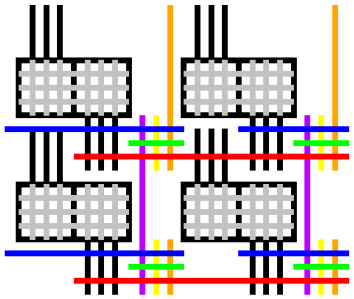
Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks

Yuanlong Xiao, Dongjoon Park, Andrew Butt, Hans Giesen, Zhaoyang Han, Rui Ding, Nevo Magnezi, Raphael Rubin, and André DeHon (andre@acm.org)

Implementation of Computation Group
University of Pennsylvania
December 12th, 2019



Penn

Story

Our target is decreasing the **compilation time**

- **Problem:** Compilation is slow -- limiting FPGA use and optimization
- **Idea:** Divide into smaller problems
  - Solve in parallel
  - Incrementally compile just the part that changed
- **Tool**: PRflow
- **Impact:** Able to achieve **12-18 minutes** using Vivado
  - Contrast **42-160 minutes** no PRflow
- Plausible to achieve **2-5 minutes** with open source Symbiflow

Story

Our target is decreasing the **compilation time**

- **Problem:** Compilation is slow -- limiting FPGA use and optimization
- **Idea:** Divide into smaller problems
  - Solve in parallel
  - Incrementally compile just the part that changed
- **Tool**: PRflow
- **Impact:** Able to achieve **12-18 minutes** using Vivado
  - Contrast **42-160 minutes** no PRflow
- Plausible to achieve **2-5 minutes** with open source Symbiflow

Story

Our target is decreasing the **compilation time**

- **Problem:** Compilation is slow -- limiting FPGA use and optimization
- **Idea:** Divide into smaller problems
  - Solve in parallel
  - Incrementally compile just the part that changed
- **Tool**: PRflow
- **Impact:** Able to achieve **12-18 minutes** using Vivado
  - Contrast **42-160 minutes** no PRflow
- Plausible to achieve **2-5 minutes** with open source Symbiflow

Motivation:

- Today's FPGA compilation is slow
  - **30-178 minutes** for Rosetta [1] Benchmarks on Xilinx ZCU102 board
- Problems due to slow compilation
  - Slow debug and development time
  - Limit the scope of design space exploration
- Why is it slow?
  - Compile and co-optimize the entire design

[1] Yuan Zhou et al. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs. Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Motivation:

- Today's FPGA compilation is slow
  - **30-178 minutes** for Rosetta [1] Benchmarks on Xilinx ZCU102 board
- Problems due to slow compilation
  - Slow debug and development time
  - Limit the scope of design space exploration
- Why is it slow?
  - Compile and co-optimize the entire design

[1] Yuan Zhou et al. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs. Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Motivation:

- Today's FPGA compilation is slow
  - **30-178 minutes** for Rosetta [1] Benchmarks on Xilinx ZCU102 board
- Problems due to slow compilation
  - Slow debug and development time
  - Limit the scope of design space exploration
- Why is it slow?
  - Compile and co-optimize the entire design

[1] Yuan Zhou et al. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs. Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

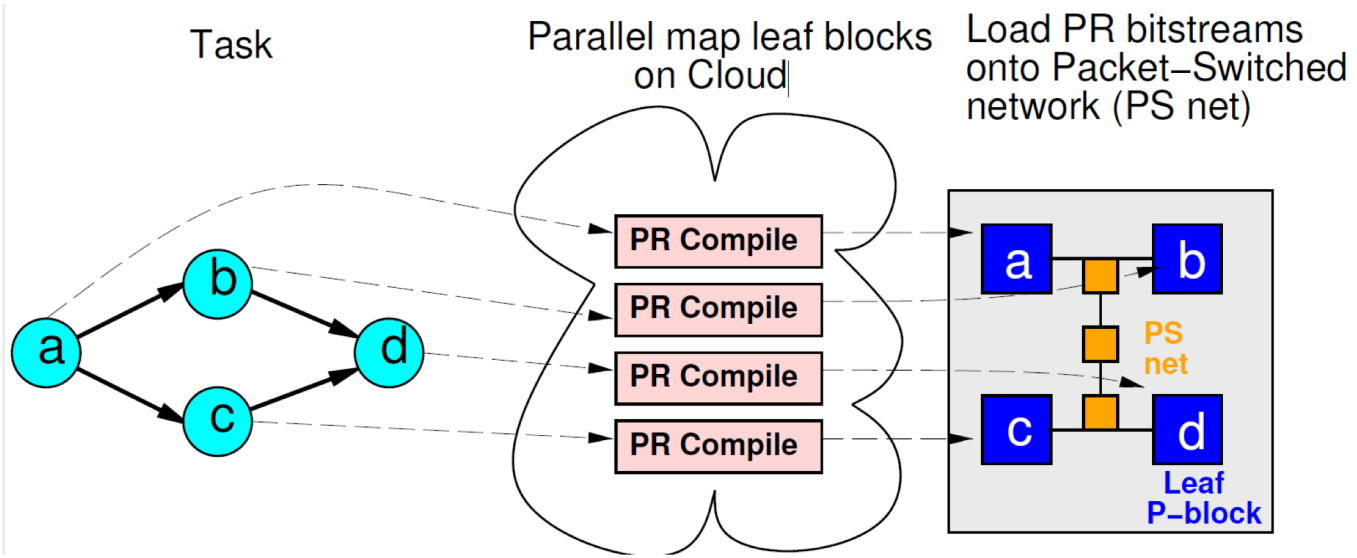
Motivation:

- Today's FPGA compilation is slow
  - **30-178 minutes** for Rosetta [1] Benchmarks on Xilinx ZCU102 board
- Problems due to slow compilation
  - Slow debug and development time
  - Limit the scope of design space exploration
- Why is it slow?
  - Compile and co-optimize the entire design

[1] Yuan Zhou et al. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs. Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Ideas:

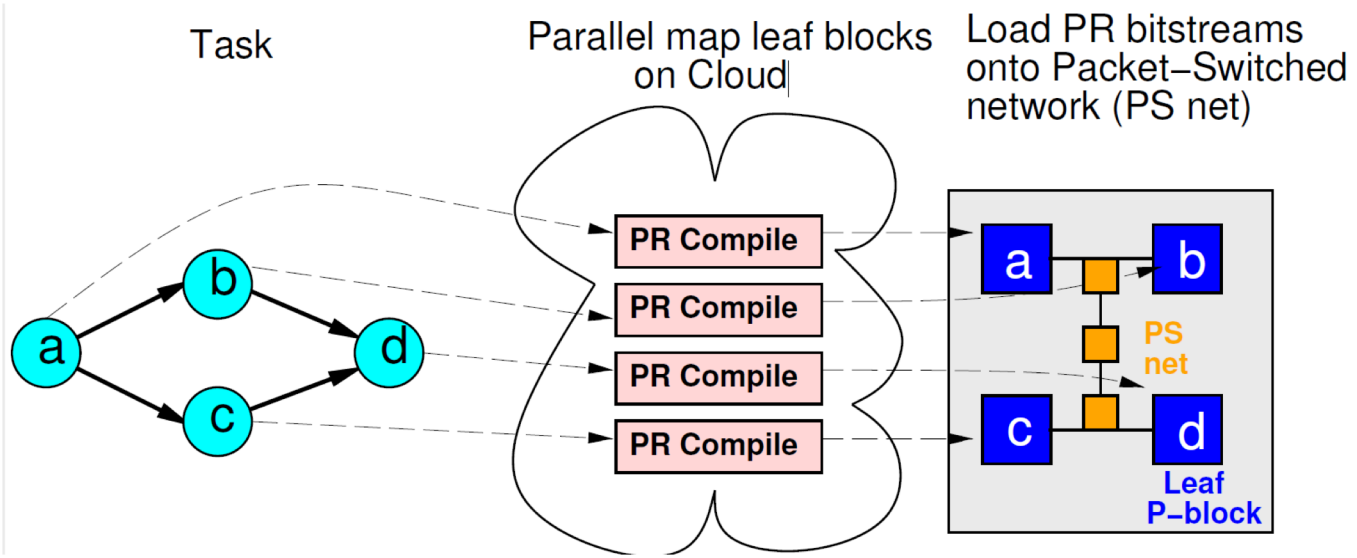
- Divide-and-conquer compilation strategy based on utilizing partial reconfiguration



Ideas:

- Divide-and-conquer compilation strategy based on utilizing partial reconfiguration

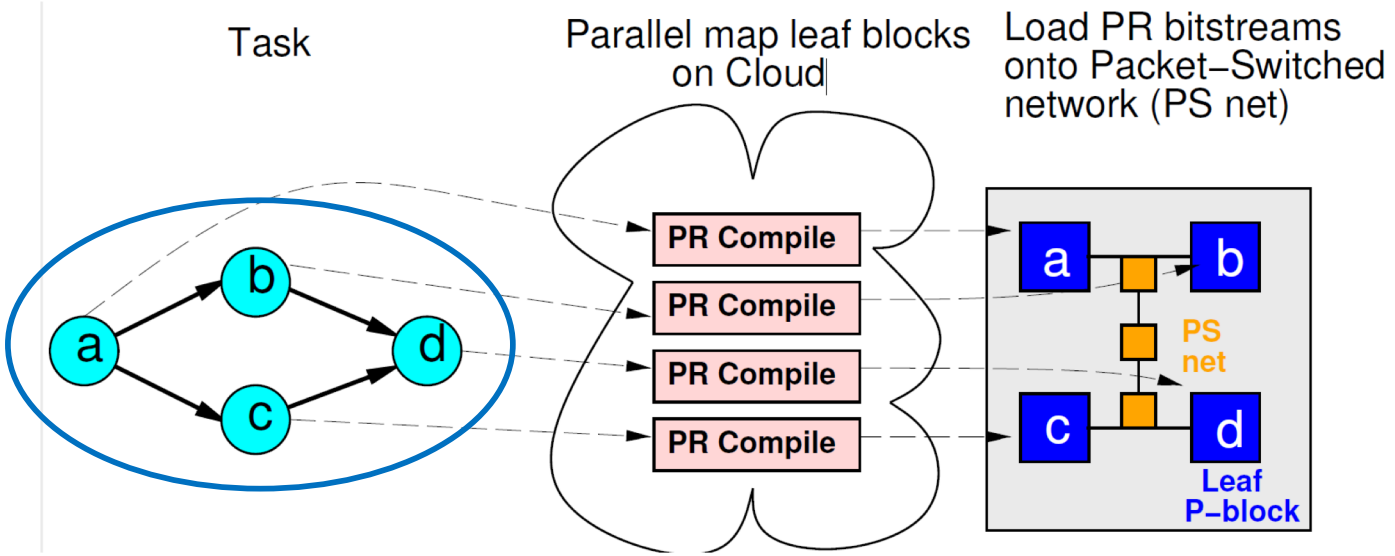
$$T_{new} = \frac{T_{origin}}{\#\ of\ Partition}$$



Ideas:

- Divide-and-conquer compilation strategy based on utilizing partial reconfiguration

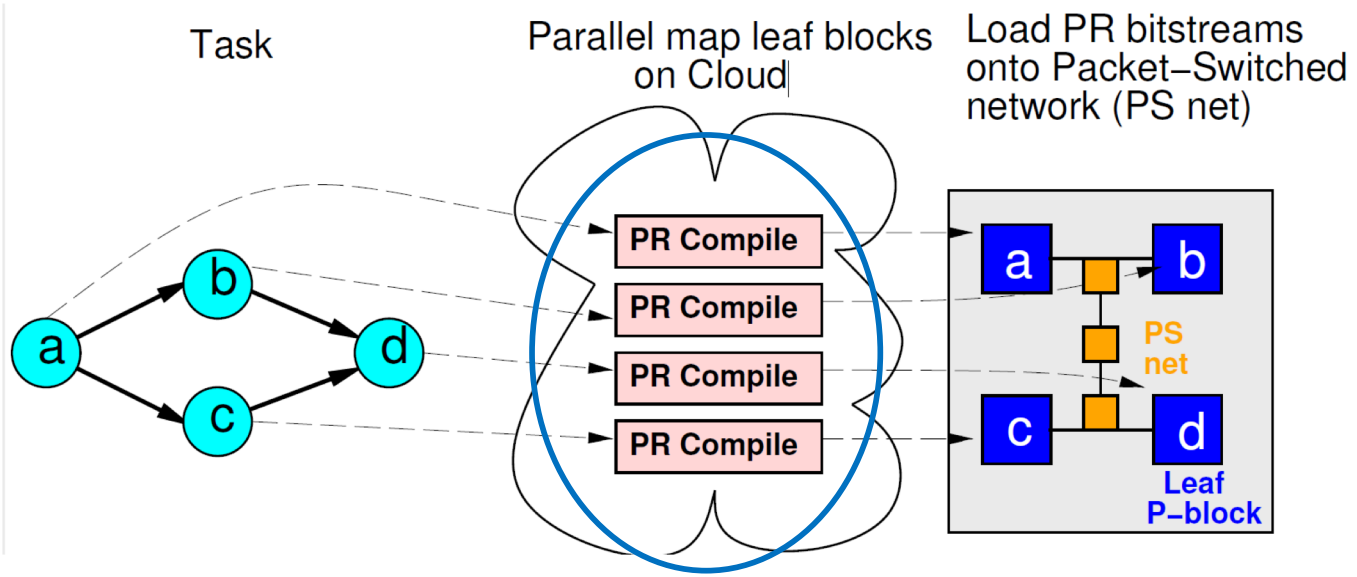
$$T_{new} = \frac{T_{origin}}{\#\ of\ Partition}$$



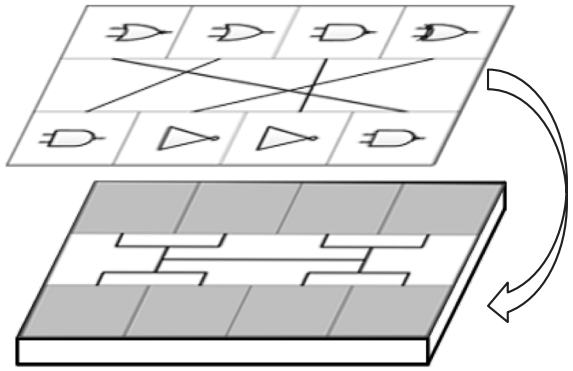
Ideas:

