PyMTL3

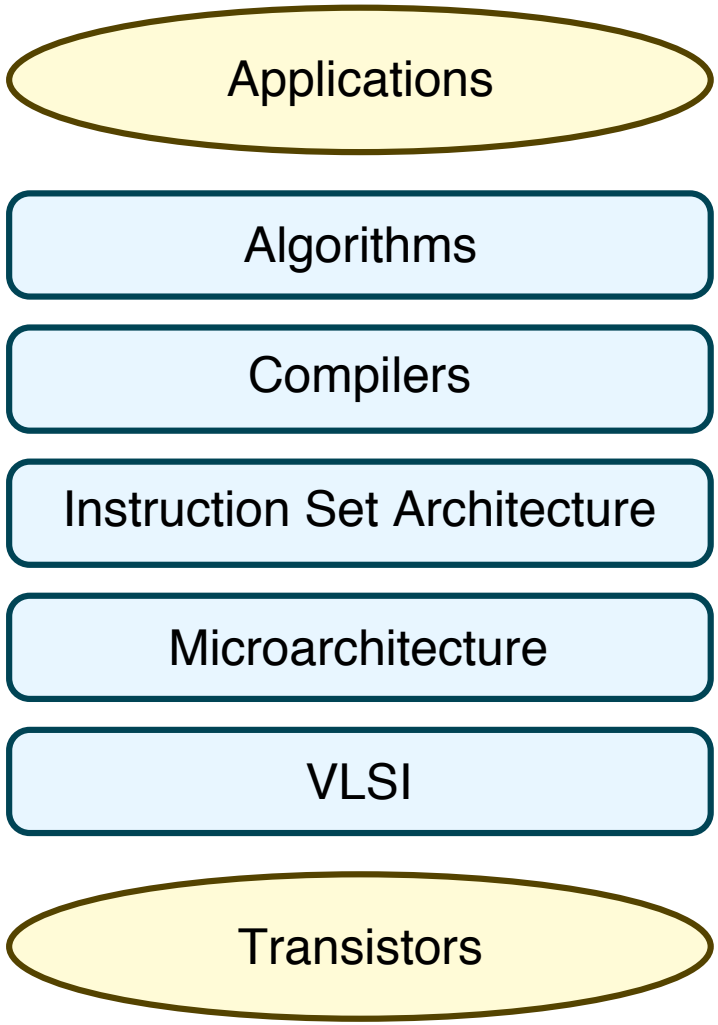
A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification

`https://pymtl.github.io`
`https://github.com/pymtl/pymtl3-chipkit-isca2020`

Christopher Batten

Electrical and Computer Engineering
Cornell University
CHIPKIT Tutorial @ ISCA'20

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

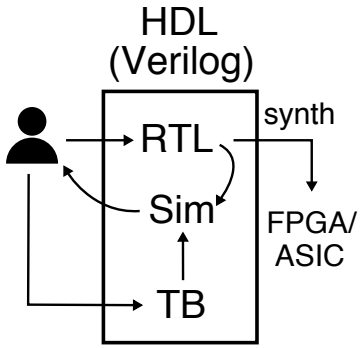
- Design-Space Exploration
- C++ Simulation Framework
- SW-Focused Object-Oriented
- gem5, SESC, McPAT

