# 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

1146 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
**Benchmarks ..... 8**
Wireguard project ..... 8
UberDDR3 project ..... 8
UberDDR3 project ..... 8
..... 10
UberDDR3 project ..... 11
UberDDR3 project ..... 11
UberDDR3 project ..... 11
TISG project ..... 12
Larger designs ..... 13
**Work done analysis and conclusions ..... 14**
Main results ..... 14
Future work plan ..... 15

## 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/prjxray

https://github.com/f4pga/prjxray-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: PTP gateware project: https://nlnet.nl/project/PTP-timingcard-gateware/ 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. Overal 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:

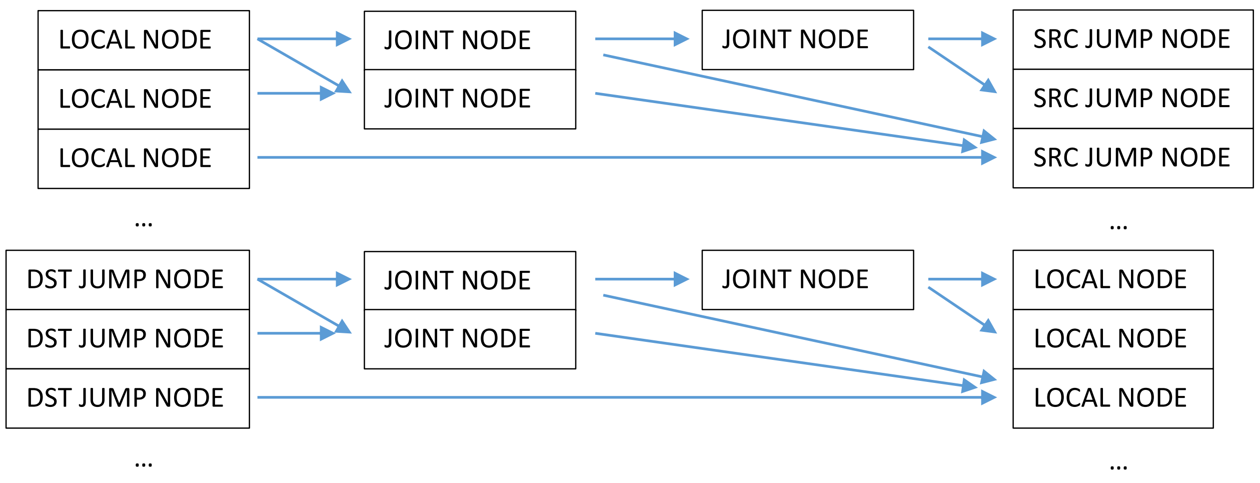
- **Tiles** – 2D array of square blocks, each containing *Logic Elements* and *Crossbox* to connect elements to other Tiles
- **Logic Elements** – RTL level elements like FDxx, LUTx and other. Their names and ports come with RTL, generated by *Yosys* and already contain vendor-specific data.
- **Crossbox**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

## Crossbar abstract architecture

Crossbar abstract architecture used in *scalepnr* 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 *scalepnr* 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**).

| *Filename* | *Changes made* | *New lines of code* |
|---|---|---|
| format/DeviceFormat.h | Changes for reading of *prjxray* format to new description structures | 87 |
| fpga/Crossbar.h | New description of an FPGA crossbox, including 256-bit maps for nodes specification | 127 |
| fpga/Crossbar.cpp | Algorithms for search optimization during iterative routing using bitmap for multiple stages of joint nodes search | 402 |
| 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 | 109 |
| route/RouteDesign.cpp | Routing algorithms changes according to a new data structures | 220 |
| route/RouteDesign.h | Routing algorithms changes according to a new data structures | 69 |
| *Total* | | 1146 |

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

## Newly introduces classes hierarchy

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

| *Class/struct name* | *Purpose* | *Comments* |
|---|---|---|
| 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, *CBSate:: iterateOut()* method is used, which alternately tries to use each free path to access *JUMP* node of a *Tile's* 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 is 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

