

nextpnr

Open source FPGA place &
route



Introduction

Rowan Goemans

Mail: goemansrowan@gmail.com

Master student @Radboud university in Nijmegen, The Netherlands

Thesis: timing constraint in nextpnr @ZITI under Myrtle “gatecat” Shah

Software background



Introduction



- I work at BDT Holland as an capability engineer:
 - Service partner for large OEM (HP, IBM, ...)
 - My responsibility is to figure out how to repair/validate the equipment they send us



- I also work at Lumi Guide as R&D engineer
 - Lumi Guide build bicycle and car detection system for parking garages
 - I work on new technologies and cost optimization of our current systems

Lumi Guide



Lumi Guide



Lumi Guide





Hardware design @ Lumi Guide

- Embedded Software background and learned FPGA as part of my job @ Lumi Guide
- First project implement a LED-matrix sign driver
- Main software implementation language used is Haskell
- Use Haskell to write LED-matrix sign driver
- Clash (<https://github.com/clash-lang/clash-compiler>) can synthesize Haskell to Verilog/VHDL.
- Not HLS
- Possible future work: Run AI inference on FPGA

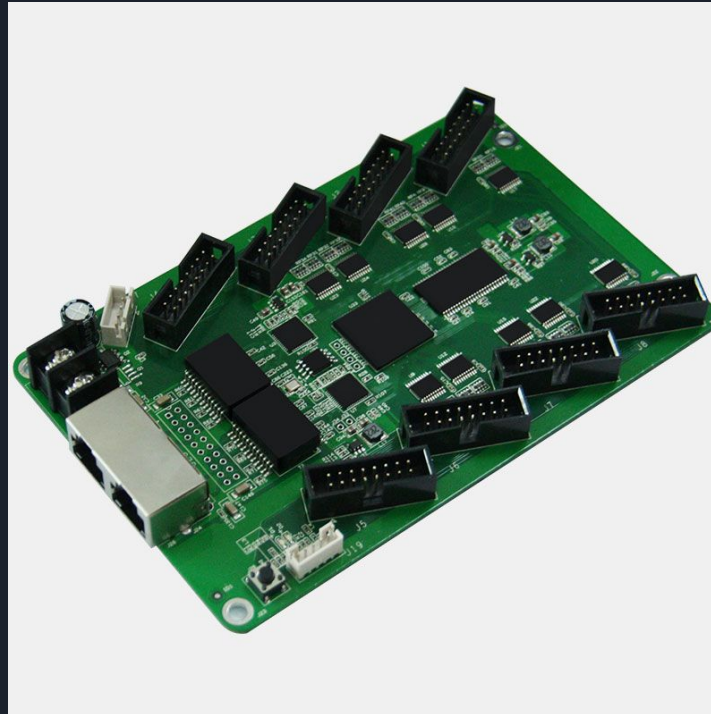


Hardware design @ Lumi Guide

- First FPGA version sign driver used DE0-Nano-SoC + Quartus
- Of all the things I had to learn when Quartus was by far the most problematic
- Constant issues surrounding CI, autogenerated interfaces, deprecated IP
- Enter Yosys + nextpnr
- Month long exploration of using Yosys + nextpnr and a “cheap” LED-matrix FPGA board from china

Hardware design @ Lumi Guide

Cost: ~10-15\$ per board





Hardware design @ Lumi Guide

- PoC was very successful
- Our V2 system is now build using the fully open source yosys + nextpnr flow
- Goodbye quartus!!
- Many advantages:
 - Time to bitstream is much lower
 - Ability to fix bugs or add features, (Ex: Yosys plugin that packs FFs into BRAM)
 - Actual useful support



Hardware design @ Lumi Guide

- But nextpnr is missing one crucial feature I would really like to have: timing constraints
- Master thesis topic under Dirk Koch & nextpnr maintainer Myrtle Shah



Short history of open source FPGA flow

- 2012: Start of Yosys synthesis tool as Claire's thesis work
- 2015: First release of icestorm for ice40, simple full flow with arachne-pnr
- 2017: Project X-Ray begins analysing xilinx 7-series bitstreams
- 2018: Development on nextpnr starts, project trellis for the ecp5
- 2019: nextpnr and trellis scale to significantly larger designs
- 2020: experimental X-Ray based Xilinx support in nextpnr
- 2021: Initial release of Project Mistral for the Intel Cyclone V and associated nextpnr support
- 2023: Full flow demonstrated for a fully open FPGA on an open process (FABulous on sky130)

