OpenFPGA: Bringing Open-source Hardware to FPGAs

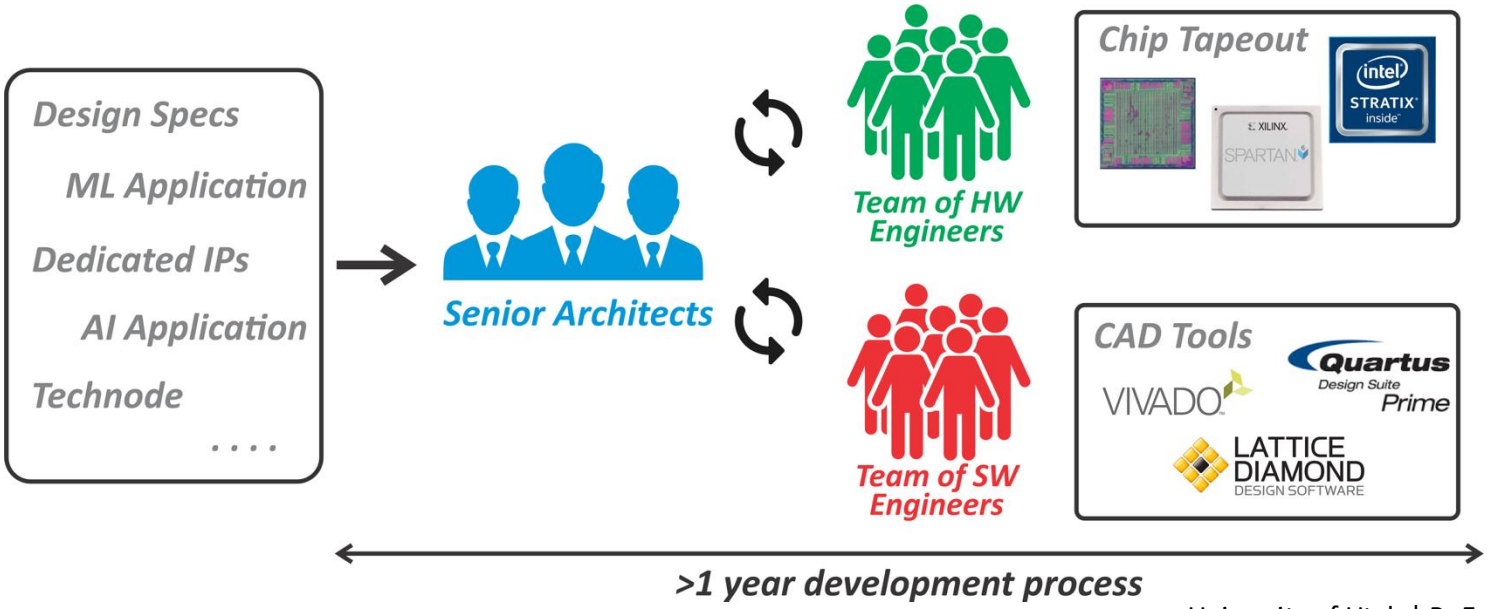
Pierre-Emmanuel Gaillardon
Department of Electrical and Computer Engineering
University of Utah

Hawai'i Scientific Data Workshop
06/18/2025

Why OpenFPGA?

- FPGAs are an attractive solution for *flexibly* deploying domain acceleration to modern workloads on both data centers and edge targets
- Domain-specific applications demand a specific type of computing resources of FPGA
  - AI applications are typically DSP-hungry
  - Commercial FPGAs could be sub-optimal since they are tailored for generic applications

Designing FPGA fabrics is traditionally a cumbersome process



The OpenFPGA Framework – In a nutshell

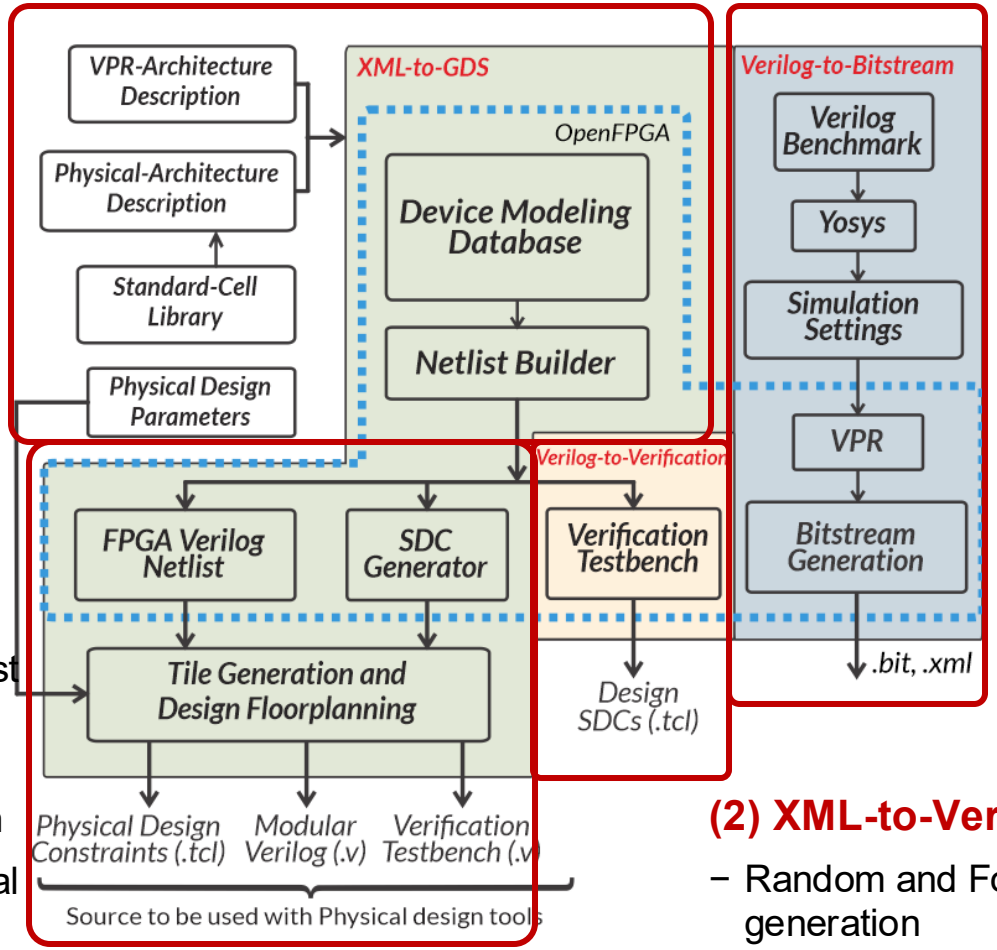
Unified code base for fabric generation, design verification and end-user bitstream generation