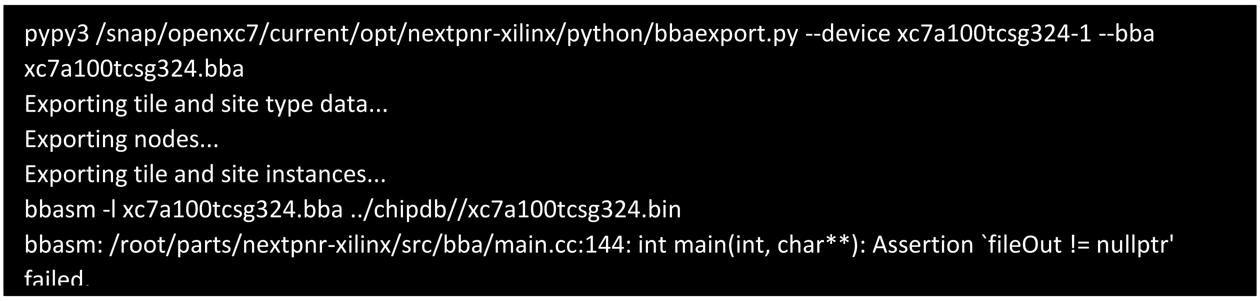
Whole process of the toolset development is accomplished with image generation functionality for iterative steps of placing and routing. Image generation is mandatory during early stages of development considering complexity of the input design and FPGA device configuration spaces. This functionality also helps development and analysis of assignments done and needs to be enhanced further to be able to show routing traces in details with precision.

## Benchmarks

### Wireguard project

Project compilation was in two ways: *nextpnt* compilation and *scalepnr* compilation (does not support bit-file generation tool yet).

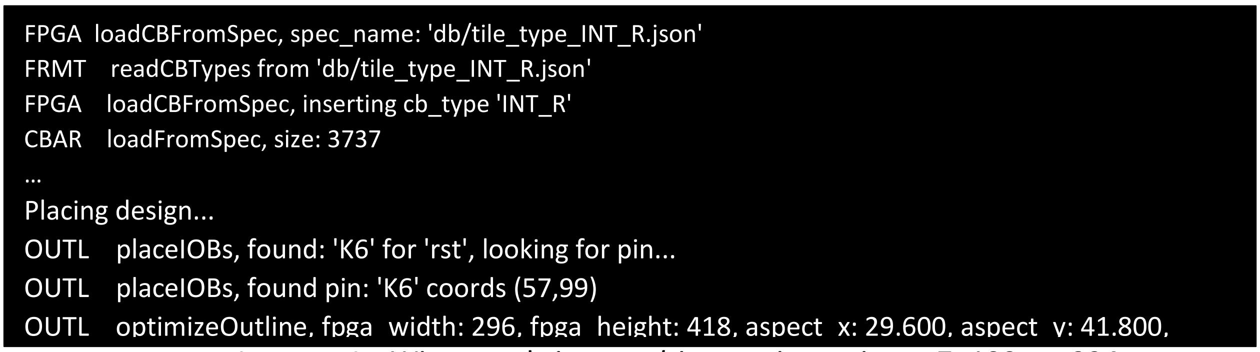
The results of old *nextpnt* tool for *riscv_multicyc*: The exception happens. Process was interrupted, no output.



```
pypy3 /snap/openxc7/current/opt/nextpnr-xilinx/python/bbaexport.py --device xc7a100tcsg324-1 --bba
xc7a100tcsg324.bba
Exporting tile and site type data...
Exporting nodes...
Exporting tile and site instances...
bbasm -l xc7a100tcsg324.bba ../chipdb//xc7a100tcsg324.bin
bbasm: /root/parts/nextpnr-xilinx/src/bba/main.cc:144: int main(int, char**): Assertion `fileOut != nullptr'
failed.
```

Picture 2. Output of *nextpnr* for Wineguard *riscv_multicyc* project using *xc7a100tcsg324* part

