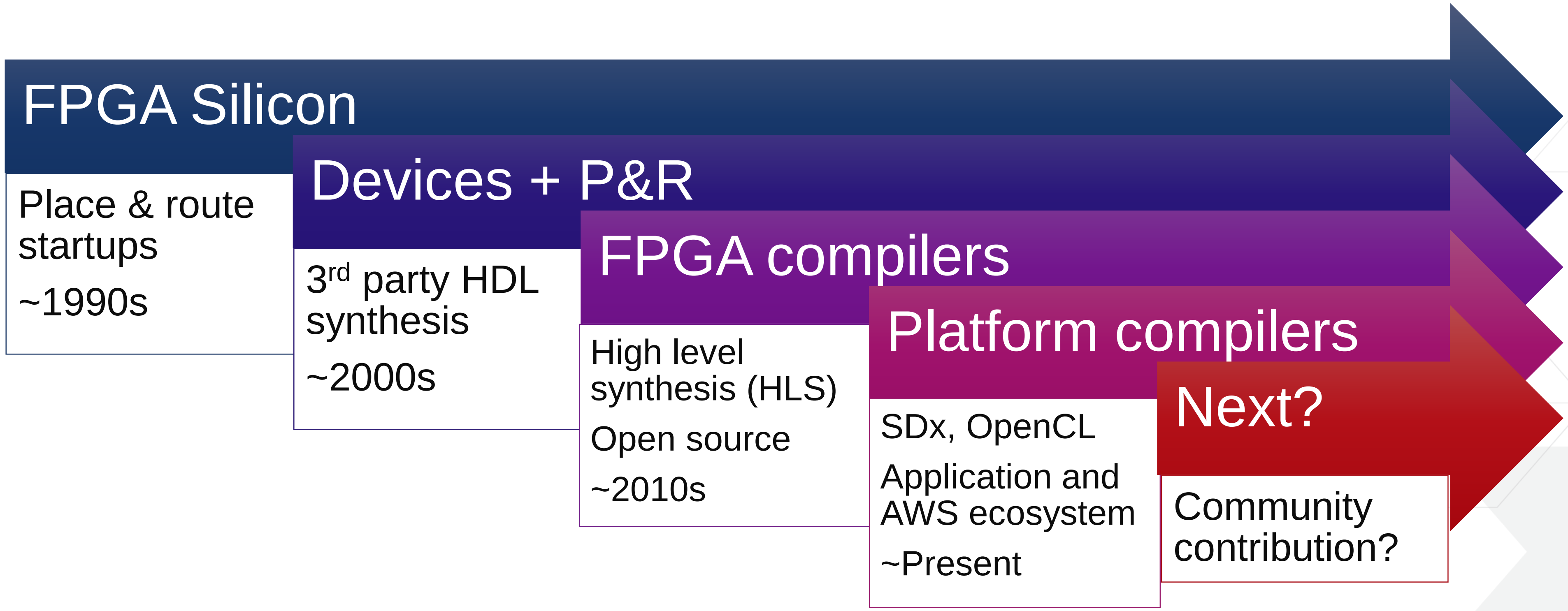
Build Your Own Domain-specific Solutions with RapidWright

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

XILINX.

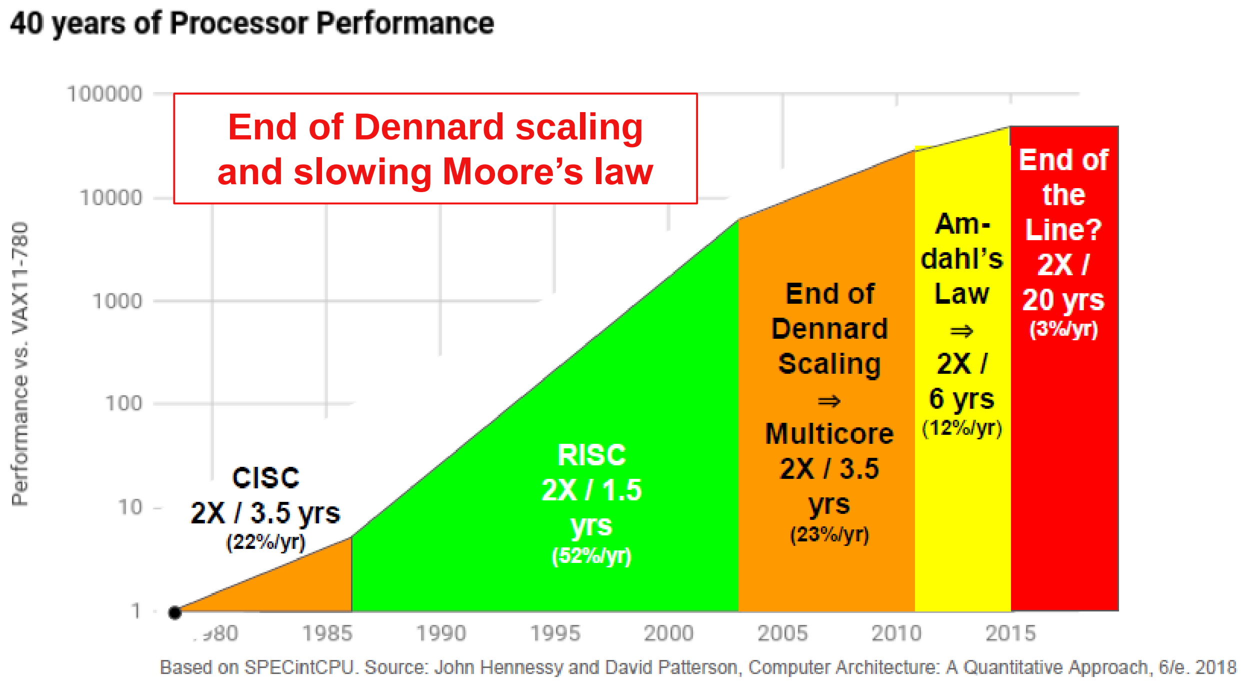
- Why are Domain-specific solutions important?
  - RapidWright value proposition
  - Why open source?
- What is RapidWright?
- How to use RapidWright?

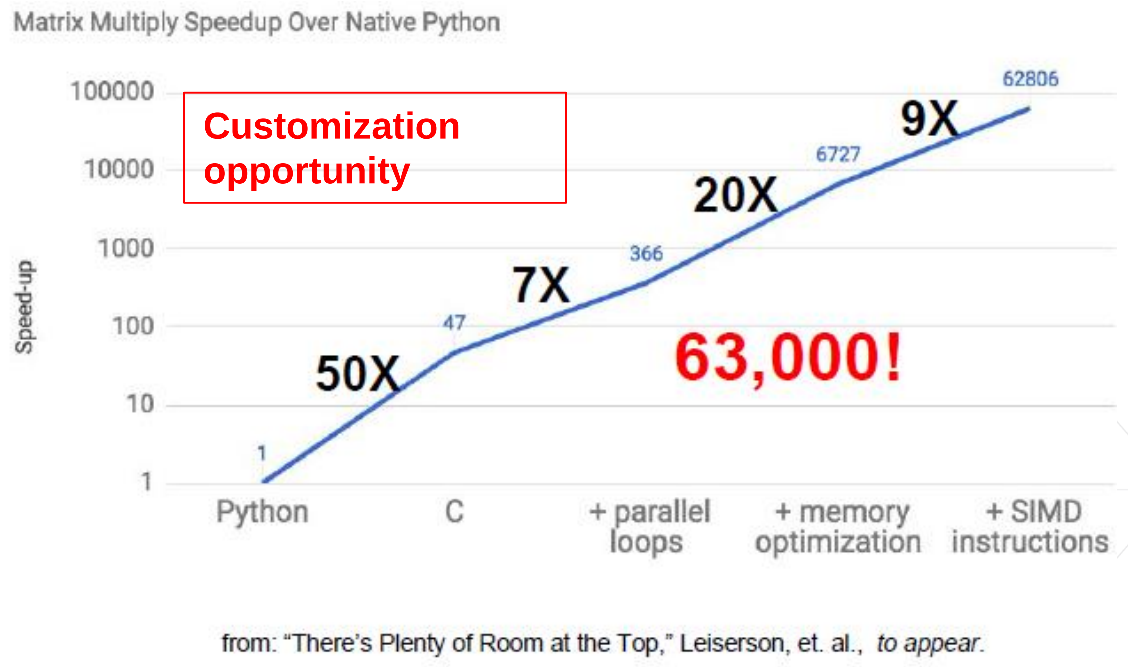
FPGA Industry and Community Dynamics



> Continuous industry and community engagement

The Age of Domain Specific Architectures

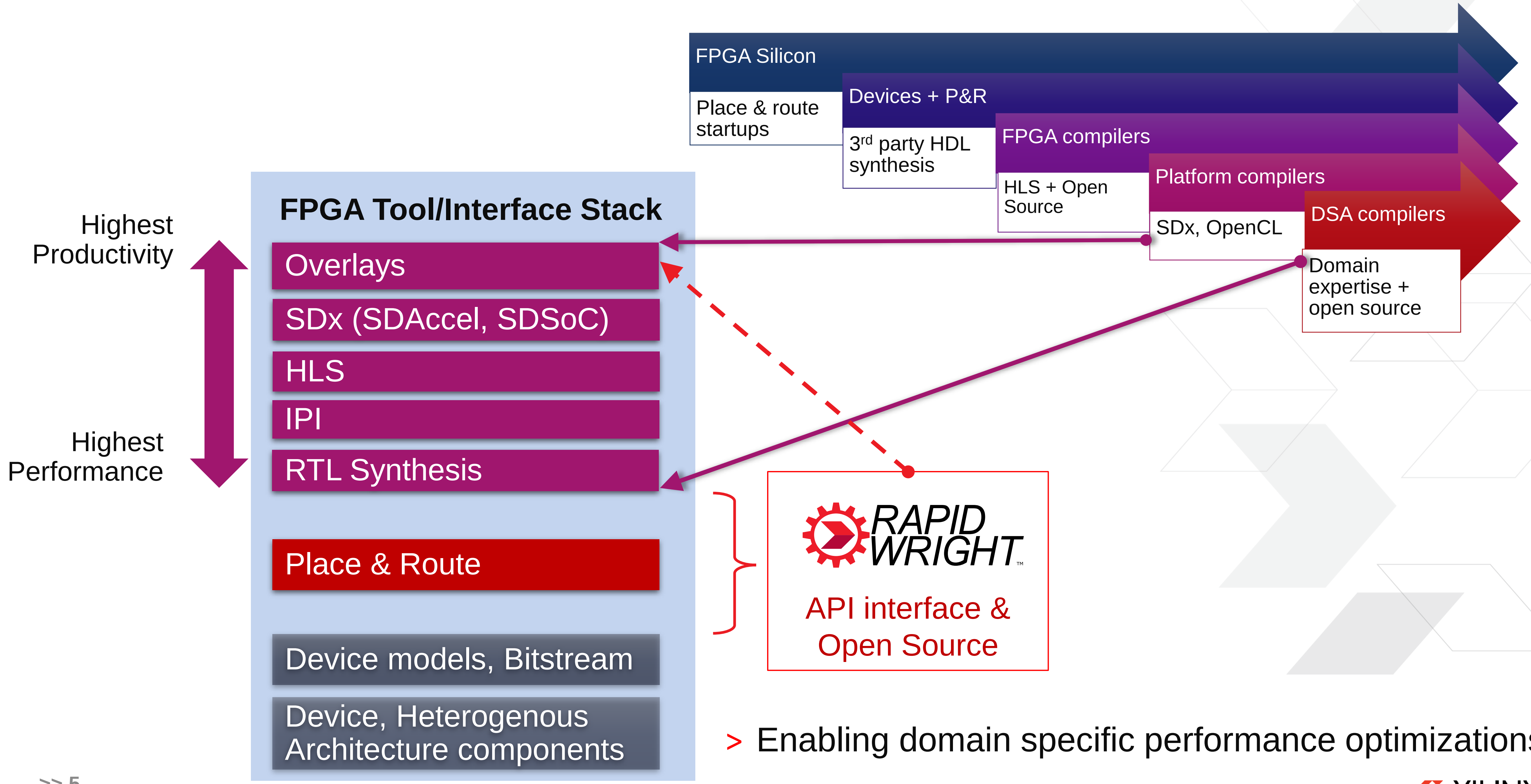




- Achieve higher efficiency by tailoring the architecture to characteristics of the domain
  - More effective parallelism for a specific domain, More effective use of memory bandwidth
  - Domain specific programming language

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

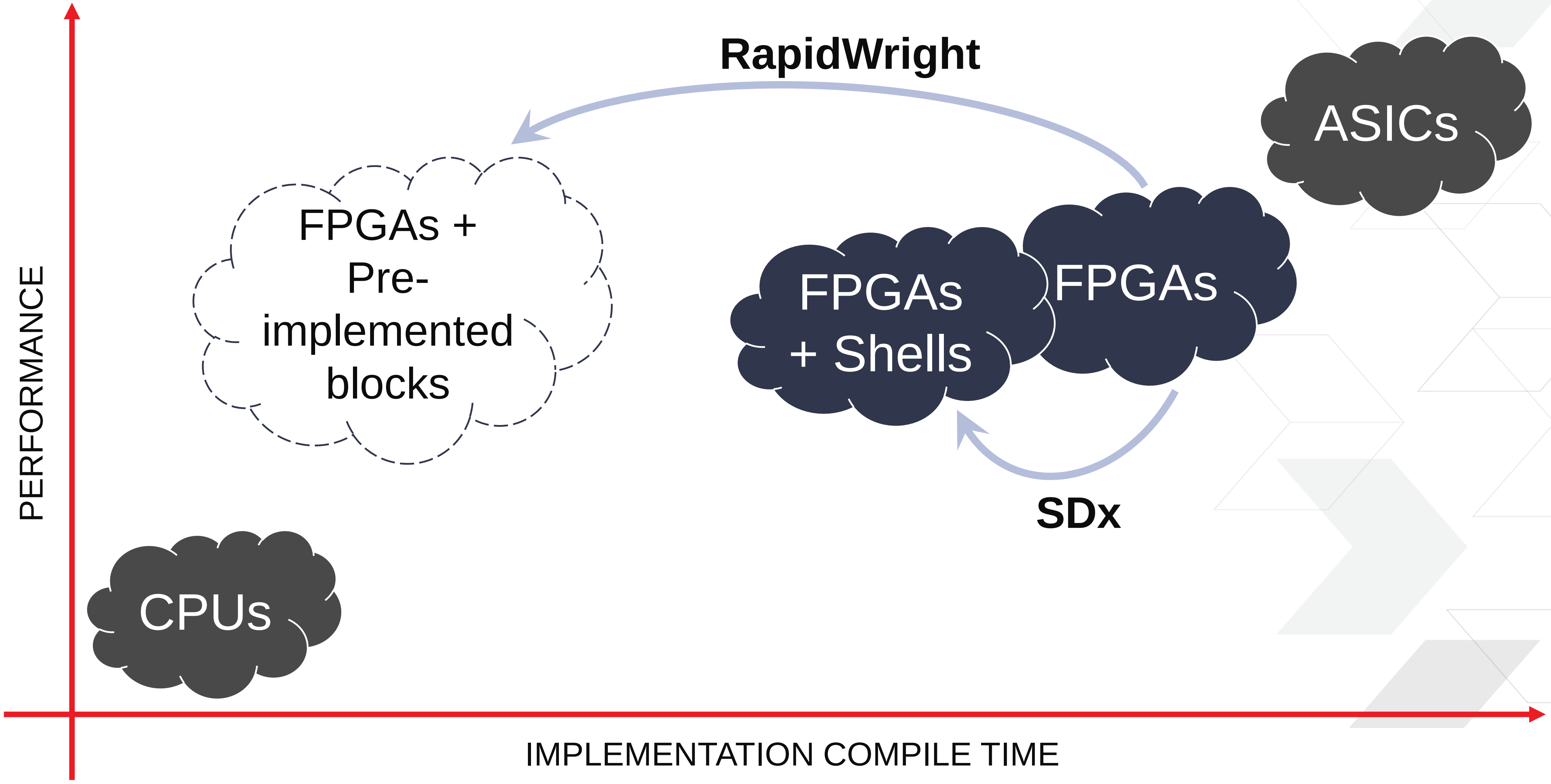
Raising the Abstraction of Design Entry



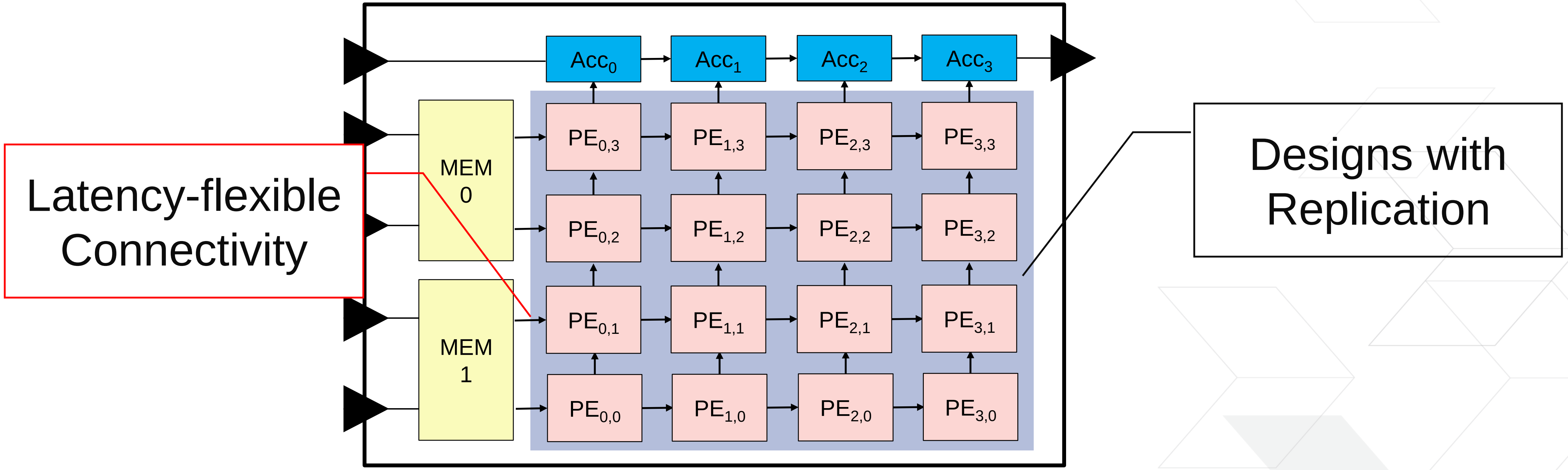
> Enabling domain specific performance optimizations

