PyMTL Tutorial

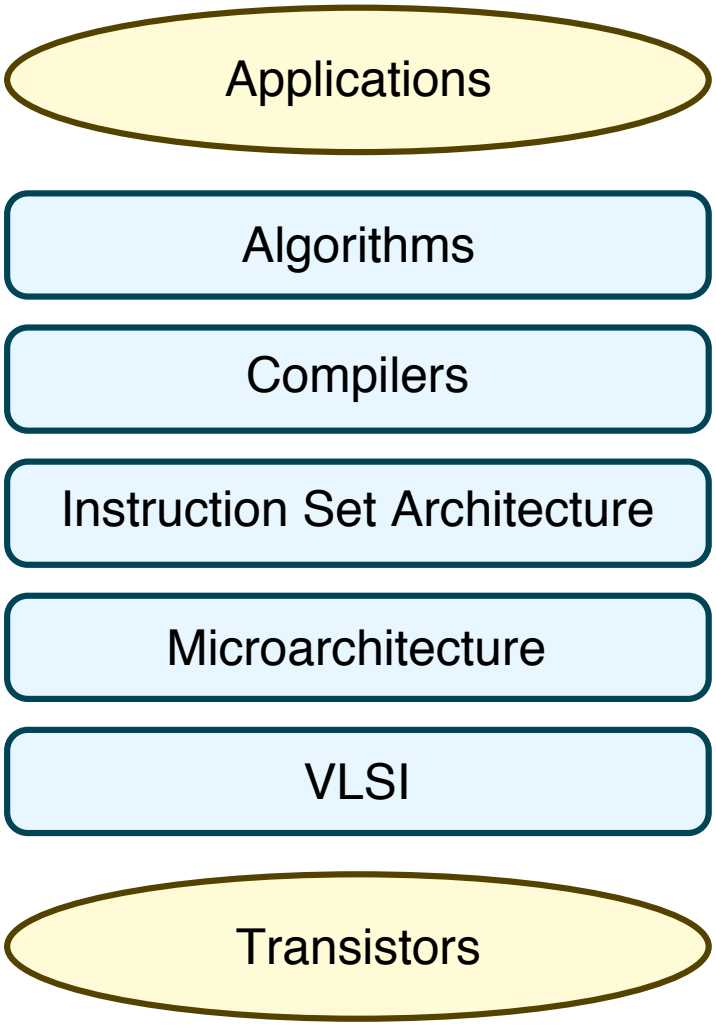
A Next-Generation Python-Based Framework for Hardware Generation, Simulation, and Verification

https://www.csl.cornell.edu/pymtl2019
https://github.com/cornell-brg/pymtl-tutorial-isca2019

Batten Research Group

Electrical and Computer Engineering
Cornell University
ISCA 2019

Multi-Level Modeling Methodologies



Functional-Level Modeling

– Behavior

Cycle-Level Modeling

– Behavior
– Cycle-Approximate
– Analytical Area, Energy, Timing

Register-Transfer-Level Modeling

– Behavior
– Cycle-Accurate Timing
– Gate-Level Area, Energy, Timing

Multi-Level Modeling Methodologies

Multi-Level Modeling Challenge

FL, CL, RTL modeling use very different languages, patterns, tools, and methodologies

SystemC is a good example of a unified multi-level modeling framework

Is SystemC the best we can do in terms of **productive** multi-level modeling?

Functional-Level Modeling

- Algorithm/ISA Development
- MATLAB/Python, C++ ISA Sim

Cycle-Level Modeling

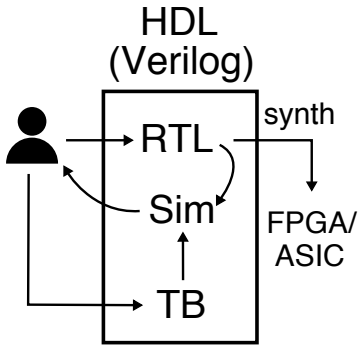
- Design-Space Exploration
- C++ Simulation Framework
- SW-Focused Object-Oriented
- gem5, SESC, McPAT

Register-Transfer-Level Modeling

- Prototyping & AET Validation
- Verilog, VHDL Languages
- HW-Focused Concurrent Structural
- EDA Toolflow

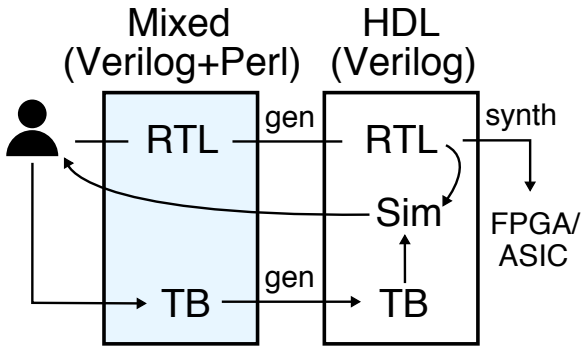
Traditional VLSI Design Methodologies

HDL
Hardware Description Language



- ✓ Fast edit-sim-debug loop
- ✓ Single language for structural, behavioral, + TB
- ✗ Difficult to create highly parameterized generators