Register-Transfer-Level Modeling

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

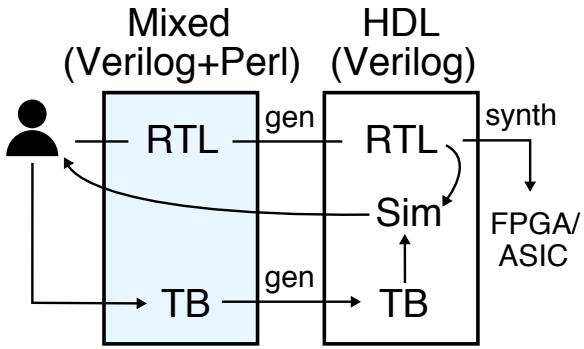
Traditional VLSI Design Methodologies

HDL
Hardware Description Language



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

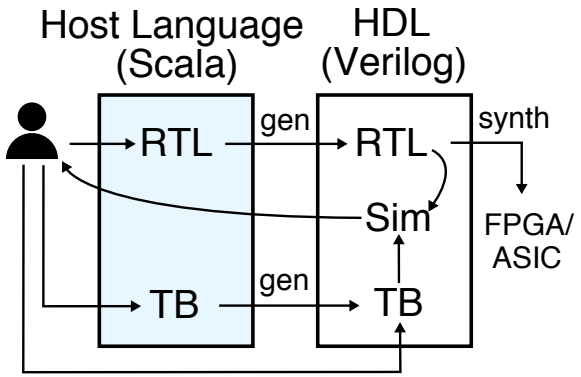
HPF
Hardware Preprocessing Framework



Example: Genesis2

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

HGF
Hardware Generation Framework

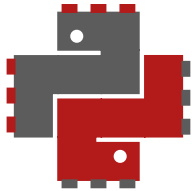


Example: Chisel

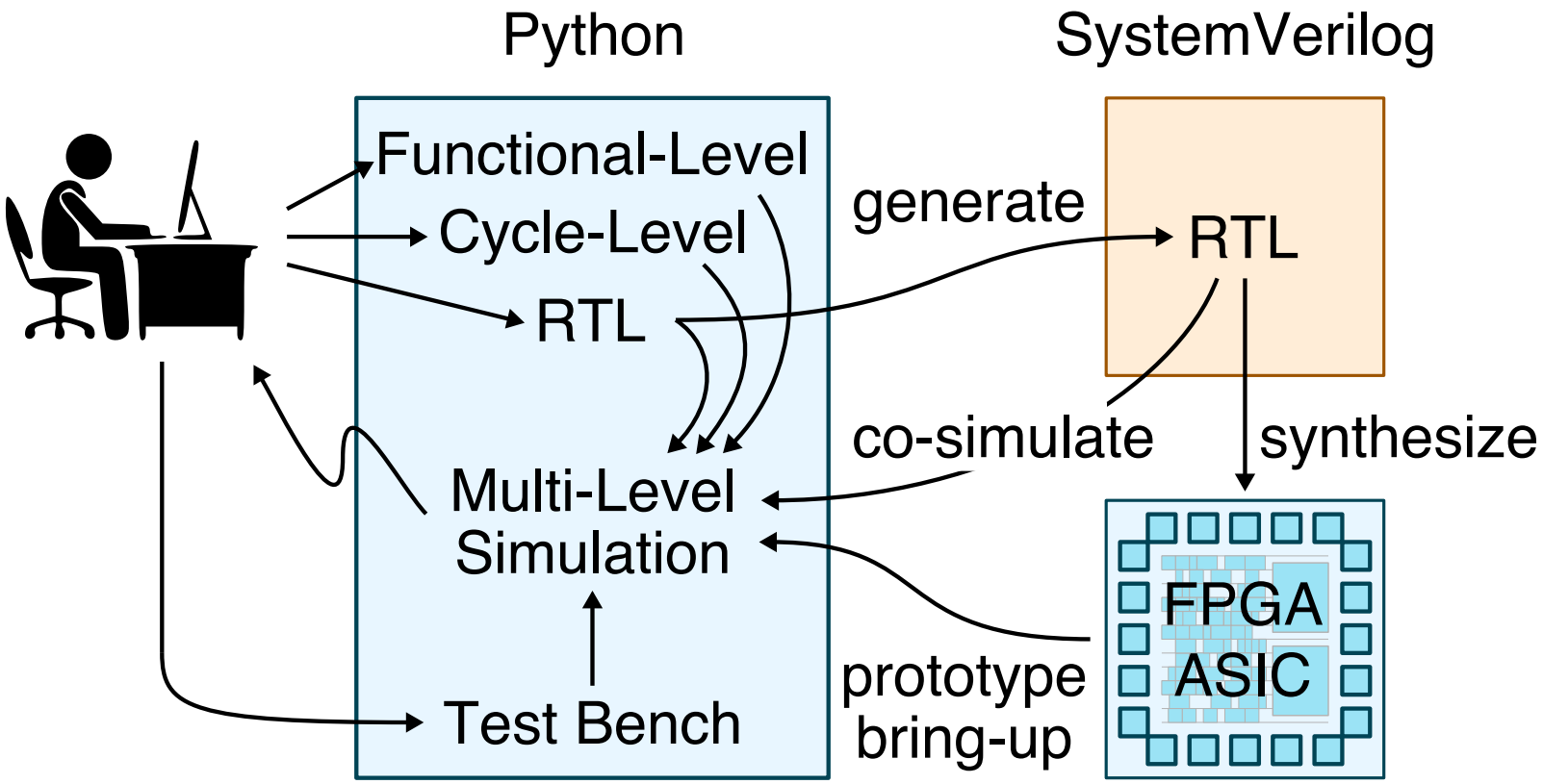
- ✗ Slower edit-sim-debug loop
- ✓ Single language for structural + behavioral
- ✓ Easier to create highly parameterized generators
- ✗ Cannot use power of host language for verification

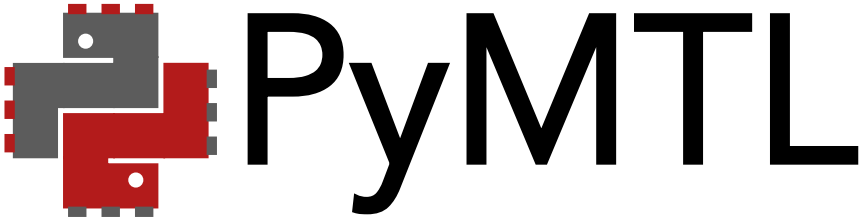
Is Chisel the best we can do in terms of a **productive** VLSI design methodology?