XILINX

RapidWright Value Proposition



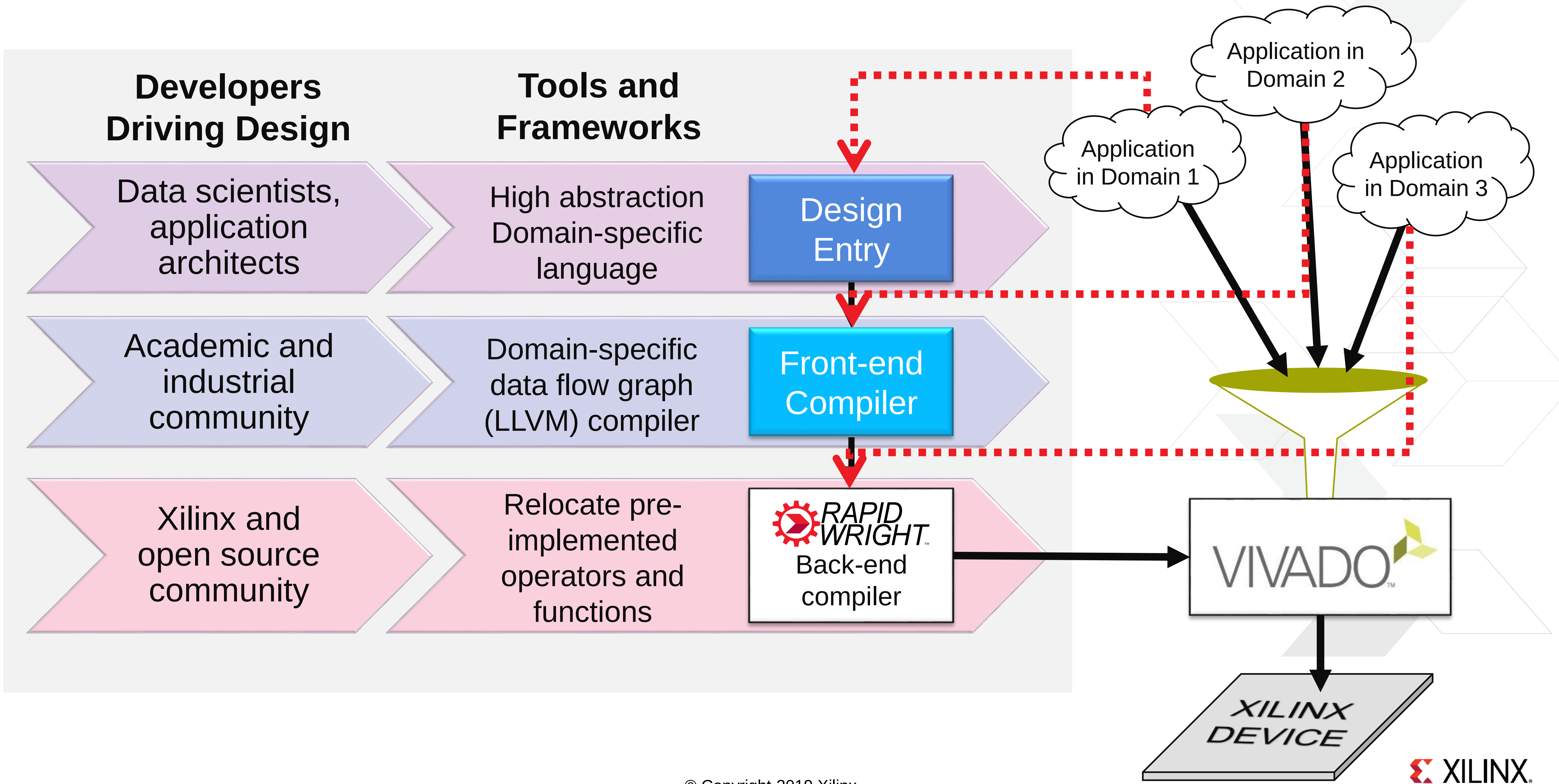
Focus on Emerging Applications



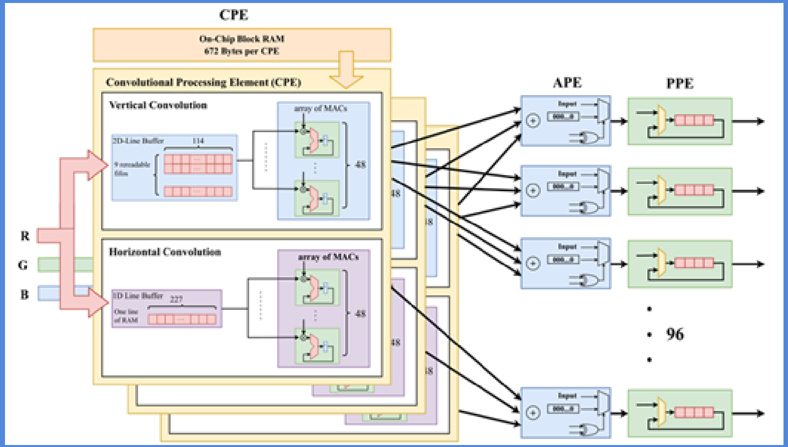
- Module-based approach to implementation
  - Lock-in performance with reusable modules
  - Fewer inter-block timing closure issues

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

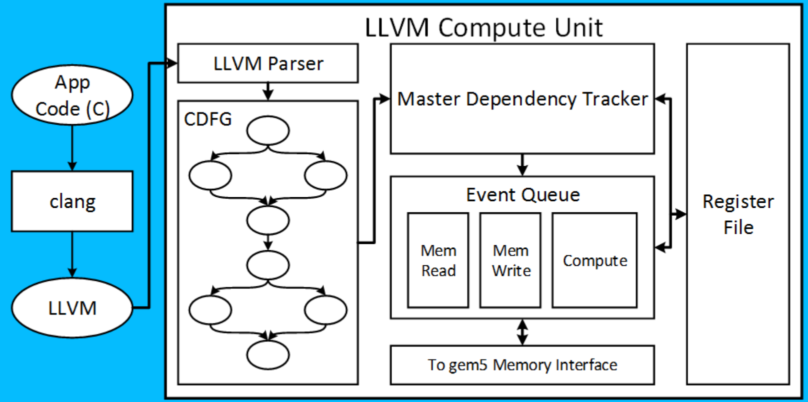
Proposed Domain-specific Tool Flows


Developers Driving Design
Tools and Frameworks
Data scientists, application architects
High abstraction Domain-specific language
Design Entry
Academic and industrial community
Domain-specific data flow graph (LLVM) compiler
Front-end Compiler
Xilinx and open source community
Relocate pre-implemented operators and functions
RAPID WRIGHT
Back-end compiler
Application in Domain 1
Application in Domain 2
Application in Domain 3
VIVADO
XILINX DEVICE

Domain Tool Flow Example



Design Entry



Front-end Compiler

RAPID WRIGHT

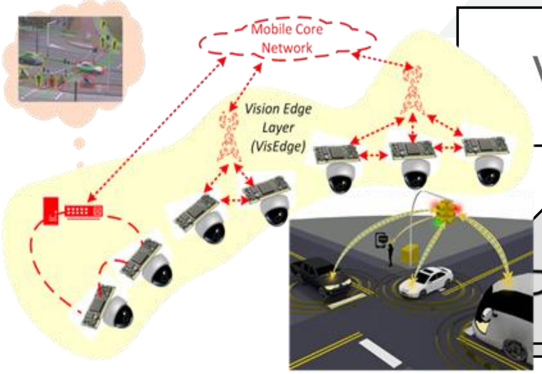
Back-end compiler

VIVADO

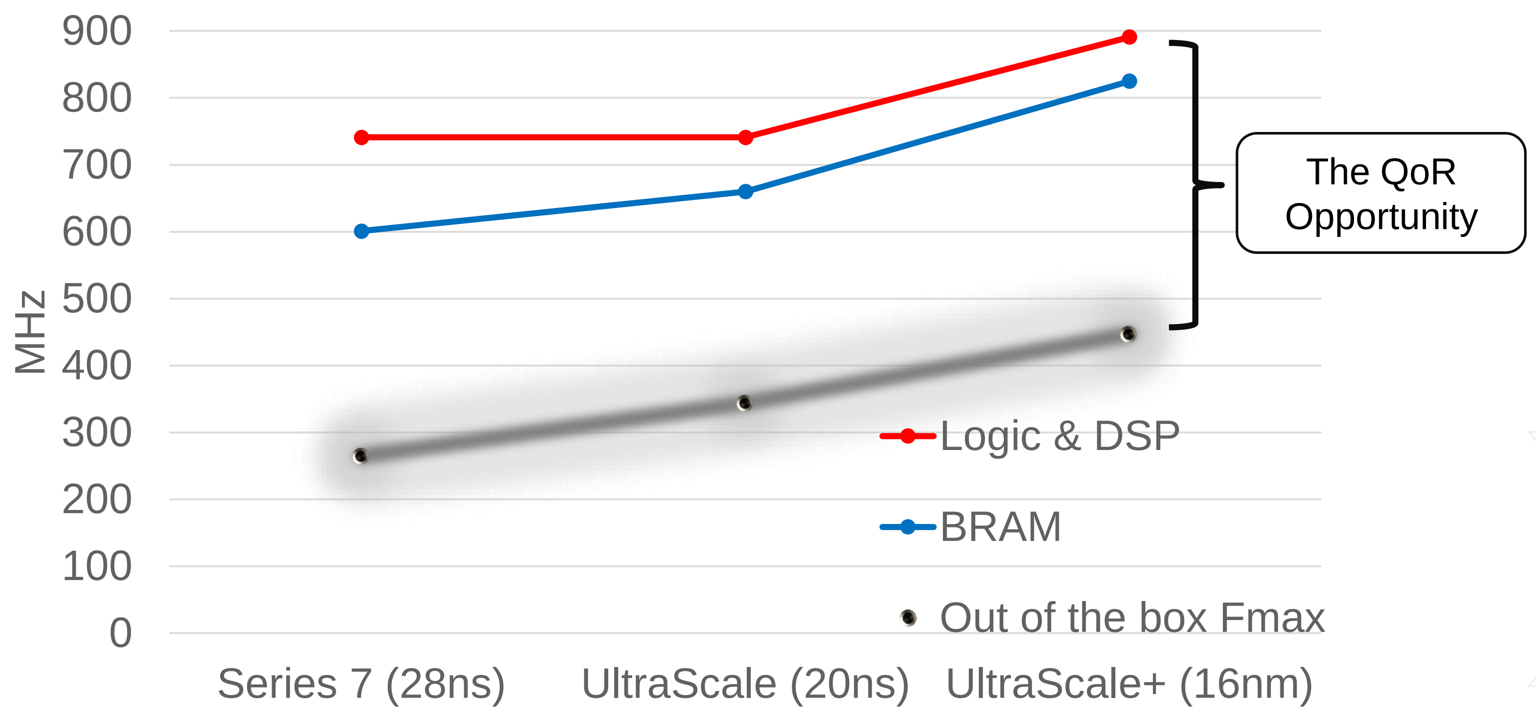
> Fact
>> Emerging domains such as surveillance or vision have high replication

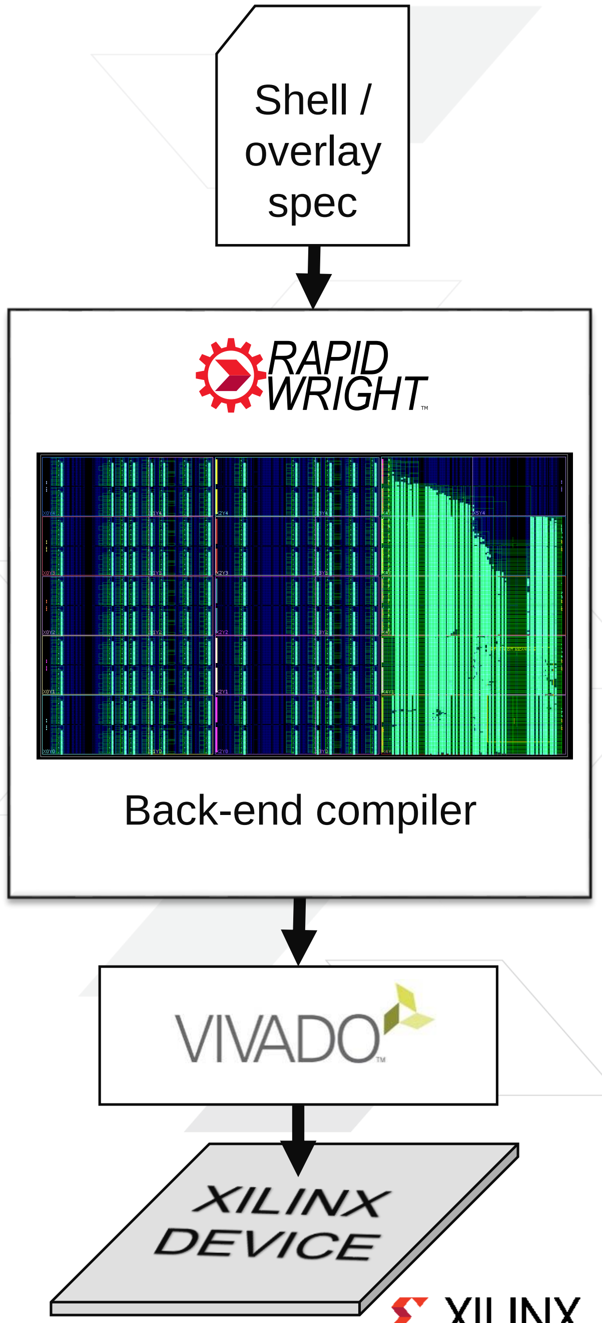
> Community role
>> Identify and extract operators and functions in the domain

> RapidWright value proposition
>> Assemble relocatable pre-implemented domain operators
>> Deliver the best inference/watt



