PyMTL3

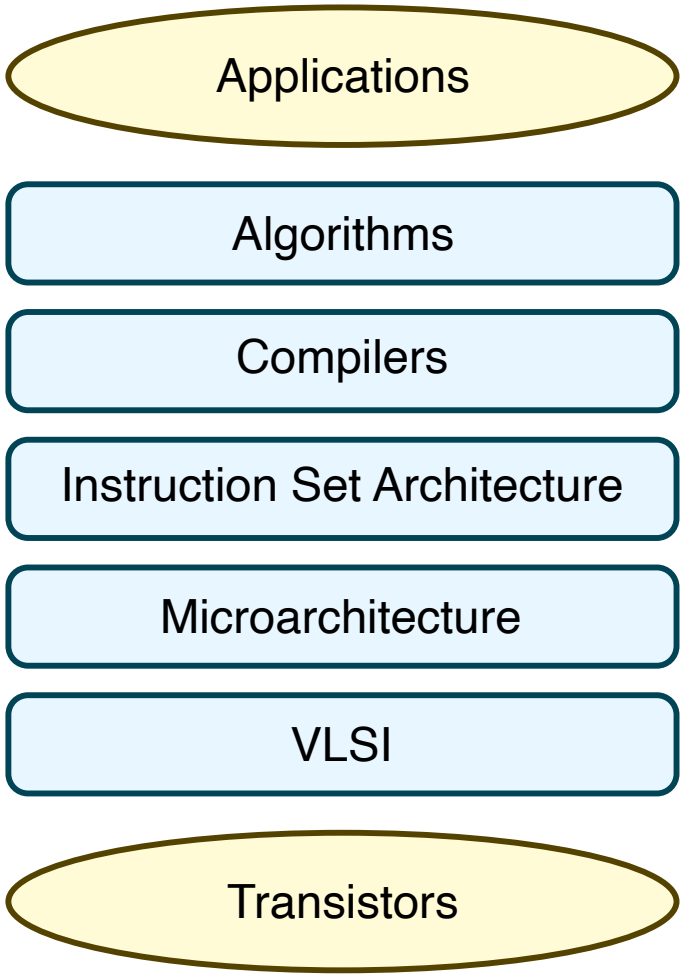
A Python Framework for Custom ASIC Design and Verification

`https://github.com/pymtl/pymtl3`

Christopher Batten

Electrical and Computer Engineering
Cornell University

Traditional 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

Traditional 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

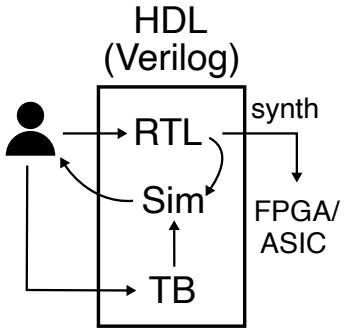
- Design-Space Exploration
- C++ Simulation Framework
- SW-Focused Object-Oriented
- gem5, SESC, McPAT

Register-Transfer-Level Modeling

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

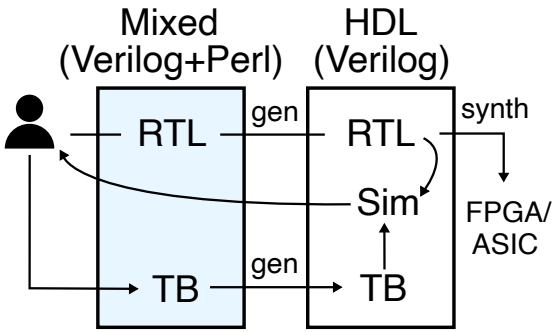
Traditional RTL Design Methodologies

HDL
Hardware Description Language



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

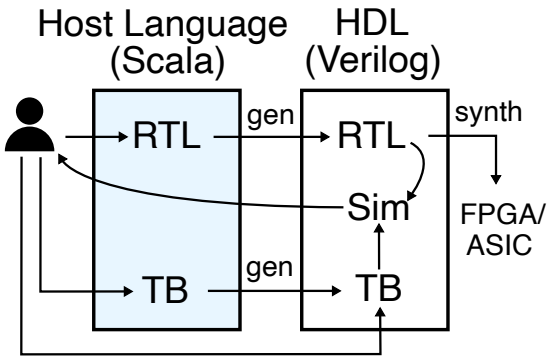
HPF
Hardware Preprocessing Framework



Example: Genesis2

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

HGF
Hardware Generation Framework



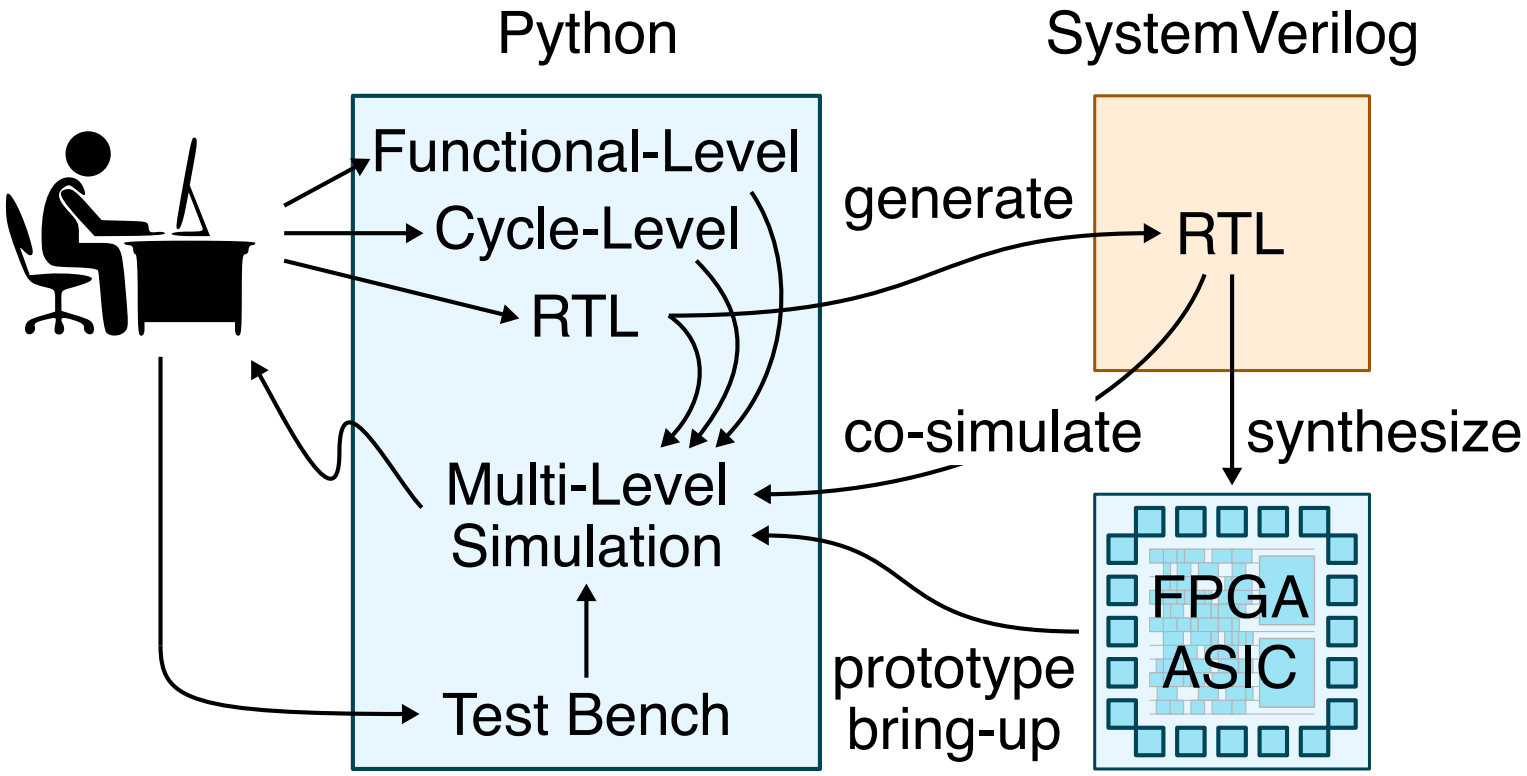
Example: Chisel

- ✗ 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** RTL design methodology?