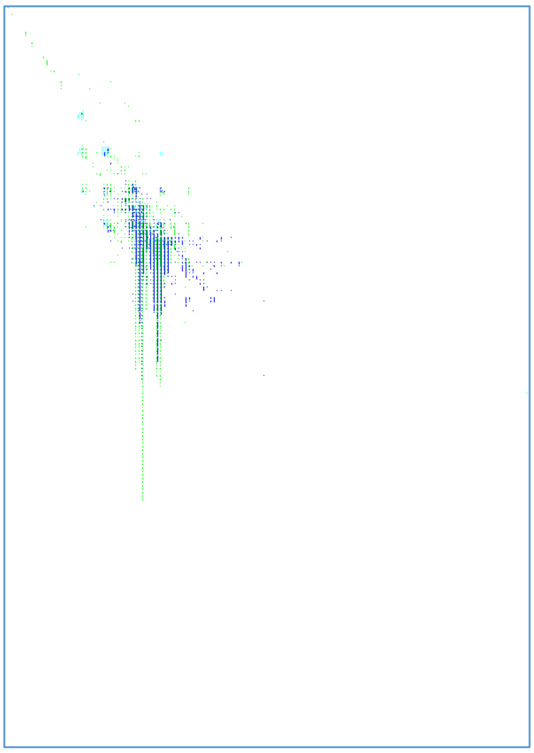
The results of a new *scalepnr* tool:



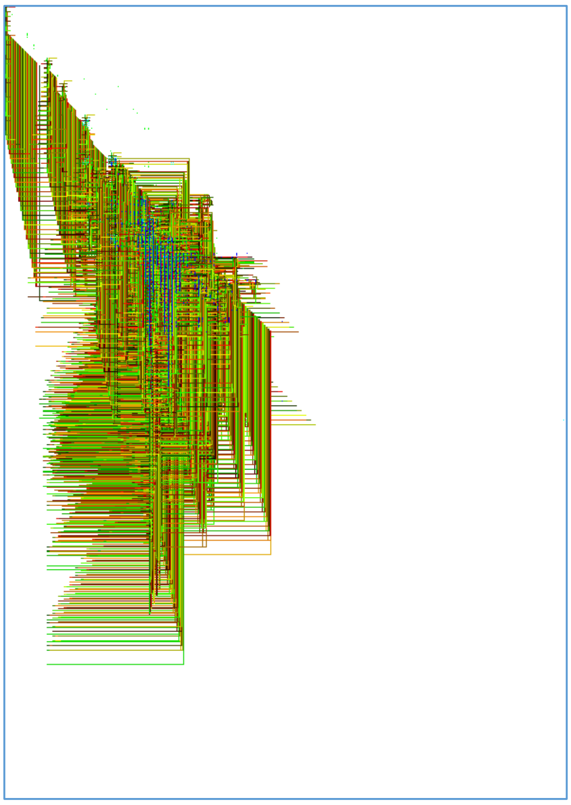
```
FPGA  loadCBFromSpec, spec_name: 'db/tile_type_INT_R.json'
FRMT    readCBTypes from 'db/tile_type_INT_R.json'
FPGA    loadCBFromSpec, inserting cb_type 'INT_R'
CBAR    loadFromSpec, size: 3737
...
Placing design...
OUTL    placeIOBs, found: 'K6' for 'rst', looking for pin...
OUTL    placeIOBs, found pin: 'K6' coords (57,99)
OUTL    optimizeOutline, fpga_width: 296, fpga_height: 418, aspect_x: 29.600, aspect_y: 41.800,
```

Picture 3. Output of *nextpnr* for Wineguard *riscv_multicyc* project using *xc7a100tcsg324* part

Placement and routing results for Wineguard riscv_multicyc using scalepnr are presented on Picture 4 and Picture 5.