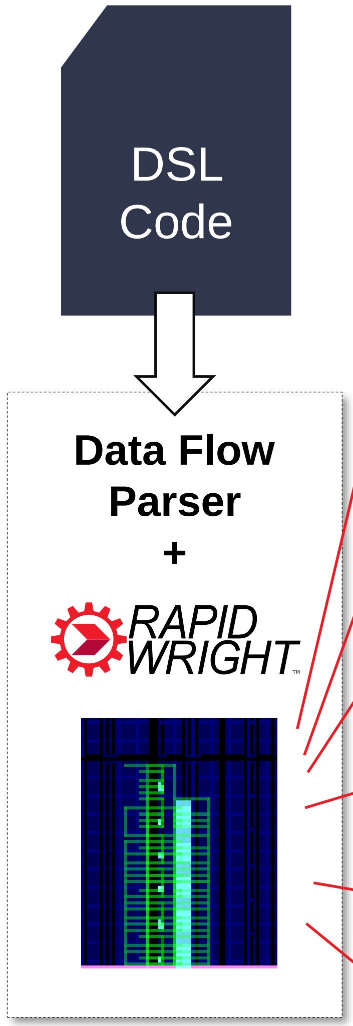
Building Relocatable Domain-specific Shells

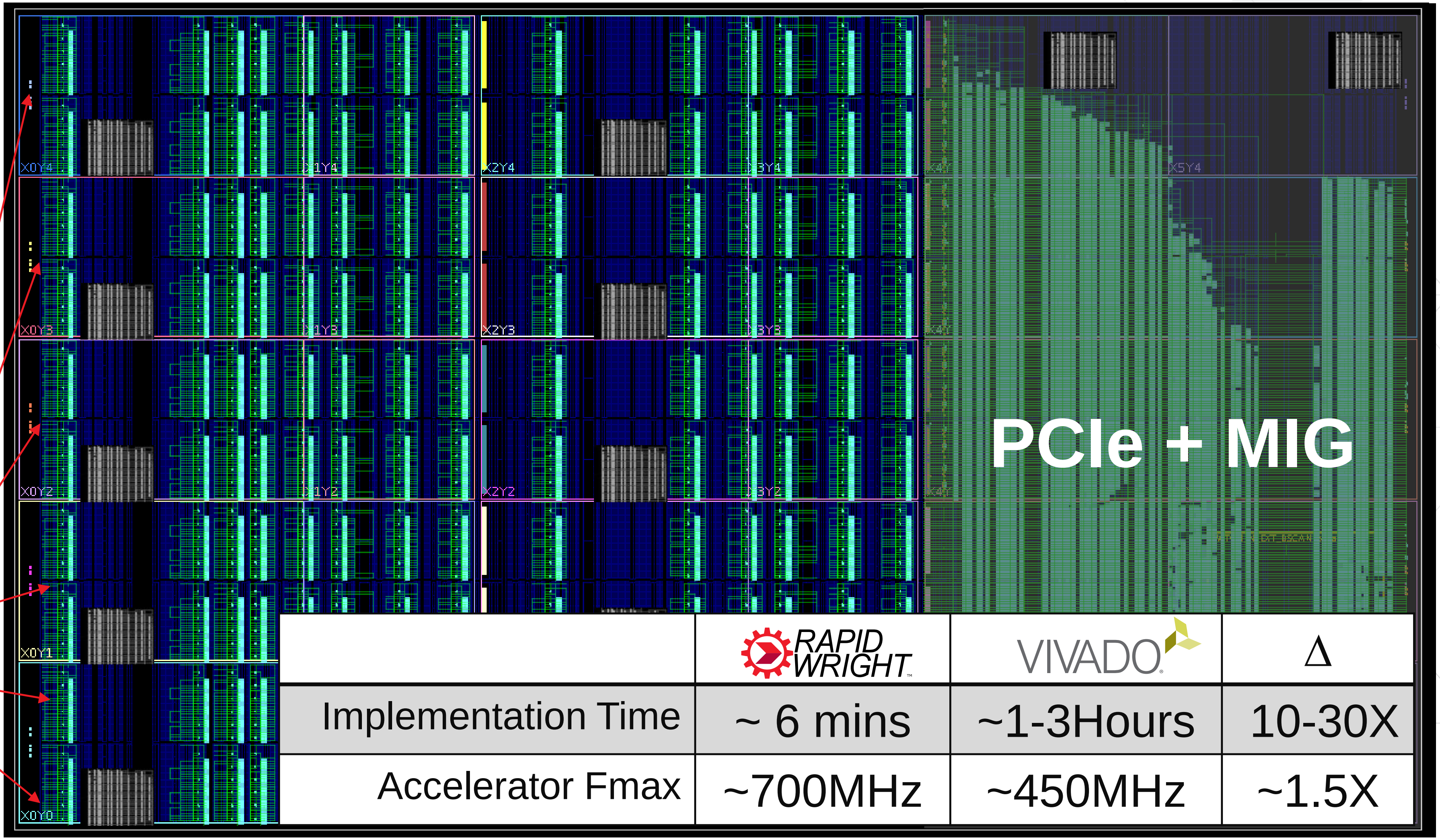




- Fact
  - Advances in silicon have created QoR opportunity
- Community role
  - Domain-specific shell design or overlays
- RapidWright value proposition
  - Achieve near-spec performance

Success Scenario: Rapid Domain-specific Assembly





	RAPIDWRIGHT	VIVADO	Δ
Implementation Time	~ 6 mins	~1-3Hours	10-30X
Accelerator Fmax	~700MHz	~450MHz	~1.5X

XILINX

What is RapidWright?

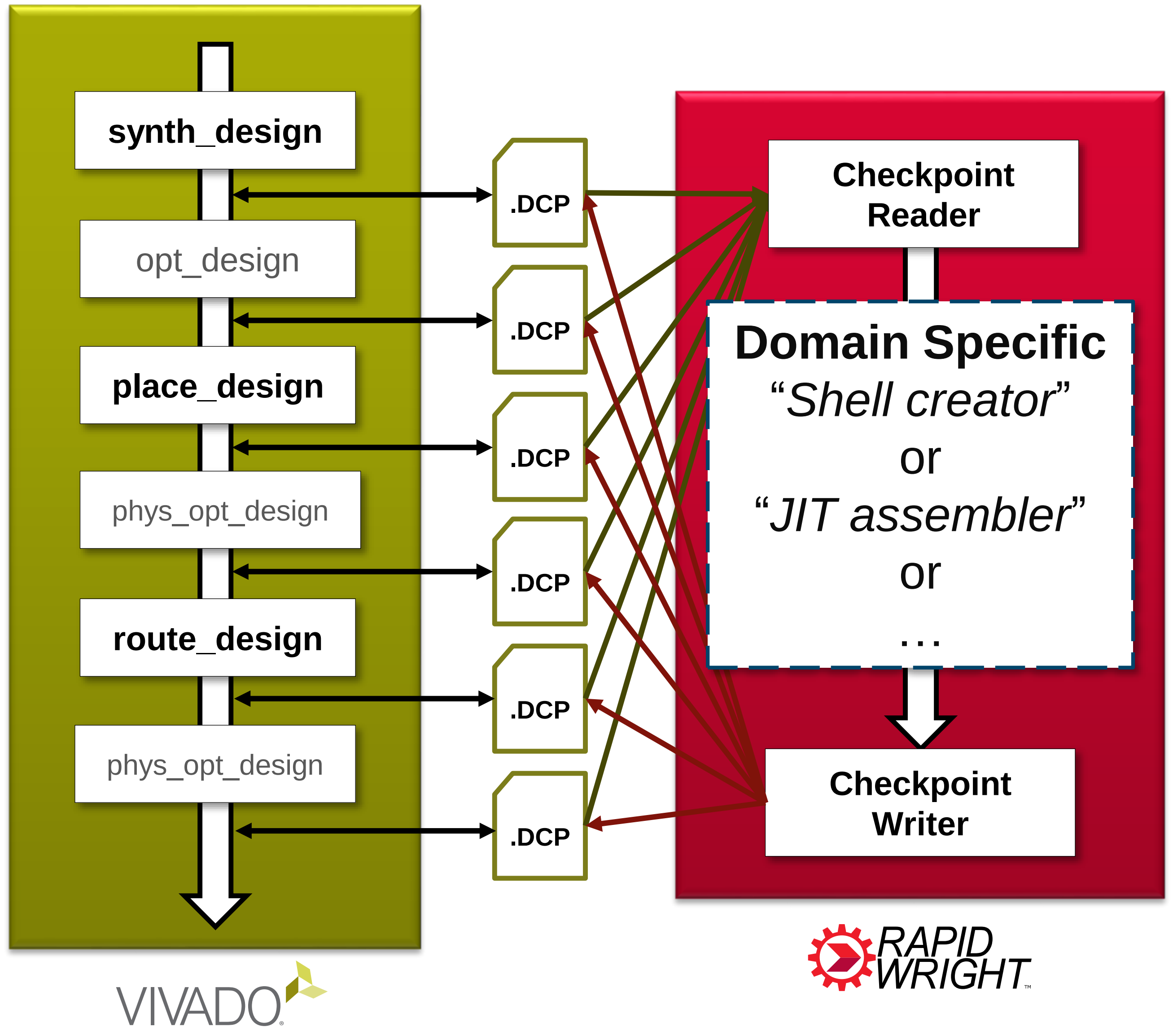
XILINX

RapidWright Overview

> **Companion framework for Vivado**
  >> Fast, light-weight, open source
  >> Communicates through Design CheckPoints[1] (DCPs)
  >> Java code, Python scripting

> **Enables targeted solutions**
  >> Reuse & relocate pre-implemented modules
  >> Just-in-time implementations
  >> Create shells & overlays

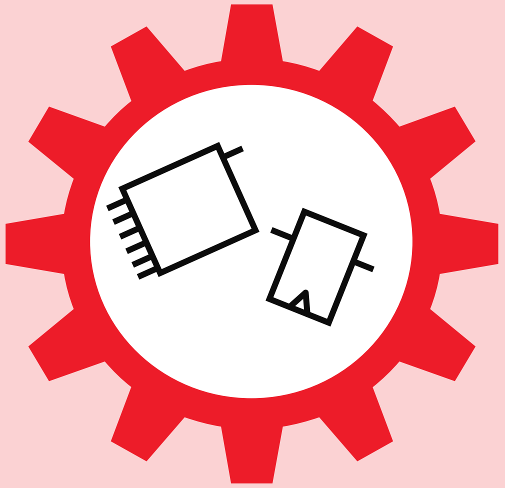
> **Power user ecosystem**
  >> Academic algorithm validation
  >> Rapid prototyping of CAD concepts



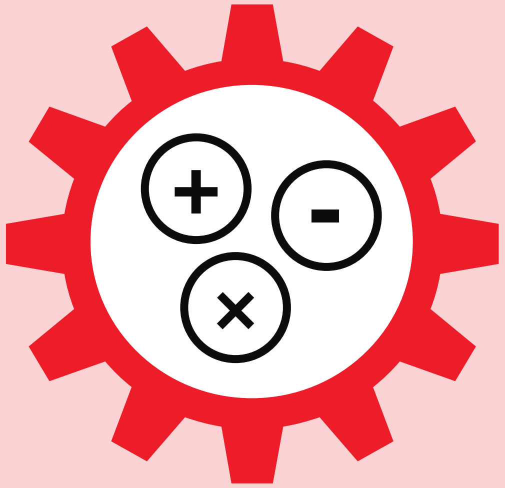
[1]DCP = netlist + P&R data + constraints

4 Ways to Design in RapidWright

BUILD ROUTED CIRCUITS



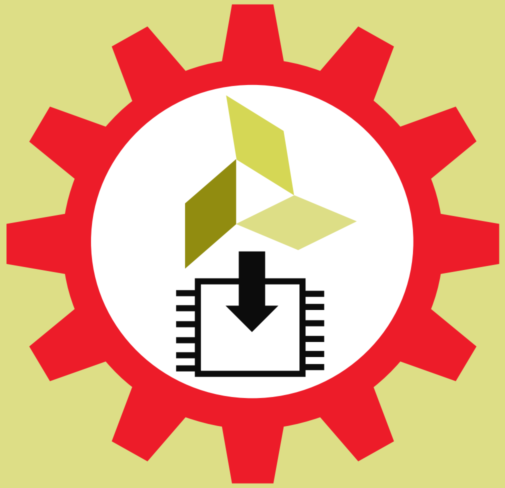
FROM SCRATCH



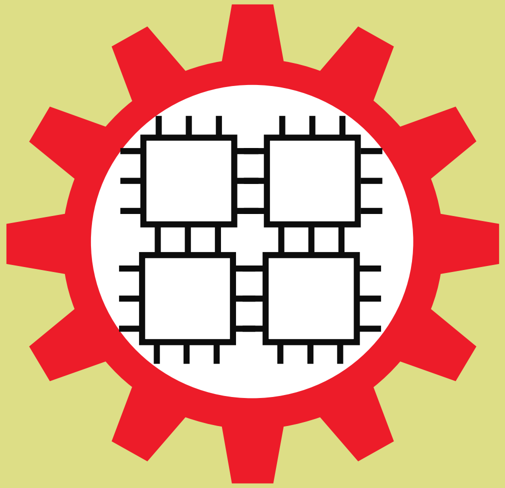
GENERATORS

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

REUSE P&R CIRCUITS



FROM VIVADO



SHELLS & OVERLAYS

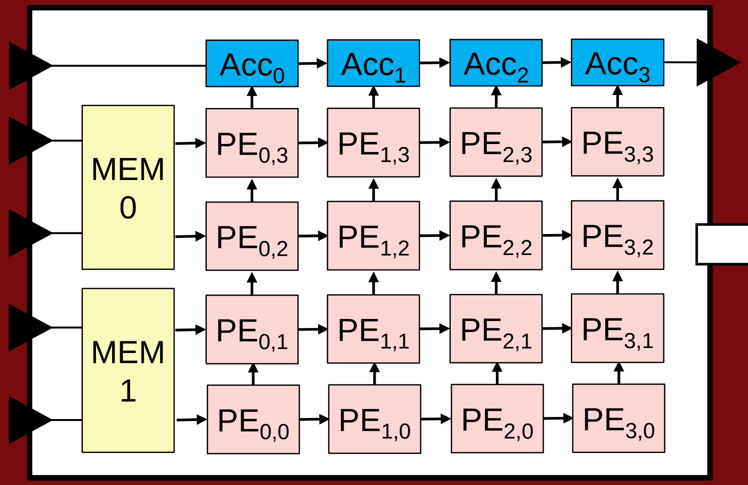
- Reuse/relocate P&R circuits from Vivado
- Combine P&R circuits together

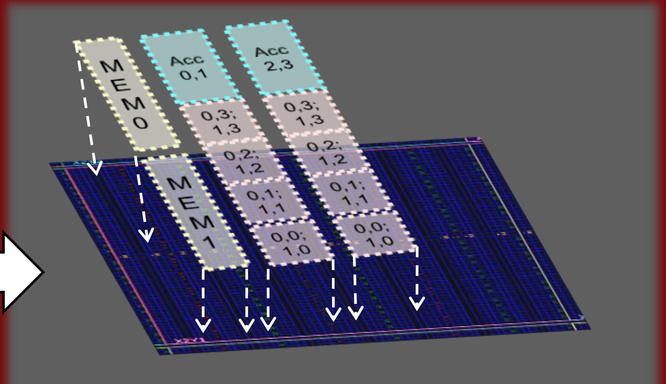
A Modular Pre-implemented Methodology

USER TASKS (MANUAL)

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

2. Placement planning



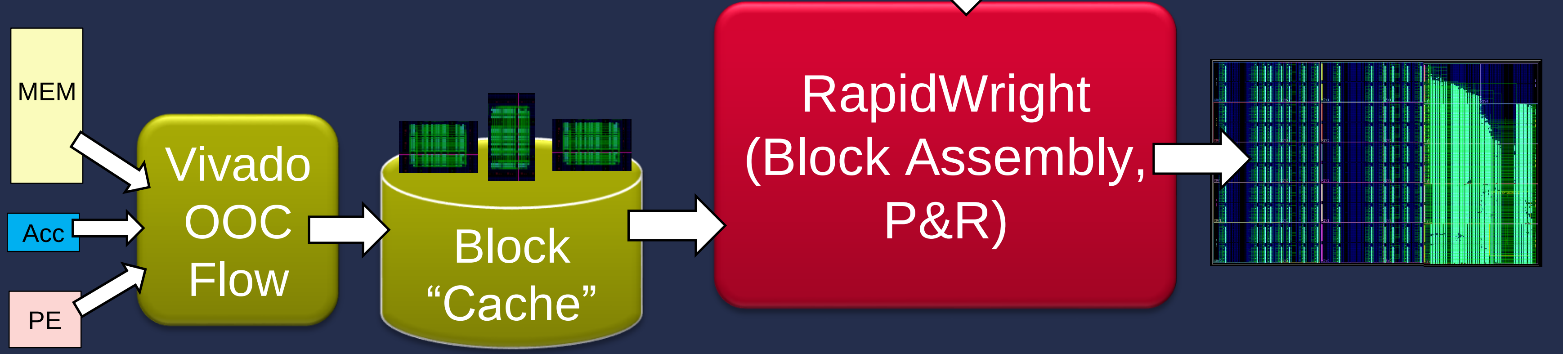


Match Design Structure to Architecture Patterns

TOOL TASKS (AUTOMATED)

