PyMTL3

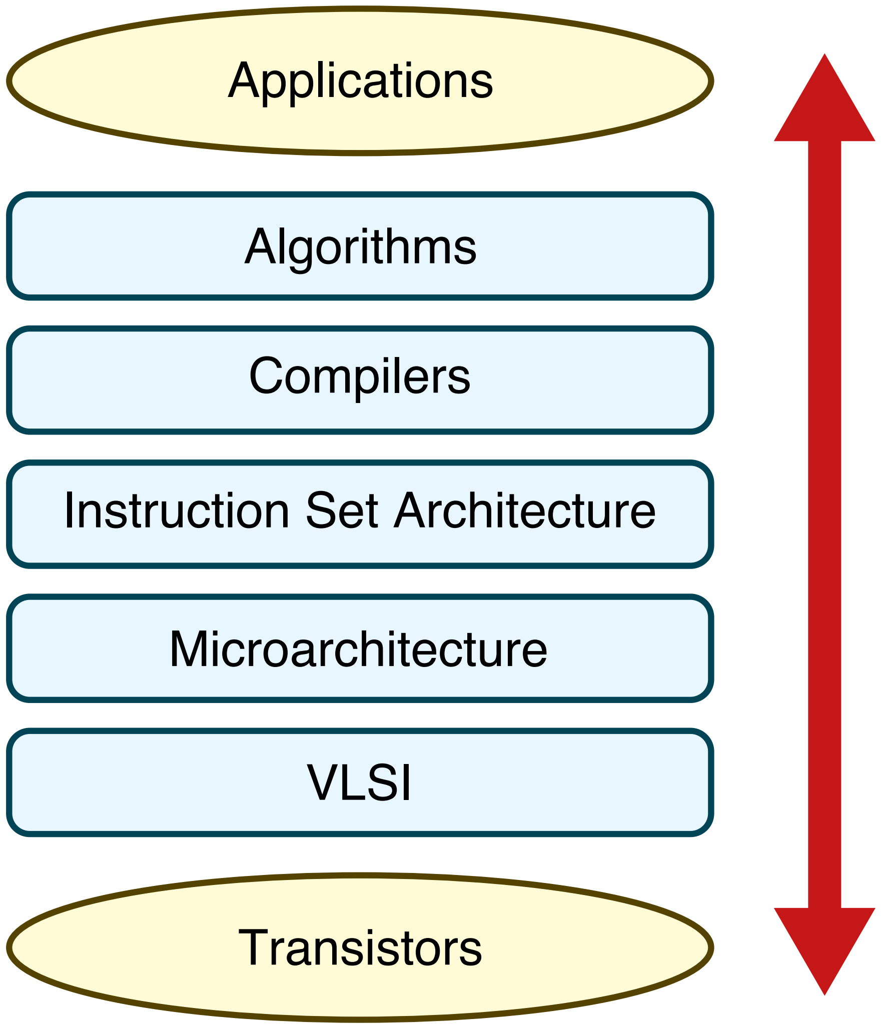
A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification

`https://pymtl.github.io`

Christopher Batten

Electrical and Computer Engineering
Cornell University

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

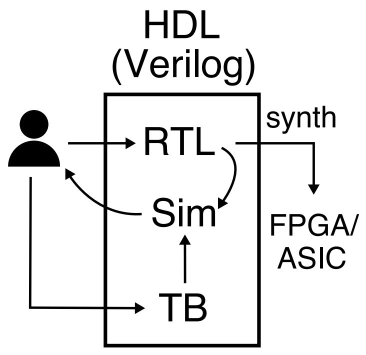
– Design-Space Exploration
– C++ Simulation Framework
– SW-Focused Object-Oriented
– gem5, SESC, McPAT

Register-Transfer-Level Modeling

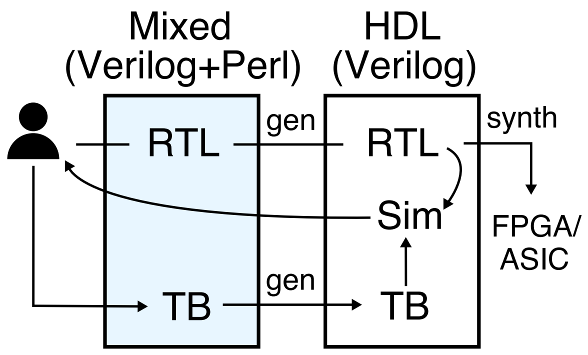
– Prototyping & AET Validation
– Verilog, VHDL Languages
– HW-Focused Concurrent Structural
– EDA Toolflow