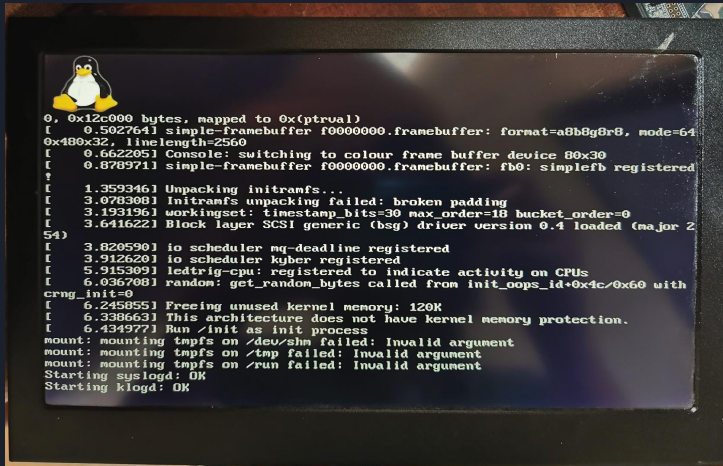


nextpnr

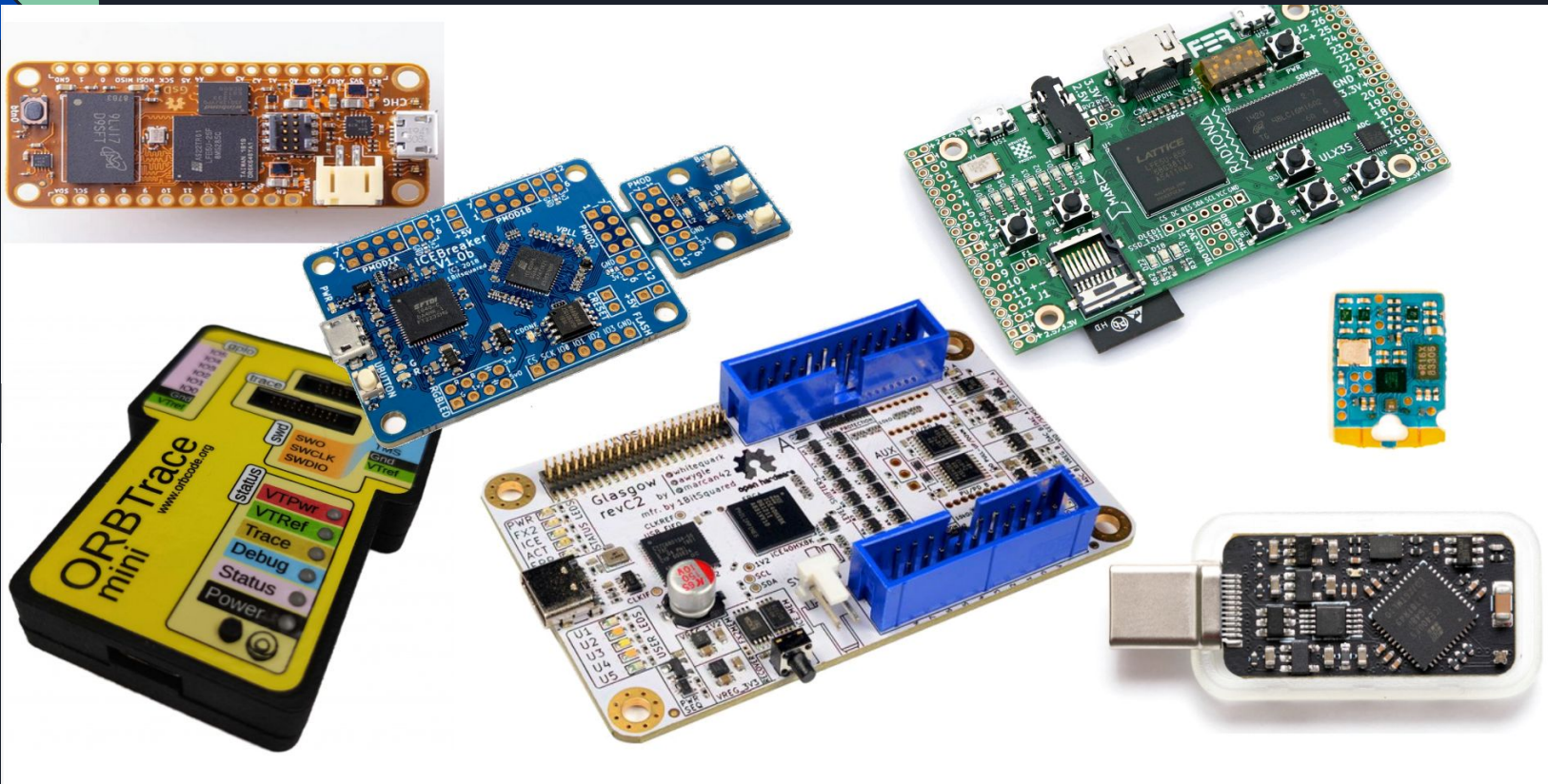
- Open source multi-architecture FPGA place & route aimed at real world FPGAs
- Support for a range of commercial FPGAs
- Supported commercially by YosysHQ GmbH in Vienna