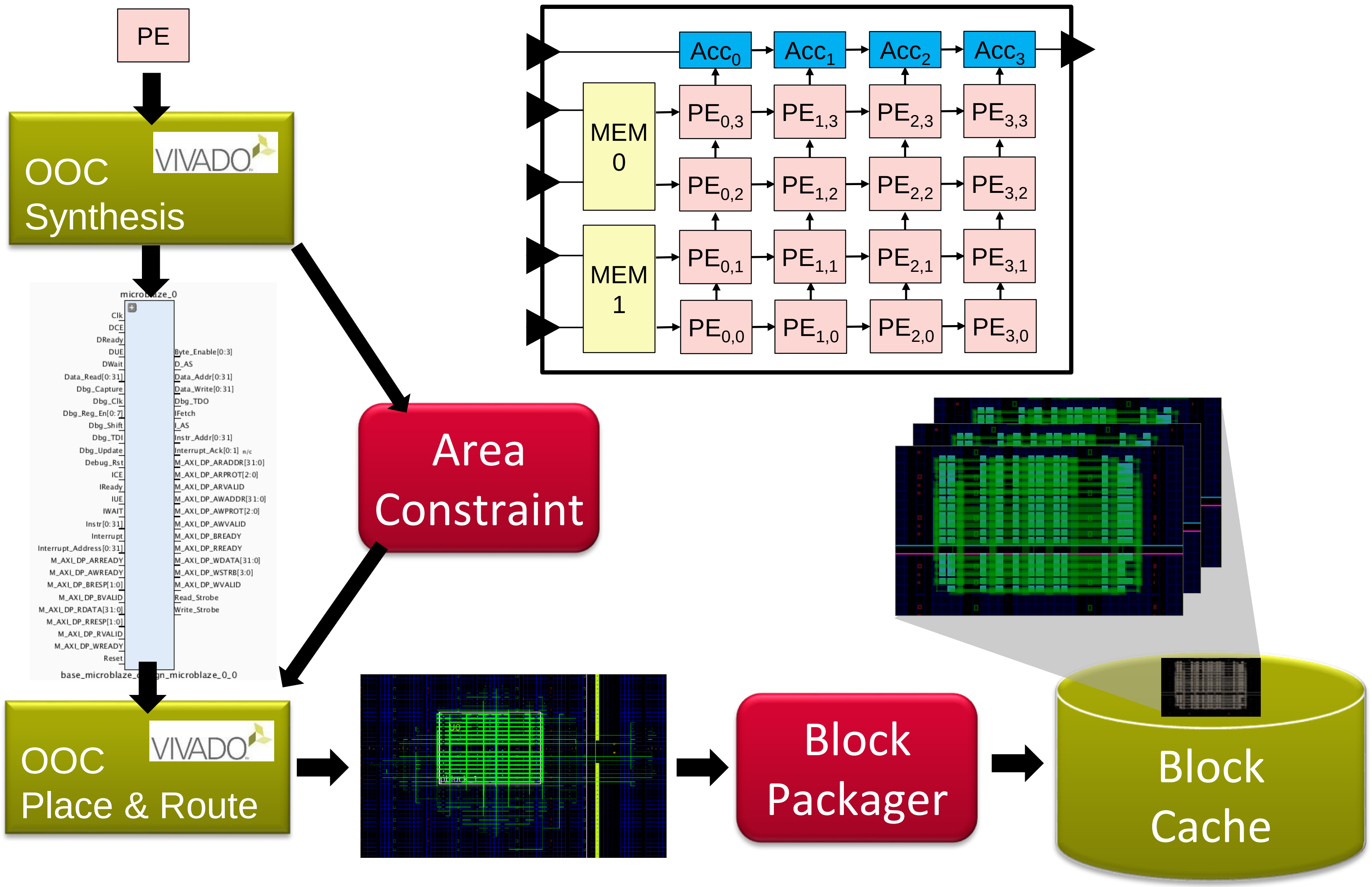
3. P&R modules cached:
   - Relocatable
   - Reusable
   - Timing predictable

4. Run implementation

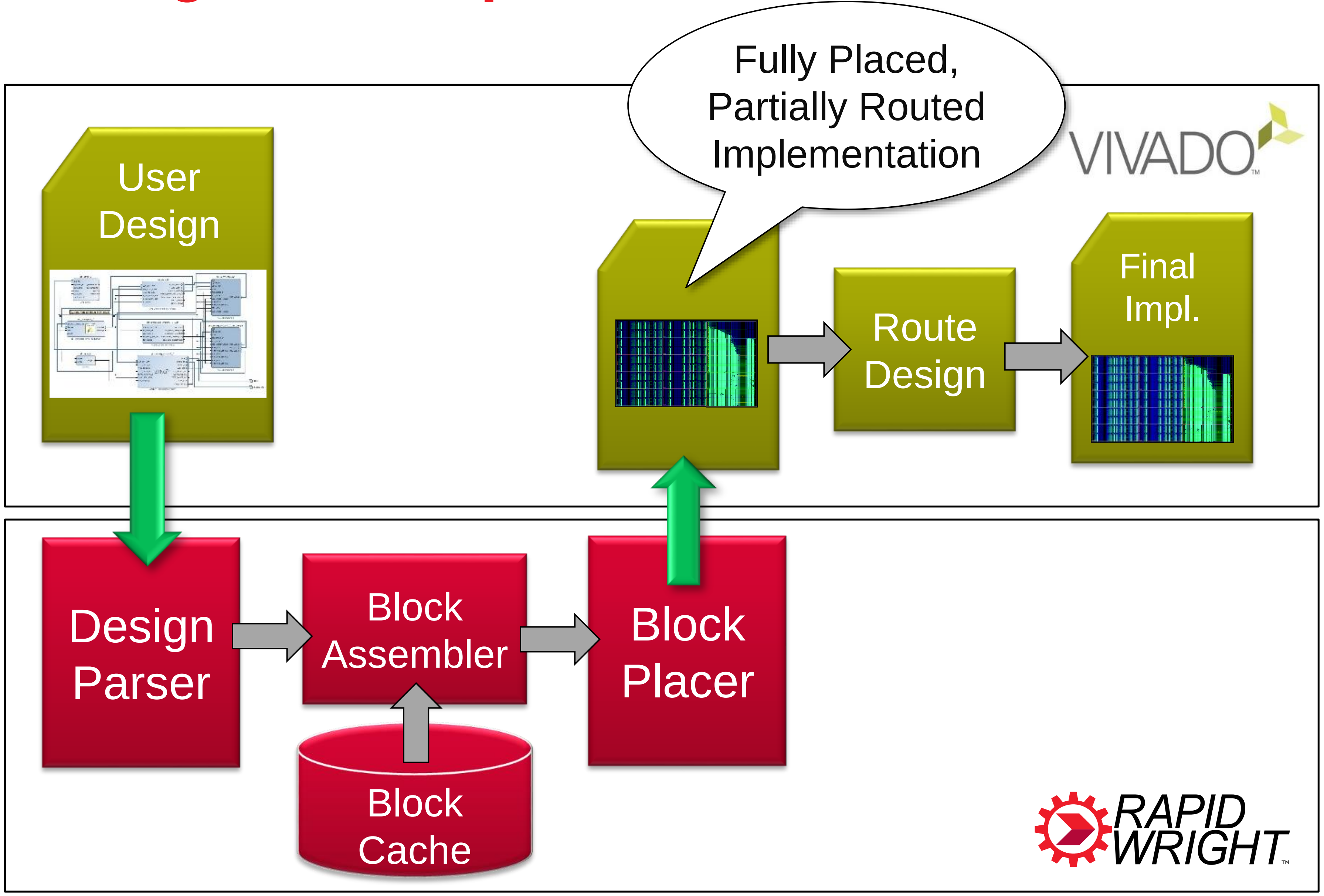


XILINX

Creating Pre-implemented Modules (Vivado OOC Flow)



RapidWright Pre-implemented Module Flow



FROM VIVADO

XILINX

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%

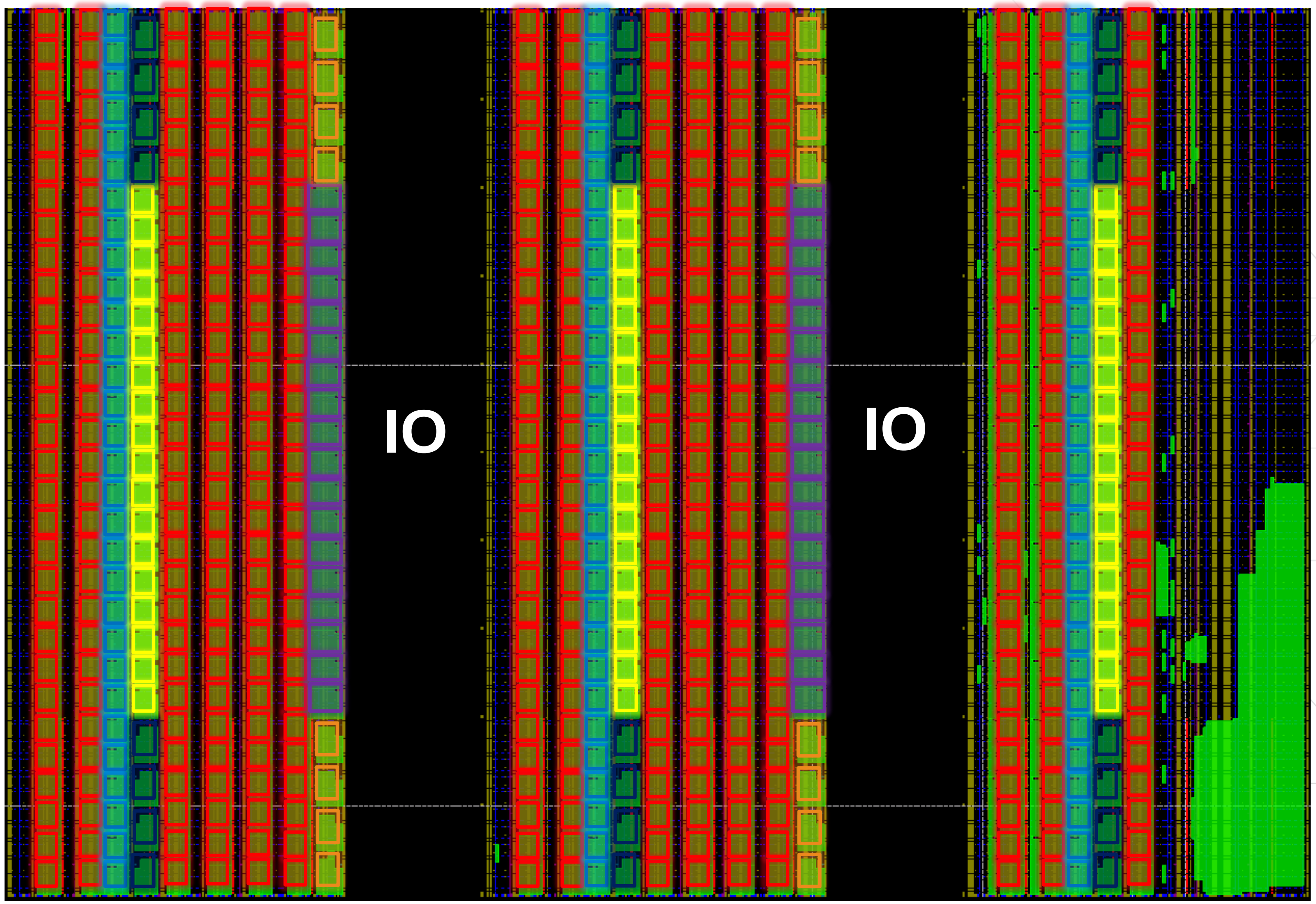
[1]RapidWright: Enabling Custom Crafted Implementations for FPGAs, FCCM 2018

Re-locatability & Reuse of Multiple Implementations

RUN	F_{MAX} (MHz)
Vivado	**270**
RapidWright	**417 (+53%)**

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

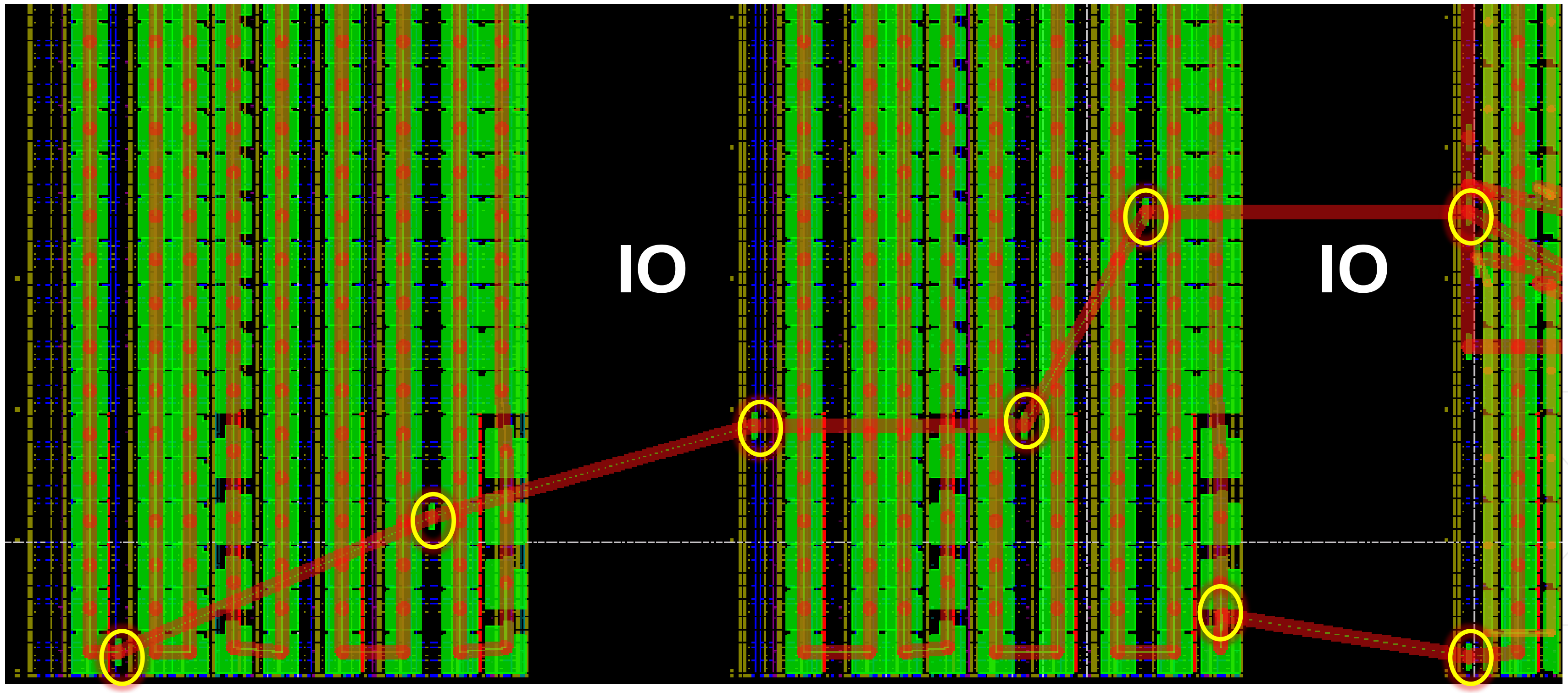
🟥 Impl #0	🟨 Impl #2	🟦 Impl #4
🟦 Impl #1	🟪 Impl #3	🟧 Impl #5



XILINX.