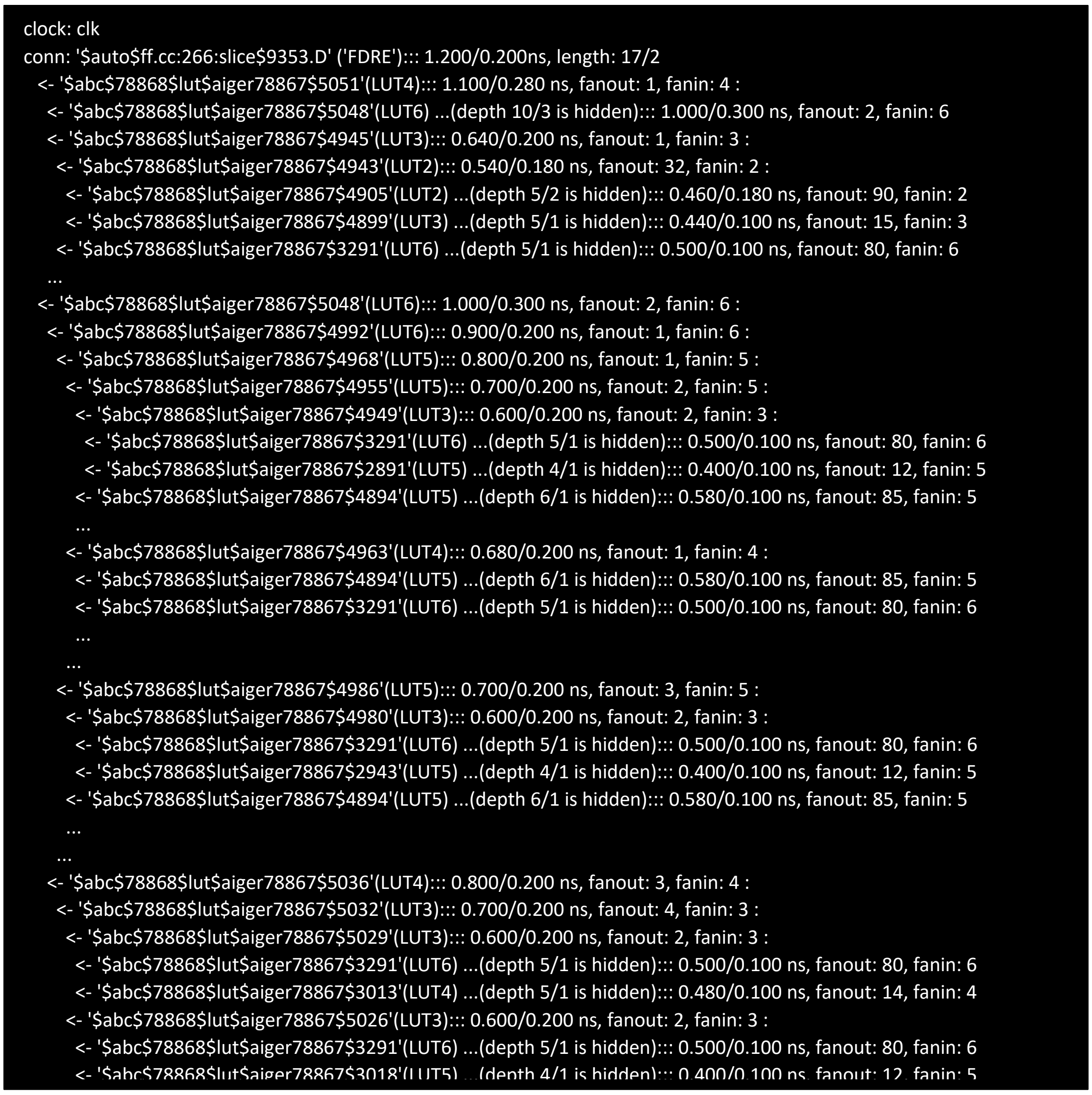


Picture 4. Placement result of Wineguard *riscv_multicyc* project using *xc7a100tcsg324* part



Picture 5. Routing result of Wineguard *riscv_multicyc* project using *xc7a100tcsg324* part