- Divide-and-conquer compilation strategy based on utilizing partial reconfiguration

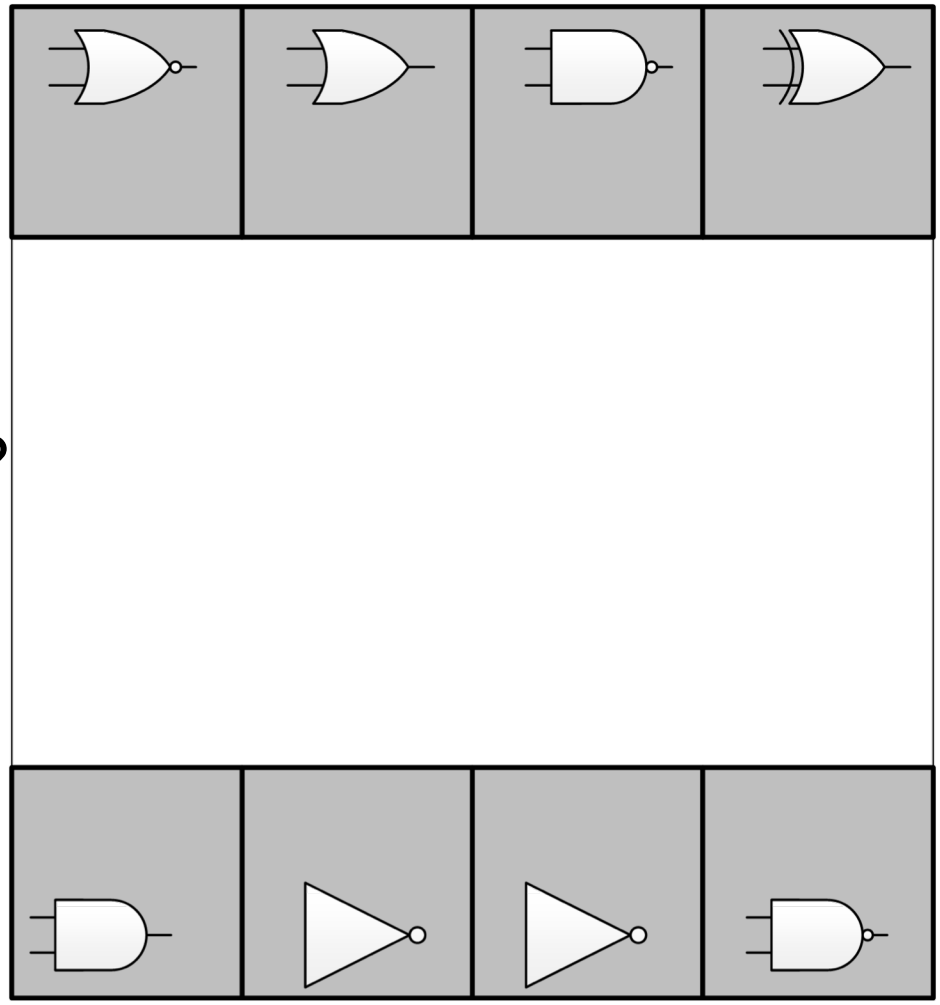
$$T_{new} = \frac{T_{origin}}{\#\ of\ Partition}$$



PR-blocks



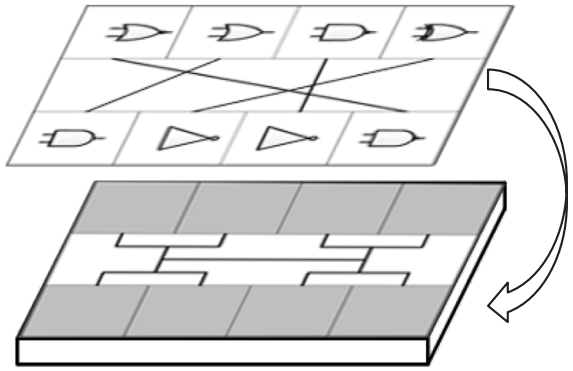
Ideas:



- How to Link PR-blocks together?
  - Standardized interfaces into each blocks[2]
  - Leaf interface: Arbitrary number of inputs and outputs to user logic
  - **B**utterfly **F**at **T**ree (**BFT**)
  - Packet-switched: Arbitrary interconnection between 2 leaves

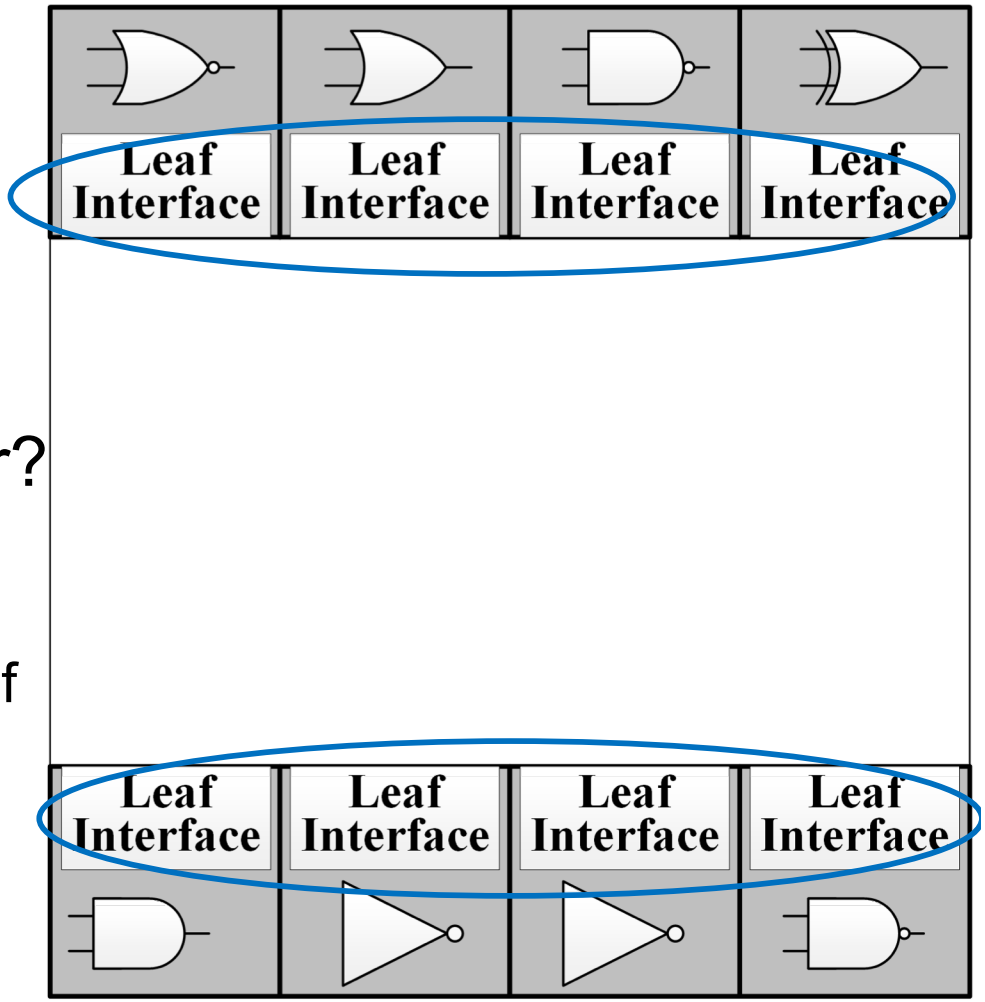
[2] Caspi, Eylon, et al. "Stream computations organized for reconfigurable execution (SCORE)." International Workshop on Field Programmable Logic and Applications. Springer, Berlin, Heidelberg, 2000.

PR-blocks



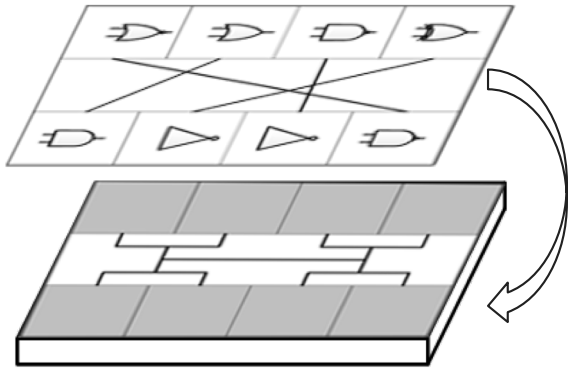
Ideas:

- How to Link PR-blocks together?
  - Standardized interfaces into each block[2]
  - Leaf interface: Arbitrary number of inputs and outputs to user logic
  - **B**utterfly **F**at **T**ree (**BFT**)
  - Packet-switched: Arbitrary interconnection between 2 leaves



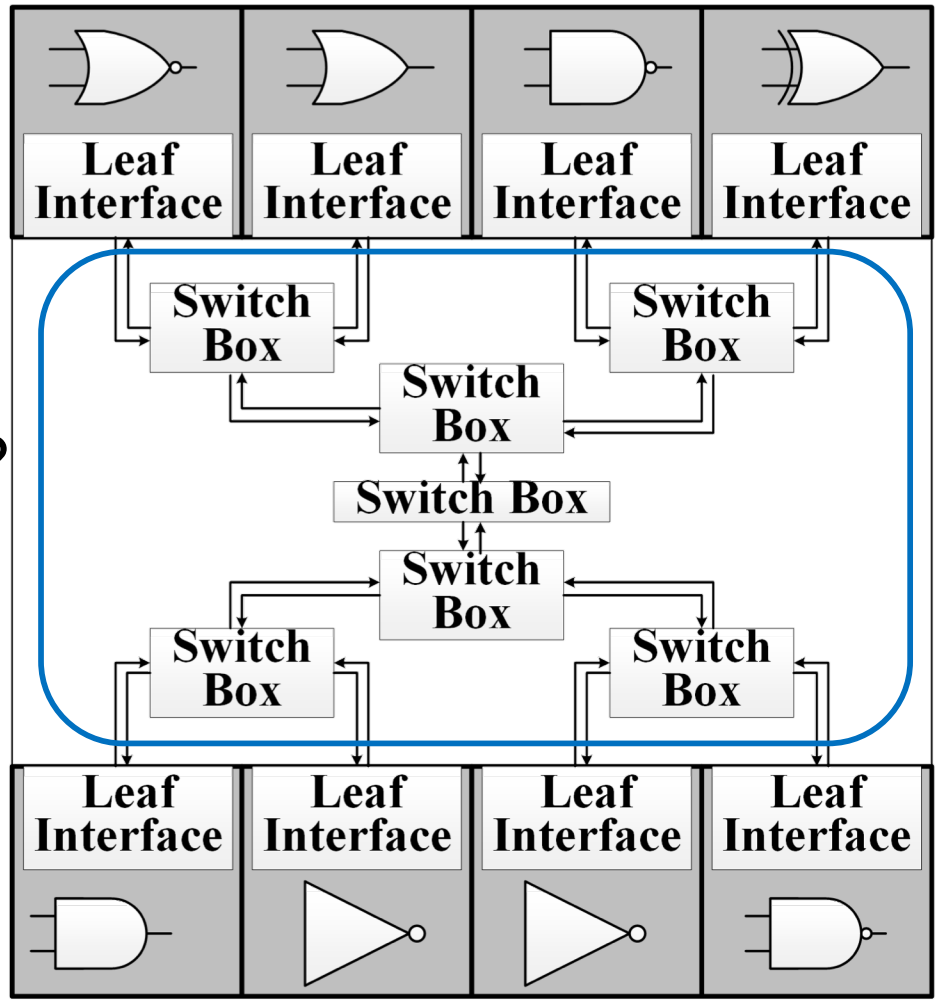
[2] Caspi, Eylon, et al. "Stream computations organized for reconfigurable execution (SCORE)." International Workshop on Field Programmable Logic and Applications. Springer, Berlin, Heidelberg, 2000.

PR-blocks



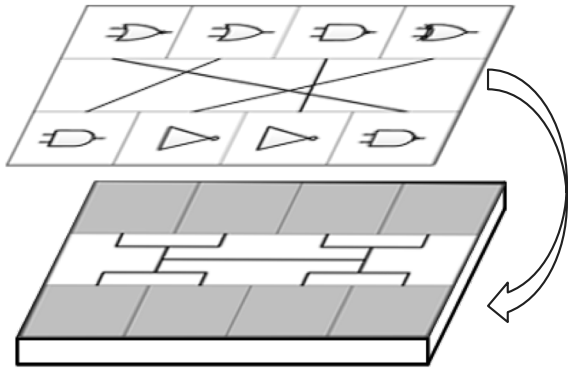
Ideas:

- How to Link PR-blocks together?
  - Standardized interfaces into each blocks
  - Leaf interface: Arbitrary number of inputs and outputs to user logic
  - **B**utterfly **F**at **T**ree (**BFT**)[3]
  - Packet-switched: Arbitrary interconnection between 2 leaves

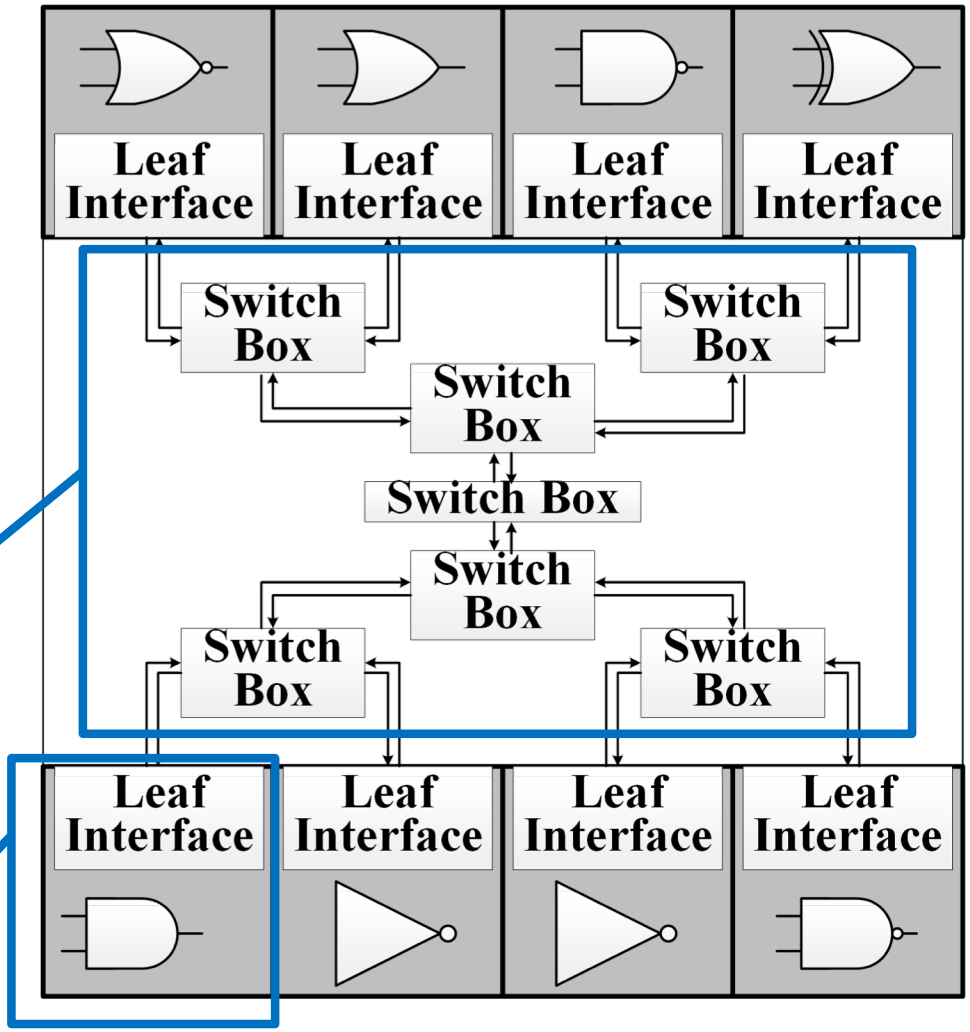


[3] Kapre, N., 2017, September. Deflection-routed butterfly fat trees on FPGAs. In 2017 27th International Conference on Field Programmable Logic and Applications (FPL) (pp. 1-8). IEEE.