(1) Inputs

- FPGA Modelled in XML language
- Standard Cell mapping XML
- Physical constraint information

(4) XML-to-GDS

- Synthesizable and modular Verilog netlist generation
- Modular design constraint conversion
- Tiling and Hierarchical transformation to ease physical design



(3) Verilog-to-Bitstream

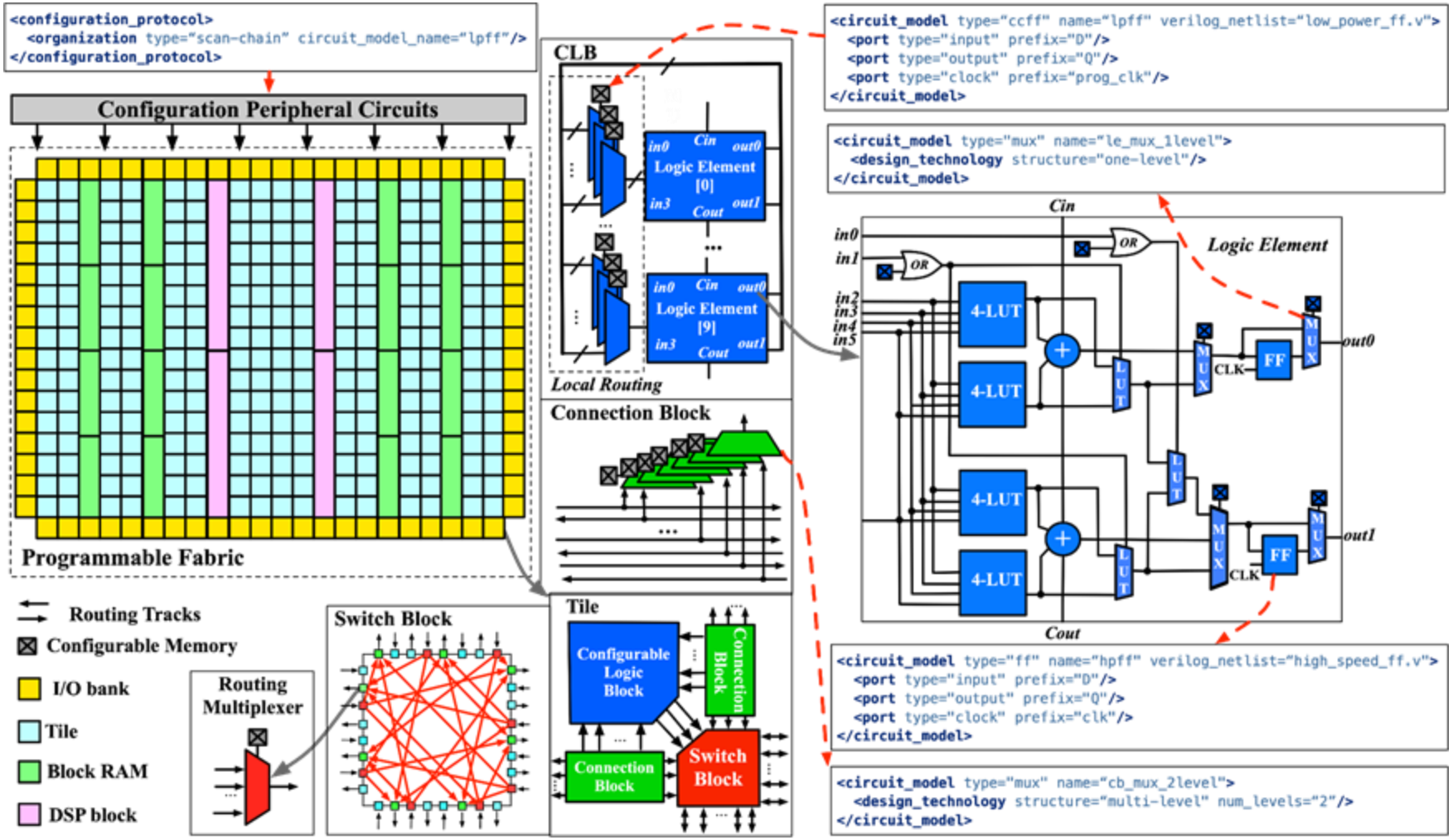
- Tight integration existing opensource tools
- Synthesis using Yosys or ODIN-II
- Pack-Place-Route using Verilog-to-routing project
- Compatible Bitstream generation based of selected configuration protocol