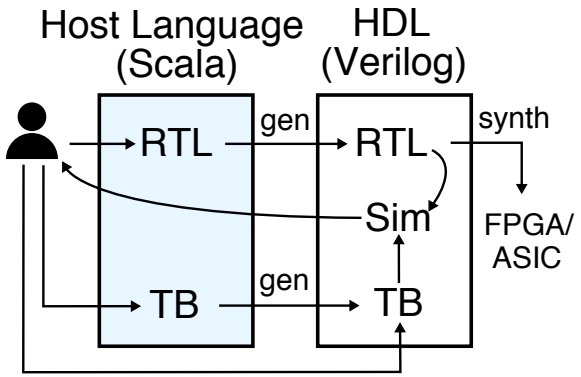
HPF
Hardware Preprocessing Framework



Example: Genesis2

- ✗ Slower edit-sim-debug loop
- ✗ Multiple languages create "semantic gap"
- ✓ Easier to create highly parameterized generators

HGF
Hardware Generation Framework

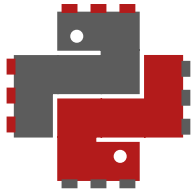


Example: Chisel

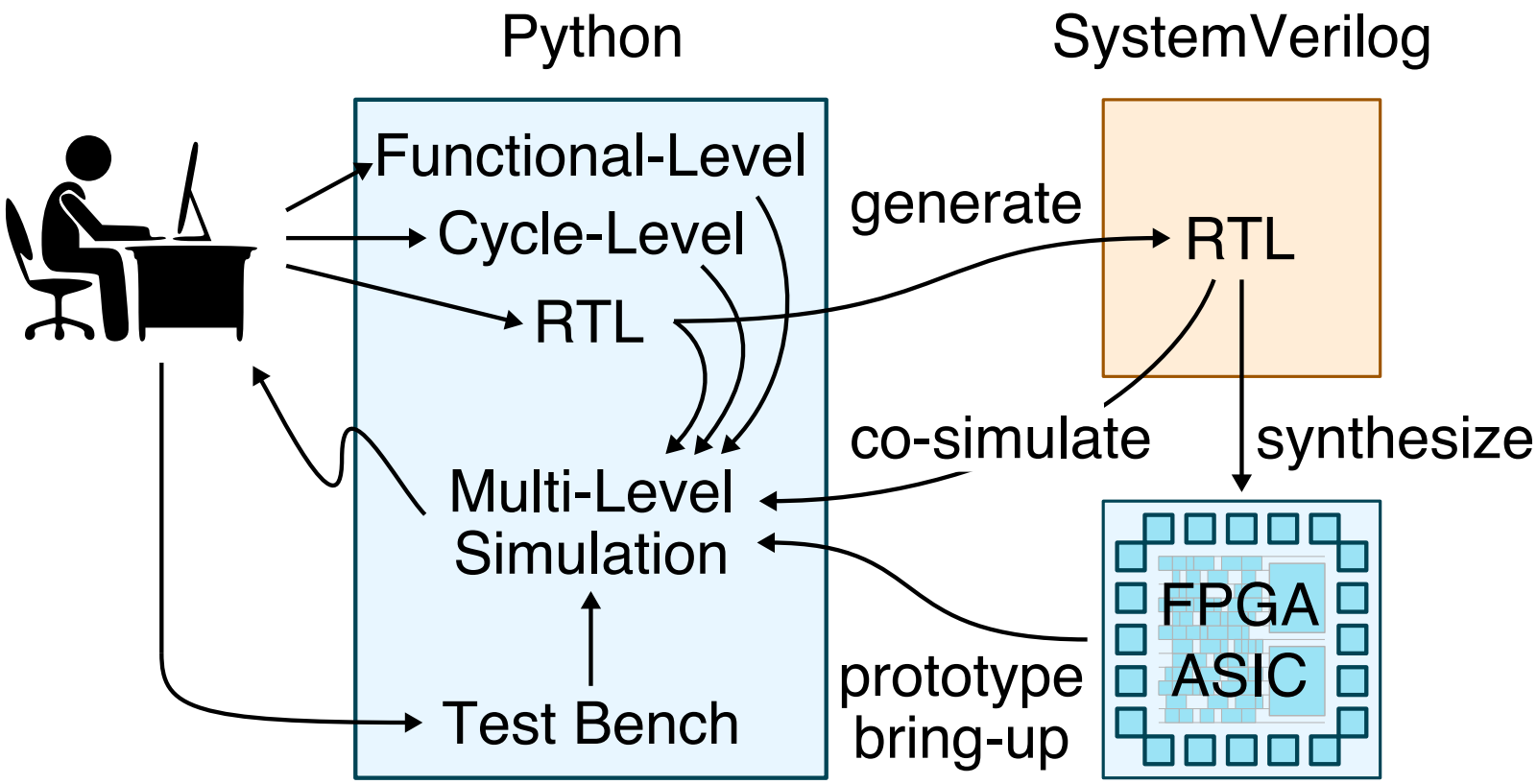
- ✗ Slower edit-sim-debug loop
- ✓ Single language for structural + behavioral
- ✓ Easier to create highly parameterized generators
- ✗ Cannot use power of host language for verification

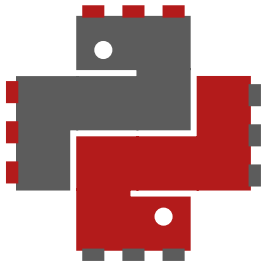
Is Chisel the best we can do in terms of a **productive** VLSI design methodology?