PyMTL



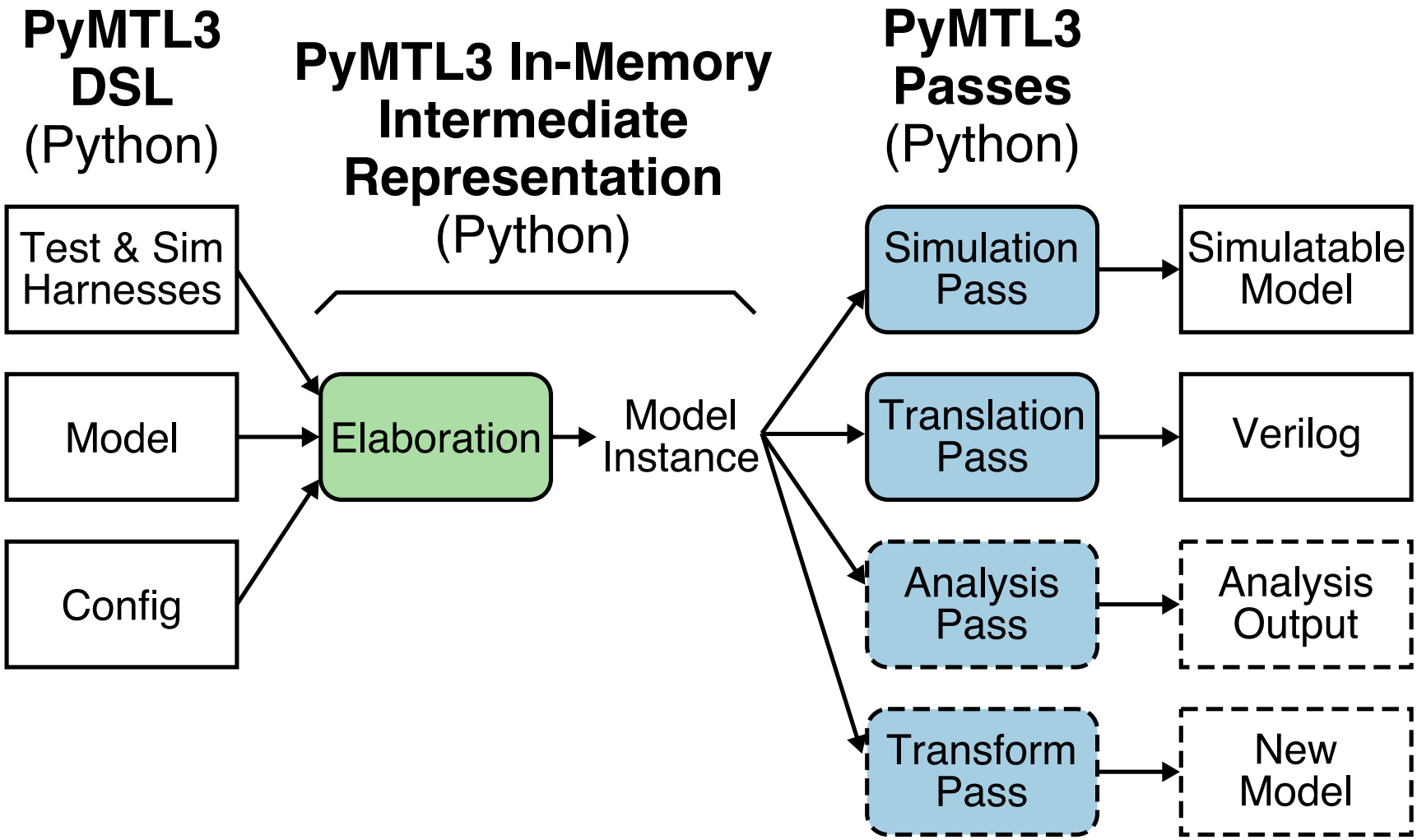
Python-based hardware generation, simulation, and verification framework which enables productive multi-level modeling and VLSI design





- **PyMTL2**: `https://github.com/cornell-brg/pymtl`
  - released in 2014
  - extensive experience using framework in research & teaching
- **PyMTL3**: `https://github.com/pymtl/pymtl3`
  - official release earlier this month
  - adoption of new Python3 features
  - significant rewrite to improve productivity & performance
  - cleaner syntax for FL, CL, and RTL modeling
  - completely new Verilog translation support
  - first-class support for method-based interfaces

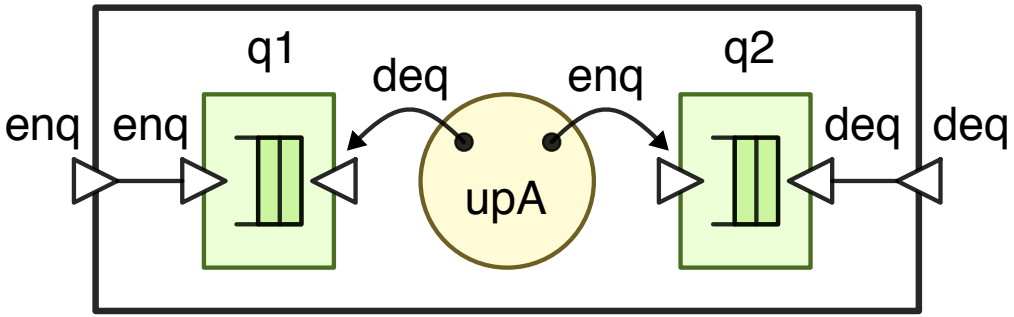
The PyMTL3 Framework



PyMTL3 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()
```

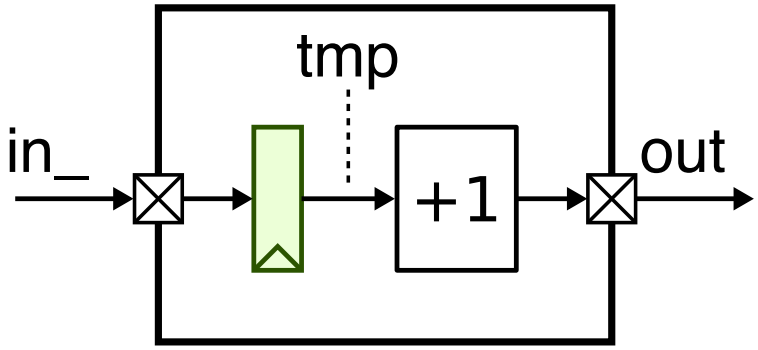
- FL/CL components can use method-based interfaces
- Structural composition via connecting methods




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

PyMTL3 Low-Level Modeling

```
1  from pymtl3 import *
2
3  class RegIncrRTL( Component ):
4
5    def construct( s, nbits ):
6      s.in_ = InPort ( nbits )
7      s.out = OutPort( nbits )
8      s.tmp = Wire   ( nbits )
9
10     @update_ff
11     def seq_logic():
12       s.tmp <<= s.in_
13
14     @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 PyMTL3 for and not (currently) for?