(2) XML-to-Verification

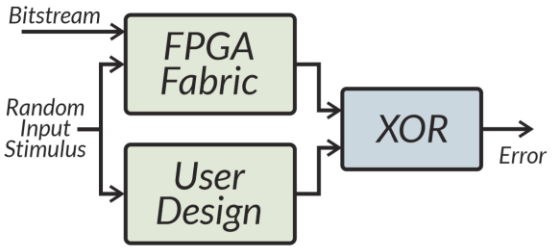
- Random and Formal testbench generation
- Customizable configuration and operating phase verification

(1) Architecture modelling with OpenFPGA

Designers can customize any circuit element at any location of FPGA fabric

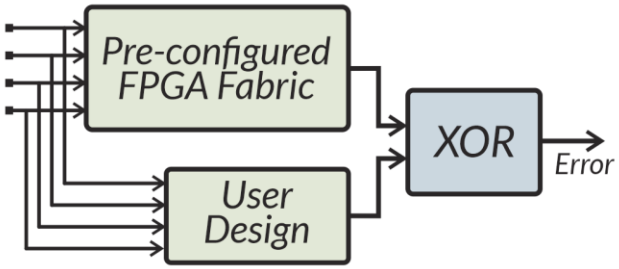


(2) Customizable Design Verification



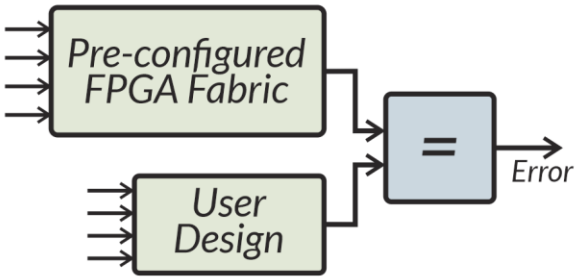
Random vector testbench generation

- XOR outputs generated from user design and FPGA Fabric
- Optionally executes the configuration phase
- Flags all unmatched output



Pre-Configured fabric verification

- Generates pre-configured FPGA fabric
- Generate 1:1 wrapper to represent user design IOs
- Can be used with user design testbench



Formal Verification

- Generates pre-configured FPGA fabric with design wrapper
- Can be used with formal verification tools (such as formality) to perform faster formal verification

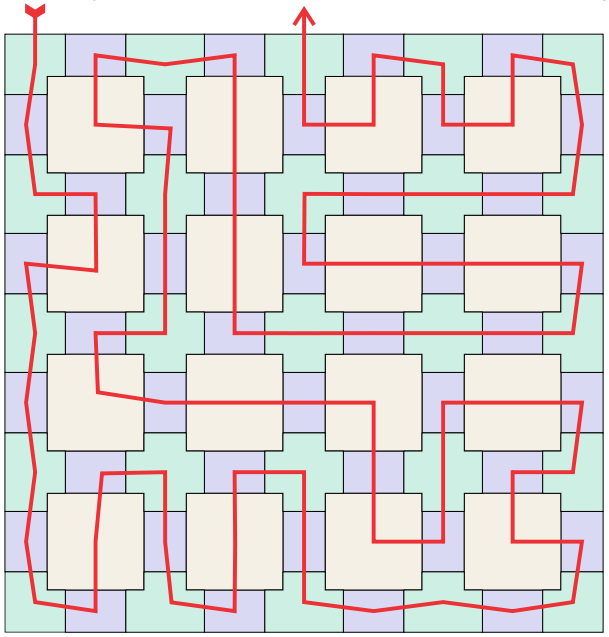
(3) Bitstream Generation Features

- Generates the bitstream **compatible with the selected configuration protocol**
- Generates the **unencrypted bitstream** files in various formats, like .bit/.txt/.xml for different use case (**like bitstream manipulation**)

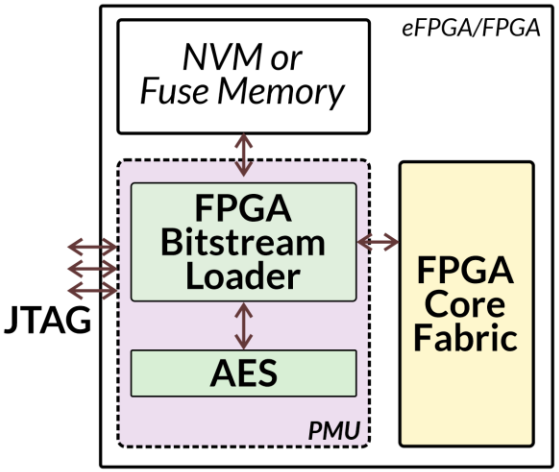
Human readable bitstream format

```
<bitstream_block name="fpga_top" hierarchy_level="0">
  <!-- Bitstream block of a 4-input Look-Up Table in a Configur
  <bitstream_block name="grid_clb_1_1" hierarchy_level="1">
    <bitstream_block name="logical_tile_clb_mode_clb__0" hierar
      <bitstream_block name="logical_tile_clb_mode_default__fle
        <bitstream_block name="logical_tile_clb_mode_default__f
          <bitstream_block name="logical_tile_clb_mode_default_
            <bitstream_block name="lut4_config_latch_mem" hiera
              <hierarchy>
                <instance level="0" name="fpga_top"/>
                <instance level="1" name="grid_clb_1_1"/>
                <instance level="2" name="logical_tile_clb_mode
                <instance level="3" name="logical_tile_clb_mode
                <instance level="4" name="logical_tile_clb_mode
                <instance level="5" name="logical_tile_clb_mode
                <instance level="6" name="lut4_config_latch_mem
              </hierarchy>
              <bitstream>
                <bit memory_port="mem_out[0]" value="0"/>
                <bit memory_port="mem_out[1]" value="0"/>
                <bit memory_port="mem_out[2]" value="0"/>
                <bit memory_port="mem_out[3]" value="0"/>
                <bit memory_port="mem_out[4]" value="0"/>
                <bit memory_port="mem_out[5]" value="0"/>
                <bit memory_port="mem_out[6]" value="0"/>
                <bit memory_port="mem_out[7]" value="0"/>
```