Latency Flexibility: AXI Stream Register Slices

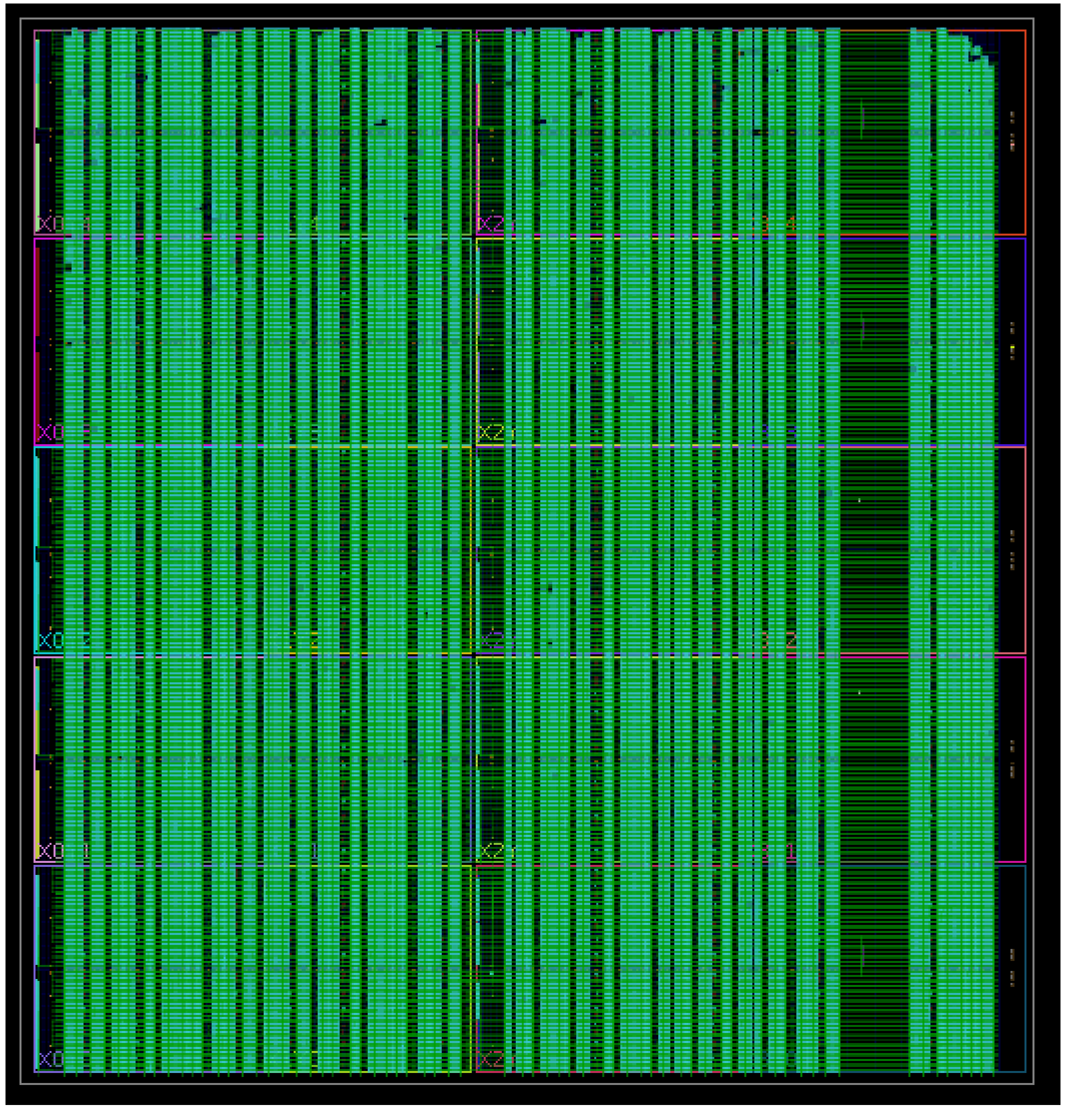
GENERATORS



> 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!



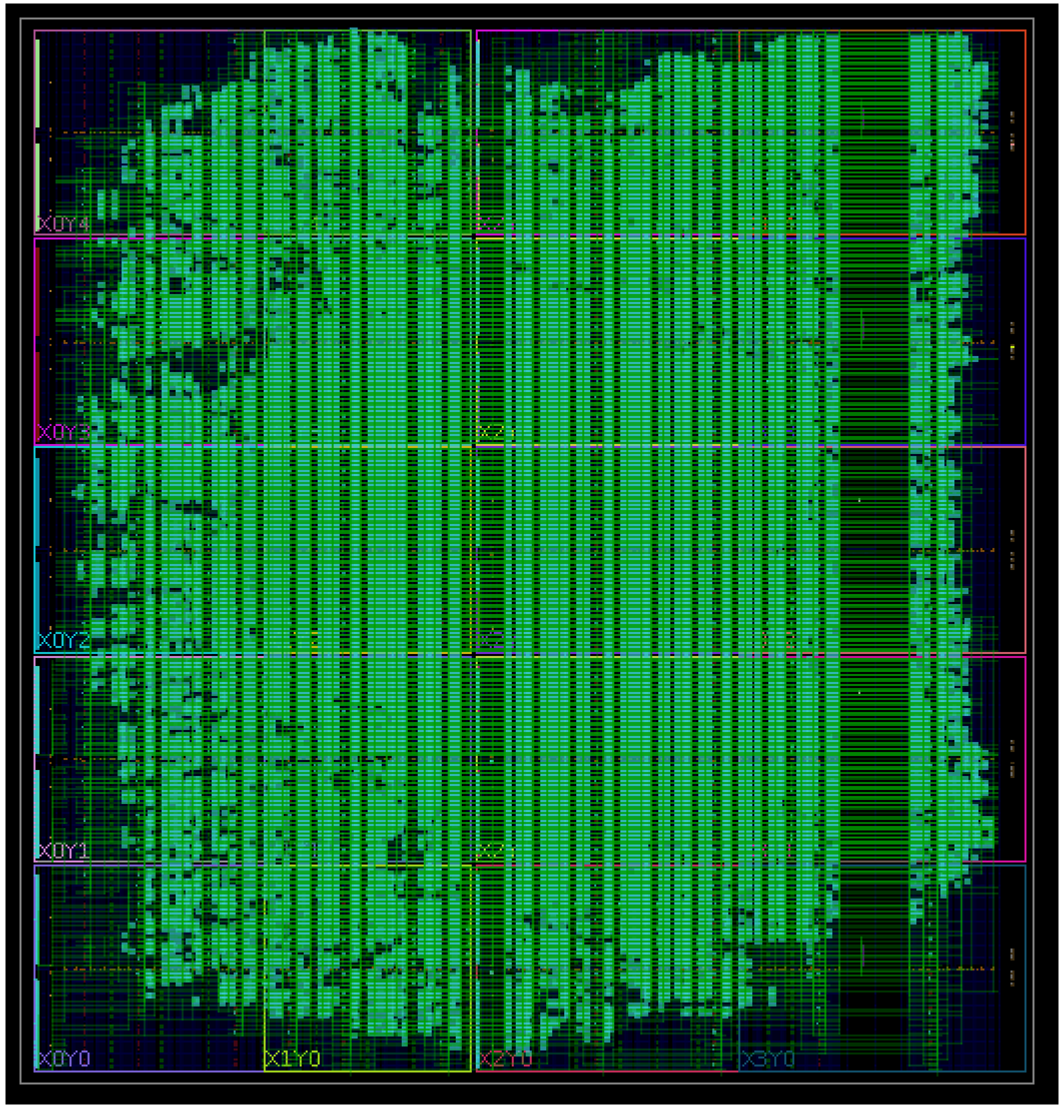
You'll need to recompile with an ILA to debug it.

I added an ILA, but the bug is gone!

Experiment: Insert Pre-implemented ILA

> Preserves existing
  >> Placement
  >> Routing

> Only occupy unused resources

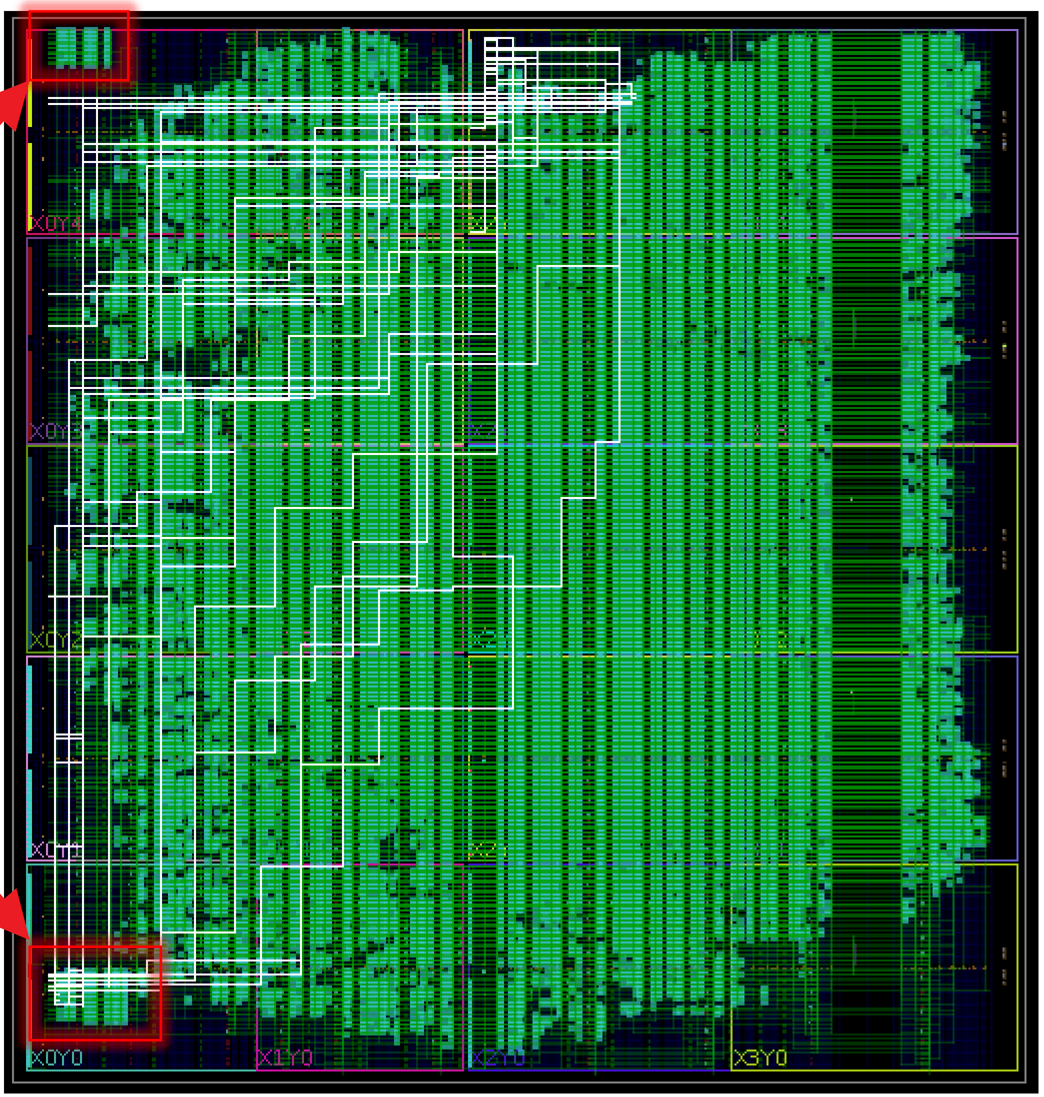


XILINX

Preserve Existing Placement & Routing

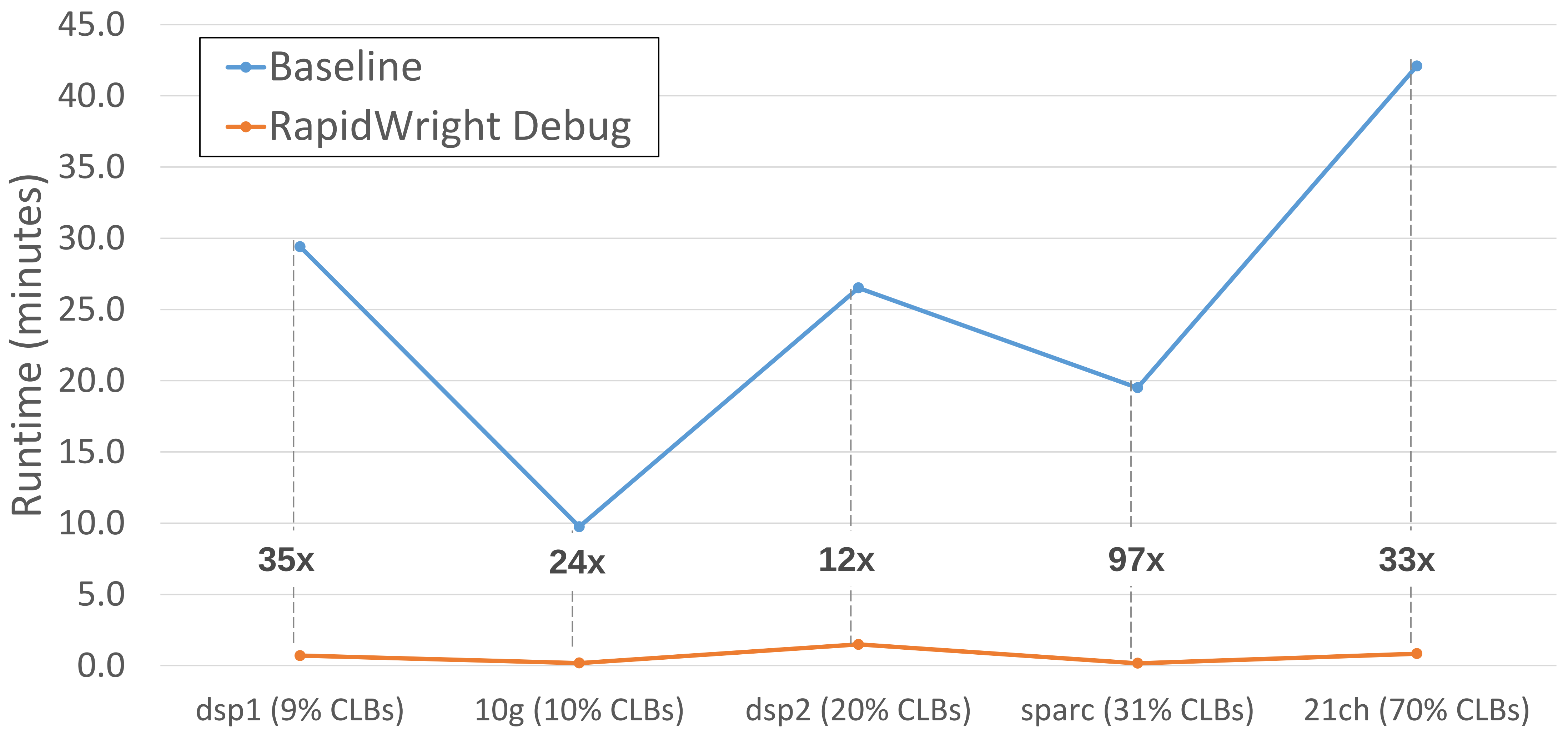
FROM VIVADO

Debug Blocks Inserted by RapidWright



XILINX

Debug Instrumentation Speedup



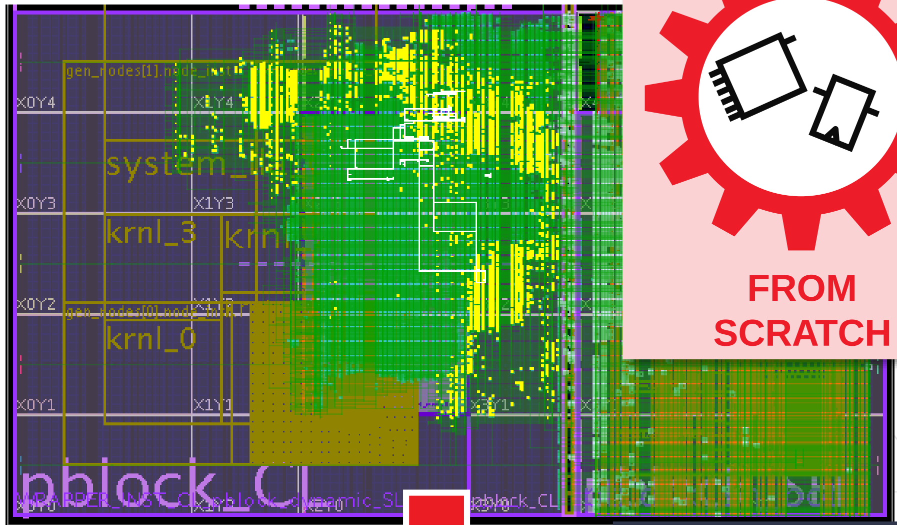
XILINX.