PyMTL



Python-based hardware generation, simulation, and verification framework which enables productive multi-level modeling and VLSI design

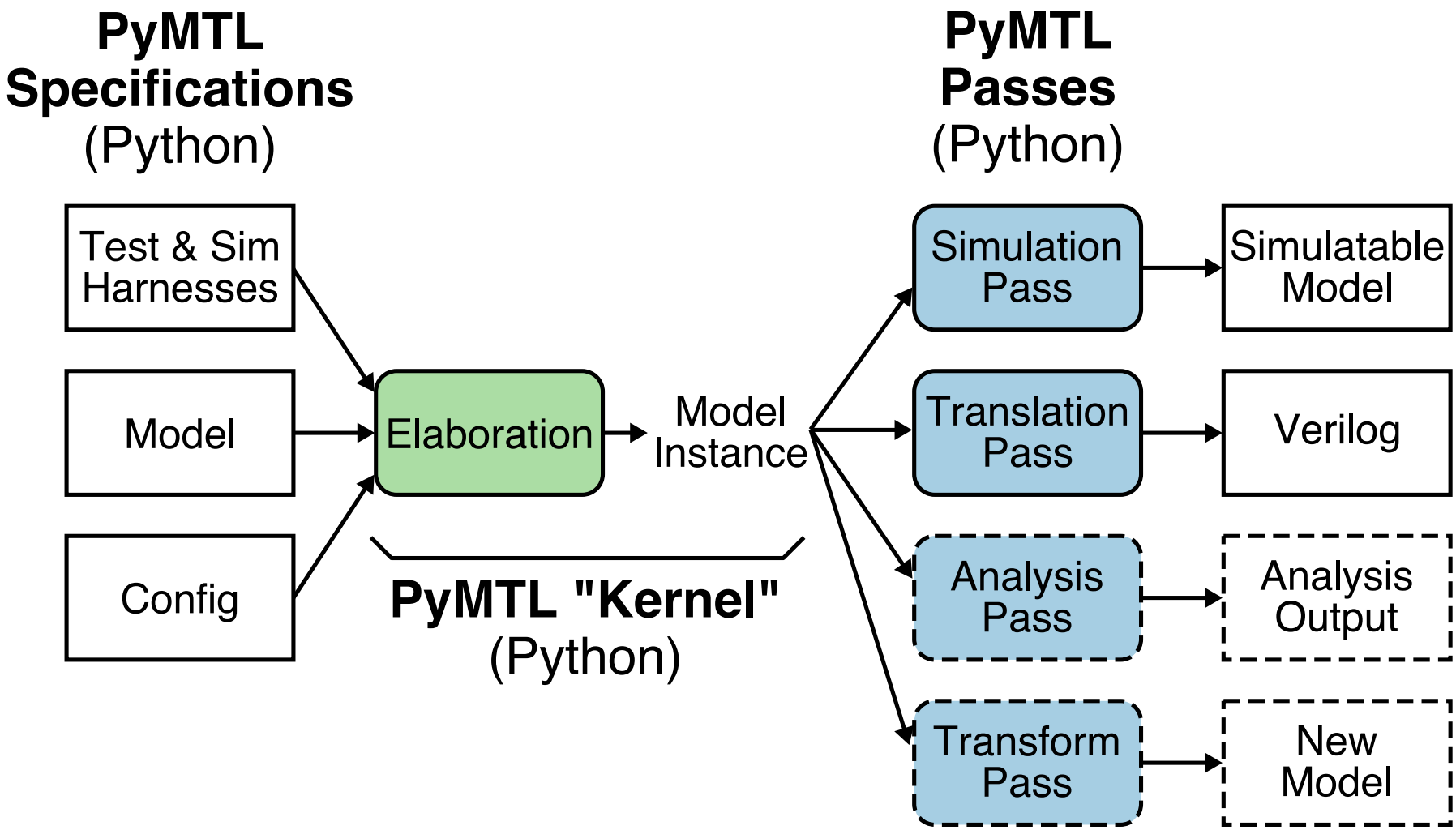




PyMTL

- **PyMTLv2**: `https://github.com/cornell-brg/pymtl`
  - released in 2014
  - extensive experience using framework in research & teaching
- **PyMTLv3**: `https://github.com/cornell-brg/pymtl3`
  - codenamed Mamba
  - beta-release today, official release later this summer
  - significant rewrite to improve productivity & performance

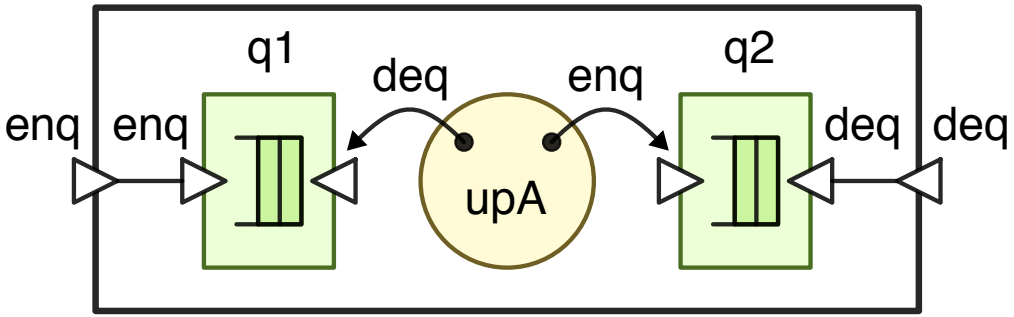
The PyMTL Framework



PyMTLv3 High-Level Modeling

```
1  class QueueFL( Component ):
2   def construct( s, maxsize ):
3     s.q = deque( maxlen=maxsize )
4
5   @non_blocking(
6     lambda s: len(s.q) < s.q.maxlen )
7   def enq( s, value ):
8     s.q.appendleft( value )
9
10  @non_blocking(
11    lambda s: len(s.q) > 0 )
12  def deq( s ):
13    return s.q.pop()
```