PyMTL

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



PyMTL3: A Python Framework for Custom ASIC Design and Verification

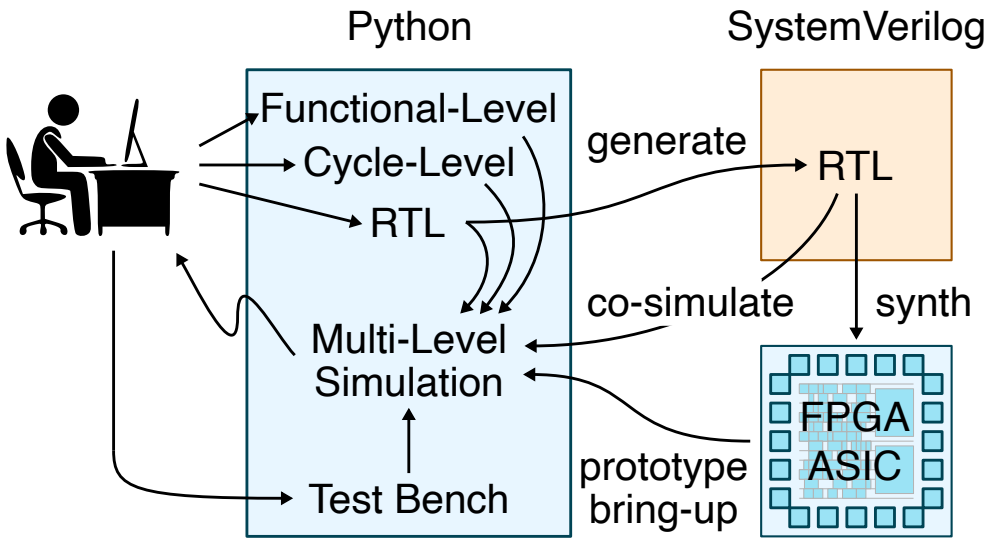
PyMTL3 Motivation

PyMTL3 for Design

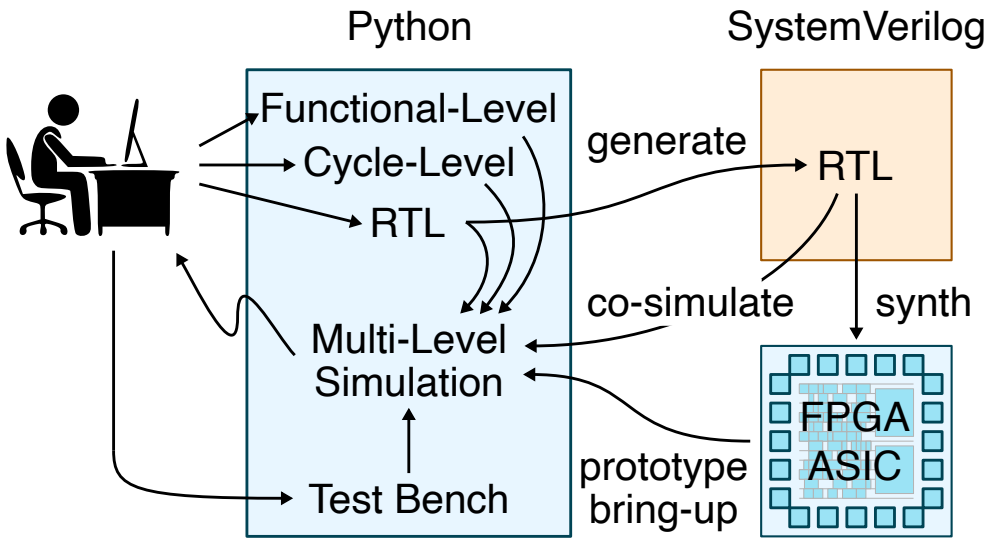
PyMTL3 for Testing

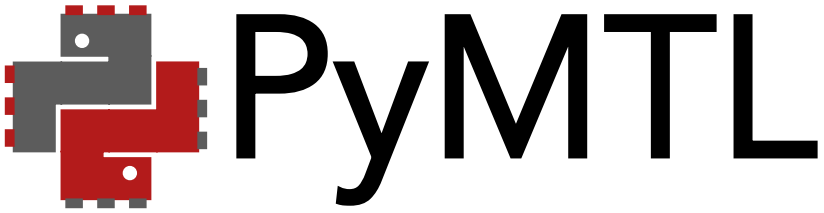
PyMTL3 in Practice

PyMTL3 Retrospective



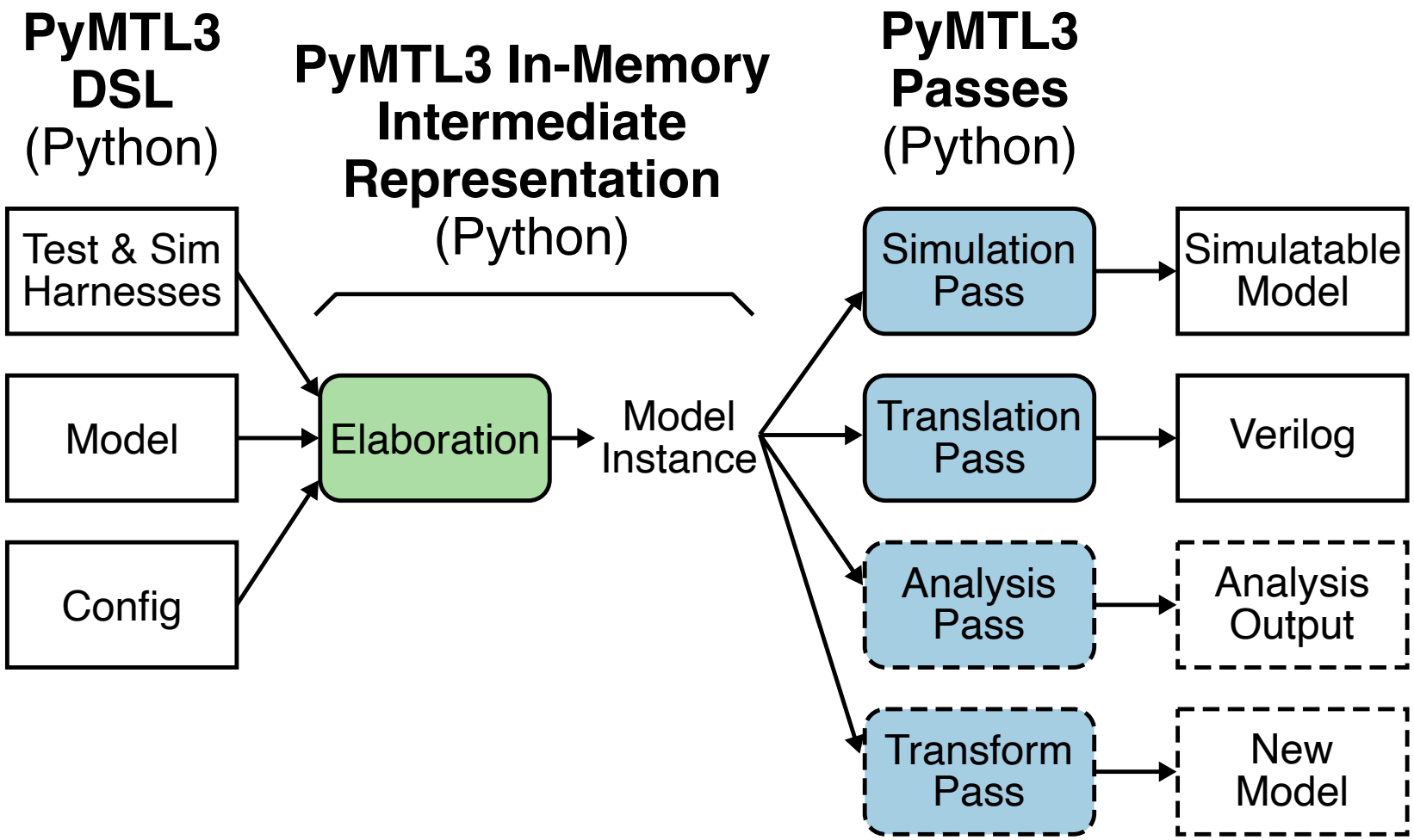
PyMTL3: A Python Framework for Custom ASIC Design and Verification





