get_ports in_bits[2]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[3]]
set_property PACKAGE_PIN K13 [get_ports in_bits[3]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[4]]
set_property PACKAGE_PIN R7 [get_ports in_bits[4]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[5]]
set_property PACKAGE_PIN J4 [get_ports in_bits[5]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[6]]
set_property PACKAGE_PIN N2 [get_ports in_bits[6]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[7]]
set_property PACKAGE_PIN G3 [get_ports in_bits[7]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[8]]
set_property PACKAGE_PIN E17 [get_ports in_bits[8]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[9]]
set_property PACKAGE_PIN H14 [get_ports in_bits[9]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[10]]
set_property PACKAGE_PIN E6 [get_ports in_bits[10]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[11]]
set_property PACKAGE_PIN R11 [get_ports in_bits[11]]
set_property IOSTANDARD LVCMOS33 [get_ports in_bits[12]]
set_property PACKAGE_PIN B3 [get_ports in_bits[12]]
set_property PACKAGE_PIN M16 [get_ports out_bits[23]]
set_property IOSTANDARD LVCMOS33 [get_ports out_bits[24]]
set_property PACKAGE_PIN J15 [get_ports out_bits[24]]
set_property IOSTANDARD LVCMOS33 [get_ports out_bits[25]]
set_property PACKAGE_PIN A3 [get_ports out_bits[25]]
set_property IOSTANDARD LVCMOS33 [get_ports out_bits[26]]
set_property PACKAGE_PIN B6 [get_ports out_bits[26]]
set_property IOSTANDARD LVCMOS33 [get_ports out_bits[27]]
set_property PACKAGE_PIN G17 [get_ports out_bits[27]]
set_property IOSTANDARD LVCMOS33 [get_ports out_bits[28]]
set_property PACKAGE_PIN L16 [get_ports out_bits[28]]
set_property IOSTANDARD LVCMOS33 [get_ports out_bits[29]]
set_property PACKAGE_PIN M18 [get_ports out_bits[29]]
set_property IOSTANDARD LVCMOS33 [get_ports out_bits[30]]
set_property PACKAGE_PIN P2 [get_ports out_bits[30]]
set_property IOSTANDARD LVCMOS33 [get_ports out_bits[31]]
set_property PACKAGE_PIN U16 [get_ports out_bits[31]]

open_design
place_design
route_design
#print_design .*slice.20647.* 1000
```

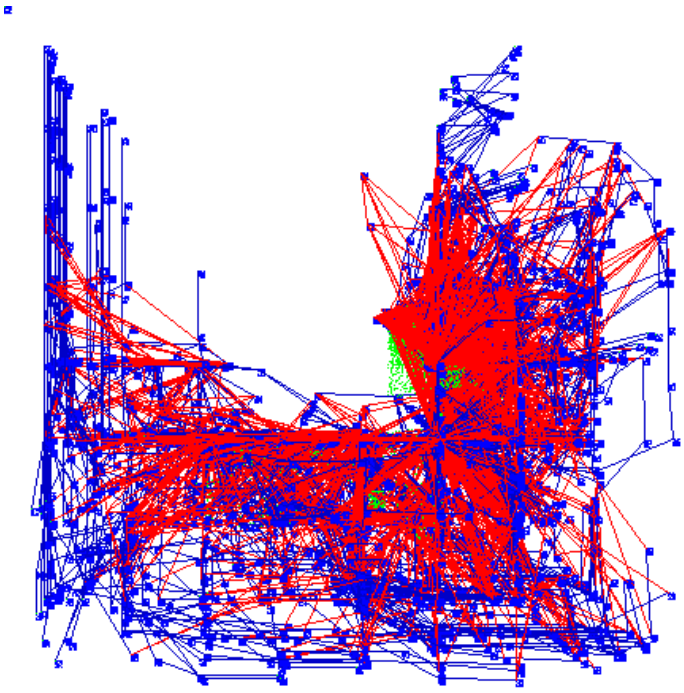
Picture 2. Example script for running *scalepnr* work

Image generation

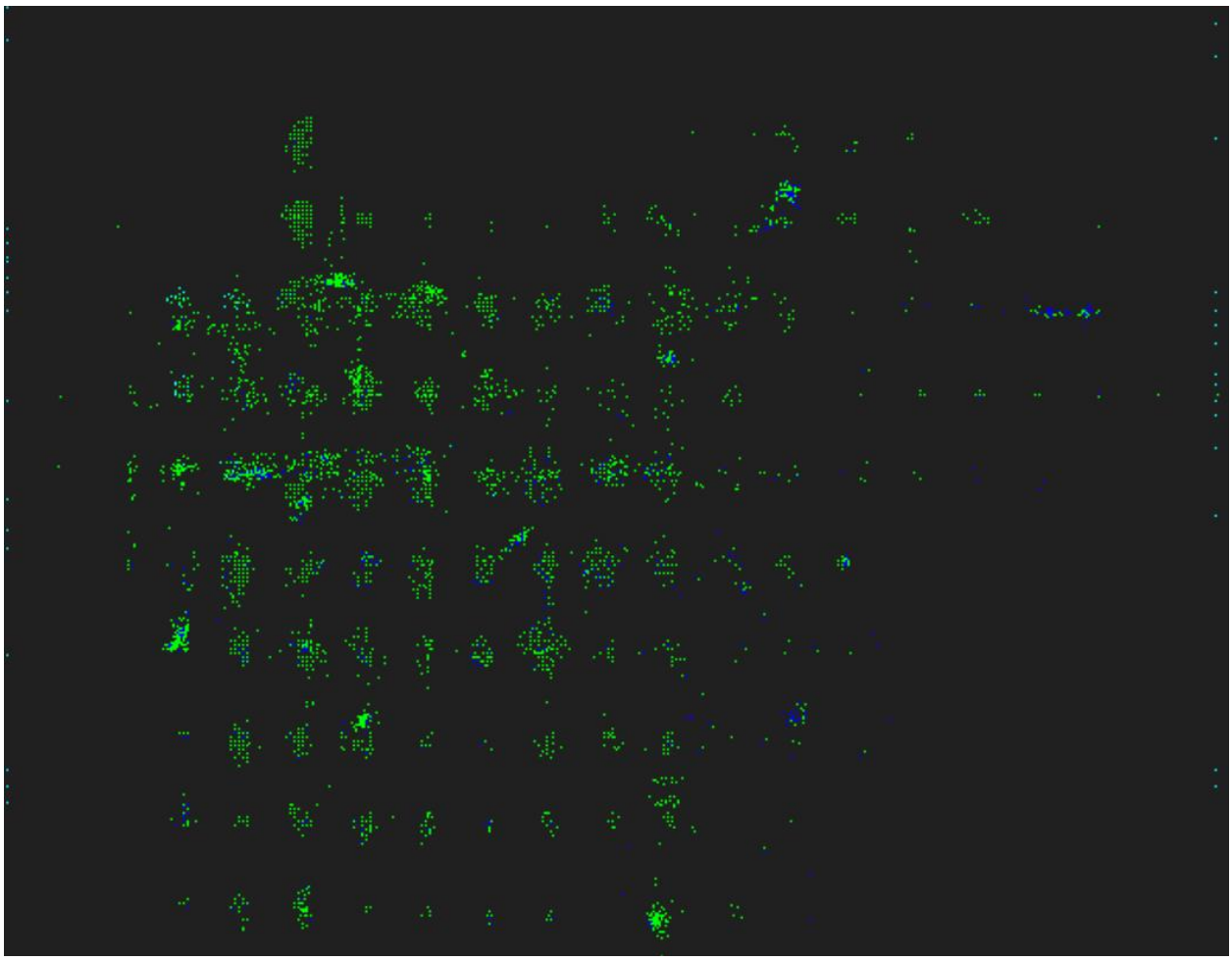
