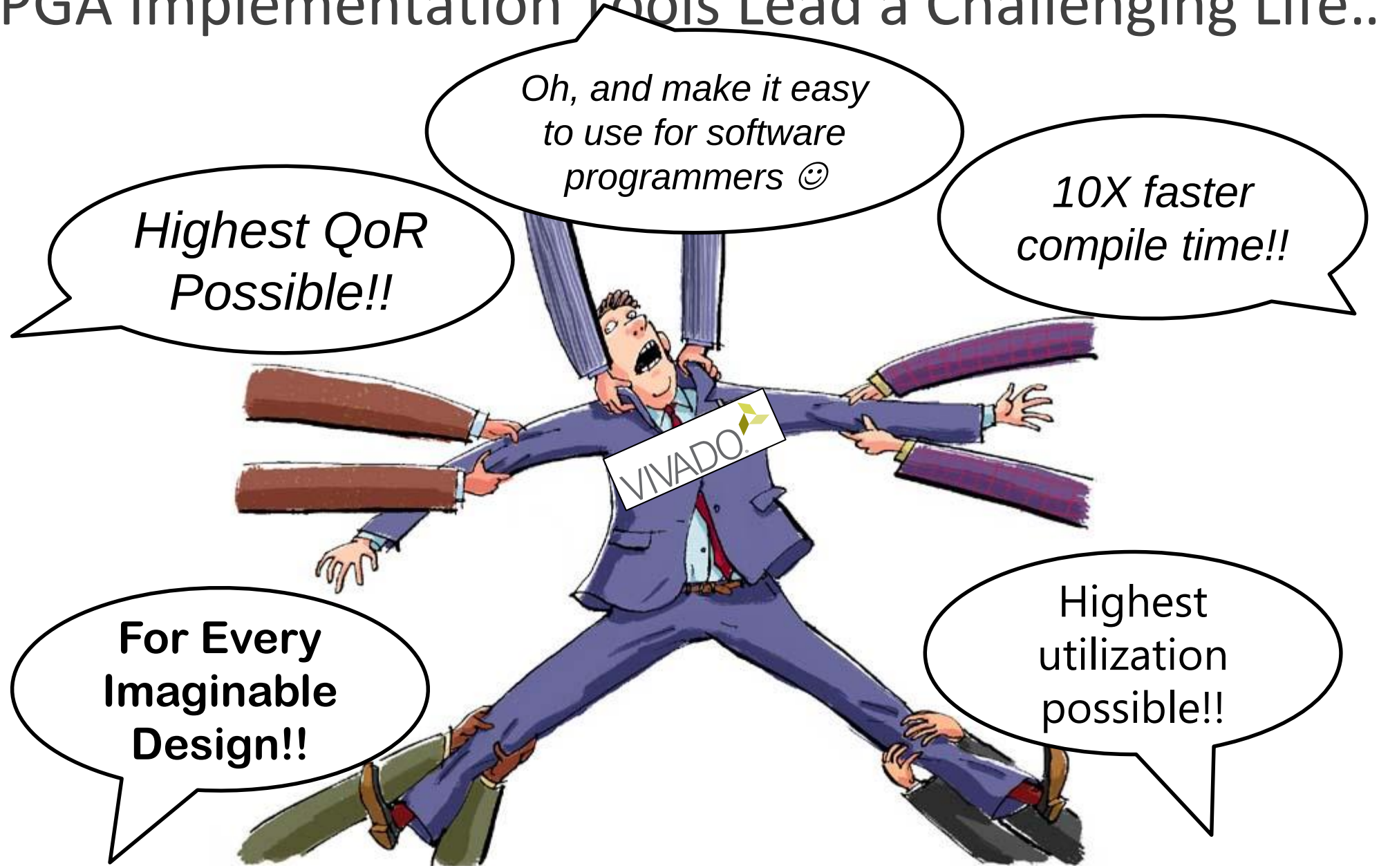
RapidWright[1]: Enabling Custom Crafted Implementations for FPGAs