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

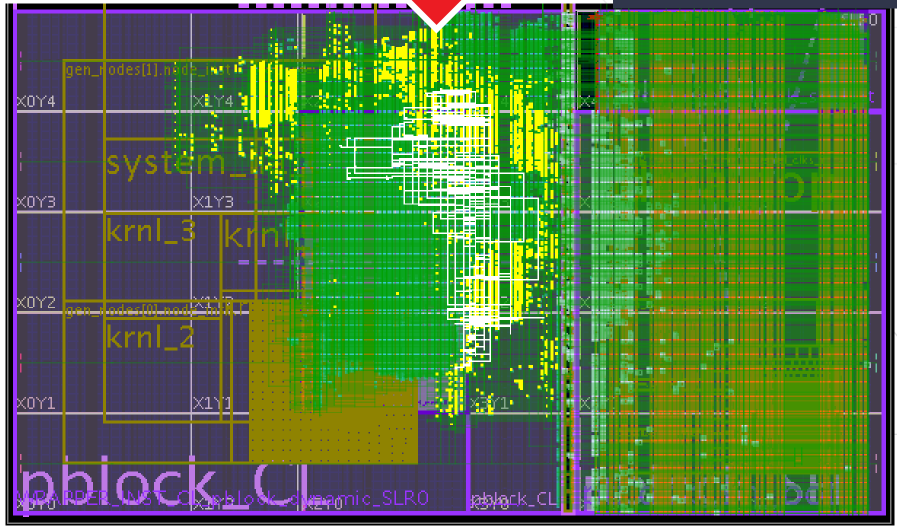
Vivado modify_debug_probes	**RapidWright** ProbeRouter	Δ
130 mins	6.3 mins	21X

FROM SCRATCH



Original

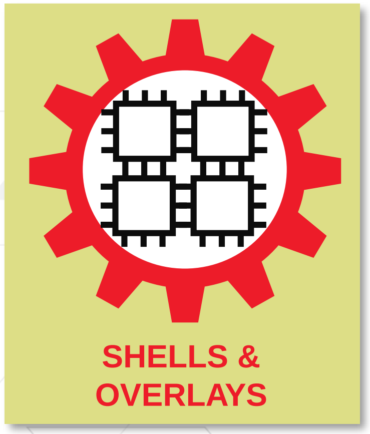
🟨 ILA Cells
— Probe Routes



RapidWright Probes Rerouted

XILINX

Pre-implemented Data Movement Shell

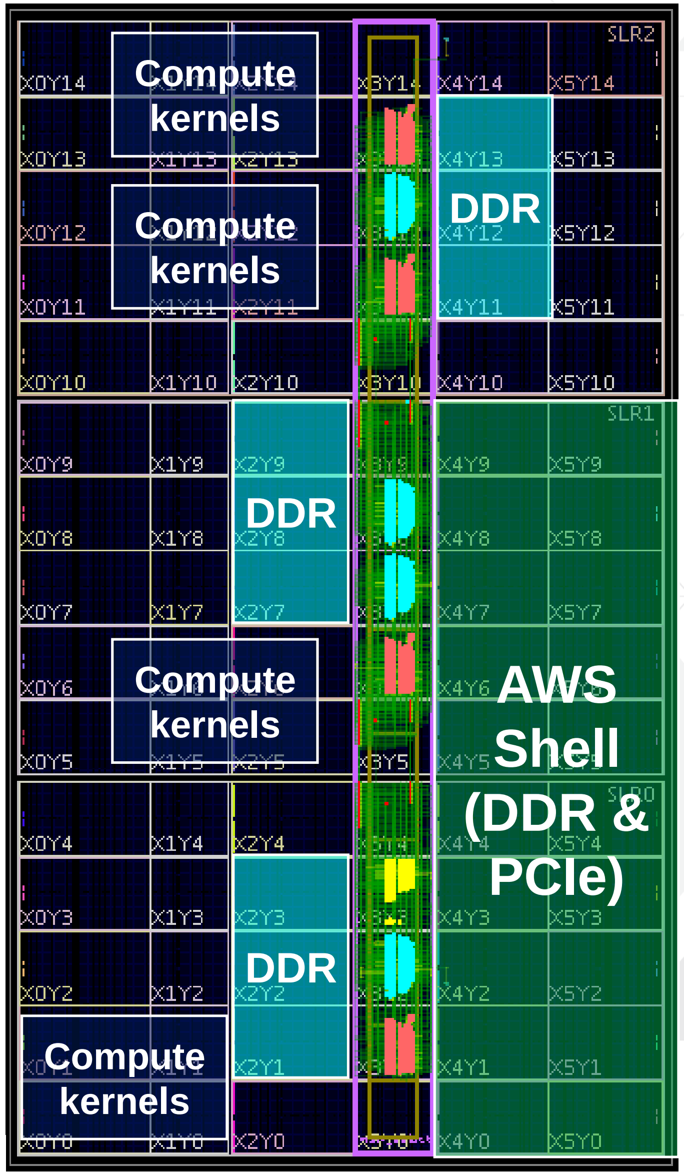


- Goals
  - Minimize overhead of compute (and overlays)
  - Prove shell assembly model

- Build-to-order LinkBlaze[1] shell
  - 512 bit, bi-directional
  - RapidWright Pre-implemented modules

Vivado	RapidWright
516MHz	620MHz (+20%)

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

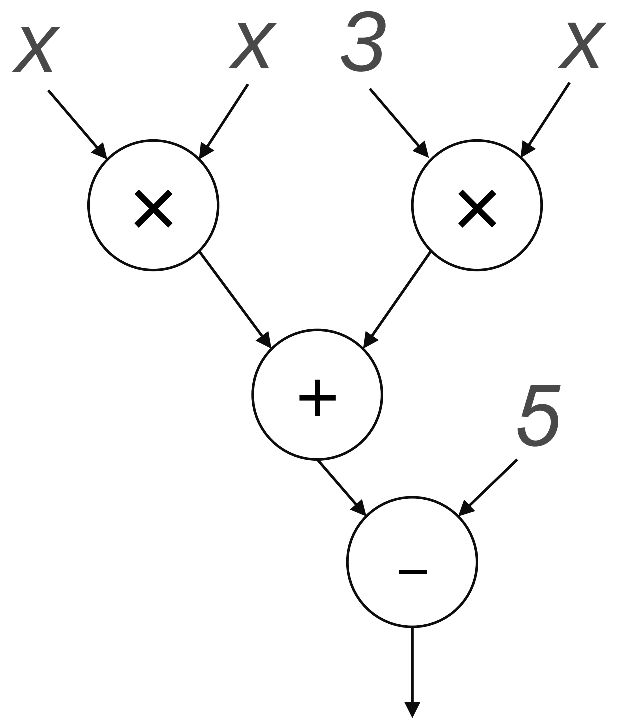


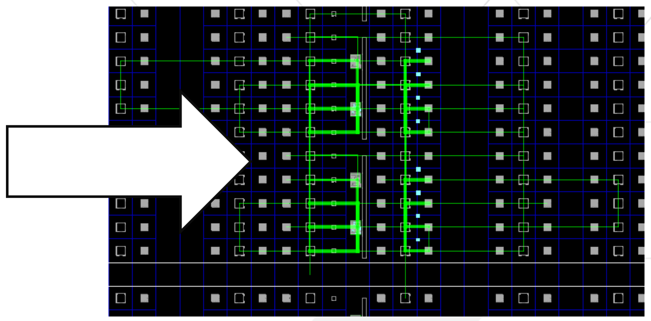
Just-in-time, Circuit Module Generators

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





XILINX

RapidWright SLR Crossing DCP Creator

GENERATORS

- SLR crossing module from scratch
  - >> Parameterizable
  - >> Closes timing at 760MHz
    - – Clk Period: 1.313ns
  - >> Routed clock, placed and routed
  - >> Runs in seconds

