Symbiotic EDA

# The nextpnr FOSS FPGA place-and-route tool

Clifford Wolf
Symbiotic EDA

# FOSS FPGA PnR

- **VPR (Versatile Place-and-Route)**
  - Over 20 years old (1997). Timing driven. Portable.
  - Focuses more on architecture exploration via architecture XML files, not PnR for existing real-world FPGAs.
  - Easy to get “close to actual hardware” for algorithm experimentation and architecture exploration. Much harder to get to working bitstream generation for actual hardware.

- **Arachne-PNR**
  - Three years old (2015). Not timing driven. Only supports iCE40.
  - First fully functional FOSS FPGA PnR tool with industrial user base.

- **nextpnr**
  - Brand new (2018). Timing driven. Portable. Team effort.
  - With GUI! (But still works well with Makefiles. :)
  - API for supporting new architectures. Currently: iCE40, ECP5
  - Architectures under construction or planned: “generic”, Torc, Xilinx 7-series
  - Further plans: floorplanning, multi-clock designs, complex constraints

# nextpnr-ice40

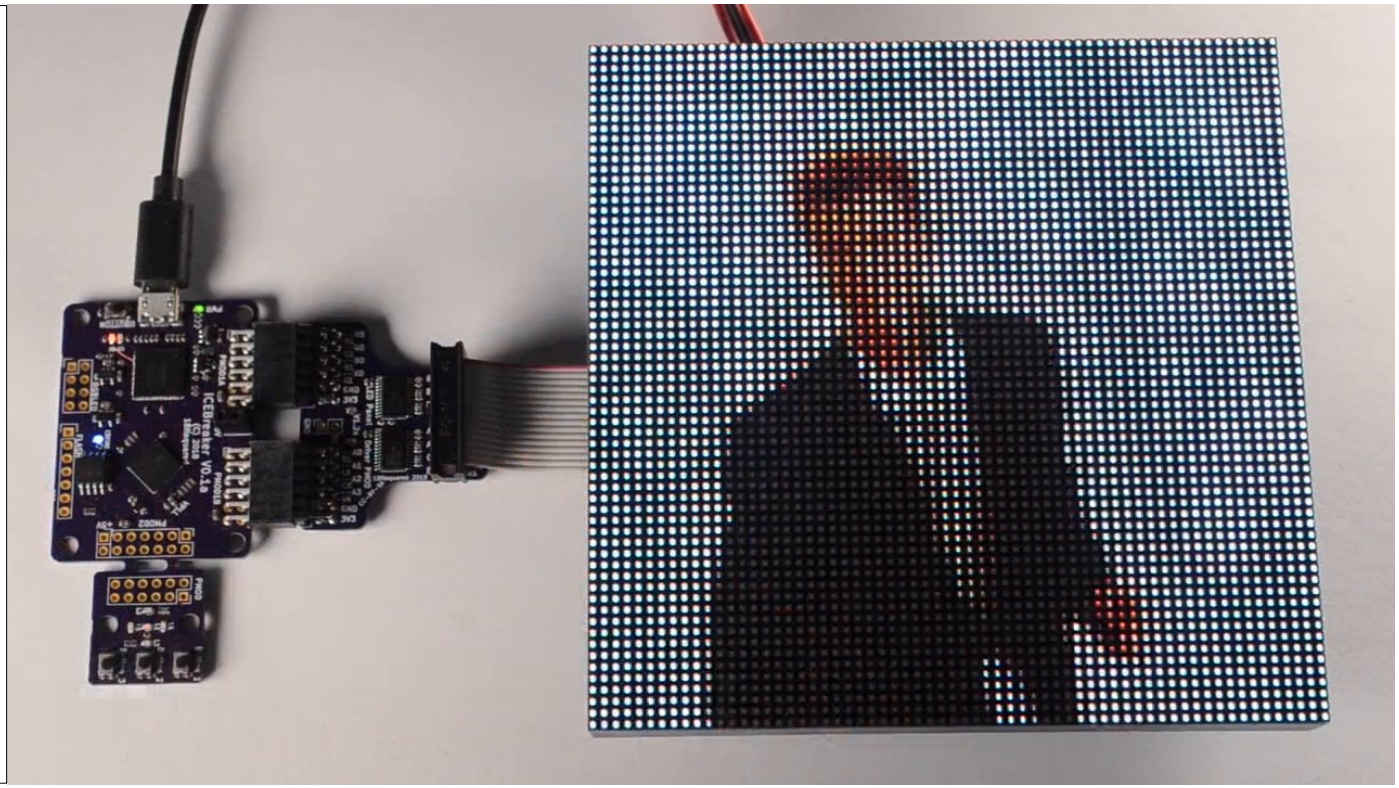
- **Place-and-Route for Lattice iCE40 FPGAs**
  - Support for LP384, LP1K, LP4K, LP8K, HX1K, HX4K, HX8K, UP3K, and UP5K devices
  - Full Verilog-to-Bitstream flow
    - Using Yosys (synth_ice40) for synthesis to JSON netlist file
    - Using Project Icestorm for bitgen and device programming

iCEBreaker FPGA

- iCE40UP5K
  - 5280 Logic Cells
  - 128 Kbit BRAM
  - 1 Mbit SPRAM
  - 8 DSPs
  - 1 PLL