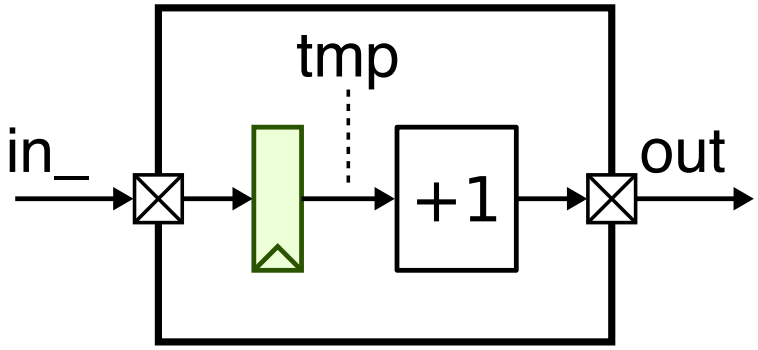
- FL/CL components can use method-based interfaces
- Structural composition via connecting methods




```
14 class DoubleQueueFL( Component ):
15  def construct( s ):
16    s.enq = NonBlockingCalleeIfc()
17    s.deq = NonBlockingCalleeIfc()
18
19    s.q1 = QueueFL(2)
20    s.q2 = QueueFL(2)
21
22    s.connect( s.enq,    s.q1.enq )
23    s.connect( s.q2.deq, s.deq    )
24
25    @s.update
26    def upA():
27      if s.q1.deq.rdy() and s.q2.enq.rdy():
28        s.q2.enq( s.q1.deq() )
```

PyMTLv3 Low-Level Modeling

```
1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5    def construct( s, dtype ):
6      s.in_  = InPort ( dtype )
7      s.out  = OutPort( dtype )
8      s.tmp  = Wire   ( dtype )
9
10     @s.update_on_edge
11     def seq_logic():
12       s.tmp = s.in_
13
14     @s.update
15     def comb_logic():
16       s.out = s.tmp + 1
```



- RTL components can be translated into *readable* Verilog
- This translated Verilog can then be automatically imported back into PyMTL for co-simulation via Verilator
- External Verilog IP can also be co-simulated via Verilator

What is PyMTL for and not (currently) for?

- **PyMTL is for ...**
  - Taking an accelerator design from concept to implementation
  - Construction of highly-parameterizable CL models
  - Construction of highly-parameterizable RTL design generators
  - Rapid design, testing, and exploration of hardware mechanisms
  - Interfacing models with other C++ or Verilog frameworks
- **PyMTL is not (currently) for ...**
  - Python high-level synthesis
  - Many-core simulations with hundreds of cores
  - Full-system simulation with real OS support
  - Users needing a complex OOO processor model “out of the box”
  - Users needing a mature modeling framework that will not change (consider using PyMTLv2!)

Let’s see some examples of how PyMTLv2 has been used in practice ...

PyMTL in Architecture and EDA Research

MICRO'18 T. Chen, S. Srinath, C. Batten, E. Suh. **"An Architectural Framework for Accelerating Dynamic Parallel Algorithms on Reconfigurable Hardware."**

MICRO'17 J. Kim, S. Jiang, C. Torng, M. Wang, S. Srinath, B. Ilbeyi, K. Al-Hawaj, C. Batten. **"Using Intra-Core Loop-Task Accelerators to Improve the Productivity and Performance of Task-Based Parallel Programs."**

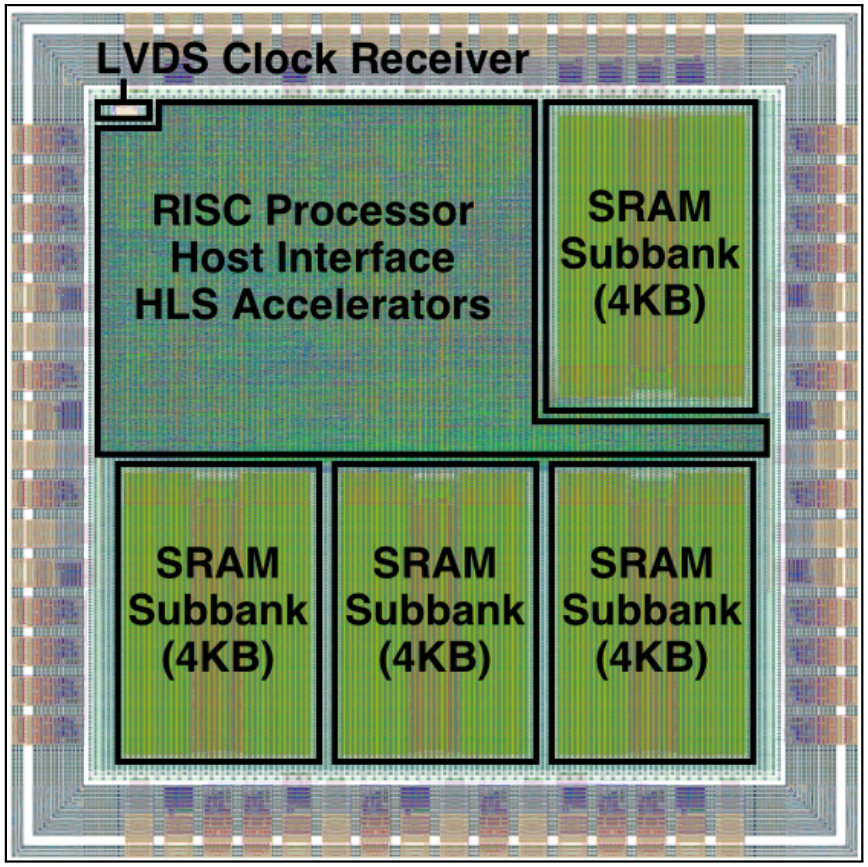
FPGA'17 H. Dai, R. Zhao, G. Liu, S. Srinath, U. Gupta, C. Batten, Z. Zhang. **"Dynamic Hazard Resolution for Pipelining Irregular Loops in High-Level Synthesis."**