Due to a lack of possibility to analyze the intermediate work results special image generation engine was introduced to provide global optimization and placing results step-by-step visualization. **Pictures 3-6** show example of optimization process stage and placing stage.



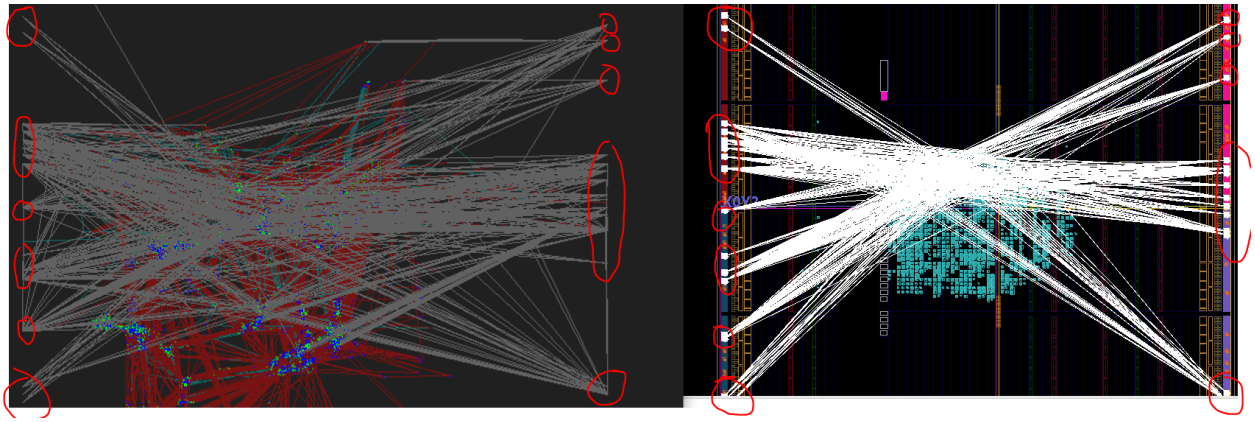
Picture 3. Global optimization stage visualization example in *scalepnr*



Picture 4. Global optimization results visualization example in *scalepnr*



Picture 5. Placing results visualization in *scalepnr*



Picture 6. Pins placing in *scalepnr* in comparison to third party tool

Picture work was not part of requirements but introduced in *salepnr* design to enable feedback of placing (and routing) procedures to estimate the decision making visually.

Benchmarking

Benchmarking vs third party tool

Testing performed using *pnr_tests* project

Model size	Results	Note
10-40 K cells	Results fits third party application slack	
40-80 K cells	TBD	
80-100 K cells	TBD	

Table 5. *Testing conditions*

Work done analysis and conclusions

Main results

Main results achieved during version 1 *scalepnr* project development

1. The architecture of *scalepnr* application was developed
2. The project with all required libraries was build supporting portability and easy compilation
3. Yosys RTL importing and estimation developed
4. Basic image production engine was developed
5. Basic Placing and Routing principal algorithms developed
6. TCL command system designed and tested
7. Global optimization algorithms developed and tested VS third party application

Future work plan

The following work items are proposed for future work:

1. The following enhancements should be done as first priority:
    a. Multi-clock timing calculation system
    b. Timing calculation support during routing tasks
2. Placing packing combinatorial algorithm should be used
3. Scale testing and moving to >100K cells designs
4. Optimization of routing algorithms
5. Development of next version of *scalepnr* with the following requirements:
    a. Propose and implement new version of the placing algorithm
    b. Propose and implement new version of the routing algorithm
    c. New version of the place-and-route algorithm with its implementation and the technical report with the benchmarking results.