- 3 PMODs
- 39 IO Pins
- FT2232H Interface
- 128 Mbit SPI flash

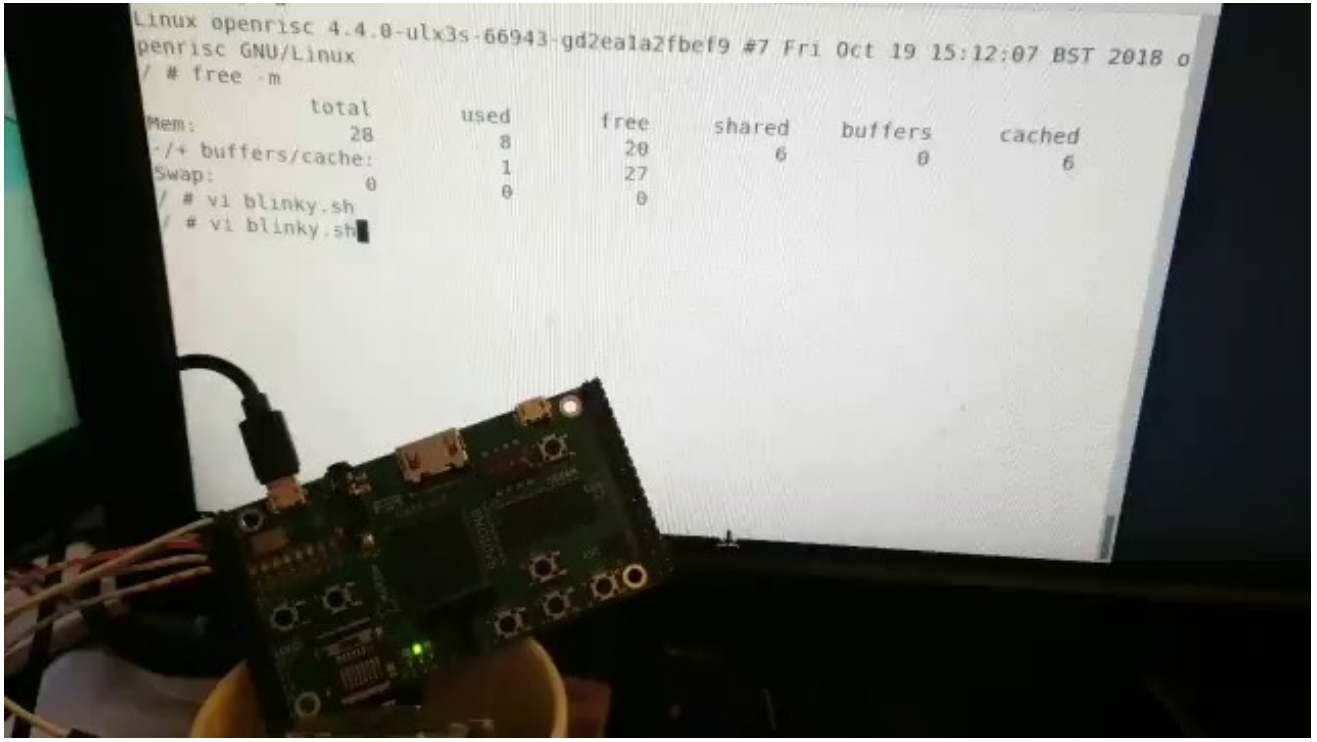
iCEBreaker is our preferred FPGA board for workshops.



https://github.com/icebreaker-fpga
https://www.crowdsupply.com/1bitsquared/icebreaker-fpga

# nextpnr-ecp5

- **Place-and-Route for Lattice ECP5 FPGAs**
  - Support for 5U-25F/45F/85F, 5UM-25F/45F/85F, and 5UM5G-25F/45F/85F devices
  - Full Verilog-to-Bitstream flow
    - Using Yosys (synth_ecp5) for synthesis to JSON netlist file
    - Using Project Trellis for bitgen and device programming