Chris Lavin & Alireza Kaviani

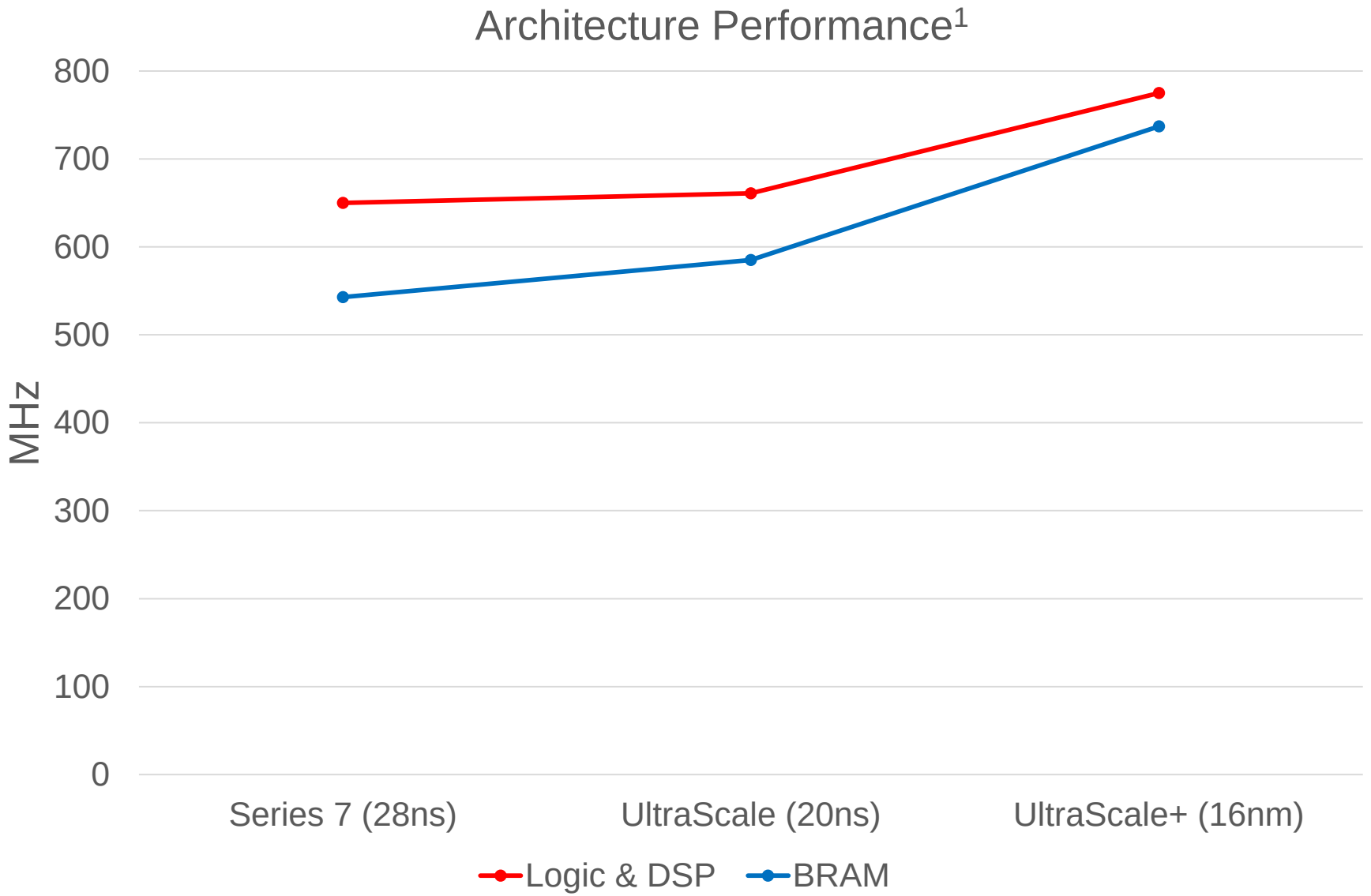
FCCM - May 1, 2018

Wright[1] = maker or builder

FPGA Implementation Tools Lead a Challenging Life…



Xilinx Silicon Capability

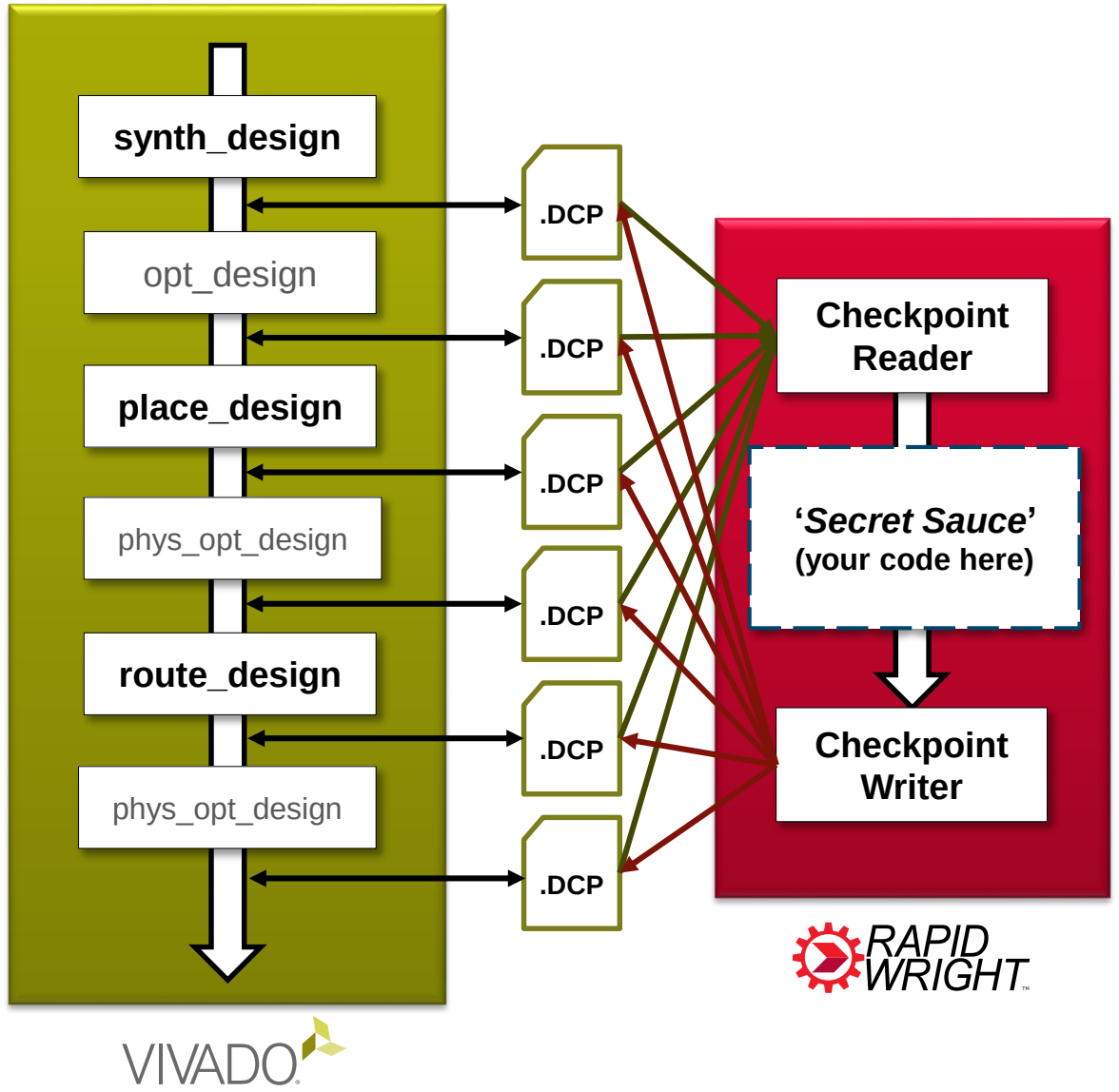


[1]Speed Grade: -2

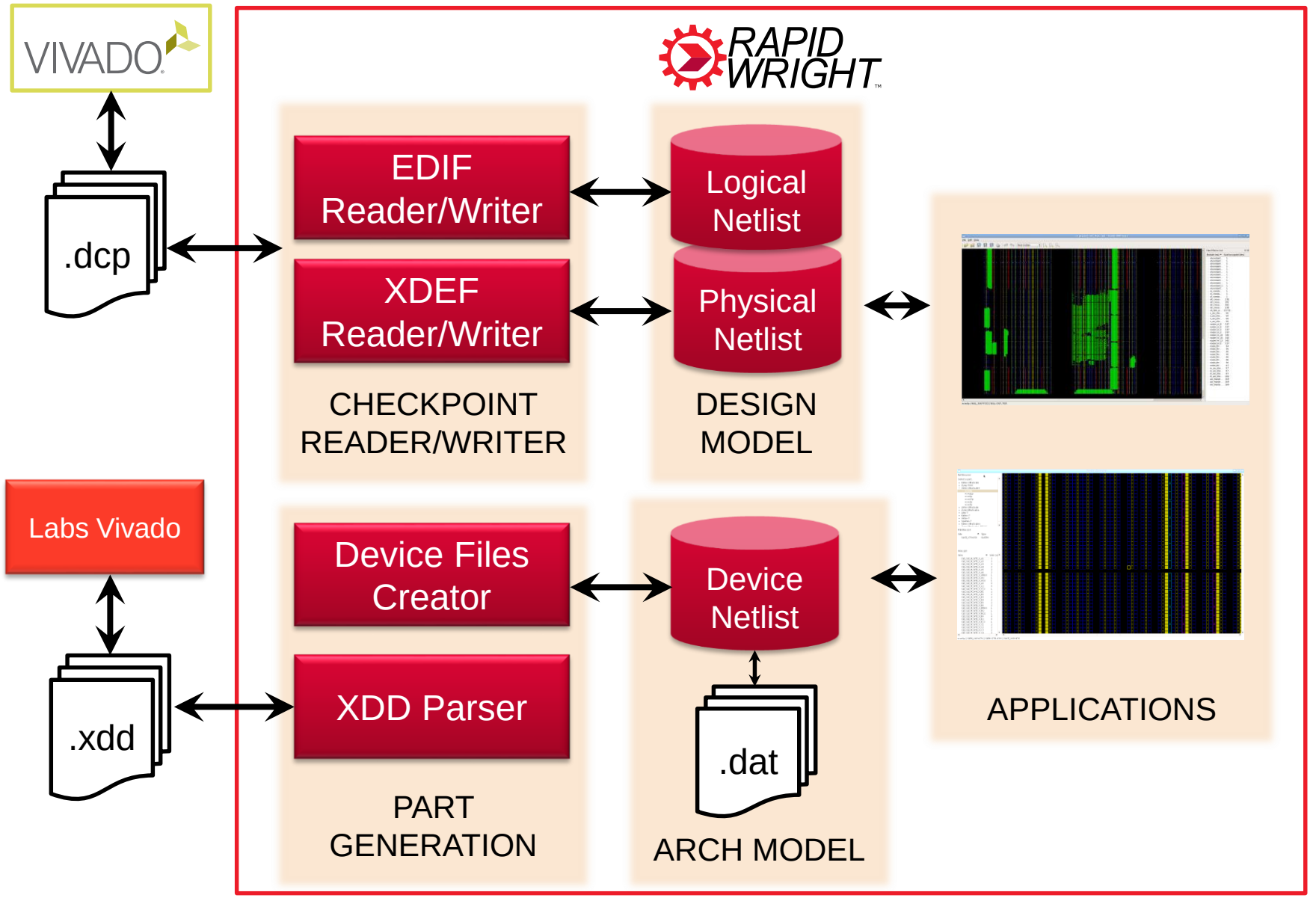
XILINX

What is RapidWright?

- Companion framework for Vivado
  - Communicates through DCPs
  - Fast, light-weight, open source
  - Java, Python support
- Enables targeted solutions
  - Reuse & relocate pre-implemented modules
  - Systematic shells & overlays
  - Generate on-the-fly implementations
- Academic ecosystem
  - Algorithm validation
  - Rapid prototyping of CAD concepts