- **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 in May 2020
  - 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

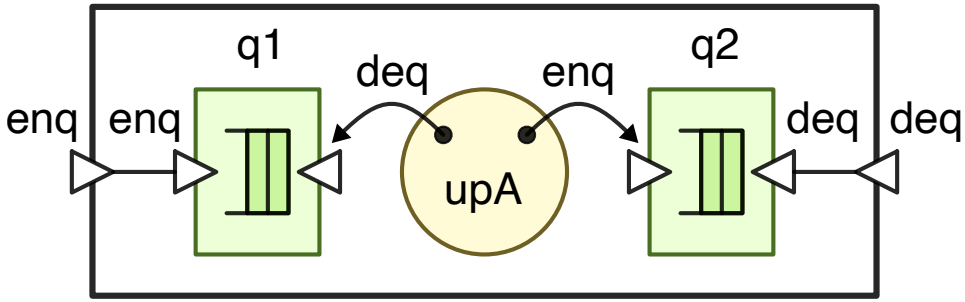
The PyMTL3 Framework



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()
```

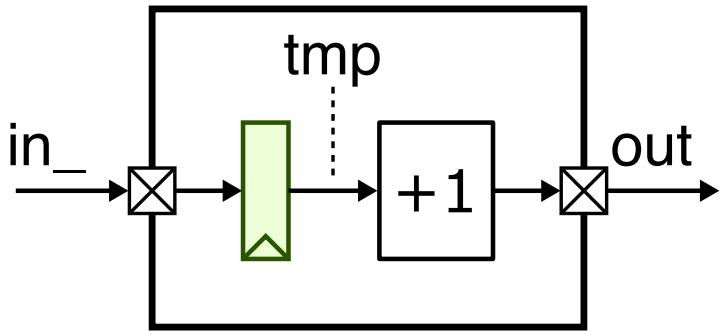
- FL/CL components can use method-based interfaces
- Structural composition via connecting methods




```
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() )
```

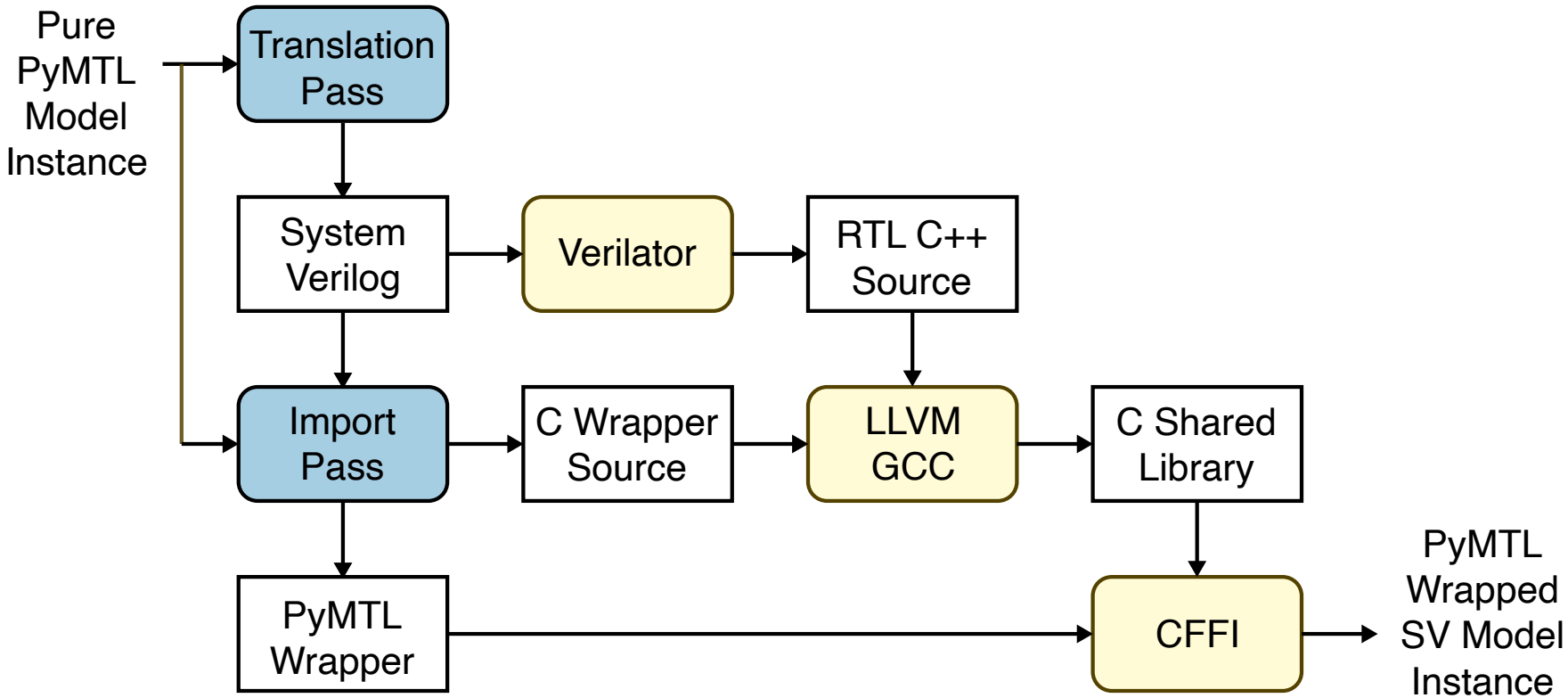
PyMTL3 Low-Level Modeling

```
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
```



- Hardware modules are Python classes derived from `Component`
- `construct` method for constructing (elaborating) hardware
- `ports` and `wires` for signals
- `update` blocks for modeling combinational and sequential logic

SystemVerilog Translation and Import



- Translation+import enables easily testing translated SystemVerilog
- Also acts like a JIT compiler for improved RTL simulation speed
- Can also import external SystemVerilog IP for co-simulation

PyMTL3: A Python Framework for Custom ASIC Design and Verification

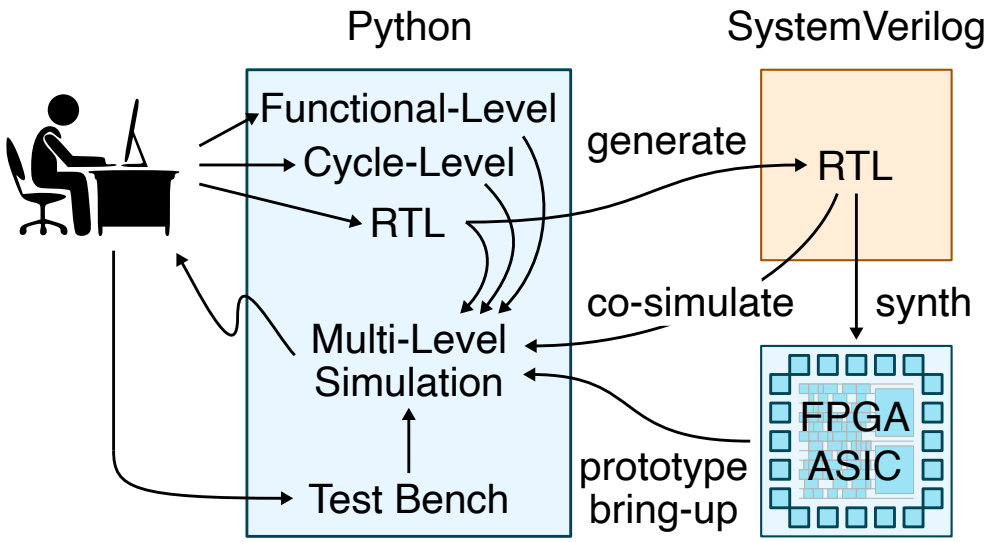
PyMTL3 Motivation

PyMTL3 for Design

PyMTL3 for Testing

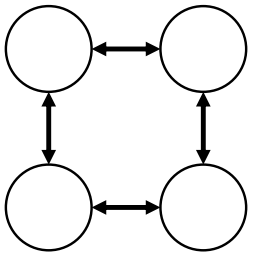
PyMTL3 in Practice

PyMTL3 Retrospective



Testing highly parameterized designs is challenging

Testing a specific ring network instance requires a number of different test cases

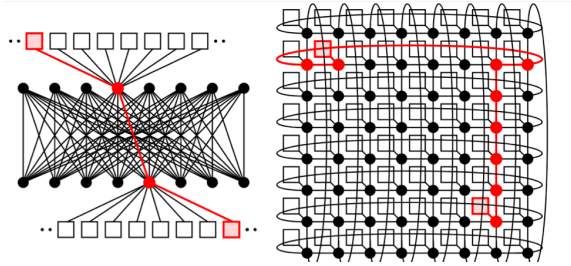