Fabric Key (Encrypt the bitstream sequence)



Platform management Units

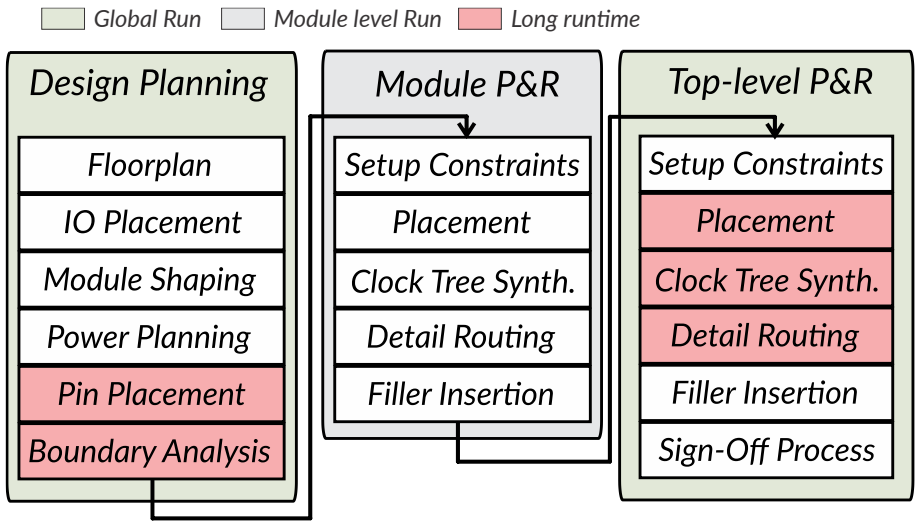


(4) OpenFPGA-Physical: XML-to-GDS flow

- Physical design most critical part of entire FPGA fabric design flow
- Standard hierarchical ASIC toolchain flow can not exploit the regularity and results in long implementation runtime

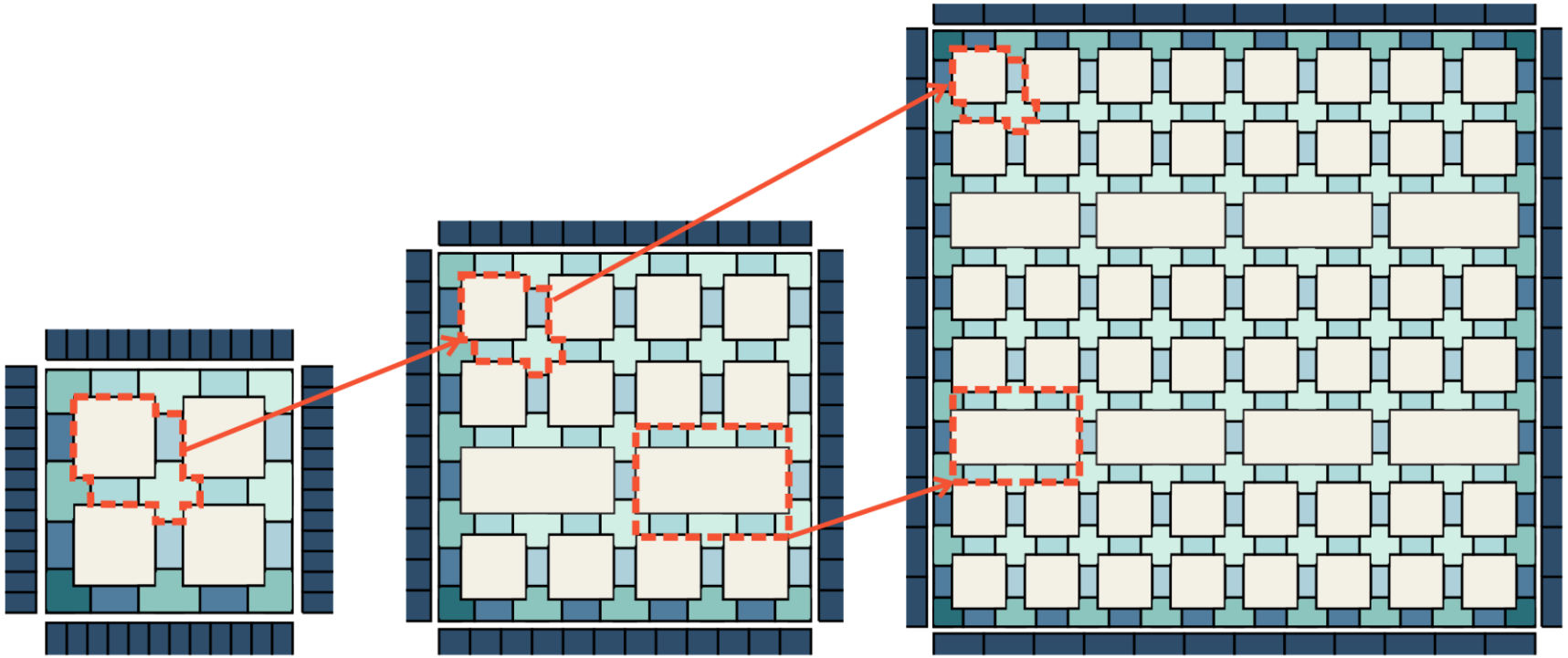
How does OpenFPGA address physical design challenges?