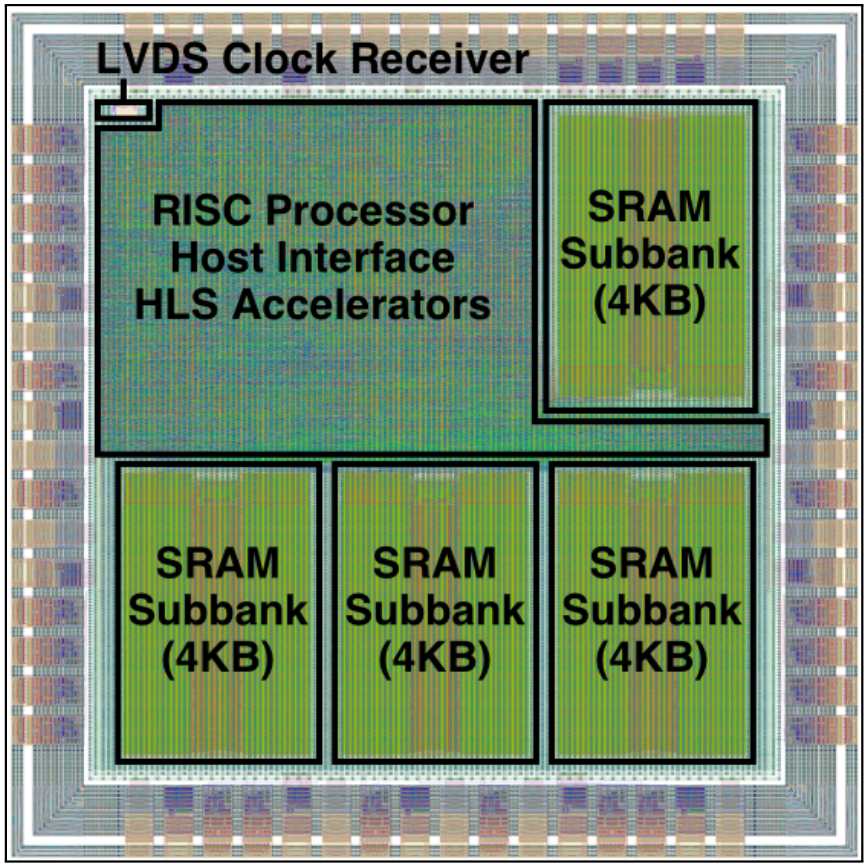
- **PyMTL3 is for ...**
  - Taking an accelerator design from concept to implementation
  - Construction of highly-parameterizable CL models
  - Construction of highly-parameterizable RTL design generators
  - Rapid design, testing, and exploration of hardware mechanisms
  - Interfacing models with other C++ or Verilog frameworks
- **PyMTL3 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”

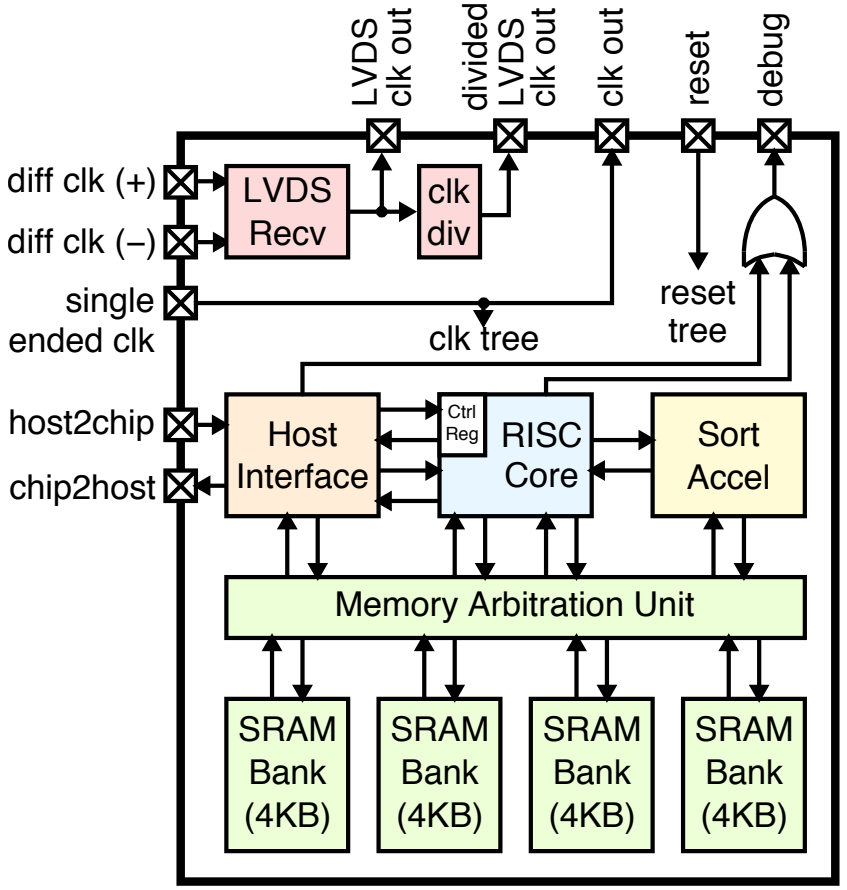
Let’s see some examples of how PyMTL2 has been used in practice ...