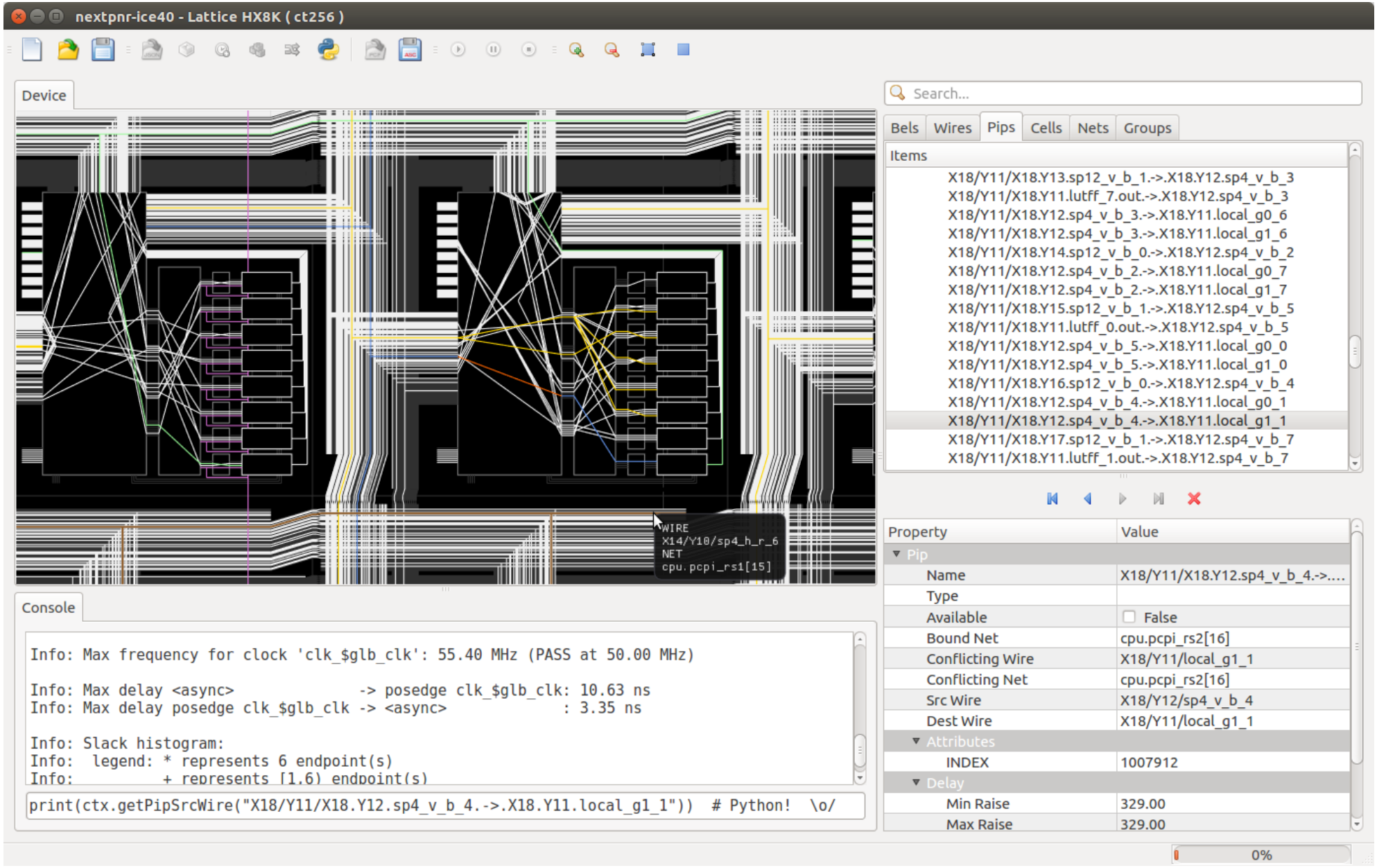



Linux running on an or1k SoC implemented on an ECP5 FPGA.
https://github.com/daveshah1/ulx3s/