Timing results for Wineguard riscv_multicyc looks feasible but require further methodology development to automatically test and compare bunches of random designs, which is scheduled for the last stage of the project.



```
clock: clk
conn: '$auto$ff.cc:266:slice$9353.D' ('FDRE'):::  1.200/0.200ns, length: 17/2
  <- '$abc$78868$lut$aiger78867$5051'(LUT4)::: 1.100/0.280 ns, fanout: 1, fanin: 4 :
   <- '$abc$78868$lut$aiger78867$5048'(LUT6) ...(depth 10/3 is hidden)::: 1.000/0.300 ns, fanout: 2, fanin: 6
   <- '$abc$78868$lut$aiger78867$4945'(LUT3)::: 0.640/0.200 ns, fanout: 1, fanin: 3 :
    <- '$abc$78868$lut$aiger78867$4943'(LUT2)::: 0.540/0.180 ns, fanout: 32, fanin: 2 :
     <- '$abc$78868$lut$aiger78867$4905'(LUT2) ...(depth 5/2 is hidden)::: 0.460/0.180 ns, fanout: 90, fanin: 2
     <- '$abc$78868$lut$aiger78867$4899'(LUT3) ...(depth 5/1 is hidden)::: 0.440/0.100 ns, fanout: 15, fanin: 3
    <- '$abc$78868$lut$aiger78867$3291'(LUT6) ...(depth 5/1 is hidden)::: 0.500/0.100 ns, fanout: 80, fanin: 6
   ...
  <- '$abc$78868$lut$aiger78867$5048'(LUT6)::: 1.000/0.300 ns, fanout: 2, fanin: 6 :
   <- '$abc$78868$lut$aiger78867$4992'(LUT6)::: 0.900/0.200 ns, fanout: 1, fanin: 6 :
    <- '$abc$78868$lut$aiger78867$4968'(LUT5)::: 0.800/0.200 ns, fanout: 1, fanin: 5 :
     <- '$abc$78868$lut$aiger78867$4955'(LUT5)::: 0.700/0.200 ns, fanout: 2, fanin: 5 :
      <- '$abc$78868$lut$aiger78867$4949'(LUT3)::: 0.600/0.200 ns, fanout: 2, fanin: 3 :
       <- '$abc$78868$lut$aiger78867$3291'(LUT6) ...(depth 5/1 is hidden)::: 0.500/0.100 ns, fanout: 80, fanin: 6
       <- '$abc$78868$lut$aiger78867$2891'(LUT5) ...(depth 4/1 is hidden)::: 0.400/0.100 ns, fanout: 12, fanin: 5
      <- '$abc$78868$lut$aiger78867$4894'(LUT5) ...(depth 6/1 is hidden)::: 0.580/0.100 ns, fanout: 85, fanin: 5
      ...
     <- '$abc$78868$lut$aiger78867$4963'(LUT4)::: 0.680/0.200 ns, fanout: 1, fanin: 4 :
      <- '$abc$78868$lut$aiger78867$4894'(LUT5) ...(depth 6/1 is hidden)::: 0.580/0.100 ns, fanout: 85, fanin: 5
      <- '$abc$78868$lut$aiger78867$3291'(LUT6) ...(depth 5/1 is hidden)::: 0.500/0.100 ns, fanout: 80, fanin: 6
      ...
     ...
    <- '$abc$78868$lut$aiger78867$4986'(LUT5)::: 0.700/0.200 ns, fanout: 3, fanin: 5 :
     <- '$abc$78868$lut$aiger78867$4980'(LUT3)::: 0.600/0.200 ns, fanout: 2, fanin: 3 :
      <- '$abc$78868$lut$aiger78867$3291'(LUT6) ...(depth 5/1 is hidden)::: 0.500/0.100 ns, fanout: 80, fanin: 6
      <- '$abc$78868$lut$aiger78867$2943'(LUT5) ...(depth 4/1 is hidden)::: 0.400/0.100 ns, fanout: 12, fanin: 5
     <- '$abc$78868$lut$aiger78867$4894'(LUT5) ...(depth 6/1 is hidden)::: 0.580/0.100 ns, fanout: 85, fanin: 5
     ...
    ...
   <- '$abc$78868$lut$aiger78867$5036'(LUT4)::: 0.800/0.200 ns, fanout: 3, fanin: 4 :
    <- '$abc$78868$lut$aiger78867$5032'(LUT3)::: 0.700/0.200 ns, fanout: 4, fanin: 3 :
     <- '$abc$78868$lut$aiger78867$5029'(LUT3)::: 0.600/0.200 ns, fanout: 2, fanin: 3 :
      <- '$abc$78868$lut$aiger78867$3291'(LUT6) ...(depth 5/1 is hidden)::: 0.500/0.100 ns, fanout: 80, fanin: 6
      <- '$abc$78868$lut$aiger78867$3013'(LUT4) ...(depth 5/1 is hidden)::: 0.480/0.100 ns, fanout: 14, fanin: 4
     <- '$abc$78868$lut$aiger78867$5026'(LUT3)::: 0.600/0.200 ns, fanout: 2, fanin: 3 :
      <- '$abc$78868$lut$aiger78867$3291'(LUT6) ...(depth 5/1 is hidden)::: 0.500/0.100 ns, fanout: 80, fanin: 6
      <- '$abc$78868$lut$aiger78867$3018'(LUT5) ...(depth 4/1 is hidden)::: 0.400/0.100 ns, fanout: 12, fanin: 5
```