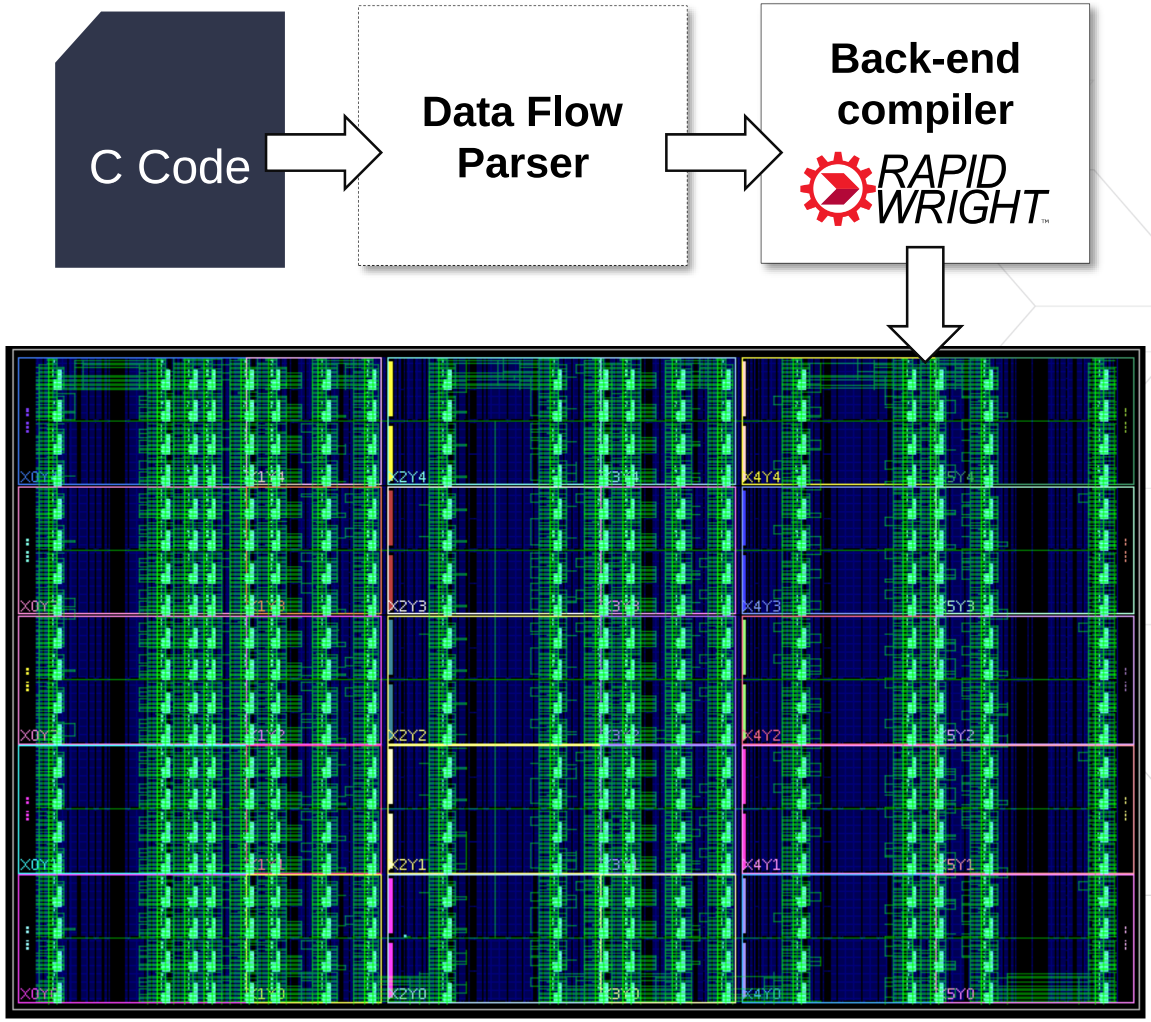
```
=================================================================
==                          SLR Crossing DCP Generator
=================================================================
This RapidWright program creates a placed and routed DCP that car
imported into UltraScale+ designs to aid in high speed SLR crossi
RapidWright documentation for more information.

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

XILINX.

Ongoing Work: C Code to Full Chip Accelerator in Seconds

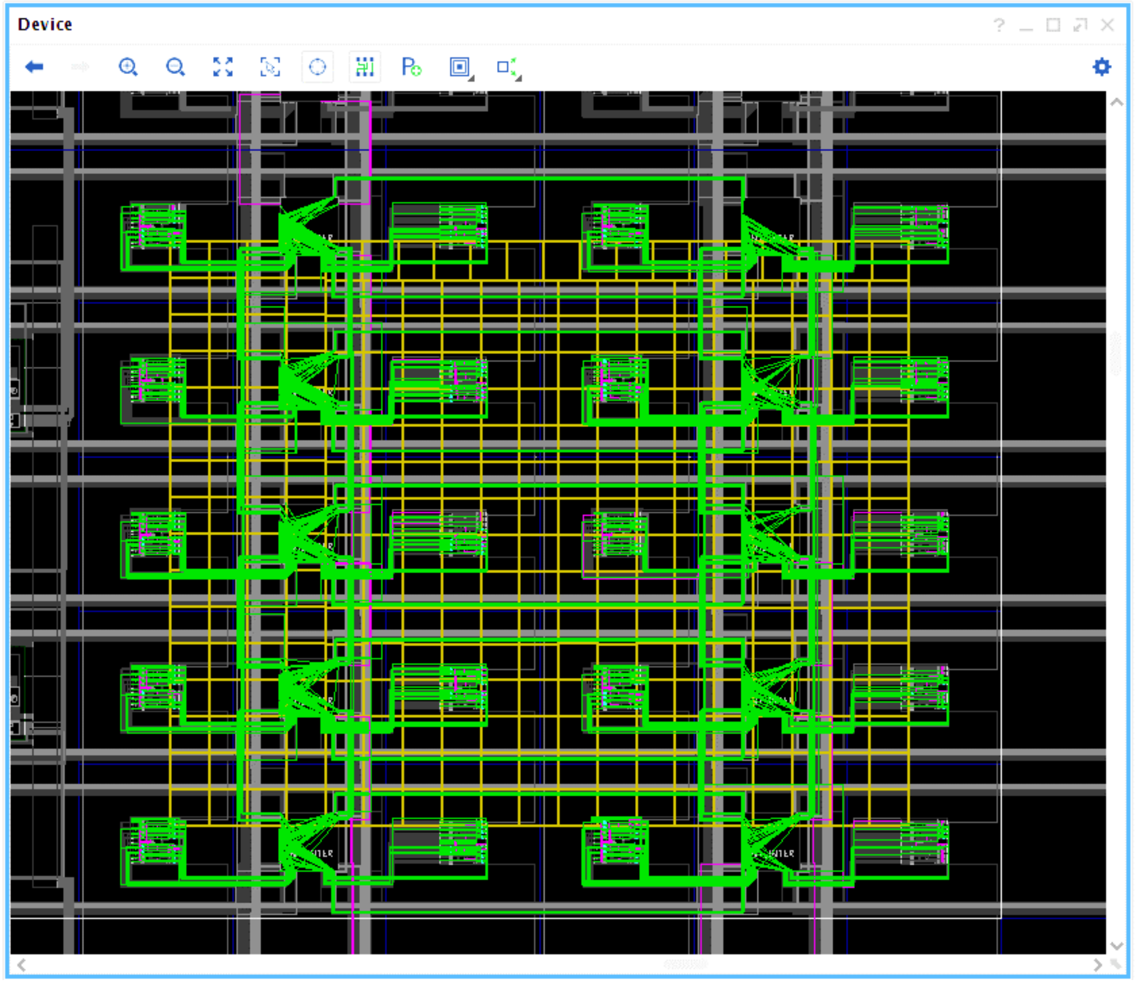
> **RapidWright generator capabilities**
  - UltraScale+ VU3P, 100% DSP utilization
  - Front-end C code parser still in development
  - Prototype back-end flow
  - Runs in seconds (37 seconds)
  - Achieves spec frequency (775 MHz)

> **Future integration work:**
  - SLR crossing generator - target 750 MHz
  - LinkBlaze (data movement) solution



Leveraging Algorithmic Engines

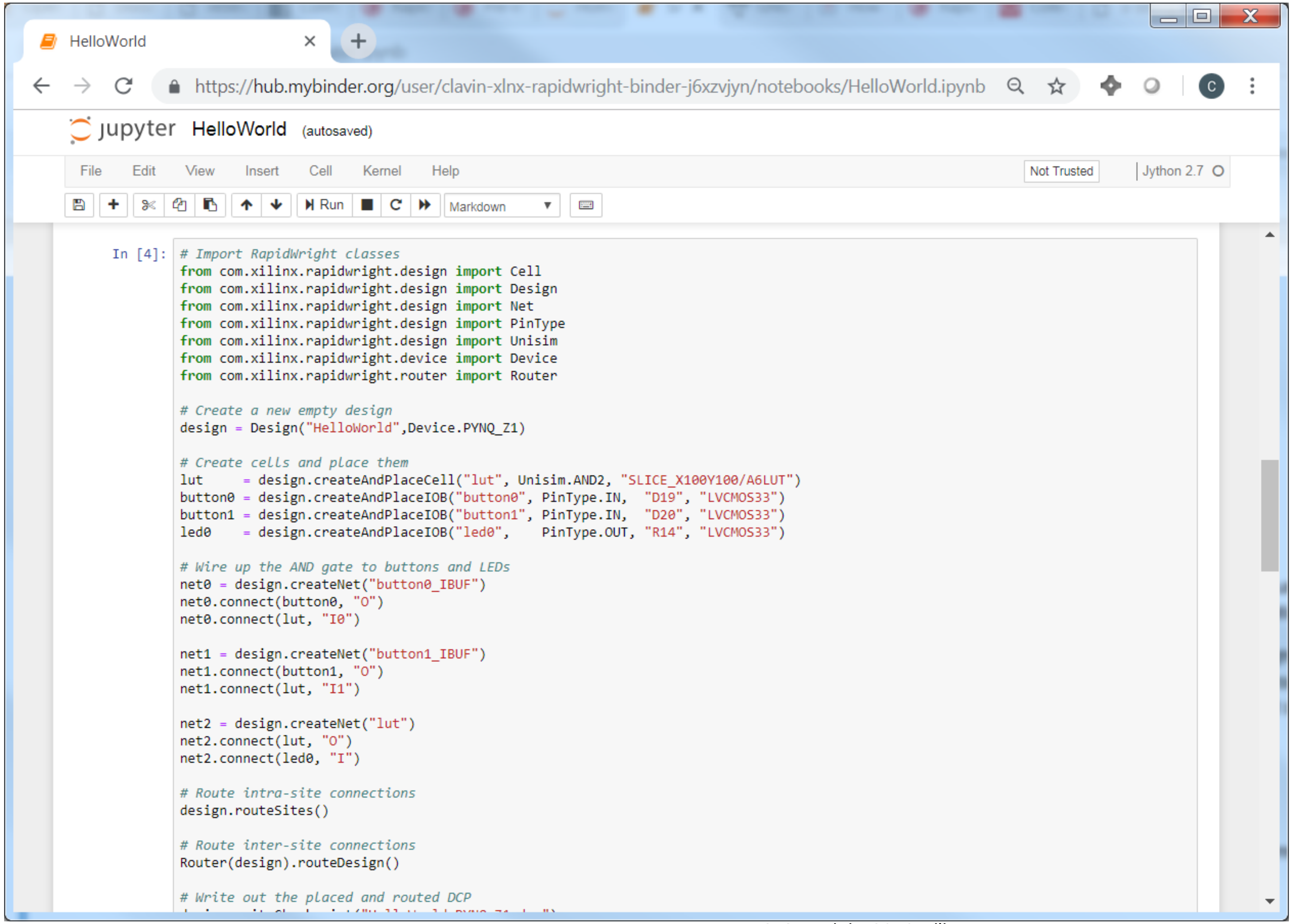
- SAT Solver
  - Resolve difficult, localized congestion routing
    - Finds solutions where Vivado cannot
  - RapidWright front-end to SAT solver engine[1]
- Future Work
  - Simultaneous SAT placement and routing solution
  - ILP Solvers
    - Potential for placement solutions



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

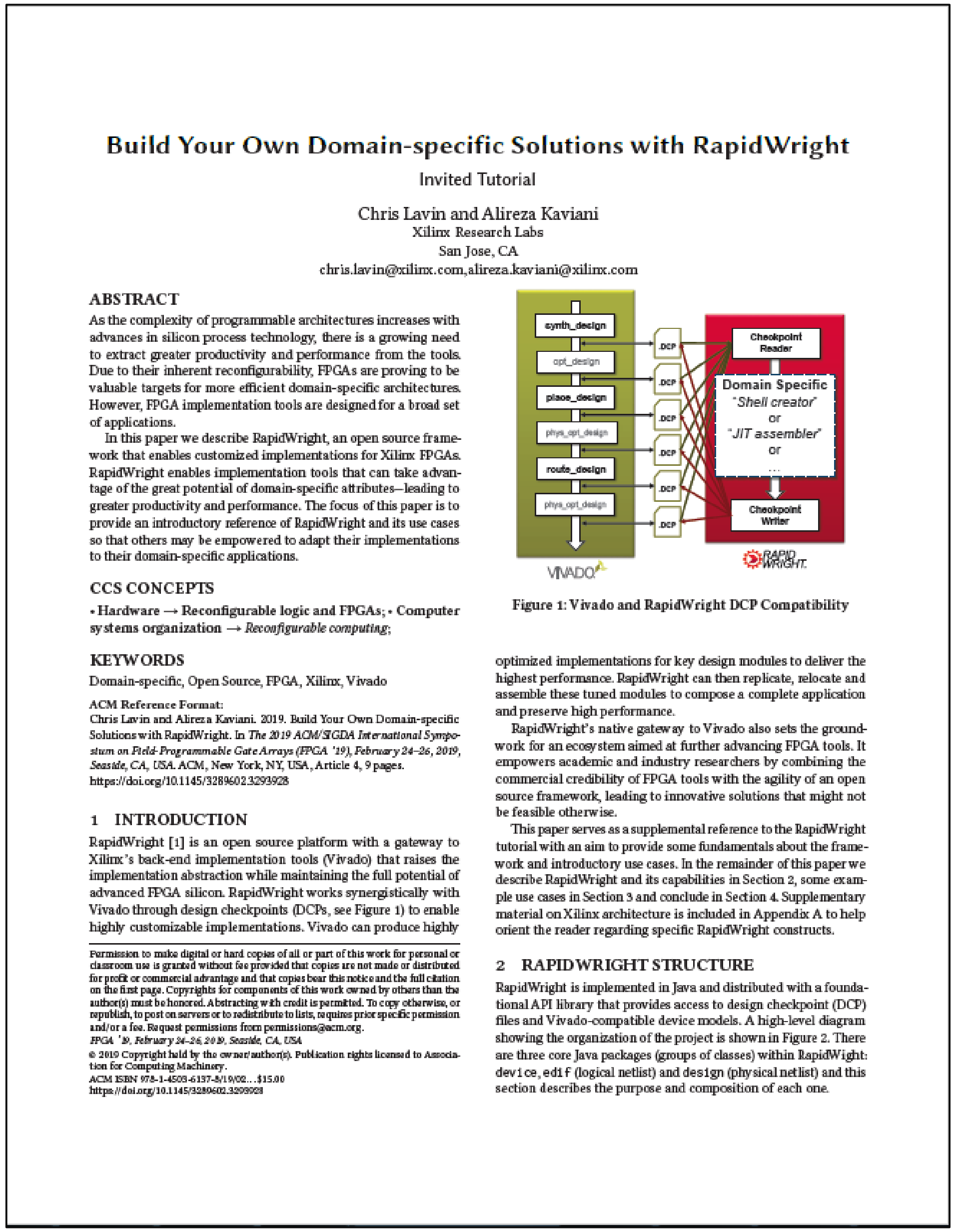
How do I get started with RapidWright?

Run RapidWright in Your Browser


HelloWorld
https://hub.mybinder.org/user/clavin-xlnx-rapidwright-binder-j6xzvjyn/notebooks/HelloWorld.ipynb
jupyter HelloWorld (autosaved)
File Edit View Insert Cell Kernel Help
Not Trusted
Jython 2.7
Run
Markdown
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
```

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/10.1145/3289602.3293928

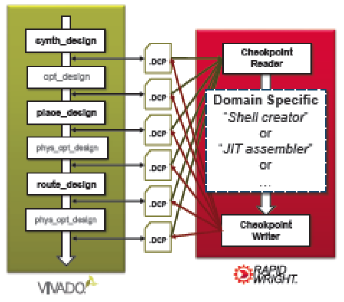


Figure 1: Vivado and RapidWright DCP Compatibility

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 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`, `edif` (logical netlist) and `design` (physical netlist) and this section describes the purpose and composition of each one.

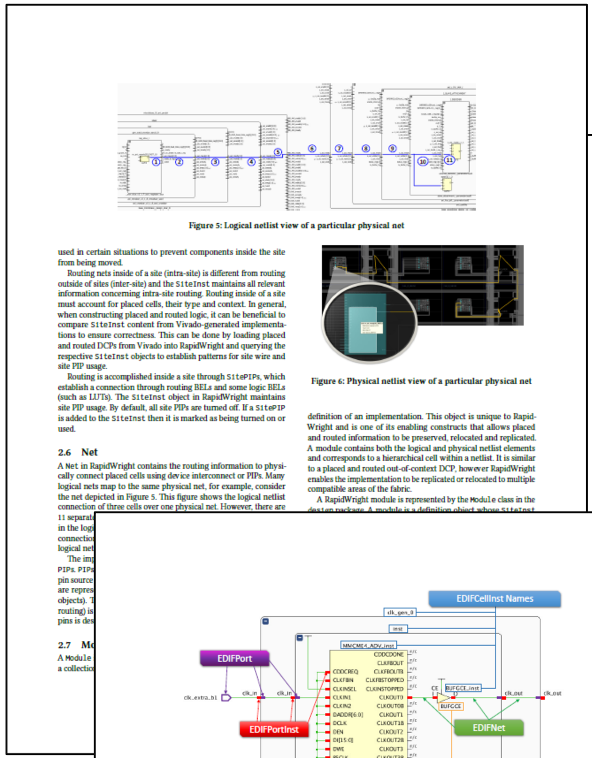


Figure 5: Logical netlist view of a particular physical net

Figure 6: Physical netlist view of a particular physical net

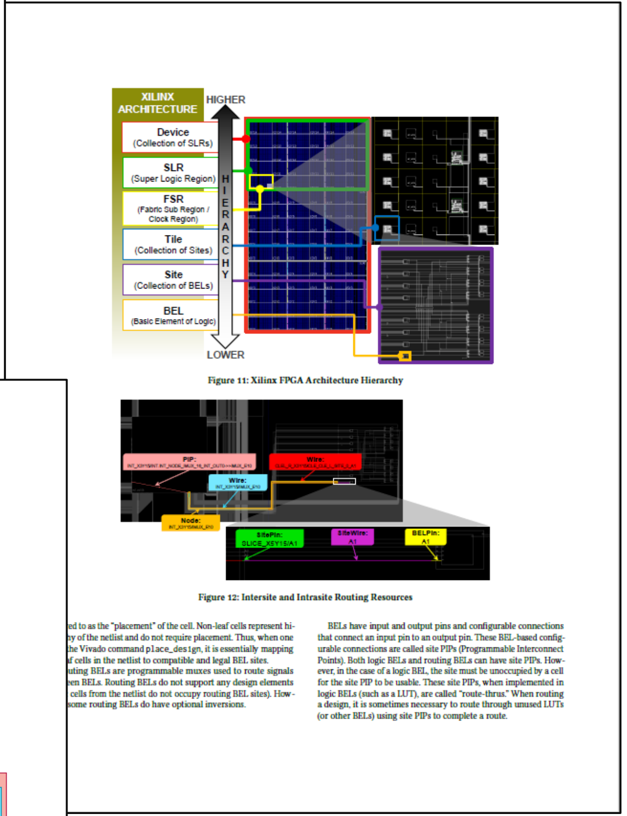


Figure 11: Xilinx FPGA Architecture Hierarchy

Figure 12: Intersite and Intrasite Routing Resources

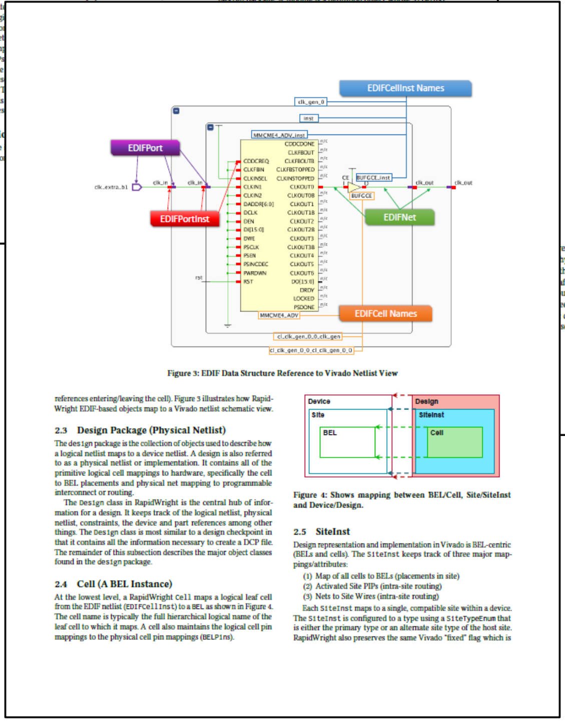
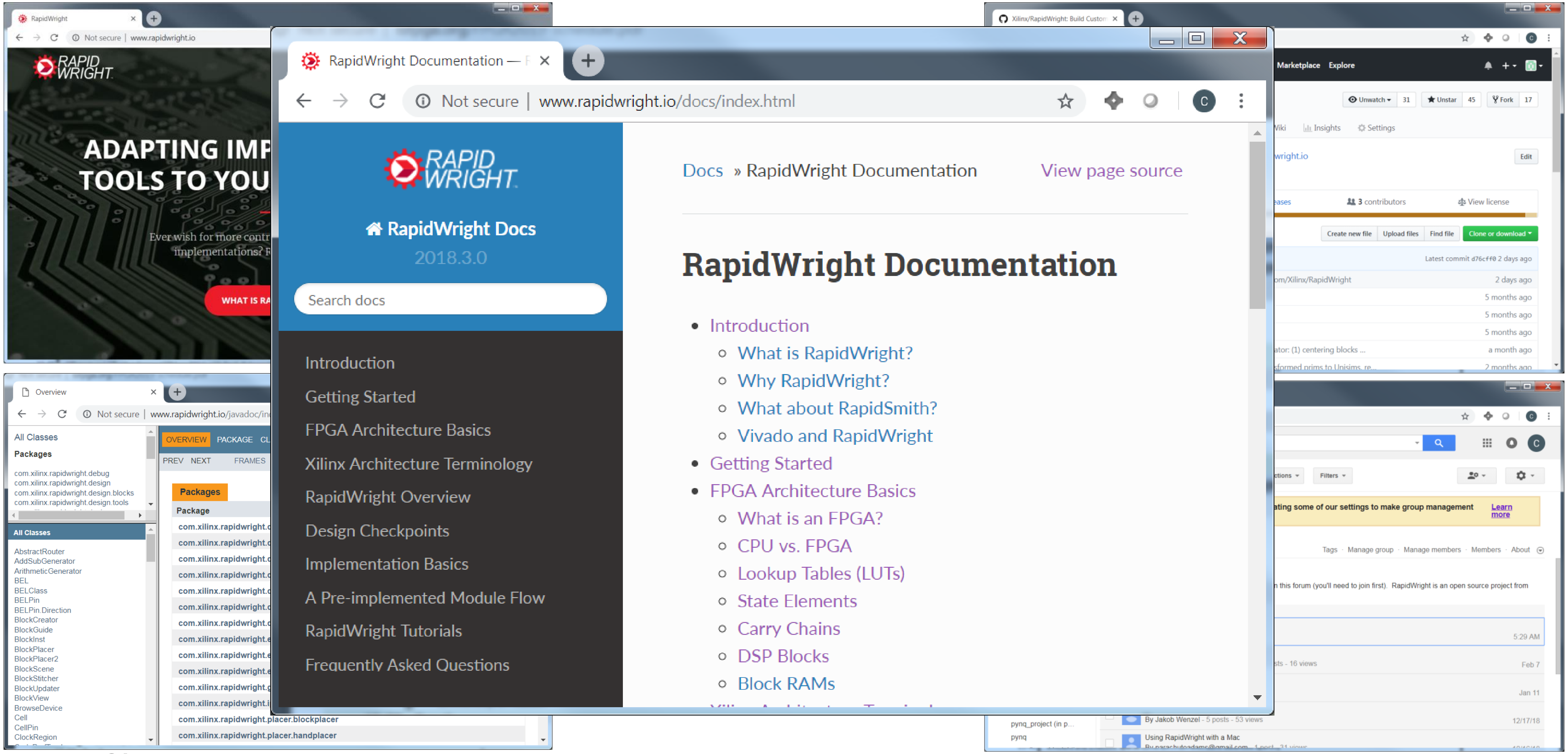


Figure 3: EDIF Data Structure Reference to Vivado Netlist View

Figure 4: Shows mapping between BEL/Cell, Site/SiteInst and Device/Design.