Traditional RTL Design Methodologies

HDL
Hardware Description Language

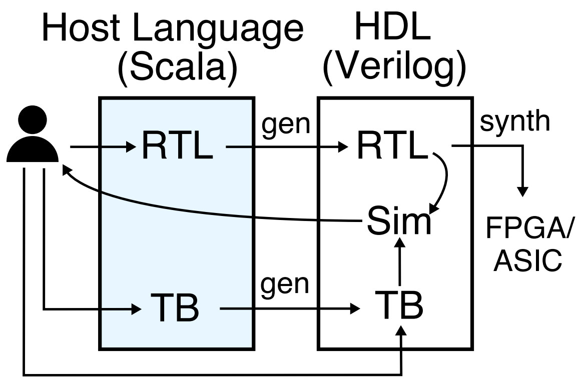


HPF
Hardware Preprocessing Framework



Example: Genesis2

HGF
Hardware Generation Framework



Example: Chisel

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

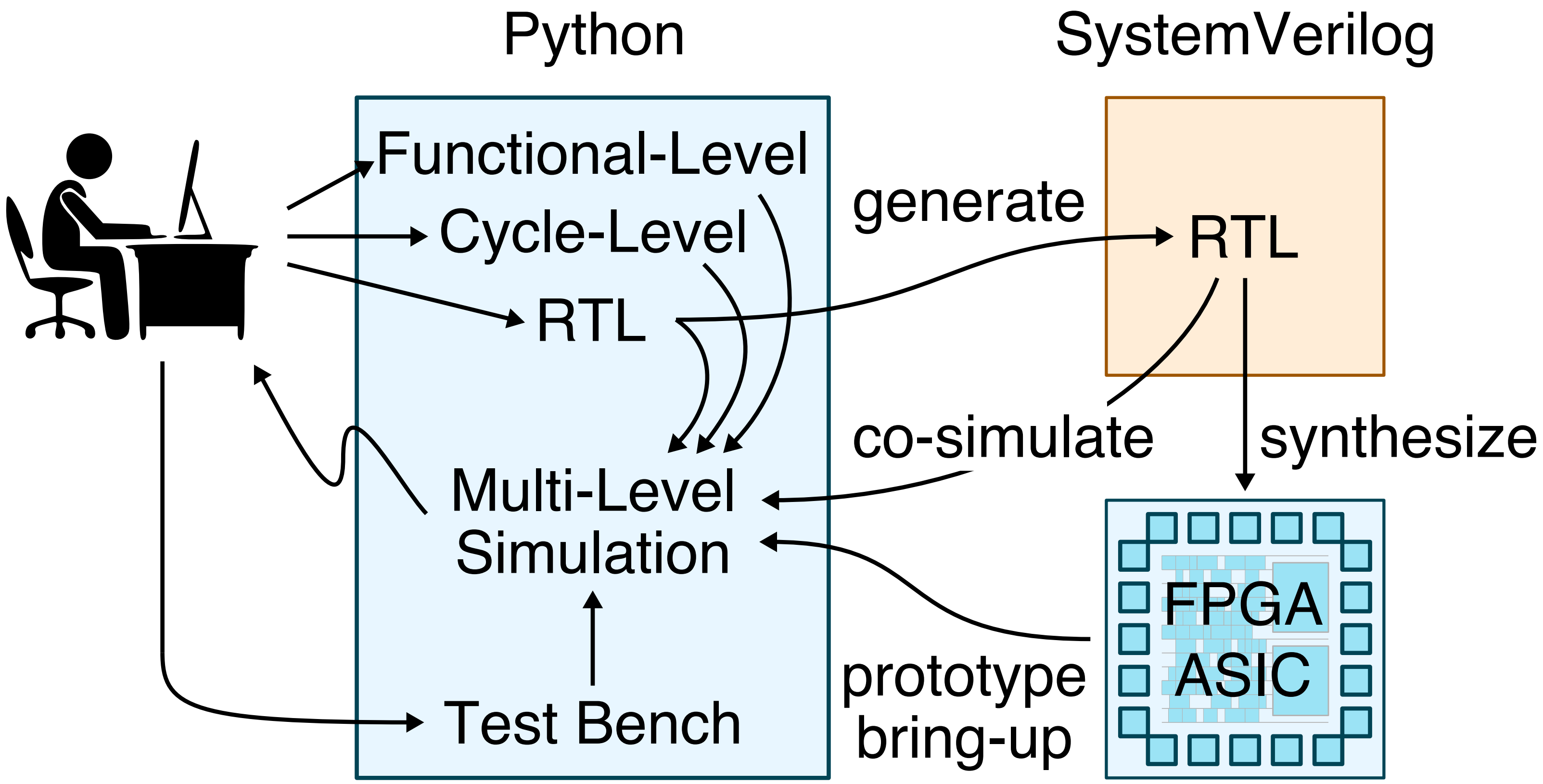
HPF:
- ✗ Slower edit-sim-debug loop
- ✗ Multiple languages create "semantic gap"
- ✓ Easier to create highly parameterized generators

HGF:
- ✗ 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 Open-Source Hardware Modeling, Generation, Simulation, and Verification