Picture 6. Timing result of Wineguard *riscv_multicyc* project using *xc7a100tcsg324* part

## UberDDR3 project

The tcl script:

```
load_design ddr3_top.json ddr3_top
create_clock -name clk -period 5.0 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN C9 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports rst]
…
open_design
place_design
route_design
```

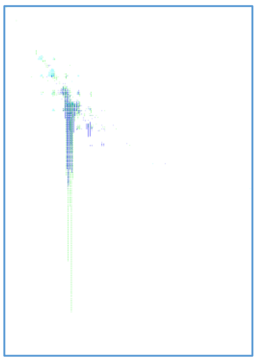
Picture 7. Tcl script for *scalepnr* Wineguard *riscv_multicyc* project using *xc7a100tcsg324* part

The results of a new *scalepnr* tool:

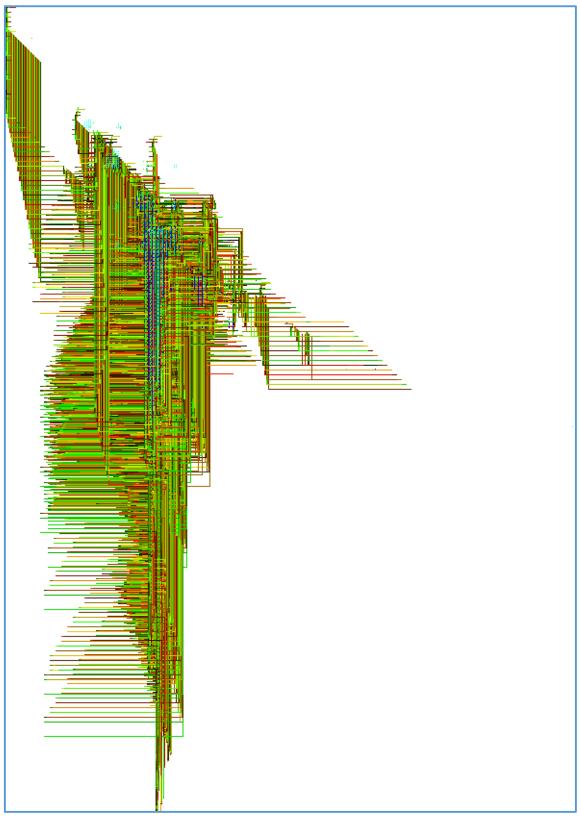
```
FPGA  loadCBFromSpec, spec_name: 'db/tile_type_INT_R.json'
FRMT    readCBTypes from 'db/tile_type_INT_R.json'
FPGA    loadCBFromSpec, inserting cb_type 'INT_R'
CBAR    loadFromSpec, size: 3737
…
Placing design...
OUTL    placeIOBs, found: 'K6' for 'rst', looking for pin...
OUTL    placeIOBs, found pin: 'K6' coords (57,99)
OUTL    optimizeOutline, fpga_width: 296, fpga_height: 418, aspect_x: 29.600, aspect_y: 41.800,
step_x: 0.034, step_y: 0.024, total_regs: 1439, total_comb: 67254, total_bunches: 1439, combs
Rtl usage:
---------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
-------------------------------------------------------
|  |BUFG: 8 CARRY4: 192 FDRE: 4388 FDSE: 56 IBUF: 454 IDELAYCTRL: 2 IDELAYE2: 36 INV: 310 IOBUF:
32 IOBUFDS: 4 ISERDESE2: 36 LUT1: 2 LUT2: 1168 LUT3: 3188 LUT4: 1184 LUT5: 2858 LUT6: 1208 M
UXF7: 16 MUXF8: 4 OBUF: 458 OBUFDS: 2 OSERDESE2: 88   |
```

