

Build Your Own Domain-specific Solutions with **RapidWright**

The background features a dark blue gradient with a subtle grid pattern. Overlaid are several large, stylized arrows: a red chevron shape pointing right, a grey arrow pointing right, and a grey triangle pointing down.

Chris Lavin and Alireza Kaviani
Xilinx Research Labs
2/24/19



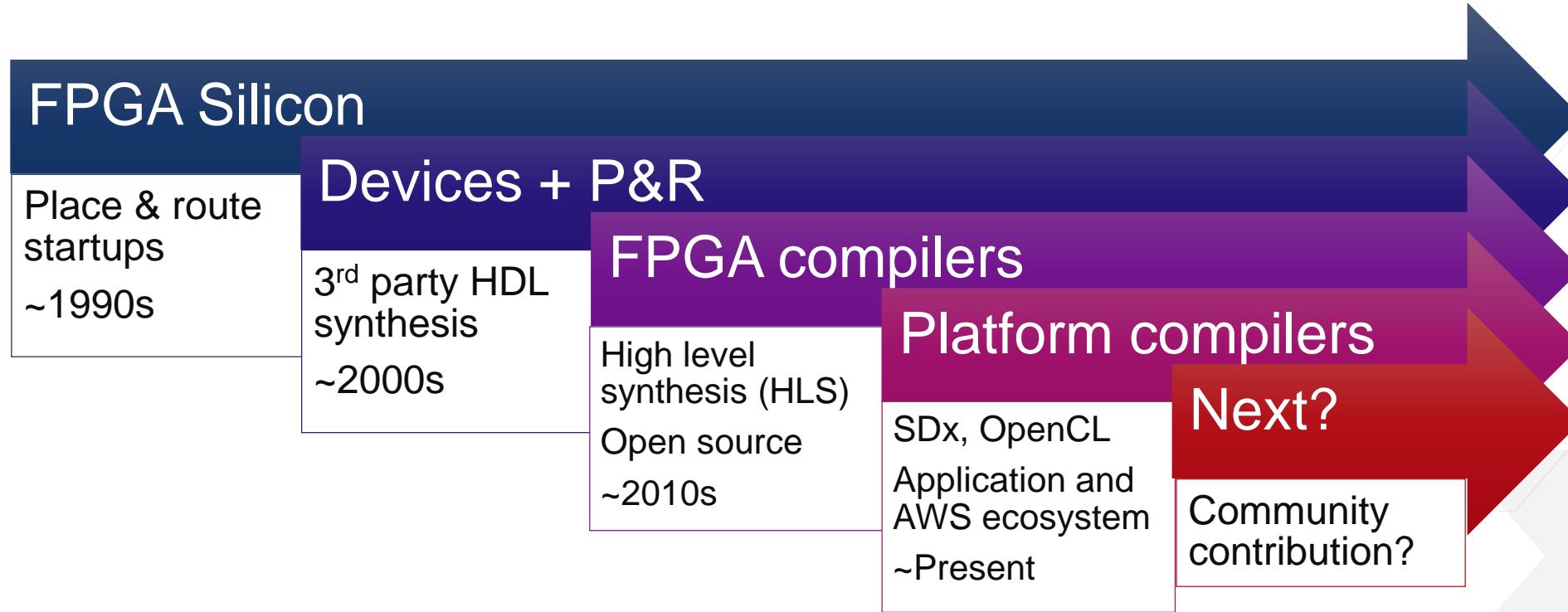
> Why are Domain-specific solutions important?

- >> RapidWright value proposition
- >> Why open source?

> What is RapidWright?

> How to use RapidWright?

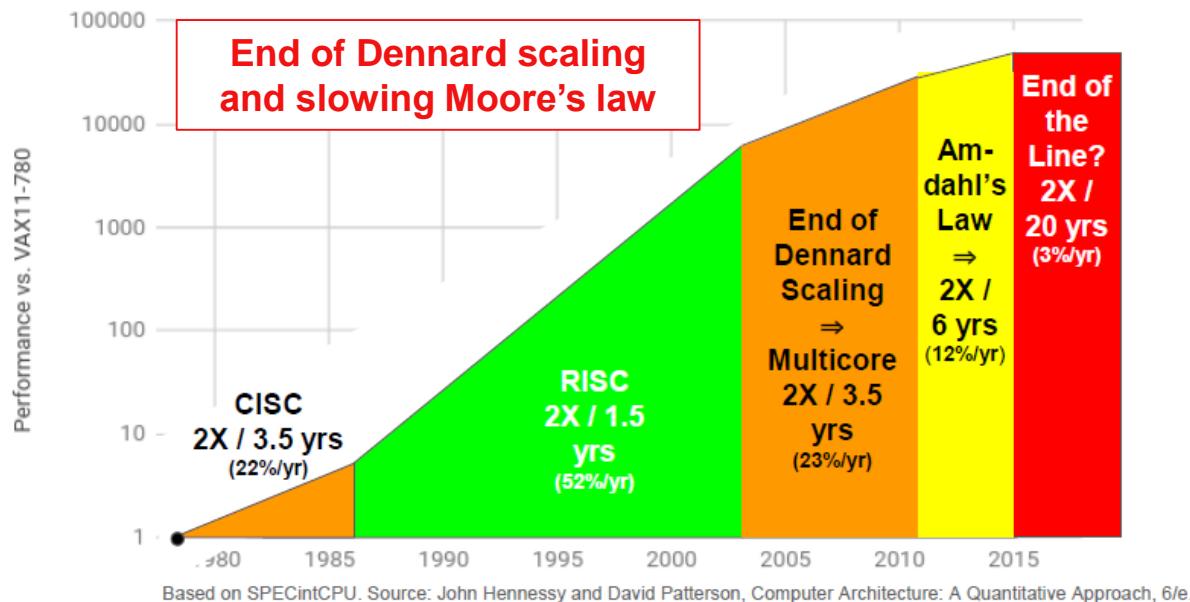
FPGA Industry and Community Dynamics



- > Continuous industry and community engagement

The Age of Domain Specific Architectures

40 years of Processor Performance



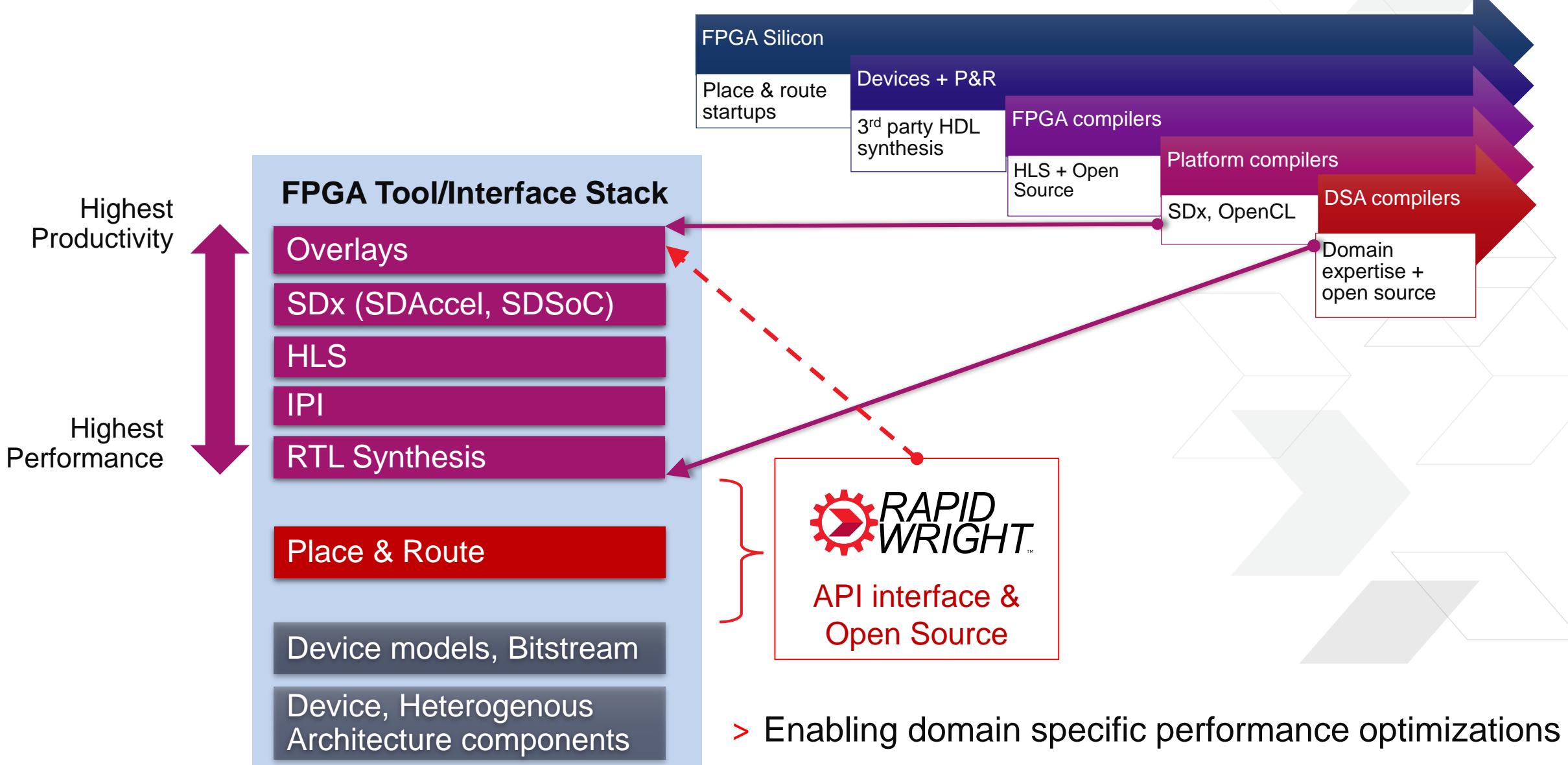
Matrix Multiply Speedup Over Native Python



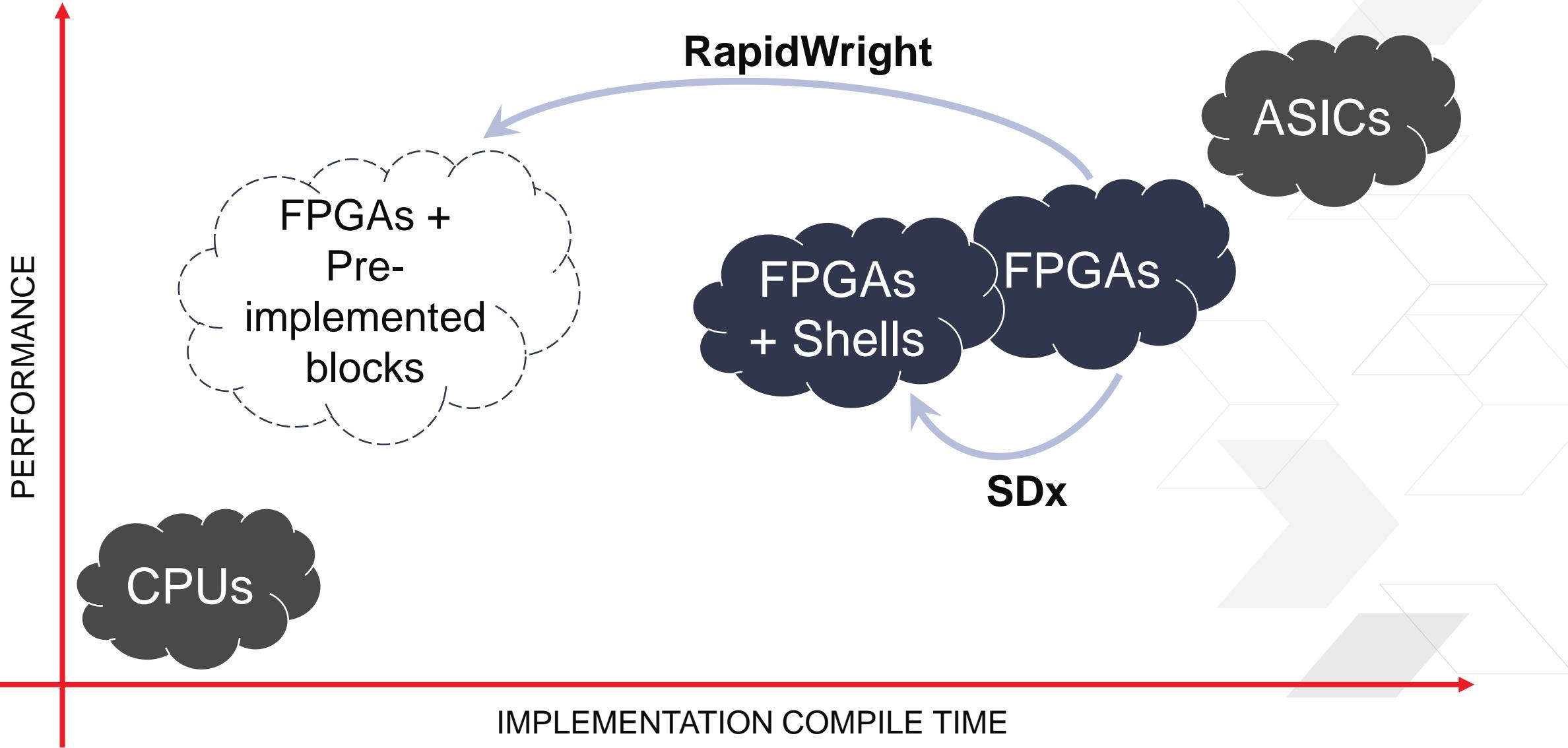
- > Achieve higher efficiency by tailoring the architecture to characteristics of the domain
 - > More effective parallelism for a specific domain, More effective use of memory bandwidth
 - > Domain specific programming language

Source: A New Golden Age for Computer Architecture: (Domain-Specific Hardware/Software Co-Design)
John Hennessy and David Patterson
Stanford and UC Berkeley, 13 June 2018