PR-blocks



Ideas:

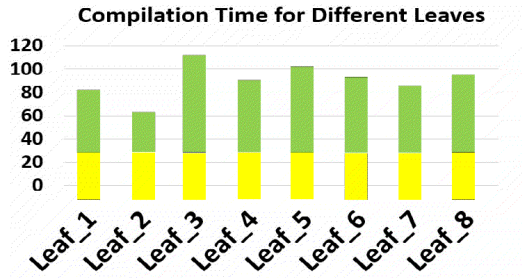


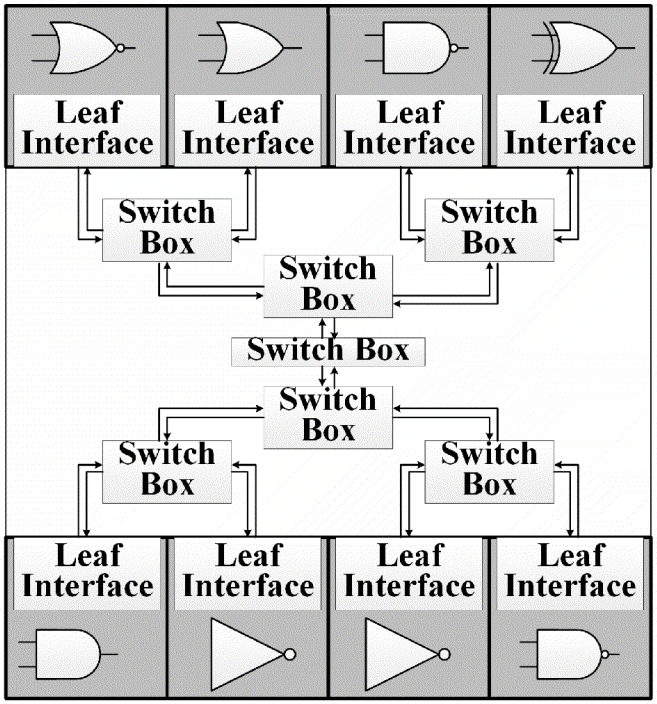
Static Region

PR Region

PRflow Using Vivado

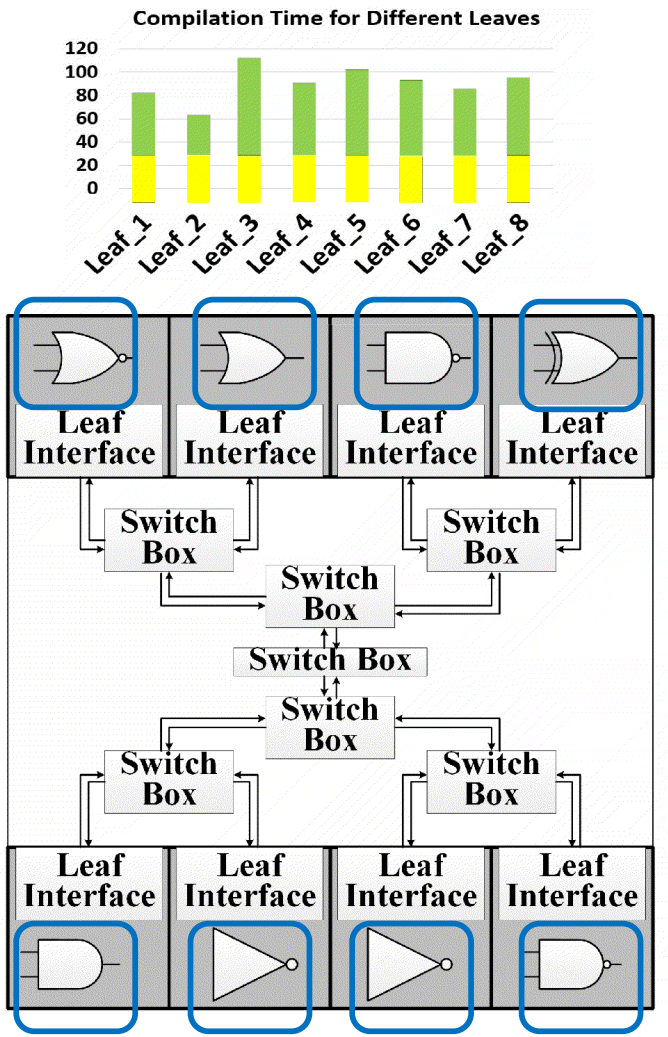
- Partial Reconfiguration mapping time increase with size of logic mapped
- Large fixed mapping time is independent of logic
  - Load up full device description
  - Map static region
    - time proportional to logic in static region
- Implications
  - Lower bound on speedup
  - Premium to minimize logic in static region
    - Mitigation: move PS BFT Overlay network out of static region





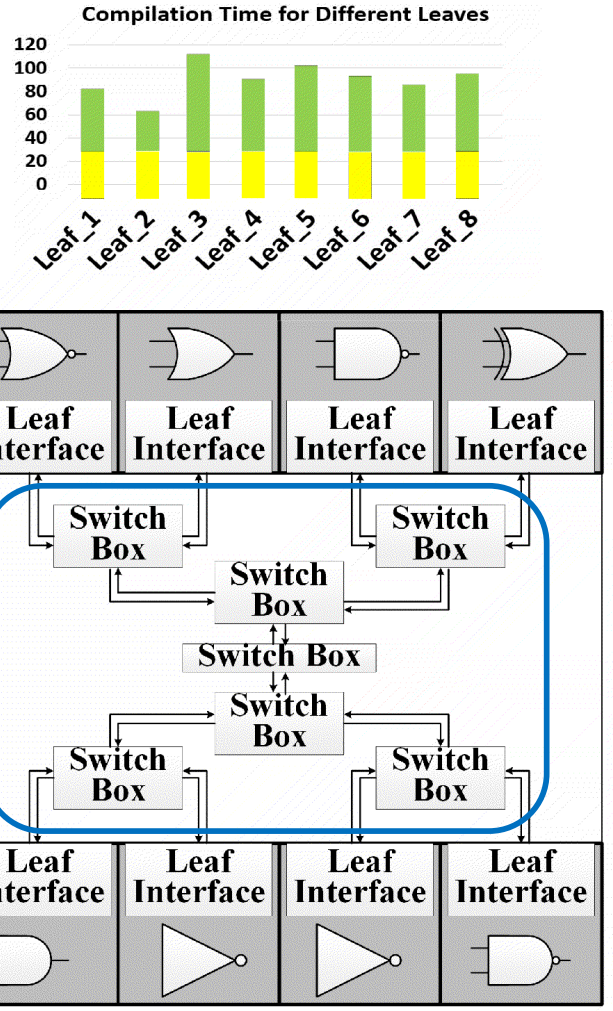
PRflow Using Vivado

- Partial Reconfiguration mapping time increase with size of logic mapped
- Large fixed mapping time is independent of logic
  - Load up full device description
  - Map static region
    - time proportional to logic in static region
- Implications
  - Lower bound on speedup
  - Premium to minimize logic in static region
    - Mitigation: move PS BFT Overlay network out of static region



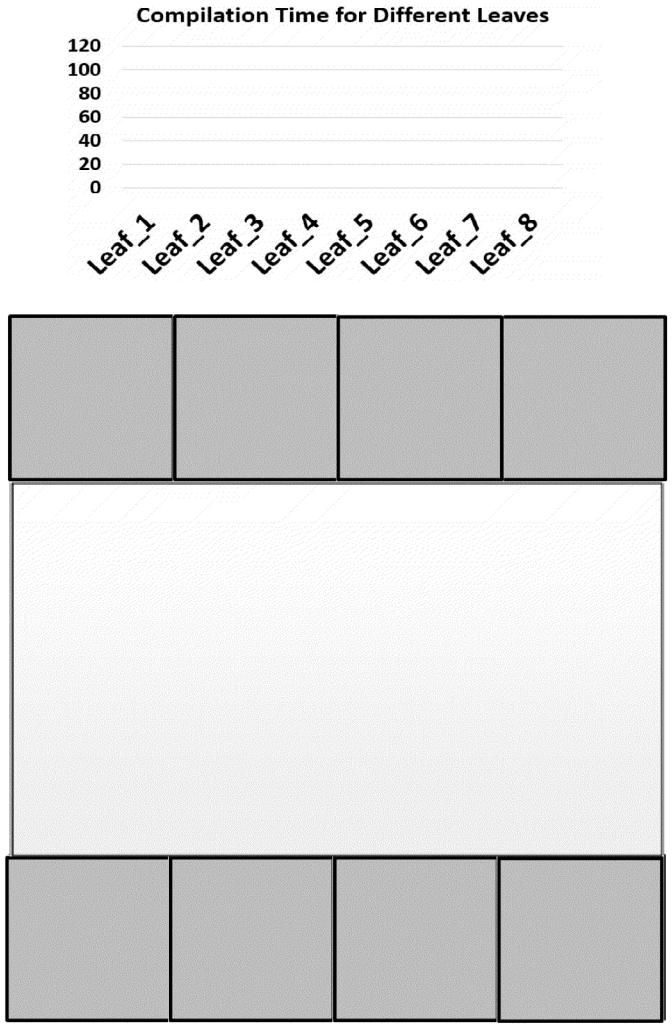
PRflow Using Vivado

- Partial Reconfiguration mapping time increase with size of logic mapped
- Large fixed mapping time is independent of logic
  - Load up full device description
  - Map static region
    - time proportional to logic in static region
- Implications
  - Lower bound on speedup
  - Premium to minimize logic in static region
    - Mitigation: move PS BFT Overlay network out of static region



PRflow Using Vivado

- Partial Reconfiguration mapping time increase with size of logic mapped
- Large fixed mapping time is independent of logic
  - Load up full device description
  - Map static region
    - time proportional to logic in static region
- Implications
  - Lower bound on speedup
  - Premium to minimize logic in static region
    - Mitigation: move PS BFT Overlay network out of static region

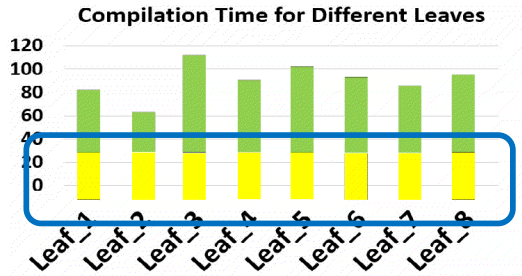


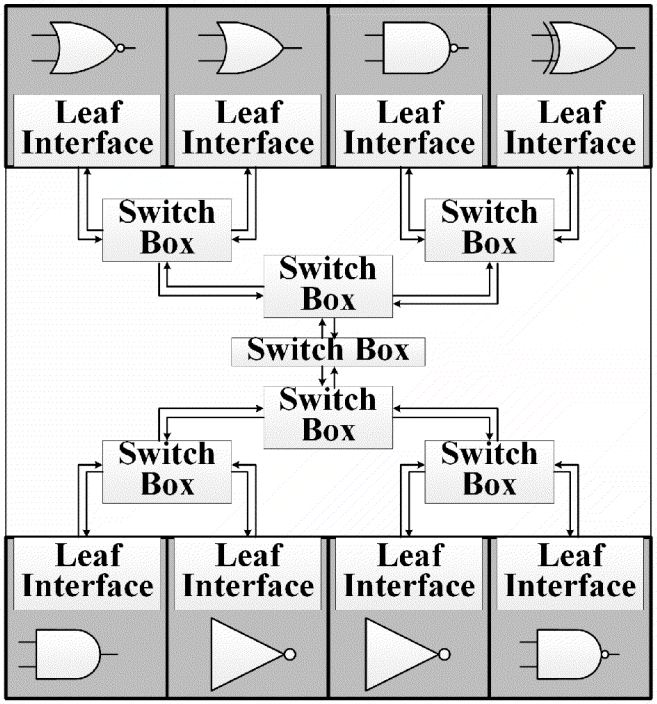
PRflow Using Vivado

- Implications
  - Lower bound on speedup
  - Premium to minimize logic in static region
    - Mitigation: move BFT overlay network out of static region

