Project number:
2023-08-244

Task Report ----------

2024-10-06, Mike Reznikov

Task

1. Analysis of the existing toolchain and problem statement formalization
The goal of this task is to get on board with the nextpnr project, understand its interaction with the rest of the toolchain, be able to run existing benchmarks and to understand the limitations and bottlenecks of the existing solution. Gain the detailed knowledge of the problem and build a formalization of the problem into a problem statement that can be coined as a discrete optimization problem by specifying its objective function, problem constraints and parameters involved.

Detailed steps

a. Analysis of the existing toolchain and available architecture APIs

b. Replicate the existing benchmarks, create tooling for benchmarking

c. Understanding the limitations of the existing place-and-route algorithms

d. Publish code & report documenting the findings

e. Formalization of the problem statement

Table of contents

General considerations and notes

Source code links and details

The *nextpnr-xilinx* from *openXC7* project sources was taken from the following repository:

https://github.com/gatecat/nextpnr-xilinx/tree/xilinx-upstream/xilinx

There were 2 versions used in the tests:

1. Branch '*xilinx-upstream*' of an original repository
2. There was also version '*stable-backports*' taken from the address https://github.com/openXC7/toolchain-installer and installed by the script provided.

Looking ahead, there were no difference found between these versions in terms of scaling performance and visible amount of problems. Therefore, unless otherwise stated, the first - most fresh version - was used.

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

1. Only Xilinx Artix FPGA support was testes in this work. The sizes of used parts were from 10K to 200K of LEs. There is a support for Kintex. There is no support for Ultrascale products.
2. All experiments were run under Virtual Machine with OS Linux Ubuntu 22.04 Server using 16 CPU cores and 32GB RAM. It was not a limitation to testing process since all found problems don't require large memory size or CPU time to be repeated.
3. None of compiled designs were tested in hardware. This work does not check the quality of synthesis as well as place and routing process.
4. All considered toolchains use *Yosys* open source synthesis tool, which is not part of *nextpnt*. With this, *nextpnr-xilinx* is strongly depends on *synth_xilinx* function and objectively *Yosys* plays a significant role in Place and Route process.
5. GTP transceivers and IOSERDES were not tested 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

Auxiliary projects and tools used during testing and evaluation:

https://github.com/f4pga/prjxray

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

https://github.com/chipsalliance/f4pga

https://github.com/chipsalliance/fasm

https://github.com/chipsalliance/chisel

Own sources developed during this work:

https://github.com/mirekez/pnr_tests

https://github.com/mirekez/scalepnr

Analysis of the existing toolchain and available architecture APIs

First steps

First runs of *nextpnr-xilinx* toolchain, as it was recommended, were made for *openXC7-TetriSaraj* project using tools from https://github.com/openXC7/toolchain-installer. The tools are installed by *Snap* system and don't require OS libraries and headers and don't depends on any versions of software installed in Linux. This allows easy start even for developers not very common with Linux OS.

The most of first runs lead to successful project compilation. Some of compilations required slight *Makefile* changing. The results of first tests listed in **Table 1**.

Project name	*Result*	*Notes*
openXC7-TetriSaraj	Successful project compilation	
blinky-digilent-arty	Successful project compilation	
litex-ddr-arty-s7	Successful project compilation	
openeye-CamSI-main	**Failed.**	Requires SystemVerilog support while Yosys doesn't support it. In many situations where Yosys can compile SystemVerlog code the result does not work for nextpnr.
openXC7-TetriSaraj with (* ram_style = "logic" *)	**Failed.**	Routing stalls. Debugging shows that inaccurate placing of many high-fanout elements leads to infinite routing attempts.
openXC7-TetriSaraj with (* ram_style = "distributed" *)	**Failed.**	*Nextpnr-xilinx* doesn't support distributed memory.

Table 1. *First testing results*

Notes. Most of runs required a long "chipdb" writing process first time for each FPGA part number.

First experience shows that there is a range of conditions in which this toolchain can be used: design should use not too much space of an FPGA and should not require unimplemented

features of *nextpnr-xilinx*. In particular, designs, which are using more than 50K of LUTs and FFs can lead to infinite routing. The incomplete list of features, which are not fully implemented in *nextpnr-xilinx*, is presented below:

1. Block RAM (of 18 and 36 size) – partially supported, problems may occur
2. DSP48 – partially supported, problems may occur
3. Distributed memory - not implemented
4. Clock constraints, multi-clock constraints

The following problems were observed during first testing:

1. All considered toolchains use external tool to write bitstreams – *fasm2frames*, which always complain about not optimal usage of it because of "Antlr4 parser is missing".
2. There were many problems found like when *nextpnr-xilinx* gives non useful error messages in return to wrong settings or wrong input. In such situations messages don't help to find a source of the problem.
3. There were problems found in reporting of statistics information after placing and routing.
4. *Nextpnr-xilinx* uses third party *f4pga* scripts to build chip database and to assemble a *bitstream*.