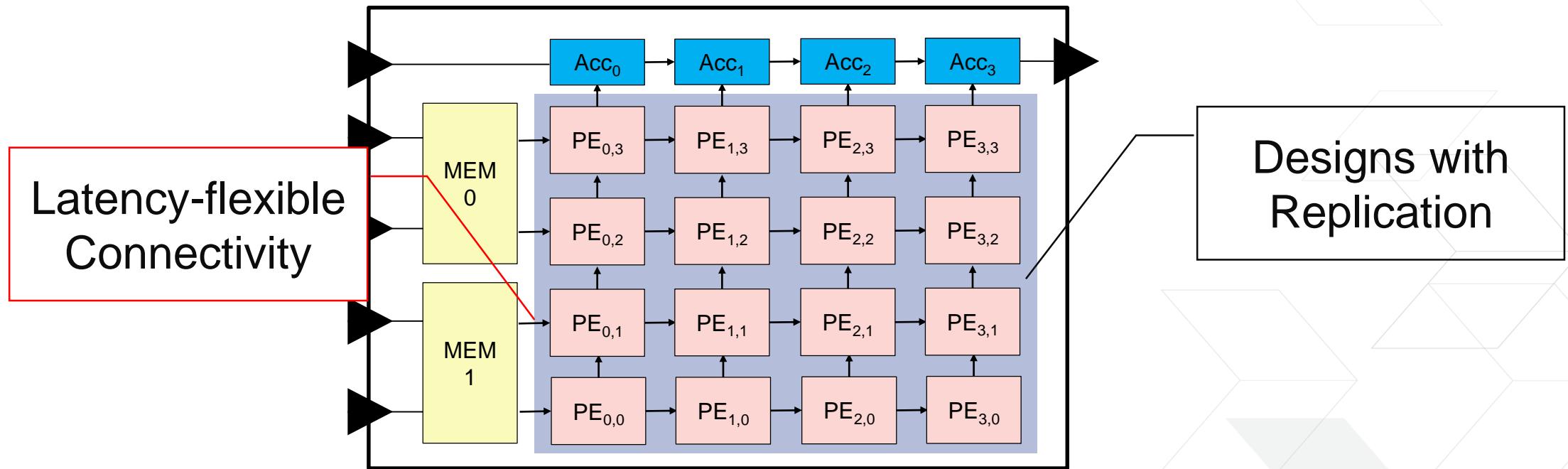
Raising the Abstraction of Design Entry



RapidWright Value Proposition



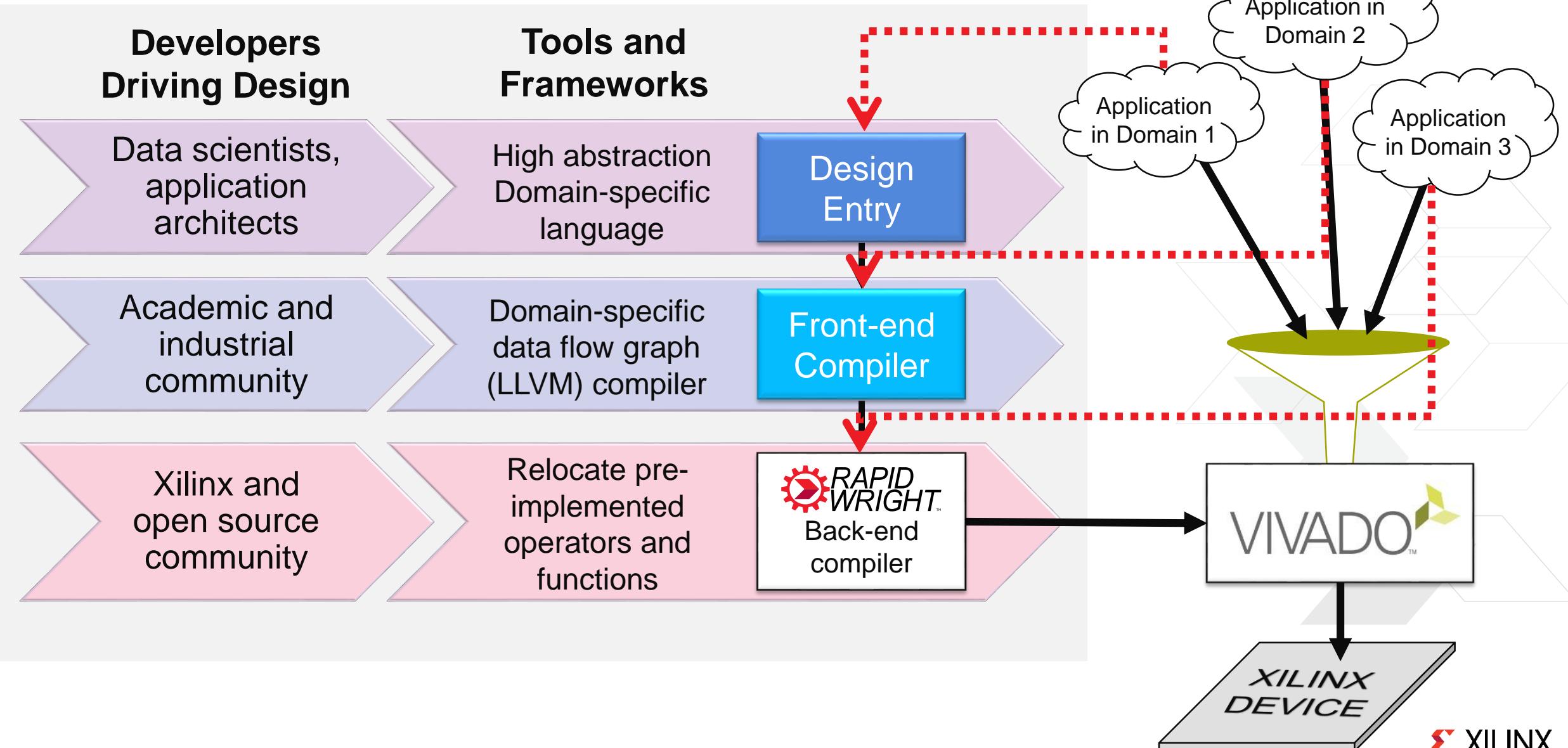
Focus on Emerging Applications



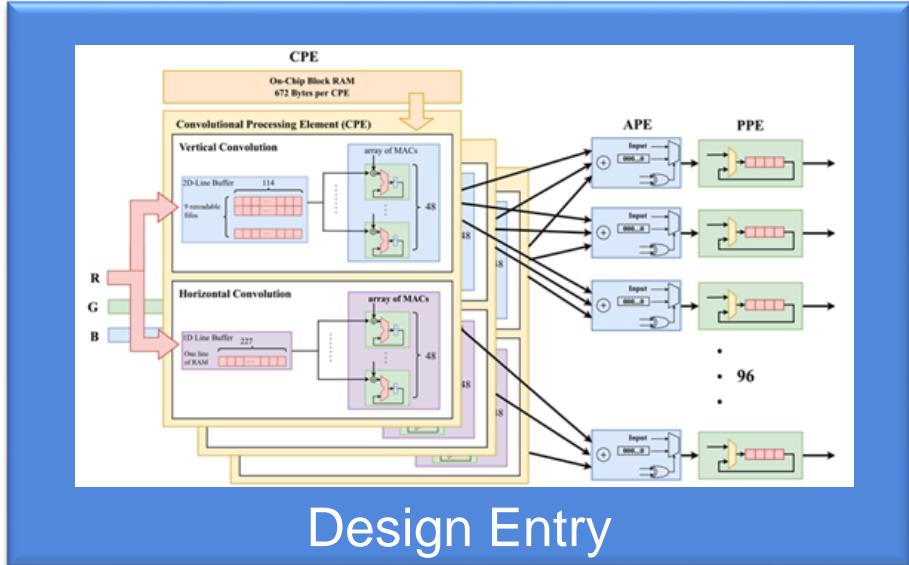
- > Module-based approach to implementation
 - >> Lock-in performance with reusable modules
 - >> Fewer inter-block timing closure issues

- > Goals
 - >> Productivity
 - Order of magnitude reduction in compile time per domain
 - >> Performance (near-spec)
 - >> Predictable timing closure

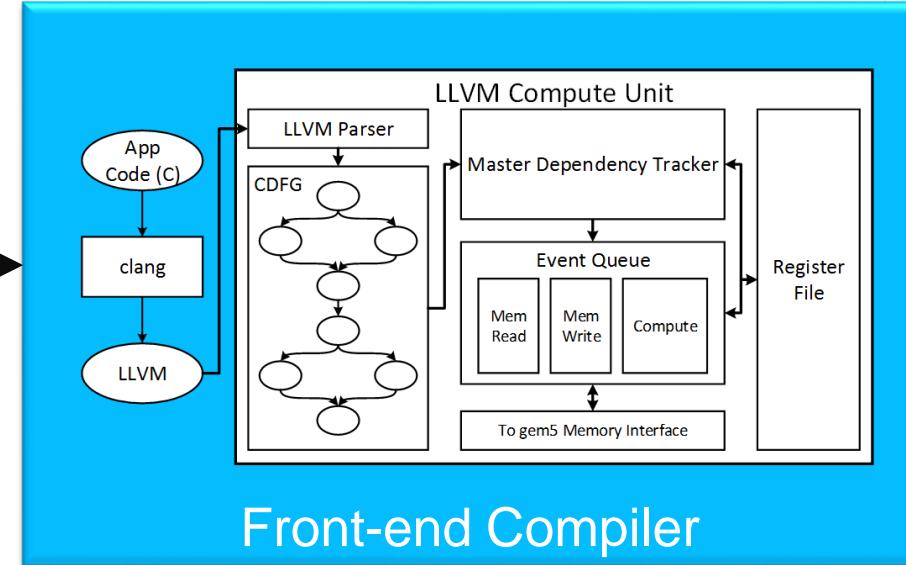
Proposed Domain-specific Tool Flows



Domain Tool Flow Example

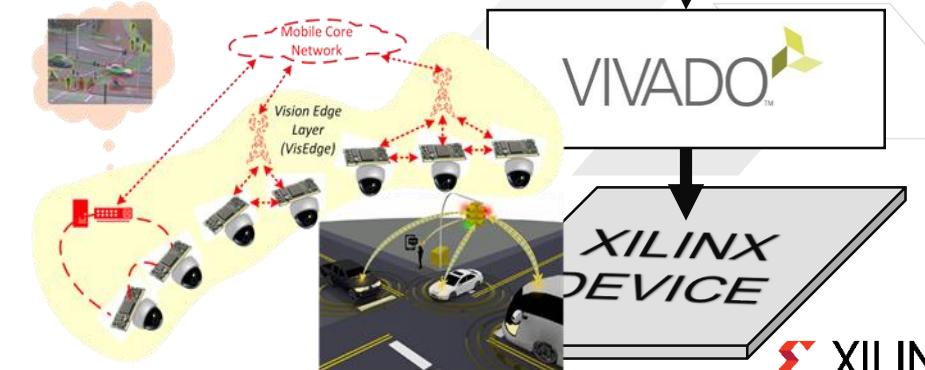


Design Entry

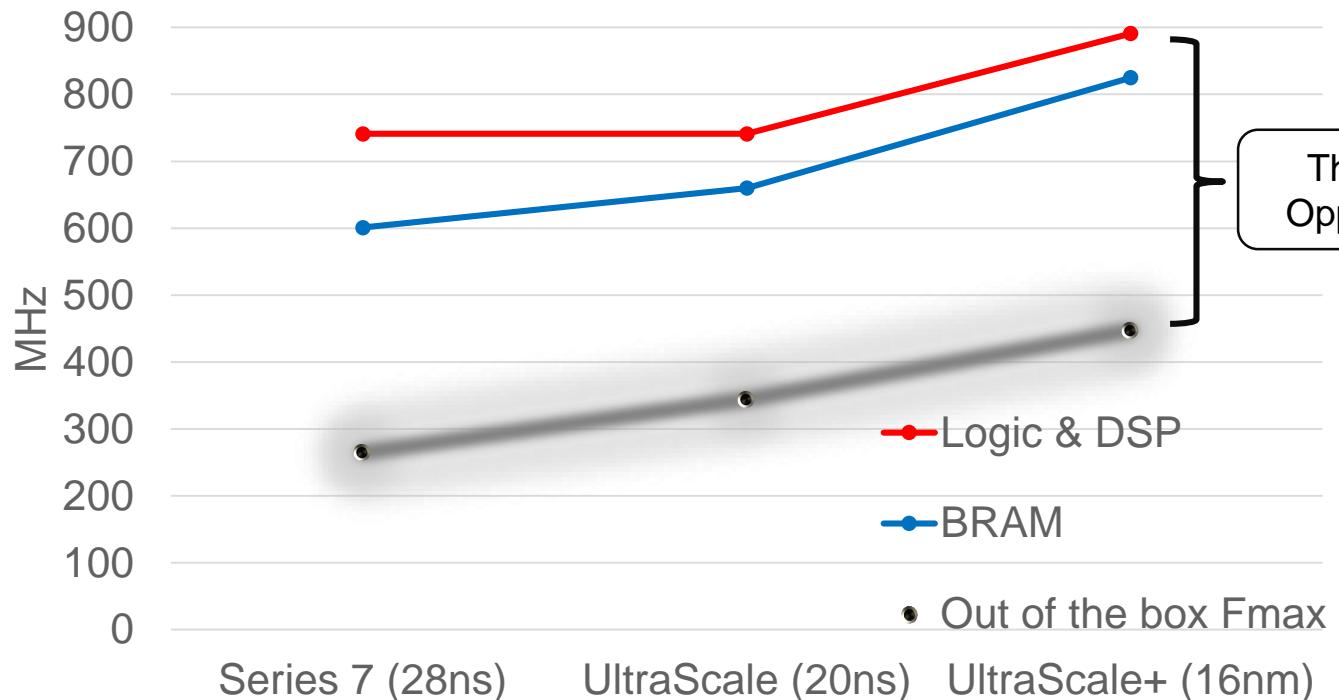


Front-end Compiler

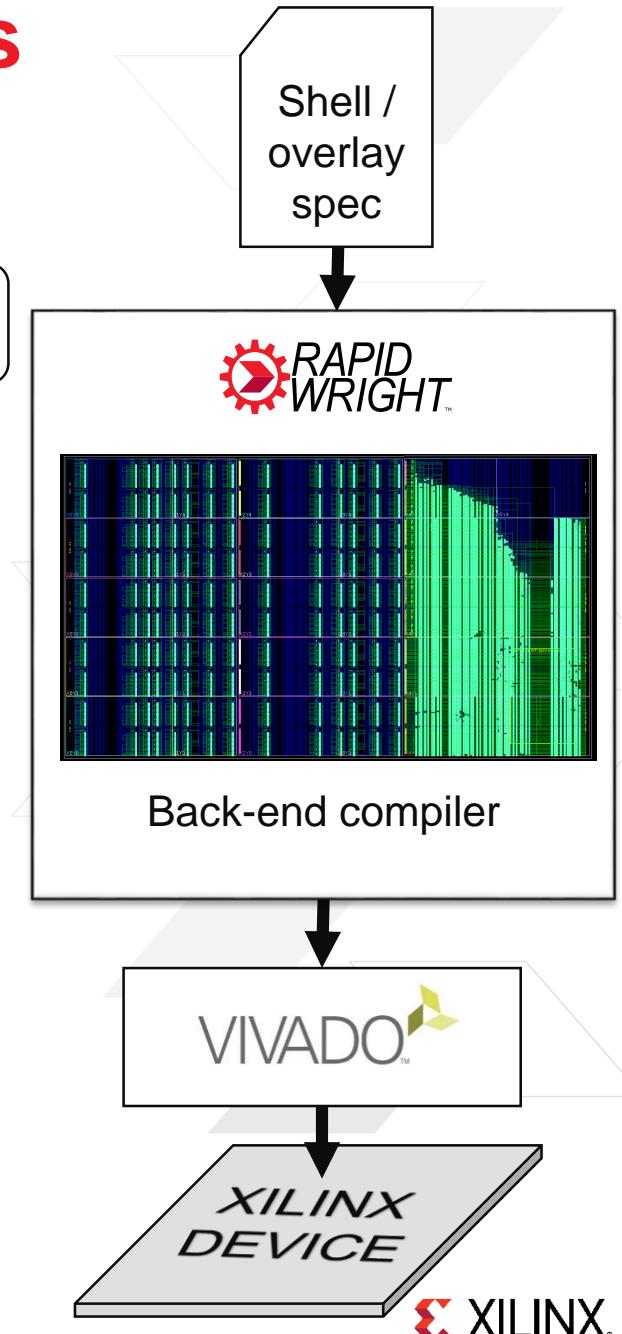
- > Fact
 - >> Emerging domains such as surveillance or vision have high replication
- > Community role
 - >> Identify and extract operators and functions in the domain
- > RapidWright value proposition
 - >> Assemble relocatable pre-implemented domain operators
 - >> Deliver the best inference/watt