$$T_{new} = \frac{T_{origin}}{\#\ of\ Partitions}$$

$$T_{new} = \frac{T_{origin} - F_{fix}}{\#\ of\ Partitions} + T_{fix}$$



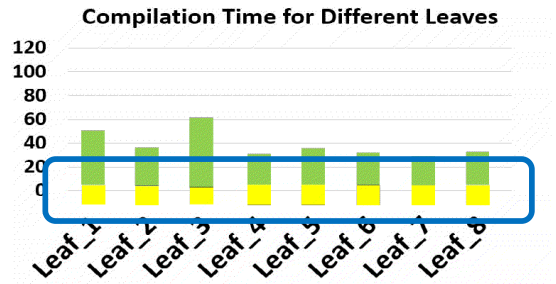


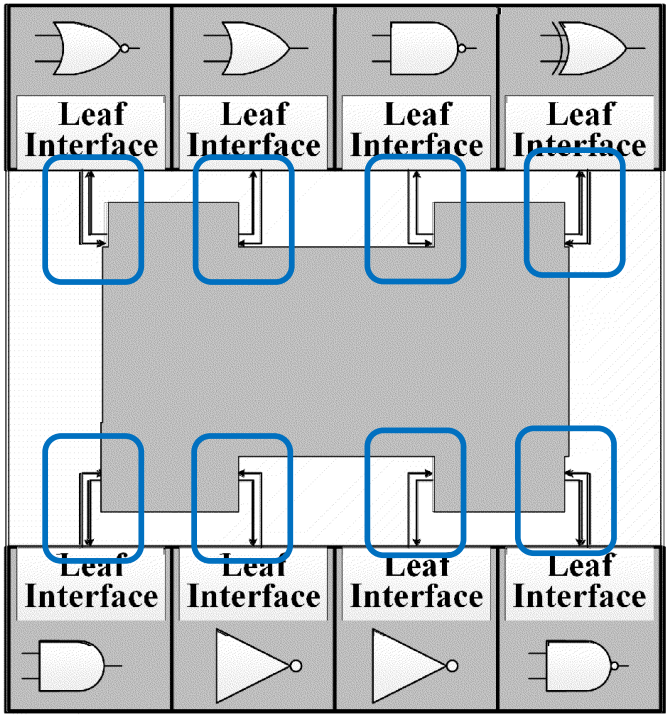
PRflow Using Vivado

- Implications
  - Lower bound on speedup
  - Premium to minimize logic in static region
    - Mitigation: move BFT overlay network out of static region

$$T_{new} = \frac{T_{origin}}{\#\ of\ Partitions}$$

$$T_{new} = \frac{T_{origin} - F_{fix}}{\#\ of\ Partitions} + T_{fix}$$

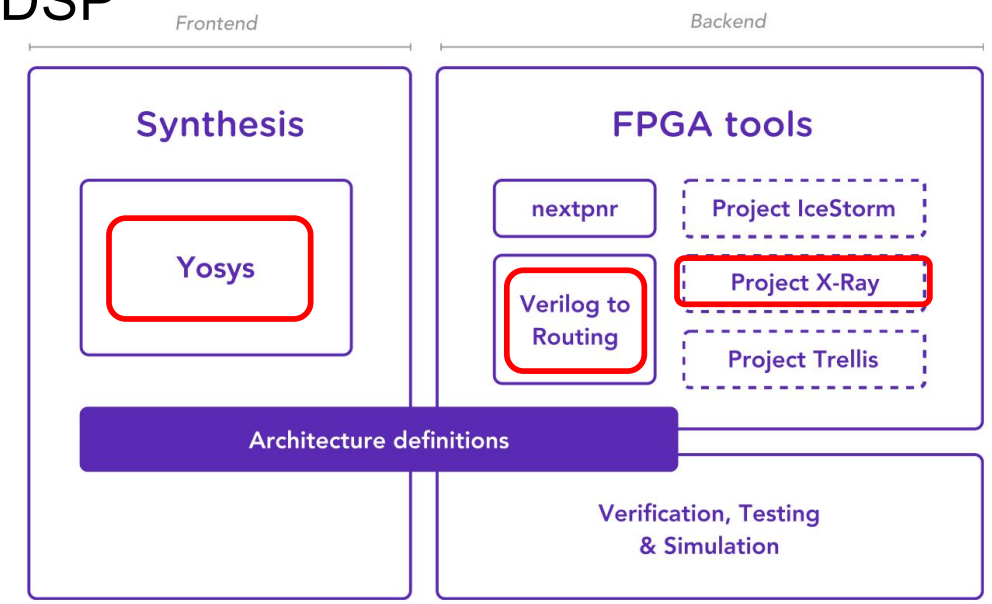




PRflow Using Symbiflow

An open-source Verilog-to-Bitstream FPGA Flow

- Synthesis
  - Yosys supports Logic, BRAM, DSP
- Implementation
  - VPR supports customized FPGA architecture
- Bitgen
  - X-Ray supports Xilinx 7 Series
  - Support Boards:
    - Digilent Arty A7-35T
    - Digilent Basys 3 Artix-7
    - Digilent Zybo Z7



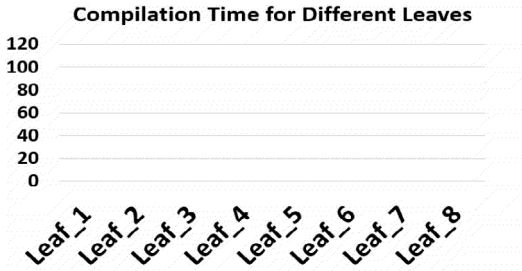
Source: https://symbiflow.github.io/#downloads

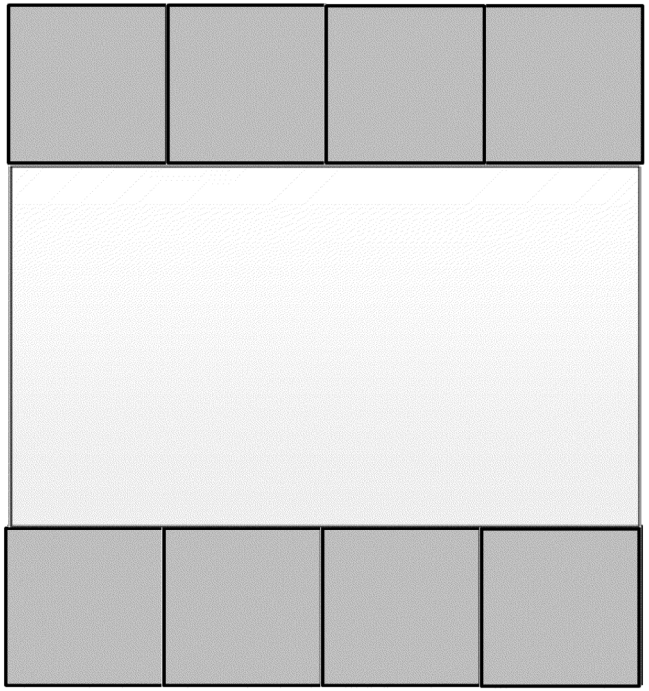
What can Symbiflow offer us?

- Avoid loading full chip database
- Avoid Mapping time for Fix logic
- Customize Quality vs. Runtime
- **Fixed time can go away!**

$$T_{new} = \frac{T_{origin} - F_{fix}}{\#\ of\ Partitions} + T_{fix}$$

$$T_{new} = \frac{T_{origin}}{\#\ of\ Partitions}$$



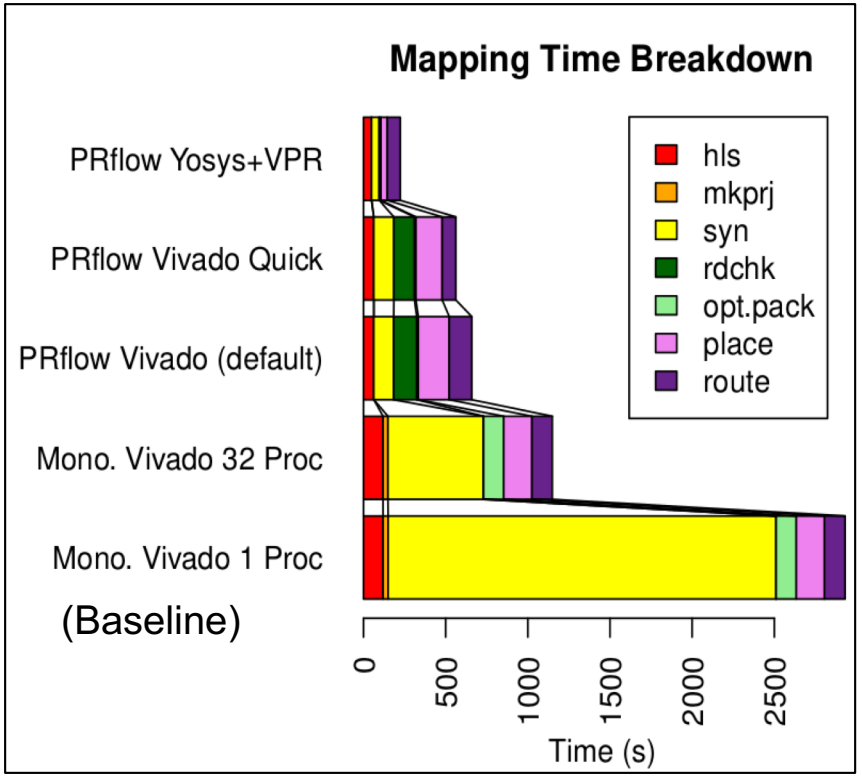


PRflow using Vivado vs. PRflow using Symbiflow

	PRflow on Vivado	PRflow on Symbiflow
Load complete device	Needed	**Not need**
Map static region	Needed	**Not need**
Quality vs. Runtime tradeoff	‘default’ or ‘quick’ mode	**Customizable**
Bitstream support	**All Xilinx Devices**	**7 Series**

Example Compilation Speedup

	Syn	Implementation			Total
		Cluster	Place	Route	
Vivado 1 process (**Baseline**)	2361s	123 s	171 s	124 s	**2931 s**
Vivado 32 processes	4X	1X	1X	1X	**2.55X**
Our PRflow on Vivado	19X	12.3X	0.9X	0.9X	**5.17X**
Our PRflow on Symbiflow	40X	4.0X	7.4X	3.1X	**15.8X**

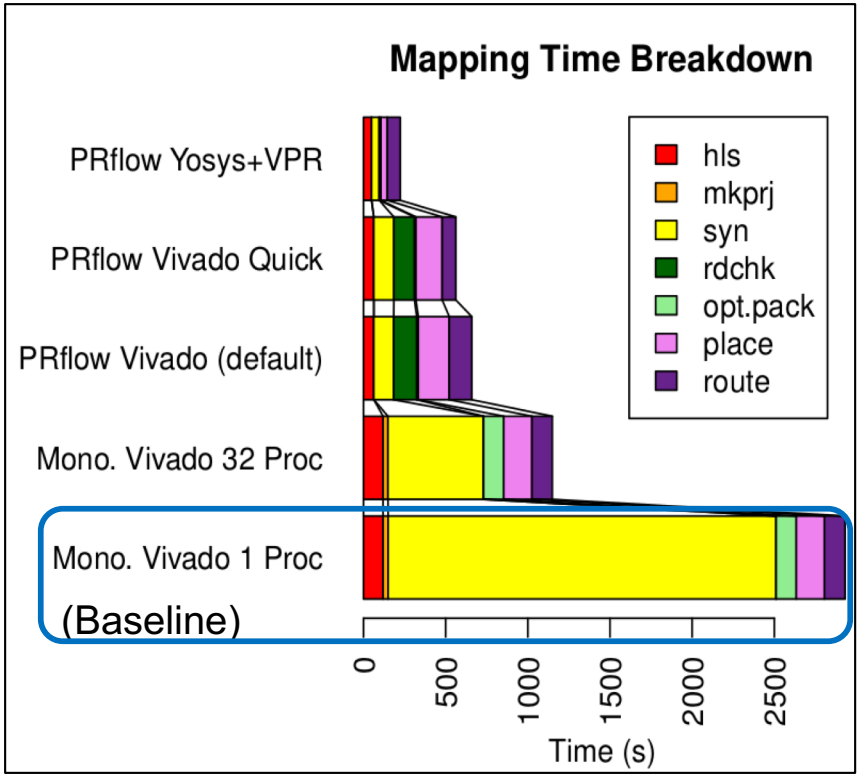


3D- Rendering Benchmark from Rosetta[1]

[1] Yuan Zhou et al. **Rosetta**: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs.Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Example Compilation Speedup

	Syn	Implementation			Total
		Cluster	Place	Route	
Vivado 1 process (**Baseline**)	2361s	123 s	171 s	124 s	**2931 s**
Vivado 32 processes	4X	1X	1X	1X	**2.55X**
Our PRflow on Vivado	19X	12.3X	0.9X	0.9X	**5.17X**
Our PRflow on Symbiflow	40X	4.0X	7.4X	3.1X	**15.8X**

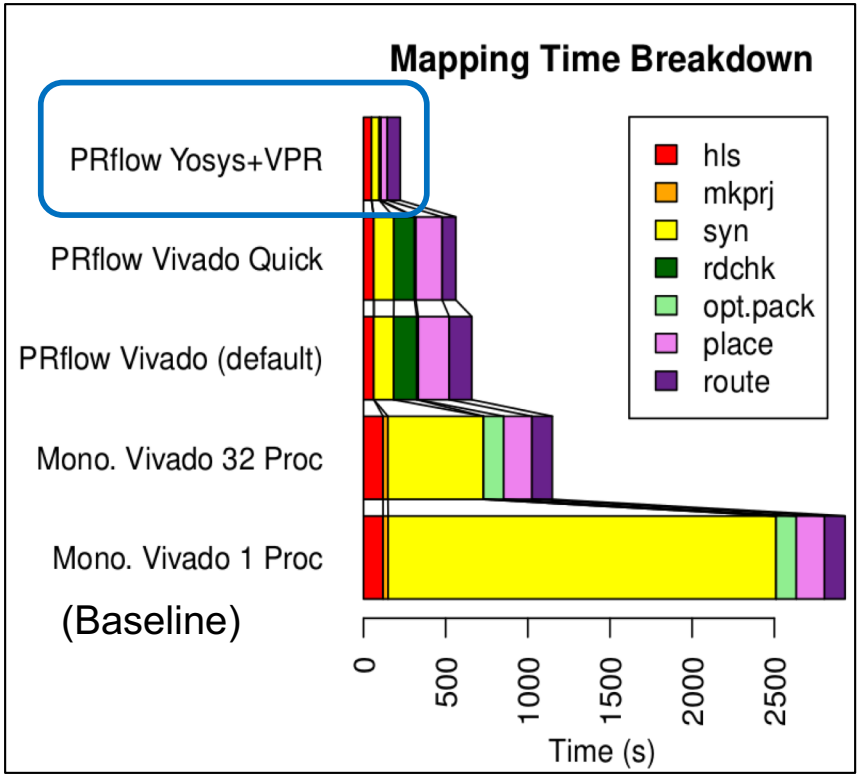


3D- Rendering Benchmark from Rosetta[1]