RapidWright Framework Overview



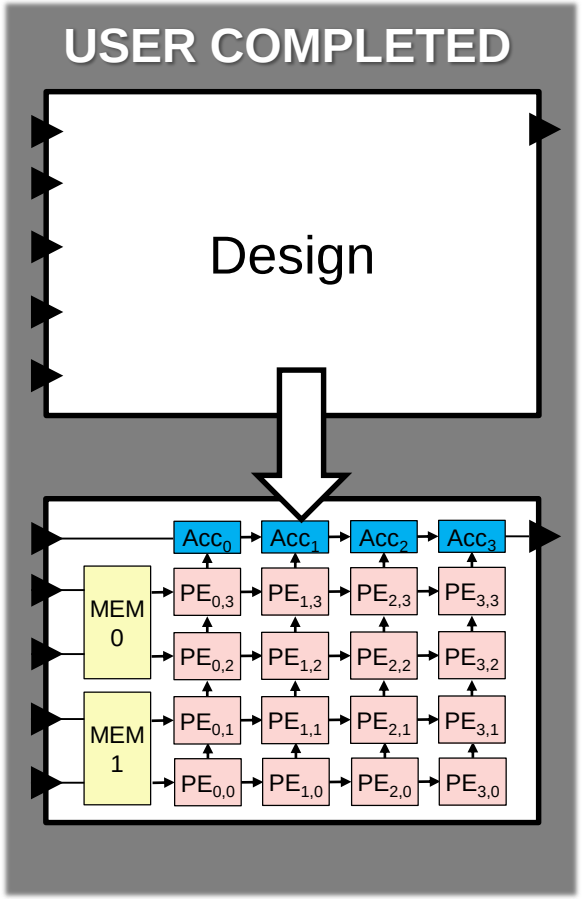
XILINX

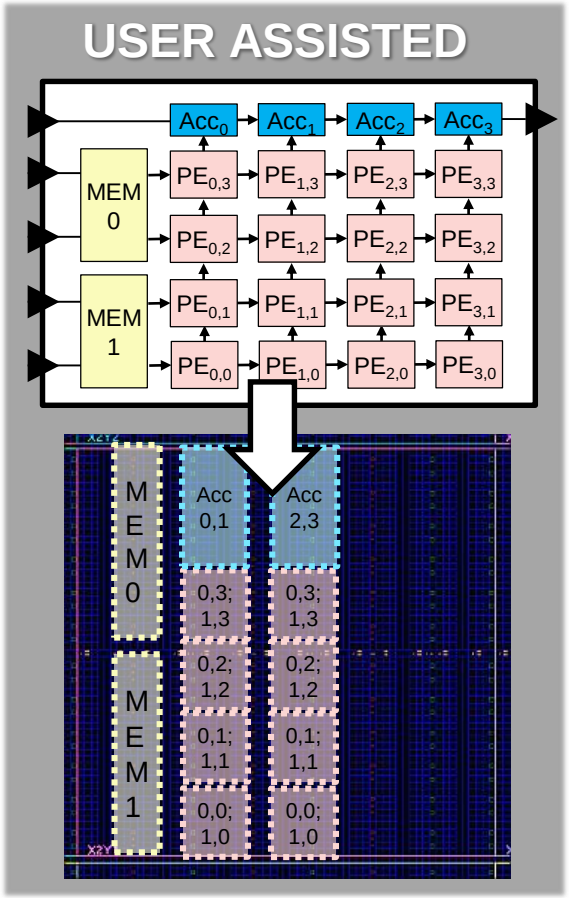
Modular Pre-Implemented Methodology

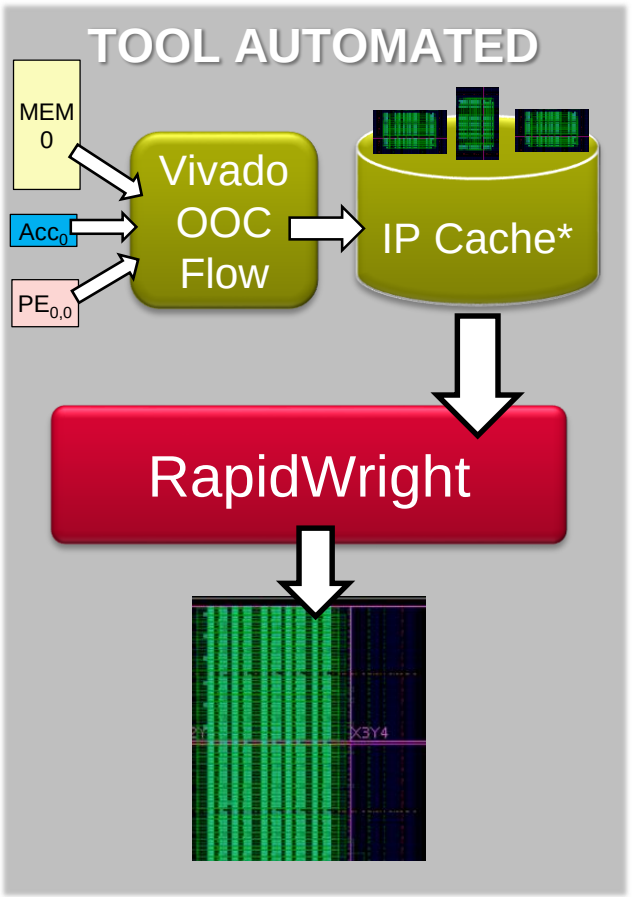
1. Restructure Design

2. Packing & Placement Planning

3. Stitch, Place & Route Implementation

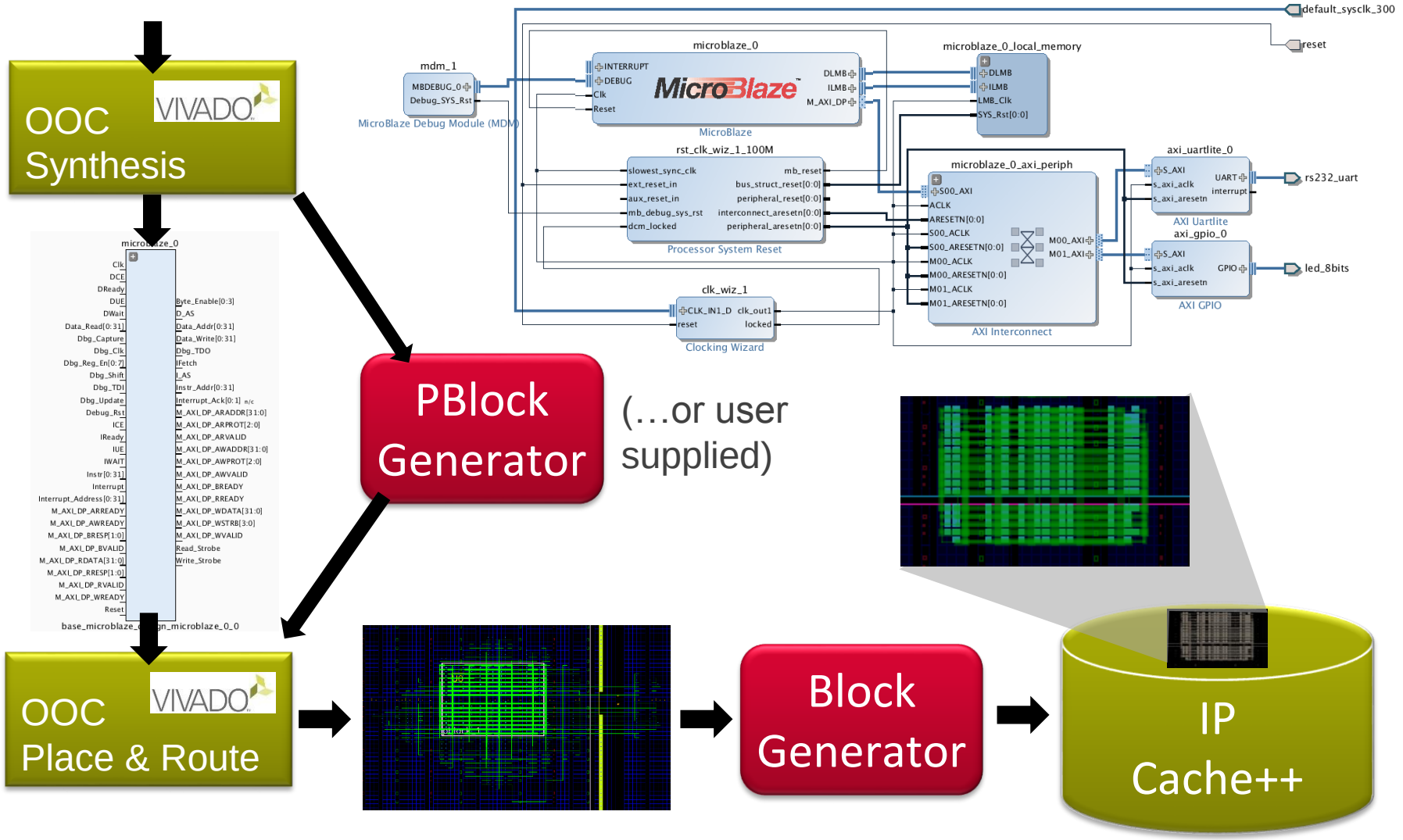






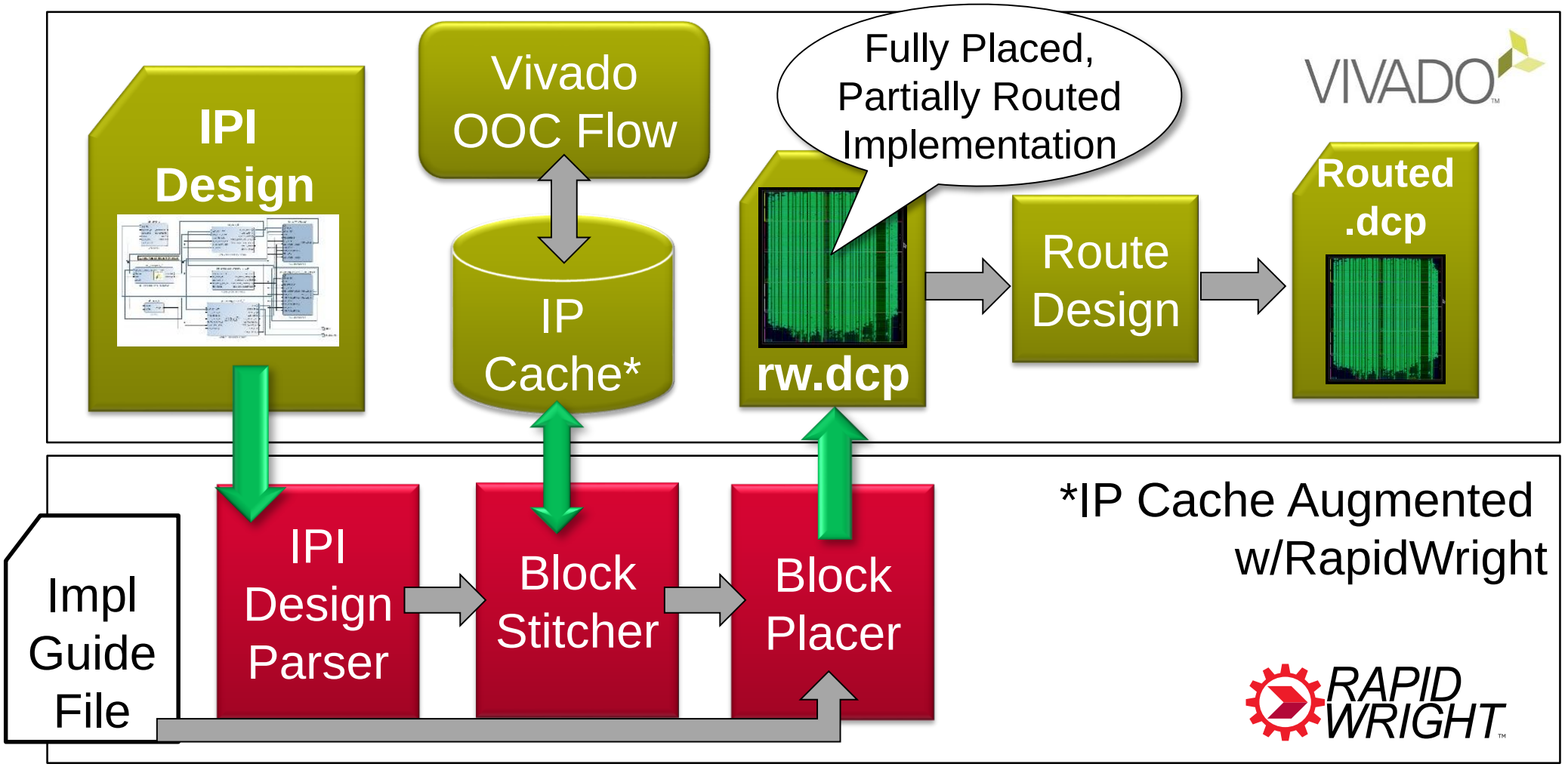
XILINX

Creating Pre-implemented Modules (Vivado OOC Flow)



XILINX

Step 3: RapidWright Pre-implemented Module Flow



Design Results[1]

			(Step 1)			(Step 3)	
Design	**Target Device**	**Baseline (initial design)**	**Vivado (restructured design)**	**Δ**	**Pre-implemented Flow**	**Δ**	**Total Δ**
Seismic	KU040	270MHz	354MHz	**31%**	**390MHz**	**10%**	**41%**
FMA	KU115	270MHz	273MHz	**1%**	**417MHz**	**53%**	**54%**
GEMM	KU115*	391MHz	437MHz	**10%**	**462MHz**	**6%**	**16%**