Picture 8. Output of *scalepnr* for Wineguard *riscv_multicyc* project using *xc7a100tcsg324* part

Placement and routing results for Wineguard riscv_multicyc using scalepnr are presented on Picture 9 and Picture 10.



Picture 9. Placement result of UberDDR3 *ddr3_top* project using *xc7a100tcsg324* part



Picture 10. Routing result of UberDDR3 *ddr3_top* project using *xc7a100tcsg324* part

*Note. Some unimplemented peripheral blocks were removed from design*

## TISG project

Failed to compile with Yosys:

```
ERROR: Module `IDDR2' referenced in module `MIPI_Reciever' in cell `lane1' does not have a port named 'ov_fl'.
```

Picture 11. Output for Yosys for TISG project using *xc7a100tcsg324* part

**Table 4** contains commands used during benchmarking.

| *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 | |

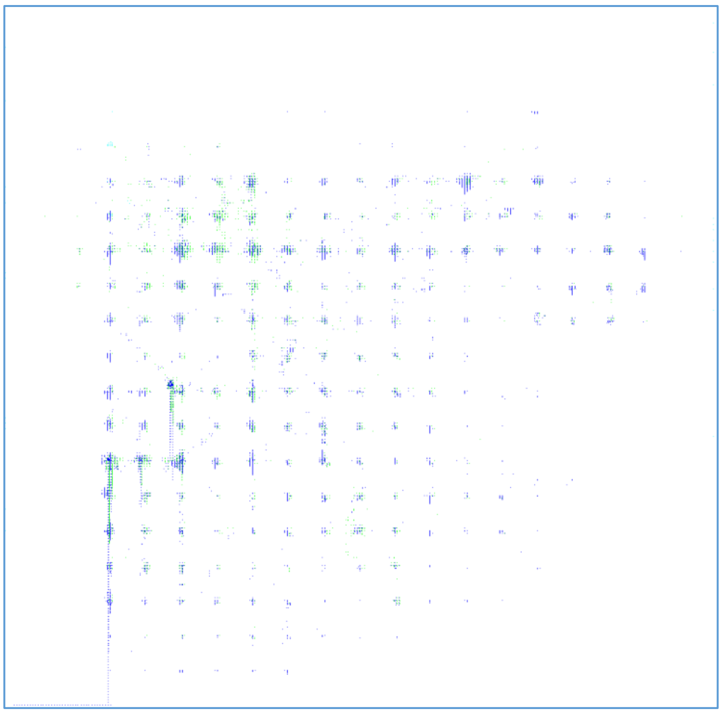
| | | |
|---|---|---|
| load_design | Loads RTL from Yosys JSON | |
| open_design | Performs RTL processing and estimation | |
| place_design | Does design placing | |
| print_design | Prints design or it's part in ASCII hierarchical format | |
| set_property | Sets PINs assignments and constraints | |
| route_design | Does design routing | |