PyMTL2 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."**

PyMTL2 ASIC Tapeout #1 (2016)



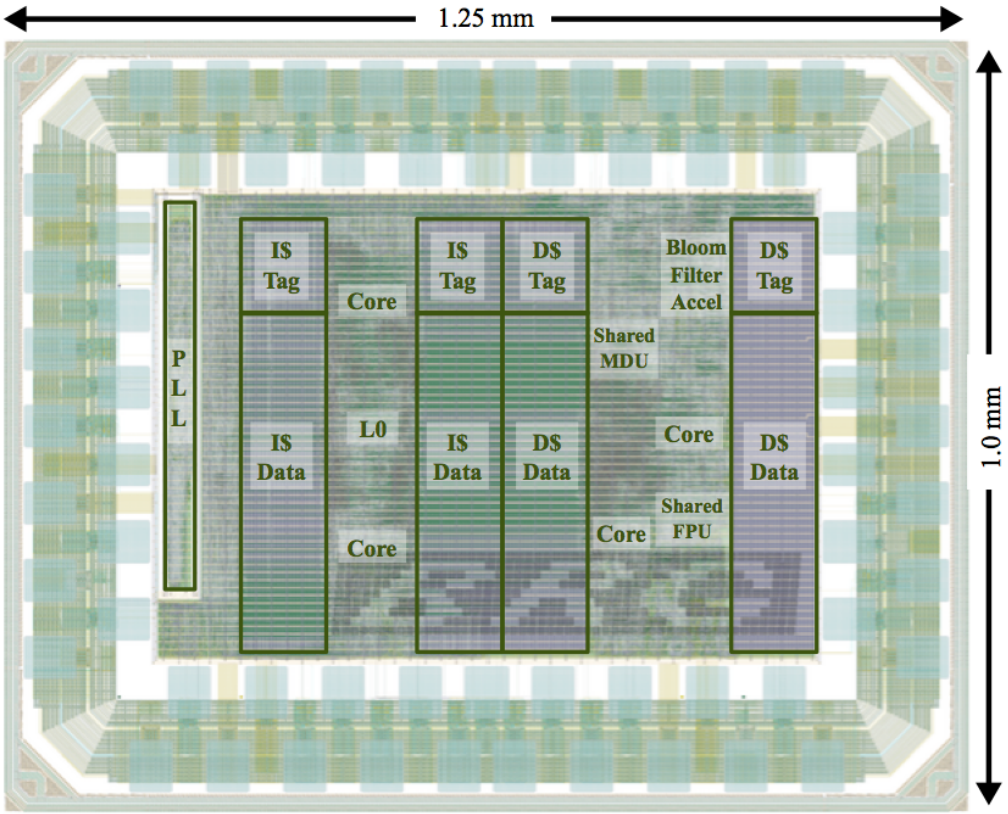


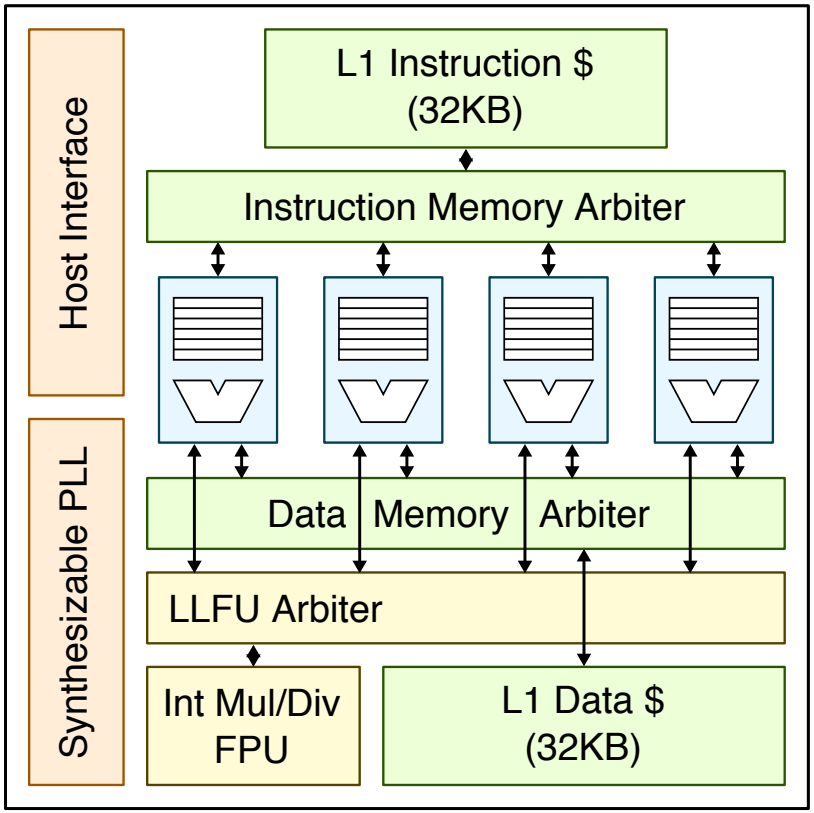
RISC processor, 16KB SRAM, HLS-generated accelerator