Heart of Gold	ZU9EG*	368MHz	569MHz	**55%**	**541MHz**	**-5%**	**50%**

*Constrained area of the device

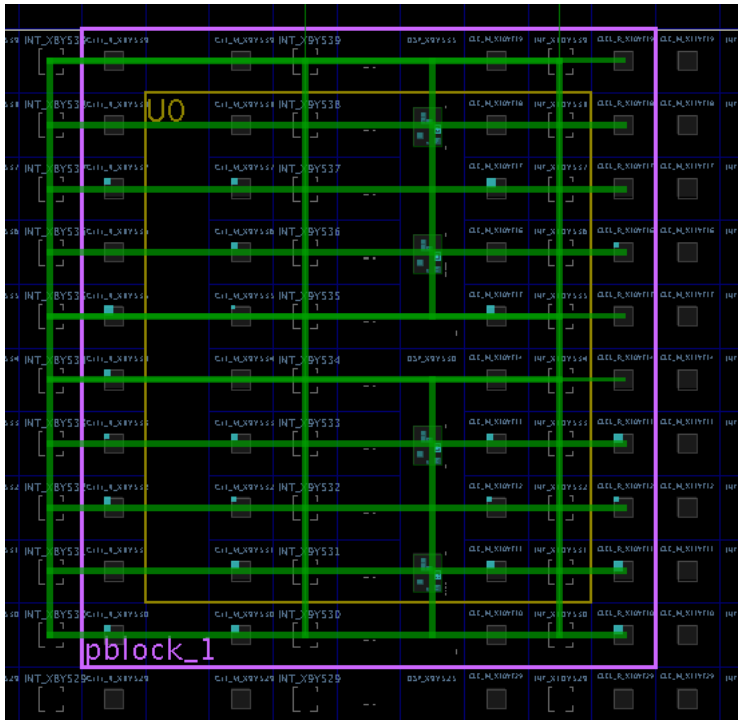
Design	LUT Utilization	FF Utilization	DSP Utilization	BRAM Utilization
Seismic	93% (226k)	5% (26k)	-	-
FMA	25% (166k)	50% (656k)	97% (5360)	6% (130)
GEMM	19% (64k)	20% (134k)	87% (2400)	-
Heart of Gold	46% (30k)	29% (38k)	42% (272)	96% (208)

[1]Speed Grade: -2

2. FMA Design

- **GOAL**
  - Highest compute (TeraOp/s) possible
  - 16-bit fused multiply accumulate
- KU115 Implementation
  - 1340 kernel instances
    - 4x10 CLEs, 1x4 DSPs
  - 97% DSP utilization
  - 4.4 TeraOp/s
- “Fabric discontinuites”
  - SLR boundary
  - IO Columns
  - Depopulated CLEs (SLR crossing)

RUN	F_{MAX} (MHz)
Baseline (initial design)	**270**
Vivado (restructured design)	**273 (+1%)**
Pre-implemented Flow	**417 (+53%)**



