Open FPGA introduction

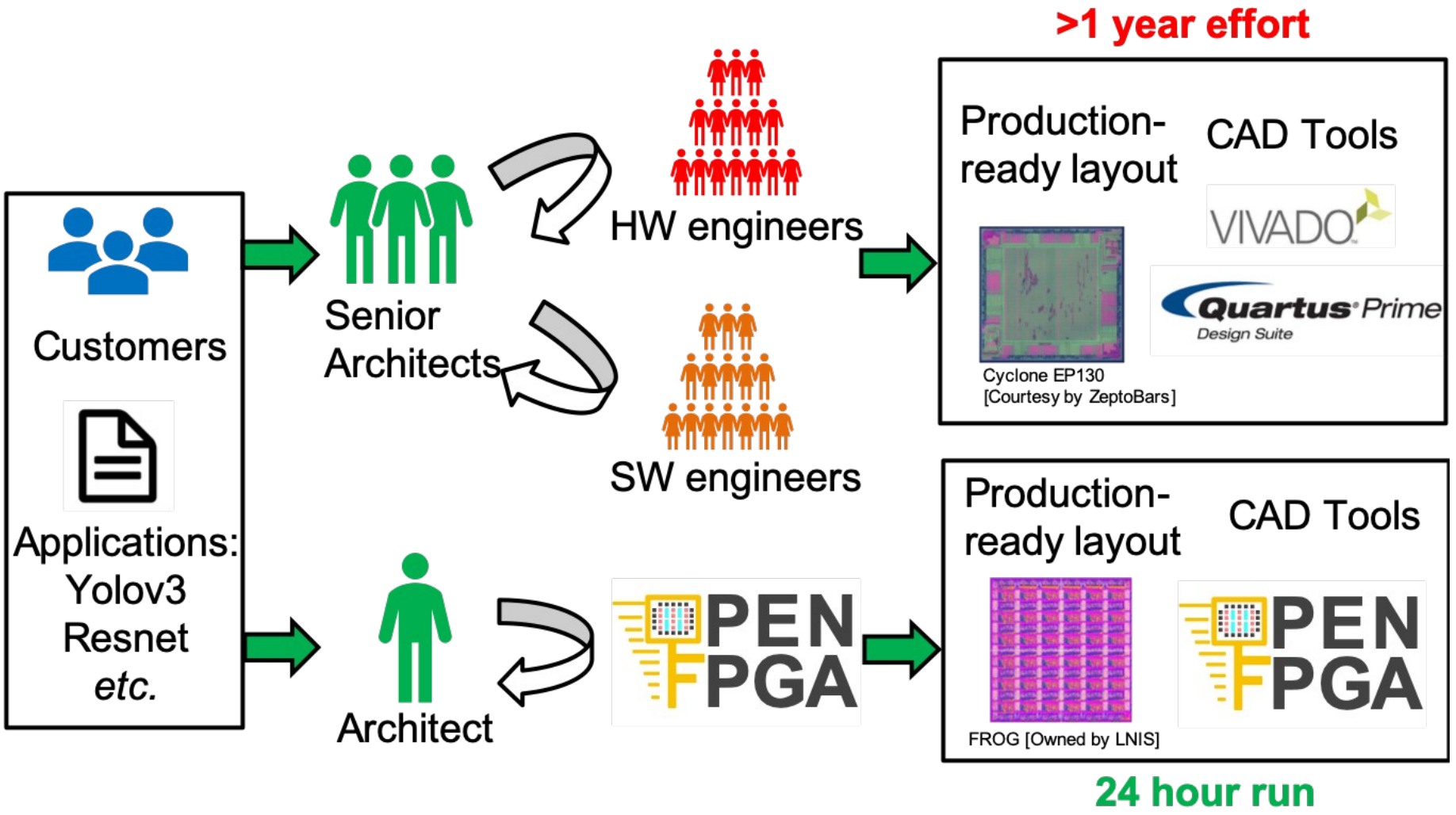
Nanditha Rao

Agenda

- What is OpenFPGA
- Need and advantages
- Verilog-to-routing (VTR)
- Versatile Place and Route (VPR)
- Demo: Run the automated VTR flow

OpenFPGA

- Current methodologies to produce an FPGA involve several hardware and software engineers and development for several months
- How do we improve the design/development times?
- OpenFPGA: Open source framework which can be used to quickly generate a fabric for a custom FPGA (specific to your design)
  - Automation techniques used
  - Reduces FPGA development cycle of a new FPGA to a few days
  - Provides open source design tools