First look code analysis

Nextpnr-xilinx sources are located mostly in two folders: *generic* and *xilinx*. *Generic* folder contains some common data structures / classes, methods and functions which are not xilinx-dependent, like *Cells* and *Pips*, *Bells* and *Wires* related. Despite the fact that *nextpnr-xilinx* was developed for multi-vendor support, currently the xilinx branch is 100% xilinx-dependent. Moreover, in *xilinx* folder new objects are defined for objects like *Cells*, *Pips*, *Bells* and wires. Therefore, it looks like *xilinx*-related code was put standalone to this project does not form the single shared source space in the project. The code in *xilinx* folder implements functions for reading Yosys json output files with RTL and implements several functions for placing and routing. It contains two versions of router functions, old and new, and it's not obvious what's the difference. The code does not contain comments.

As result, the following list of *nextpnr-xilinx* source code peculiarities were concluded:

1. The code is monolithic and does not contain separated layers like objects model or algorithms/methods.
2. Processing stages like Placing, Routing, DRC are not separated enough, there is no possibility to run stages separately, check intermediate result.
3. The code is not open for large-team development or bug-fixing, there is no attachment points and interfaces between layers, stages, no way to make plugins and attachments.
4. Since xilinx placing and routing data is stored in textual mode in f4pga project, *nextpnt* uses many objects as strings, operates strings during frequent operations.

5. *Nextpnr-xilinx* uses much optimized search key-value storages, which are not easy for understanding in all details. Probably, the optimization focus was made to secondary, bottom-level algorithms, while top-level logic looks insufficient.
6. *Nextpnr-xilinx* uses very brief one-clock timing estimation during placing and routing.

Conclusions

As a conclusion of this chapter, the following conclusions were made: that *nextpnr-xilinx* project was not developed to be easy for maintaining or enhancing. There is a lack of architecture backbone, interfaces, attachment points and comments. *Nextpnr-xilinx* is not integrated into *nextpnr* project shared source space.

Replicate the existing benchmarks, create tooling for benchmarking

Current benchmarking and testing state

There were no testing system or tests observed in source code of *nextpnr-xilinx*, therefore an approach to building a benchmarking system was proposed.

Development of benchmarking tools

The benchmarking system *pnr_tests* was developed based on random design generation approach to allow whole task input parameters space evaluation. The Chisel/Scala HDL generation language was used to write a generator for several possible processing topologies: single pipeline, mesh and star. The generator is controlled by the four parameters: **chain node complexity**, **chain length**, **chain width**, **chain depth** and can generate any number of random designs and automatically run *nextpnr-xilinx* toolchain to compile each of them. With this, *pnr_tests* project is able to make random IO assignments for each run.

Currently *pnr_tests* system contains nodes of the following types: **decode**, **math**, **memory**, **mux** and **queue**. There a few main generators for each of types were developed, like simple decoder, div/mul, logic/block memory, mux/demux and FIFO, with ability to generate each node randomly (random size, random logic), based on required complexity input parameters. By using this nodes in a complex computational topology like mesh, or star, *pnr_tests* system implements powerful tool to provide benchmark and stress testing for Place and Route tool, as well as randomly cause congestions, high-fanout, long wires and pipelines, and other random (not known before) patterns which can lead to failure. It is easy to develop own node types and adding specific RTL functions and new topologies to a code, since Chisel/Scala language makes automatic inferring of needed wires for connections, cross-boxes and different types of casting.

Problems, found by random testing

During thousands of random designs generation there were found several problems in support of different functions. The found problems are listed in **Table 2**.

Description	*Link to the issue*
Monitoring pins used as IO lead to nextpnr-*xilinx* crash with message `terminate called after throwing an instance of 'std::out_of_range' what()`	https://github.com/openXC7/nextpnr-xilinx/issues/36
Non clock pins used as clocks lead to non-informative error messages	https://github.com/openXC7/nextpnr-xilinx/issues/38
Failed to route DSP48	https://github.com/openXC7/nextpnr-xilinx/issues/35 https://github.com/openXC7/nextpnr-xilinx/issues/37
Failed to route RAMB36	https://github.com/openXC7/nextpnr-xilinx/issues/39
Unexplainable failed to route	https://github.com/openXC7/nextpnr-xilinx/issues/40 https://github.com/openXC7/nextpnr-xilinx/issues/41

Table 2. *Problems found by random testing*

With this, found problems are the secondary product of benchmarking, while the main question was possibility of *nextpnt-xilinx* to place and route large designs. The following testing excluded all features and functions, which lead to compilation problems. It is possible to continue testing of *nextpnr-xilinx* stability, especially after fixing the bugs found.