...

RapidWright Resources: www.rapidwright.io



RapidWright Documentation — R

Not secure | www.rapidwright.io/docs/index.html

RAPID WRIGHT

RapidWright Docs

2018.3.0

Search docs

Introduction
Getting Started
FPGA Architecture Basics
Xilinx Architecture Terminology
RapidWright Overview
Design Checkpoints
Implementation Basics
A Pre-implemented Module Flow
RapidWright Tutorials
Frequently Asked Questions

Docs » RapidWright Documentation   View page source

RapidWright Documentation

- Introduction
  - What is RapidWright?
  - Why RapidWright?
  - What about RapidSmith?
  - Vivado and RapidWright
- Getting Started
- FPGA Architecture Basics
  - What is an FPGA?
  - CPU vs. FPGA
  - Lookup Tables (LUTs)
  - State Elements
  - Carry Chains
  - DSP Blocks
  - Block RAMs

RapidWright

Not secure | www.rapidwright.io

ADAPTING IMP
TOOLS TO YOU

Ever wish for more contr
implementations? F

WHAT IS RA

Overview

Not secure | www.rapidwright.io/javadoc/in

All Classes
Packages
com.xilinx.rapidwright.debug
com.xilinx.rapidwright.design
com.xilinx.rapidwright.design.blocks
com.xilinx.rapidwright.design.tools

All Classes
AbstractRouter
AddSubGenerator
ArithmeticGenerator
BEL
BELClass
BELPin
BELPin Direction
BlockCreator
BlockGuide
BlockInst
BlockPlacer
BlockPlacer2
BlockScene
BlockStitcher
BlockUpdater
BlockView
BrowseDevice
Cell
CellPin
ClockRegion

OVERVIEW PACKAGE CL
PREV NEXT FRAMES
Packages
Package
com.xilinx.rapidwright.placer.blockplacer
com.xilinx.rapidwright.placer.handplacer

Xilinx/RapidWright: Build Custom

Marketplace Explore

Unwatch 31 Unstar 45 Fork 17

Insights Settings

wright.io Edit

3 contributors View license

Create new file Upload files Find file Clone or download

Latest commit d76cff0 2 days ago

om/Xilinx/RapidWright 2 days ago
5 months ago
5 months ago
5 months ago
ator: (1) centering blocks ... a month ago

Filters

ating some of our settings to make group management Learn more

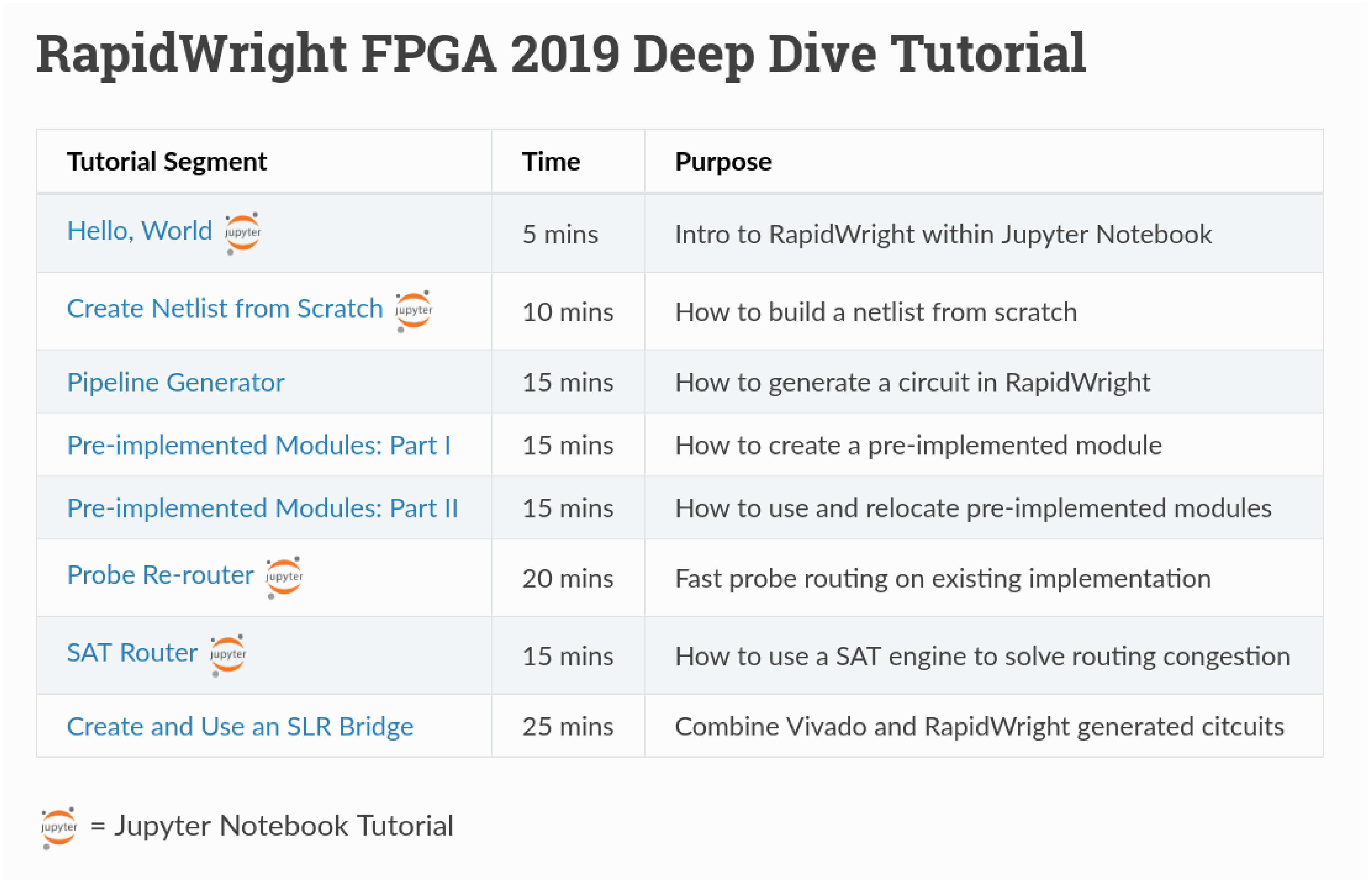
Tags · Manage group · Manage members · Members · About

n this forum (you'll need to join first). RapidWright is an open source project from

5:29 AM
sts - 16 views Feb 7
Jan 11
pynq_project (in p... By Jakob Wenzel - 5 posts - 53 views 12/17/18
pynq Using RapidWright with a Mac

XILINX

Today After Lunch (1:45PM)



RapidWright FPGA 2019 Deep Dive Tutorial

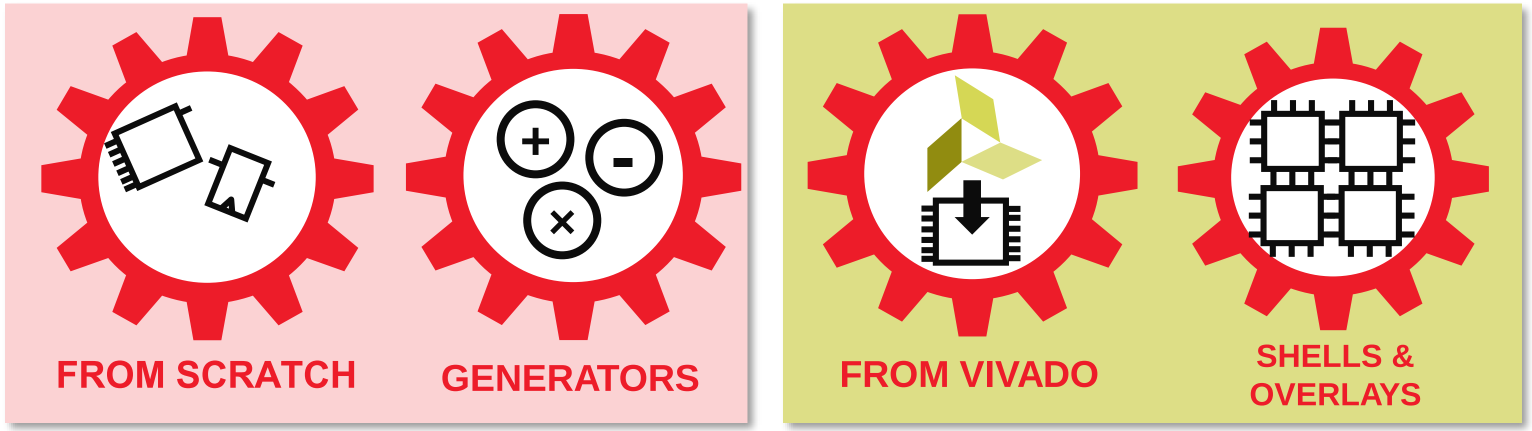
Tutorial Segment	Time	Purpose
Hello, World jupyter	5 mins	Intro to RapidWright within Jupyter Notebook
Create Netlist from Scratch jupyter	10 mins	How to build a netlist from scratch
Pipeline Generator	15 mins	How to generate a circuit in RapidWright
Pre-implemented Modules: Part I	15 mins	How to create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

jupyter = Jupyter Notebook Tutorial

Conclude

XILINX

Summary



> Build routed circuits & reuse P&R circuits

> RapidWright enables:

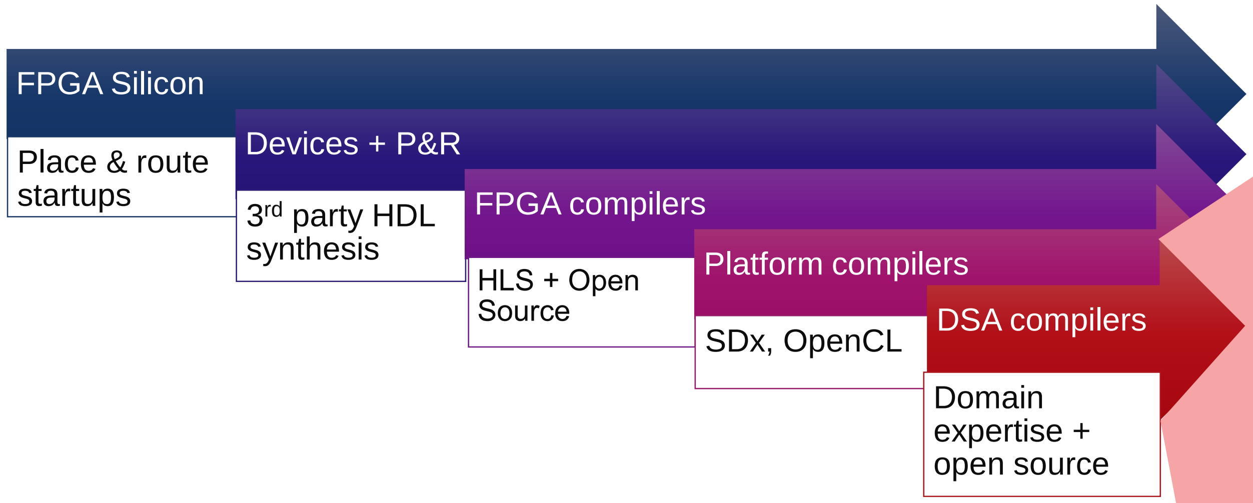
    Performance by 50%

    Debug productivity >10X

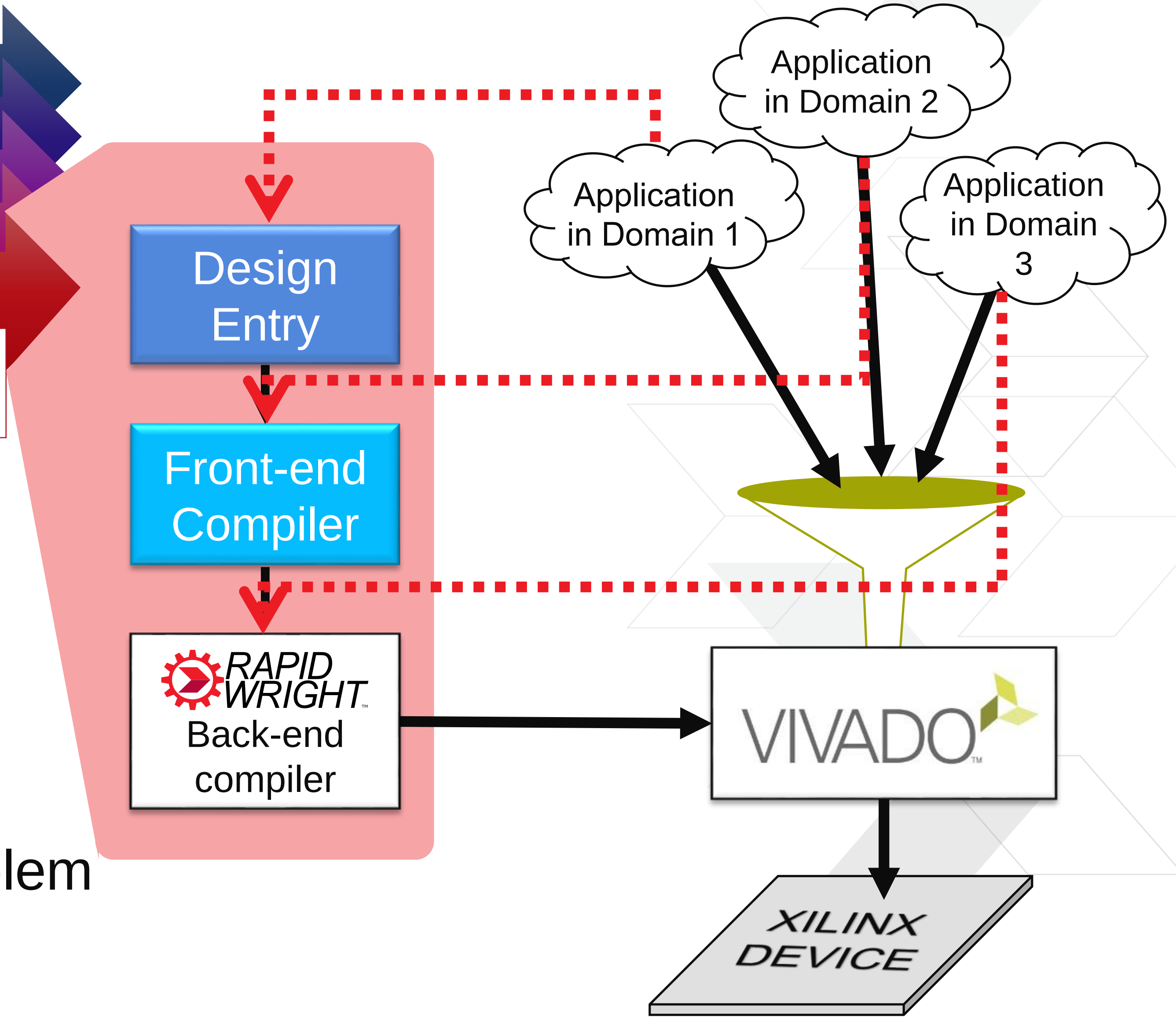
> Leverage algorithmic engines (SAT, ILP, …)

> www.rapidwright.io

RapidWright Enables DSA Compilers



- > Hard problems, let's work together
- > Domain-specific optimizations
- > Architecture exploration
- > Empower those closest to the problem



Adaptable.
Intelligent.

XILINX®