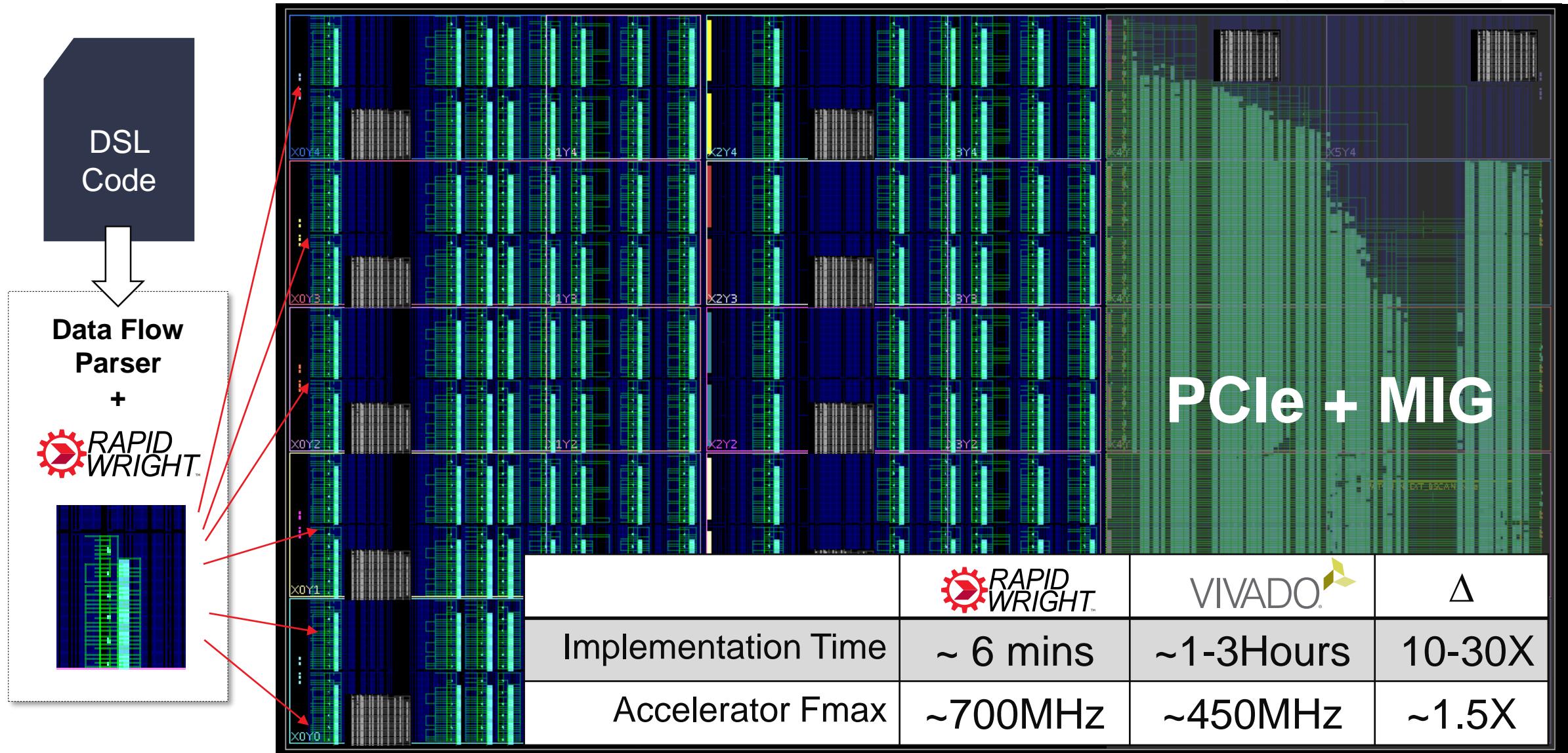
Building Relocatable Domain-specific Shells



- > Fact
 - >> Advances in silicon have created QoR opportunity
- > Community role
 - >> Domain-specific shell design or overlays
- > RapidWright value proposition
 - >> Achieve near-spec performance



Success Scenario: Rapid Domain-specific Assembly

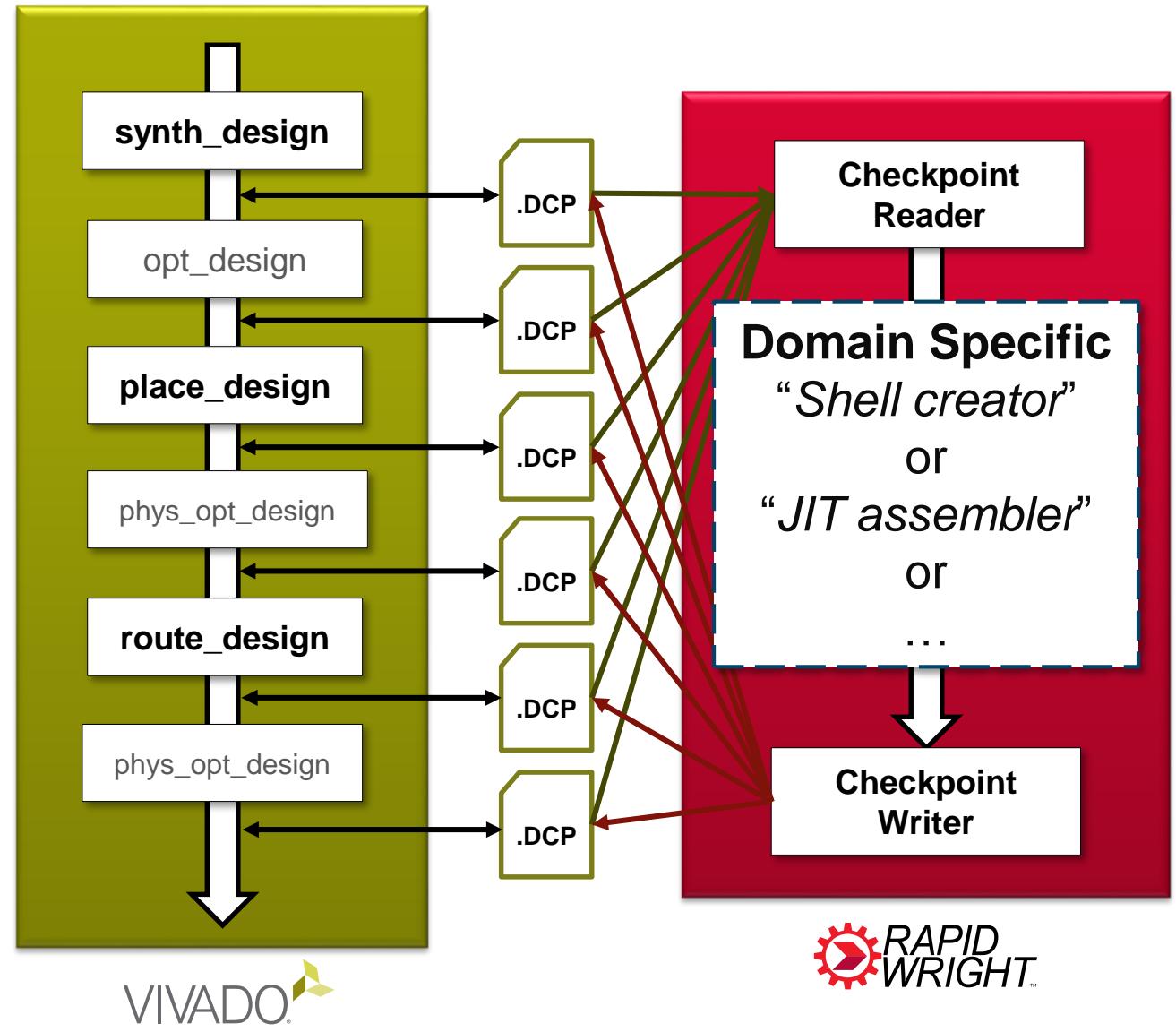


What is RapidWright?



RapidWright Overview

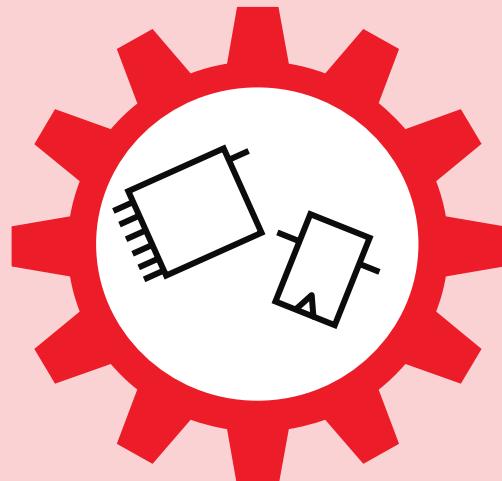
- > Companion framework for Vivado
 - >> Fast, light-weight, open source
 - >> Communicates through Design CheckPoints¹ (DCPs)
 - >> Java code, Python scripting
- > Enables targeted solutions
 - >> Reuse & relocate pre-implemented modules
 - >> Just-in-time implementations
 - >> Create shells & overlays
- > Power user ecosystem
 - >> Academic algorithm validation
 - >> Rapid prototyping of CAD concepts



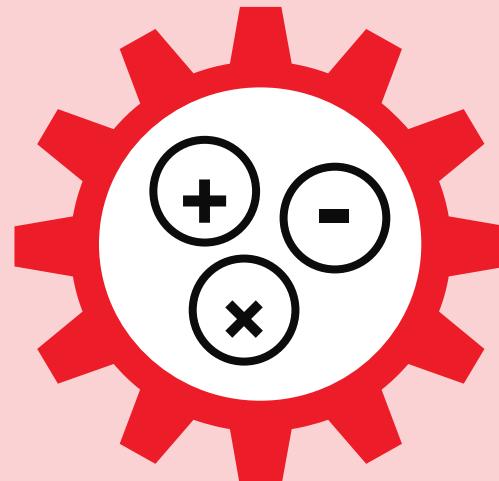
¹DCP = netlist + P&R data + constraints

4 Ways to Design in RapidWright

BUILD ROUTED CIRCUITS



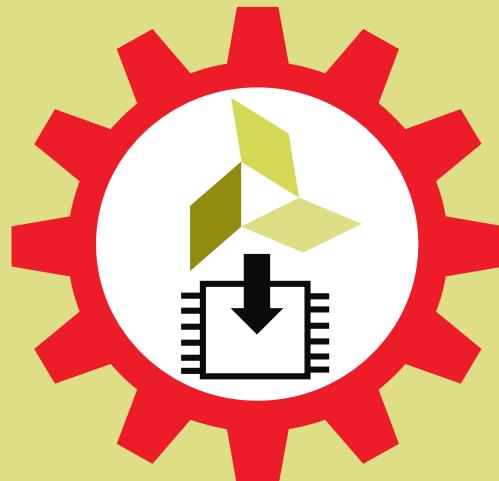
FROM SCRATCH



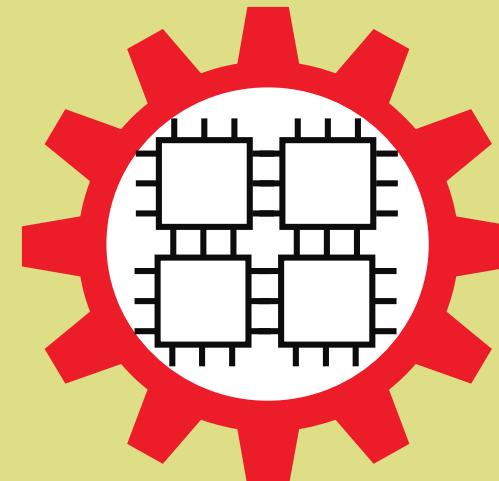
GENERATORS

- > Well-defined circuits in seconds
- > Parameterizable library of generators

REUSE P&R CIRCUITS



FROM VIVADO



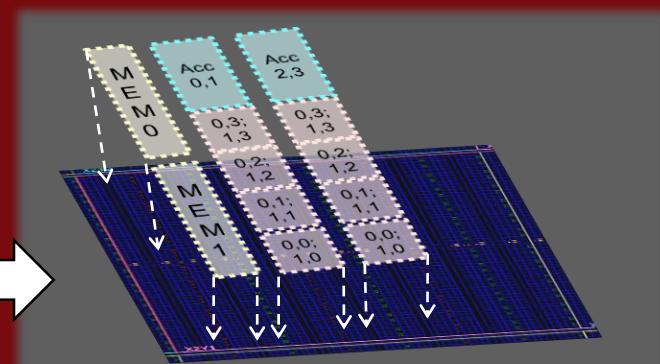
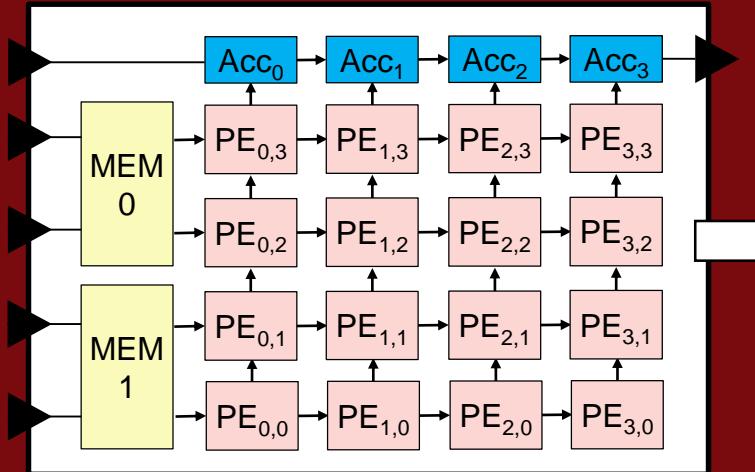
SHELLS & OVERLAYS

- > Reuse/relocate P&R circuits from Vivado
- > Combine P&R circuits together

A Modular Pre-implemented Methodology

USER TASKS (MANUAL)

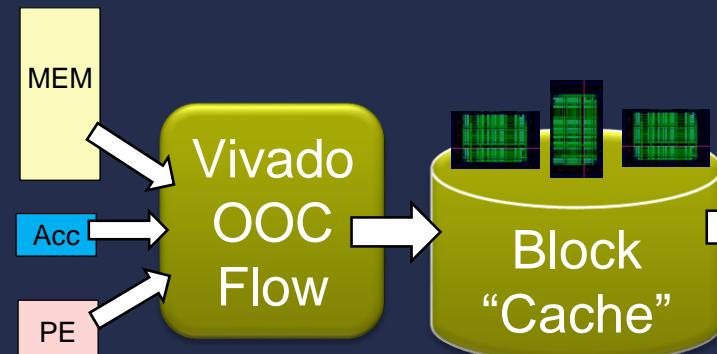
1. Design selection attributes:
 - Modular
 - Latency tolerant
 - Prefers replication
2. Placement planning



Match Design Structure to
Architecture Patterns

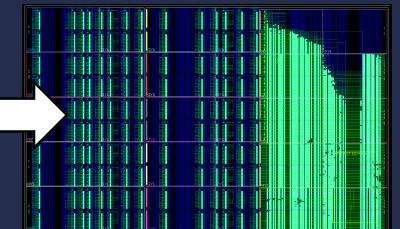
TOOL TASKS (AUTOMATED)