Benchmarking results

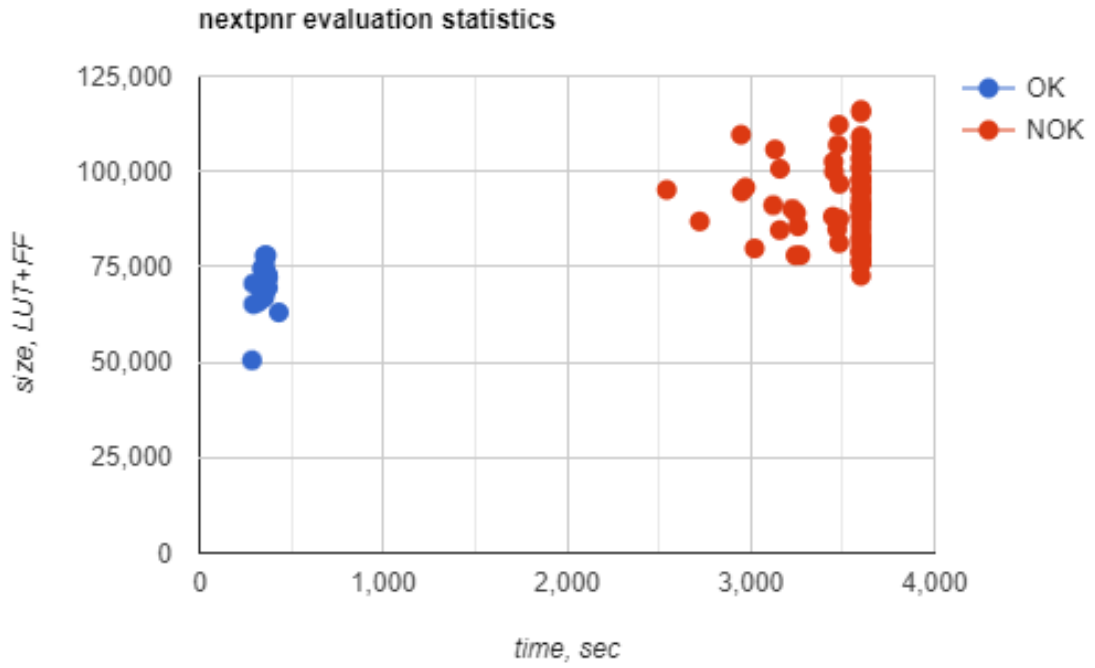
The random benchmarking results with are presented in **Table 3**.

Number of logic elements	Average Result	Comments
<50K	100% successful	
<100K	50% successful	1H kill timeout
<150K	5% successful	1H kill timeout
<200K	Not evaluated	

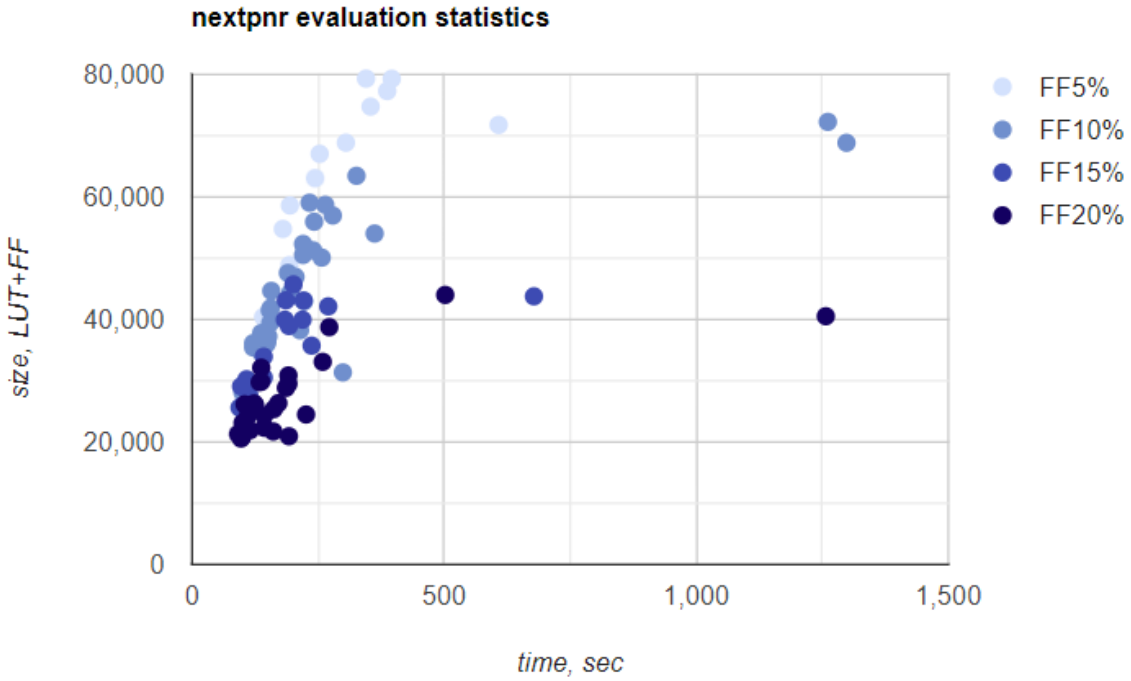
Table 2. *Random performance testing results after problematic inputs exclusion*