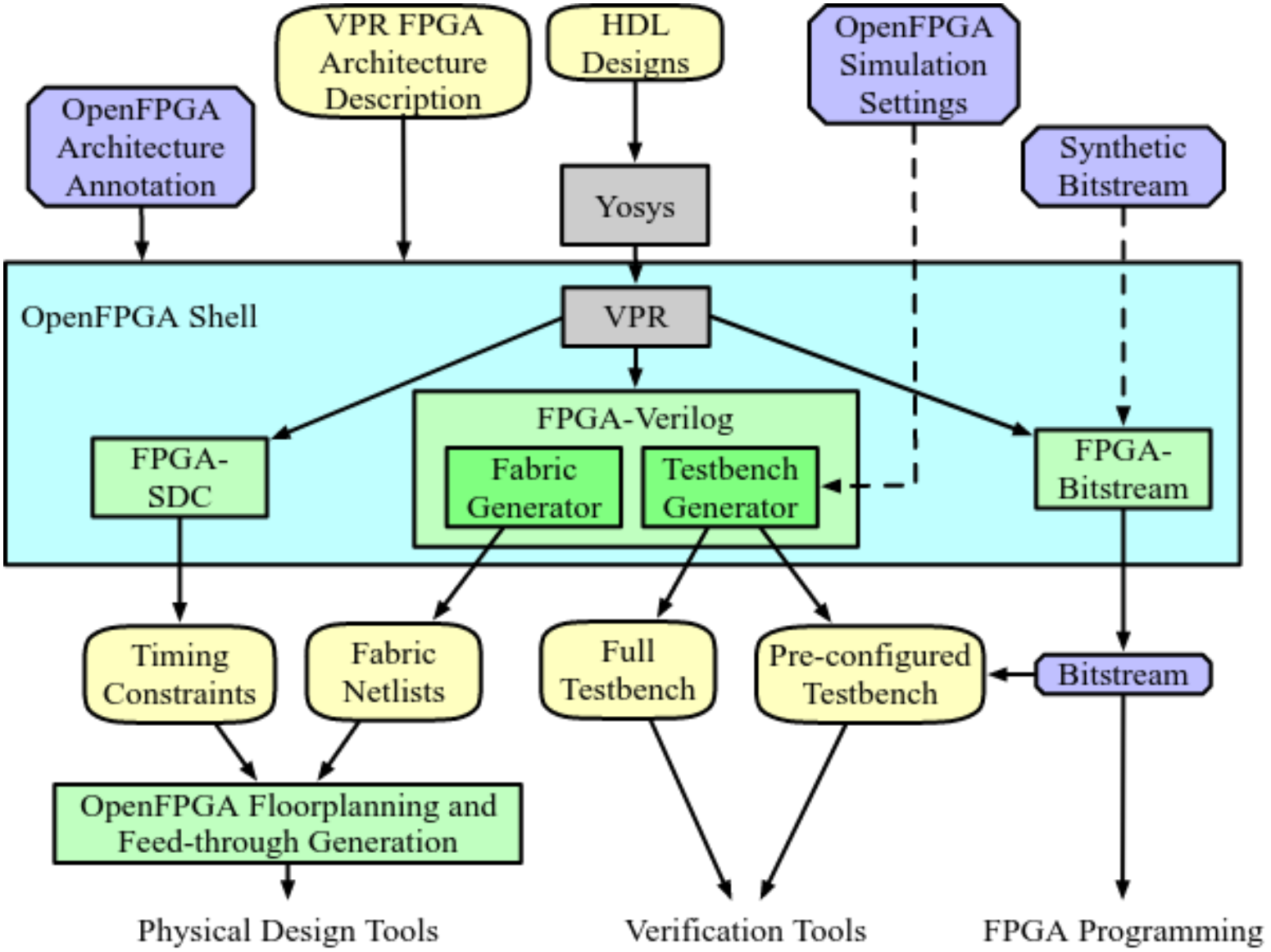


Taken from: https://openfpga.readthedocs.io/en/master/overview/motivation/

OpenFPGA

- Need for custom FPGAs?
  - Accelerate domain-specific applications: FPGA architectures have to be custom made, to provide maximum computing. Prototyping and producing a custom FPGA is costly and time-consuming
- Customise your own FPGA fabric using a set of templates (> 20 FPGA architectures- in xml files optimised for different applications)
- Generates Verilog netlists describing an FPGA fabric based on an XML-based description file: VPR's (Versatile Place and Route) architecture description language
- Allows you to write your own FPGA fabric (for a specific application) using OpenFPGA's architecture description language
- Automatically generates Verilog testbenches to validate the correctness of FPGA fabric
- Bitstream generation support based on the same XML-based description file

FPGA-Verilog, FPGA-SDC, FPGA-Bitstream and FPGA-SPICE

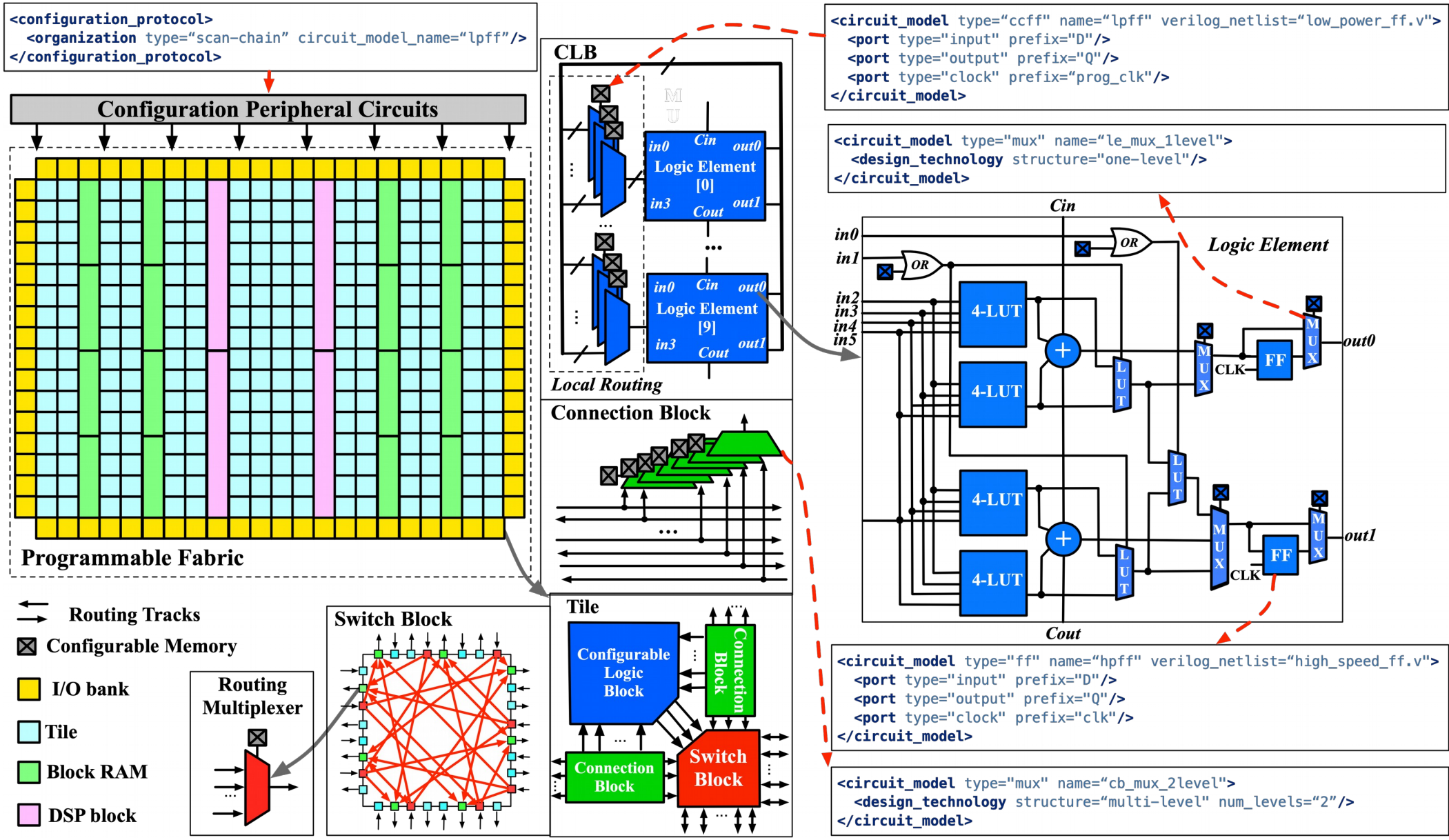


Taken from: https://openfpga.readthedocs.io/en/master/overview/motivation/

Documentation

- Read about OpenFPGA initiative: -->
  https://osfpga.org/https://osfpga.org/
  https://openfpga.readthedocs.io/en/master/ --> git repository:
  https://github.com/os-fpga/open-source-fpga-resource

- First step: To have OpenFPGA installed (done on the cloud).
  https://github.com/lnis-uofu/OpenFPGA#compilation