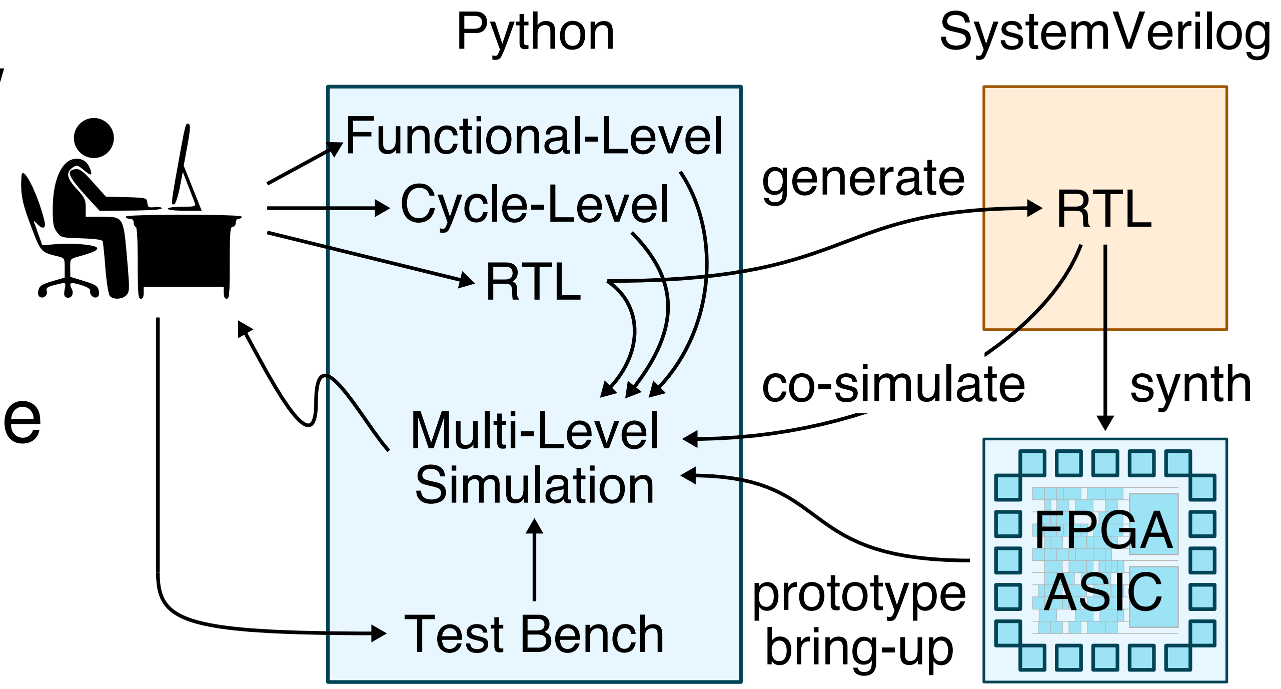
PyMTL3 Motivation

PyMTL3 Overview

PyMTL3 Demo

PyMTL2 in Practice

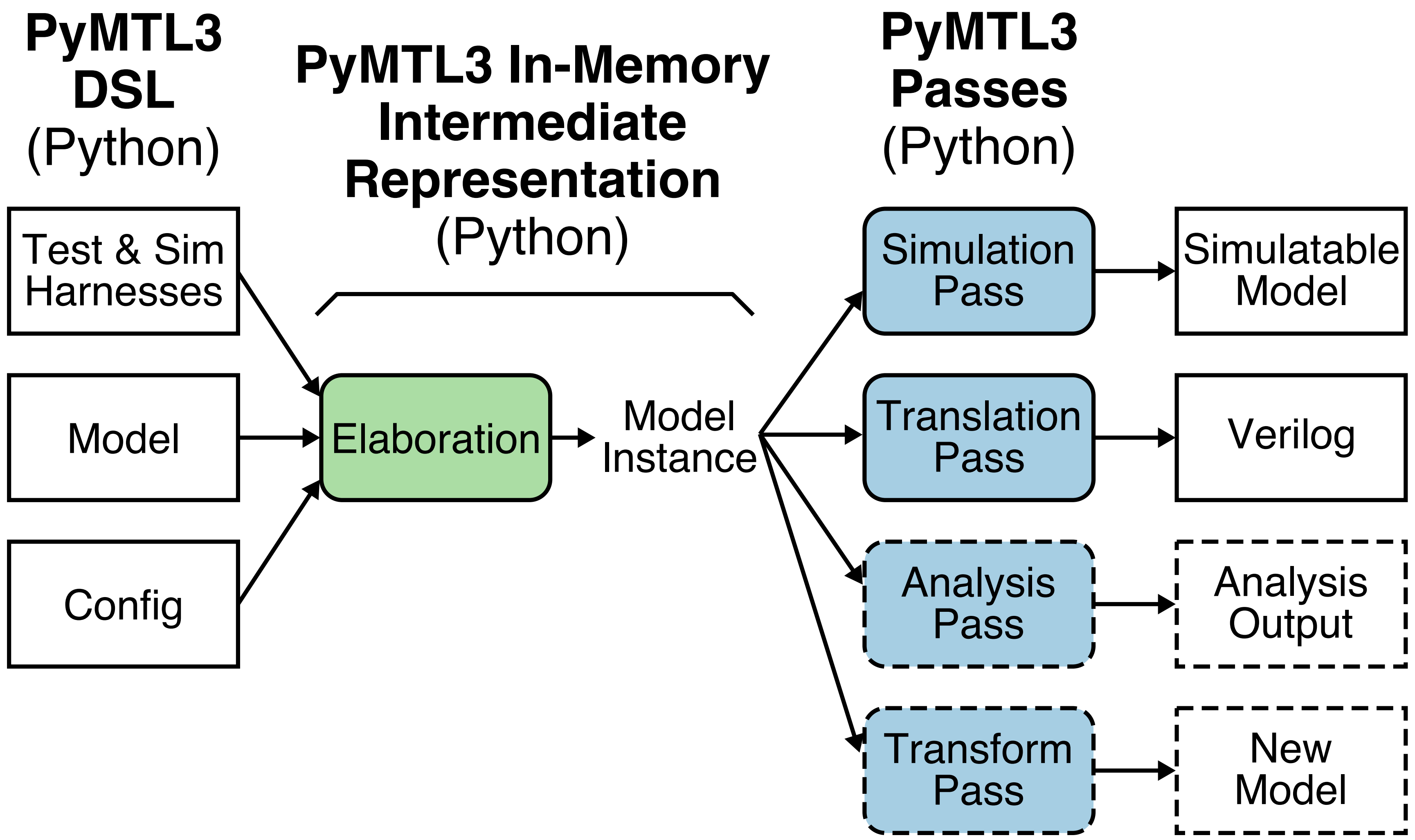
PyMTL3 + Xilinx



PyMTL

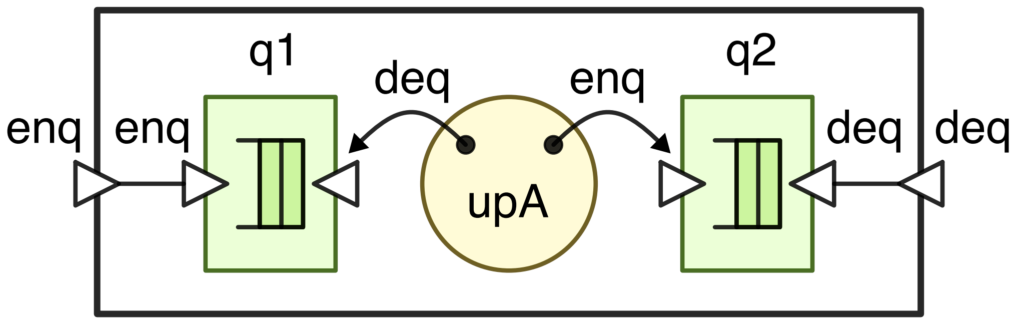
- **PyMTL2**: `https://github.com/cornell-brg/pymtl`
  - released in 2014
  - extensive experience using framework in research & teaching
- **PyMTL3**: `https://github.com/pymtl/pymtl3`
  - official release in May 2020
  - adoption of new Python3 features
  - significant rewrite to improve productivity & 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

```python
class QueueFL( Component ):
 def construct( s, maxsize ):
   s.q = deque( maxlen=maxsize )

 @non_blocking(
   lambda s: len(s.q) < s.q.maxlen )
 def enq( s, value ):
   s.q.appendleft( value )

 @non_blocking(
   lambda s: len(s.q) > 0 )
 def deq( s ):
   return s.q.pop()
```

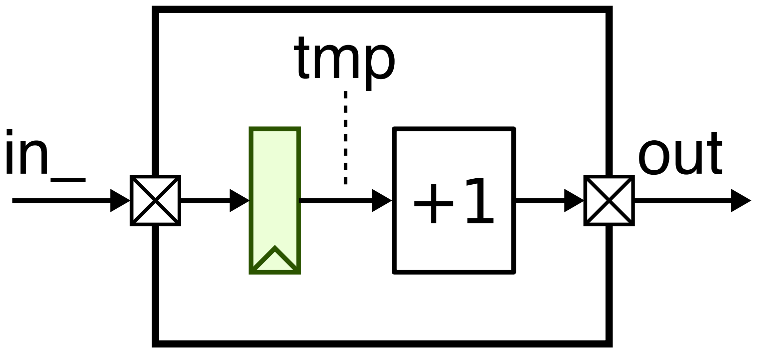


```python
class DoubleQueueFL( Component ):
 def construct( s ):
   s.enq = CalleeIfcCL()
   s.deq = CalleeIfcCL()

   s.q1 = QueueFL(2)
   s.q2 = QueueFL(2)

   connect( s.enq,    s.q1.enq )
   connect( s.q2.deq, s.deq    )

   @update
   def upA():
     if s.q1.deq.rdy() and s.q2.enq.rdy():
       s.q2.enq( s.q1.deq() )
```

