

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

<https://github.com/mirekez/scalepnr>

Second version of the place-and-route algorithm

1046 total unique lines of code delivered

Mike Reznikov

Tivat

15.12.2025

Table of contents

General considerations and notes	3
Source code links and details.....	3
Testing OS and conditions, projects were used in testing, other notes.....	3
Development process.....	4
Requirements to version 2.0	4
Additional requirements.....	4
Main changes in v2.....	5
Abstract interfaces and vendor-independent architecture.....	5
Crossbar abstract architecture	5
Source code difference structure	6
Newly introduces classes hierarchy.....	6
New routing algorithm.....	7
Images generation during routing	7
Brief output and using guides.....	8
User interface	8
Work done analysis and conclusions	9
Main results	9
Future work plan.....	9

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 target is at two OS types: Win64 and Linux.
2. All experiments executed under Virtual Machine with OS Linux Ubuntu 22.04 Server using 16 CPU cores and 32GB RAM.
3. 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.
4. GTP transceivers, IOSERDES, DSP and BRAM blocks were not touched yet in this work.

The testing projects used:

<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/prixdray>

<https://github.com/f4pga/prixdray-db>

Development process

Requirements to version 2.0

Next step of *scalepnr* project implies significant changes of algorithms of PnR. According to experience during previous steps, optimizations expected to improve performance and consistency of the most resource consumptive part of the PnR process. General formula for “better quality” of algorithms includes list of projects for testing and general consideration of improvements:

“The experience gained during the work on task 2 will be used to redesign and reimplement the algorithm's core components to meet our expectations about its quality and performance on the selected set of benchmarked instances. The focus will be on providing a better-quality solutions/faster computation time than the first version of the algorithm developed in task 2. We also select new benchmarks depending on their readiness/maturity to test the new capabilities of the place-and-route algorithm. The considered list of benchmarks include: Wireguard project: <https://nlnet.nl/project/KlusterLab-Wireguard/> PTP gateway project: <https://nlnet.nl/project/PTP-timingcard-gateway/> uberDDR3 project: <https://nlnet.nl/project/UberDDR3>, https://github.com/AngeloJacobo/DDR3_Controller TISG project: <https://nlnet.nl/project/TISG/>.”

Main requirements to development stage are listed in **Table 1**.

Number	Requirement	Notes
1	Based on experiments and tests for a list of projects reconsider and probably reinvent some structures and algorithms for better performance or consistency	Mostly architectural and development strategy questions
2	Development of optimized algorithms	
3	Testing stage	

Table 1. Requirements for the version 2.0

Additional requirements

Additional requirements which also took time for development

1. There is need for vendor independent abstraction for device specification.
2. Abstract device specification should not worsen performance and code structure.
3. Overall incremental process of source code structuration should be maintained.

Main changes in v2

Abstract interfaces and vendor-independent architecture

Removed all vendor-related hardware blocks, structures and expressions and replaced by abstract FPGA architecture, containing generic well-known elements. Flexible system of technology descriptions used to *prjxray* vendor-related databases support. This work involves only high-speed C++ approaches, including large 256-bit bitmasks for FPGA crossbar nodes search during the routing process.

During the last architecture review, only the following block types describe now the FPGA internals:

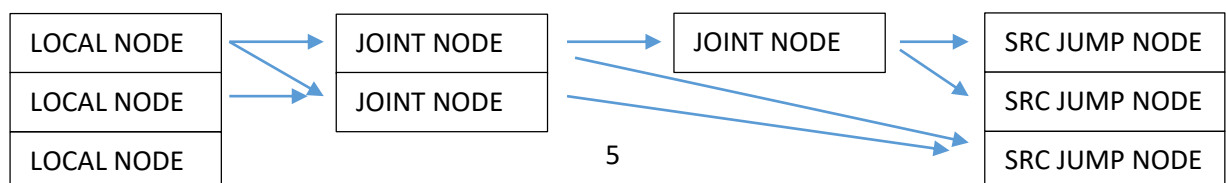
- **Tiles** – 2D array of square blocks, each containing *Logic Elements* and *Crossbox* to connect elements to other Tiles
- **Logic Elements** – RTL level elements like FDxx, LUTx and other. Their names and ports come with RTL, generated by *Yosys* and already contain vendor-specific data.
- **Crossbox** – large 2-level MUX, providing external connectivity for a *Tile*.

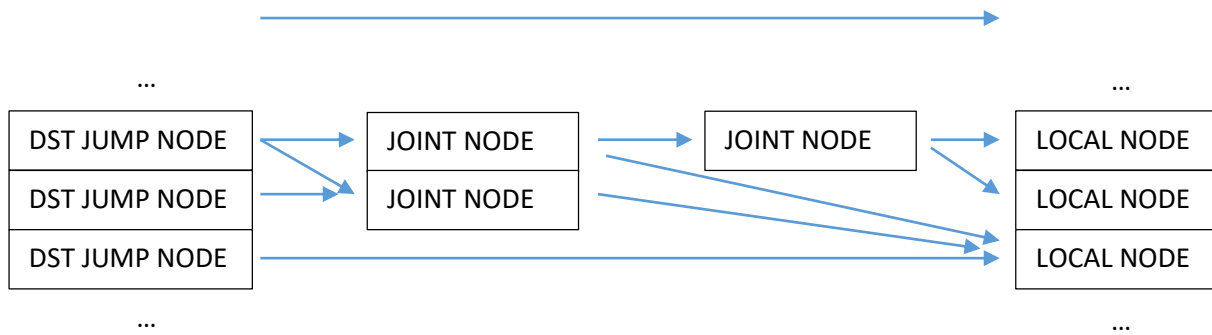
Crossbar abstract architecture

Crossbar abstract architecture used in *scalegnr* uses three types of nodes: LOCAL, JOINT and JUMP. This covers all considered vendor's architectures and allows flexible configuration to use fast bitmap search engine.