```
test_ring_1pkt_2x2_0_chnl
test_ring_2pkt_2x2_0_chnl
test_ring_2pkt_2x2_0_chnl
test_ring_self_2x2_0_chnl
test_ring_clockwise_2x2_0_chnl
test_ring_aclockwise_2x2_0_chnl
test_ring_neighbor_2x2_0_chnl
test_ring_tornado_2x2_0_chnl
test_ring_backpressure_2x2_0_chnl
…
```

```
pkt( src=0, dst=1, payload=0xdeadbeef )
pkt( src=0, dst=3, payload=0x00000003 )
pkt( src=1, dst=0, payload=0x00010000 )
pkt( src=1, dst=2, payload=0x00010002 )
pkt( src=2, dst=1, payload=0x00020001 )
pkt( src=2, dst=3, payload=0x00020003 )
pkt( src=3, dst=2, payload=0x00030002 )
pkt( src=3, dst=0, payload=0x00030000 )
pkt( src=0, dst=1, payload=0x00001000 )
pkt( src=1, dst=2, payload=0x10002000 )
pkt( src=2, dst=3, payload=0x20003000 )
pkt( src=3, dst=0, payload=0x30000000 )
pkt( src=0, dst=3, payload=0x00003000 )
pkt( src=1, dst=0, payload=0x10000000 )
pkt( src=2, dst=1, payload=0x20001000 )
pkt( src=3, dst=2, payload=0x30002000 )
…
```

Ideal testing technique:

1. Detect error quickly with **small number of test cases**
2. The failing test case has **minimal number of transactions**
3. The bug trace has **simplest transactions**
4. The failing test case has the **simplest design**



A design generator can have many parameters: topology, routing, flow control, channel latency

Software Testing Techniques

- Complete Random Testing (CRT)
  - Randomly generate input data
  - Detects error quickly
  - Debug complicated test case
- Iterative Deepened Testing (IDT)
  - Gradually increase input complexity
  - Finds bug with simple input
  - Takes many test cases to find bug
- Property-Based Testing (PBT)
  - Define invariants
  - Automatically generate examples
  - Automatically shrinking failing tests
  - Increasingly state-of-the-art in software testing

```
def gcd( a, b ):
  while b > 0:
    a, b = b, a % b
  return a

def test_crt():
  for _ in range( 100 ):
    a = random.randint( 1, 128 )
    b = random.randint( 1, 128 )
    assert gcd( a, b ) == math.gcd( a, b )

def test_idt():
  for a_max in range( 1, 128 ):
    for b_max in range( 1, 128 ):
      assert gcd( a, b ) == math.gcd( a, b )

@hypothesis.given(
  a = hypothesis.strategies.integers( 1, 128 ),
  b = hypothesis.strategies.integers( 1, 128 ),
)
def test_pbt( a, b ):
  assert gcd( a, b ) == math.gcd( a, b )
```

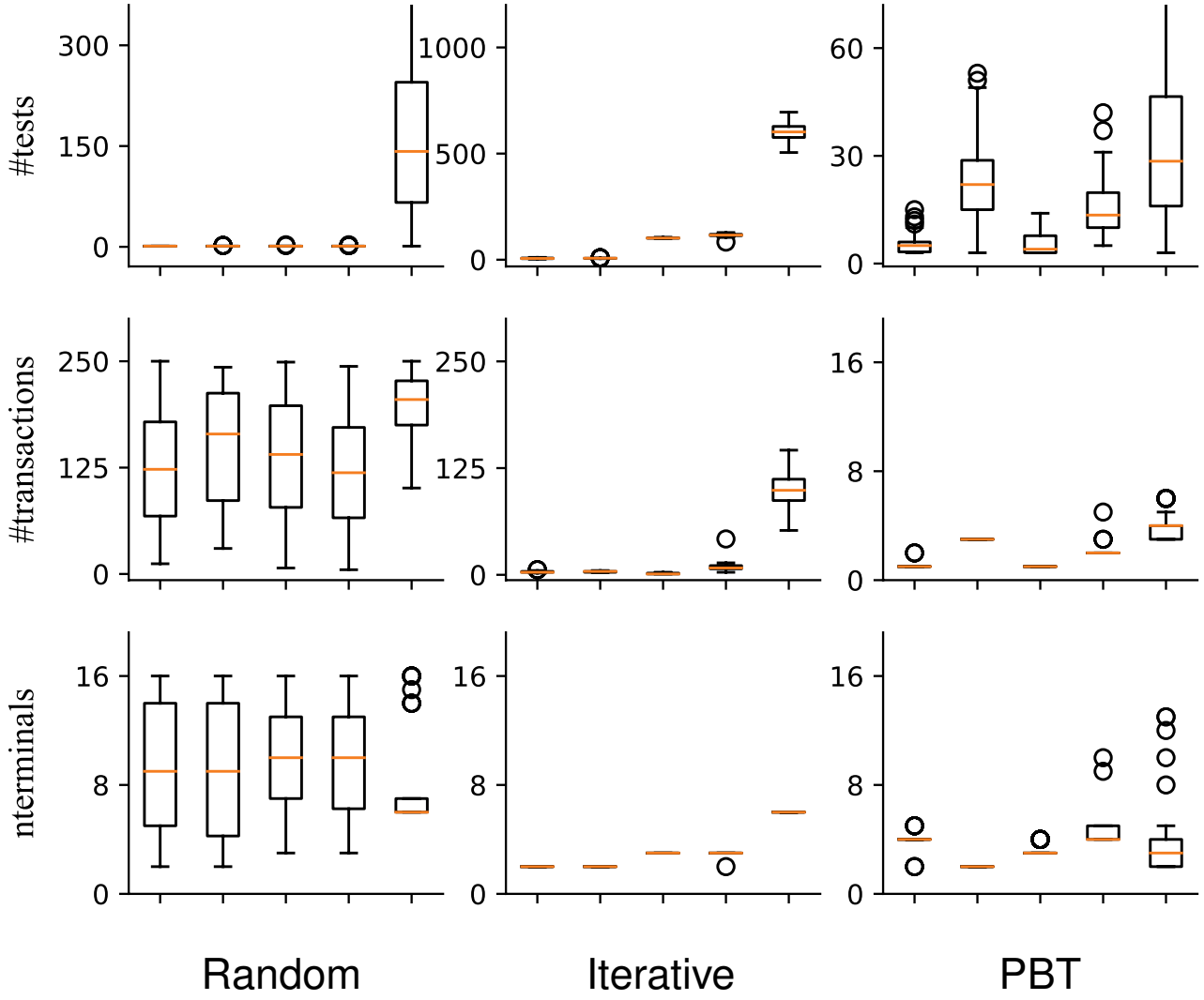
PyMTL3 creatively adapts PBT to test HW

- PyMTL3 uses **Hypothesis**, a property-based testing (PBT) framework for Python software, to create a PBT framework for hardware
- PyMTL3 leverages PBT to explore not just the input values for an RTL design but to also **explore the parameter values** used to configure an RTL design generator

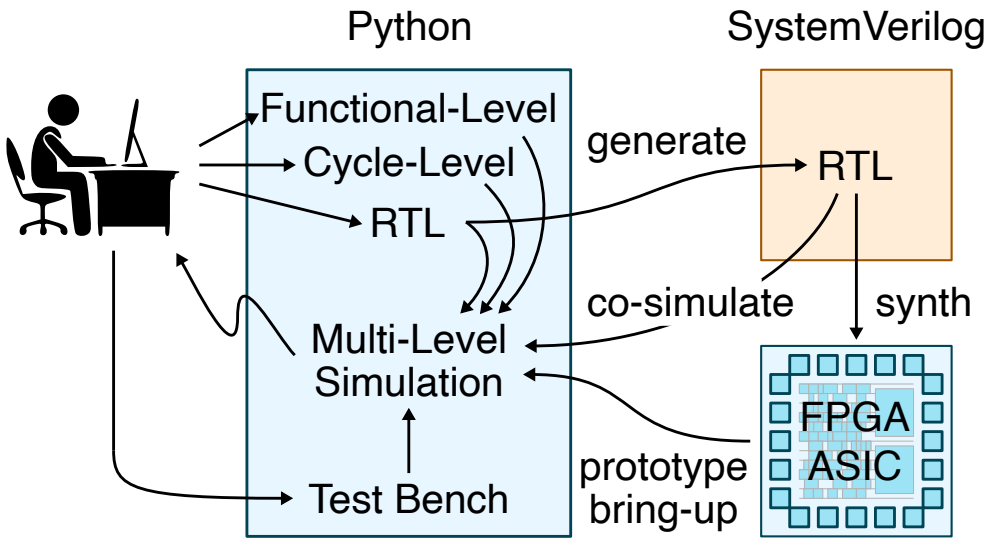
Case Study: On-chip network generator for ring topology with shortest path routing and virtual channels to avoid deadlock