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

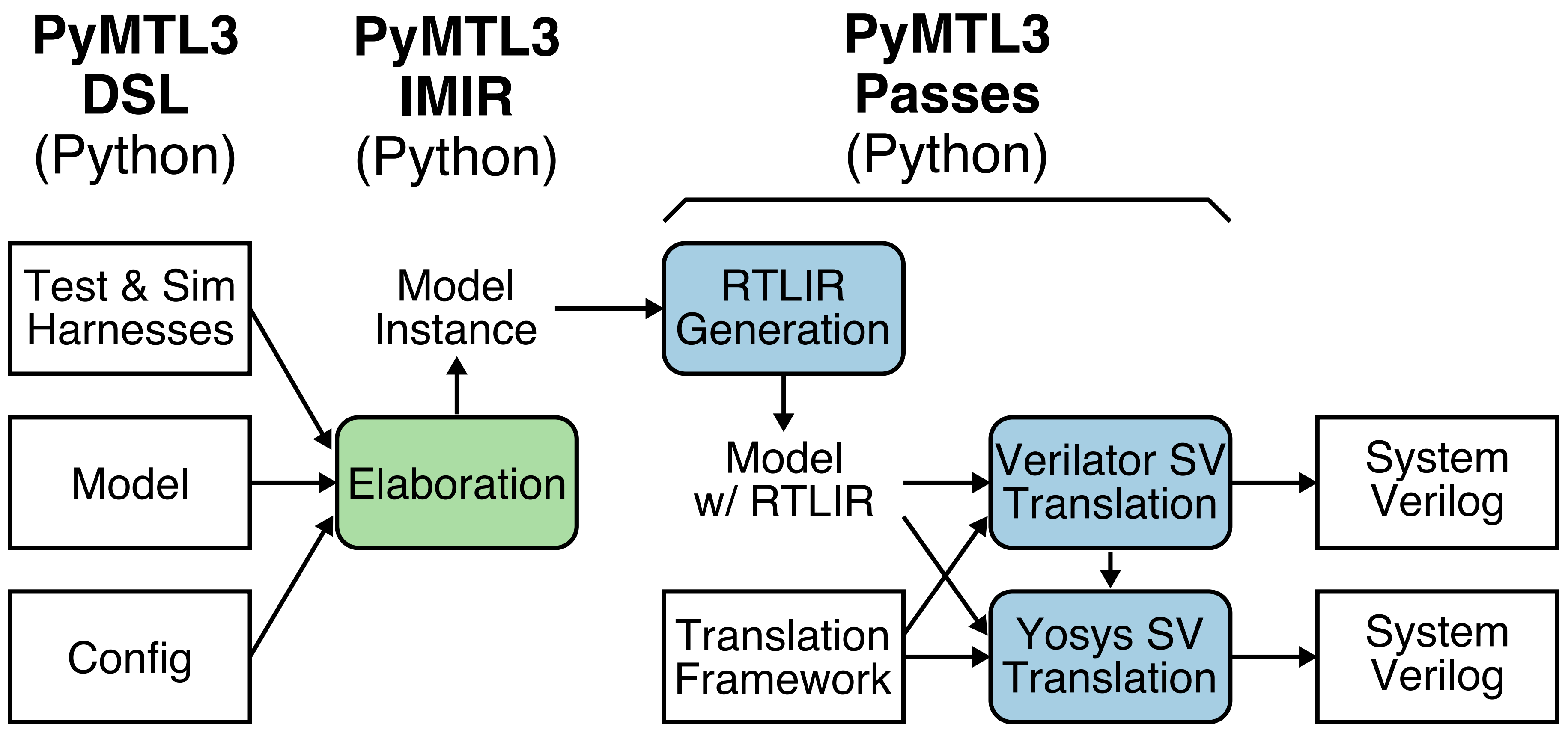
PyMTL3 Low-Level Modeling

```python
from pymtl3 import *

class RegIncrRTL( Component ):

  def construct( s, nbits ):
    s.in_ = InPort ( nbits )
    s.out = OutPort( nbits )
    s.tmp = Wire   ( nbits )

    @update_ff
    def seq_logic():
      s.tmp <<= s.in_

    @update
    def comb_logic():
      s.out @= s.tmp + 1
```



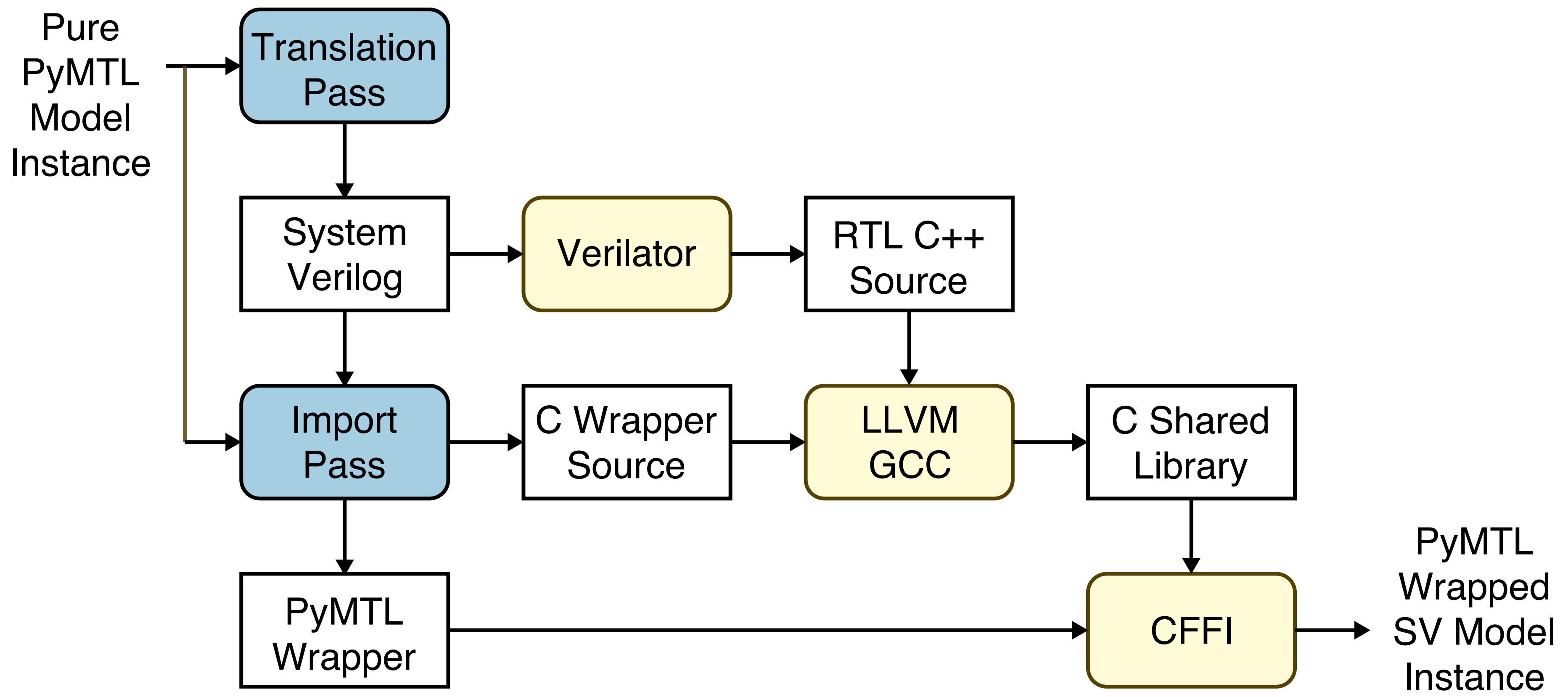
- Hardware modules are Python classes derived from `Component`
- `construct` method for constructing (elaborating) hardware
- `ports` and `wires` for signals
- `update` blocks for modeling combinational and sequential logic

SystemVerilog RTLIR/Translation Framework



- RTLIR simplifies RTL analysis passes and translation
- Translation framework simplifies implementing new translation passes

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

Translating to *Readable* SystemVerilog

```python
class StepUnit( Component ):
 def construct( s ):
   s.word_in  = InPort ( 16 )
   s.sum1_in  = InPort ( 32 )
   s.sum2_in  = InPort ( 32 )
   s.sum1_out = OutPort( 32 )
   s.sum2_out = OutPort( 32 )

   @update
   def up_step():
     temp1 = b32(s.word_in) + s.sum1_in
     s.sum1_out @= temp1 & b32(0xffff)

     temp2 = s.sum1_out + s.sum2_in
     s.sum2_out @= temp2 & b32(0xffff)
```