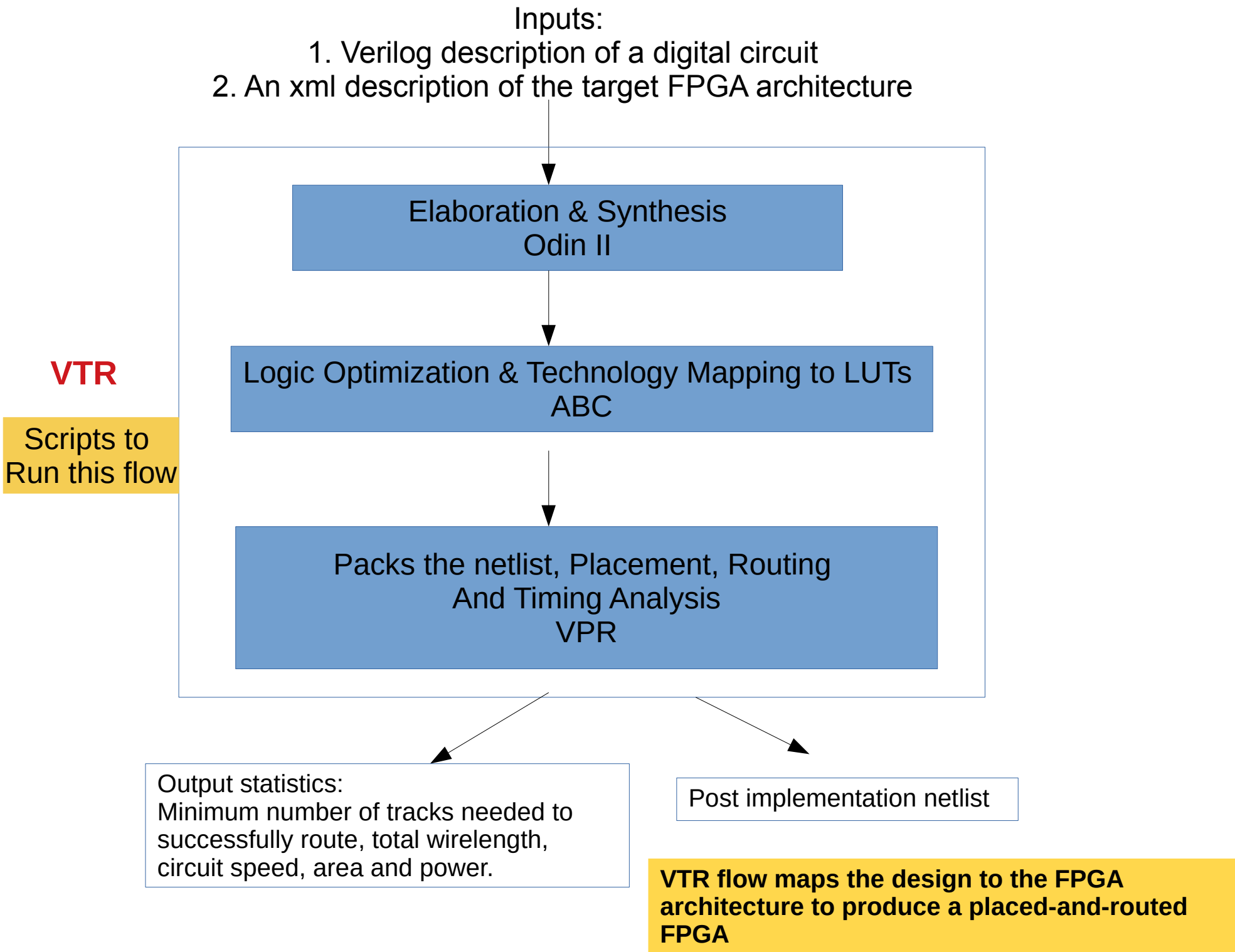
OpenFPGA architecture description enables user to describe a complete FPGA fabric: programmable fabric and the configuration peripherals.

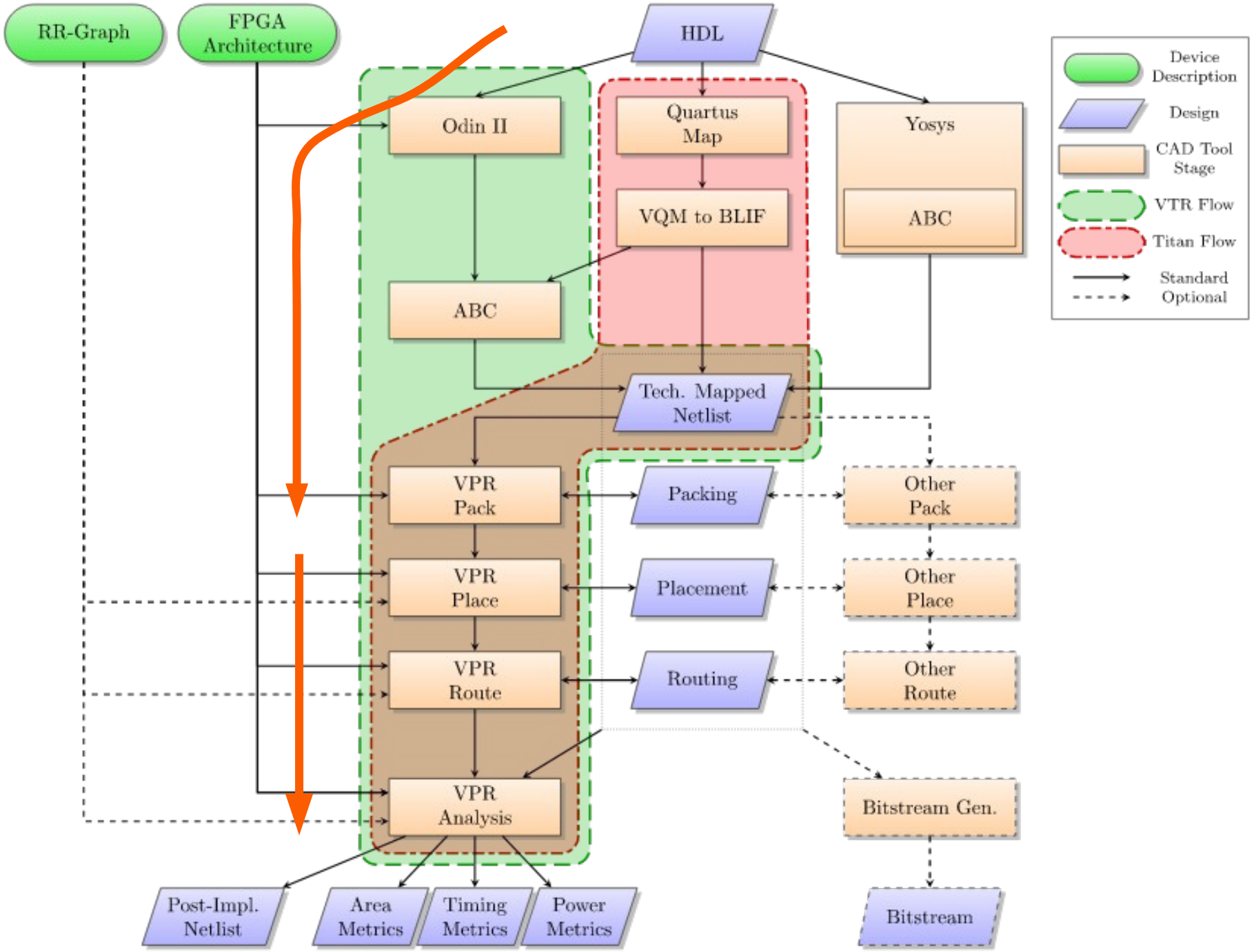
https://openfpga.readthedocs.io/en/master/overview/motivation/