```
1  from hypothesis import strategies as st
2
3  @hypothesis.given(
4    nterminals = st.integers(2,16),
5    test_pkts = st.lists(pkt_strategy())
6  )
7  def test_ring( nterminals, test_pkgs ):
8    dut = RingNetwork( nterminals )
9    th  = TestHarness( dut, test_pkts )
10   run_sim( th )
```

Example of PyMTL3 + PBT for Ring Network



PyMTL3: A Python Framework for Custom ASIC Design and Verification



PyMTL for Cycle-Level Modeling

Appears in the Proceedings of the 47th Int'l Symp. on Microarchitecture (MICRO-47), December 2014

Architectural Specialization for Inter-Iteration Loop Dependence Patterns

Shreesha Srinath, Berkin Ilbeyi, Mingxing Tan, Gai Liu, Zhiru Zhang, and Christopher Batten

School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{ss2783,bi45,mt453,gl387,zhiruz,cbatten}@cornell.edu

Using Intra-Core Loop-Task Accelerators to Improve the Productivity and Performance of Task-Based Parallel Programs

Ji Kim Shunning Jiang Christopher Torng Moyang Wang
Shreesha Srinath Berkin Ilbeyi Khalid Al-Hawaj Christopher Batten

School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{ jyk46, sj634, clt67, mw828, ss2783, bi45, ka429, cbatten }@cornell.edu

In practice, we use PyMTL more for RTL modeling than CL modeling

PyMTL for RTL Modeling

Appears in the Proceedings of the 51st ACM/IEEE Int'l Symp. on Microarchitecture (MICRO-51), October 2018

An Architectural Framework for Accelerating Dynamic Parallel Algorithms on Reconfigurable Hardware

Tao Chen, Shreesha Srinath, Christopher Batten and G. Edward Suh
Cornell University
Ithaca, NY 14850, USA
{tc466, ss2783, cbatten, gs272}@cornell.edu

Appears in the Proceedings of the Int'l Symp. on Networks-on-Chips (NOCS-14), September 2020

Implementing Low-Diameter On-Chip Networks for Manycore Processors Using a Tiled Physical Design Methodology

Special Session Paper

Yanghui Ou, Shady Agwa, Christopher Batten
School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{ yo96, sr972, cbatten }@cornell.edu

Appears in the Proceedings of the 27th IEEE Int'l Symp. on High-Performance Computer Architecture (HPCA-27), Feb 2021

Ultra-Elastic CGRAs for Irregular Loop Specialization

Christopher Torng[2*], Peitian Pan[1], Yanghui Ou[1], Cheng Tan[1], and Christopher Batten[1]