2x2mm, 1.2M-trans, IBM 130nm

95% done using PyMTL2

PyMTL2 ASIC Tapeout #2 (2018)

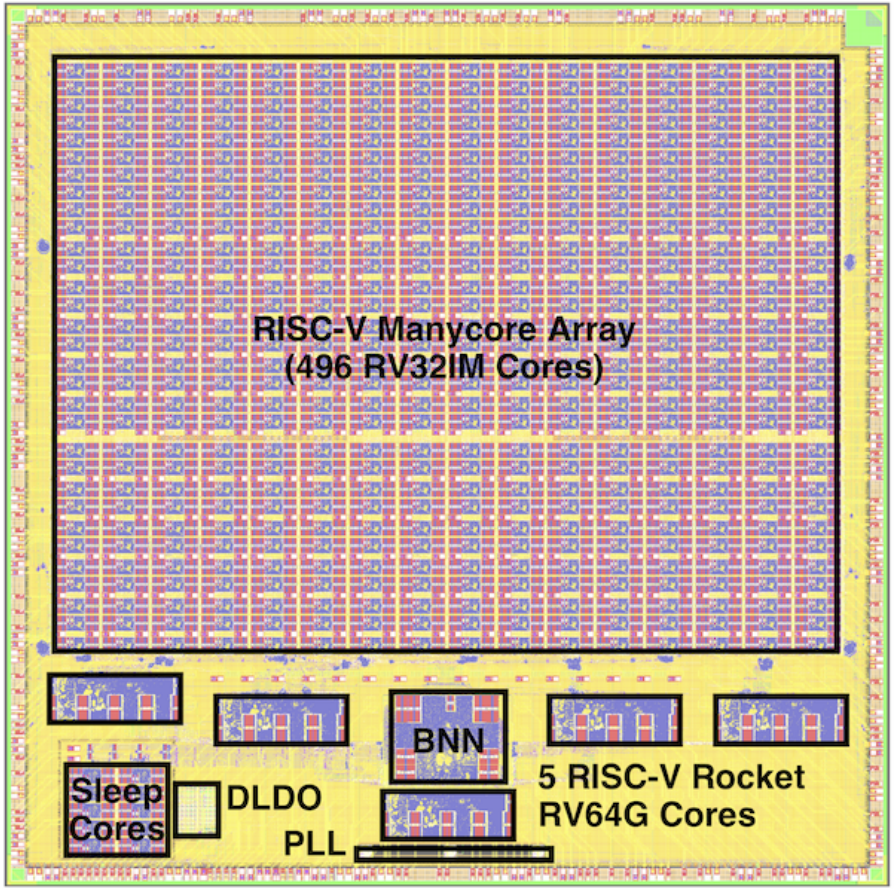




Four RISC-V RV32IMAF cores with “smart” sharing of L1$/LLFU
1x1.2mm, 6.7M-trans, TSMC 28nm
95% done using PyMTL2

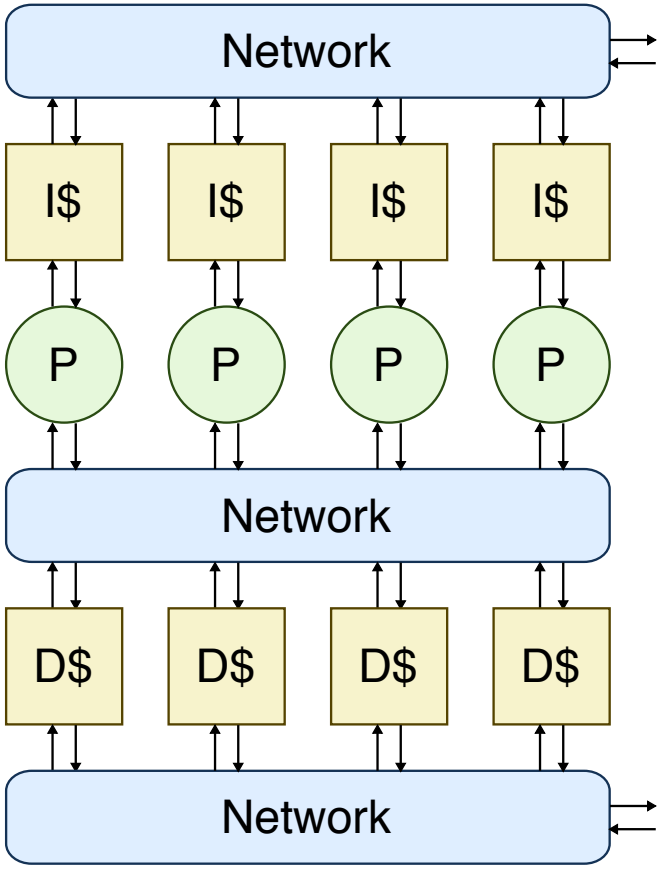
Celerity SoC through DARPA CRAFT Program