Verilog to Routing (VTR)

- VTR: Uses an XML-based architecture description language to describe the custom FPGA architecture
- VTR Verilog to Routing documentation:
  https://docs.verilogtorouting.org/en/latest/vtr/

  https://docs.verilogtorouting.org/en/latest/quickstart/
- VTR is downloadable from (done on cloud):
  https://github.com/verilog-to-routing/vtr-verilog-to-routing


Inputs:
1. Verilog description of a digital circuit
2. An xml description of the target FPGA architecture
VTR
Scripts to Run this flow
Elaboration & Synthesis
Odin II
Logic Optimization & Technology Mapping to LUTs
ABC
Packs the netlist, Placement, Routing
And Timing Analysis
VPR
Output statistics:
Minimum number of tracks needed to successfully route, total wirelength, circuit speed, area and power.
Post implementation netlist
VTR flow maps the design to the FPGA architecture to produce a placed-and-routed FPGA

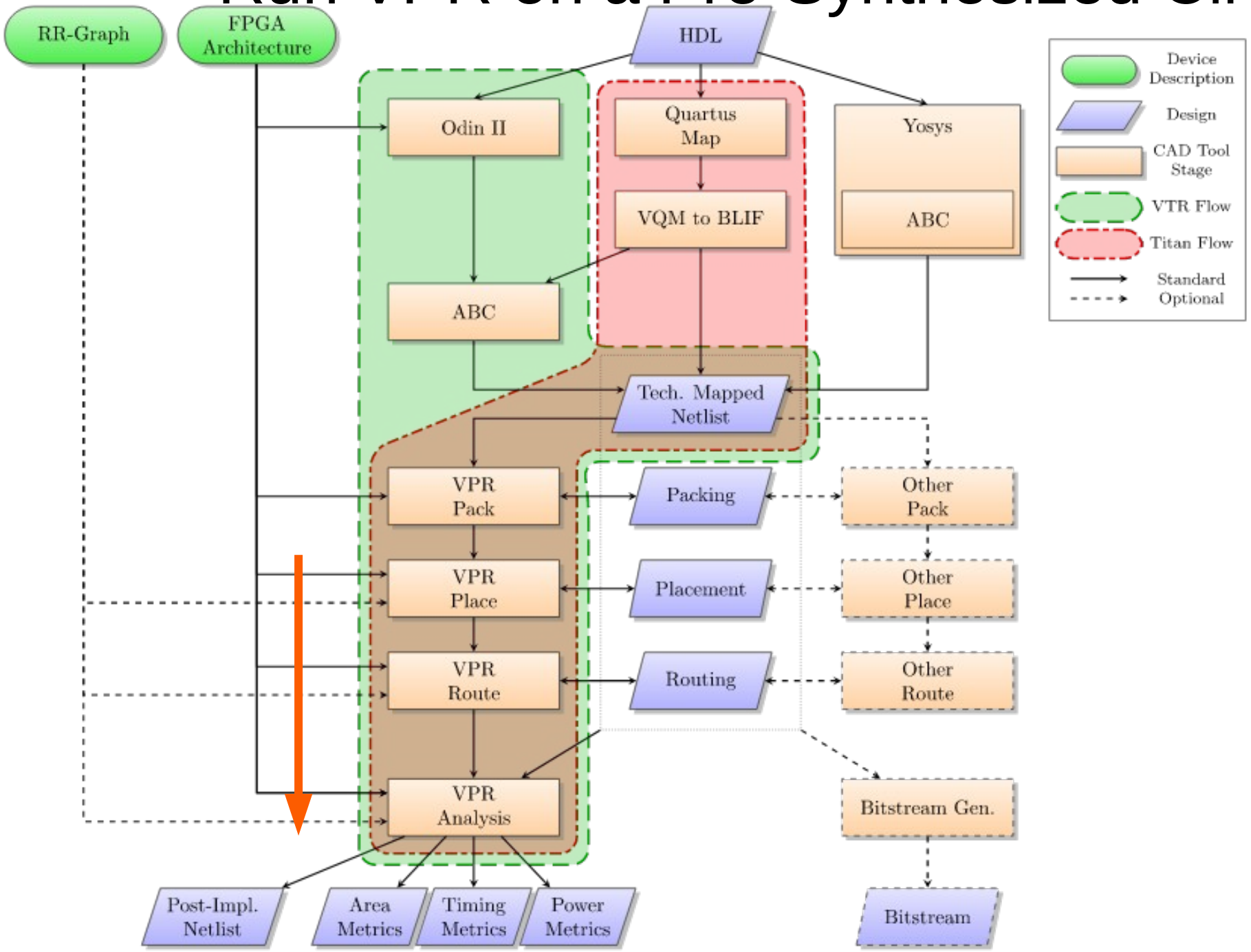


https://docs.verilogtorouting.org/en/latest/vtr/cad_flow/

Running the tool

- VTR:
  - https://docs.verilogtorouting.org/en/latest/quickstart/
- Build OpenFPGA (done on cloud)
- Build VTR (done on cloud)
- Run VPR on a Pre-Synthesized Circuit
  - Observe the result files
  - Visualize (GUI) circuit implementation
- Run the entire VTR flow automatically
  - Implement our own circuit (blink.v and counter.v) on a pre-existing FPGA architecture Earch.xml (VTR_ROOT/vtr_flow/arch)
  - Use an automated approach (Odin II and ABC are automatically run)
  - Perform timing simulation on the generated fabric

Run VPR on a Pre-Synthesized Circuit



https://docs.verilogtorouting.org/en/latest/vtr/cad_flow/

Run VPR on a Pre-Synthesized Circuit

- Run VPR on a Pre-Synthesized Circuit

  https://docs.verilogtorouting.org/en/latest/vpr/

  - Packing (combines primitives into complex blocks)
  - Placement (places complex blocks within the FPGA grid)
  - Routing (determines interconnections between blocks)
  - Analysis (analyzes the implementation)
- Input: Blif file, Earch
- Command:> $VTR_ROOT/vpr/vpr \