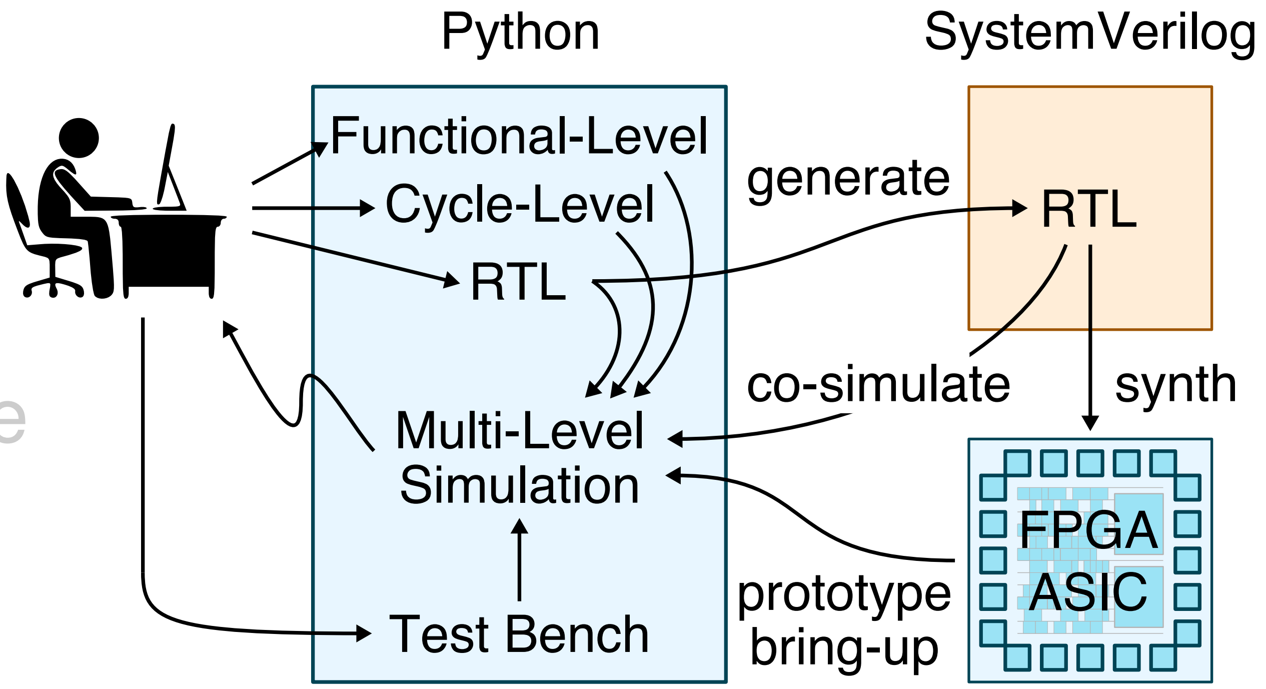
- Readable signal names
- Generates useful comments
- Simple type inference for temporary variables

```verilog
module StepUnit
(
  input  logic [0:0]    clk,
  input  logic [0:0]    reset,
  input  logic [31:0]   sum1_in,
  output logic [31:0]   sum1_out,
  input  logic [31:0]   sum2_in,
  output logic [31:0]   sum2_out,
  input  logic [15:0]   word_in
);
  // Temporary wire definitions
  logic [31:0]   __up_step$temp1;
  logic [31:0]   __up_step$temp2;

  // PYMTL SOURCE:
  // ...

  always_comb begin : up_step
    __up_step$temp1 = {{16{1'b0}},word_in} + sum1_in;
    sum1_out = __up_step$temp1 & 32'd65535;
    __up_step$temp2 = sum1_out + sum2_in;
    sum2_out = __up_step$temp2 & 32'd65535;
  end

endmodule
```

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”

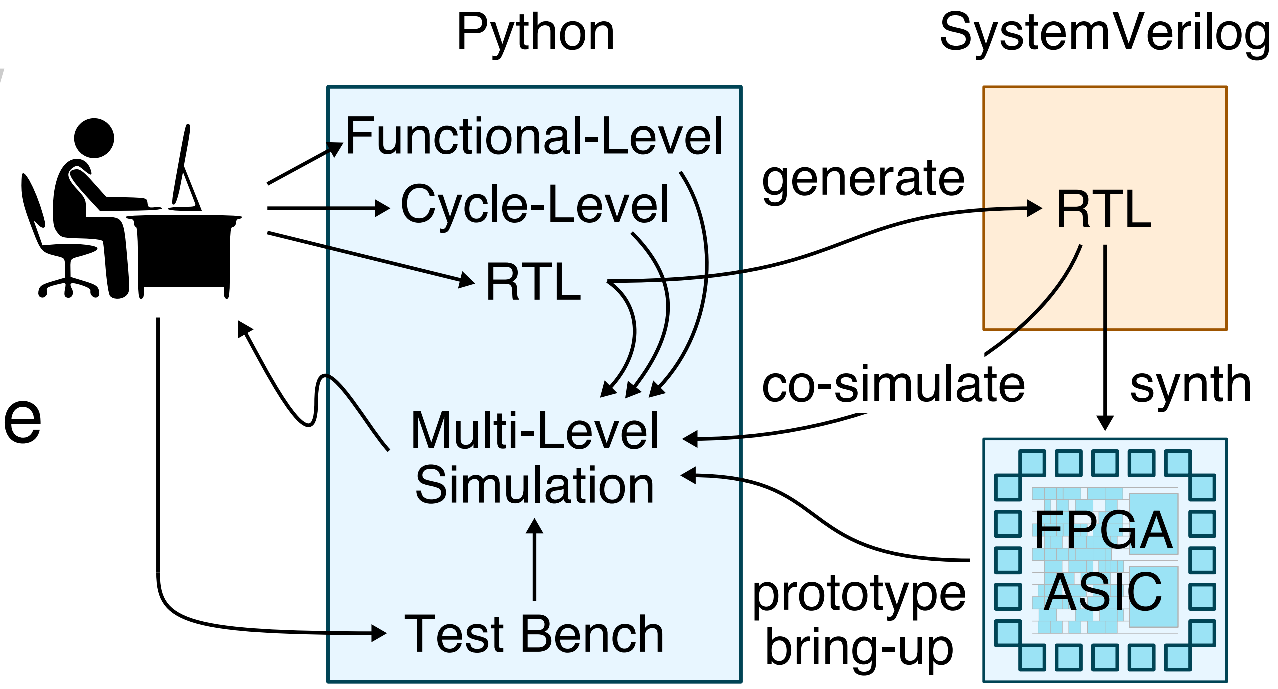
PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification



```
% python3 -m venv pymtl3
% source pymtl3/bin/activate
% pip install pymtl3
% python

>>> from pymtl3 import *

>>> a = Bits8(6)
>>> a
>>> b = Bits8(3)
>>> b
>>> a | b
>>> a << 4

>>> c = (a << 4) | b
>>> c
>>> c[4:8]
```

```
>>> from pymtl3.examples.ex00_quickstart \
      import FullAdder
>>> import inspect
>>> print(inspect.getsource(FullAdder))

>>> fa = FullAdder()
>>> fa.apply(
         DefaultPassGroup(textwave=True) )
>>> fa.sim_reset()

>>> fa.a   @= 0
>>> fa.b   @= 1
>>> fa.cin @= 0
>>> fa.sim_tick()

>>> fa.a   @= 1
>>> fa.b   @= 0
>>> fa.cin @= 1
>>> fa.sim_tick()

>>> fa.print_textwave()
```