FMA KERNEL

Re-locatability & Multiple Implementations

Impl #0

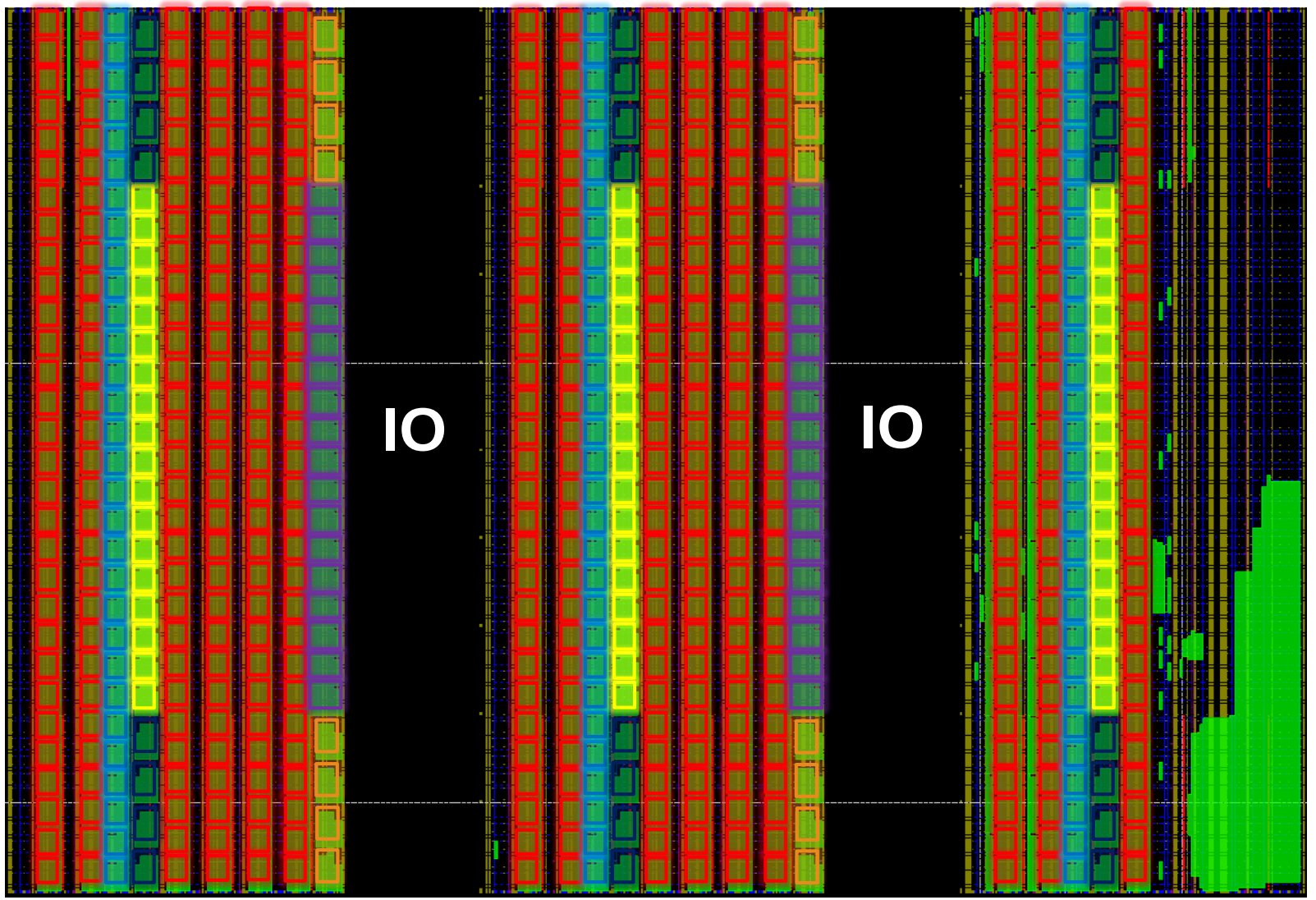
Impl #1

Impl #2

Impl #3

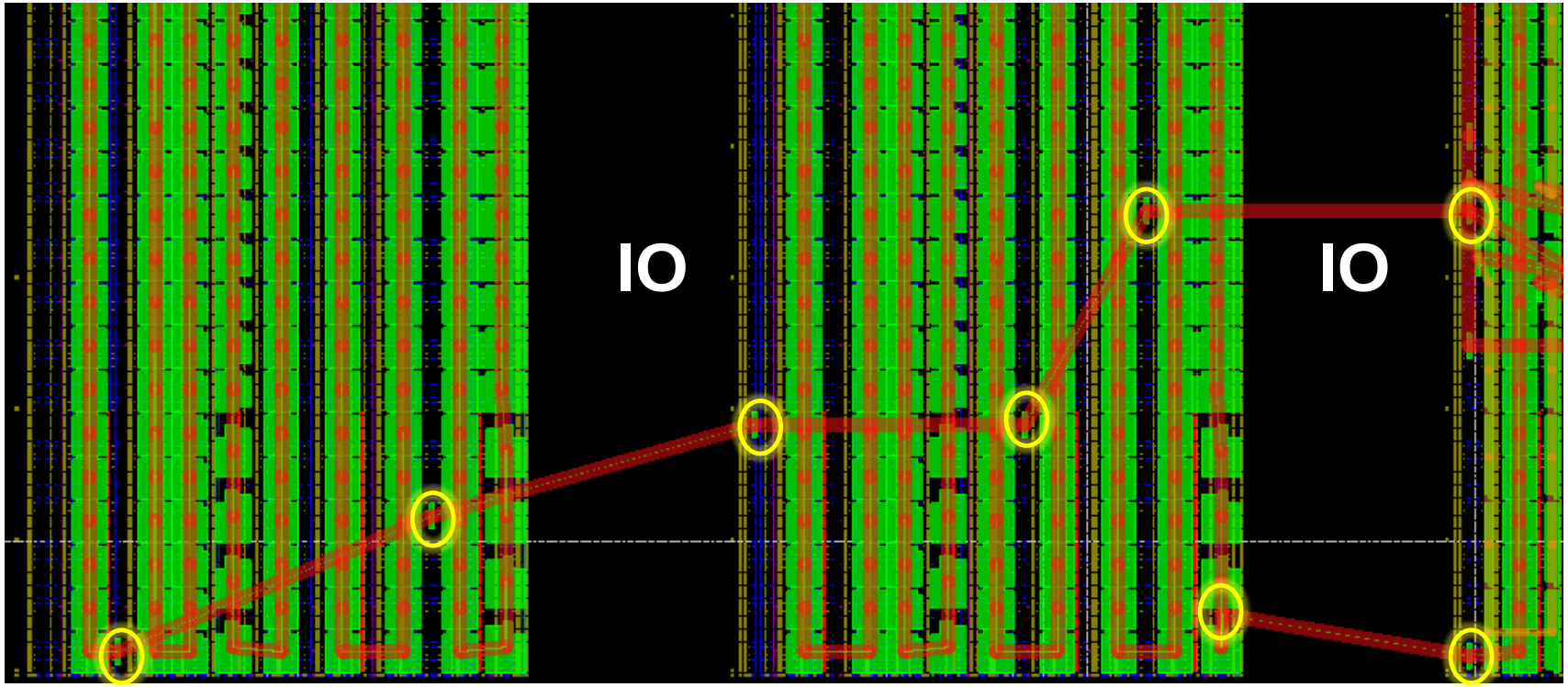
Impl #4

Impl #5



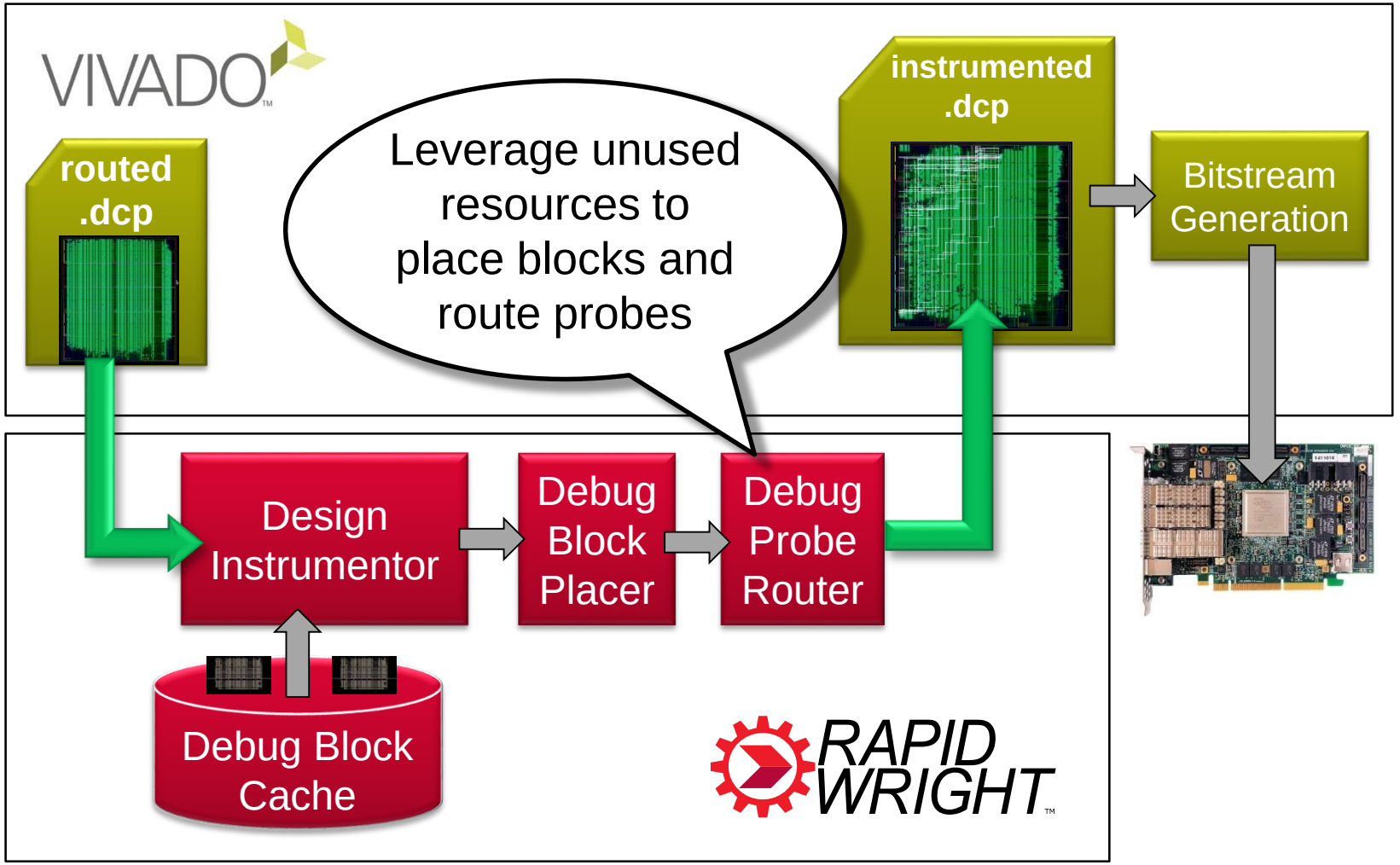