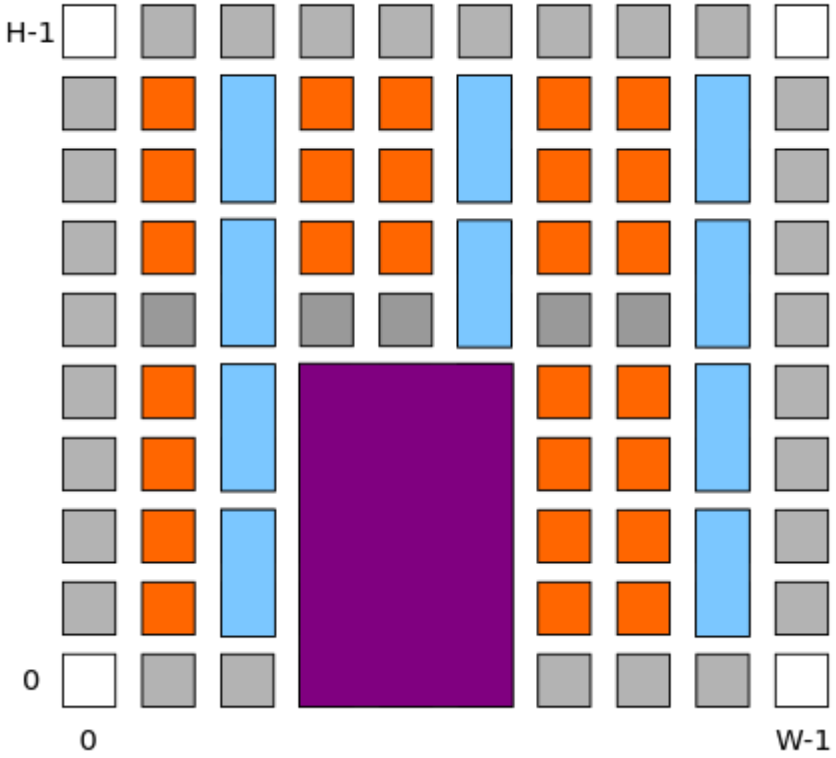
  $VTR_ROOT/vtr_flow/arch/timing/EArch.xml \

  $VTR_ROOT/vtr_flow/benchmarks/blif/tseng.blif \ --route_chan_width 100

Run VPR on a Pre-Synthesized Circuit

- BLIF Netlist (.blif)
  - https://docs.verilogtorouting.org/en/latest/vpr/file_formats/
  - The technology mapped circuit to be implement on the target FPGA is specified as a Berkely Logic Interchange Format (BLIF) netlist.
  - The netlist must be flattened and consist of only primitives (e.g. .names, .latch, .subckt)
  - Clock and delay constraints can be specified with an SDC File.

Earch: Example Xml file explanation




```
<layout>
    <!-- Specifies an auto-scaling square FPGA floorplan -->
    <auto_layout aspect_ratio="1.0">
        <!-- Create I/Os around the device perimeter -->
        <perimeter type="io" priority=10"/>

        <!-- Nothing in the corners -->
        <corners type="EMPTY" priority="100"/>

        <!-- Create a column of RAMs starting at column 2, and
             repeating every 3 columns. Note that a vertical offset (starty)
             of 1 is needed to avoid overlapping the IOs-->
        <col type="RAM" startx="2" repeatx="3" starty="1" priority="3"/>

        <!-- Create a single PCIE block along the bottom, overriding
             I/O and RAM slots -->
        <single type="PCIE" x="3" y="0" priority="20"/>

        <!-- Create an additional row of I/Os just above the PCIE,
             which will not override RAMs -->
        <row type="io" starty="5" priority="2"/>

        <!-- Fill remaining with CLBs -->
        <fill type="CLB" priority="1"/>
    </auto_layout>
</layout>
```

Earch: https://docs.verilogtorouting.org/en/latest/arch/

https://docs.verilogtorouting.org/en/latest/arch/reference/#arch-grid-layout

Run VPR on a Pre-Synthesized Circuit

- Outputs:
- .net file: The circuit .net file is an xml file that describes a post-packed user circuit. It represents the user netlist in terms of the complex logic blocks of the target architecture. This file is generated from the packing stage and used as input to the placement stage in VPR.

VTR

- To check installation:
  - $VTR_ROOT/vtr_flow/scripts/ run_vtr_task.py basic_flow
  - Expected output:
    - k6_N10_memSize16384_memData64_40 nm_timing/ch_intrinsics...OK

- Counter:
- $VTR_ROOT/vtr_flow/scripts/run_vtr_flow.pl \
- $VTR_ROOT/doc/src/quickstart/counter.v \
- $VTR_ROOT/vtr_flow/arch/timing/EArch.xml \
- -temp_dir . \
- --route_chan_width 100
-
- - To open GUI for above design
- $VTR_ROOT/vpr/vpr \
- $VTR_ROOT/vtr_flow/arch/timing/EArch.xml \
- counter --circuit_file counter.pre-vpr.blif \
- --route_chan_width 100 \
- --analysis --disp on

- Post impl netlist timing simulation
- Run VPR:
- We also need to provide the vpr --gen_post_synthesis_netlist option to generate the post-implementation netlist and dump the timing information in Standard Delay Format (SDF):
-
- $VTR_ROOT/vpr/vpr $VTR_ROOT/vtr_flow/arch/timing/EArch.xml counter.pre-vpr.blif --gen_post_synthesis_netlist on

VTR

- VTR can also extract useful information and statistics from executions of the flow such as area, speed tool execution time etc.
- For single benchmarks parse_vtr_flow extrastics statistics from a single execution of the flow.

Counter- power, timing, area

- Logic Element (fle) detailed count:
- Total number of Logic Elements used : 6
- LEs used for logic and registers : 3
- LEs used for logic only : 3
- LEs used for registers only : 0

  FPGA sized to 3 x 3 (auto)
- Device Utilization: 0.21 (target 1.00)
- Block Utilization: 0.22 Type: io
- Block Utilization: 1.00 Type: clb

Counter- area

Block Type	# Blocks	Avg. # of input clocks and pins used	Avg. # of output pins used
EMPTY	0	0	0
io	7	0.571429	0.428571
clb	1	4	4
mult_36	0	0	0
memory	0	0	0

Absorbed logical nets 14 out of 21 nets, 7 nets not absorbed.