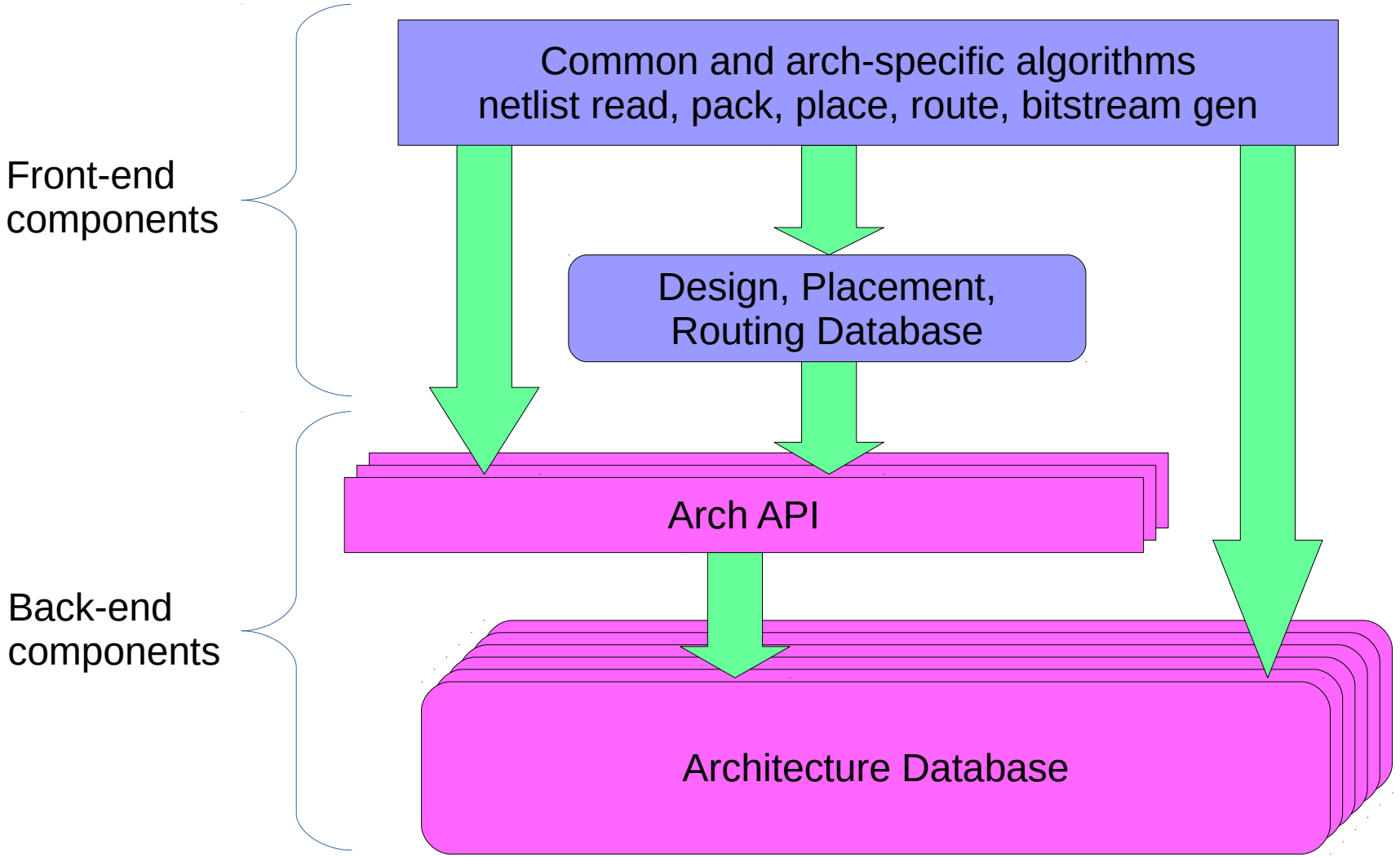
# nextpnr-torc
## (experimental support for various Xilinx devices)

- **Place-and-Route for Xilinx FPGAs via Torc**
  - Support for various Xilinx devices, including 7-series Zynq
  - Not a full Verilog-to-Bitstream flow yet
    - Using Yosys (synth_ecp5) for synthesis to JSON netlist file
    - Using ISE for bitgen and device programming
    - Alternatives to Torc+ISE: Project X-Ray and/or Xilinx RapidWright

- Mainly an experiment to see how our algorithms scale to large devices
- Ultimately we want to support Xilinx devices directly (w/o torc)
- If you are very curious (and brave) you might want to check out the “xc7” branch in our nextpnr github repo.