1. Relies on semi-custom design flows
2. Tileable architecture to limit the variation across the fabric
3. Priorities regularity for lower runtime and faster signoff
4. Scaling homogeneous designs to heterogeneous architectures
5. Post module P&R optimization
6. Feed throughs and buffer insertion to perform global optimization



Top-level LEGO assembly of FPGA fabric

1. Similar type of blocks used in each scaled version of the fabric
2. The idea can scale a **homogeneous design to heterogenous fabric**



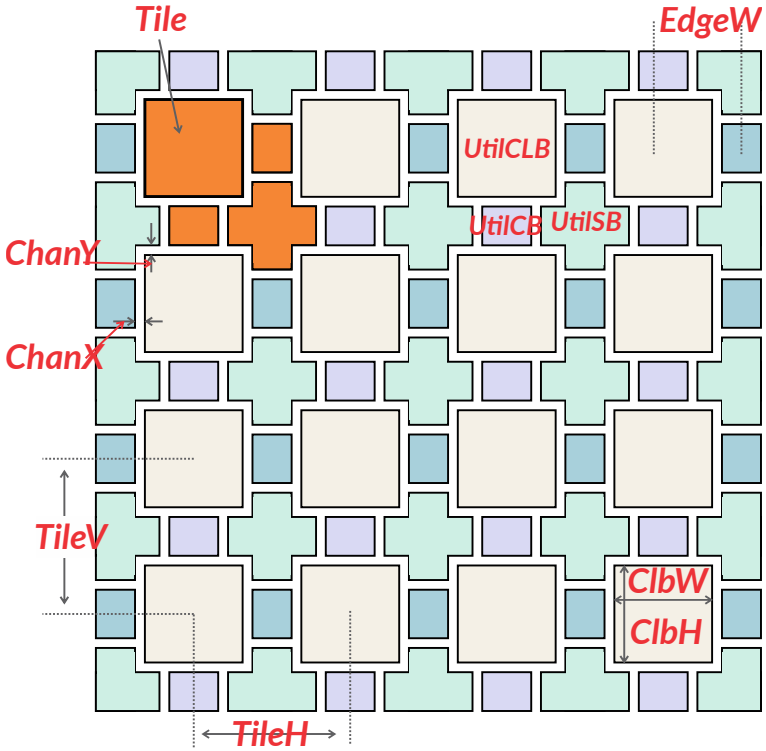
Homogeneous Fabric

Heterogenous Fabric

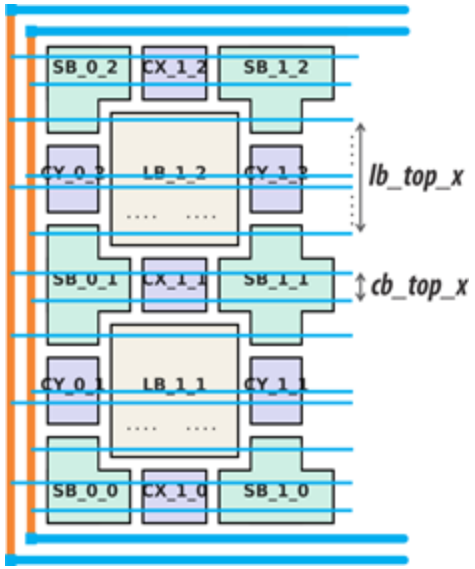
Scaled Heterogenous Fabric

Focus on the Hyper-Regularity

- An **auto shaping** is performed based on few key parameters like **utilization of each unique module**
- A regular power grid **scales gracefully** with the size of the FPGA



Auto Shaping Parameters

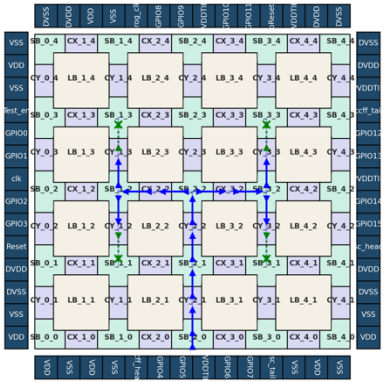


Regular Power Distribution

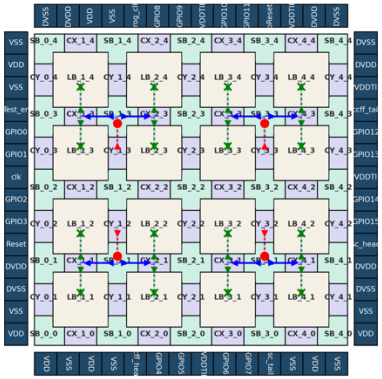
Global Hierarchical FPGA Clock Tree Synthesis