nextpnr

Designs on Lattice ECP5 have included 64-bit Linux-capable RISC-V SoCs and an open source SNES game console replica



nextpnr

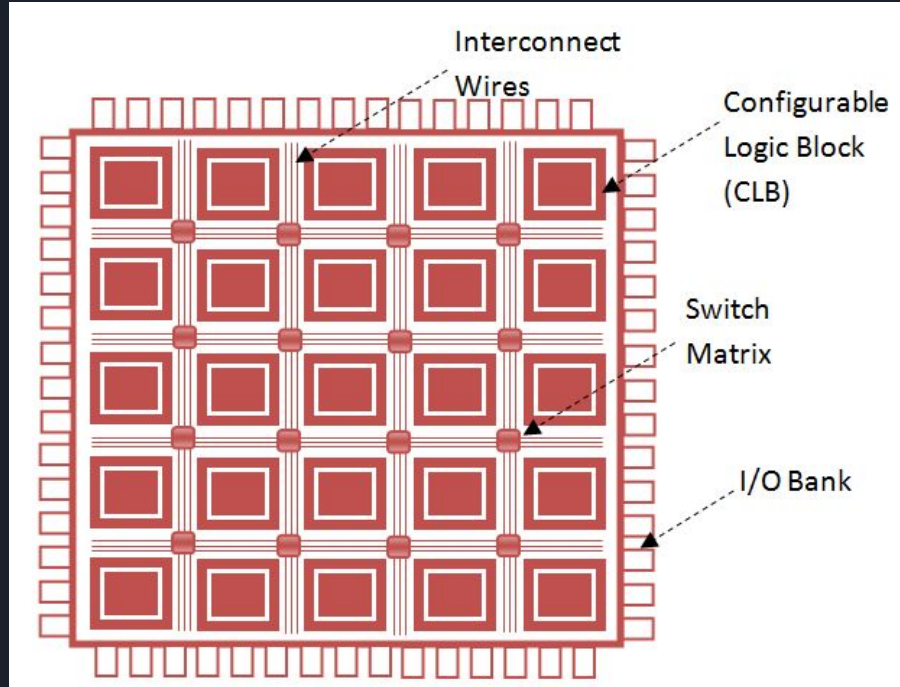




FPGA Architecture

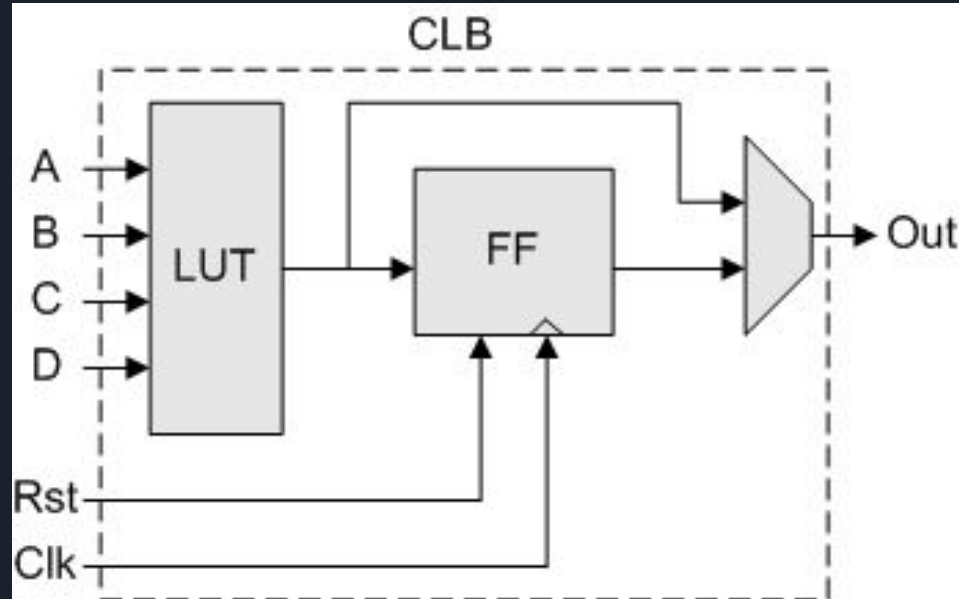
- An FPGA or Field-Programmable Gate Array is a special IC that contains configurable logic and connections
- Configuring the logic and connections allow implementation of arbitrary digital logic
- Basic structure is a switch matrix which with Programmable Interconnect Points (PIP) + Configurable Logic Blocks (CLBs)
- Programmable Interconnect Points can route signals based on the configuration
- The CLBs are used to implementation combinational and sequential logic
- Together configuring PIPs and CLBs allows the implementation of arbitrary digital logic.