Counter area

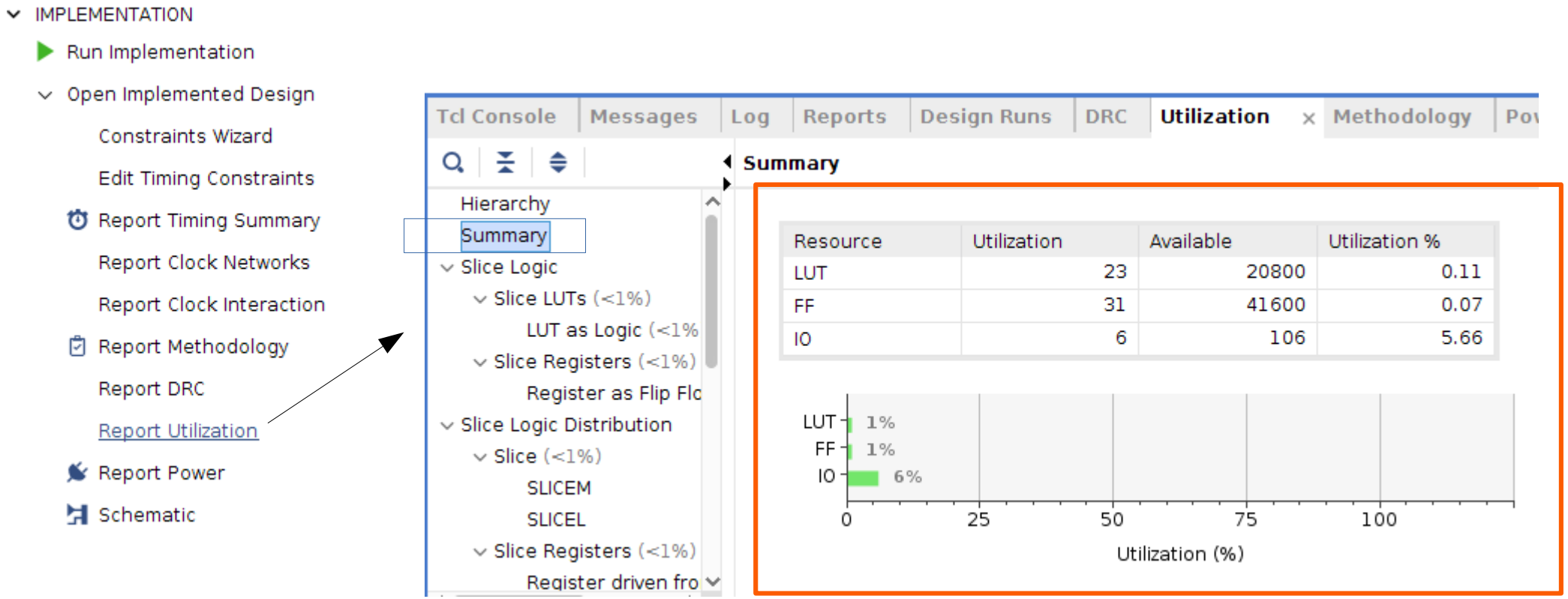
```
Resource usage...
        Netlist
                7       blocks of type: io
        Architecture
                32      blocks of type: io
        Netlist
                1       blocks of type: clb
        Architecture
                1       blocks of type: clb
        Netlist
                0       blocks of type: mult_36
        Architecture
                0       blocks of type: mult_36
        Netlist
                0       blocks of type: memory
        Architecture
                0       blocks of type: memory

Device Utilization: 0.21 (target 1.00)
        Physical Tile io:
        Block Utilization: 0.22 Logical Block: io
        Physical Tile clb:
        Block Utilization: 1.00 Logical Block: clb
```

Counter- power

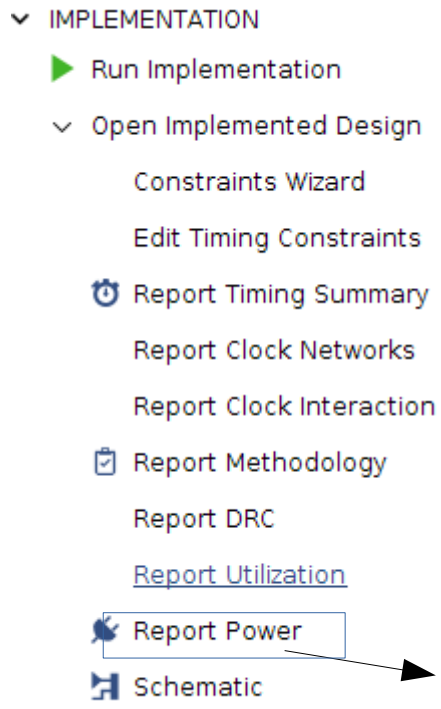
counter.power

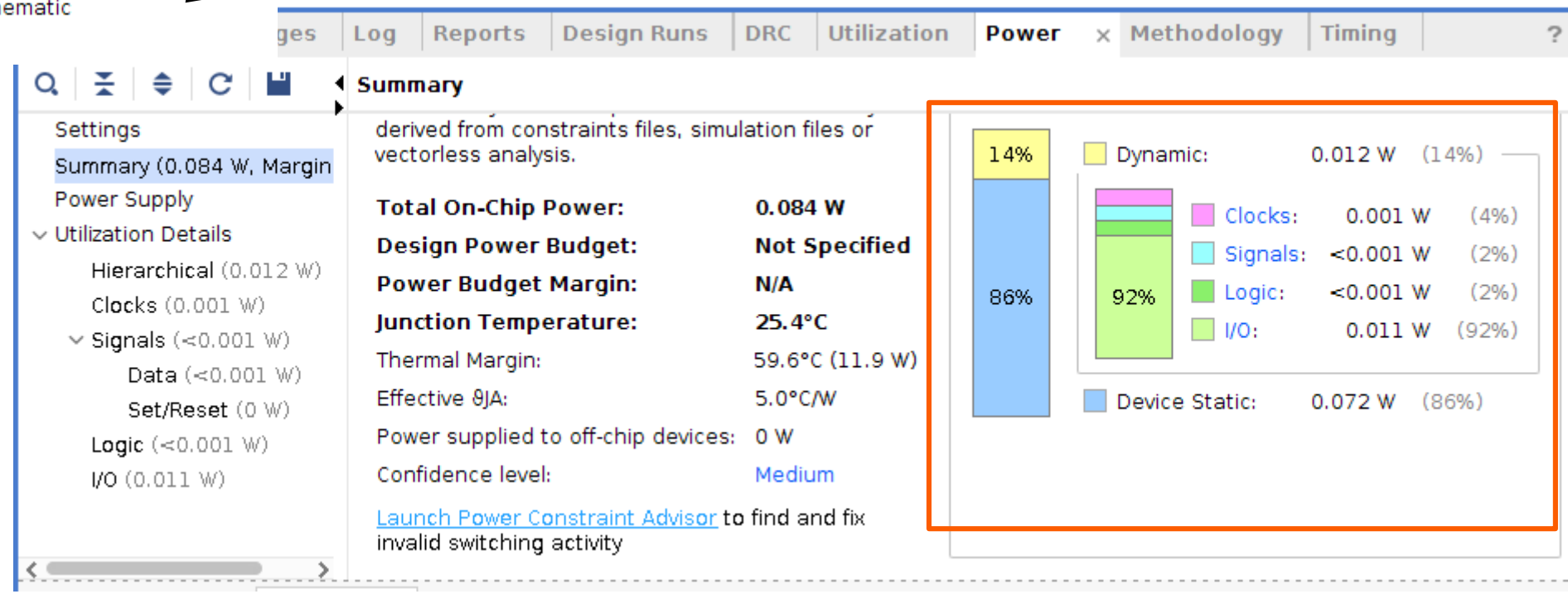
```
-------------------------------- Power Breakdown --------------------------------
Component                        Power (W)   %-Total     %-Dynamic   Method

Total                            0.0003212   1           0.7689
 Routing                         0.0001066   0.3319      0.4223
  Switch Box                     9.697e-05   0.3019      0.3837
  Connection Box                 5.033e-06   0.01567     0.6368
  Global Wires                   4.617e-06   0.01437     1
 PB Types                        8.287e-05   0.258       0.8582
  Primitives                     6.158e-05   0.1917      0.9314
  Interc Structures              7.248e-06   0.02256     0.6487
  Buffers and Wires              1.404e-05   0.0437      0.6451
  Other Estimation Methods       0           0           -nan
 Clock                           0.0001317   0.4101      0.9932
```