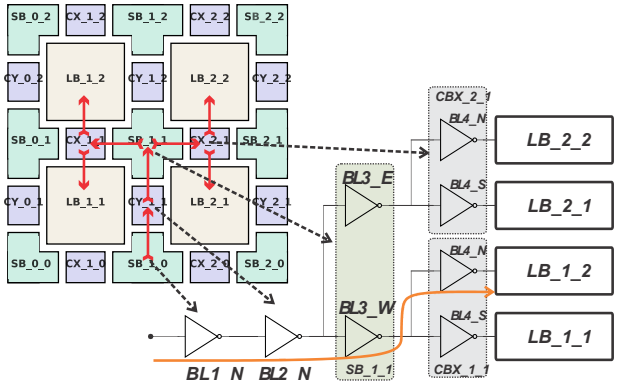
- **Multi-level clock connectivity** to allow flexible buffering
- The pre-routed buffers are scaled after the top-level place and route to **optimize latency and skew** of the entire design

Level 2

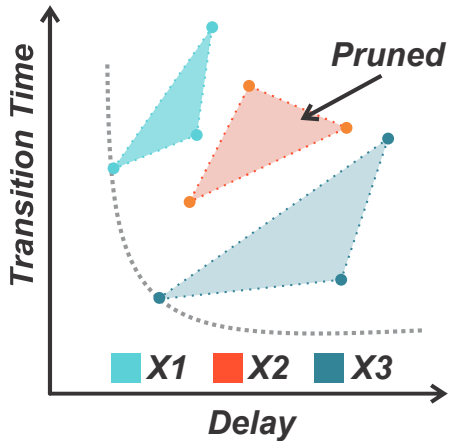


Level 1





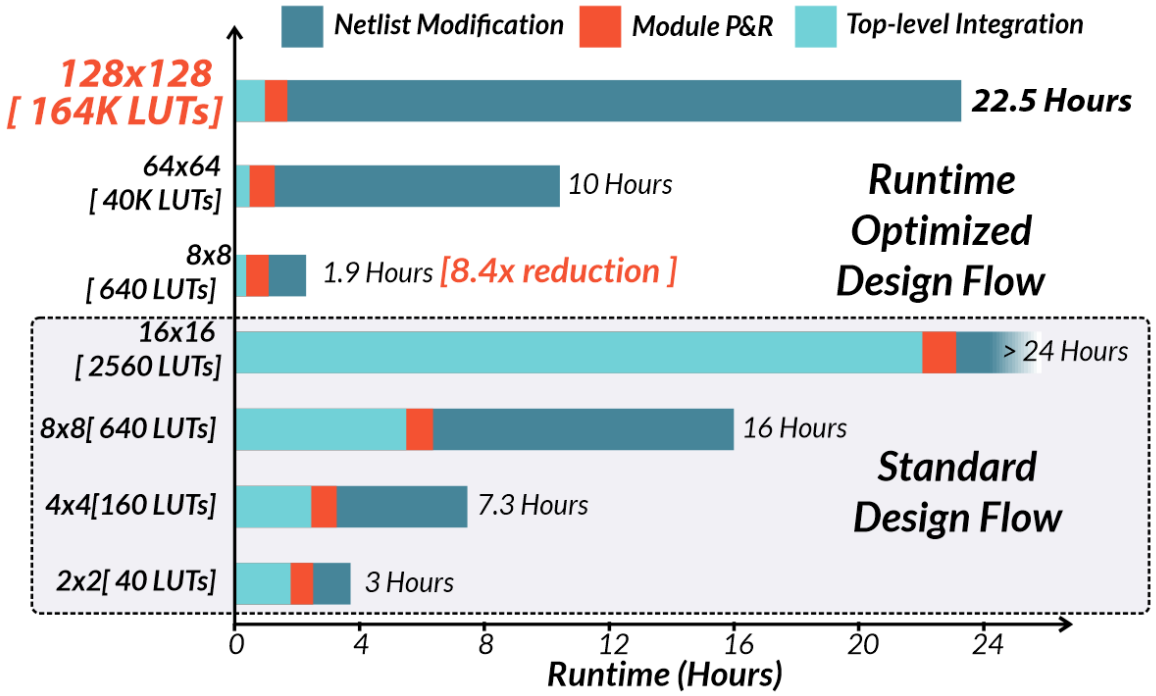
Buffer tree placement



Van Ginneken Buffer sizing with Pareto Pruning Strategy

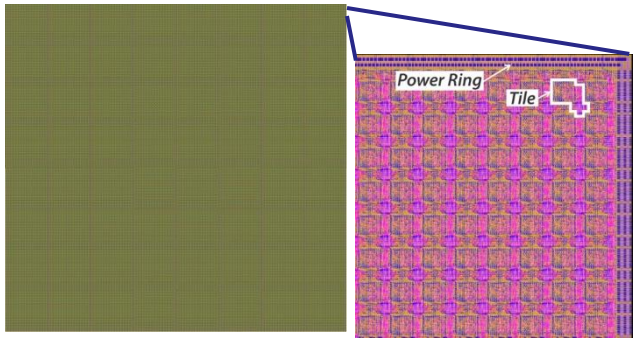
Scalable design flow runtime

- Over **8.4x runtime improvement for small fabrics** (8x8 with 640 LUTs)
- **100K+ LUTs design achievable in <24hours**



Design	Opt. Hier. (ps) Latency	Skew
2×2	134	10.11
4×4	304	12.45
8×8	437	15.4
16×16	2658	19.87
32×32	4566	32.2
64×64	8954	50.5
128×128	21012	70.56

Scaled FPGA Design and Clock Timing



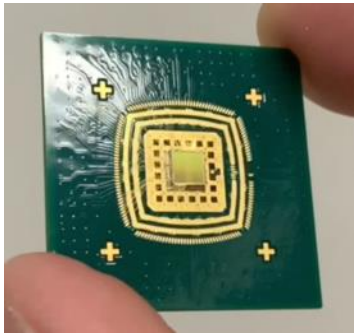
128x128 FPGA 164K LUTs

OpenFPGA is Silicon Proven!