FPGA Architectures



Source: <https://allaboutfpga.com/fpga-architecture/>

FPGA Architectures



Source: <https://www.fpgaakey.com/>



FPGA Architecture

- The CLB + PIP are the bread and butter of FPGA
- Every CLB can be slightly different between FPGA models
 - LUT4, LUT5 or even LUT6
 - Multiple LUTs and registers per CLB
 - Specialized in/outputs for carry chain support
 - And many more



FPGA Architecture

- Often used functionality are included as special tiles for efficiency reasons
 - blockRAM for memories
 - Multipliers/Adders
 - Floating point cores
 - Sometimes even Hard processors are embedded
 - Specialized I/O: SerDes, DDR etc
 - In the hackathon on friday we will extend an FPGA with our own custom tile.



nextpnr - Overview

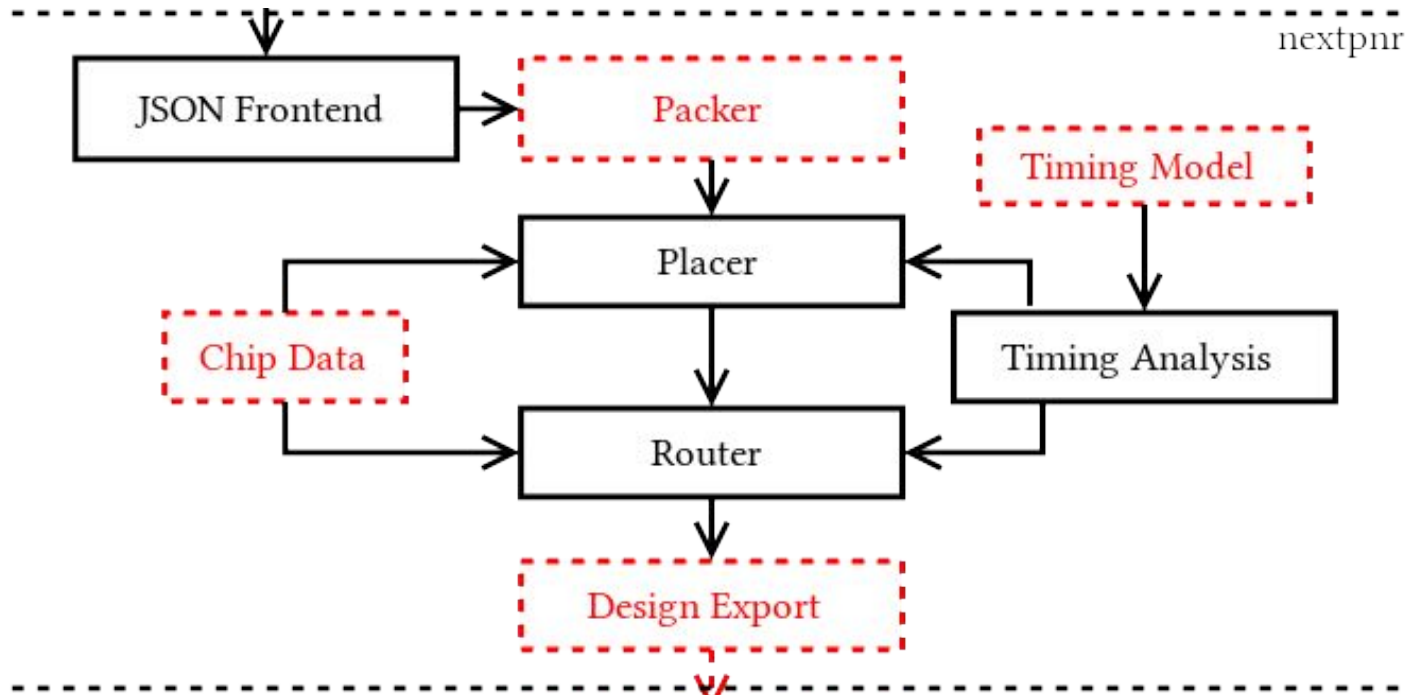
- Timing driven throughout using reverse engineered timings
- Analytical placer
- Custom scripting via Python bindings
- Customisable place and route algorithms
- Relatively low footprint and dependency requirements
- (Optional) Qt based GUI



nextpnr - Overview

- Support for various vendors and models
 - Lattice Ice40, ECP5, Nexus, MachXO2(Experimental)
 - Gowin LittleBe
 - Intel: Cyclone V (Experimental)
- Architected to allow rapid prototyping of novel architectures using the Viaduct and Himbächel frameworks
- Can scale to about 1M LUTs

nextpnr - Overview



Red boxes are architecture specific



nextpnr - Terminology

- **Bel:** Basic Element, the functional blocks of an FPGA such as LUTs, FFs, IO cells, blockrams, etc. Up to one cell may be placed at each Bel
- **Wire:** a fixed connection (“piece of metal”) inside the FPGA between Pips and/or Bel pins.
- **Pip:** Programmable Interconnect Point
- See [docs/faq.md](#), [docs/archapi.md](#), and [docs/coding.md](#) in the nextpnr repo for further discussion.