3. P&R modules cached:
 - Relocatable
 - Reusable
 - Timing predictable

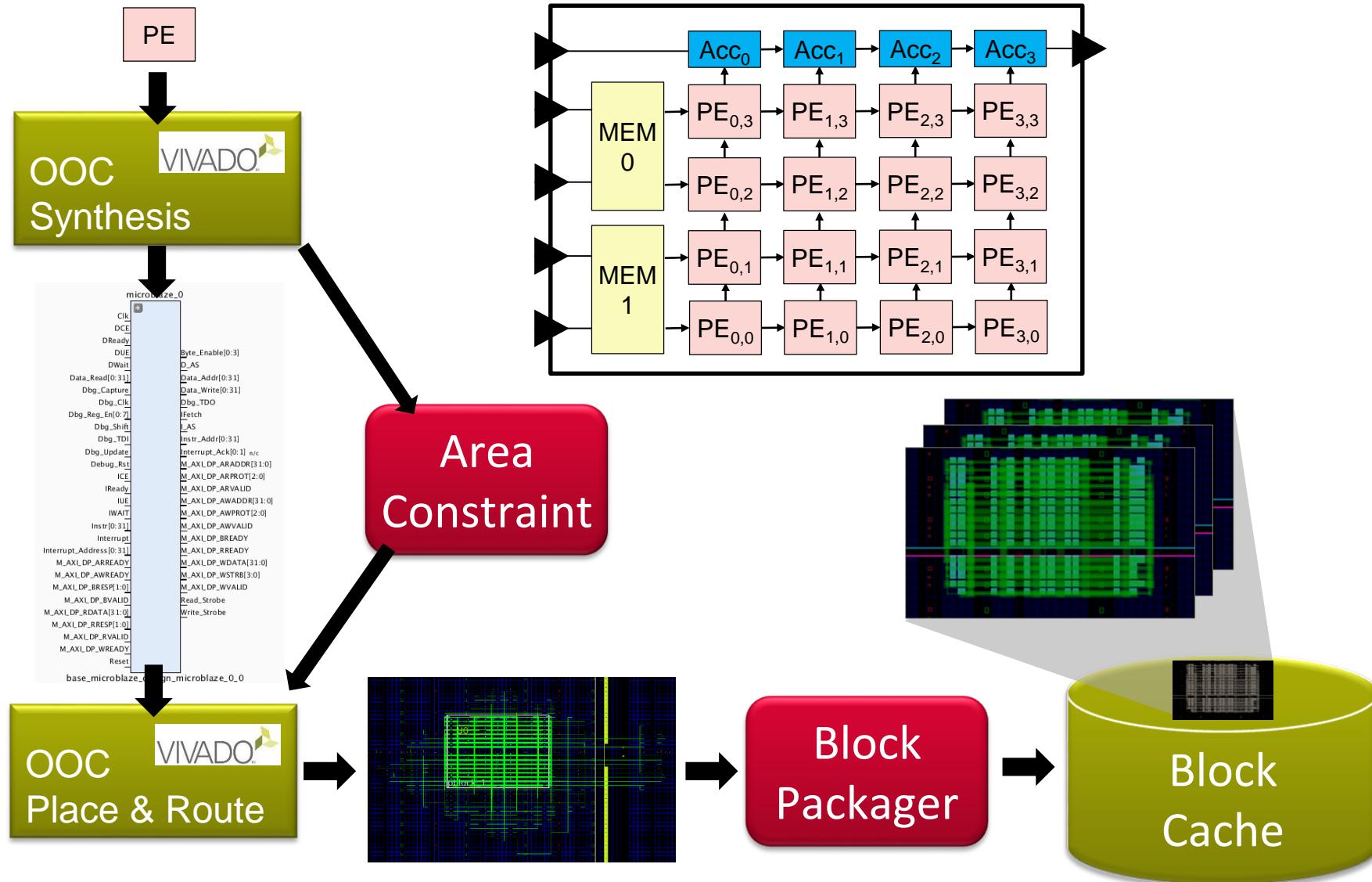


4. Run implementation

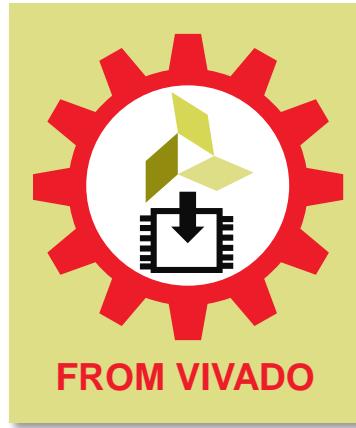
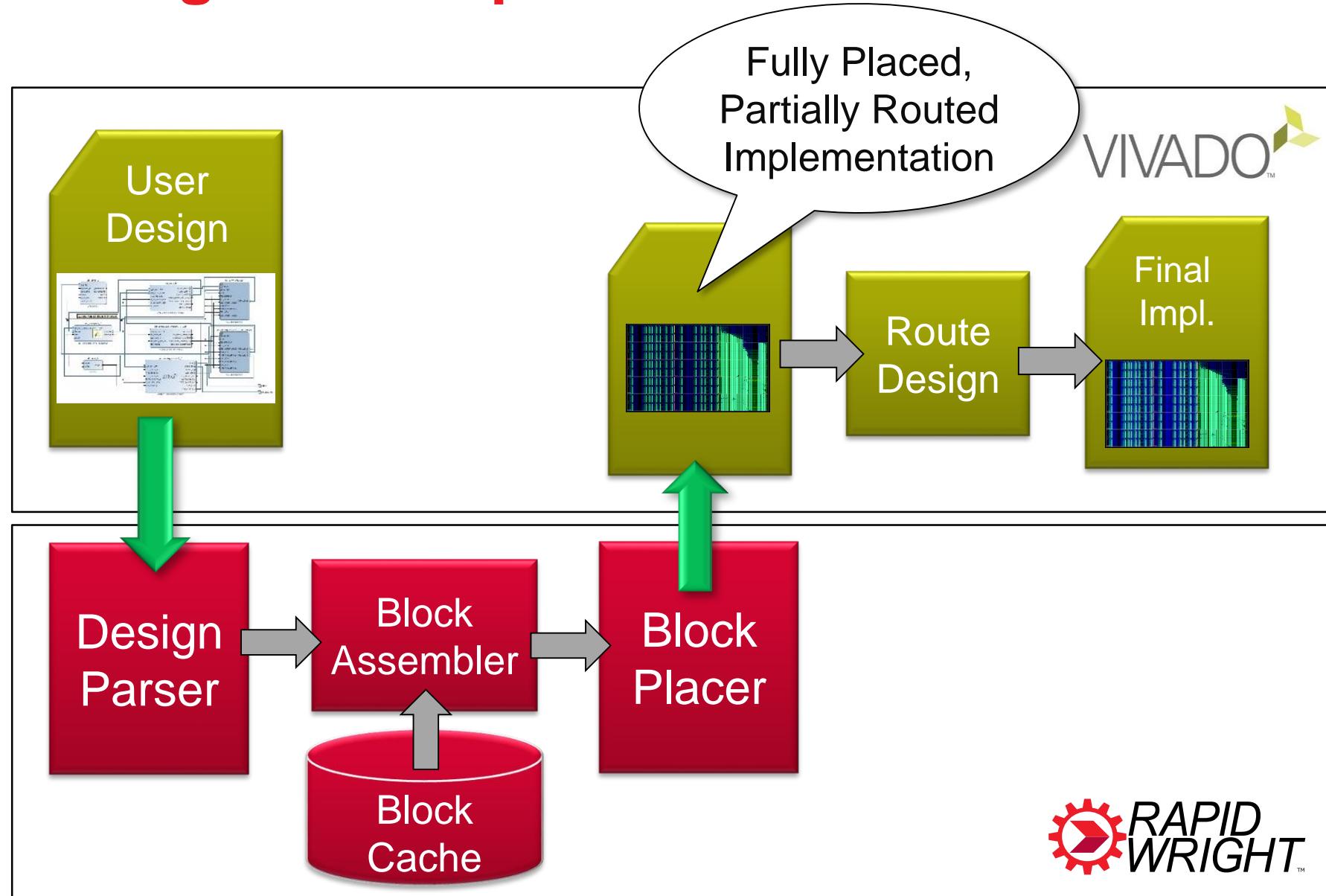
RapidWright
(Block Assembly,
P&R)



Creating Pre-implemented Modules (Vivado OOC Flow)



RapidWright Pre-implemented Module Flow



Design Performance Results

Design	Target Device	Baseline (initial design)	RapidWright ¹ Flow	Gain
Seismic	KU040	270MHz	390MHz	41%
FMA	KU115	270MHz	417MHz	54%
GEMM	KU115	391MHz	462MHz	16%
ML overlay	ZU9EG	368MHz	541MHz	50%

Speed Grade: -2

Utilization table

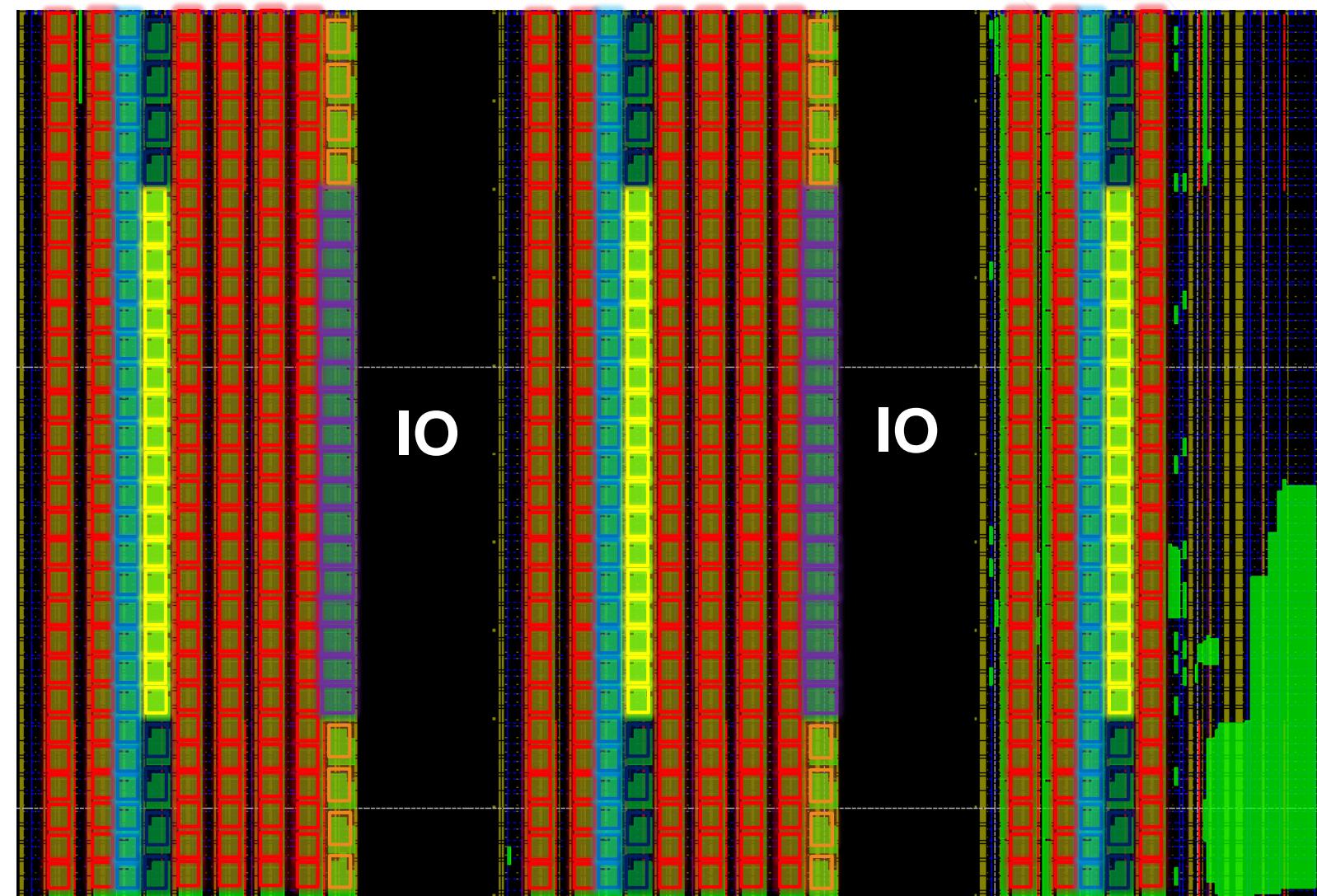
Design	LUT	FF	DSP	BRAM
Seismic	93%	5%	-	-
FMA (HPC design)	25%	50%	97%	6%
GEMM	19%	20%	87%	-
ML overlay	46%	29%	42%	96%

¹RapidWright: Enabling Custom Crafted Implementations for FPGAs, FCCM 2018

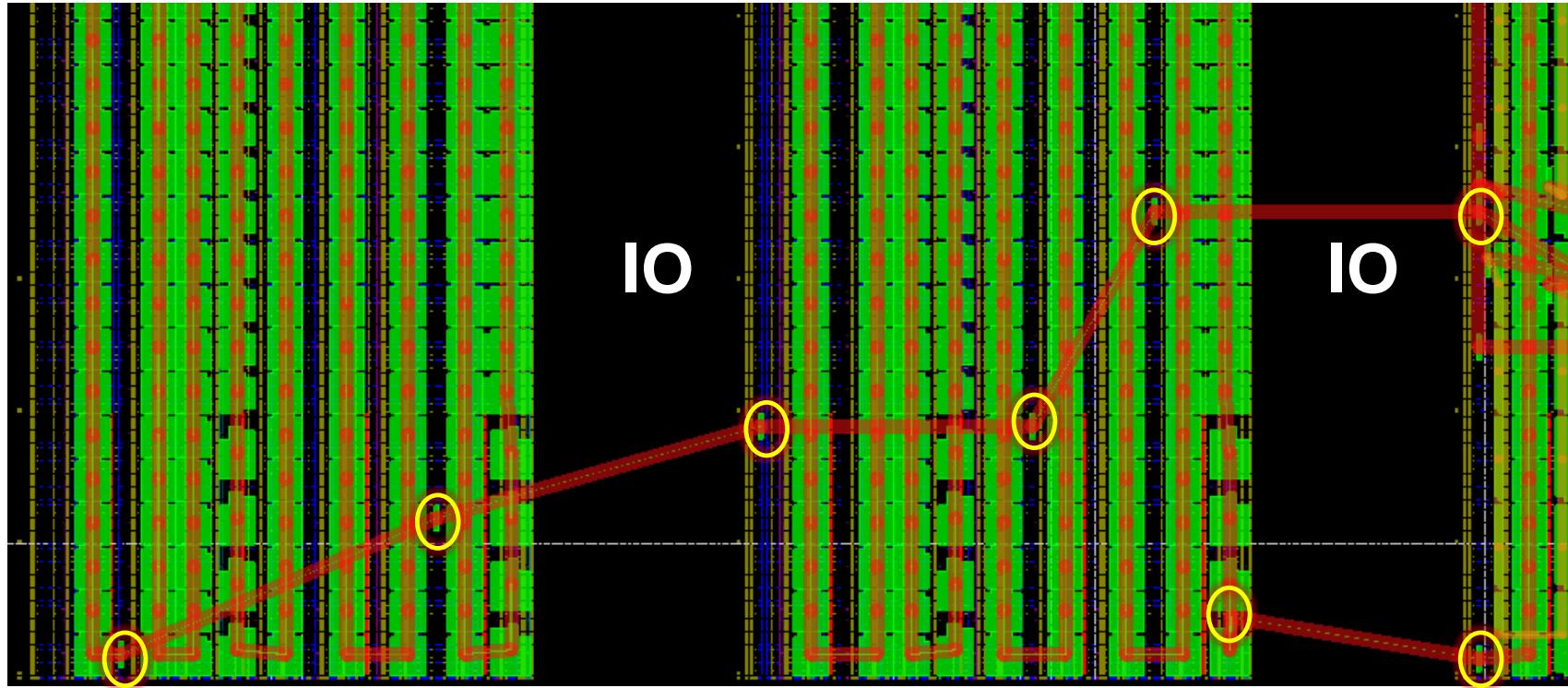
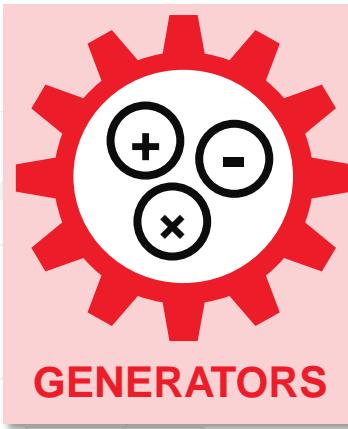
Re-locatability & Reuse of Multiple Implementations