nextpnr-ice40 - Lattice HX8K ( ct256 )
Device
WIRE
X14/Y10/sp4_h_r_6
NET
cpu.pcpi_rs1[15]
Console
Info: Max frequency for clock 'clk_$glb_clk': 55.40 MHz (PASS at 50.00 MHz)
Info: Max delay <async> -> posedge clk_$glb_clk: 10.63 ns
Info: Max delay posedge clk_$glb_clk -> <async> : 3.35 ns
Info: Slack histogram:
Info: legend: * represents 6 endpoint(s)
print(ctx.getPipSrcWire("X18/Y11/X18.Y12.sp4_v_b_4.->.X18.Y11.local_g1_1")) # Python! \o/
Search...
Bels
Wires
Pips
Cells
Nets
Groups
Items
X18/Y11/X18.Y13.sp12_v_b_1.->.X18.Y12.sp4_v_b_3
X18/Y11/X18.Y11.lutff_7.out.->.X18.Y12.sp4_v_b_3
X18/Y11/X18.Y12.sp4_v_b_3.->.X18.Y11.local_g0_6
X18/Y11/X18.Y12.sp4_v_b_3.->.X18.Y11.local_g1_6
X18/Y11/X18.Y14.sp12_v_b_0.->.X18.Y12.sp4_v_b_2
X18/Y11/X18.Y12.sp4_v_b_2.->.X18.Y11.local_g0_7
X18/Y11/X18.Y12.sp4_v_b_2.->.X18.Y11.local_g1_7
X18/Y11/X18.Y15.sp12_v_b_1.->.X18.Y12.sp4_v_b_5
X18/Y11/X18.Y11.lutff_0.out.->.X18.Y12.sp4_v_b_5
X18/Y11/X18.Y12.sp4_v_b_5.->.X18.Y11.local_g0_0
X18/Y11/X18.Y12.sp4_v_b_5.->.X18.Y11.local_g1_0
X18/Y11/X18.Y16.sp12_v_b_0.->.X18.Y12.sp4_v_b_4
X18/Y11/X18.Y12.sp4_v_b_4.->.X18.Y11.local_g0_1
X18/Y11/X18.Y12.sp4_v_b_4.->.X18.Y11.local_g1_1
X18/Y11/X18.Y17.sp12_v_b_1.->.X18.Y12.sp4_v_b_7
X18/Y11/X18.Y11.lutff_1.out.->.X18.Y12.sp4_v_b_7
Property
Value
Pip
Name
X18/Y11/X18.Y12.sp4_v_b_4.->....
Type
Available
False
Bound Net
cpu.pcpi_rs2[16]
Conflicting Wire
X18/Y11/local_g1_1
Conflicting Net
cpu.pcpi_rs2[16]
Src Wire
X18/Y12/sp4_v_b_4
Dest Wire
X18/Y11/local_g1_1
Attributes
INDEX
1007912
Delay
Min Raise
329.00
Max Raise
329.00
0%

# nextpnr architecture



# Design Database

- **Cell**: an instantiation of a physical block inside the netlist. The packer may combine or otherwise modify cells; and the placer places them onto Bels.
- **Port**: an input or output of a Cell, can be connected to a single net.
- **Net**: a connection between cell ports inside the netlist. One net will be routed using one or more wires inside the chip. Nets are always one bit in size, multibit nets are always split.
- **Source**: The cell output port driving a given net
- **Sink**: A cell input port driven by a given net
- **Arc**: A source-sink-pair on a net

# Architecture Database

- **Bel**: Basic Element, the functional blocks of an FPGA such as logic cells, IO cells, blockrams, etc. Up to one cell may be placed at each Bel.
- **Pin**: an input or output of a Bel, permanently connected to a single wire.
- **Pip**: Programmable Interconnect Point, a configurable connection in one direction between two wires
- **Wire**: a fixed physical connection inside the FPGA between Pips and/or Bel pins.
- **Group**: a collection of bels, pips, wires, and/or other groups

# Adding a new Architecture

- Add a **new top-level directory** and add CMake config.
- Add **`archdefs.h`** with types for delay and addressing bels, wires, pips, etc. in the new architecture.
- Add **`arch.h`** with implementation of **`Arch`** class with ~100 methods for accessing the architecture database.
- Add **`main()`** and arch-specific algorithms such as **packer**.

Note: We are still finalizing the architecture API. Don’t add an avalanche of new architecture back-ends just yet. :)

# Some arch API details

~100 methods sounds like a lot, but most of them are very elementary. For example for bels:

```
BelId getBelByName(IdString name) const;
IdString getBelName(BelId bel) const;
Loc getBelLocation(BelId bel) const;
BelId getBelByLocation(Loc loc) const;
const_range<BelId> getBelsByTile(int x, int y) const;
bool getBelGlobalBuf(BelId bel) const;
uint32_t getBelChecksum(BelId bel) const;
void bindBel(BelId bel, CellInfo *cell, PlaceStrength strength);
void unbindBel(BelId bel);
bool checkBelAvail(BelId bel) const;
CellInfo *getBoundBelCell(BelId bel) const;
CellInfo *getConflictingBelCell(BelId bel) const;
const_range<BelId> getBels() const;
IdString getBelType(BelId bel) const;
const_range<std::pair<IdString, std::string>> getBelAttrs(BelId bel) const;
WireId getBelPinWire(BelId bel, IdString pin) const;
PortType getBelPinType(BelId bel, IdString pin) const;
const_range<IdString> getBelPins(BelId bel) const;
```

# Strategies for implementing nextpnr architecture databases

- Flat **database**
  - Each bel, pip, wire has its own explicit entry in the database
  - Because of this the database is rather big, but it's also very simple and fast
  - Very flexible with regard to "one off" oddities in the architecture
  - See iCE40 architecture
  - Also see "generic" architecture (under construction)

- **De-duplicated database**
  - First create a flat database, then find similar entries and consolidate them
  - Most advantages of flat database, but less memory usage
  - Requires compute-intensive de-duplication step
  - See ECP5 architecture