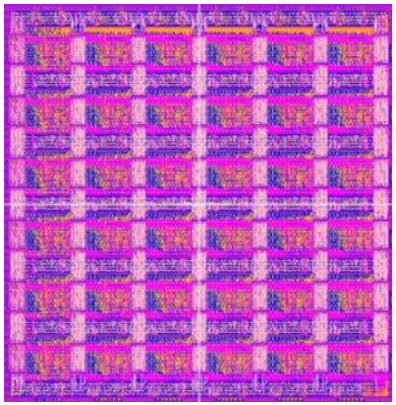
FROG – First Reconfigurable Open-source Gate array

12nm GlobalFoundries

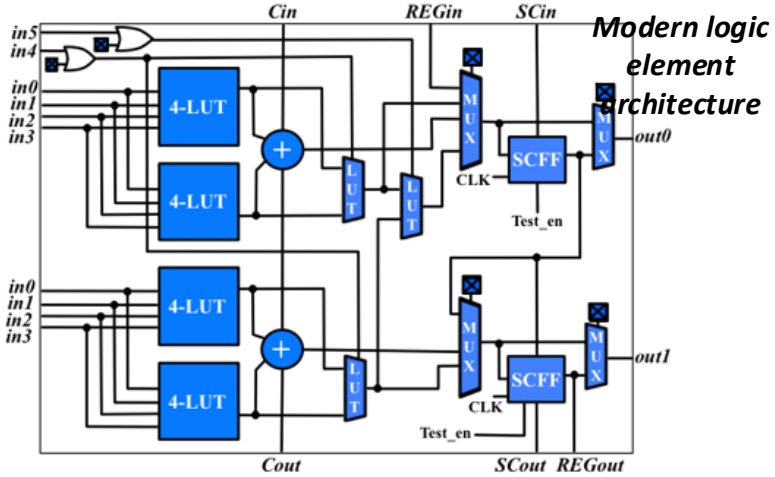
Effort 2.5 person.month



[Author's Own]



[Author's Own]