Counter area Basys3- Vivado



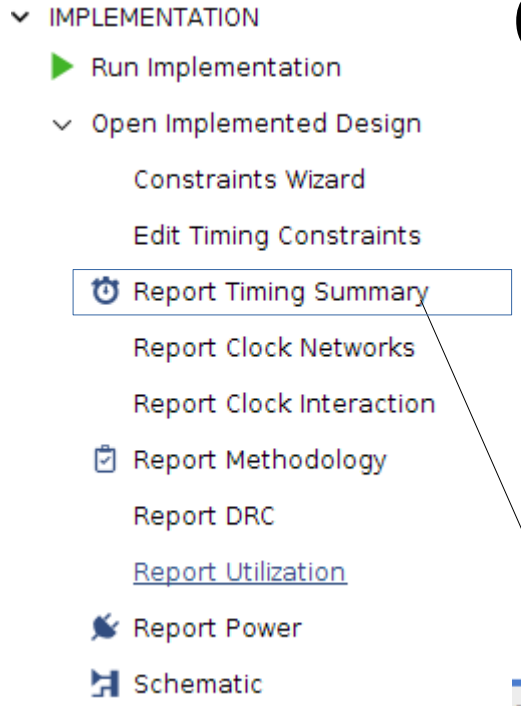
nter power Basys3 - Vivado

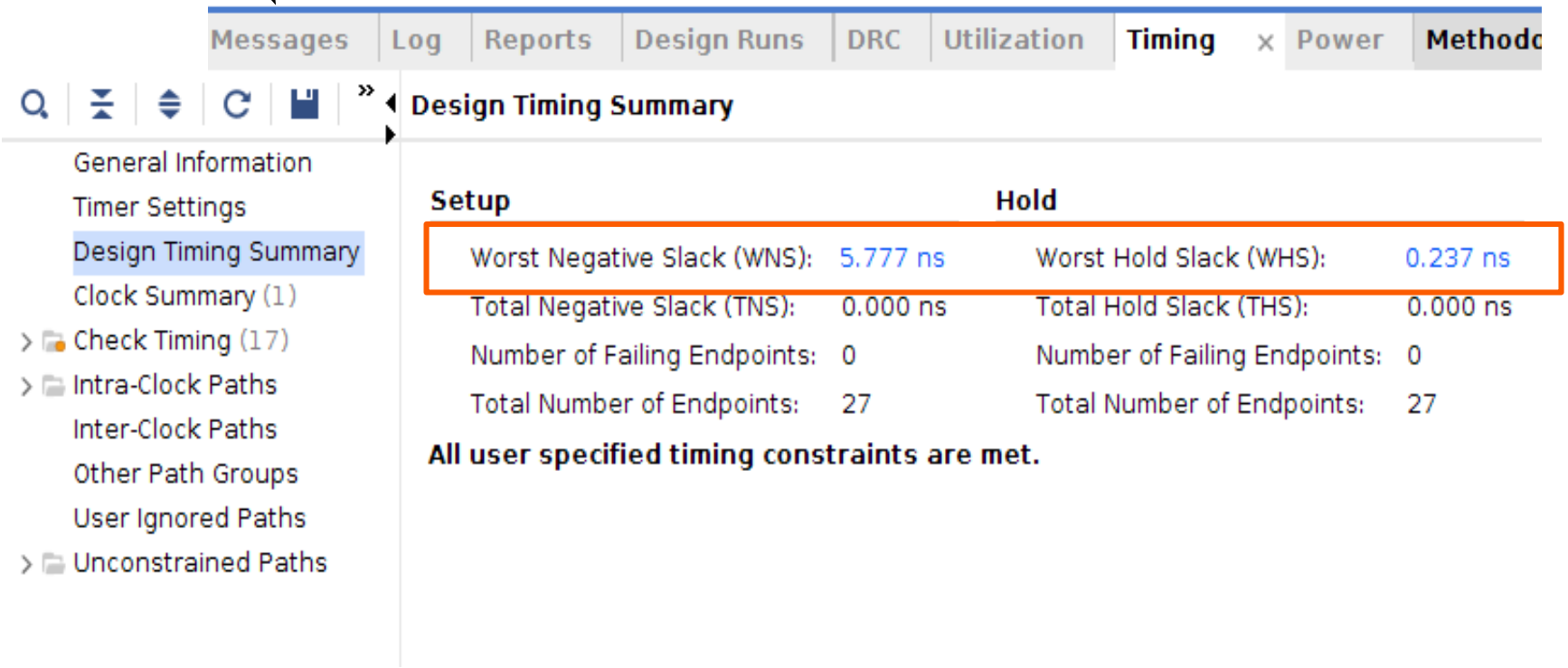




Counter- timing Basys3

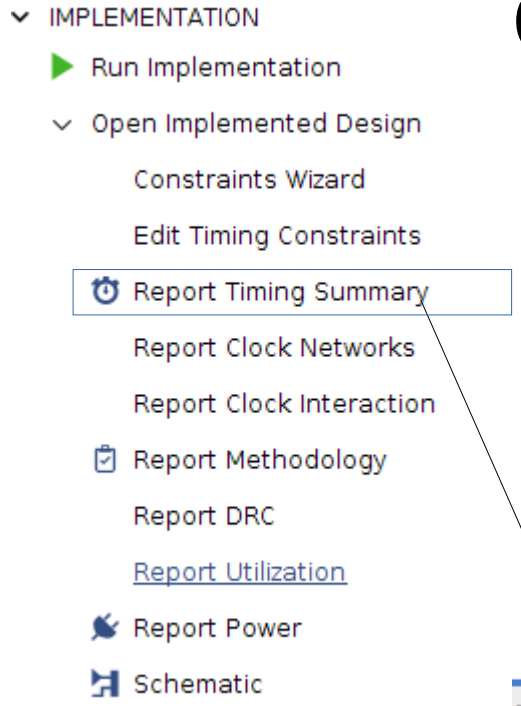
Vivado

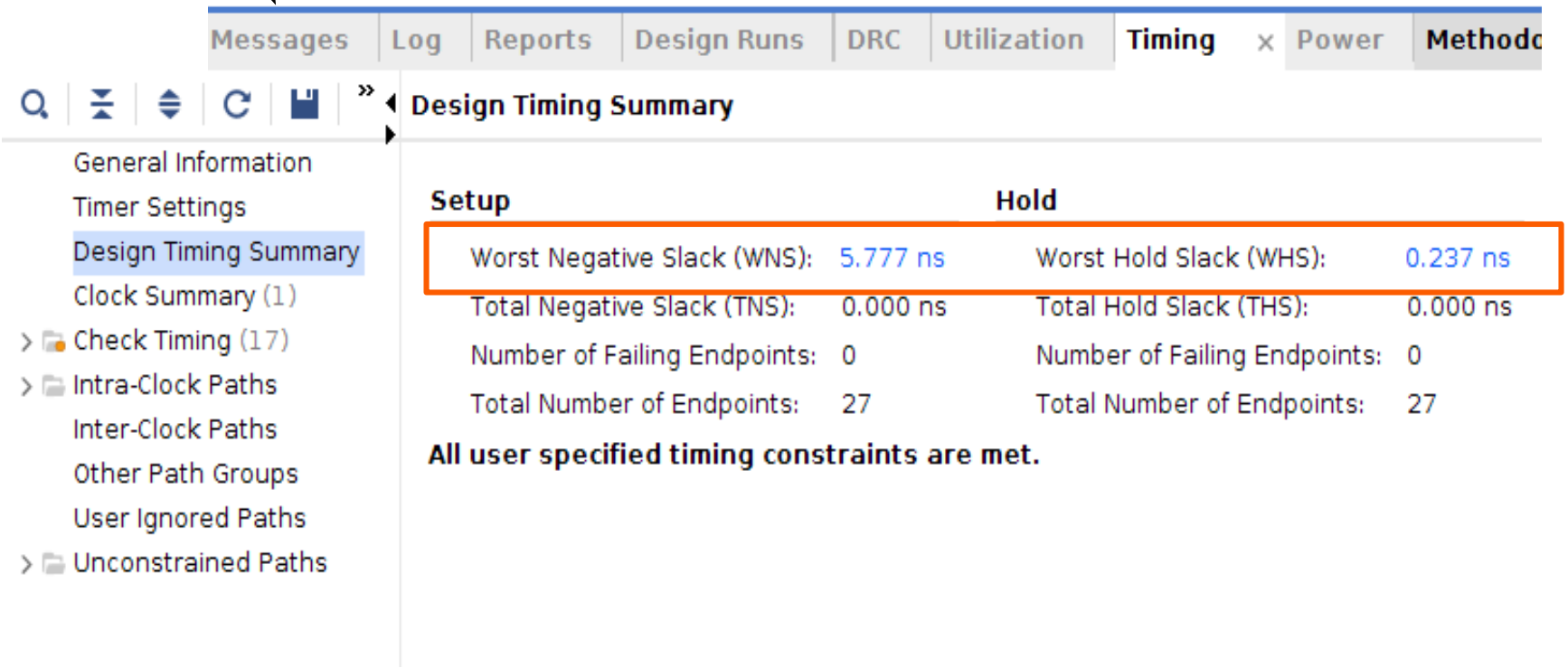




Counter- timing Basys3

Vivado





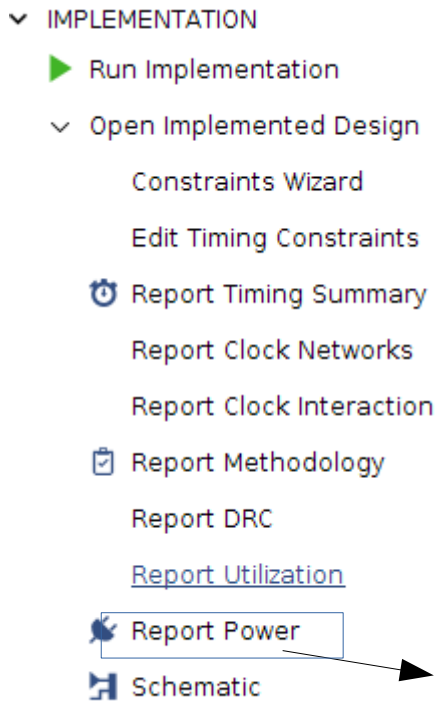