***Table 4.** TCL commands supported in scalepnr*

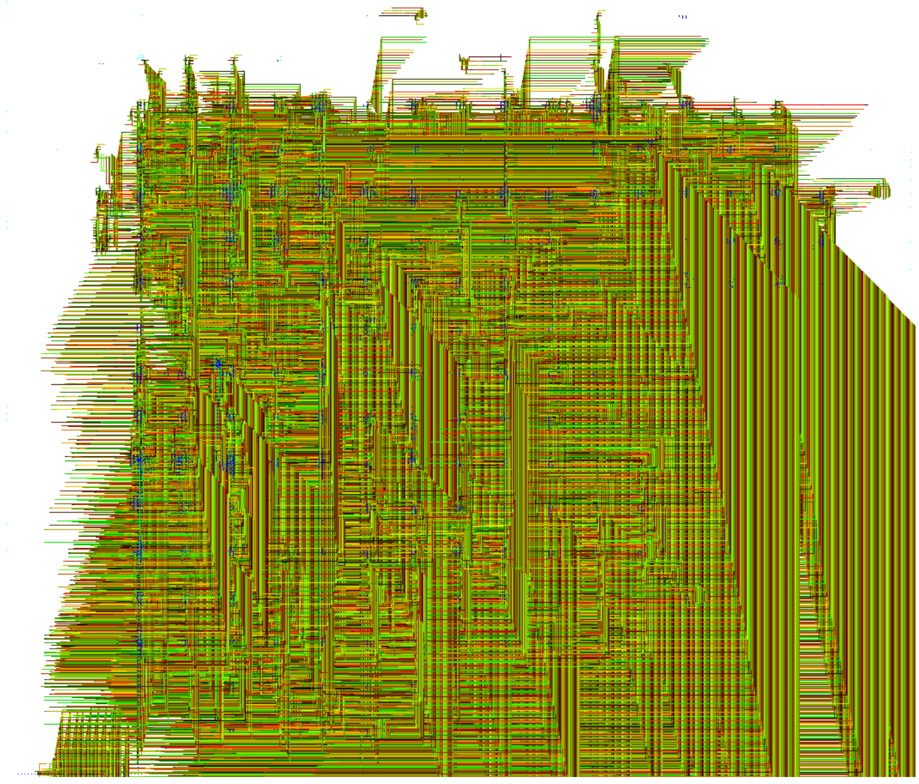
Example TCL script for processing of *TestMesh.json* RTL is presented on **Picture 2.**

## Larger designs

All previous designs were placed in a small part of a second. For next comparison stage designs required with controllable size and other parameters which is possible using *pnt_test* tool. Fine test methodology is scheduled to next stage of the project, while some generated designs layouts shared in the current chapter.



Picture 12. Output of placing by *scalepnr* for generated projects using *xc7a100tcsg324* part



Picture 13. Output of routing by *scalepnr* for generated projects using *xc7a100tcsg324* part

## 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* database specifications: no vendor specific code allowed in scalepnr now, except RTL blocks
2. New routing algorithm based on abstract *Crossbox* nodes representation using 256-bit maps for accelerated search
3. Routing images now allow to estimate quality of routing, which is the very starting point in *scalepnr* visualization abilities
4. Projects were benchmarked in PnR for *scalepnt* and *nextpnr* tool. Results are not comparable yet since there is not system to estimate quality of the result:
    1. *nextpnr* results can't be visualized and output is stingy
    2. in situations where tools spend different time in PnR there need to be quality of result estimation in many dimensions

*c.* this should be done using *pnr_test* tool to automatically iterate designs with different properties and requires inventing of an quality estimation formula

## Future work plan

1. Future work requires lots of testing: testing using random design generation using *pnt_test* in cycle utility with results estimation (need to invent quality estimation and comparison methodology)
2. Integration of other toolset programs is required, especially bitstream generation tools
3. Series of hardware testing using bitstreams and hardware examples
4. Further development plan is required for incremental forward progress in adaptation of the *scalepnr* for real projects and bring next level of quality to a project
5. Implementing more *tcl* functions to make whole process flexible