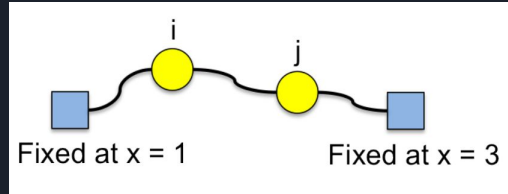


nextpnr - Placement

- Current primary placement pass is analytical placement (based roughly on the HeAP paper)
- Repeat until converged:
 - solving equations to compute optimal wirelength placement per some wirelength proxy
 - spreading cells so they don't overlap
 - legalising cells so validity rules are met
 - adding an increasing weight arc from cells to their legalised positions

nextpnr - Placement

- Simple demonstration: 2 fixed cells and 2 movable cells i and j



- Create equation for quadratic wirelength to be minimized

$$\Phi_x = (x_i - 1)^2 + (x_i - x_j)^2 + (x_j - 3)^2$$

- All variables are minimal if all partial derivatives are 0

$$\frac{\delta \Phi_x}{\delta x_i} = 2(x_i - 1) + 2(x_i - x_j) = 0$$

$$\frac{\delta \Phi_x}{\delta x_j} = -2(x_i - x_j) + 2(x_j - 3) = 0$$



nextpnr - Placement

- Rewrite to system of linear equalities and solve

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_i \\ x_j \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

- Solution: $x_i = 5/3, x_j = 7/3,$
- Solve for both X and Y locations
- Outsource to Eigen a industrial strength off the-shelf linear algebra solution



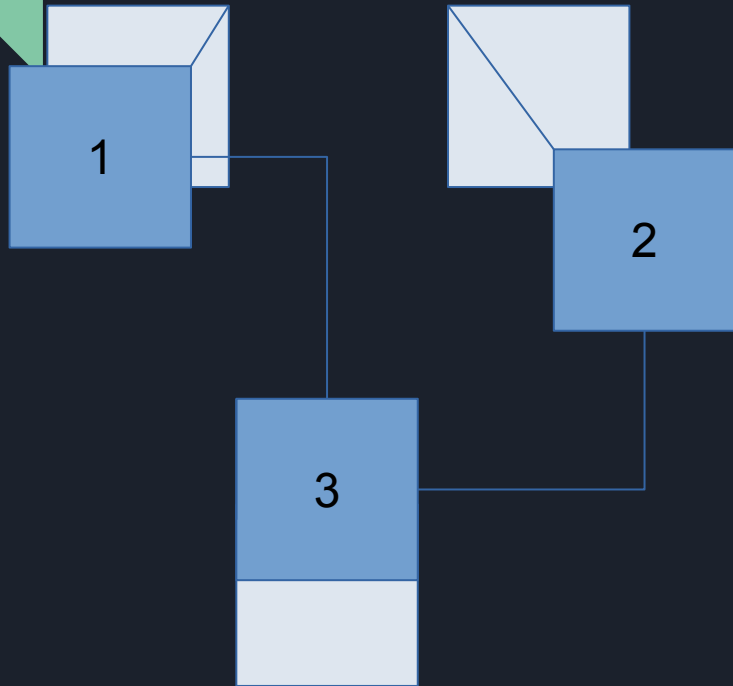
nextpnr - Placement

- Prior algorithm in nextpnr is augmented with weight per wire length called Bound2bound
- Weight: $(1 + \text{criticality}) / (\text{fanout} * |x_i - x_j|)$


Criticality:

- value between 0.0 and 1.0 given by timing engine
- 0.0 = don't care, 1.0 critical path of the circuit
- Result: Very tight, but also very illegal placement

nextpnr - Placement



- Iteratively spread to legal locations
- spread to grid locations
- add arc to spread position
- Repeat with increasing weight until a legal placement has been found
- Afterwards refinement steps take place (Simulated Annealing, swapping cells)



nextpnr - Router

- Route the wires of the fabric using the PIP
- Minimize wire lengths to optimize circuit performance
- Main router (router1) bespoke implementation by Claire Wolf of Yosys fame
- Unfortunately I do not know in detail how it works.
- Main takeaway: Uses Criticality from timing engine as well to guide routing



nextpnr - Timing engine

- Informs the placer and router how “good” their results are
- Criticality is the central number
 - 0.0 = don't care
 - 1.0 critical path of the circuit
- Higher quality place and route results means your circuit can run faster
- After place and route it prints out fmax figures per clock domain

nextpnr - Timing engine

- Generic for every architecture
- Informed by fabric and cell timings
- Ex: ECP5 multiplier timing
(http://yosyshq.net/prjtrellis-db/ECP5/timing/cell_timing_6.html)
- Analyses all combinational paths starting or ending in a FF