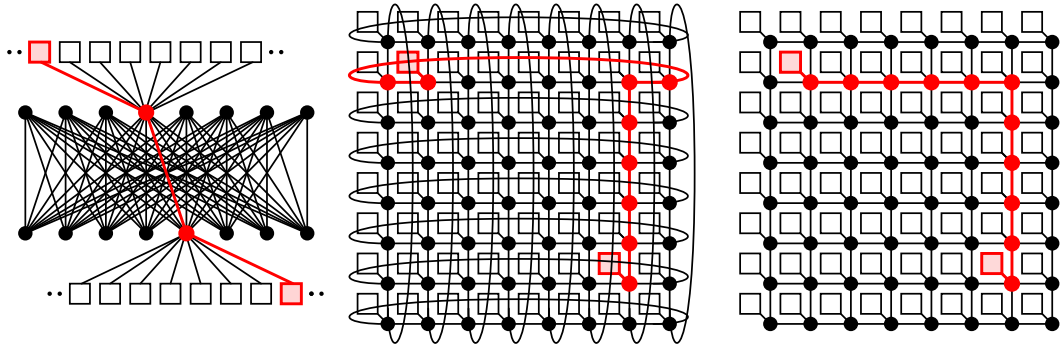
- 5 × 5mm in TSMC 16 nm FFC
- 385 million transistors
- 511 RISC-V cores
  - 5 Linux-capable Rocket cores
  - 496-core tiled manycore
  - 10-core low-voltage array
- 1 BNN accelerator
- 1 synthesizable PLL
- 1 synthesizable LDO Vreg
- 3 clock domains
- 672-pin flip chip BGA package



PyMTL2 played a small but important role in testing the BNN and automatically generating appropriate wrappers to interface with the Rocket core via RoCC

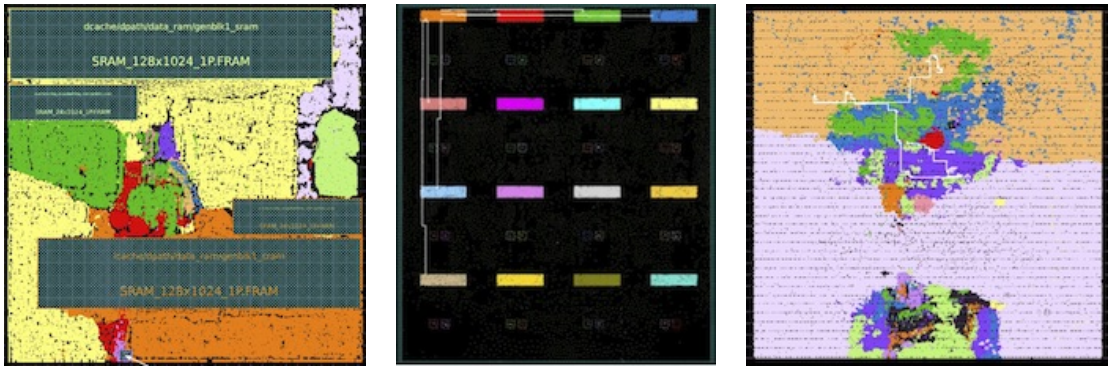
PyMTL2 in Teaching and POSH





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

Two Upcoming PyMTL3 Publications

IEEE Micro

PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification

Shunning Jiang, Peitian Pan, Yanghui Ou, and Christopher Batten
Cornell University

***Abstract*—We present PyMTL3, a Python framework for open-source hardware modeling, generation, simulation, and verification. In addition to the compelling benefits from using the Python language, PyMTL3 is designed to provide productive, flexible, and extensible workflows for both hardware designers and computer architects. PyMTL3 supports a seamless multi-level modeling environment and carefully designed modular software architecture using a sophisticated in-memory intermediate representation and a collection of passes that analyze, instrument, and transform PyMTL3 hardware models. PyMTL3 can play an important role in jump-starting the open-source hardware ecosystem.**

DUE TO THE BREAKDOWN of transistor scaling and the slowdown of Moore's law, there has been an increasing trend towards energy-efficient system-on-chip (SoC) design using heterogeneous architectures with a mix of general-purpose and specialized computing engines. Heterogeneous SoCs emphasize both flexible parameterization of a single design block and versatile composition of numerous different design blocks, which have imposed significant challenges to state-of-the-art hardware modeling and verification methodologies. Developing, open-sourcing, and collaborating on *hardware generators* is a compelling solution to increase the reuse of highly parametrized and thoroughly tested hardware blocks in the community. However, the general lack of high-quality open-source hardware designs and hardware verification methodologies have been a major concern that limits the widespread adoption of open-source hardware.

To respond to these challenges, the open-source hardware community is augmenting or even replacing traditional domain-specific hardware description languages (HDLs) with productive hardware development frameworks empowered by high-level general-purpose programming languages such as C++, Scala, Perl, and Python. *Hardware preprocessing frameworks* (e.g., Genesis2 [1]) intermingle a high-level language for macro-processing and a low-level HDL for

IEEE Design & Test

PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies

Shunning Jiang*, Yanghui Ou*, Peitian Pan, Kaishuo Cheng, Yixiao Zhang, and Christopher Batten
Cornell University