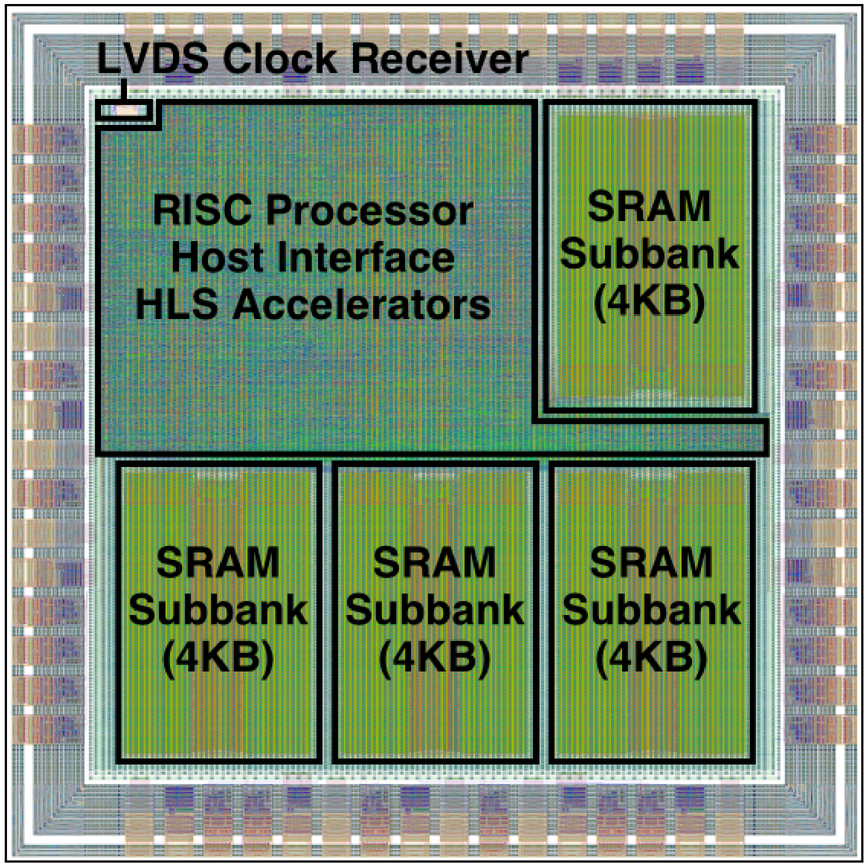
PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification

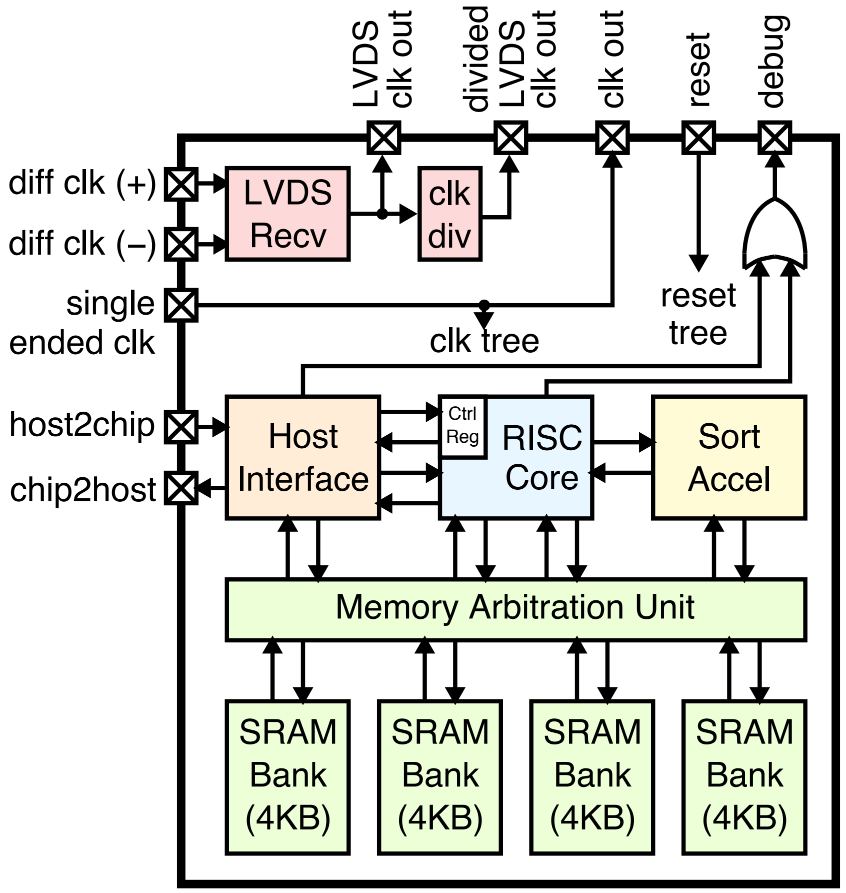


PyMTL2 in Architecture and EDA Research

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."**

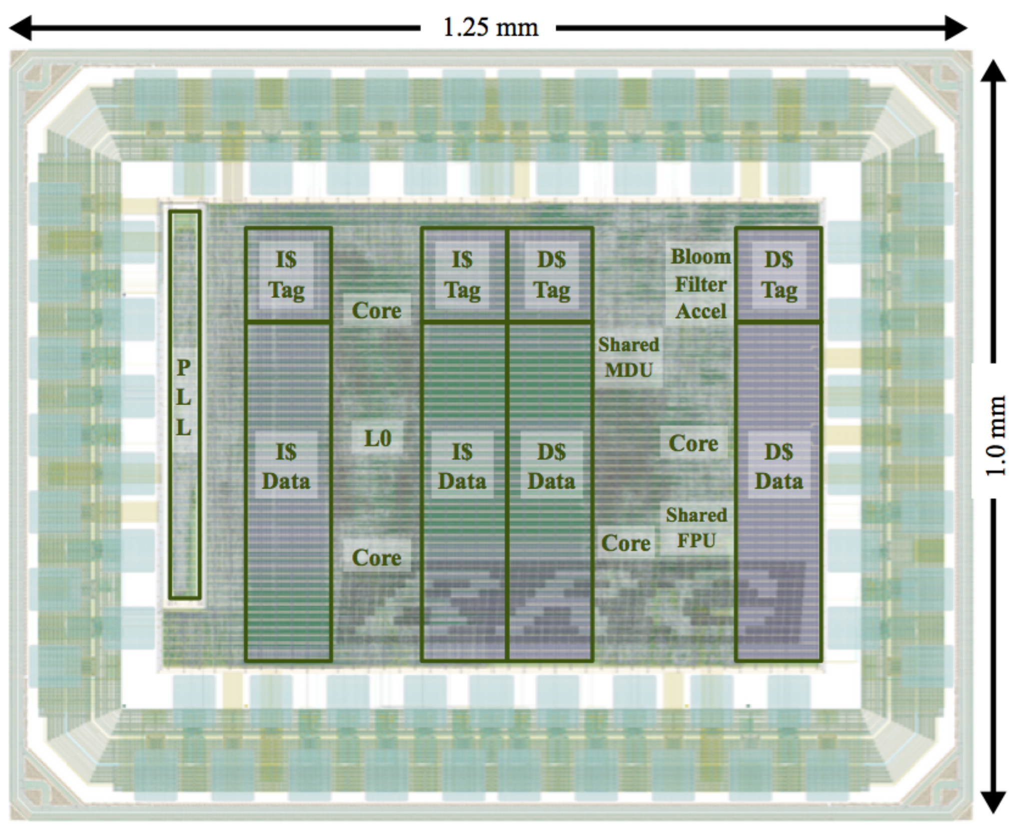
PyMTL2 ASIC Tapeout #1 (2016)