XILINX

Crossing IOs: AXI Stream Register Slices

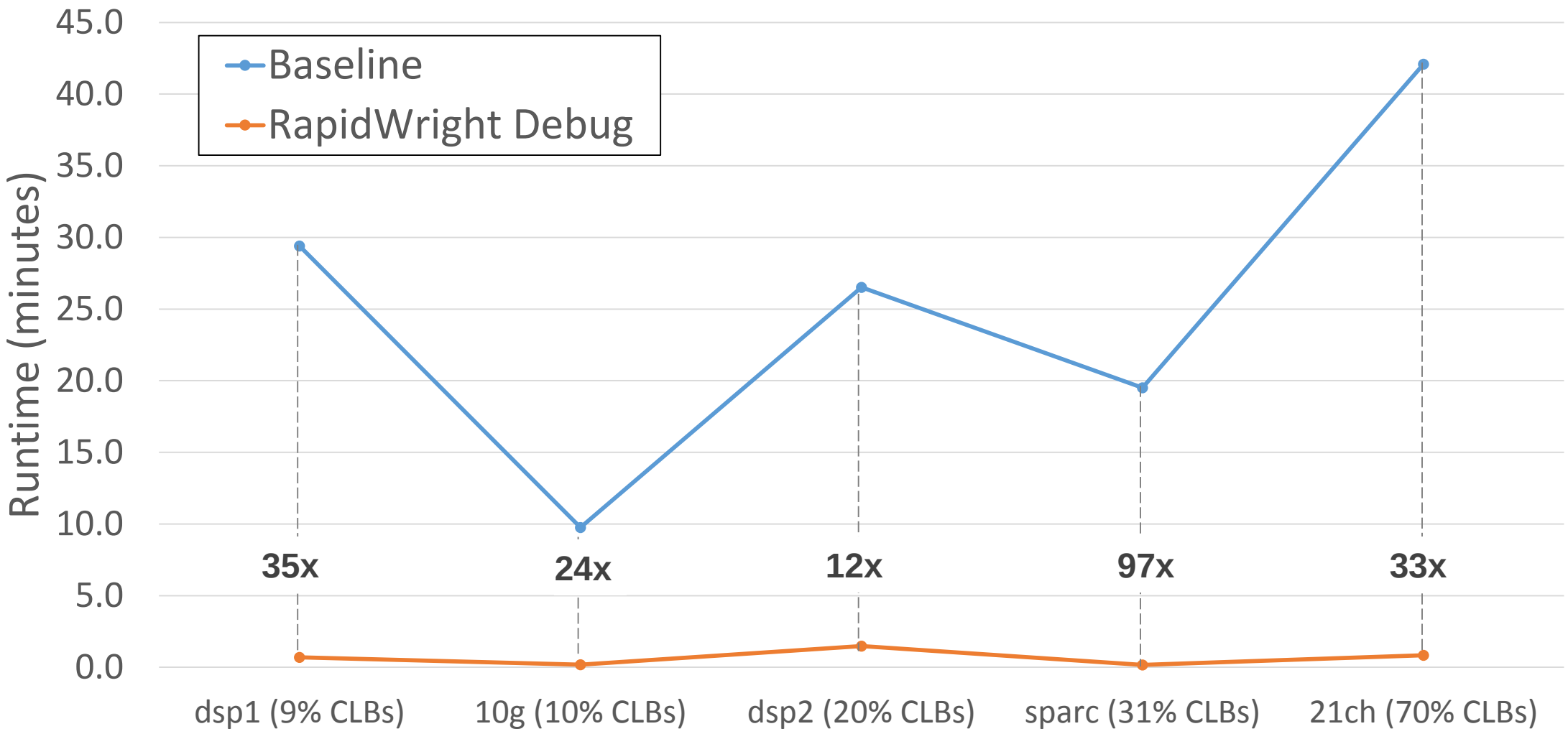


- Strategically placed register slices to cross long distances
  - cross chip IO columns
- Latency insertion/connectivity is easily automated