[1] Yuan Zhou et al. **Rosetta**: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs.Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Example Compilation Speedup

	Syn	Implementation			Total
		Cluster	Place	Route	
Vivado 1 process (**Baseline**)	2361s	123 s	171 s	124 s	**2931 s**
Vivado 32 processes	4X	1X	1X	1X	**2.55X**
Our PRflow on Vivado	19X	12.3X	0.9X	0.9X	**5.17X**
Our PRflow on Symbiflow	40X	4.0X	7.4X	3.1X	**15.8X**

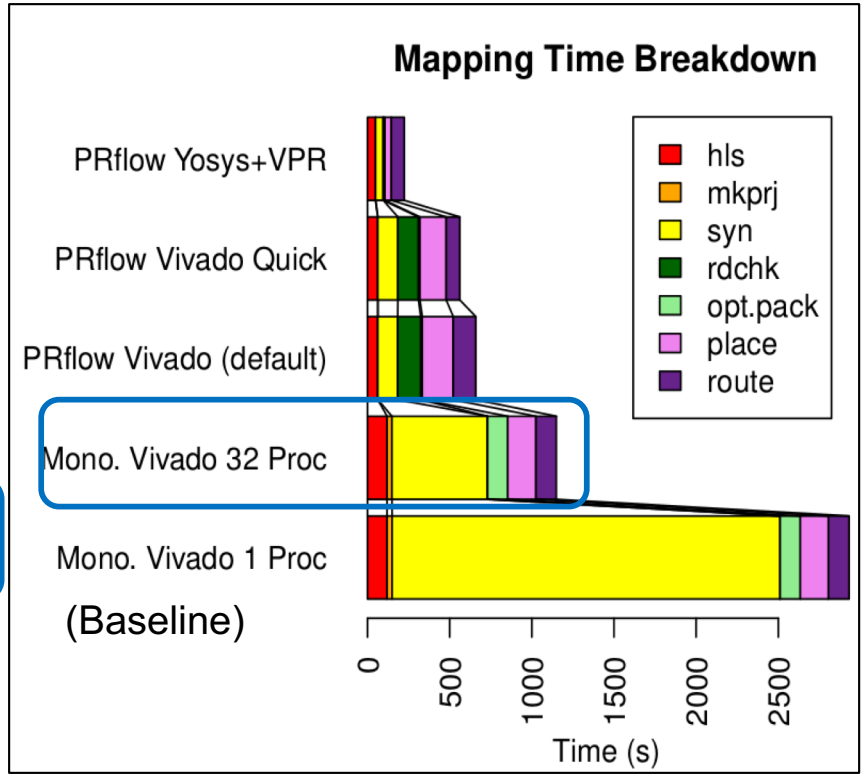


3D- Rendering Benchmark from Rosetta[1]

[1] Yuan Zhou et al. **Rosetta**: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs.Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Example Compilation Speedup

	Syn	Implementation			Total
		Cluster	Place	Route	
Vivado 1 process (**Baseline**)	2361s	123 s	171 s	124 s	**2931 s**
Vivado 32 processes	4X	1X	1X	1X	**2.55X**
Our PRflow on Vivado	19X	12.3X	0.9X	0.9X	**5.17X**
Our PRflow on Symbiflow	40X	4.0X	7.4X	3.1X	**15.8X**

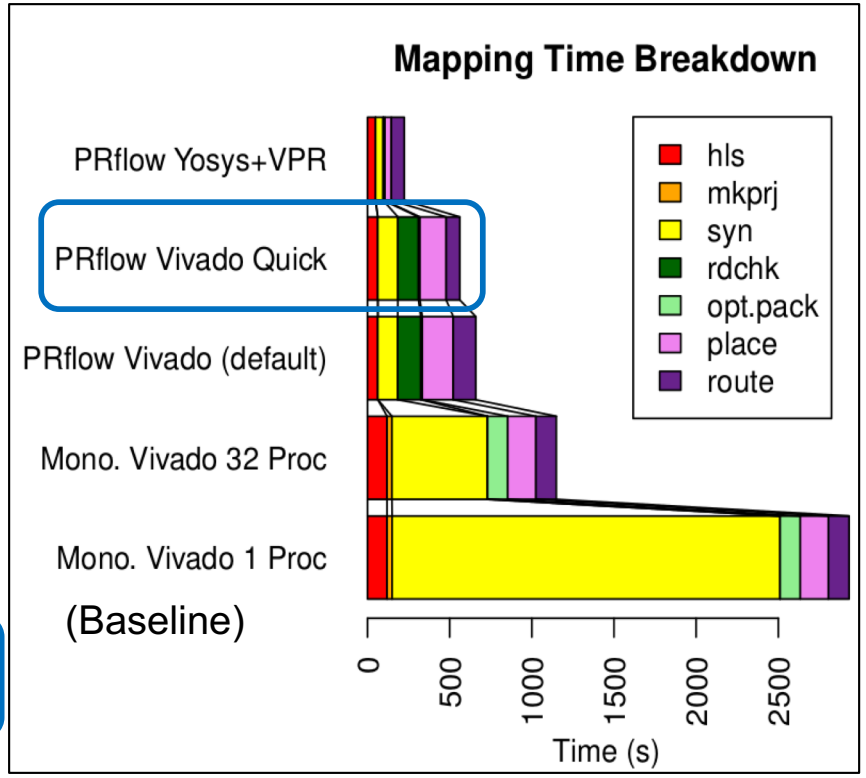


3D- Rendering Benchmark from Rosetta[1]

[1] Yuan Zhou et al. **Rosetta**: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs.Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Example Compilation Speedup

	Syn	Implementation			Total
		Cluster	Place	Route	
Vivado 1 process (**Baseline**)	2361s	123 s	171 s	124 s	**2931 s**
Vivado 32 processes	4X	1X	1X	1X	**2.55X**
Our PRflow on Vivado	19X	12.3X	0.9X	0.9X	**5.17X**
Our PRflow on Symbiflow	40X	4.0X	7.4X	3.1X	**15.8X**

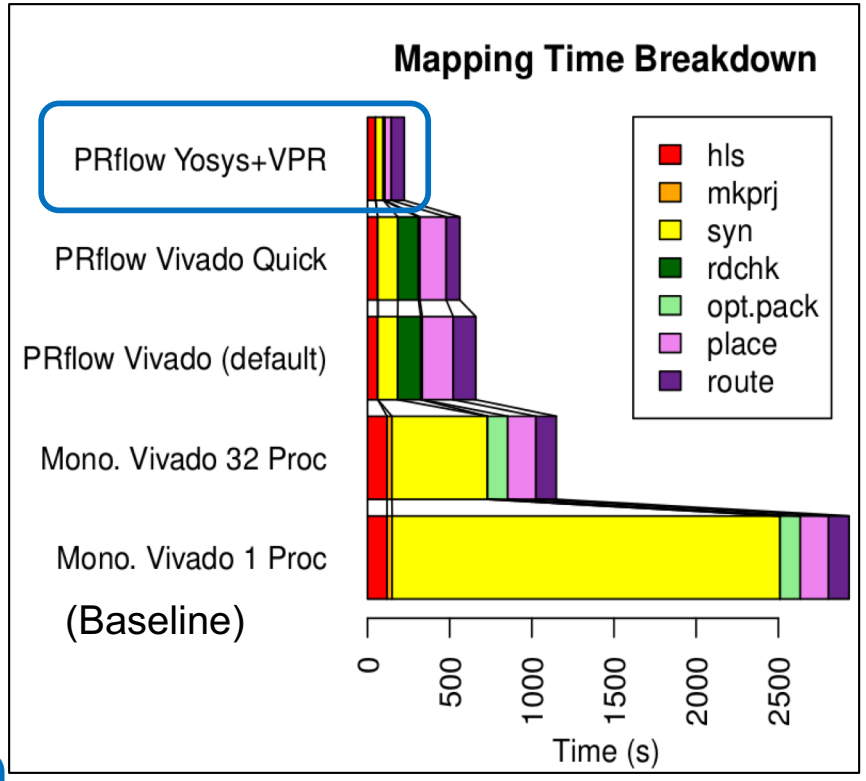


3D- Rendering Benchmark from Rosetta[1]

[1] Yuan Zhou et al. **Rosetta**: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs.Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Example Compilation Speedup

	Syn	Implementation			Total
		Cluster	Place	Route	
Vivado 1 process (**Baseline**)	2361s	123 s	171 s	124 s	**2931 s**
Vivado 32 processes	4X	1X	1X	1X	**2.55X**
Our PRflow on Vivado	19X	12.3X	0.9X	0.9X	**5.17X**
Our PRflow on Symbiflow	40X	4.0X	7.4X	3.1X	**15.8X**

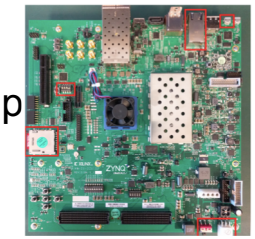


3D- Rendering Benchmark from Rosetta[1]

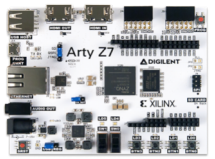
[1] Yuan Zhou et al. **Rosetta**: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs.Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

Methodology

- A cluster of 8 compute servers
  - Dual 2.7GHz Intel **E5-2680 CPUs**, 128GB of RAM (total of 8x2x8=128 cores)
- Platform for PRflow on Vivado 2018.2
  - **Xilinx ZCU102 board** with xczu9eg-ffvb1155-2-e MP-SOC chip
  - 274K LUTs, 912 BRAM36, 2520 DSPs
  - 775 MHz clock for Fabric



- Platform for PRflow on Symbiflow
  - **Digilent Arty A7-35T** with XC7A35TICSG324-1L FPGA chip
  - 21K LUTs, 50 BRAM36, 90 DSPs
  - 464MHz clock for Fabric

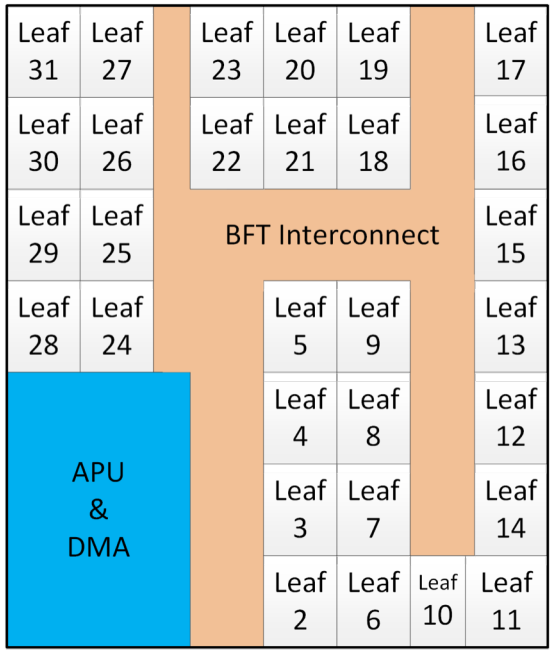


- Rosetta HLS Benchmark [1]
  - 6 C-based design for High Level Synthesis Benchmark
  - 3-D Rendering, Digit-Recognition, Spam-filter, Optical-flow, BNN, Face-detection
  - We partitioned the benchmarks into small pieces, details in the paper

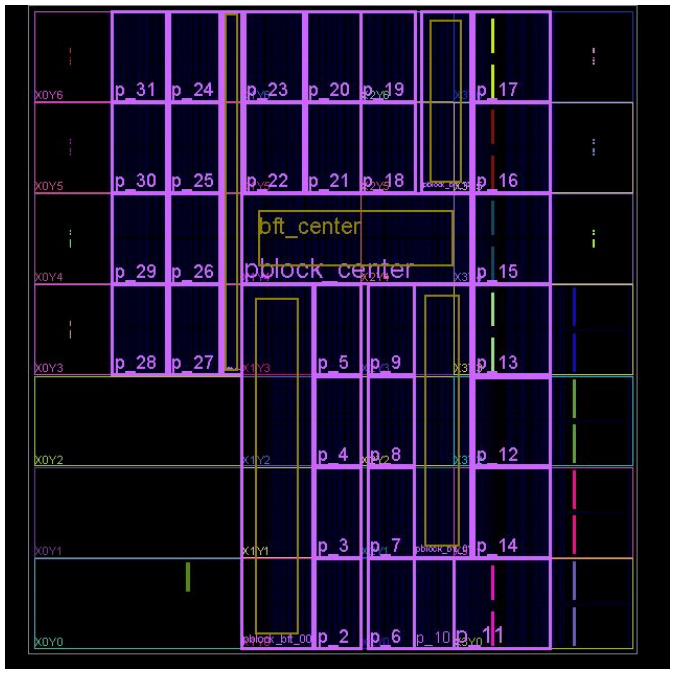
[1] Yuan Zhou et al. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs.Int'l Symp. on Field-Programmable Gate Arrays (FPGA), Feb 2018.