[1]Cornell University, Ithaca, NY [2]Stanford University, Stanford, CA
{ clt67, pp482, yo96, ct535, cbatten }@cornell.edu

Appears in the Proceedings of the 55th ACM/IEEE Int'l Symp. on Microarchitecture (MICRO-55), October 2022

big.VLITTLE: On-Demand Data-Parallel Acceleration for Mobile Systems on Chip

Tuan Ta, Khalid Al-Hawaj, Nick Cebry, Yanghui Ou, Eric Hall, Courtney Golden, and Christopher Batten

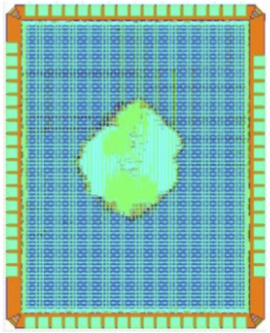
School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{qtt2,ka429,nfc35,yo96,ewh73,ckg35,cbatten}@cornell.edu

PyMTL3 has been used in may tapeouts

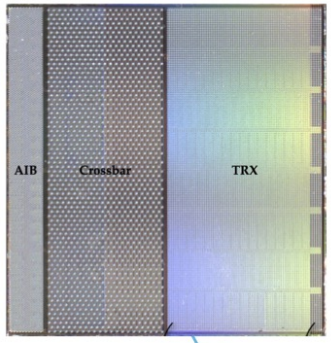
TSMC 180/65/28/16nm
GF 130/12nm; Intel 22FFL

- RISC-V microcontrollers
- Manycore architectures
- Accelerators
- Mesh on-chip networks
- Crossbar interconnects

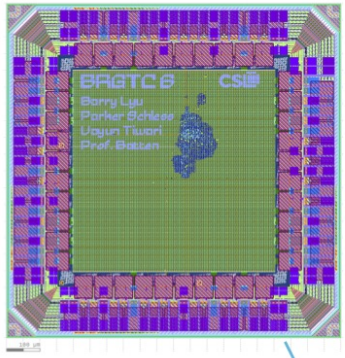
BRGTC3&4
TSMC 180nm
2x2.5mm

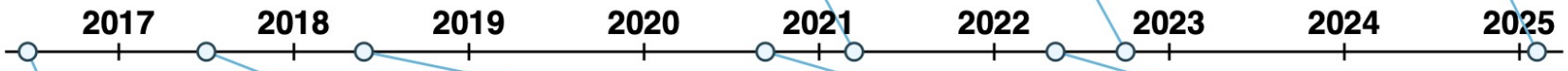


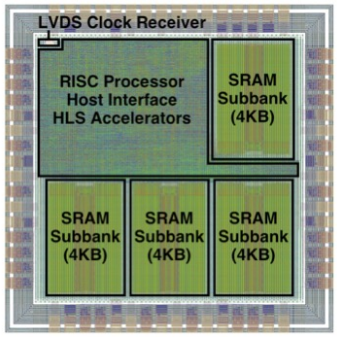
PIPES EIC
Intel 16nm
8x8mm



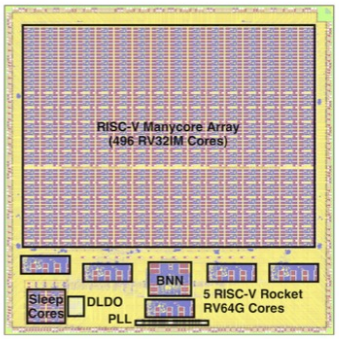
BRGTC6
TSMC 65nm
1x1mm



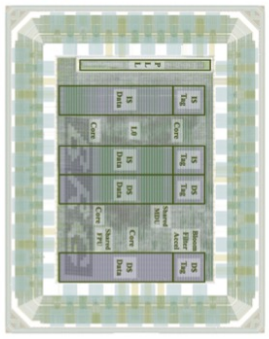




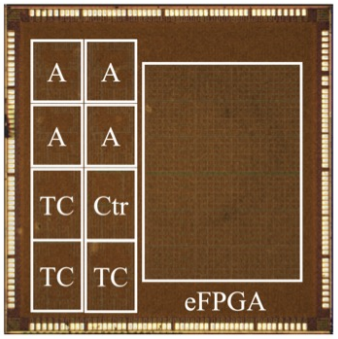
BRGTC1
IBM 130nm
2x2mm



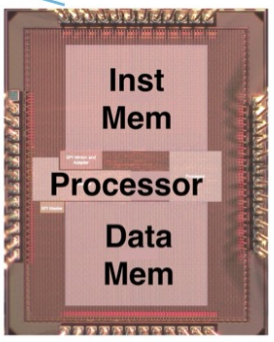
Celerity
TSMC 16nm
5x5mm



BRGTC2
TSMC 28nm
1x1.25mm

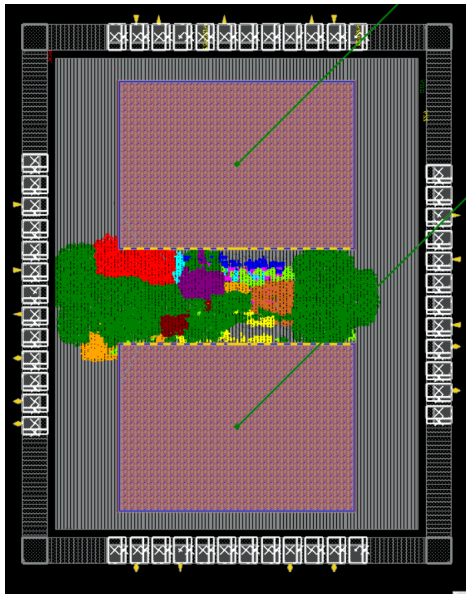


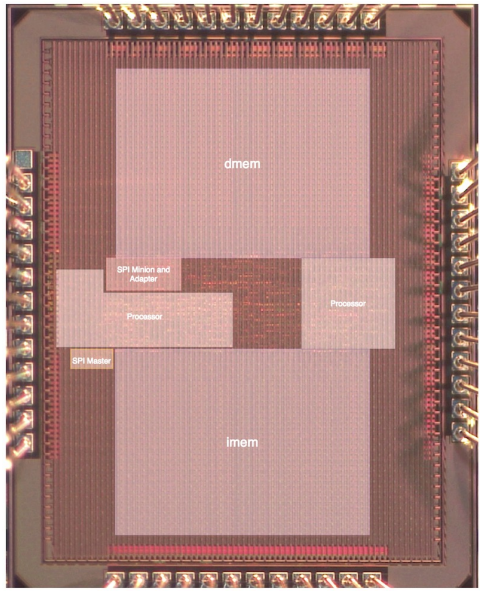
CIFER
GF 12nm
4x4.5mm

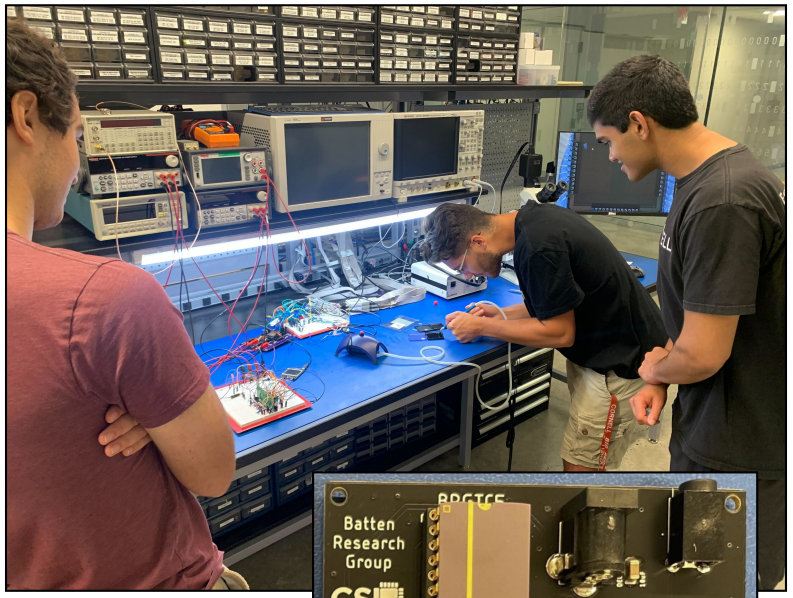


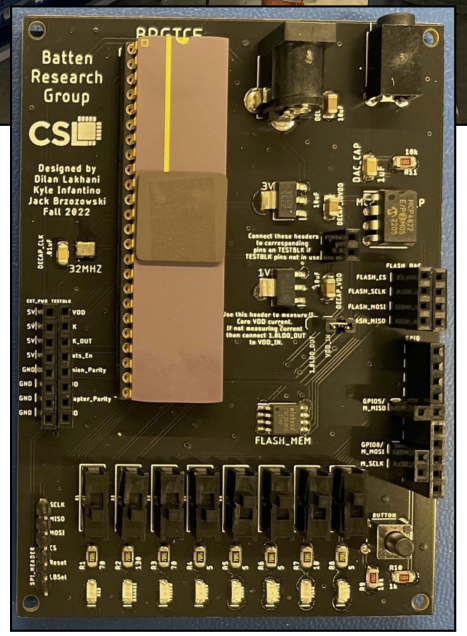
BRGTC5
TSMC 180nm
2x2.5mm

BRG Test Chip #5 (2022)





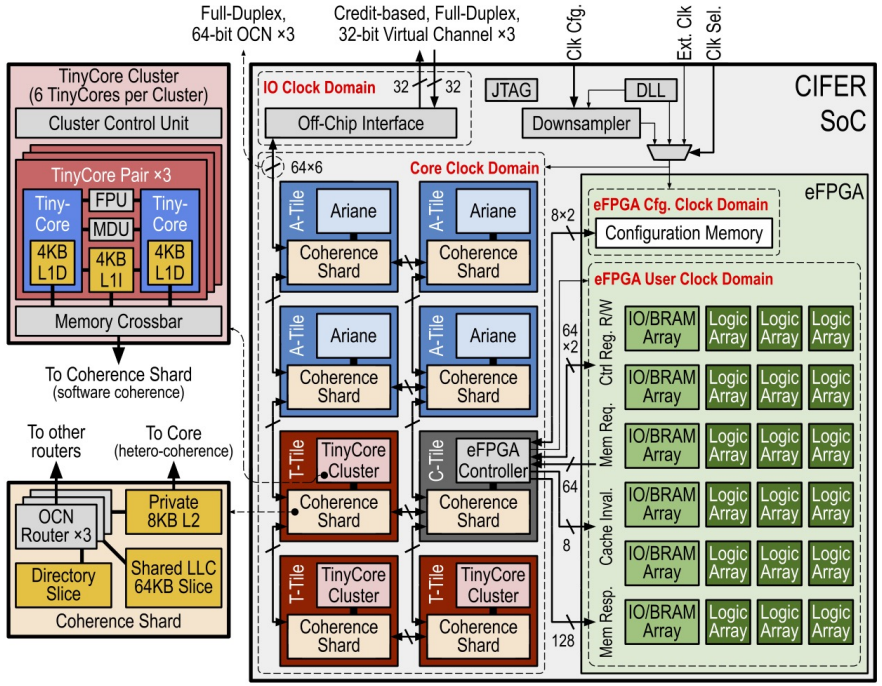




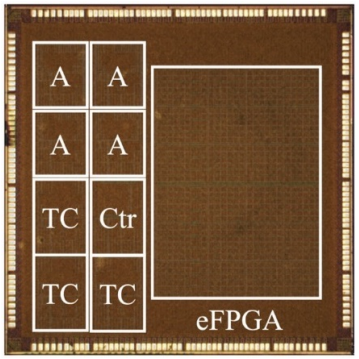
- Three undergraduates $\rightarrow$ MEng
- 2$\times$2.5mm in TSMC 180nm
- RISC-V RV32IM micro-controller
- 16KB of instruction SRAM, 16KB of data SRAM
- SPI interface for config, SPI master, GP I/O
- 100% done using PyMTL3 (including chip bring-up)

Cifer SoC for DARPA POSH (2021)

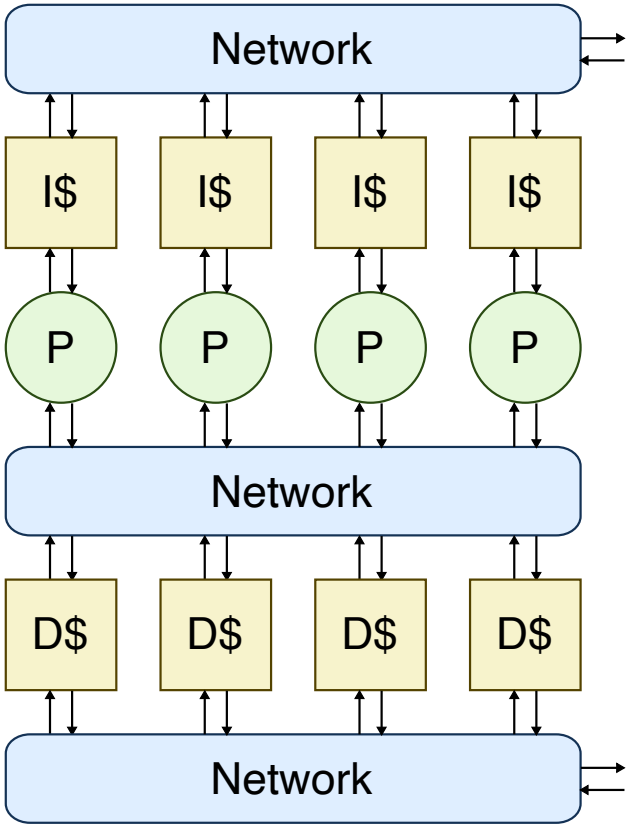
- Project led by David Wentlzaff's group at Princeton
- 4 × 4mm in GF 12 nm
- 450 million transistors
- 4 Linux-capable Ariane cores
- 1 Embedded FPGA
- 3 TinyCore clusters
  - 6 RISC-V RV32IMAF cores
  - 4KB private L1 data cache
  - Pairs share icache, MDU, FPU
  - Software-centric coherence