Boolean

- Redo all steps with VIO, and showing remote connection
- Power, area and timing analysis with Boolean

Counter area Boolean- Vivado

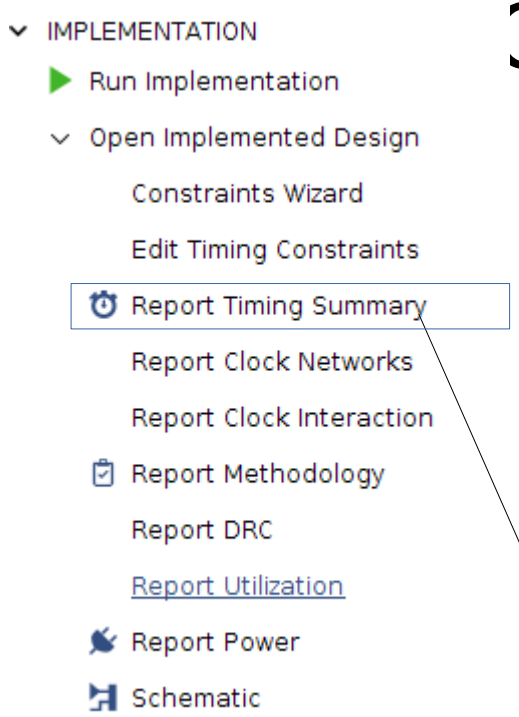
- Area includes VIO’s area

power Boolean- Vivado



Counter- timing Boolean

Vivado



Counter- timing Basys3

Vivado