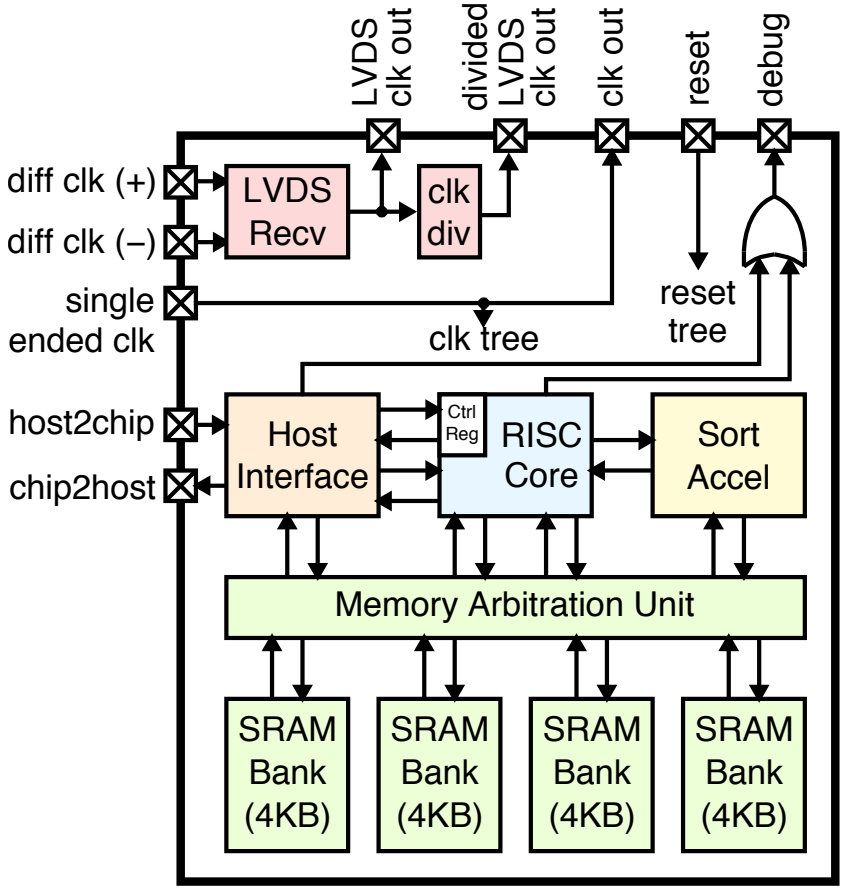
MICRO'16 T. Chen and E. Suh. **"Efficient Data Supply for Hardware Accelerators with Prefetching and Access/Execute Decoupling."**

DAC'16 R. Zhao, G. Liu, S. Srinath, C. Batten, Z. Zhang. **"Improving High-Level Synthesis with Decoupled Data Structure Optimization."**

MICRO'14 S. Srinath, B. Ilbeyi, M. Tan, G. Liu, Z. Zhang, C. Batten. **"Architectural Specialization for Inter-Iteration Loop Dependence Patterns."**

PyMTLv2 ASIC Tapeout #1 (2016)