RUN	F _{MAX} (MHz)
Vivado	270
RapidWright	417 (+53%)

- > 97% DSP utilization
- > 4.4 TeraOp/s
- > “Fabric discontinuities”
 - >> SLR boundary
 - >> IO Columns
 - >> Laguna Tiles



Latency Flexibility: AXI Stream Register Slices

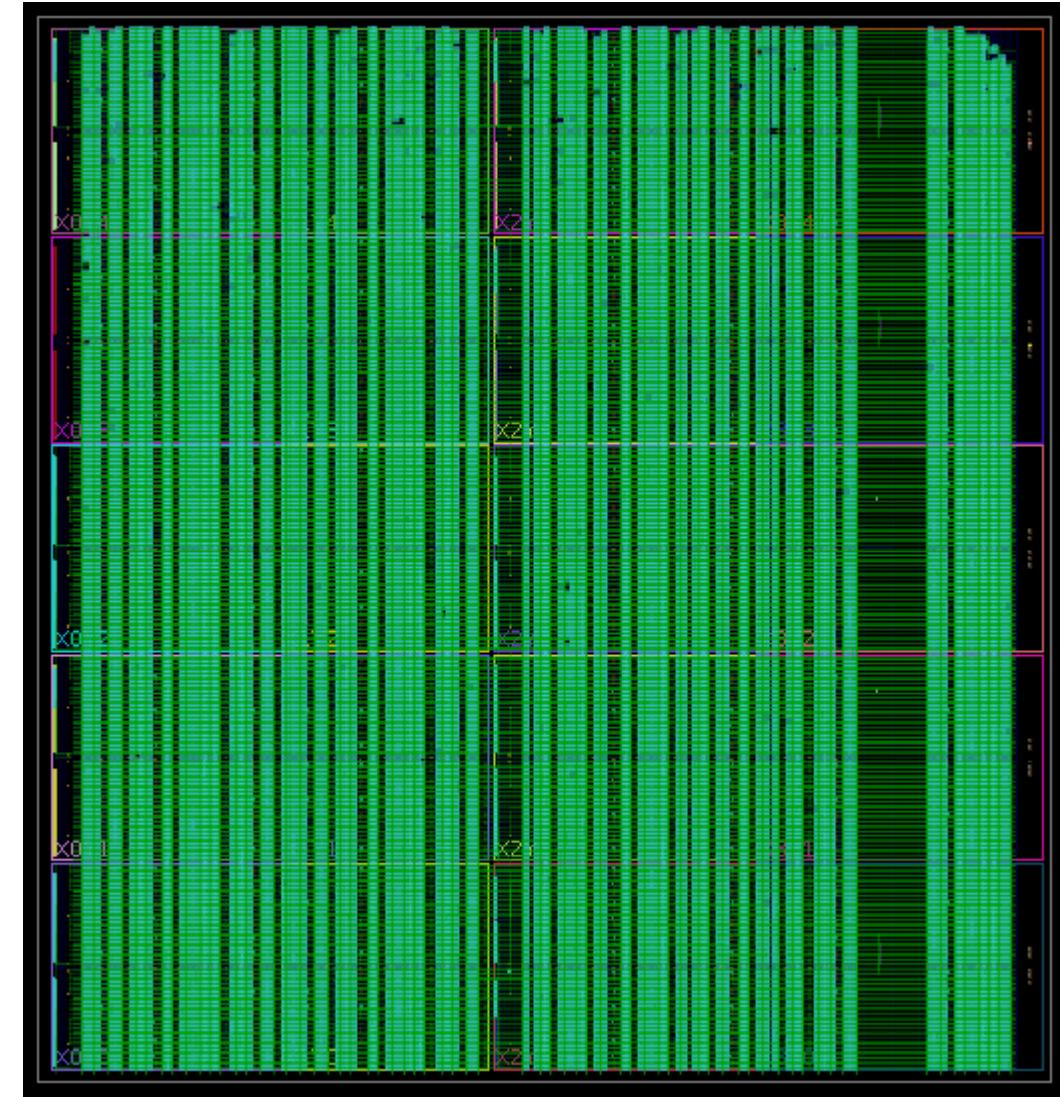


- > Exploiting latency-tolerance and architectural knowledge
 - >> Automatic insertion of latency blocks

Debugging with an ILA (ChipScope)

I downloaded my design
and it's not working. But it
works in simulation!

I added an ILA, but the
bug is gone!

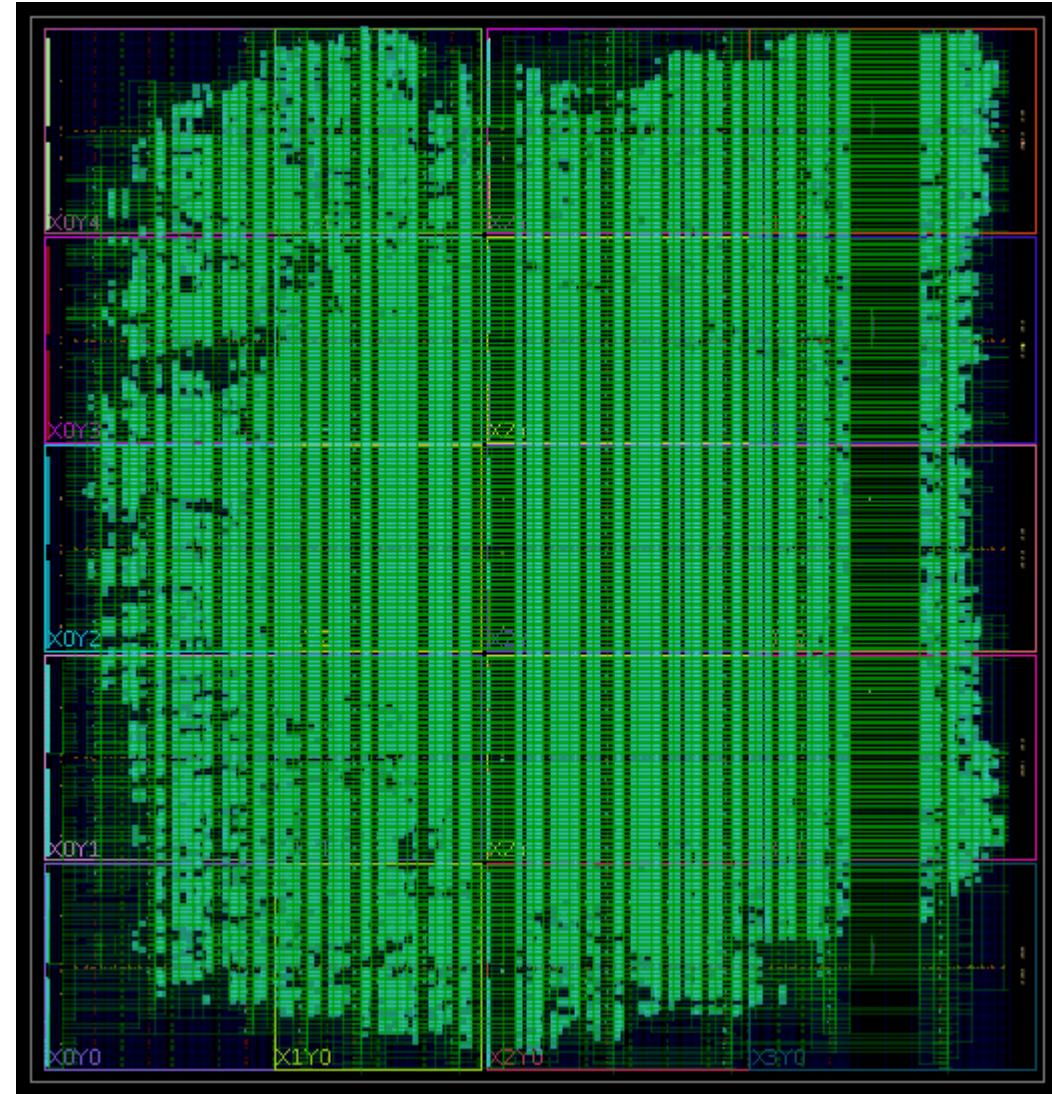


You'll need to recompile
with an ILA to debug it.



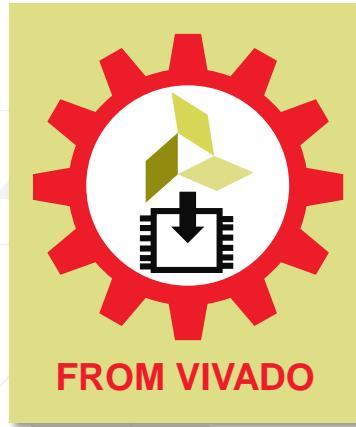
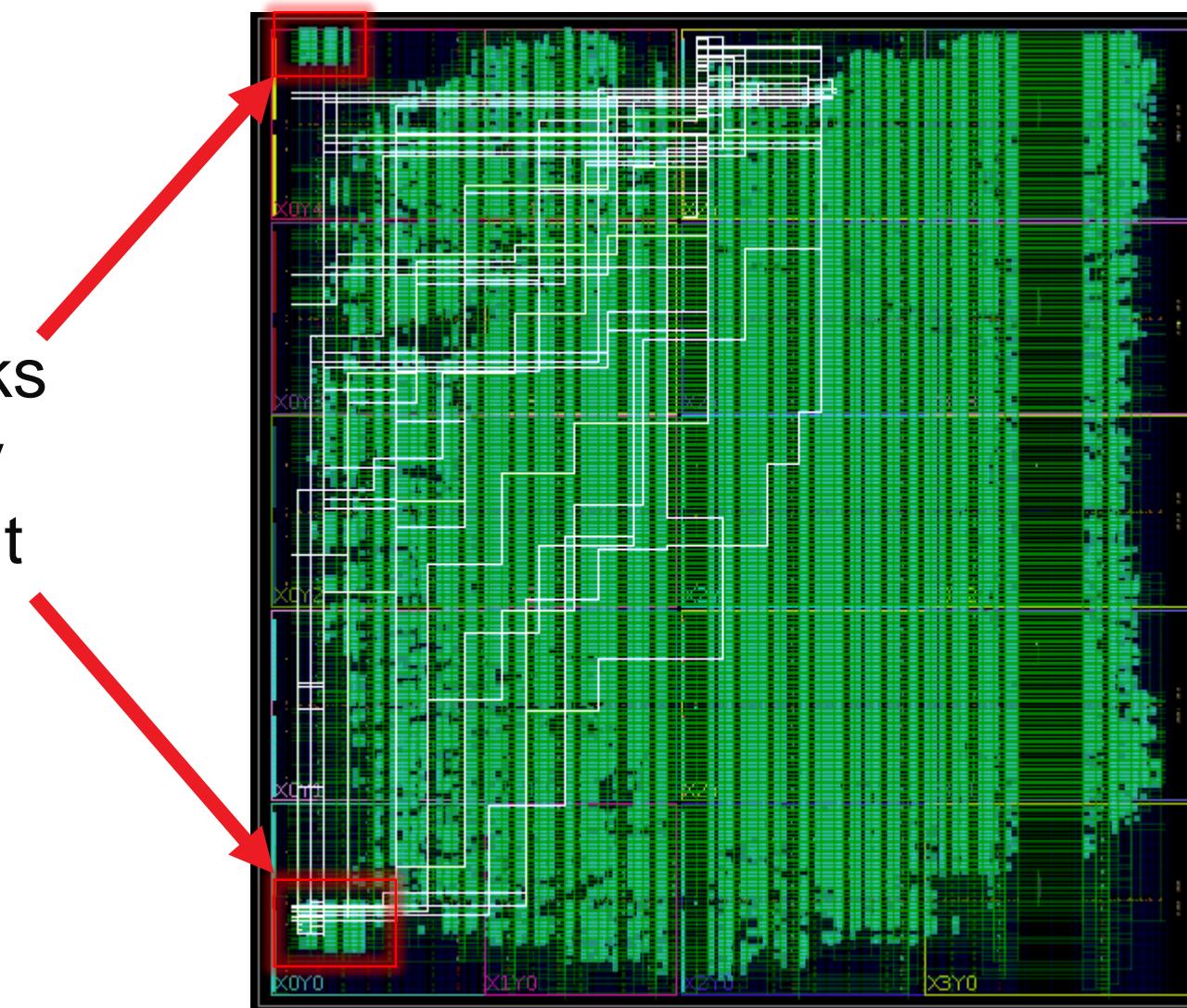
Experiment: Insert Pre-implemented ILA

- > Preserves existing
 - >> Placement
 - >> Routing
- > Only occupy unused resources