Picture 1 shows that *nextpnr-xilinx* processing time increases to orders of magnitude when the cumulative number of LUTs+FFs is achieving level about 75K. This lets us make no positive expectations about process finishing in hours or days and allows us to consider all right zone points as hanged processes and stop them after 1 hour of work.

These tests were evaluated for designs with DSPs, but containing a low relative number of FFs, but later it turned out that the number of FFs significantly influences the compilation time and success possibility. BRAMs were not used during testing.



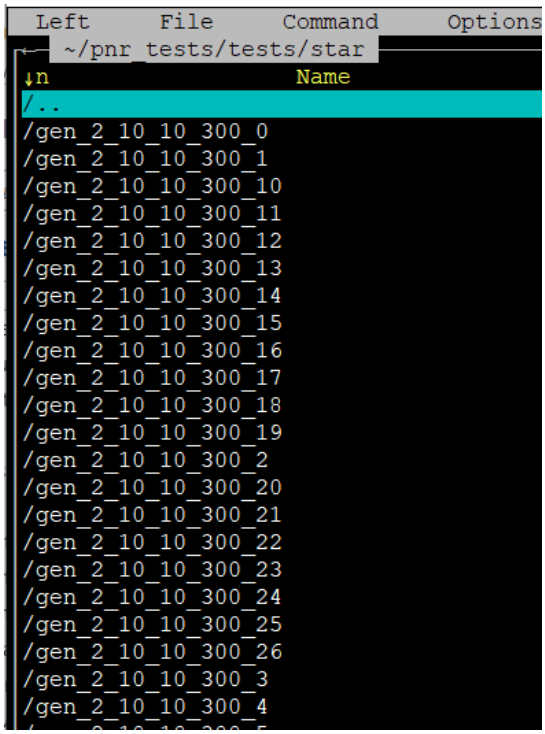
Picture 1. Statistics of *nextpnr-xilinx* evaluation with **FFs number <5% of LUTs**



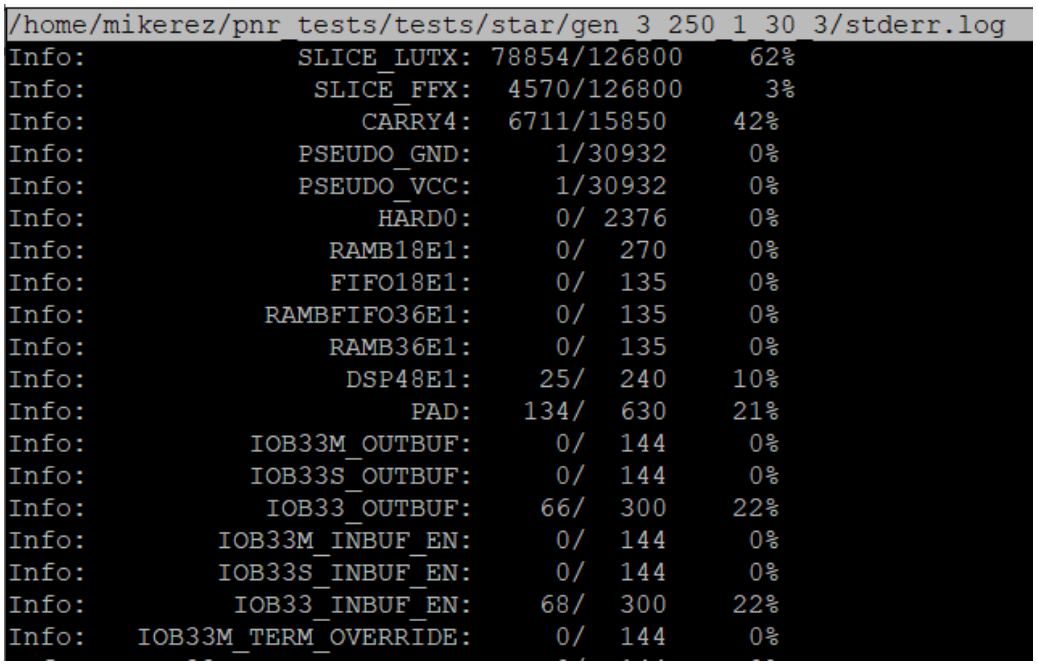
Picture 2. Statistics of *nextpnr-xilinx* evaluation for different **FFs % number of LUTs**

Picture 2 shows that increasing percentage of FFs in the design makes *nextpnr-xilinx* work more complex and long. In addition, with FFs relative number increasing, *Yosys* synthesis application from openXC7 toolset starts consuming a lot of RAM and hanging for half of an hour (probably, some features can be disabled to avoid this). Then it was stopped by OOM event at level about 20GB RAM usage.