- 30 leaves for application logic
- 1 leaf for 4-core ARM processor
- 1 leaf for DMA interface
- 32 leaves are connected by BFT

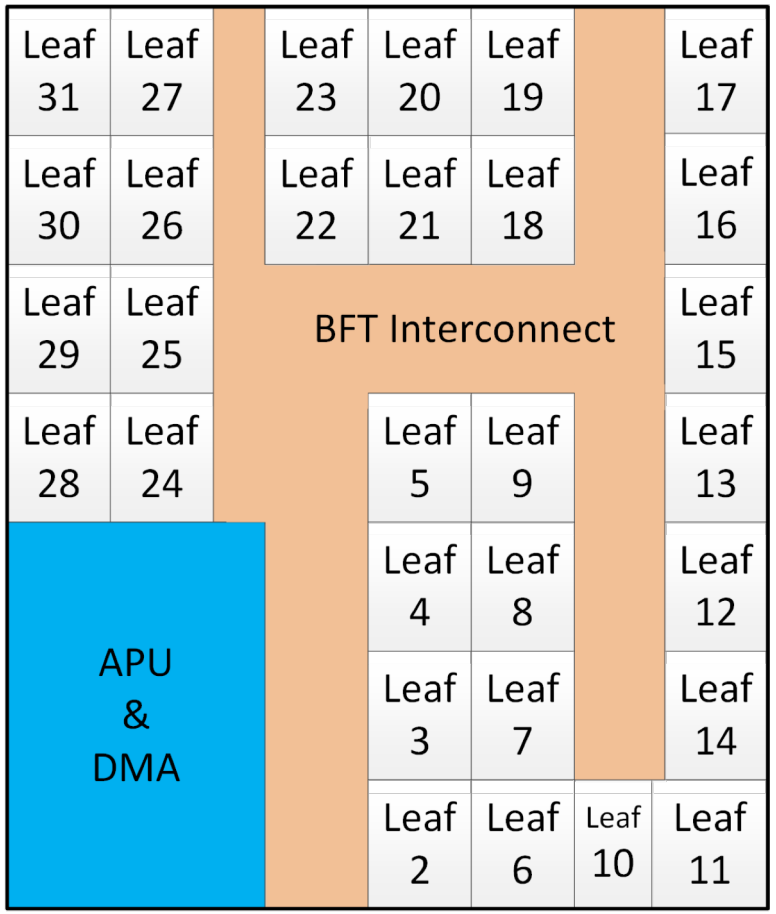
Floorplan for ZCU9EG



Physical Layout



Floorplan for ZCU9EG



Resource Distribution

Type	LUT	FF	RAM18	DSP	# of Leaf
1	5760	11520	48	48	12
2	4800	9600	24	72	4
3	4800	9600	48	48	4
4	5760	11520	24	72	2
5	6720	13440	48	48	6
6	4320	8640	24	48	1
7	9120	18240	72	48	1
Total	**173K**	**345K**	**1296**	**1584**	**30**

Rosetta Benchmark Compilation Time (seconds)

Design	SDSoC	PRflow on Vivado	PRflow on Yosys & VPR
Digit Recognition	2472	638 (3↑)	337 **(7↑)**
SPAM Filter	1770	658 (2.7↑)	295 **(6↑)**
3-D Rendering	1769	659 (2.7↑)	185 **(9↑)**
Optical Flow	2660	744 (3↑)	311 **(8↑)**
Binarized NN	10726	1000 (10↑)	309 **(34↑)**
Face Detection	4347	972 (4↑)	–†
Average Speedup	**1X**	**4.6↑**	**12.9↑**

PRflow on Vivado

- Speedup from **2.7x to 10.72x**

PRflow on Yosys&VPR

- Speedup **6x to 34.7x**
- † : Some leaf cannot be mapped due to complex interconnect and floating point multipliers

Rosetta Benchmark Compilation Time (seconds)

Design	SDSoC	PRflow on Vivado	PRflow on Yosys & VPR
Digit Recognition	2472	638 (3↑)	337 **(7↑)**
SPAM Filter	1770	658 (2.7↑)	295 **(6↑)**
3-D Rendering	1769	659 (2.7↑)	185 **(9↑)**
Optical Flow	2660	744 (3↑)	311 **(8↑)**
Binarized NN	10726	1000 (10↑)	309 **(34↑)**
Face Detection	4347	972 (4↑)	–†
Average Speedup	**1X**	**4.6↑**	**12.9↑**

PRflow on Vivado

- Speedup from **2.7x to 10.72x**

PRflow on Yosys&VPR

- Speedup **6x to 34.7x**
- † : Some leaf cannot be mapped due to complex interconnect and floating point multipliers

Rosetta Benchmark Compilation Time (seconds)

Design	SDSoC	PRflow on Vivado	PRflow on Yosys & VPR
Digit Recognition	2472	638 (3↑)	337 **(7↑)**
SPAM Filter	1770	658 (2.7↑)	295 **(6↑)**
3-D Rendering	1769	659 (2.7↑)	185 **(9↑)**
Optical Flow	2660	744 (3↑)	311 **(8↑)**
Binarized NN	10726	1000 (10↑)	309 **(34↑)**
Face Detection	4347	972 (4↑)	_†
Average Speedup	**1↑**	**4.6↑**	**12.9↑**

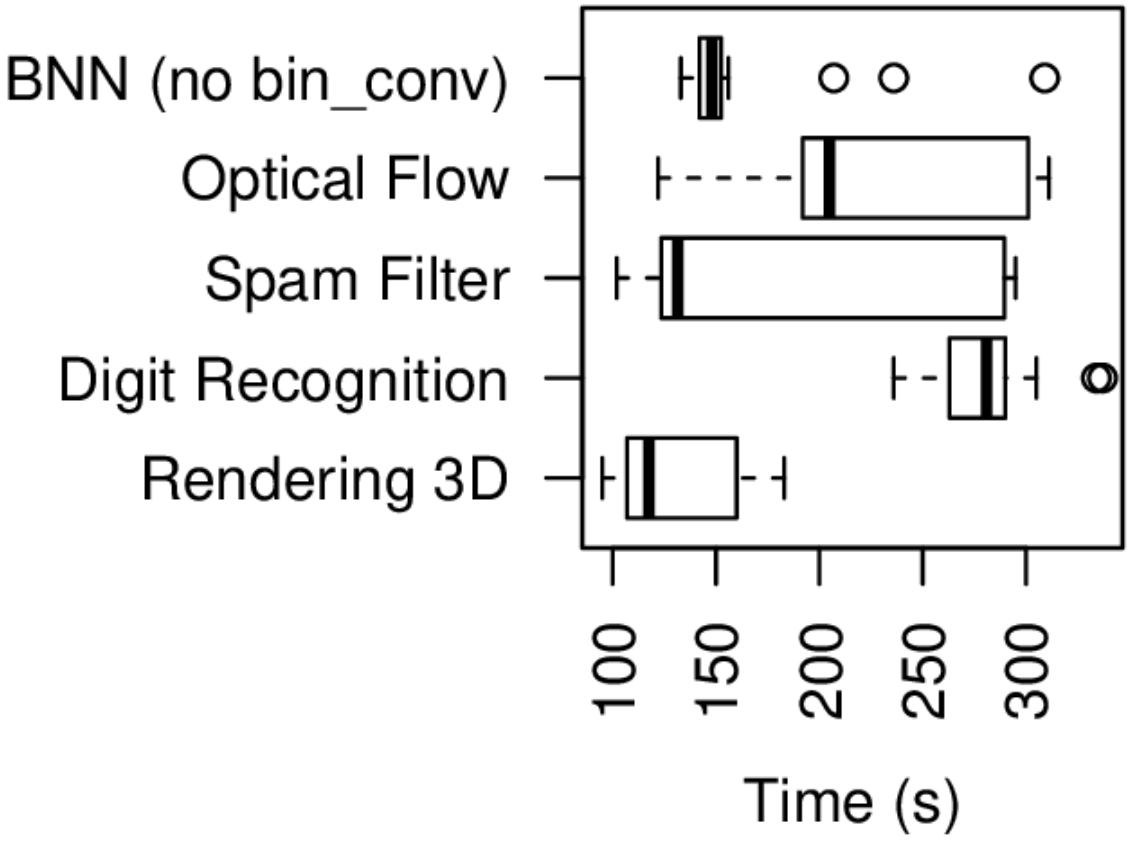
PRflow on Vivado

- Speedup from **2.7x to 10.72x**

PRflow on Yosys&VPR

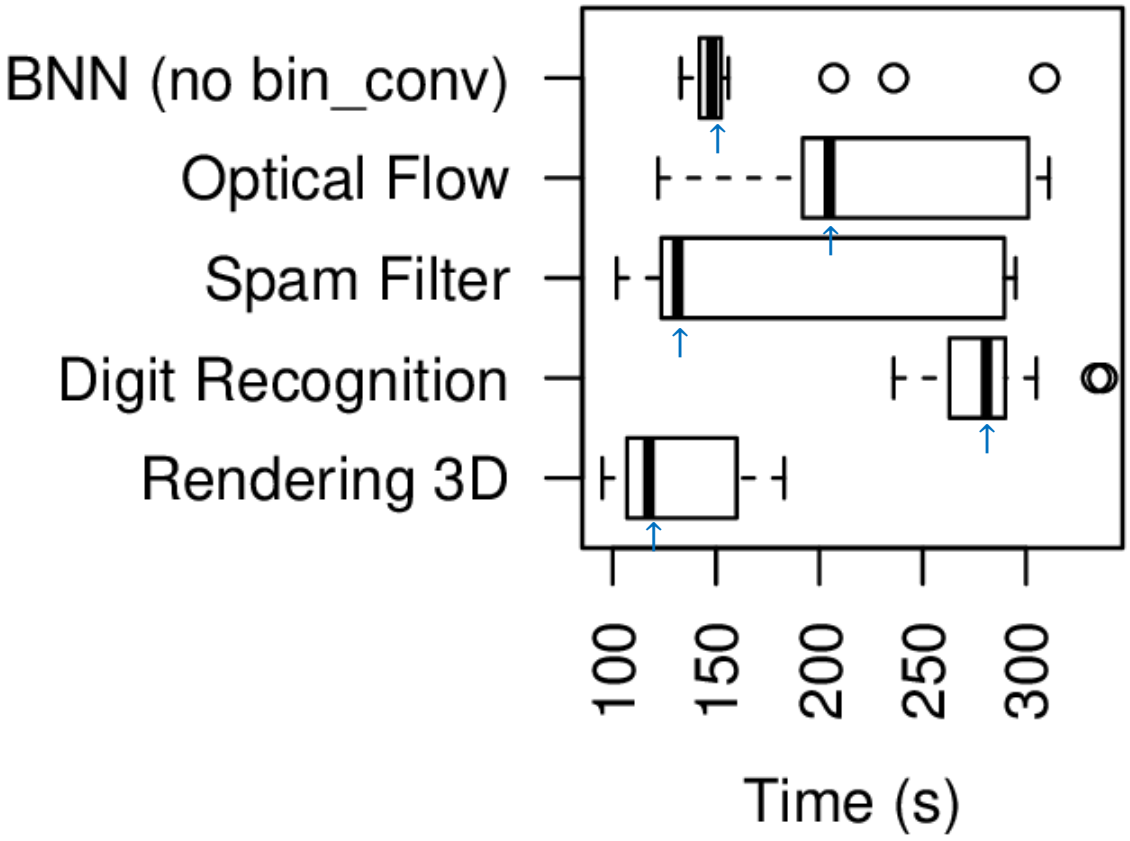
- Speedup **6x to 34.7x**
- † : Some leaf cannot be mapped due to complex interconnect and floating point multipliers

Distribution of Compilation Time on Symbiflow



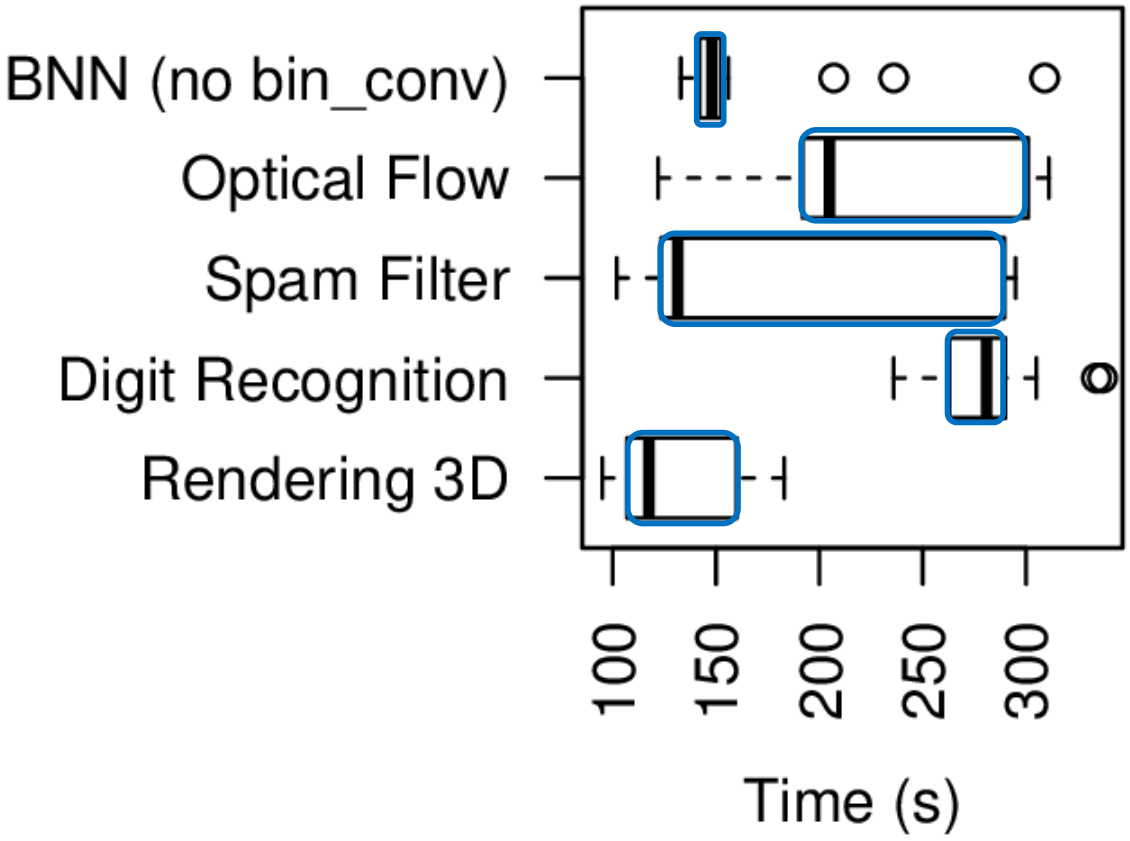
- Mapping time from all the design pieces
- Most of them are within 5 minutes
- Run pretty fast for most single-leaf changes