- **Tile-based database**
  - Create a description of each tile, and describe chips in terms of tile array
  - Much more complicated than the first two approaches, but yield much smaller databases, especially if one wants to support entire families of chips that use the same tiles but different tile arrangements.
  - See Xilinx 7-series architecture (to be done)

# Python API

- In addition to the C++ API nextpnr also provides a Python API
  - For implementing and/or prototyping PnR algorithms
  - For investigating the design in memory and/or device database
  - For creating simple device databases (via “generic” architecture)
  - For creating virtually arbitrary complex constraints

```
# Clock constraints
ctx.addClock("csi_rx_i.dphy_clk", 96)
ctx.addClock("video_clk", 24)
ctx.addClock("uart_i.sys_clk_i", 12)

# Placement constraints
ctx.createRectangularRegion("osc", 1, 1, 1, 4)
for cell, cellinfo in ctx.cells:
    if "ringosc" in cellinfo.attrs:
        print("Floorplanned cell %s" % cell)
        ctx.constrainCellToRegion(cell, "osc")
```

example nextpnr python constraints file

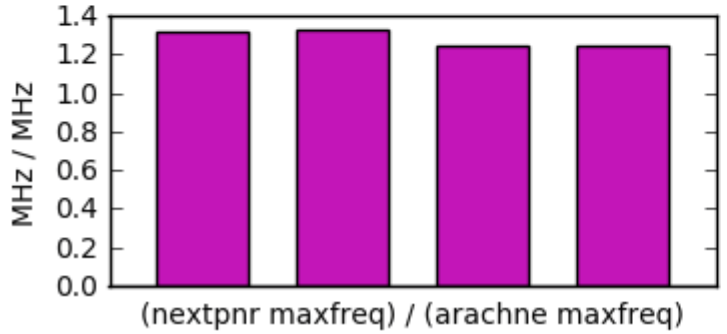
# Next steps

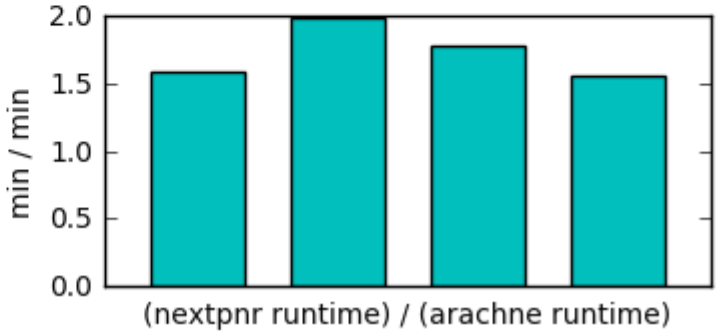
- Slowly **replace arachne-pnr** as FOSS iCE40 PnR tool in project icestorm.
  - There are still a lot of happy arachne-pnr users. We don’t want to startle them yet..
- Many **improvements** in actual placer and router
  - We hope nextpnr will also become an attractive framework for algorithms research.
- Support for **more architectures**
  - We have already documented the Xilinx 7-series bit-stream format. Therefore that would be an obvious next candidate.
  - We’d be happy to also add support for architectures that we don’t need to document by ourselves first. Please contact us at symbioticeda.com.
- **World Domination!**