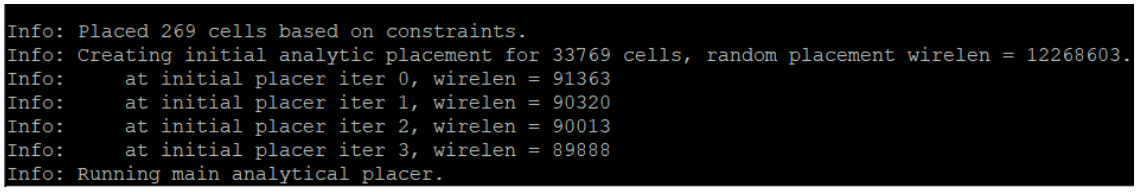
Example screenshots of the random testing system output are shown on **Pictures 3-6**.



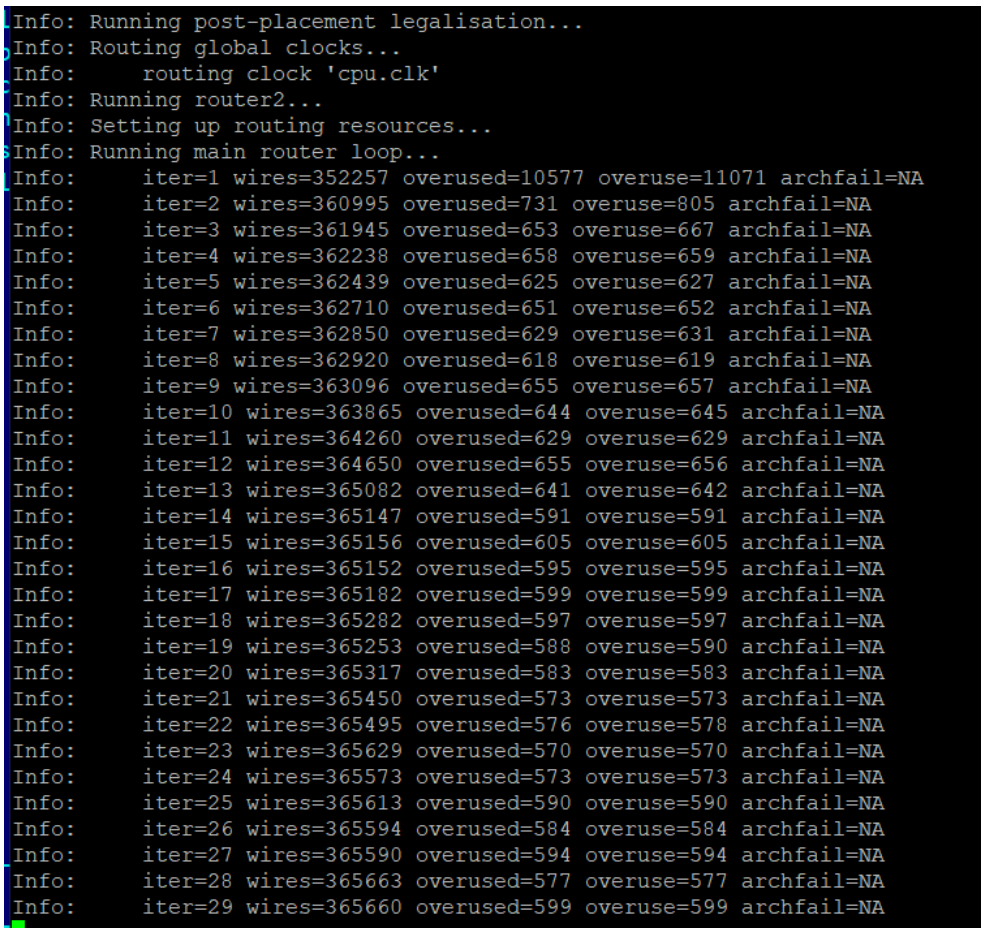
Picture 3. Testing executions list



Picture 4. One design output statistics from *nextpnt*



Picture 5. Example of a hanged placing process



Picture 6. Example of a hanged routing process

Conclusions

In general, *nextpnr-xilinx* shows good performance in all task size ranges it claims to support (small designs, less than 50K elements). The numerous problems accompanying it usage can be considered as errors and non-implemented features. The significant increase of processing time was observed for designs when FFs number increases above 3-5% of LUTs count. With higher FFs percent in designs, the *Yosys* synthesis tool becomes a stopper because of unexpectedly high consumption of RAM (more than 20GB). *OpenXC7* toolset lacks of several mandatory functions and requires ability for scale-up to 500K and >1M elements size designs.