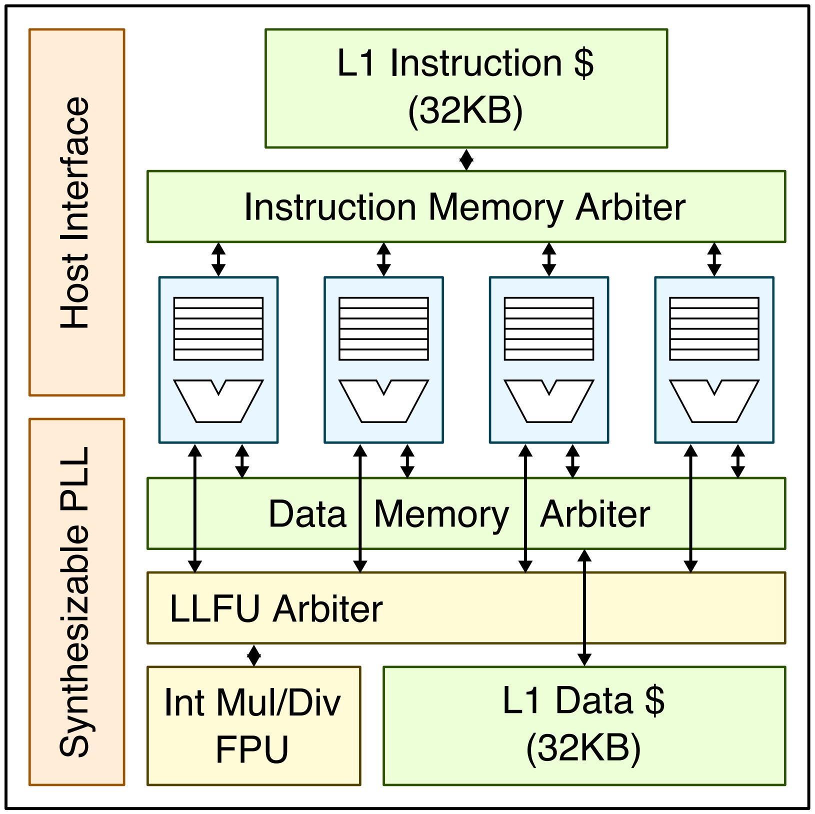




RISC processor, 16KB SRAM, HLS-generated accelerator
2x2mm, 1.2M-trans, IBM 130nm
95% done using PyMTL2

PyMTL2 ASIC Tapeout #2 (2018)

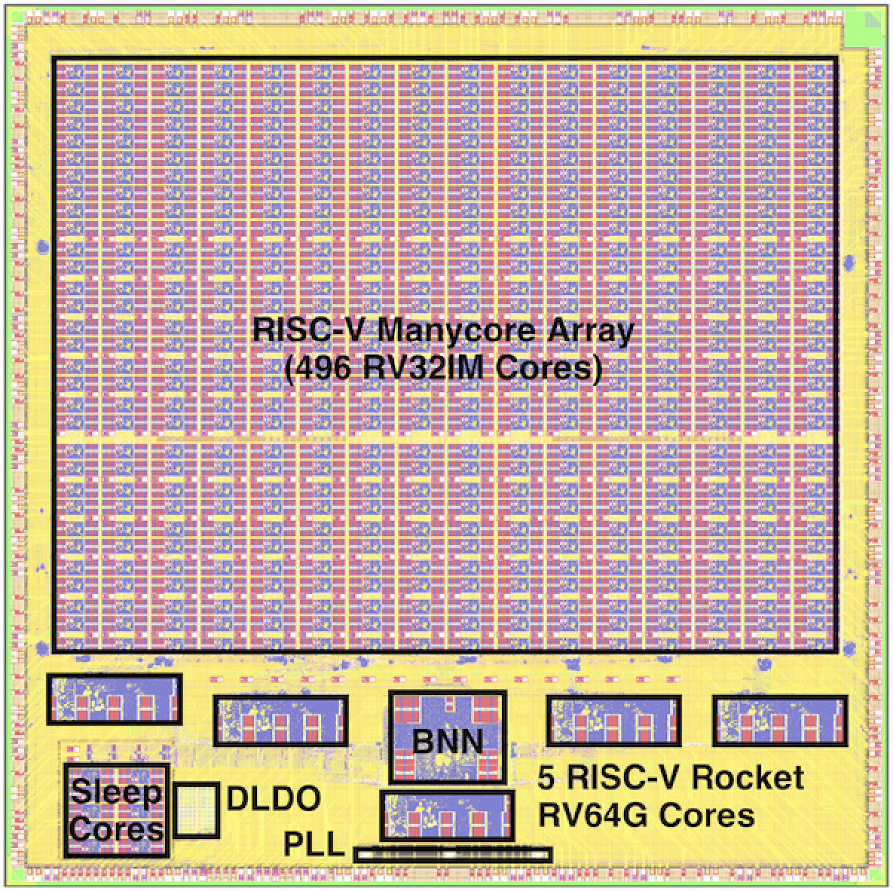




Four RISC-V RV32IMAF cores with “smart” sharing of L1$/LLFU
1x1.2mm, 6.7M-trans, TSMC 28nm
95% done using PyMTL2

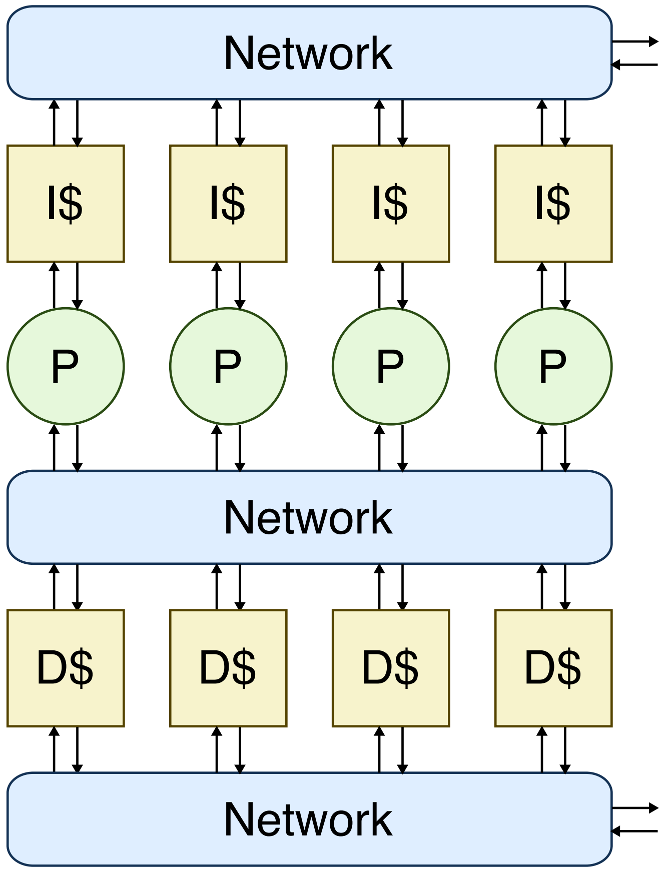
Celerity SoC through DARPA CRAFT Program