# comparison – arachne-pnr vs nextpnr

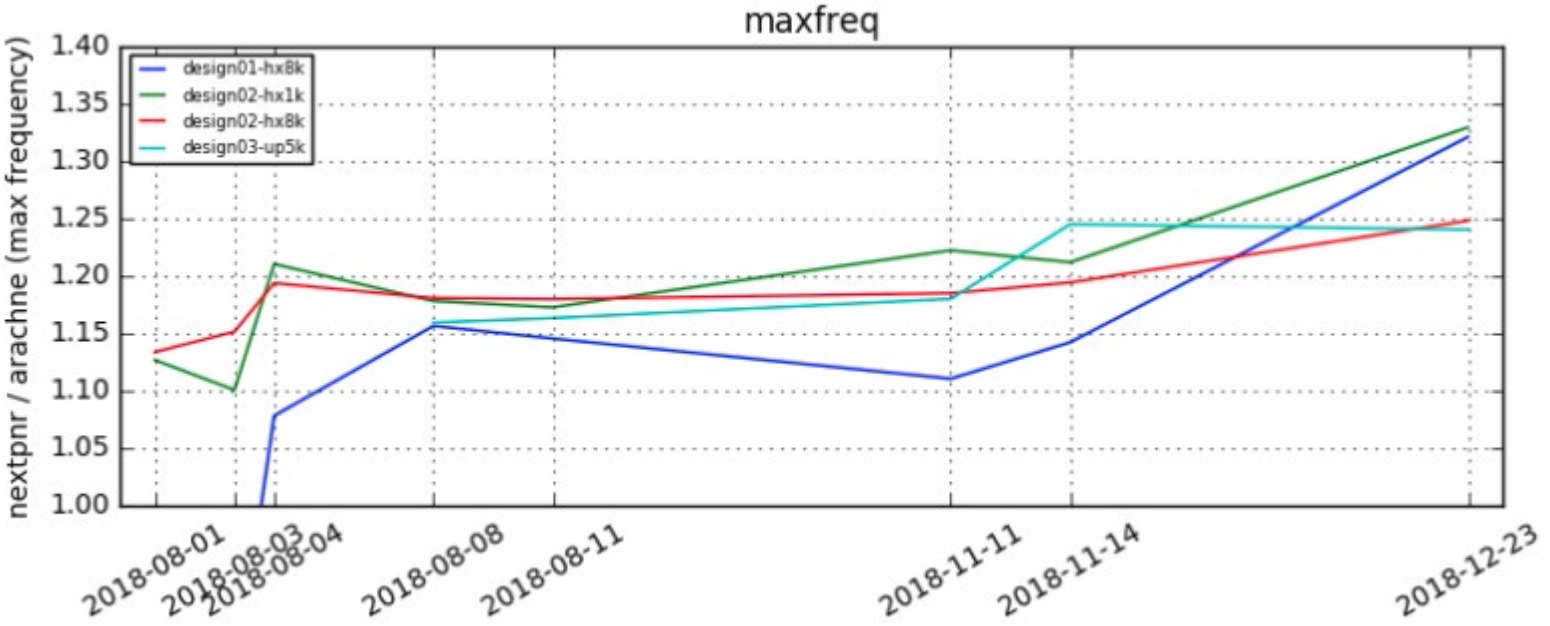
```
%matplotlib inline
import report
report.load("20181223-ice40-e76479f.dat")
report.print_summary()
```

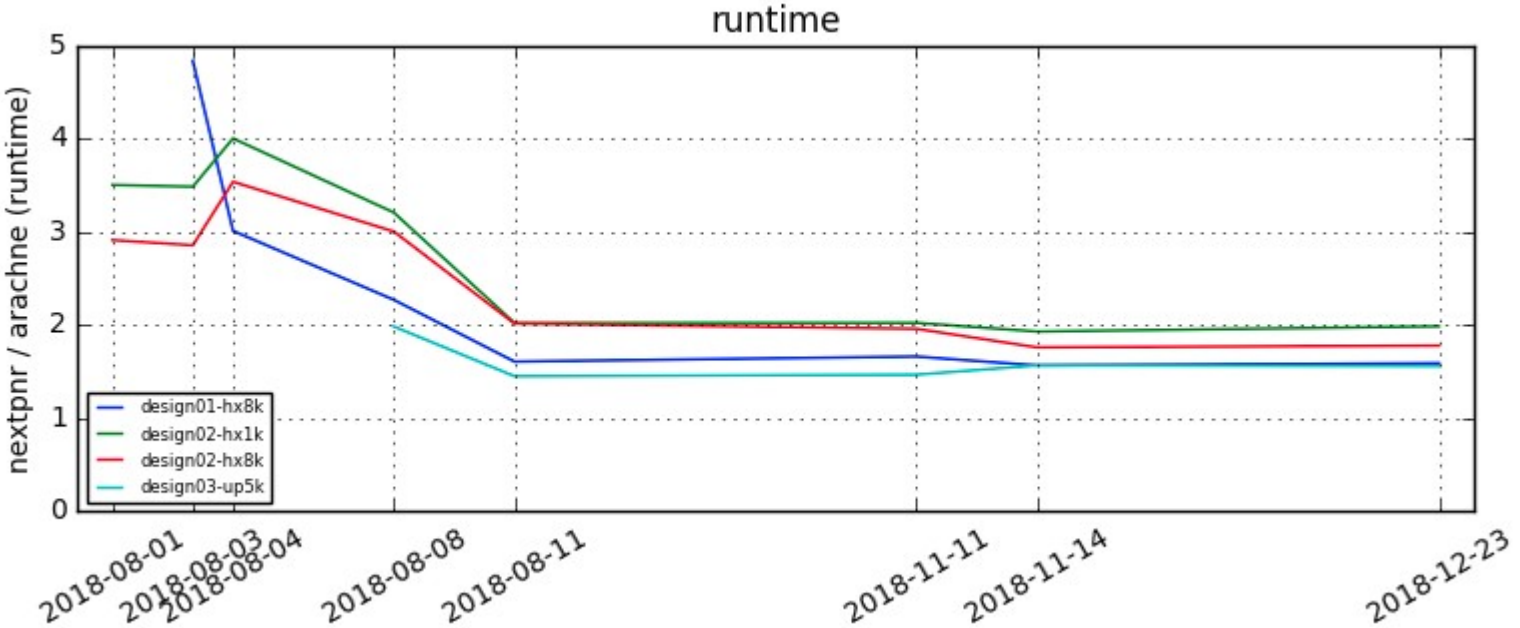
| | Max Freq (MHz) | | | Runtime (M:SS) | | |
|---|---|---|---|---|---|---|
| design | arachne | nextpnr | change | arachne | nextpnr | change |
| design01-hx8k | 27.61 | 36.45 | 1.32x | 5:51 | 9:16 | 1.59x |
| design02-hx1k | 55.36 | 73.56 | 1.33x | 0:27 | 0:54 | 1.98x |
| design02-hx8k | 57.91 | 72.25 | 1.25x | 0:44 | 1:19 | 1.78x |
| design03-up5k | 6.54 | 8.11 | 1.24x | 4:24 | 6:50 | 1.55x |