Understanding the limitations of the existing place-and-route algorithms

Scaling limitations analysis

Considering the fact, that almost 100% of randomly generated designs above 100K elements were successfully placed and routed by *nextpnr*, the most frequent message in the end of the flow was: “ERROR: Max frequency for clock 'modules_10.clock': 8.19 MHz (FAIL at 12.00 MHz)”. It emphasizes the fact, that *scalepnr* does not support clock constraining and

provides just best possible frequency for clocks it manages to recover from the design. This behavior is completely unsuitable for large-scale designs, which often require a trade-off between different clocks performance, allowing some clocks relaxation while another clocks must fit the requirements. With this, cross-clocking timing estimation is mandatory for many of peripheral interfaces, requiring multi-clock FIFO usage or control registers being accessed from different clock domains. Therefore, the whole problem of building strong multi-clock timing-driven place and route model is considered mandatory and more important goal than possible code re-usage from *nextpnr-xilinx* project. Multi-clock timing model should be accompanied with appropriate module testing system, which will allow claiming its capabilities and performance.

Missing functionality and modern challenges

The *nextpnr-xilinx* project still experiences lack of support of several mandatory features of modern place and route tools. The list of unimplemented features includes:

1. Clock constraints, multi-cycle clock constraints
2. Multiple clocks timing aware place and route, cross-clock domains
3. Partitioning workflow (or incremental compilation) and pblock constraints
4. Elements position constraining, routing constraining
5. Intermediate results saving and exploration
6. Separated DRC checking system
7. Own part numbers description database and *bitstream* generation functions
8. Builtin scripting system

In this work, all points of the list above are suggested to be milestones of the future development in *scalepnr* project. From the architecture point of view, *scalepnr* should be capable of the following availabilities:

1. To be open for third party collaboration and development
2. To have interfaces and attachments points for external algorithms and plugins
3. To contain well-specified core architecture and models

With both lists above, modern place and route tool should support up to millions of elements in the design and be stable enough to work for days.

Problem state and possible development ways

Considering all said before, it has suggested doing the work in the following way

1. Development of a basic FPGA modelling system, supporting TCL scripting language to be a foundation of a *scalepnr* project.
2. Translation of existing place and route algorithms of *nextpnr-xilinx* to a new project for evaluation.

3. Development of a new place and route functions with multi-clock constraining, which can be run and configured using TCL script language.
4. Implementation of algorithms which scale up to 500K and 1M elements PnR.
5. Adding support for unimplemented mandatory features.

Conclusions

Nextpnr-xilins is still requiring a lot of work to be done to implement modern features of place and route process, including scaling, multi-clock, partitioning, partial and incremental compilation, flow flexibility and configurability, pblocks, user-friendly messages and strong statistics output, scripting and IP-packaging, as well as full Xilinx FPGA elements space support. Therefore, the main goal at the current step should be creation of a strong source code base and models, which will allow further collaboration of third party developers and companies and make possible to combine results of their work in one place.

Publish code & report documenting the findings

The *pnr_tests* benchmarking code is published at https://github.com/mirekez/pnr_tests

The *scalepnr* proposed base code is published at https://github.com/mirekez/scalepnr

Formalization of the problem statement

Future work plan

The following work items are proposed for future work:

1. TCL-script based basic system, supporting laconic own FPGA part numbers database and clear FPGA tiles and crossbars model. Can be done as first approximation, demonstrating ability to correctly enumerate all elements of several FPGAs. This model can still support other vendors of FPGAs in theory. (almost done)
2. Very basic place and routing approach for simple designs, allowing to provide textual descriptions of locations and routes to be assembled to *bitstreams* by third party tools. It is preferable also to have a fast scripting way to get DCP files with compiled designs.
3. The development of a multi-clock timing-driven general model for placing and routing. The model is developing together with testing system to prove performance and ability to solve timing aware multi-criteria problems of placement in general. The model should be open to enhancement and should have attachment points for plugins
4. Improvement of timing-aware route and placing algorithms. Result should be no worse than existing *nextpnr-xilinx* algorithm but support more than two clock constraints.