- Mesh based on-chip network
- TinyCore clusters and on-chip network implemented using PyMTL3 and SystemVerilog import of DesignWare floating-point units



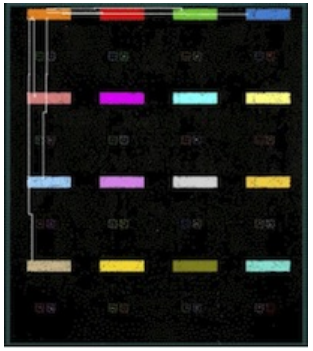
[CICC'23,SSCL'23]

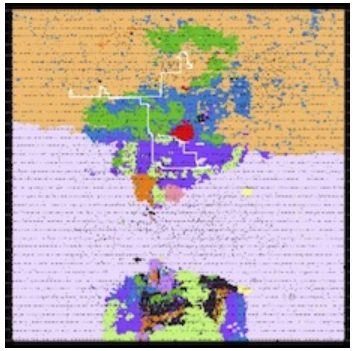


PyMTL3 for Undergraduate and Graduate Courses

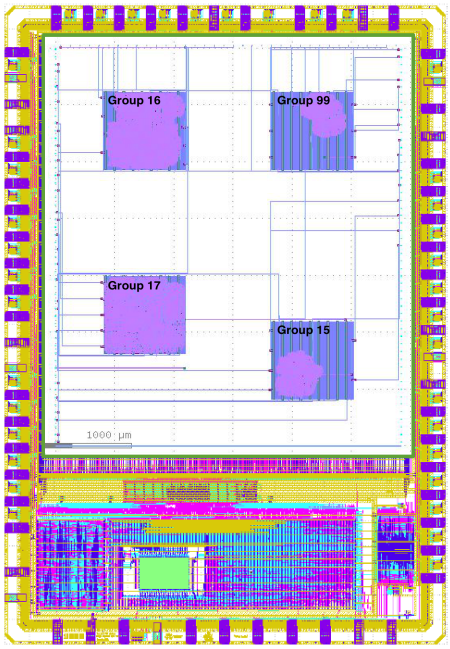


Computer Arch Course
Labs use PyMTL for verification, PyMTL or Verilog for RTL design



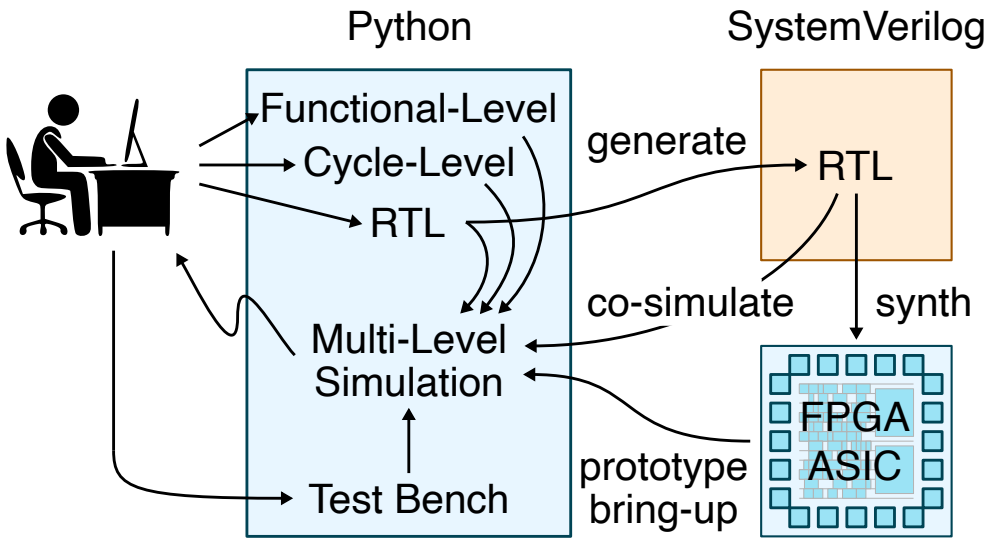


Chip Design Course
Labs use PyMTL for verification, PyMTL or Verilog for RTL design, standard ASIC flow



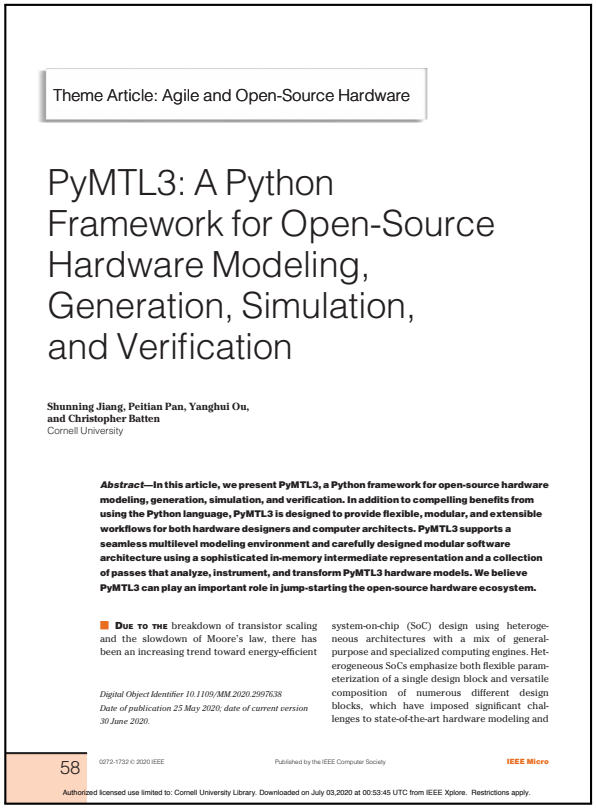
First Teaching Tapeout in 10+ years!
Four student projects
All use PyMTL for testing
Two use PyMTL for design

PyMTL3: A Python Framework for Custom ASIC Design and Verification



PyMTL3 Publications

- S. Jiang, B. Ilbeyi, et al., "Mamba: Closing the Performance Gap in Productive Hardware Development Frameworks." *DAC*, June 2018.
- S. Jiang, P. Pan, Y. Ou, et al., "PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification." *IEEE Micro*, 40(4):58–66, Jul/Aug. 2020.
- S. Jiang*, Y. Ou*, P. Pan, et al., "PyH2: Using PyMTL3 to Create Productive and Open-Source Hardware Testing Methodologies." IEEE Design & Test, 38(2):53–61, Apr. 2021.
- S. Jiang, Y. Ou, P. Pan, et al., "UMOC: Unified Modular Ordering Constraints to Unify Cycle- and Register-Transfer-Level Modeling." *DAC*, Dec. 2021.
- P. Pan, S. Jiang, et al., "Symbolic Elaboration: Checking Generator Properties in Dynamic Hardware Description Languages." *MEMOCODE*, Sep. 2023.
- P. Pan and C. Batten, "Formal Verification of the Stall Invariant Property for Latency-Insensitive RTL Modules." *MEMOCODE*, Sep. 2023.



Theme Article: Agile and Open-Source Hardware

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*—In this article, we present PyMTL3, a Python framework for open-source hardware modeling, generation, simulation, and verification. In addition to compelling benefits from using the Python language, PyMTL3 is designed to provide flexible, modular, and extensible workflows for both hardware designers and computer architects. PyMTL3 supports a seamless multilevel 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. We believe 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 toward 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