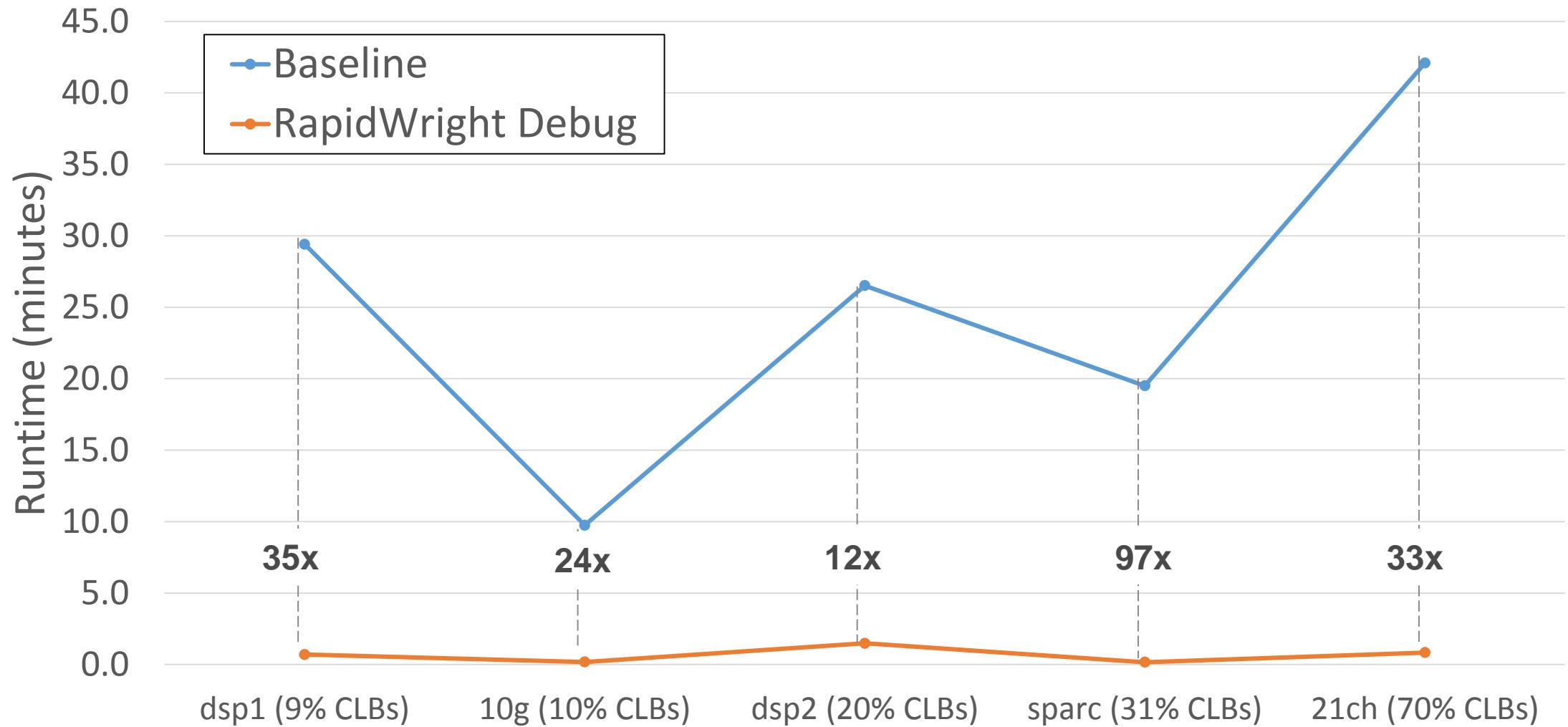


Preserve Existing Placement & Routing

Debug Blocks
Inserted by
RapidWright



Debug Instrumentation Speedup

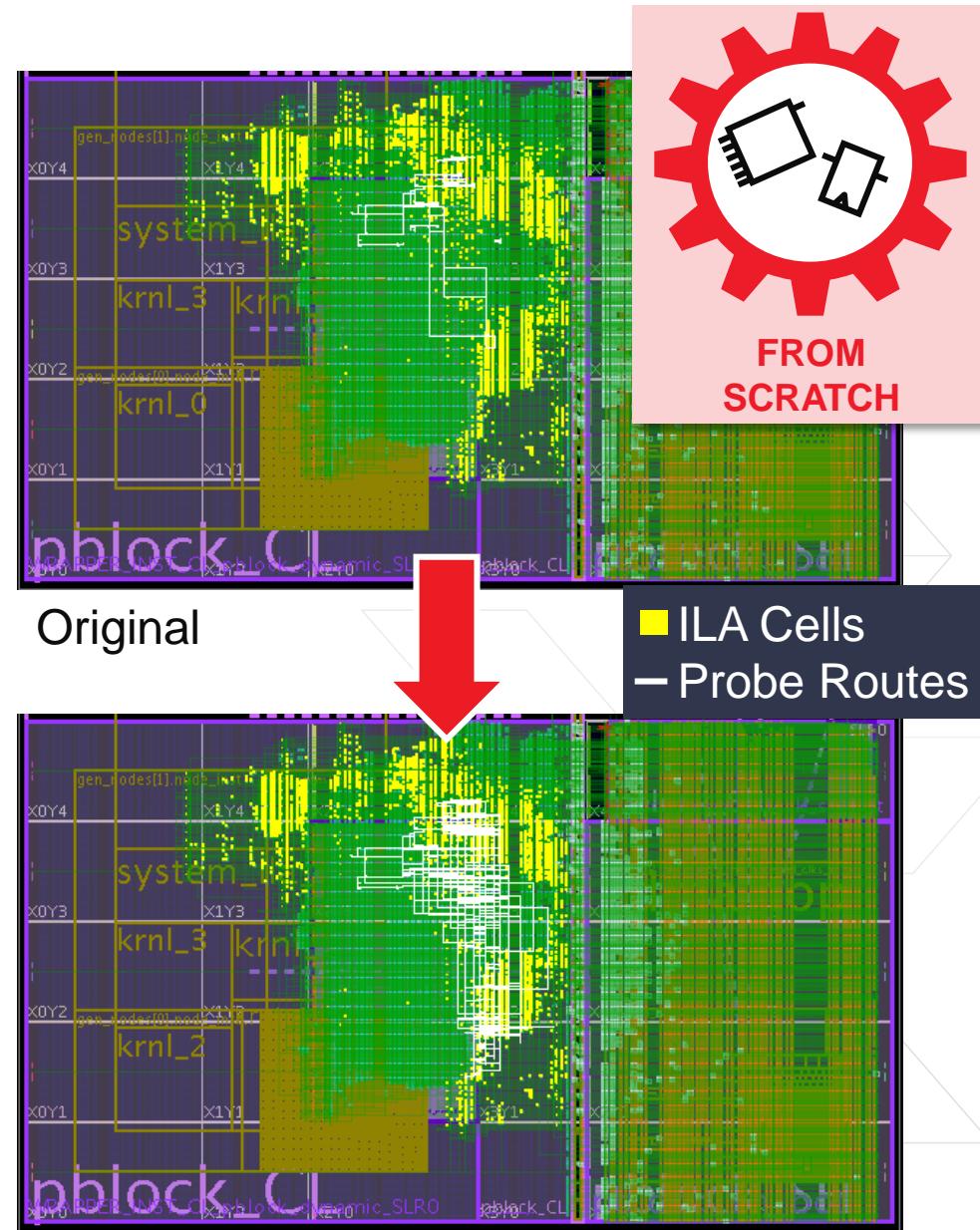


Beyond a Pre-implemented Methodology

> RapidWright probe router enables higher productivity

- » 21X more debug turns per day
- » Highest level of routing preservation possible
- » Future innovation:
 - iteration with extra probe inputs
 - Automatic insertion of pipeline flops to manage timing

Vivado modify_debug_probes	RapidWright ProbeRouter	Δ
130 mins	6.3 mins	21X

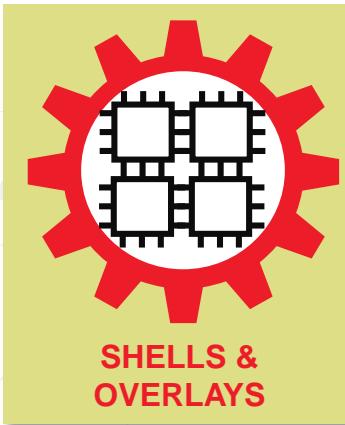
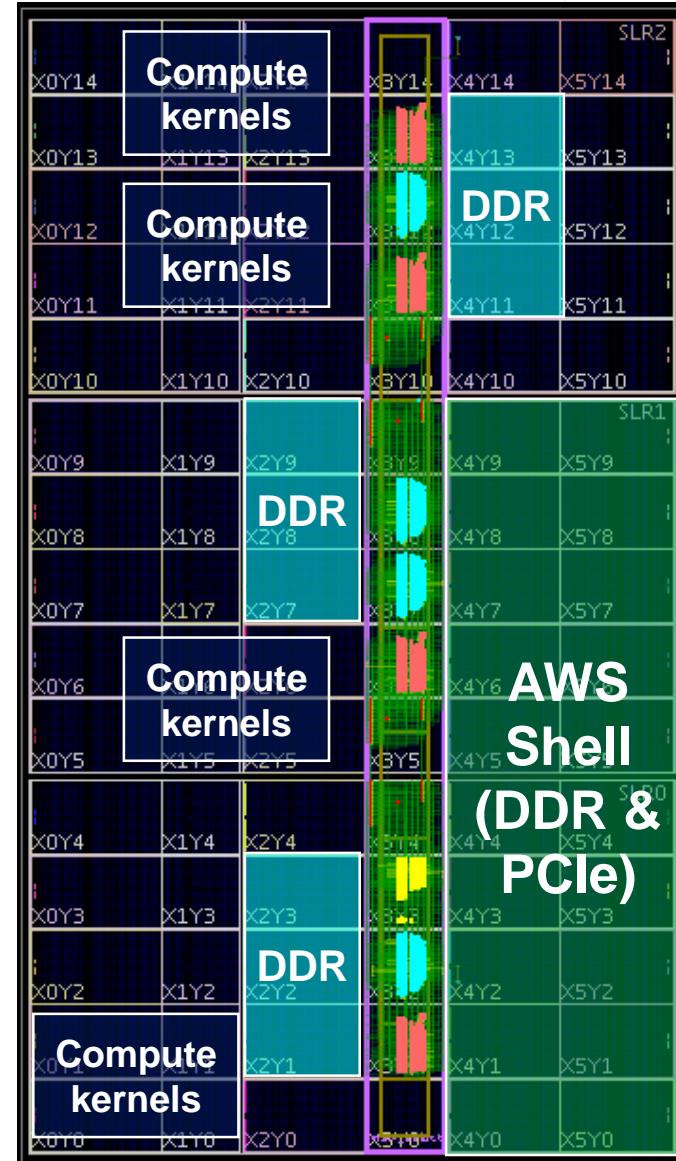


Pre-implemented Data Movement Shell

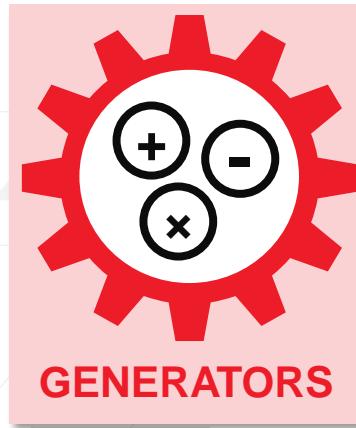
- > Goals
 - >> Minimize overhead of compute (and overlays)
 - >> Prove shell assembly model
- > Build-to-order LinkBlaze¹ shell
 - >> 512 bit, bi-directional
 - >> RapidWright Pre-implemented modules

Vivado	RapidWright
516MHz	620MHz (+20%)

¹ LinkBlaze: Efficient global data movement for FPGAs (ReConFig 2017)

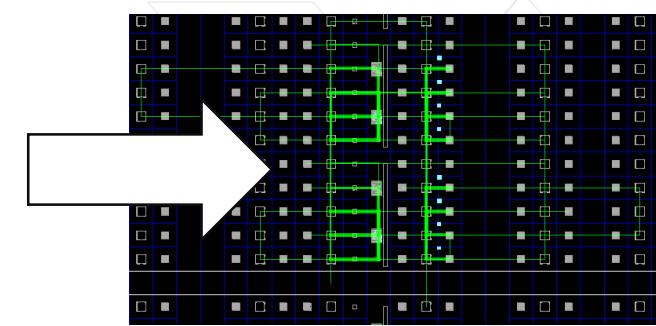
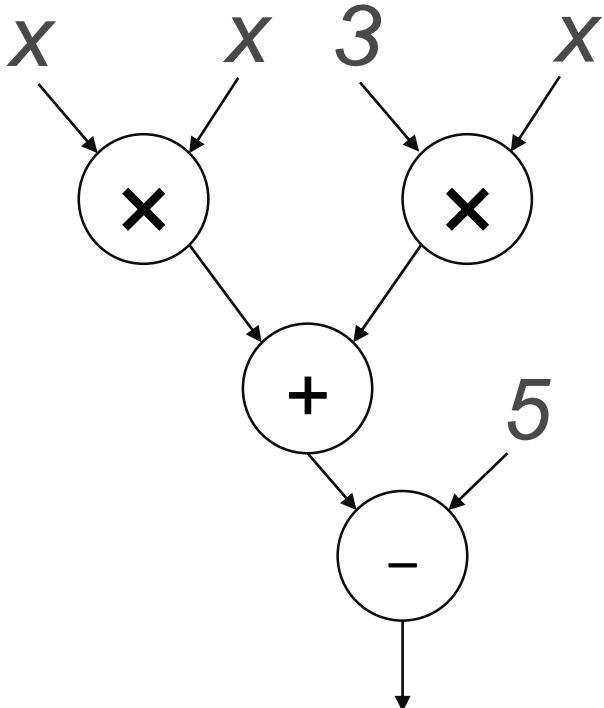


Just-in-time, Circuit Module Generators

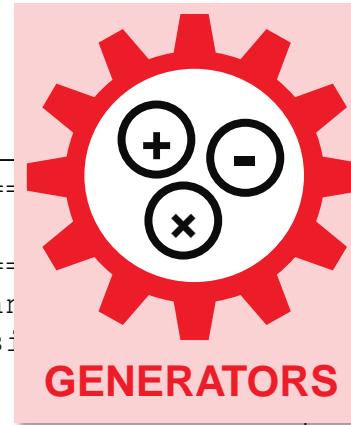


- > Build modules on-demand
 - >> Placed and routed *in seconds*
 - >> Reusable and compose-able
 - >> Target spec performance
- > Parameterizable Generators
 - >> Adder
 - >> Subtractor
 - >> Multiplier
- > Expression Generator
 - >> Invokes math generators
 - >> Built to spec: 775MHz

$$x^2 + 3*x - 5$$



RapidWright SLR Crossing DCP Creator



➤ SLR crossing module from scratch

- » Parameterizable
- » Closes timing at 760MHz
 - Clk Period: 1.313ns
- » Routed clock, placed and routed
- » Runs in seconds

```
=====
                           SLR Crossing DCP Generator
=====

This RapidWright program creates a placed and routed DCP that can be imported into UltraScale+ designs to aid in high speed SLR crossing. See the RapidWright documentation for more information.

Option                                Description
-----                                 -----
-?, -h                               Print Help
                                         (default: clk_in)
-a [String: Clk input net name]       (default: BUFGCE_X0Y218)
-b [String: Clock BUFGCE site name]  (default: clk)
-c [String: Clk net name]             (default: slr_crosser)
-d [String: Design Name]              (default: input)
-i [String: Input bus name prefix]    (default: LAGUNA_X2Y120)
-l [String: Comma separated list of   (default: _north)
     Laguna sites for each SLR crossing]
-n [String: North bus name suffix]    (default: slr_crosser.dcp)
-o [String: Output DCP File Name]    (default: xcvu9p-flgc2104-2-i)
-p [String: UltraScale+ Part Name]   (default: output)
-q [String: Output bus name prefix]  (default: GCLK_B_0_1)
-r [String: INT clk Laguna RX flops] (default: _south)
-s [String: South bus name suffix]   (default: GCLK_B_0_0)
-t [String: INT clk Laguna TX flops] (default: clk_out)
-u [String: Clk output net name]     (default: true)
-v [Boolean: Print verbose output]   (default: 512)
-w [Integer: SLR crossing bus width] (default: 1.538)
-x [Double: Clk period constraint (ns)] (default: BUFGCE_inst)
-y [String: BUFGCE cell instance name]
-z [Boolean: Use common centroid]
```