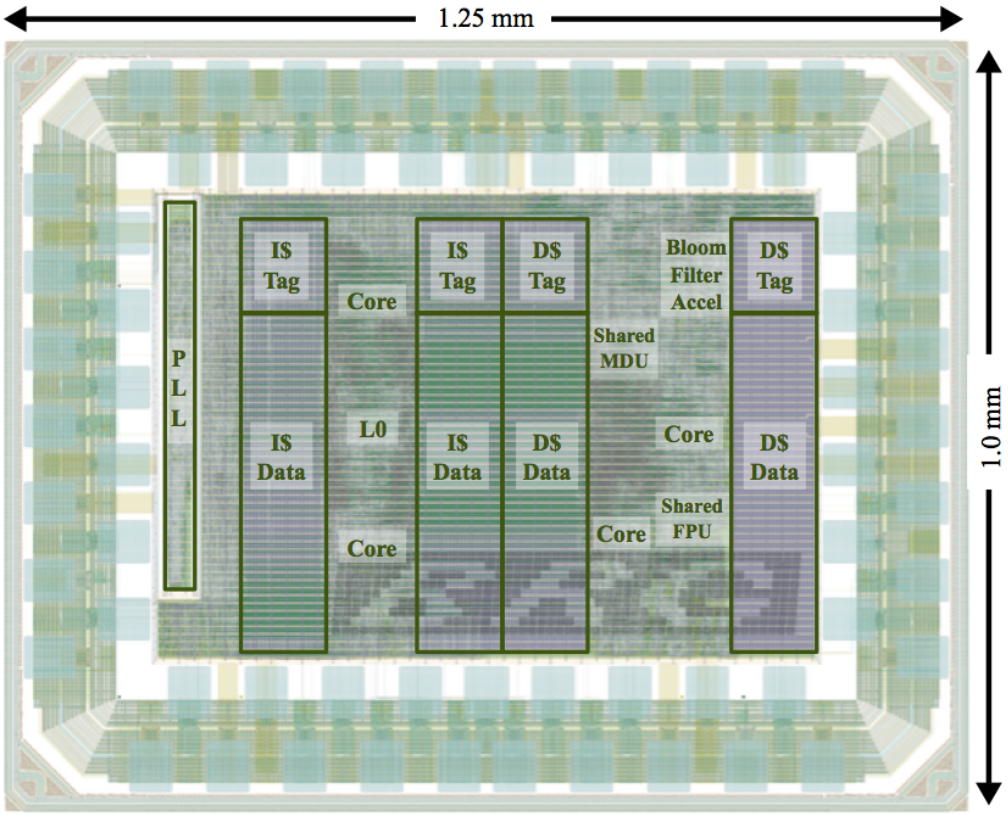


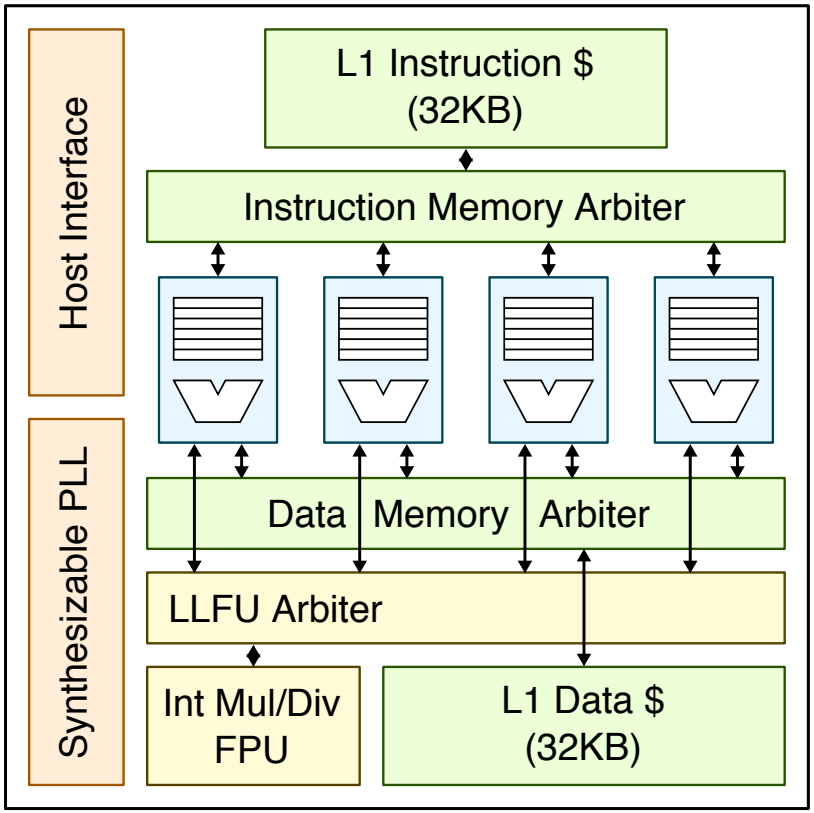
RISC processor, 16KB SRAM, HLS-generated accelerator

2x2mm, 1.2M-trans, IBM 130nm

95% done using PyMTLv2

PyMTLv2 ASIC Tapeout #2 (2018)