Distribution of Compilation Time on Symbiflow



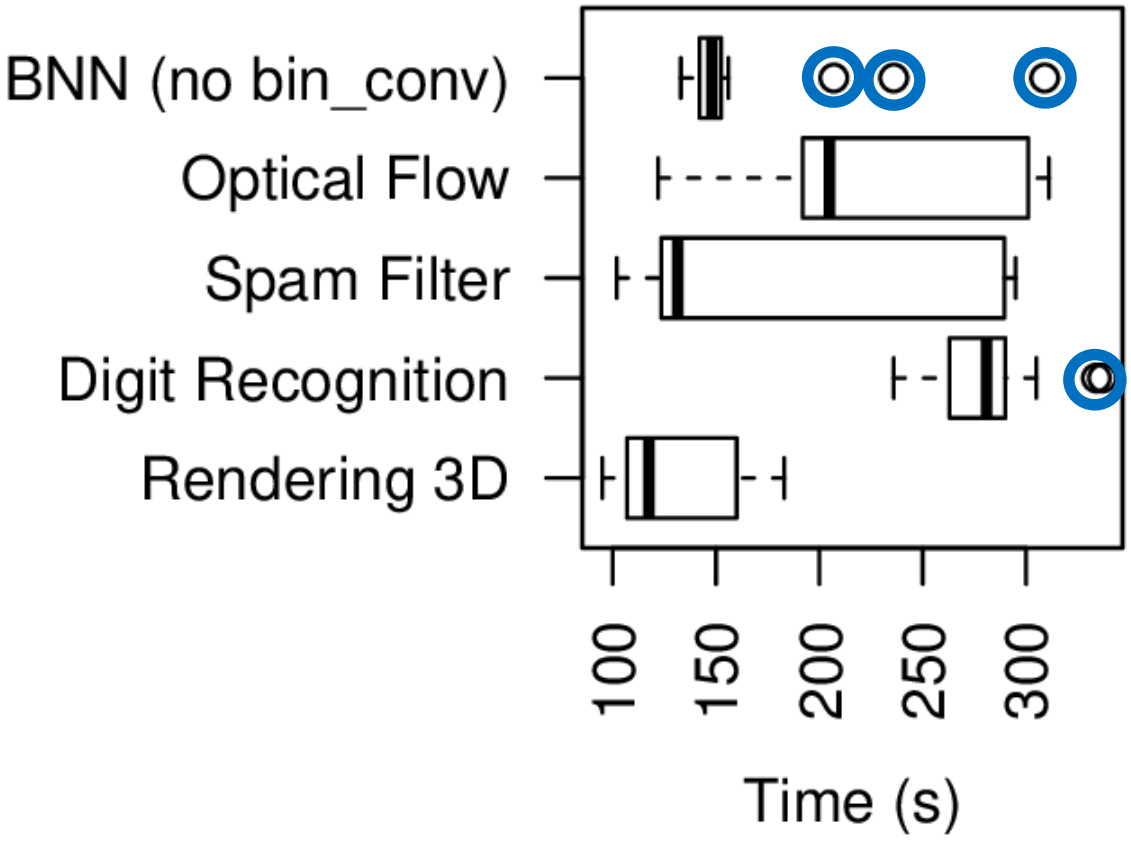
- Mapping time from all the design pieces
- Most of them are within 5 minutes
- Run pretty fast for most single-leaf changes

Distribution of Compilation Time on Symbiflow



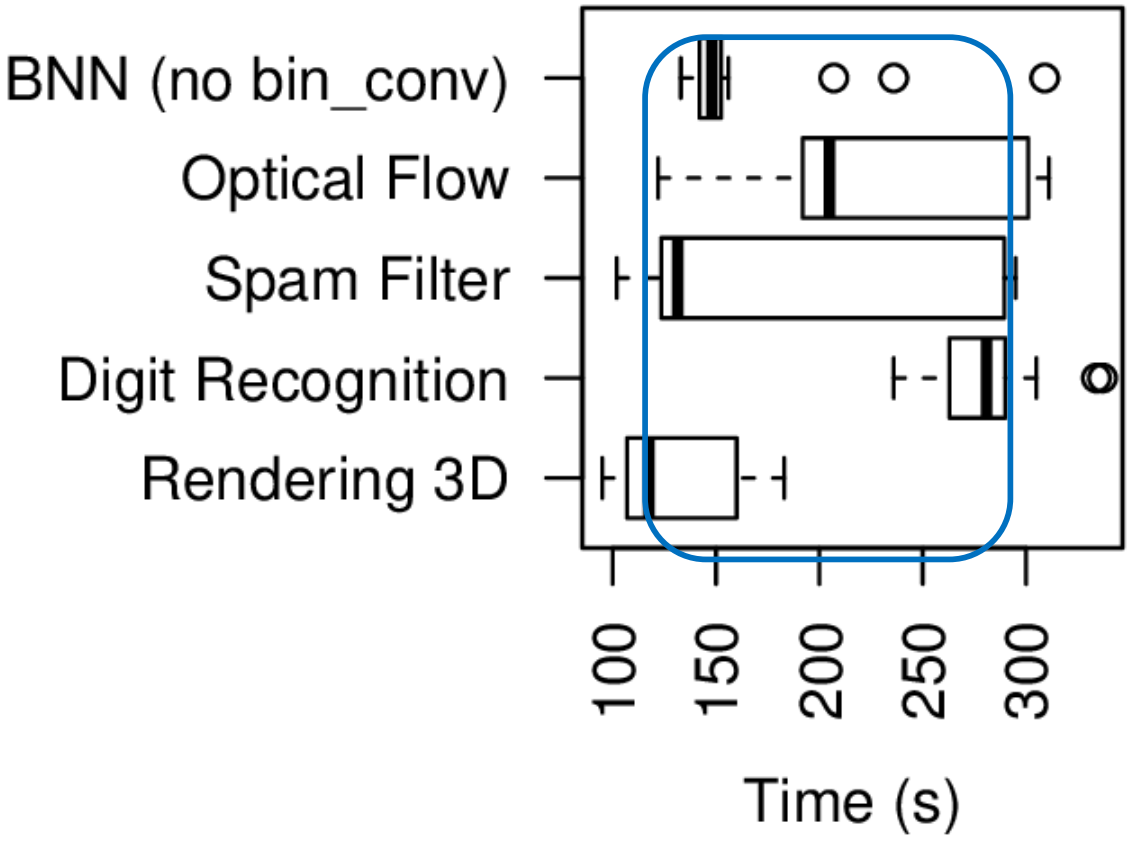
- Mapping time from all the design pieces
- Most of them are within 5 minutes
- Run pretty fast for most single-leaf changes

Distribution of Compilation Time on Symbiflow



- Mapping time from all the design pieces
- Most of them are within 5 minutes
- Run pretty fast for most single-leaf changes

Distribution of Compilation Time on Symbiflow



- Mapping time from all the design pieces
- Most of them are within 5 minutes
- Run pretty fast for most single-leaf changes

Performance Comparison

Runtime per input frame (ms)

Design	SDSoC	Our PRflow
Digit Recognition	6.17	1.18
SPAM Filter	13	16.58
3-D Rendering	82.13	48.90
Optical Flow	6.35	25.80
Binarized NN	5.3	17.42
Face Detection	28.19	351.93

- SDSoC is run on default 100MHz
- PRflow is with 300MHz BFT and 200MHz user logic
- Some cases, we can get the same or better performance
- The IO bottlenecks of BFT constrain some benchmarks performance

Performance Comparison

Runtime per input frame (ms)

Design	SDSoC	Our PRflow
Digit Recognition	6.17	1.18
SPAM Filter	13	16.58
3-D Rendering	82.13	48.90
Optical Flow	6.35	25.80
Binarized NN	5.3	17.42
Face Detection	28.19	351.93

- SDSoC is run on default 100MHz
- PRflow is with 300MHz BFT and 200MHz user logic
- Some cases, we can get the same or better performance
- The IO bottlenecks of BFT constrain some benchmarks performance

Area Comparison (LUTs)

Design	SDSoC	Our PRflow
Digit Recognition	14.11%	48.07%
SPAM Filter	4.65%	41.08%
3-D Rendering	3.24%	36.53%
Optical Flow	14.15%	43.89%
Binarized NN	16.84%	42.31%
Face Detection	24.73%	59.26%

- We use BFT and Interface to link small pieces up
- The platform costs us fixed 35% LUTs overhead
- 35% FFs overhead
- 40% BRAM overhead
- 40% DSPs overhead

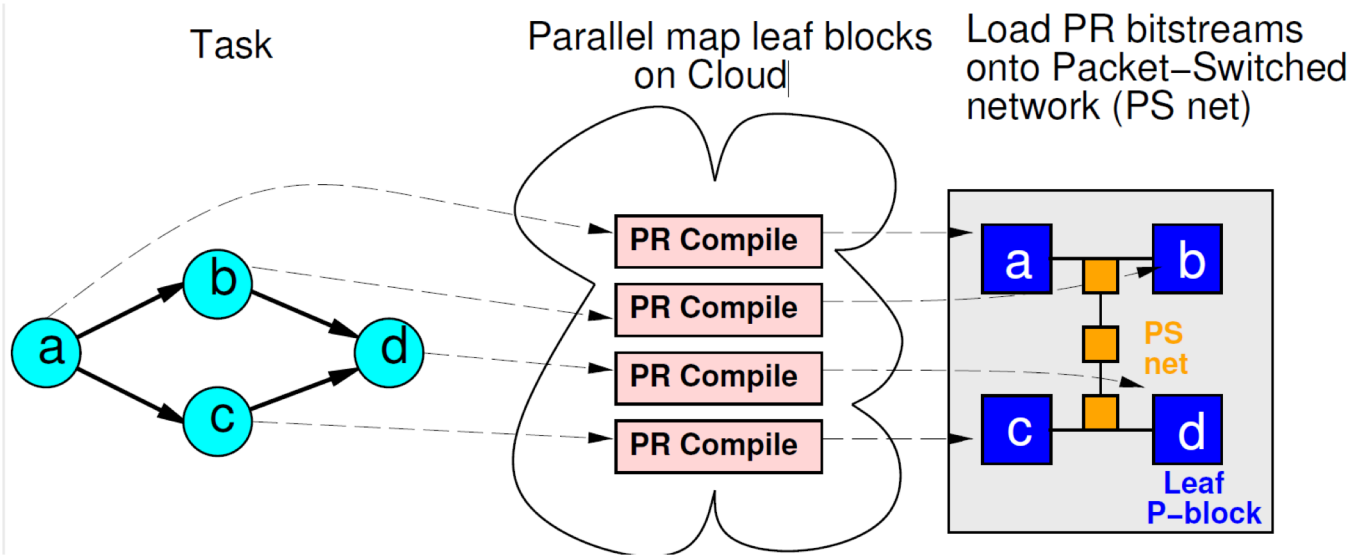
Future work:

- IO bottleneck
  - Direct interconnect between leaves
- Vivado Improvement
  - Like Symbiflow to avoid fix time for static region?
- Symbiflow Support
  - For More series like UltraScale+ MPSoC
  - Floating point multipliers and smarter P&R tool
- Automatic Design Partitioning
  - Use Stylized C/C++ patterns

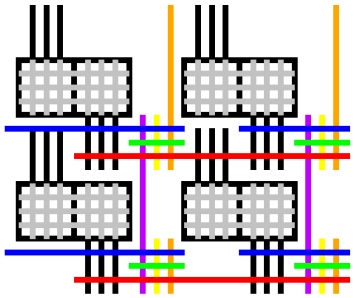
Ideas:

- Divide-and-conquer compilation strategy based on utilizing partial reconfiguration

$$T_{new} = \frac{T_{origin}}{\#\ of\ Partition}$$



Penn



Conclusion

- Compilation time does not need to take hours
- Decomposition of the design into separate pieces
  - Small compilation tasks in parallel
  - Incremental compile just the part that changed
- **Impact:** Able to achieve **12-18 minutes** using Vivado
  - Contrast **42-160 minutes** no PRflow
- Plausible to achieve **2-5 minutes** with open source Symbiflow

Thank you

Q&A

Area Comparison (FFs)

Design	SDSoC	Our PRflow
Digit Recognition	3.5%	42.10%
SPAM Filter	2.7%	36.79%
3-D Rendering	3.24%	33.9%
Optical Flow	1.76%	36.48%
Binarized NN	7.53%	37.35%
Face Detection	14.1%	46.11%

- We use BFT and Interface to link small pieces up
- The platform costs us fixed 35% LUTs overhead
- 35% FFs overhead
- 40% BRAM overhead
- 40% DSPs overhead

Area Comparison (BRAMs)

Design	SDSoC	Our PRflow
Digit Recognition	33.55%	60.31%
SPAM Filter	8.0%	48.46%
3-D Rendering	7.73%	38.15%
Optical Flow	10.14%	41.18%
Binarized NN	65.68%	92.59%
Face Detection	14.64%	63.15%

- We use BFT and Interface to link small pieces up
- The platform costs us fixed 35% LUTs overhead
- 35% FFs overhead
- 40% BRAM overhead
- 40% DSPs overhead

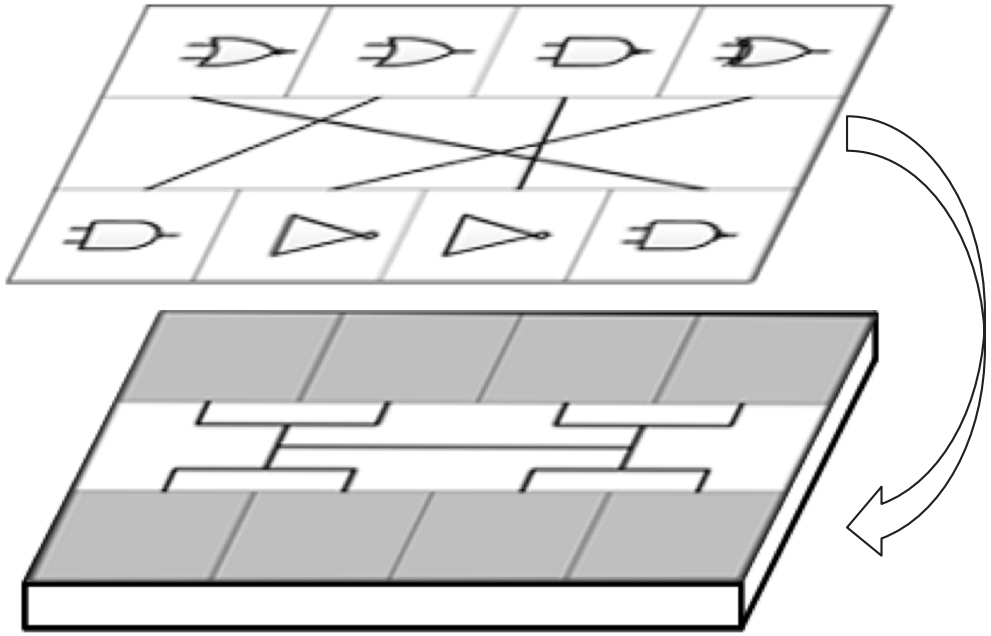
Area Comparison (DSPs)