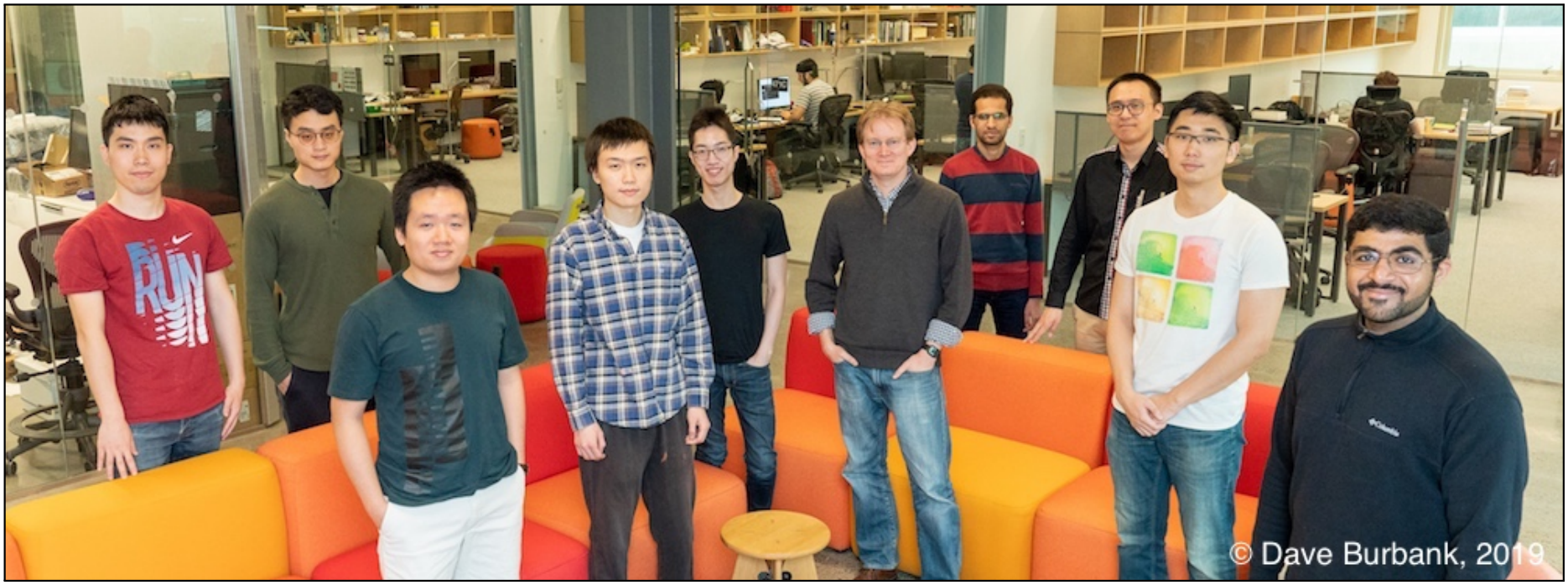
***Abstract*—The success of the emerging open-source hardware ecosystem critically depends on thoroughly tested open-source hardware blocks. Unfortunately, it is challenging to adopt traditional closed-source hardware testing approaches in the open-source hardware community. To tackle these challenges, we introduce PyH2, our vision for a productive and open-source testing methodology for open-source hardware. Leveraging PyMTL3, `pytest`, and `hypothesis`, PyH2 attempts to reduce the designers' effort in creating high-quality property-based random tests. This paper introduces and quantitatively evaluates the benefits of three PyH2 frameworks: PyH2G for design generators, PyH2P for processors, and PyH2O for testing hardware with object-oriented interfaces.**

AS Dennard scaling is over and Moore's law continues to slow down, modern system-on-chip (SoC) architectures have been moving towards heterogeneous compositions of general-purpose and specialized computing fabrics. This heterogeneity complicates the already challenging task of SoC design and verification. Building an open-source hardware community to amortize the non-recurring engineering effort of developing highly parametrized and thoroughly verified hardware blocks is a promising solution to the heterogeneity challenge. However, the widespread adoption of open-source hardware has been obstructed by the scarcity of such high quality blocks. We argue that a key missing piece in the open-source hardware ecosystem is comprehensive, productive, and open-source verification methodologies that reduce the effort required to create thoroughly tested hardware blocks. Compared to closed-source hardware, verification of open-source hardware faces several significant challenges.

First, closed-source hardware is usually

* Shunning Jiang and Yanghui Ou contributed equally to this work and are listed alphabetically.

PyMTL3 Developers



- **Shunning Jiang** : Lead researcher and developer for PyMTL3
- **Peitian Pan** : Leading work on translation & gradually-typed HDL
- **Yanghui Ou** : Leading work on property-based random testing
- Tuan Ta, Moyang Wang, Khalid Al-Hawaj, Shady Agwal, Lin Cheng

PyMTL Project Sponsors

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.

Funding partially provided by an unrestricted industry gift from the Xilinx University Program

This work was supported in part by NSF XPS Award #1337240, NSF CRI Award #1512937, NSF SHF Award #1527065, AFOSR YIP Award #FA9550-15-1-0194, DARPA Young Faculty Award #N66001-12-1-4239, a Xinux University Program industry gift, and the the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA, and equipment, tool, and/or physical IP donations from Intel, NVIDIA, Synopsys, and ARM.

Thanks to **Derek Lockhart**, Ji Kim, Shreesha Srinath, Berkin Ilbeyi, Yixiao Zhang, Jacob Glueck, Aaron Wisner, Gary Zibrat, Christopher Torng, Cheng Tan, Raymond Yang, Kaishuo Cheng, Jack Weber, Carl Friedrich Bolz, David MacIver, and Zac Hatfield-Dodds for their help designing, developing, testing, and using PyMTL2 and PyMTL3

The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation theron. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of any funding agency.