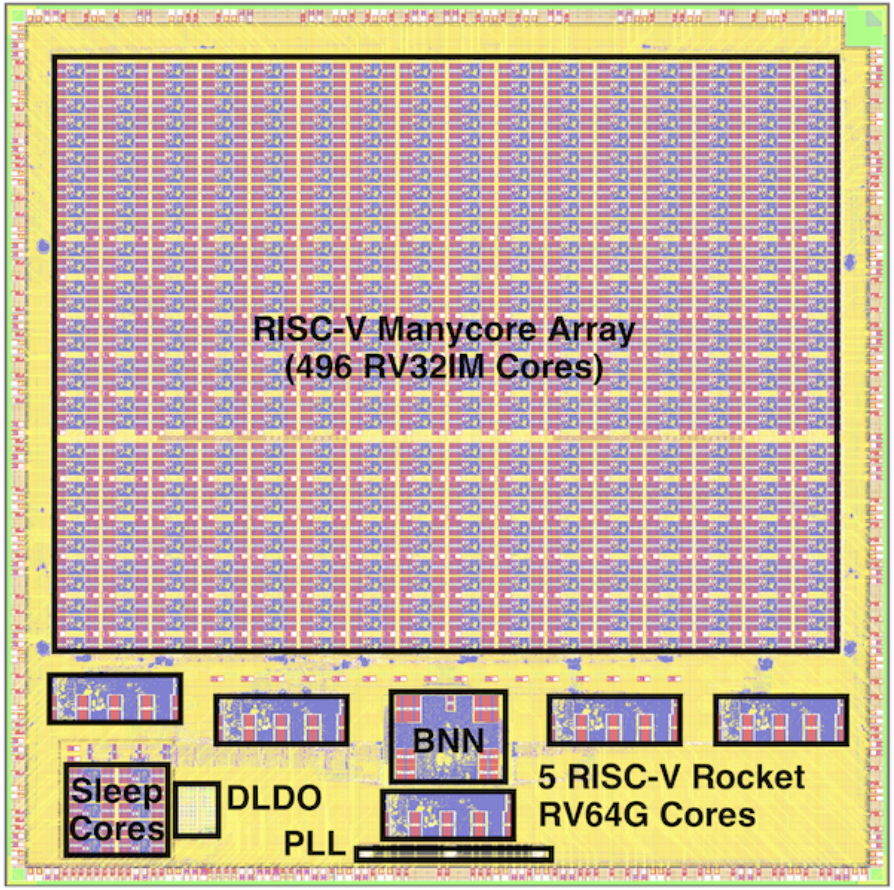




Four RISC-V RV32IMAF cores with “smart” sharing of L1$/LLFU
1x1.2mm, 6.7M-trans, TSMC 28nm
95% done using PyMTLv2

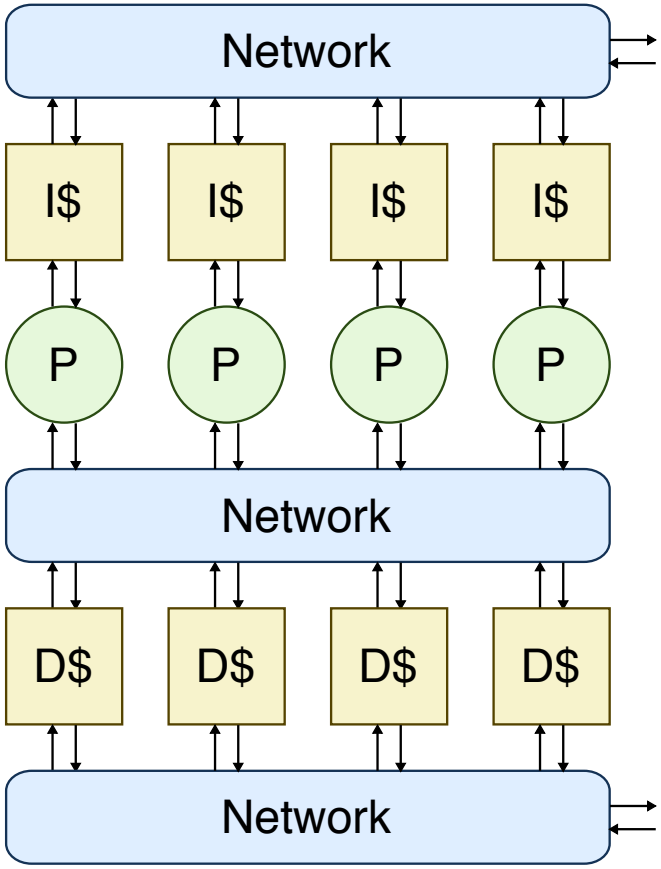
Celerity SoC through DARPA CRAFT Program

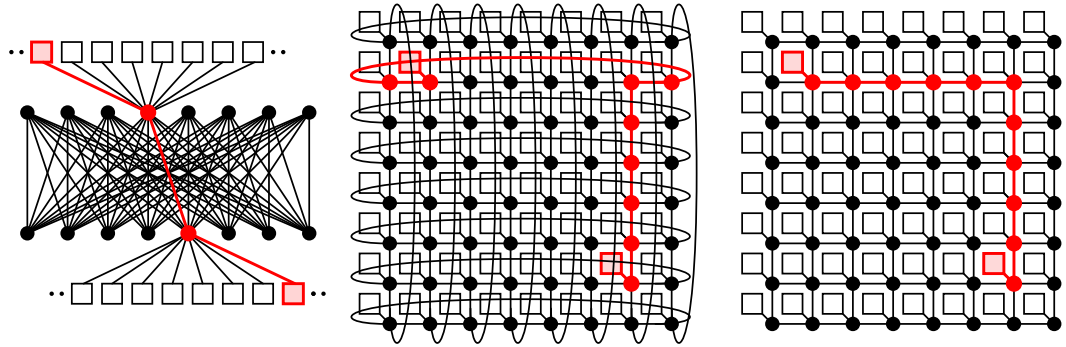
- 5 × 5mm in TSMC 16 nm FFC
- 385 million transistors
- 511 RISC-V cores
  - 5 Linux-capable Rocket cores
  - 496-core tiled manycore
  - 10-core low-voltage array
- 1 BNN accelerator
- 1 synthesizable PLL
- 1 synthesizable LDO Vreg
- 3 clock domains
- 672-pin flip chip BGA package



PyMTLv2 played a small but important role in testing the BNN and automatically generating appropriate wrappers to interface with the Rocket core via RoCC