5. The multi-stage approach to place and route should be developed, allowing serial calling of different algorithms for different parts of RTL using TCL scripting. The basic guidance algorithm is introduced as an example of top level placing planning. The system is fully open for attaching own algorithms of placing and routing and using in combination with basic algorithms.
6. Large-scale random designs testing stage which suppose optimization and enhancement of basic plan, place and route algorithms to support up to 1M elements design.
7. The work on not yet supported part numbers and larger FPGA devices support.
8. The work on own system for writing *bitfiles* (take from f4pga) and DCP files (using libs from third party). *(this work is mandatory but can be left out of scope of the current project*)
9. The work on supporting of all types of elements not supported yet, including full support for block ram, distributed ram (like RAM256X1S), distributed shiftregs (like SRL128), local clock buffers, clock switches, SERDES, IODELAY, GTP. *(this work is mandatory but can be left out of scope of the current project)*
10. Timing estimation enhancements should be done as they present in f4pga project. *(this work is mandatory but can be left out of scope of the current project)*

Next task proposal

Steps 1-3 of the list above are suggested to be solved as **Task 2** of the general plan of works.

Since, the second task description is:

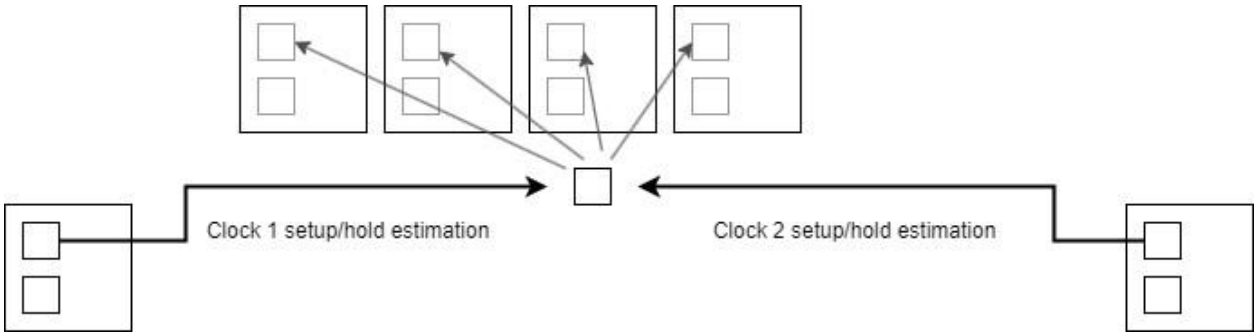
"2. First version of the place-and-route algorithm
We will design a simplified version of the place-and-route algorithm where the emphasis will be given to the full-feature functionality in terms of the formalized problem statement from task 1, but not necessarily with the focus on the quality of the solutions, scalability, and the computational speed.",
it is proposed to implement **multi-clock timing driven placing and routing** algorithm on this stage. The **multi-clock timing driven placing and routing** should be solved as multi-criteria optimization problem, allowing to balance the timing budget between different clock domain users of a shared object. On this step, the algorithm should demonstrate possibility to place and route multi-clock designs with up to 50K elements.

The next, third stage task should include optimizations to make algorithm work at least 10 times faster and include steps 4-6 of the list in the previous chapter.

Development architecture proposal

During a search phase of place and route process all clock constraints should be considered and a balancing approach should be implemented to satisfy all timing constraints as well as available positions constraints (**Picture 7**). During **Task 2** solving, lazy algorithms should be used, which give good results for designs, which require less than 20% of the device. During

Task 3 solving, multi-clock constrained packing should be implemented to allow using of up to 70% of LUTs in the device of size not less than 500K. It should use all-swapping approach, marking areas of finished placing and giving credits to each region marking how much tension it has. Top-level placing of a large RTL modules is crucial on the first stage of the PnR process and allows to make placing on lower levels more efficiently. Therefore, precise methods for top-level modules multi-clock aware placing estimation should be used. The 64GB memory limit is suggested for 500K elements devices. Algorithms should require not more than 1hour to give first effective result, which can be enhanced later by request. During placing actions should be taken to provide possibility for an element to be routed out of Crossbox.