Ongoing Work: C Code to Full Chip Accelerator in Seconds

> RapidWright generator capabilities

UltraScale+ VU3P, 100% DSP utilization

Front-end C code parser still in development

Prototype back-end flow

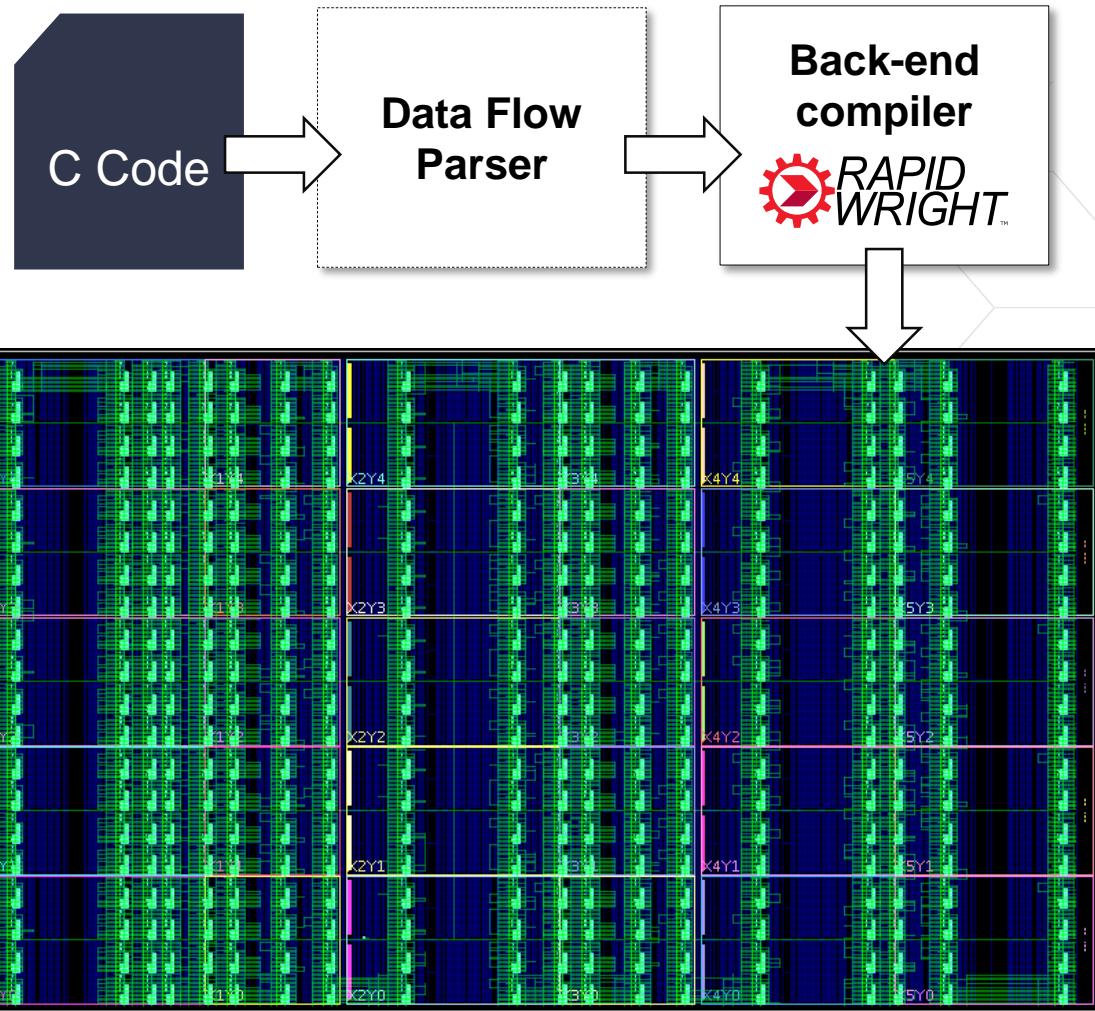
Runs in seconds (37 seconds)

Achieves spec frequency (775 MHz)

> Future integration work:

SLR crossing generator - target 750 MHz

LinkBlaze (data movement) solution



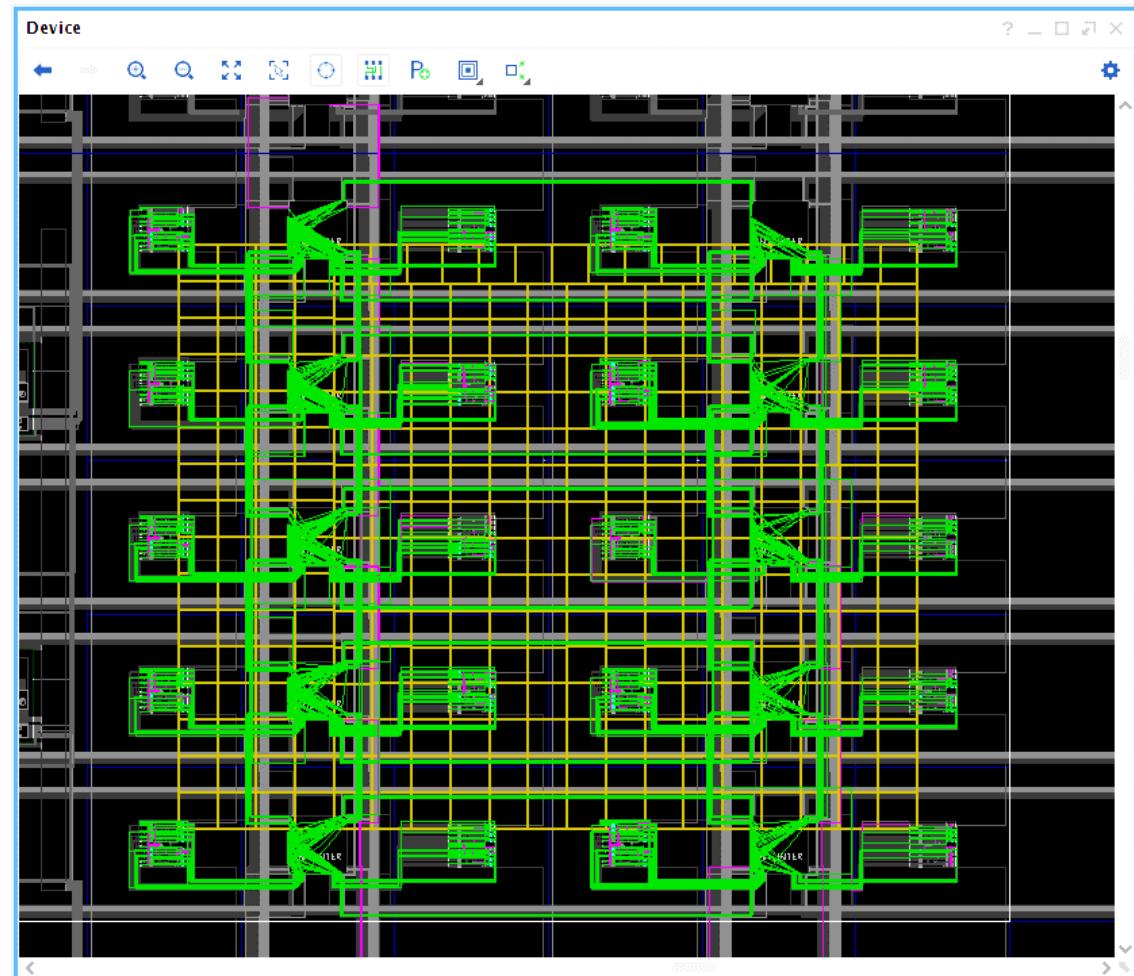
Leveraging Algorithmic Engines

➤ SAT Solver

- » Resolve difficult, localized congestion routing
 - Finds solutions where Vivado cannot
- » RapidWright front-end to SAT solver engine¹

➤ Future Work

- » Simultaneous SAT placement and routing solution
- » ILP Solvers
 - Potential for placement solutions

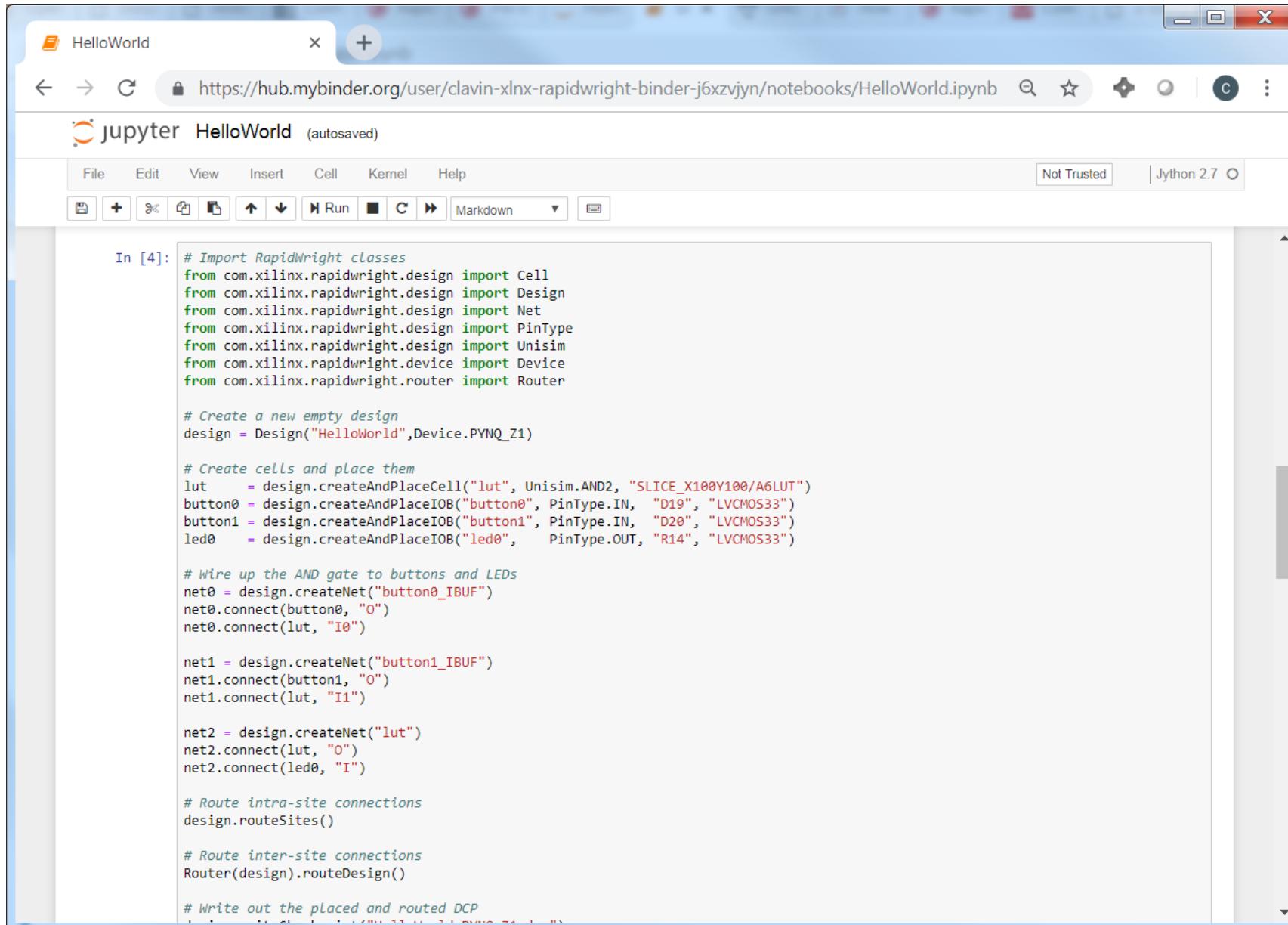


¹Fraisse, H., Gaitonde, D., *A SAT-based timing driven Place and Route flow for critical soft IP* (FPL 2018)

How do I get started with RapidWright?



Run RapidWright in Your Browser



The screenshot shows a Jupyter Notebook window titled "HelloWorld". The URL in the address bar is <https://hub.mybinder.org/user/clavin-xlnx-rapidwright-binder-j6xzvjyn/notebooks>HelloWorld.ipynb>. The notebook is titled "jupyter HelloWorld (autosaved)". The code in cell [4] is as follows:

```
In [4]: # Import RapidWright classes
from com.xilinx.rapidwright.design import Cell
from com.xilinx.rapidwright.design import Design
from com.xilinx.rapidwright.design import Net
from com.xilinx.rapidwright.design import PinType
from com.xilinx.rapidwright.design import Unisim
from com.xilinx.rapidwright.device import Device
from com.xilinx.rapidwright.router import Router

# Create a new empty design
design = Design("HelloWorld",Device.PYNQ_Z1)

# Create cells and place them
lut = design.createAndPlaceCell("lut", Unisim.AND2, "SLICE_X100Y100/A6LUT")
button0 = design.createAndPlaceIOB("button0", PinType.IN, "D19", "LVCMOS33")
button1 = design.createAndPlaceIOB("button1", PinType.IN, "D20", "LVCMOS33")
led0 = design.createAndPlaceIOB("led0", PinType.OUT, "R14", "LVCMOS33")

# Wire up the AND gate to buttons and LEDs
net0 = design.createNet("button0_IBUF")
net0.connect(button0, "O")
net0.connect(lut, "I0")

net1 = design.createNet("button1_IBUF")
net1.connect(button1, "O")
net1.connect(lut, "I1")

net2 = design.createNet("lut")
net2.connect(lut, "O")
net2.connect(led0, "I")

# Route intra-site connections
design.routeSites()

# Route inter-site connections
Router(design).routeDesign()

# Write out the placed and routed DCP
design.writeDCP("HelloWorld.dcp")
```



FPGA'19 Invited Tutorial Paper

Build Your Own Domain-specific Solutions with RapidWright

Invited Tutorial

Chris Lavin and Alireza Kaviani
Xilinx Research Labs

San Jose, CA

chris.lavin@xilinx.com, alireza.kaviani@xilinx.com

ABSTRACT

As the complexity of programmable architectures increases with advances in silicon process technology, there is a growing need to extract greater productivity and performance from the tools. Due to their inherent reconfigurability, FPGAs are proving to be valuable targets for more efficient domain-specific architectures. However, FPGA implementation tools are designed for a broad set of applications.

In this paper we describe RapidWright, an open source framework that enables customized implementations for Xilinx FPGAs. RapidWright enables implementation tools that can take advantage of the great potential of domain-specific attributes—leading to greater productivity and performance. The focus of this paper is to provide an introductory reference of RapidWright and its use cases so that others may be empowered to adapt their implementations to their domain-specific applications.

CCS CONCEPTS

- Hardware → Reconfigurable logic and FPGAs;
- Computer systems organization → Reconfigurable computing;

KEYWORDS

Domain-specific, Open Source, FPGA, Xilinx, Vivado

ACM Reference Format:

Chris Lavin and Alireza Kaviani. 2019. Build Your Own Domain-specific Solutions with RapidWright. In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19)*, February 24–26, 2019, Seaside, CA, USA. ACM, New York, NY, USA, Article 4, 9 pages.
<https://doi.org/10.1145/3299602.3299928>

1 INTRODUCTION

RapidWright [1] is an open source platform with a gateway to Xilinx's back-end implementation tools (Vivado) that raises the implementation abstraction while maintaining the full potential of advanced FPGA silicon. RapidWright works synergistically with Vivado through design checkpoints (DCPs, see Figure 1) to enable highly customizable implementations. Vivado can produce highly

permissons to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for components of this work owned by others than the author(s) must be honored. Abstracting or reindexing without prior permission is permitted, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

FPGA '19, February 24–26, 2019, Seaside, CA, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.