Debug Productivity

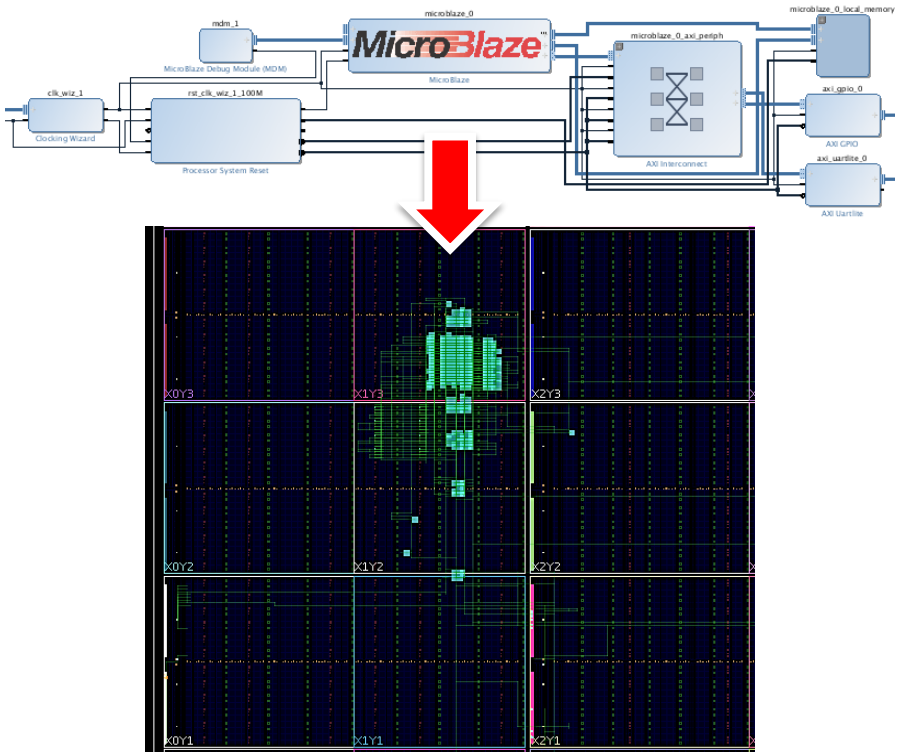


RapidWright vs. Vivado for Debug Instrumentation Speedup

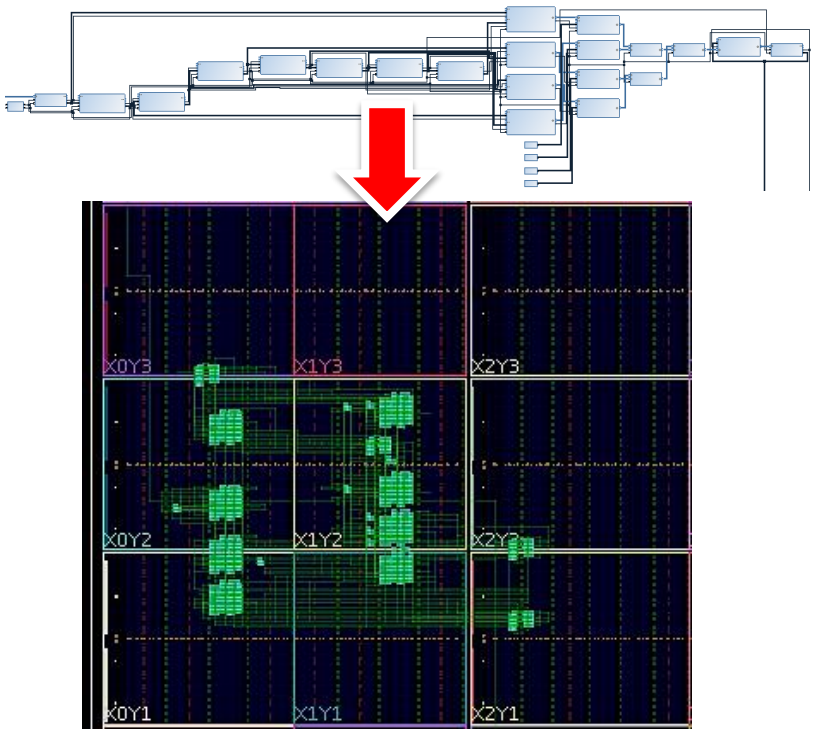


XILINX

Fully Placed and Routed Designs in Seconds



microblazeDesign



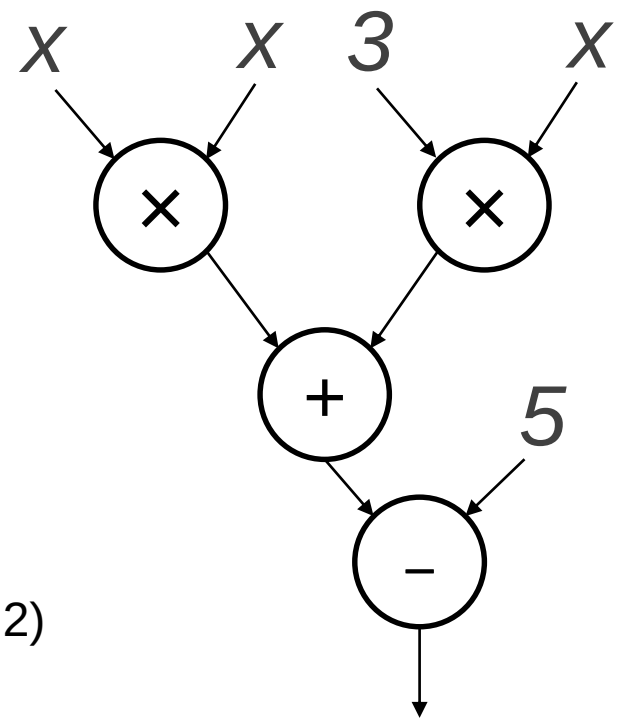
fpFIRFilter

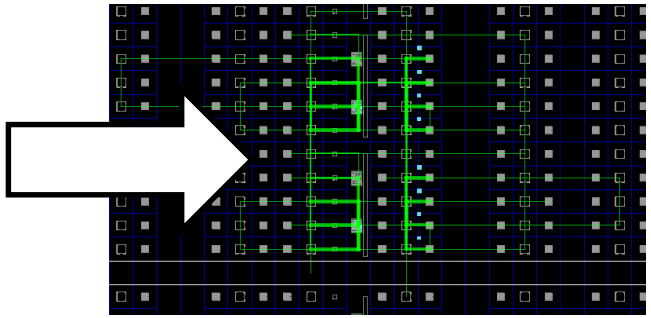
	RapidWright Flow	Vivado	Speedup
microblazeDesign	12.5 seconds	232 seconds	19x
fpFIRFilter	18.6 seconds	183 seconds	10x

On-the-fly, Pre-implemented Module Generators (Demo)

- Build modules on-demand
  - Placed and routed *in seconds*
  - Reusable and compose-able
  - Target spec performance
- Parameterizable Generators
  - Adder
  - Subtractor
  - Multiplier
- Polynomial Solution Generator
  - Runs at spec 775MHz (UltraScale+, SG2)
  - Constructed on-the-fly in seconds
  - Still in development

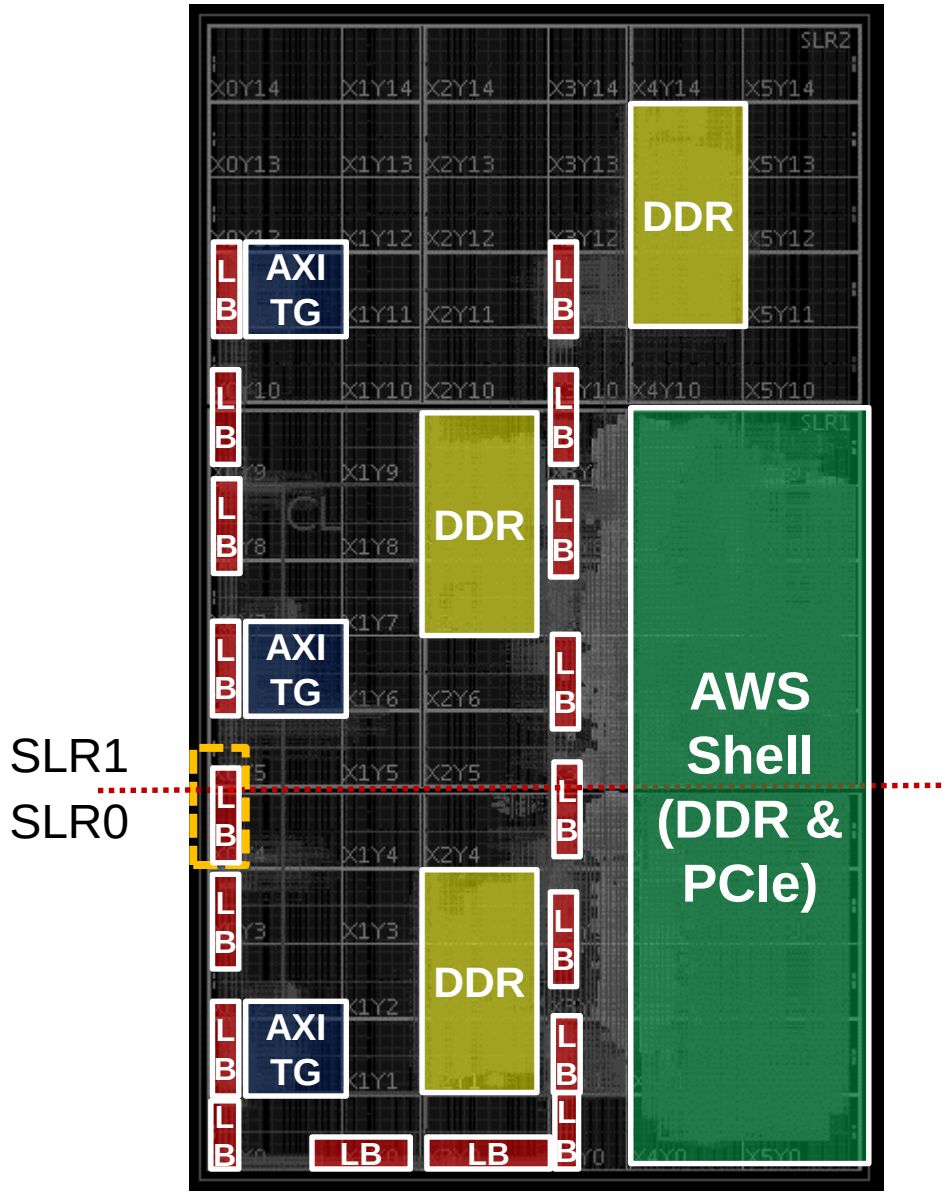
x^2+3*x-5





AWS F1 - LinkBlaze Shell Floorplan

- Goal: Achieve Spec Performance
  - 775MHz on UltraScale+, SG2
  - Minimize overhead of overlays/shells
- LinkBlaze [1]: Data movement soft NoC
  - 128 bit, bi-directional
  - Modular design
    - Pre-implemented modules
    - Captures high performance implementation
- Challenge: Crossing SLR
  - Solved using two techniques in RapidWright
    - Custom clocking of Leaf clock buffer and delay tuning
    - Custom clock root routing per SLR crossing