MULT18X18D:REGS=NONE

Propagation Delays

From Port	To Port	Low-High Transition (ps)			High-Low Transition (ps)		
		Min	Typ	Max	Min	Typ	Max
A	P	2909	3418	3927	2909	3418	3927
B	P	2909	3418	3927	2909	3418	3927
SIGNEDA	P	2740	3234	3728	2740	3234	3728
SIGNEDB	P	2740	3234	3728	2740	3234	3728



nextpnr - Timing engine Terminology

- **Setup time:** The time a signals needs to be stable before the clock edge
- **Hold time:** The time a signal needs to be stable after the clock edge
 - Limited/No support for hold time checks in nextpnr at this time
- **Required time:** The time before which a signal must arrive
- **Arrival time:** The actual time the signal will arrive
- **slack:** required - arrival time



nextpnr - Timing engine flow

- Topologically sort netlist
- Walk forward through all ports and assign arrival times:
 - Out port: Propagate delay through net and add routing delay to all users
 - In port: Propagate delay through combinational outputs and add combinational delay
- Initialize the setup/hold as the required time for all FFs
- Walk backwards through all ports:
 - Out port: Propagate delay through combinational inputs subtracting combinational delay
 - In port: Propagate delay back through all drivers subtracting routing delay



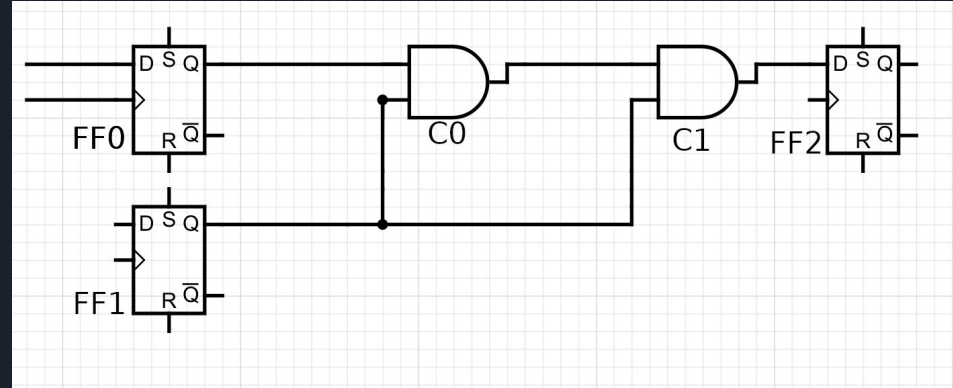
nextpnr - Timing engine

- Calculate slack based on required and arrival times
- Calculate criticalities for usage in placer and router
- Optionally report fmax

```
void TimingAnalyser::run(bool update_route_delays)
{
    ... reset_times();
    ... if (update_route_delays)
    ... | ... get_route_delays();
    ... walk_forward();
    ... walk_backward();
    ... compute_slack();
    ... compute_criticality();
}
```

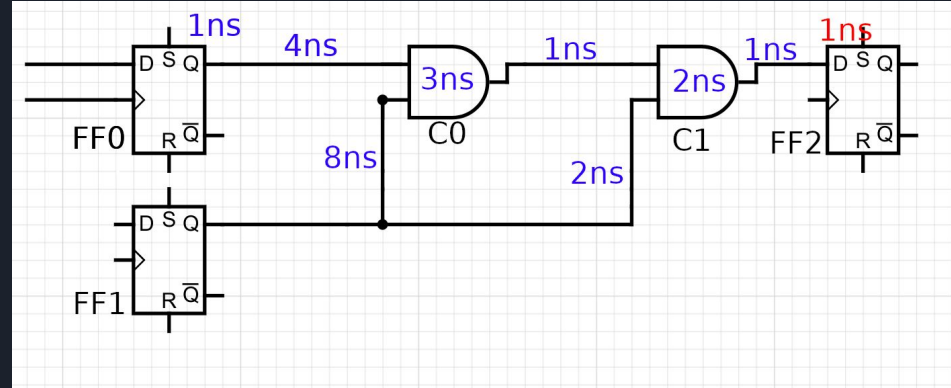
nextpnr - Timing engine worked example

- First let's annotate the routing delays and combinational **delays** and also the **setup** time