ACM ISBN 978-1-4503-6197-4/19/02...\$15.00

<https://doi.org/10.1145/3299602.3299928>

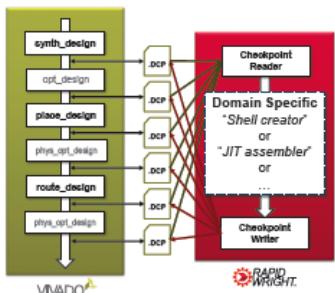


Figure 1: Vivado and RapidWright DCP Compatibility

optimized implementations for key design modules to deliver the highest performance. RapidWright can then replicate, relocate and assemble these tuned modules to compose a complete application and preserve high performance.

RapidWright's native gateway to Vivado also sets the groundwork for an ecosystem aimed at further advancing FPGA tools. It empowers academic and industry researchers by combining the commercial credibility of FPGA tools with the agility of an open source framework, leading to innovative solutions that might not be feasible otherwise.

This paper serves as a supplemental reference to the RapidWright tutorial with an aim to provide some fundamentals about the framework and introductory use cases. In the remainder of this paper we describe RapidWright and its capabilities in Section 2, some example use cases in Section 3 and conclude in Section 4. Supplementary material on Xilinx architecture is included in Appendix A to help orient the reader regarding specific RapidWright constructs.

2 RAPIDWRIGHT STRUCTURE

RapidWright is implemented in Java and distributed with a foundational API library that provides access to design checkpoint (DCP) files and Vivado-compatible device models. A high-level diagram showing the organization of the project is shown in Figure 2. There are three core Java packages (groups of classes) within RapidWright: `device`, `edf` (logical netlist) and `design` (physical netlist) and this section describes the purpose and composition of each one.

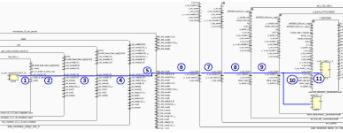


Figure 2: Logical netlist view of a particular physical net

used in certain situations to prevent components inside the site from being moved.

Routing nets outside of a site (inter-site) is different from routing outside of sites (intra-site) and SiteInst maintains all relevant information concerning intra-site routing. Routing inside of a site is typically for internal connections between components. In fact, when constraints placed and routed logic, it can be beneficial to compare SiteInst content from Vivado-generated implementations to determine correctness. This can be done by loading placed and routed DCPs from Vivado and then reading and querying the respective SiteInst objects to establish patterns and verify the site PIP usage.

Routing is accomplished inside a site through SiteIPs, which can be either direct through routing BELs and some logic BELs (such as LUTs). The SiteIP object in RapidWright maintains site PIP usage. By default, all site PIPs are turned off. A siteIP is added to the SiteInst then it is marked as being turned on or used.

2.6 Net

A Net in RapidWright contains the routing information to physical cells connected to the same physical net, for example, consider the net depicted in Figure 3. This figure shows the logical netlist connection of three cells over one physical net. However, there are

11 separate physical connections to the same physical net.

The net pin and PIP pin sources are represented as objects, resulting in pins defined as

2.7 Module

A module is a collection of objects that represent a collection of cells and interconnects. It is a top-level object that contains EDIFCellNames, EDIFPortNames, and EDIFNetNames.

2.8 EDIF Data Structure Reference to Vivado Netlist View

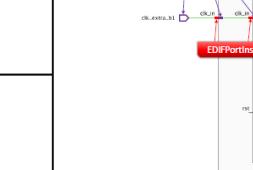


Figure 3: EDIF Data Structure Reference to Vivado Netlist View

processes entering/leaving the cell. Figure 4 illustrates how RapidWright EDIF-based objects map to a Vivado netlist schematic view.

2.9 Design Package (Physical Netlist)

The design package is the collection of objects used to describe how a logical netlist maps to a device netlist. A design is also referred to as a physical netlist or implementation. It contains all of the primary logical cell mappings to hardware, specifically the cell to I/O block, cell to routing resources, and cell to interconnect or routing.

The design class in RapidWright is the central type of intermediate representation that tracks the logical netlist, physical netlist, constraints, the device and part references among other things. The design class is most similar to a design checkpoint in that it contains all the information necessary to create a DCP. The design class of this section describes the major object classes found in a design.

2.10 Cell (A BEL Instance)

At the lowest level, a RapidWright Cell maps a logical leaf cell from the EDIF netlist (EDIFCell) to a BEL as shown in Figure 4. The cell name is typically the full hierarchical logical name of the leaf cell to which it maps. A cell also maintains the logical cell pin mappings to the physical cell pin mappings (BELPin).

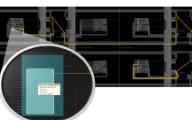


Figure 4: Physical netlist view of a particular physical net

definition of an implementation. This object is unique to RapidWright and is one of its enabling constructs that allow placed and routed information to be preserved, relocated and replicated. A module contains both the logical and physical netlist elements and is a top-level object that contains EDIFCellNames similar to a placed and routed out-of-context DCP; however, RapidWright enables the implementation to be replicated or relocated to multiple comparable areas of the fabric.

A RapidWright module is represented by the Module class in the `device` and `design` modules in definition below:

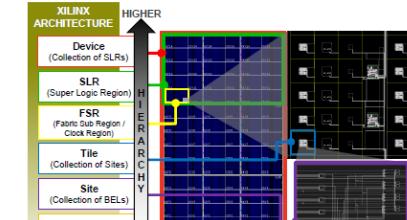


Figure 5: Xilinx Architecture Hierarchy

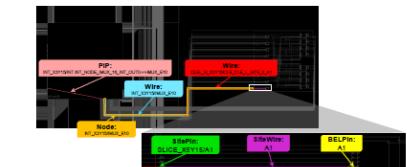


Figure 6: Inter-site and Intra-site Routing Resources

used as the “placement” of the cell. Non-leaf cells need more information to be placed and routed correctly, when one uses Vivado command `place_dts`, it is essentially mapping cells in the netlist to the compatible legal sites. Routing BELs are programmable muxes used to route signals from BELs. Routing BELs do not support any design elements cells from the netlist do not occupy routing BEL sites. However, if a design needs to route through unused LUTs (or other BELs) using site PIPs to complete a route.

2.11 SiteInst

The site representation and implementation in Vivado is BEL-centric (Basic Element of Logic). The SiteInst keeps track of three major mapping/attribute:

- (1) Map of all cells to BELs (placements in site)

- (2) Site to Site Wires (intra-site routing)

- (3) Net to Site Wires (inter-site routing)

Each SiteInst maps to a single, compatible site within a device.

The SiteInst is configured to a type using a SiteType that is either the primary type or alternate site type of the host site.

RapidWright also preserves the same Vivado “Used” flag is

RapidWright Resources: www.rapidwright.io

The image displays a collection of developer tools and documentation resources for RapidWright:

- Top Left:** A browser window showing the official RapidWright landing page (www.rapidwright.io). It features a background image of a printed circuit board (PCB) and the tagline "ADAPTING IMP TOOLS TO YOU".
- Top Middle:** A browser window showing the "RapidWright Documentation" (www.rapidwright.io/docs/index.html). The version is listed as 2018.3.0. The page includes a search bar and a sidebar with links to "Introduction", "Getting Started", "FPGA Architecture Basics", "Xilinx Architecture Terminology", "RapidWright Overview", "Design Checkpoints", "Implementation Basics", "A Pre-implemented Module Flow", "RapidWright Tutorials", and "Frequently Asked Questions".
- Top Right:** A browser window showing the GitHub repository for "Xilinx/RapidWright". The repository has 3 contributors and the latest commit was 2 days ago.
- Bottom Left:** A browser window showing the JavaDoc API documentation for the "com.xilinx.rapidwright" package. The "OVERVIEW" tab is selected. The sidebar lists various classes and packages, including AbstractRouter, AddSubGenerator, ArithmeticGenerator, BEL, BELClass, BELPin, BELPinDirection, BlockKreator, BlockGuide, BlockInst, BlockPlacer, BlockPlacer2, BlockScene, BlockStitcher, BlockUpdater, BlockView, BrowseDevice, Cell, CellPin, ClockRegion, and several subclasses like com.xilinx.rapidwright.debug, com.xilinx.rapidwright.design, com.xilinx.rapidwright.design.blocks, com.xilinx.rapidwright.design.tools, etc.
- Bottom Right:** A terminal window showing a command-line interface. The user is navigating through a directory structure and running commands related to "pyng" and "pynq".

>> 34

Today After Lunch (1:45PM)

RapidWright FPGA 2019 Deep Dive Tutorial

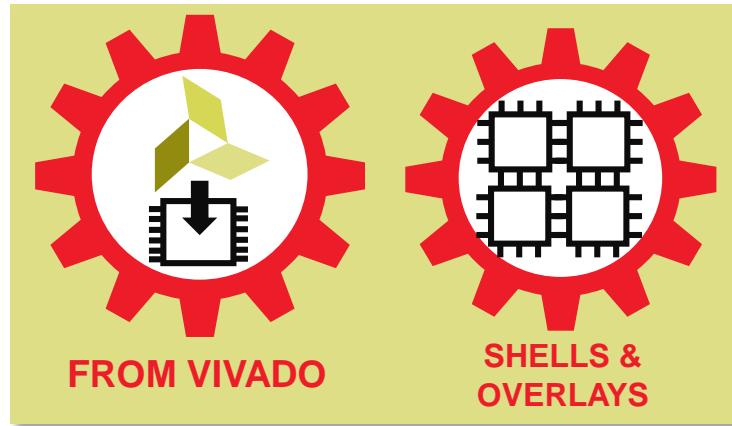
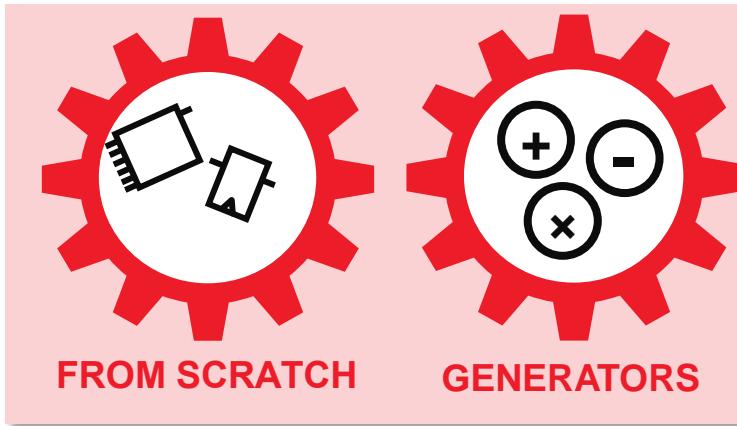
Tutorial Segment	Time	Purpose
Hello, World 	5 mins	Intro to RapidWright within Jupyter Notebook
Create Netlist from Scratch 	10 mins	How to build a netlist from scratch
Pipeline Generator	15 mins	How to generate a circuit in RapidWright
Pre-implemented Modules: Part I	15 mins	How to create a pre-implemented module
Pre-implemented Modules: Part II	15 mins	How to use and relocate pre-implemented modules
Probe Re-router 	20 mins	Fast probe routing on existing implementation
SAT Router 	15 mins	How to use a SAT engine to solve routing congestion
Create and Use an SLR Bridge	25 mins	Combine Vivado and RapidWright generated circuits



Conclude

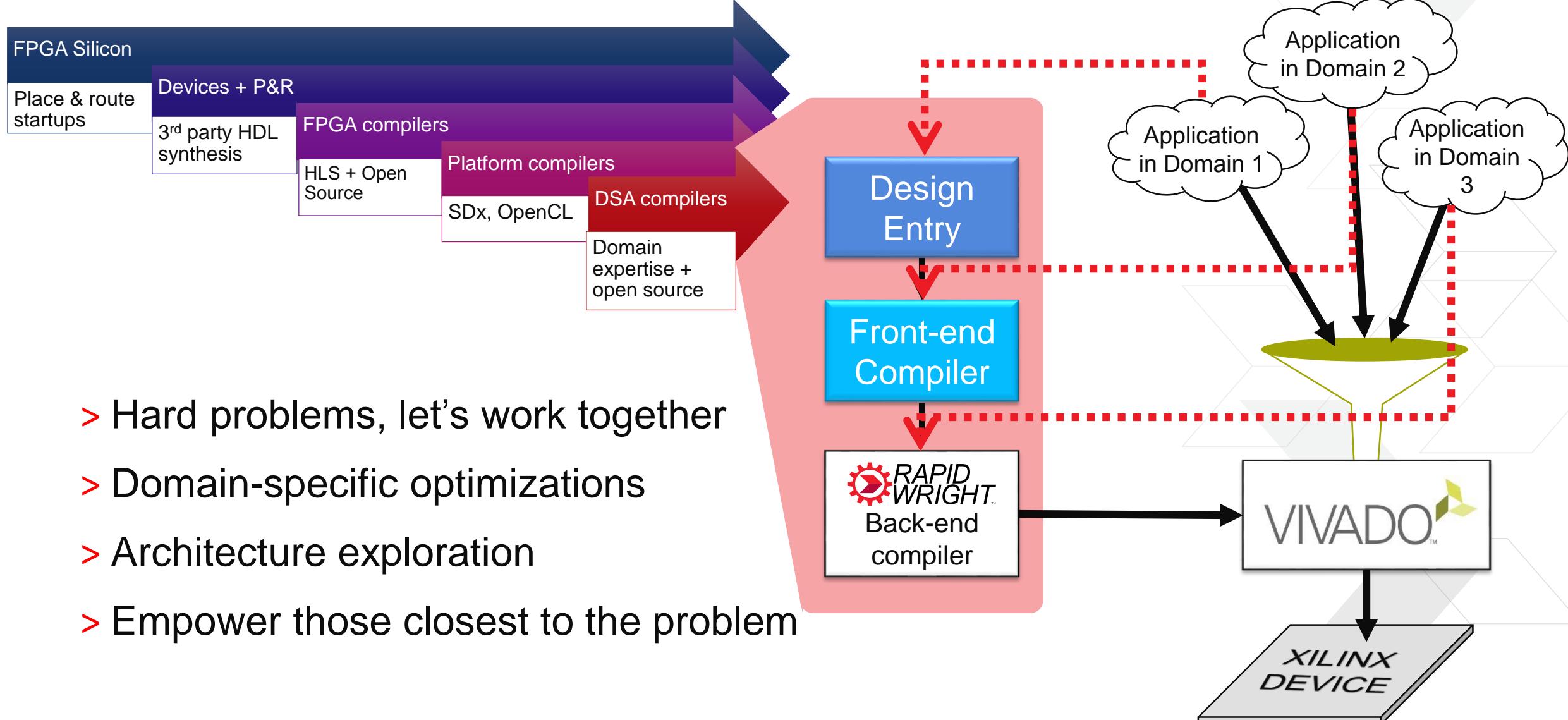


Summary



- > Build routed circuits & reuse P&R circuits
- > RapidWright enables:
 - Performance by 50%
 - Debug productivity >10X
- > Leverage algorithmic engines (SAT, ILP, ...)
- > www.rapidwright.io

RapidWright Enables DSA Compilers



**Adaptable.
Intelligent.**