Digital Object Identifier 10.1109/MM.2020.2997638
Date of publication 25 May 2020; date of current version 30 June 2020.

Published by the IEEE Computer Society IEEE Micro

Six Lessons Learned After 10+ Years

- **1. Dynamic typing concerns are uninformed**
  Most Python-based embedded DSLs for hardware design support strong typing for RTL at elaboration time; consider this a form of gradual typing

- **2. Simulation speed actually is a bottleneck**
  Python-based embedded DSLs struggle to simulate large, complex designs; Python CL models just too slow compared to C++ CL models; JIT compilation can only close the gap so much

- **3. Verification is more important than design**
  Too many academic projects focus on making design more productive or rely purely on type checking and/or correctness by construction; productive testing and formal methods are equally important!

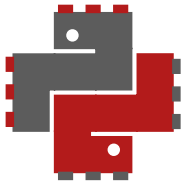
Six Lessons Learned After 10+ Years

- **4. Interoperability with standard methodologies is essential**
  RTL translation must be robust and generate readable Verilog; Verilog import must be a first-class design consideration; must support industry standard verification flows

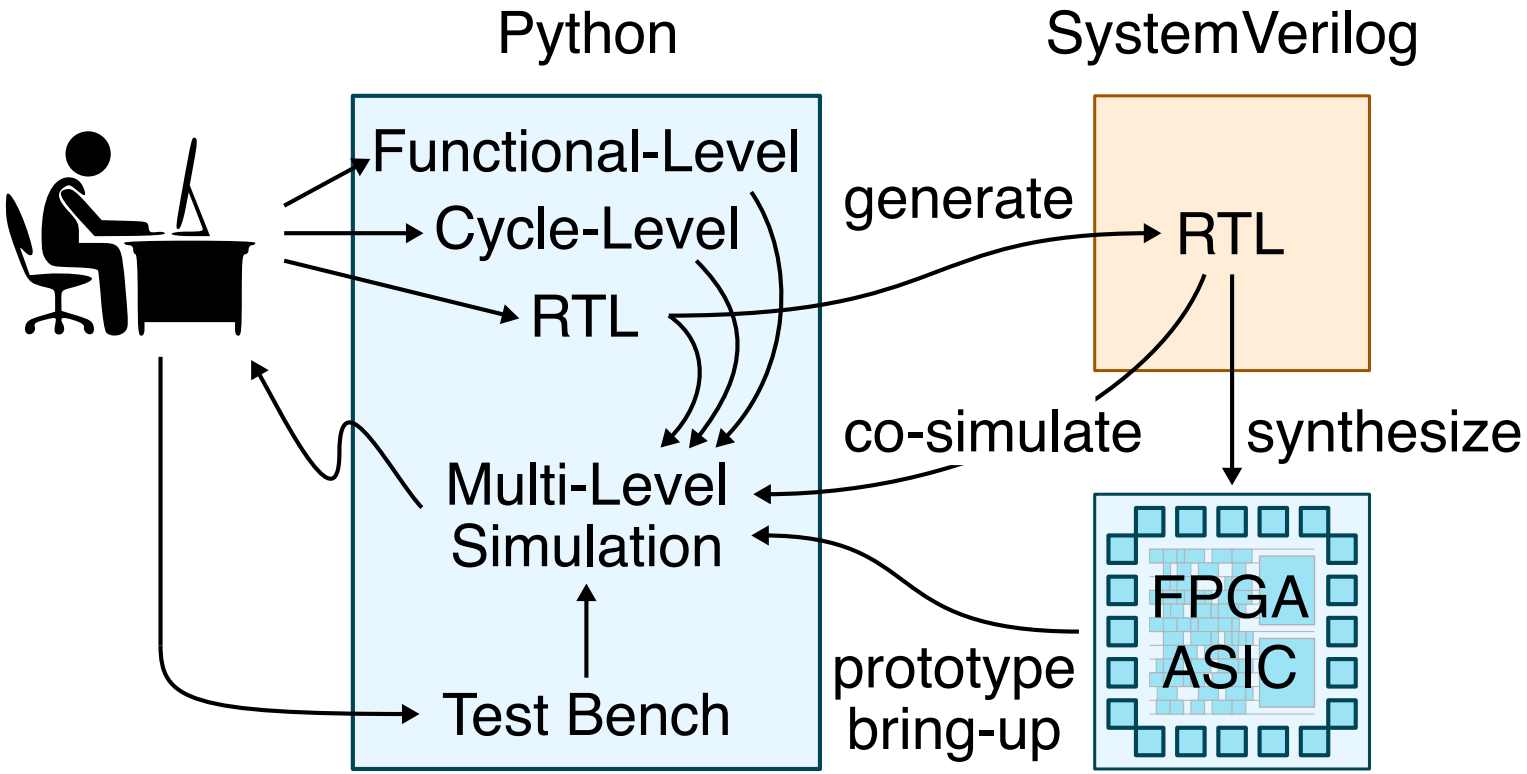
- **5. Single clock domain and synchronous reset are limiting**
  Real chips need asynchronous interfaces, source synchronous interfaces, pipelined reset signals, and/or asynchronous reset for external IP

- **6. Physical design is the crux of agile chip design**
  Python-based frameworks can help improve design and verification productivity, but physical design remains a critical bottleneck especially on small design teams with short design cycles

PyMTL



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



Core PyMTLv2 developers: Derek Lockhart, Berkin Ilbeyi

Core PyMTLv3 developers: Shunning Jian, Peitian Pan, Yanghui Ou

Thanks to Ji Kim, Shreesha Srinath, 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 testing and using PyMTL

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, DARPA POSH Award #FA8650-18-2-7852, 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.