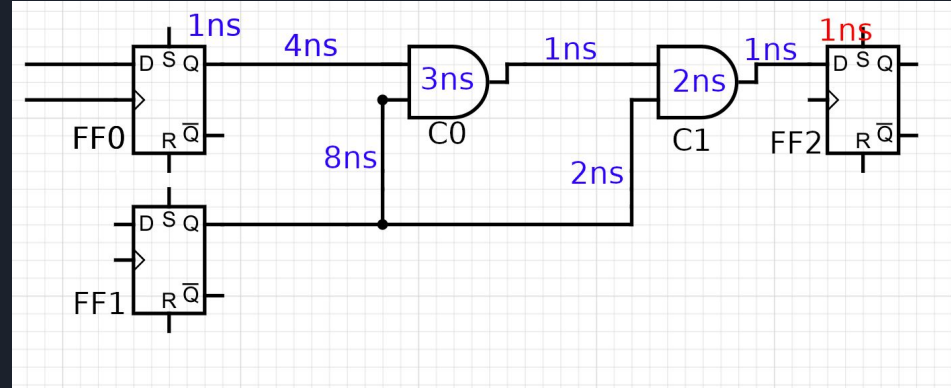
nextpnr - Timing engine worked example

- First let's annotate the routing delays and combinational **delays** and also the **setup** time
- Topological sort:



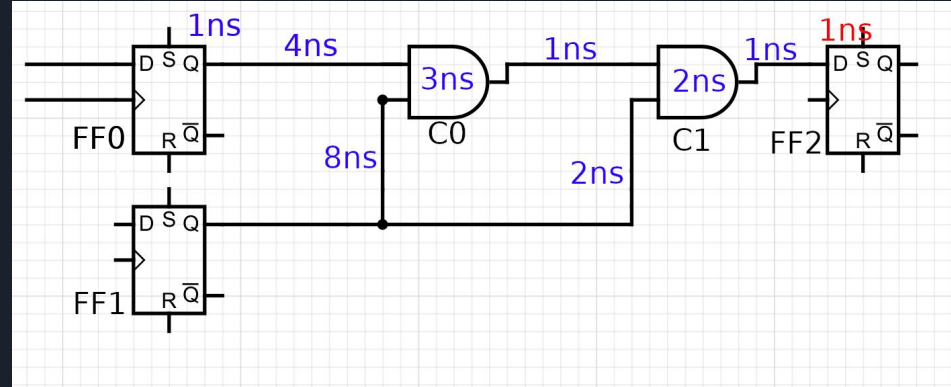
nextpnr - Timing engine worked example

- First let's annotate the routing delays and combinational **delays** and also the **setup** time
- Topological sort: FF0 -> FF1 -> C0 -> C1 -> FF2



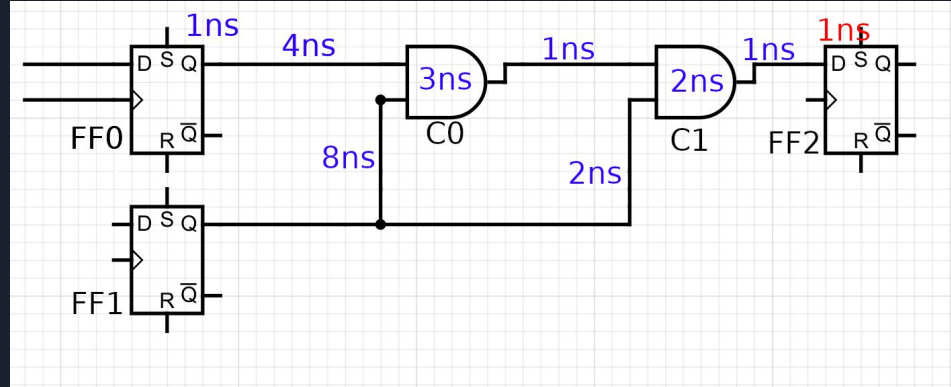
nextpnr - Timing engine worked example

- First let's annotate the routing delays and combinational **delays** and also the **setup** time
- Topological sort: FF0 -> FF1 -> C0 -> C1 -> FF2
- Delays
 - FF0 -> C0 -> C1 -> FF2: 12ns
 - FF1 -> C0 -> C1 -> FF2: 17ns
 - FF1 -> C1 -> FF2: 5ns