- **LOCAL** crossbar nodes are nodes, connected to Logic Elements of a Tile
- **JOINT** crossbar nodes are nodes, which connects LOCAL and other JOINT nodes to external nodes of a crossbar
- **JUMP** nodes are external nodes of crossbar and got their name since used to transfer signal to other Tiles of FPGA. JUMP nodes can be of two types: *SOURCE* and *DESTINATION*. LOCAL nodes and JOINT nodes links Tiles to *SOURCE* nodes, while *DESTINATION* nodes links back to JOINT nodes and LOCAL. All external to crossbar connections are between *SOURCE* and *DESTINATION* JUMP nodes and each such connection contains X and Y coordinates differential jumping values.

The *scalegnr* tool automatically classifies nodes during reading *prjxray* database and builds structures of two types: **ID** and **STATE** per each node. While *ID* is just an enumeration of the node, *STATE* is 256-bit mask to provide particular bit place for each node of same type. This property provides ability to configure all possible connections inside crossbar without using paired string descriptions. Bitmasks deliver next level of optimization to routing algorithm.





Picture 1. Crossbar connectivity architecture

Source code difference structure

The table contains all source code paths altered during this work, description of changes and number of changed lines (**Table 2**).

<i>Filename</i>	<i>Changes made</i>	<i>New lines of code</i>
format/DeviceFormat.h	Changes for reading of <i>prjxray</i> format to new description structures	87
fpga/Crossbar.h	New description of an FPGA crossbox, including 256-bit maps for nodes specification	147
fpga/Crossbar.cpp	Algorithms for search optimization during iterative routing using bitmap for multiple stages of joint nodes search	382
fpga/Device.h	Minor changes to fit new descriptions in old device structures	92
fpga/Tile.cpp	Tile descriptions according to a new abstract architecture of vendor independent FPGA abstraction	89
route/RouteDesign.cpp	Routing algorithms changes according to a new data structures	200
route/RouteDesign.h	Routing algorithms changes according to a new data structures	49
<i>Total</i>		1046

Table 2. Source code structure changes for *scalepnr* project

Newly introduces classes hierarchy

New classes shown in the table **Table 3**.

<i>Class/struct name</i>	<i>Purpose</i>	<i>Comments</i>
Crossbar	Abstract description for Tile Crossbar	

CBType	Node description	
CBJumpNode	Jump node description	
CBLocalNode	Local node description	
CBJointNode	Joint node description	
CBState		
CBJumpState	Jump state description	Uses 256-bit mask
CBLocalState	Local state description	Uses 256-bit mask
CBJointState	Joint state description	Uses 256-bit mask

Table 2. Main class hierarchy changes in *scalepnr*

New routing algorithm

New routing algorithm uses large recursive loop using three levels of recursion: exit source tile recursion, jumping through tiles recursion, entering destination tile recursion. This loop runs for each wire between element's ports and limited by the number of attempts. Bitmasks show, which exit, jump and enter paths are busy. **Table 3** shows description of each routing loop level.

Level	Description and details
Exit source Tile	During start of one wire routing operation, <i>CBSate::iterateOut()</i> method is used, which alternately tries to use each free path to access <i>JUMP</i> node of a <i>Tile's</i> crossbar. Algorithm prioritizes SOURCE JUMP nodes, leading to a jump in direction of destination Tile.
Jumping	Jumping iterates per all accessible peers of DST-SRC nodes of each Tile it passes. Each jump is a recursion with limited number of attempts.
Enter destination Tile	After DESTINATION JUMP node accepted connection for destination Tile, iterations go to connect this node with element's ports. To find if there is a possible way to connect two nodes, joint bitmasks AND operation leads to the search for the bit number which shows path joint node is accessible from both external and local nodes.

Table 3. Routing wire algorithm recursion levels in *scalepnr*

Images generation during routing

Whole process of the toolset development is accomplished with image generation functionality for iterative steps of placing and routing. Image generation is mandatory during early stages of development considering complexity of the input design and FPGA device configuration spaces. This functionality also helps development and analysis of assignments done.

Brief output and using guides

User interface

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

<i>Command</i>	<i>Description</i>	<i>Notes</i>
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**.

Work done analysis and conclusions

Main results

Main results achieved during version 2 *scalepnr* project development

1. New device representation structures were introduced to provide generic device format description using *prjxray* specifications
2. New routing algorithm developed based on abstract crossbox nodes representation using 256-bit maps for accelerated search
3. Yosys RTL importing and estimation developed

Future work plan

Future work requires lots of testing:

1. Testing using random design generation using *pnt_test* utility:
2. Intensive testing using known projects
3. First bitstream generation experiments
4. Series of hardware testing using bitstreams and hardware examples