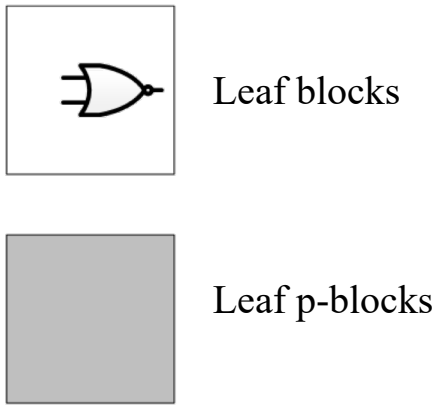
Design	SDSoC	Our PRflow
Digit Recognition	0.03%	35.23%
SPAM Filter	8.89%	45.30%
3-D Rendering	0%	35.23%
Optical Flow	4.92%	46.42%
Binarized NN	0.11%	35.48%
Face Detection	3.13%	39.64%

- We use BFT and Interface to link small pieces up
- The platform costs us fixed 35% LUTs overhead
- 35% FFs overhead
- 40% BRAM overhead
- 40% DSPs overhead

Ideas:

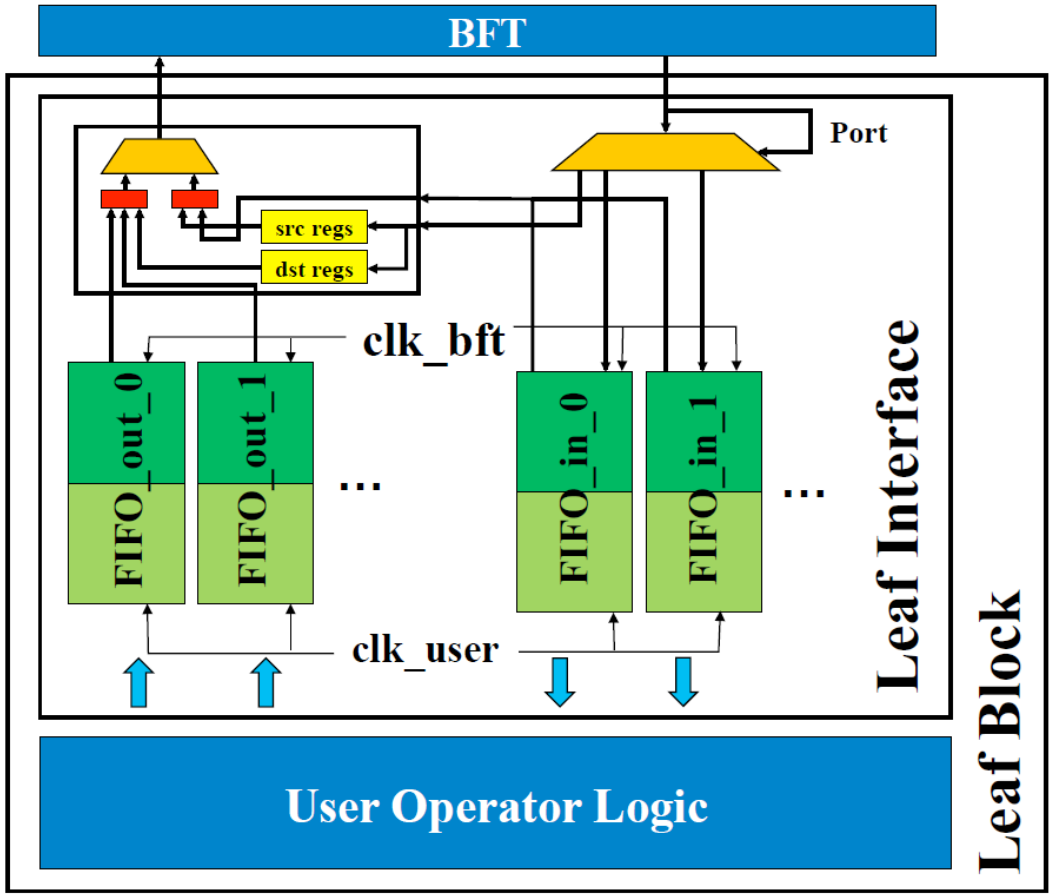
- Divide-and-conquer Compilation Strategy based on Utilize Partial Reconfiguration





Ideas:

- Leaf Interface
  - Packet-switched: Arbitrary interconnection between 2 leaves
  - **Leaf interface: Arbitrary number of inputs and outputs**



- Implementation time is not related to p-block size, but logic size

TABLE I
IMPLEMENTATION TIME VS. DESIGN AND P-BLOCK SIZE ON XCZU9EG

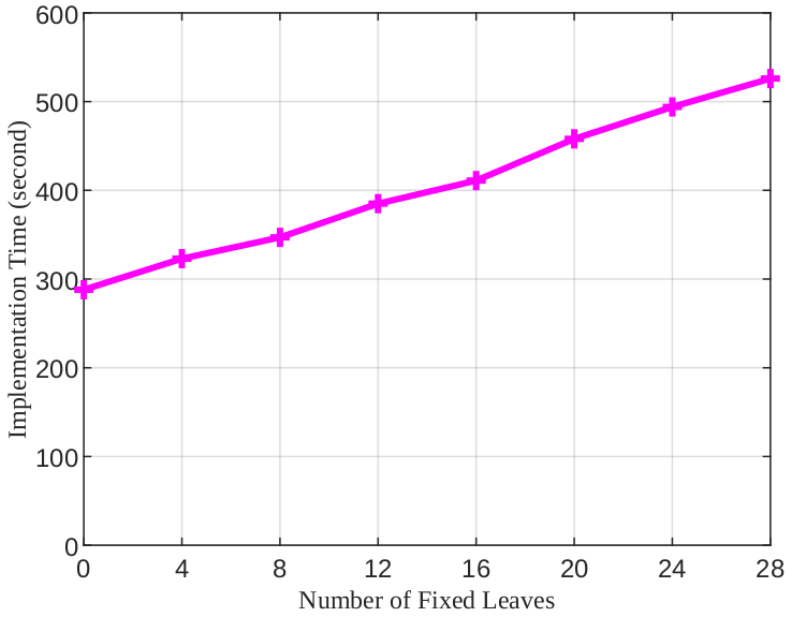
		P-Block Size (LUTs)				
Design	**Size**	3960	6160	7920	10120	15840
Shift Register	623	203	206	206	205	204
	1633	210	210	210	208	210
	2661	220	218	218	217	217
	3614	229	233	224	227	225
	4616		239	239	234	237
	5623		239	244	241	242
MicroBlaze Cores	1435	182	181	180	181	185
	2860	196	192	195	192	198
	4285		210	211	210	207
	5710		605	231	223	226

(cells show compilation time in seconds)

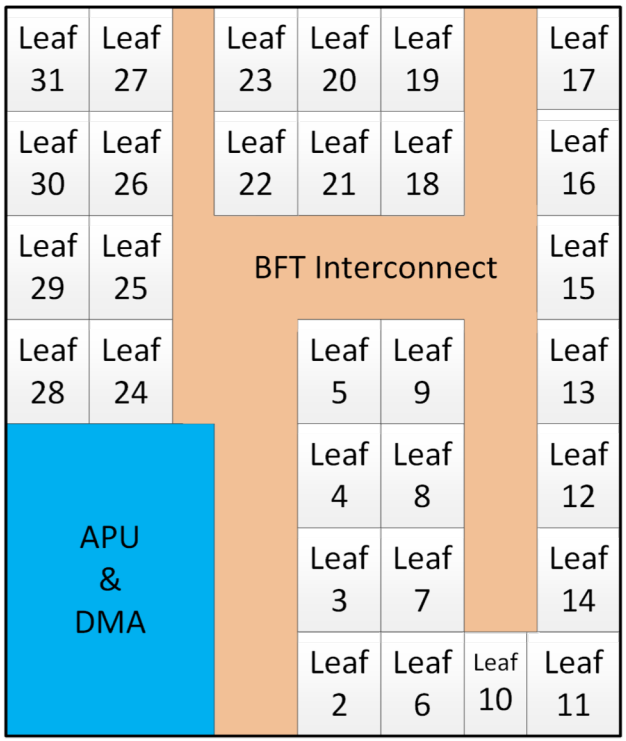
- Logic in static region affect leaf compilation time

TABLE II
STATIC REGION IMPACT ON IMPLEMENTATION TIME (32 LEAF BFT WITH 32B PAYLOAD WIDTH DATAPATH)

	PR implementation		OoC impl.
	BFT in Static	BFT as P-block	leaf only
LUTs in mapping	30611	8590	1435
optimize time (s)	29	10	79
place time (s)	238	161	27
route time (s)	170	113	74
total time (s)	437	284	180



Implementation:

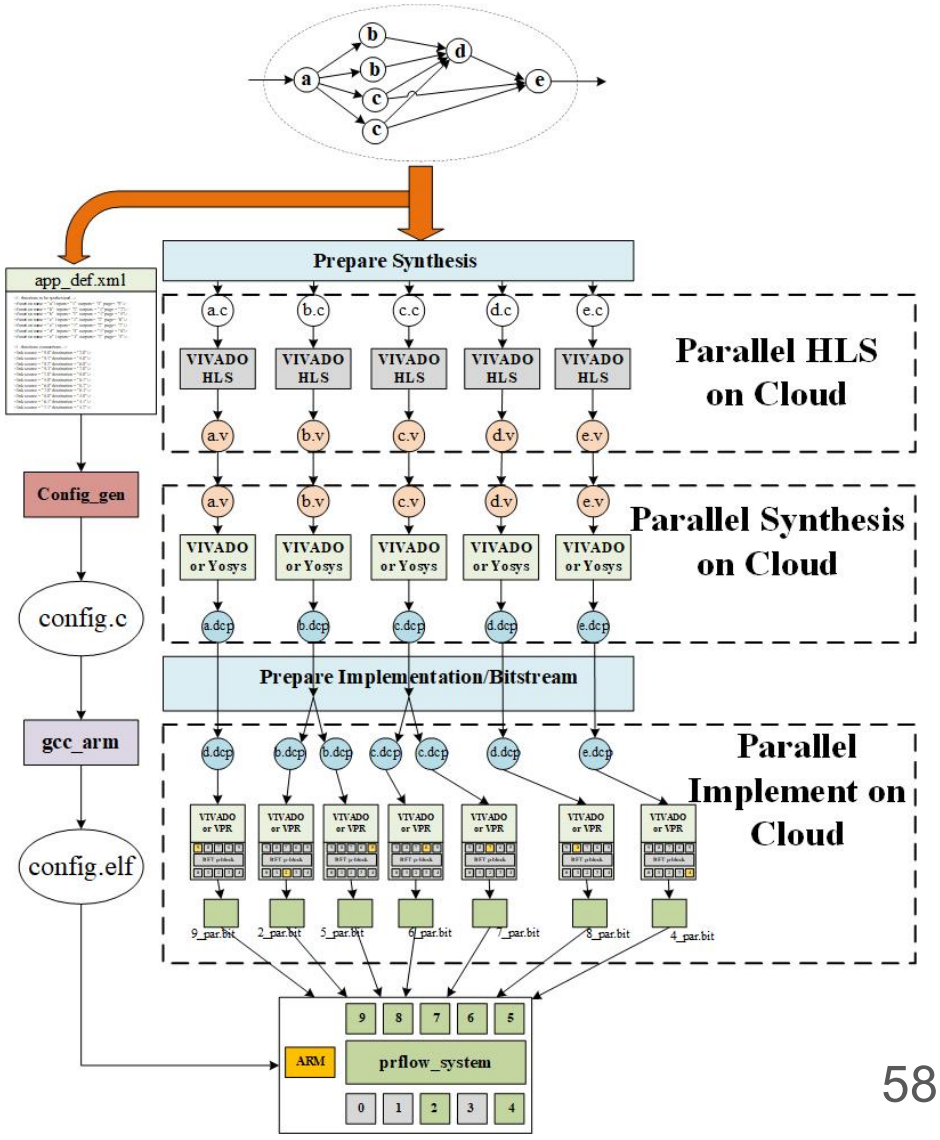


Resource Distribution

Type	LUT	FF	RAM18	DSP	# of Leaf
1	5760	11520	48	48	12
2	6720	13440	48	48	5
3	4800	9600	48	48	4
4	4800	9600	24	72	4
5	5760	11520	24	72	4
6	5960	11920	48	48	1
7	9120	18240	72	48	1
8	4320	8640	24	48	1
Total	172K	344K	1296	1584	30

Implementation:

- Use **Python** to generate the TCL scripts
- Use **qsub** to submit compilation tasks into icgrid
- git clone <yourID>@iclogin.seas.upenn.edu:/project/ese/ic/gitroot/prflow.git



Resource Utilizations

- Resource Overhead for the Overlay
  - 63% Logic Resources
- Leaf interface resource consumption equatio
  - Leaf Interface = $206+66I+227O$
  - Leaf Int. 36K BRAMs = $1+2I+O/2$
- Frequency and DDR bandwidth
  - 300MHz for the BFT
  - 200MHz for the AXI Bus
  - 200MHz for the leaf_logic

What can Symbiflow offer us?

- Avoid loading full chip database
- Avoid Mapping time for Fix logic
- Customize Quality vs. Runtime
- **Fixed time can go away!**

$$T_{new} = \frac{T_{origin} - T_{fix}}{\#\ of\ Partition} + T_{fix}$$

$$T_{new} = \frac{T_{origin}}{\#\ of\ Partition}$$