- Even though nextpnr is very new, it already consistently produces better results than arachne-pnr (in terms of max frequency).
- But arachne-pnr still produces results quicker
  - We need to do more evaluation on this. For example: How does this change if we disable the timing-driven aspects of nextpnr's algorithms.




https://github.com/YosysHQ/nextpnr-bench

Symbiotic EDA

# Demo

https://github.com/YosysHQ/nextpnr

Symbiotic EDA

</The nextpnr FOSS
FPGA place-and-route tool>

# Questions?

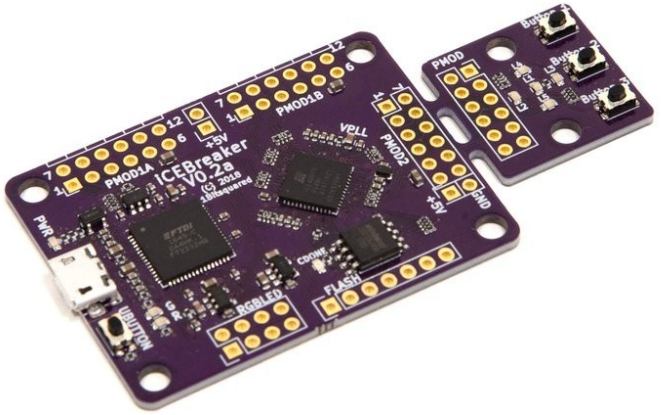
https://github.com/YosysHQ/nextpnr

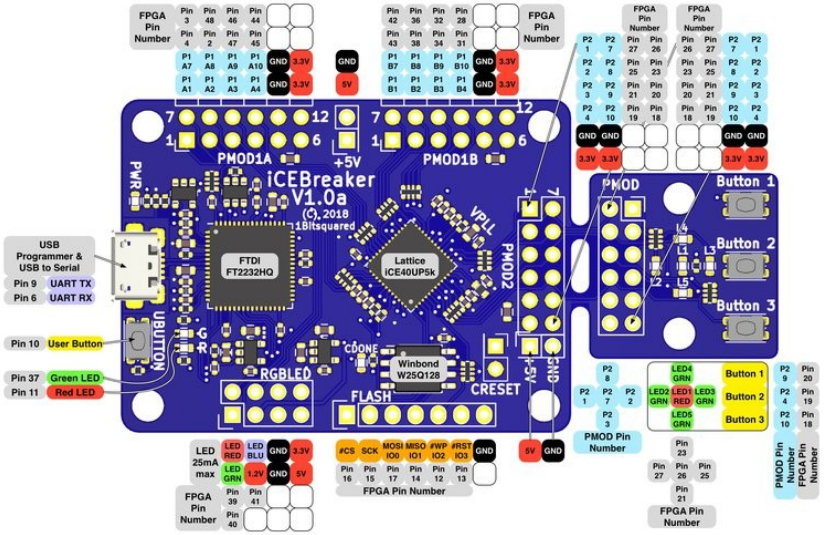
# OSDA – Open Source Design Automation Friday Workshop at DATE 2019


- Topics include:
  - Open-Source Tools, IPs, Languages, and Methodologies
  - Future directions for the open-source FPGA movement
  - Discussions on licenses, funding, and commercialization

http://osda.gitlab.io

# iCEBreaker FPGA Board






- Features iCE40UP5k
  - 5280 4-LUTs, 128kBit BRAM, 128 kByte SPRAM, 8 DSPs
  - 3x Dual-PMOD, QSPI-DDR 16 MB flash, 4 buttons, 7 LEDs
- Specifically designed as learning platform
  - There will be workshop and classes materials using all-FOSS flows
  - We are collaborating with educators on developing those courses

https://github.com/icebreaker-fpga/icebreaker

https://www.crowdsupply.com/1bitsquared/icebreaker-fpga