Modern logic element architecture

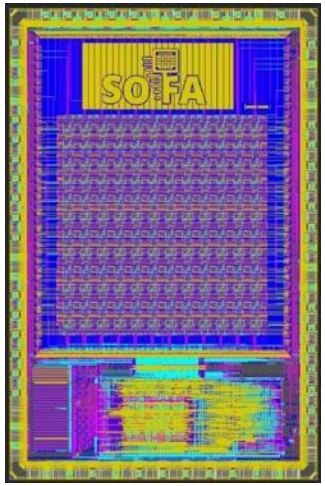
SOFA – Skywater Open-source FpgA

Integration to Caravel SoC

130nm Skywater

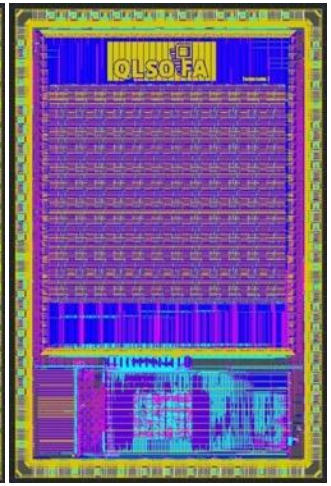
Effort 1 person.month

Google + efabless.com



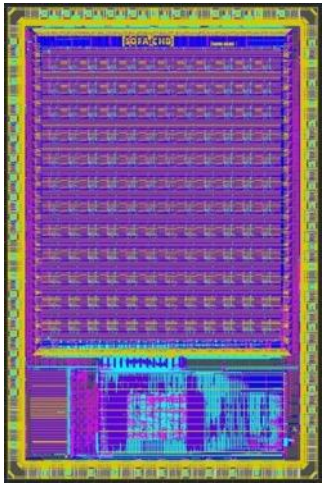
SOFA HD

Base design



QLSOFA HD

Arithmetic-enhanced



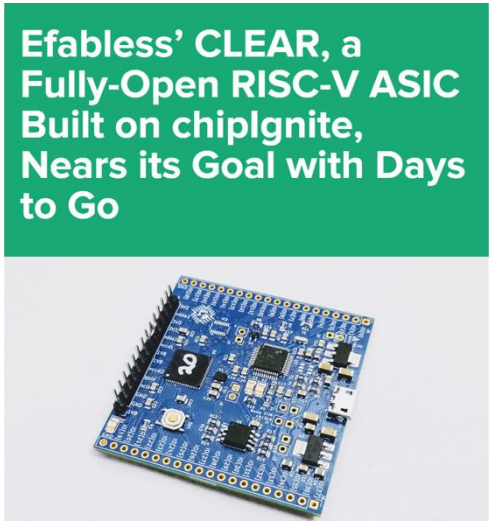
SOFA CHD

Performance-optimized

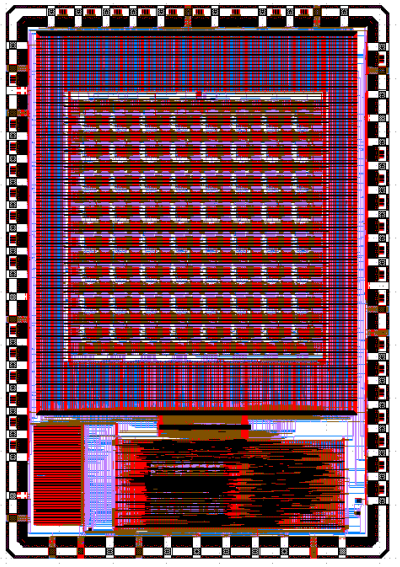
Beyond the Academic Tool

The backbone of **QuickLogic's Australis® eFPGA IP generator**

The backbone of **efabless's CLEAR** open-source eFPGA SoC platform



The enabler for the FPGA chips in **Google-Skywater MPW shuttle**



OpenFPGA Github: *https://github.com/lnis-uofu/OpenFPGA*
OpenFPGA Documentation: *https://openfpga.readthedocs.io/en/master/*

Summary

OpenFPGA is a ***leading open-source FPGA IP generator***

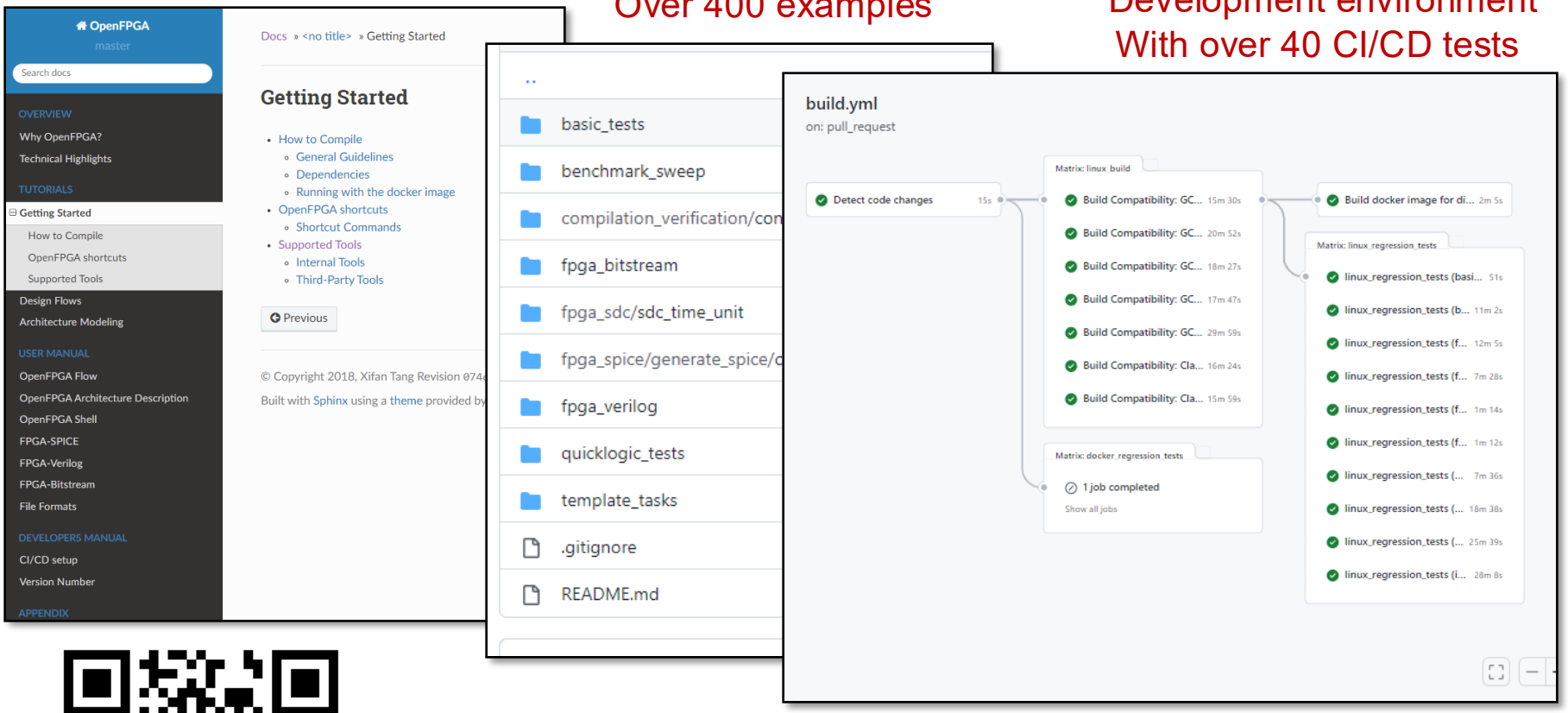
- Support highly customizable FPGA architecture design
- Provide most complete open-source EDA support
- Enable 24-hour development cycle to prototype FPGAs
- Enabled 20+ FPGA tape-outs in the past 4 years

Detailed Documentation and Development setup

Detailed Documentation

Over 400 examples

Sophisticated Development environment With over 40 CI/CD tests



OpenFPGA Github: https://github.com/lnis-uofu/OpenFPGA

Thank you for your attention

Questions?

Laboratory for NanoIntegrated Systems
Department of Electrical and Computer Engineering
SMBB building – University of Utah – Salt Lake City – UT – USA