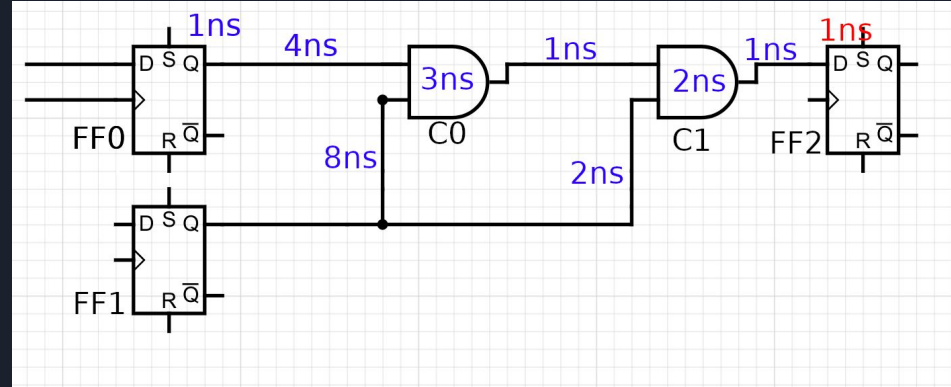
nextpnr - Timing engine worked example

- First let's annotate the routing delays and combinational **delays** and also the **setup** time
- Topological sort: FF0 -> FF1 -> C0 -> C1 -> FF2
- Delays
 - FF0 -> C0 -> C1 -> FF2: 12ns
 - FF1 -> C0 -> C1 -> FF2: 17ns
 - FF1 -> C1 -> FF2: 5ns
- Setup time is 1ns
 - FF0 -> C0 -> C1 -> FF2: 13ns
 - FF1 -> C0 -> C1 -> FF2: 18ns
 - FF1 -> C1 -> FF2: 6ns



nextpnr - Timing engine worked example

- First let's annotate the routing delays and combinational **delays** and also the **setup** time
- Topological sort: FF0 -> FF1 -> C0 -> C1 -> FF2
- Delays
 - FF0 -> C0 -> C1 -> FF2: 12ns
 - FF1 -> C0 -> C1 -> FF2: 17ns
 - FF1 -> C1 -> FF2: 5ns
- Setup time is 1ns
 - FF0 -> C0 -> C1 -> FF2: 13ns
 - FF1 -> C0 -> C1 -> FF2: 18ns
 - FF1 -> C1 -> FF2: 6ns



- Slowest path is 18ns that means this circuit has an fmax of 55.5 Mhz



nextpnr - Timing constraints

- Topic of my master thesis
- Supported: Clock frequency constraints
 - Even automatic derivation of PLL generated clocks if input is known
- Not supported:
 - **False path:** ignore a path completely
 - **Min/max delay:** Set maximum/minimum allowed delay
 - **IO delay:** Set input/output delay at the edge of the FPGA
 - **Multicycle paths:** Allow a single path to take multiple clock cycles



nextpnr - Timing constraints

- Timing constraints essential for:
 - High performance interface between FPGA and the world
 - Clock domain crossings (CDC)
 - Partial reconfiguration
- Asynchronous FIFO example
 - Domains A and B have an unknown relationship
 - Dual port blockRAM, sync write port in domain A, sync read port in domain B
 - Gray code write pointer in domain A, gray code read pointer in domain B
 - FIFO full/empty computation requires both write/read pointer
 - Synchronize write pointer to domain B using FFs synchronizer chain
 - Synchronize read pointer to domain A using FFs synchronizer chain




nextpnr - Timing constraints

Are we happy?

nextpnr - Timing constraints





nextpnr - Timing constraints

- But we have used gray pointers AND nicely synchronized them?
- Yes! But nextpnr knows nothing about the domains so it cannot decide on a criticality or an fmax figure for this path
- This means it's possible for the individual bits of the gray pointers to have vastly different delays.
- **Breaks assumption that max one bit changes each cycle**
- **Meaning the FF synchronizer chain is no longer a valid word synchronizer**

OUR POINTERS COULD BE CORRUPTED



nextpnr - Timing constraints

Max delay constraint to the rescue:

- Constrain path of output of FF in A to input of FF in B with the smaller clock period of both domains, and vice versa.
- `set_max_delay -from [get_cell src_gray_ff*] -to [get_port dst_graysync_ff*] ([expr min ($src_clk_period, $dest_clk_period)])`
 - **from:** Where to start the max delay constraint from. These are the registers named “`src_gray_ff*`” The `*` is a wildcard
 - **To:** Where to max_delay constraint ends. These are the first FFs in the synchronizer chain: “`dst_graysync_ff*`”
 - And finally we say we set the max delay to the minimum of both clock periods.



nextpnr - Timing constraints

- This is still future work and the topic of my Master Thesis
- Will require re-engineering of the timing analyser
- **false_paths/max_delay**: We can piggy-back off criticality
- **min_delay** may require an additional pass after routing.
- **io_delay** will require teaching nextpnr about delay elements which may need to be dynamically inserted
- **multicycle_path** will work with criticality as well



nextpnr

Yosys + nextpnr
demo



nextpnr

Questions?