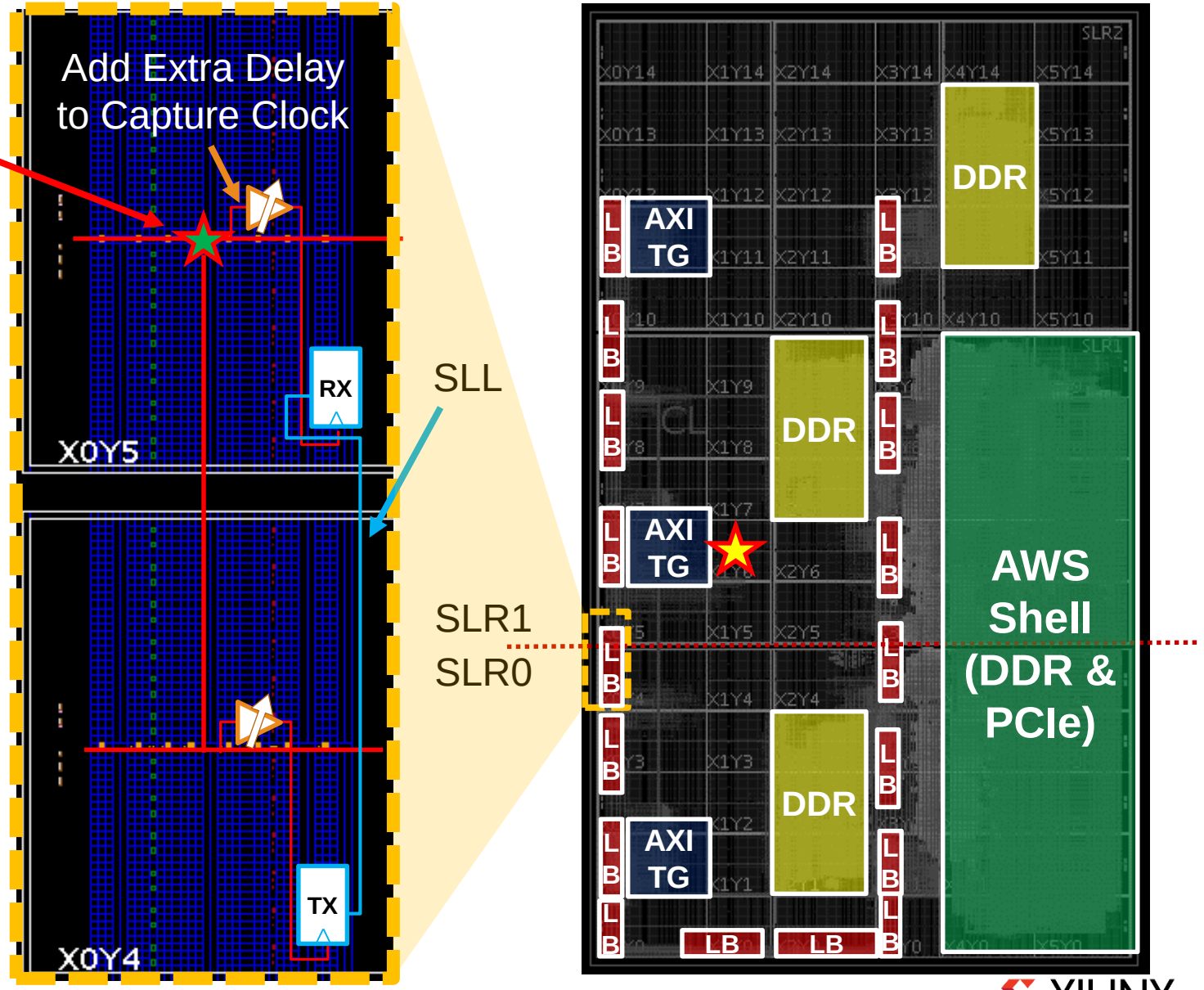
[1] LinkBlaze: Efficient global data movement for FPGAs (ReConFig 2017)

XILINX

SLR Crossing Solution – Clocking Techniques

- Leaf Clock Buffers (LCBs)
  - Custom route RX/TX to same LCB
  - Tune LCB delay to avoid hold issues
- Inter-SLR Compensation
  - 15% tax of clk delay between root and RX flop
  - Minimize by custom creating clock roots per SLR crossing

Vivado 2017.3	LCB	LCB & Custom Clk Route
549MHz	595MHz	730MHz



XILINX

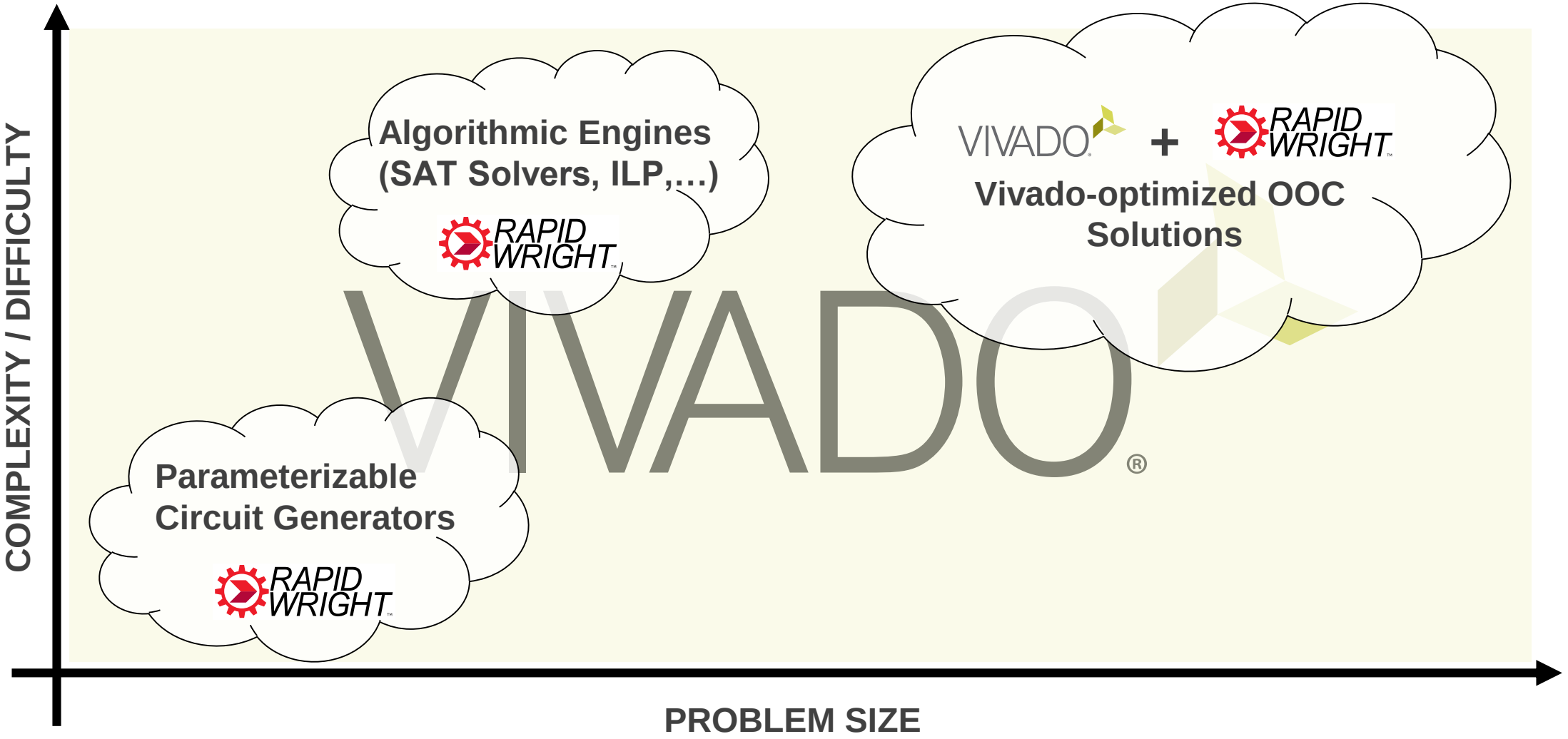
RapidWright SLR Crossing DCP Generator (Demo)

- 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 crossings.  See
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)
```

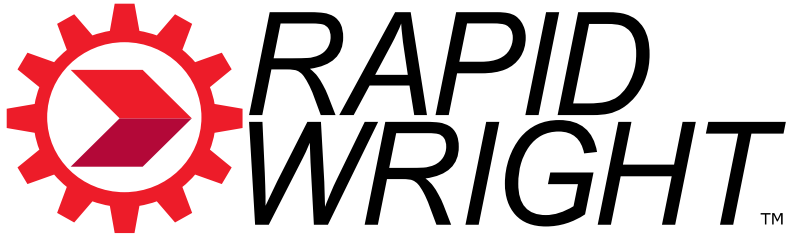
Vision: Pre-implemented Modules



XILINX

Takeaways

- RapidWright enables customized solutions
  - Relocate & replicate pre-implemented modules
  - On-the-fly circuit generators
  - Leverage algorithmic engines (SAT Solvers, ILP, …)
- Modular pre-implemented Methodology
  - Up to 50% performance improvement
  - ~10X productivity gains
  - Near-spec performance (94% of spec)
- www.rapidwright.io
  - Open source -- try it out today
  - Documentation, tutorials, source code, and demos



www.rapidwright.io