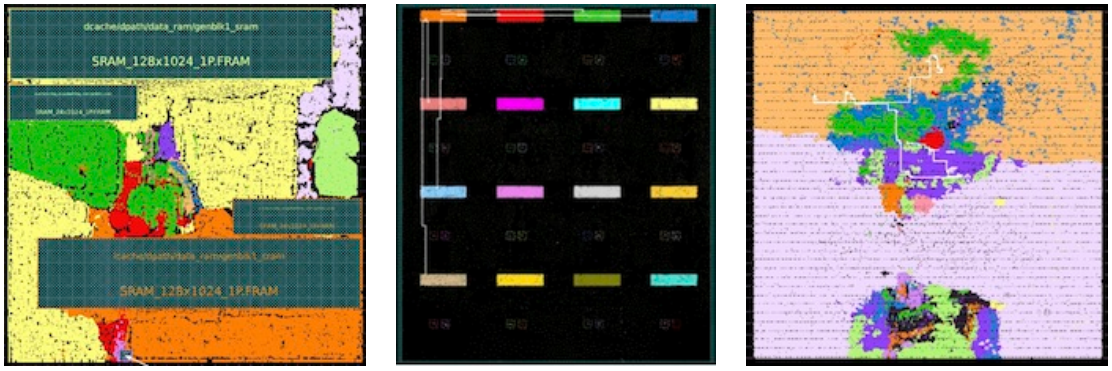
PyMTLv2 in Teaching and POSH





DARPA POSH Open-Source Hardware Program

PyMTL used as a powerful open-source generator for both design and verification



Undergraduate Comp Arch Course

Labs use PyMTL for verification, PyMTL or Verilog for RTL design

Graduate ASIC Design Course

Labs use PyMTL for verification, PyMTL or Verilog for RTL design, standard ASIC flow

PyMTL Project Sponsors and Acknowledgments

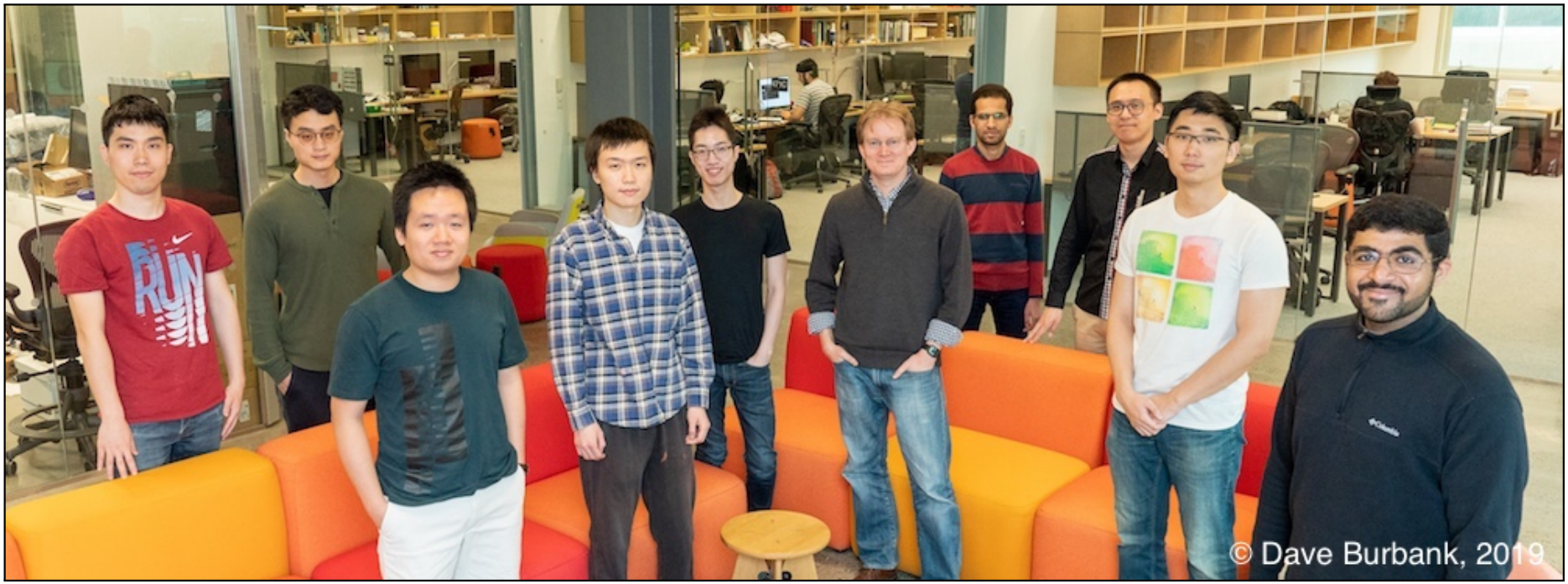
Funding partially provided by the National Science Foundation through NSF CRI Award #1512937 and NSF SHF Award #1527065.

Funding partially provided by the Defense Advanced Research Projects Agency through a DARPA POSH Award #FA8650-18-2-7852.

Funding partially provided by the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA.

Thanks to **Derek Lockhart**, Ji Kim, Shreesha Srinath, Berkin Ilbeyi, Yixiao Zhang, Jacob Glueck, Aaron Wisner, Gary Zibrat, and Carl Friedrich Bolz for their help developing, testing, and using PyMTLv2 and PyMTLv3

PyMTL Tutorial Organizers



- **Shunning Jiang** : Lead researcher and developer for PyMTLv3
- **Peitian Pan** : Leading work on translation & gradually-typed HDL
- **Yanghui Ou** : Leading work on property-based random testing
- **Chris Torng** : Leading work on integration with EDA toolflow
- Cheng Tan, Tuan Ta, Moyang Wang, Khalid Al-Hawaj, Shady Agwal

PyMTL Tutorial Schedule

1:45 – 2:00pm	Virtual Machine Installation and Setup
2:00 – 2:15pm	*Presentation:* PyMTL Overview
2:15 – 2:55pm	*Part 1:* PyMTL Basics
2:55 – 3:30pm	*Part 2:* Multi-Level Modeling with PyMTL
3:30 – 4:00pm	Coffee Break
4:00 – 4:45pm	*Part 3:* Processor Modeling with PyMTL
4:45 – 5:30pm	*Part 4:* Multi-Level Composition in PyMTL