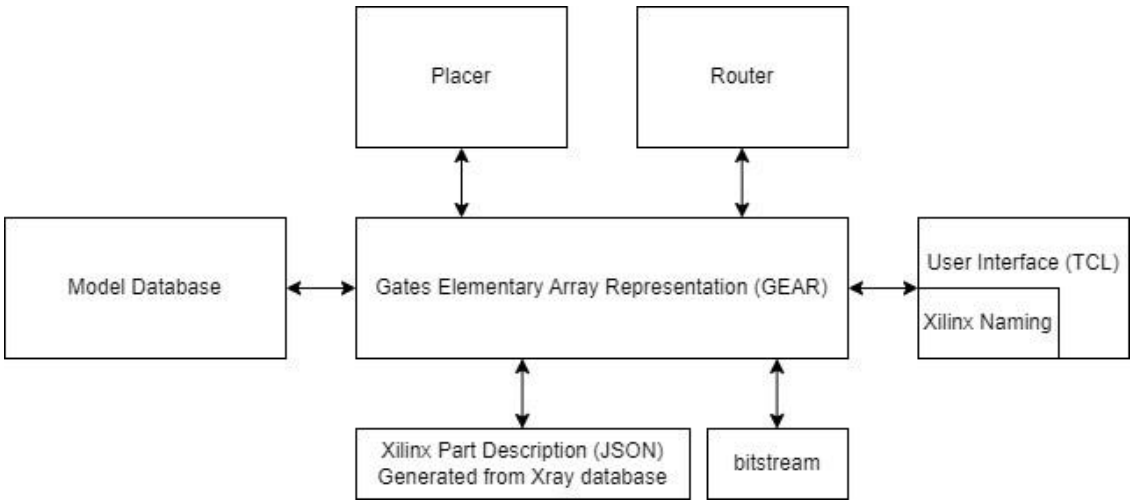


Picture 7. Looking for an appropriate place and route considering two clock constrains

The proposed class architecture is based on a universal **TileType** object, representing FPGA Tile and Crossbox elements together. TileType describes all possible types of an existent combination of Sites with Bells and corresponding Crossbox. The TileType instances are created for all tiles of an FPGA and collecting information about placed Bels and provides methods for checking availability for placing elements (**Picture 8**). All placing algorithms should use interface methods to access this elements and its state. During placement, the basic configuration should be done to Crossboxes, corresponding to Tiles to reserve a path for all inputs and outputs of Tile's Bels to be available to go outside of CB. Later, during routing process, these reserved paths can be swapped. During routing process, Crossboxes should be configured by processing all wires one by one and by building Nets. Multiple optimization steps can be applied then to the array of Tiles.

While RTL model is 100% taken from *Yosys synth_xilinx* descriptions, it should not be textual object name used. All Pins and Pips should get corresponding indices and be accessed by them. At any point of workflow, the model can be converted and printed using Xilinx device's element names. There should be functions and methods developed to provide distance and timing estimation for timing-driven place and route algorithm.

Currently the draft project is developed at https://github.com/mirekez/scalepnr, which supports shortened *json* descriptions of devices and allow working with device objects using integrated *TCL* language (**Picture 9**).



Picture 8. General model of modules and interfaces (architecture)

```
GEAR  populating rect 137:(53-77)...
GEAR  found spec 'CLBLM_R_X3Y0' for type 'CLBLM'
GEAR  populating rect 13:(183-207)...
GEAR  populating rect 13:(157-181)...
GEAR  populating rect 13:(131-155)...
GEAR  populating rect 13:(105-129)...
GEAR  populating rect 13:(79-103)...
GEAR  populating rect 13:(53-77)...
GEAR  populating rect 13:(27-51)...
GEAR  populating rect 13:(1-25)...
GEAR  populating rect 137:(53-77)...
% get_tiles CLBLL_X10Y4*
CLBLL_X10Y4 CLBLL_X10Y40 CLBLL_X10Y41 CLBLL_X10Y42 CLBLL_X10Y43 CLBLL_X10Y44 CLBLL_X10Y45 CLBLL_X10Y46 CLBL
%
```

Picture 9. Current state of *scalepnr* project draft allows to work with objects using TCL

Conclusions

During evaluation of current state of *nextpnt-xilinx* project there were found a number of problems in user-interface, feature support and scaling-up availability. Despite this project is being enhanced and developed, there is a fundamental lack of functionality of multi-clock timing constraint-driven placing and routing. The problem of multi-clock timing driven placing and routing is a multi-criteria problem of non-linear optimization, which is known to be NP-hard. If implemented, it adds an independent layer of optimization algorithms to the project. Therefore, the success of this scaling-up initiative for open source place and route project is strongly depends on the success of multi-clock placing and routing optimization algorithms development. From this point, a conclusion was made that the **First version of the place-and-route algorithm** must include basic implementation of this mandatory feature (currently planes as **Task 2**). After demonstration of successful approach to multi-clock driven placing and routing, there should be next steps applied for scale-up place and routing open source tool development (currently planned as **Task 3**).

At the same time, the following requirements proposed for future steps:

1. It is mandatory to create a project with open architecture and documented core model and interfaces to allow third parties to participate and collaborate more easily

2. It is mandatory to form a script-based media, where it will be possible to test and evaluate different algorithms doing partial work for partial RTL and being able to estimate partial results quality
3. It is important to make focus on quality of source code and models descriptions during implementation of basic place and routing algorithms, making strongly optimized and complicated approaches secondary (In other words, there is a confidence that simple fixes may provide much more progress in scaling up of current basic algorithms than complicated and dense-code approaches)