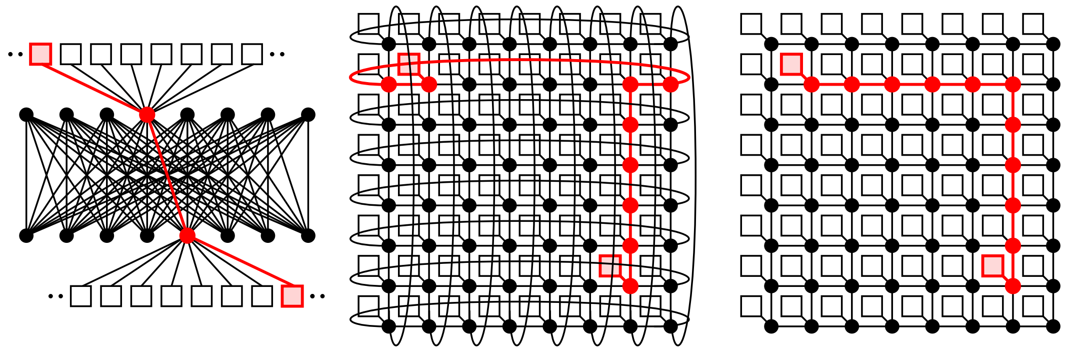
- 5×5mm in TSMC 16 nm FFC
- 385 million transistors
- 511 RISC-V cores
  - 5 Linux-capable Rocket cores
  - 496-core tiled manycore
  - 10-core low-voltage array
- 1 BNN accelerator
- 1 synthesizable PLL
- 1 synthesizable LDO Vreg
- 3 clock domains
- 672-pin flip chip BGA package



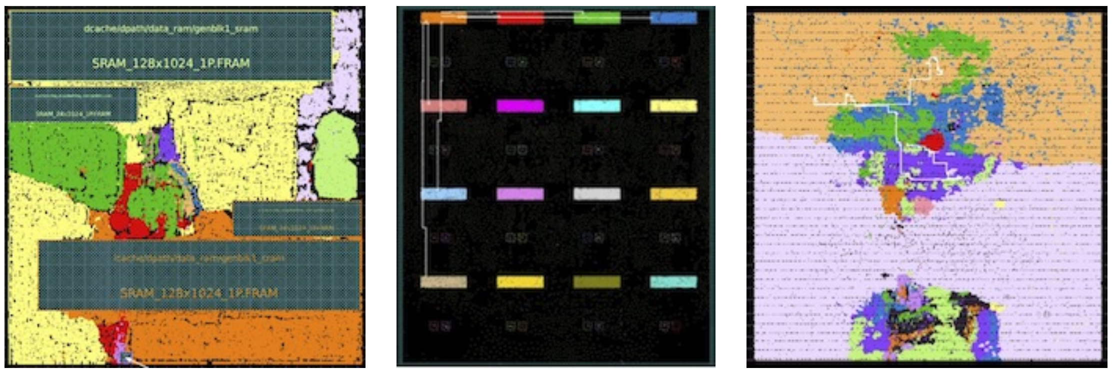
PyMTL2 played a small but important role in testing the BNN and automatically generating appropriate wrappers to interface with the Rocket core via RoCC

PyMTL2 in Teaching and POSH





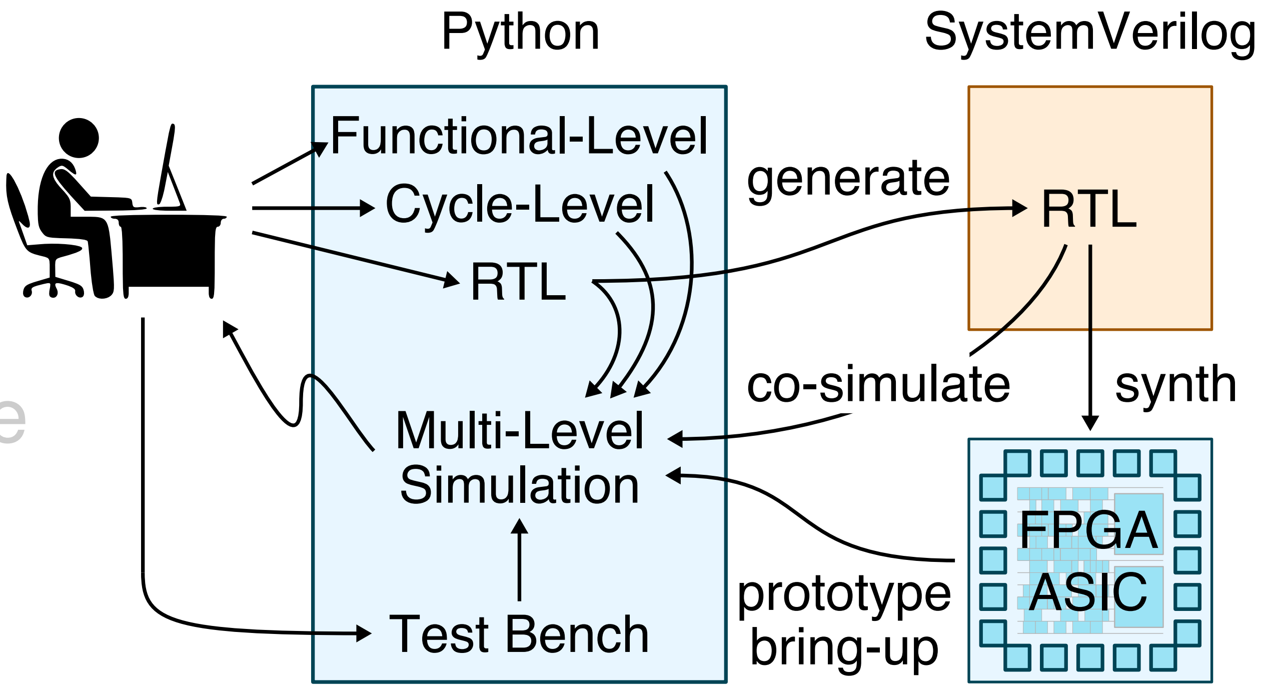
DARPA POSH Open-Source Hardware Program
PyMTL used as a powerful open-source generator for both design and verification



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

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



PyMTL + XILINX®

- **Python** for Packaging and Using Xcels
  - `pip install` your xcel of choice
  - Leverage PYNQ to use xcel

```
import pynq
import numpy as np

ol = pynq.Overlay('checksum.xclbin')
checksum = ol.checksum_1

a_buf    = pynq.allocate( (8,), 'u2', target=ol.bank0 )
a_buf[:] = np.array( [0, 9, 7, 8, 3, 7, 7, 5], dtype='u2' )
a_buf.sync_to_device()

checksum.call(a_buf)
a_buf.sync_from_device()
ol.free()
```

PyMTL + XILINX®

- **Python** for Packaging and Using Xcels
  - `pip install` your xcel of choice
  - Leverage PYNQ to use xcel
- **Python** for Developing HLS Xcels
  - Avoid C++ for HLS, active research area
  - `pip install` an existing HLS xcel
  - Start by modifying this existing HLS xcel
  - Potentially develop HLS xcel from scratch

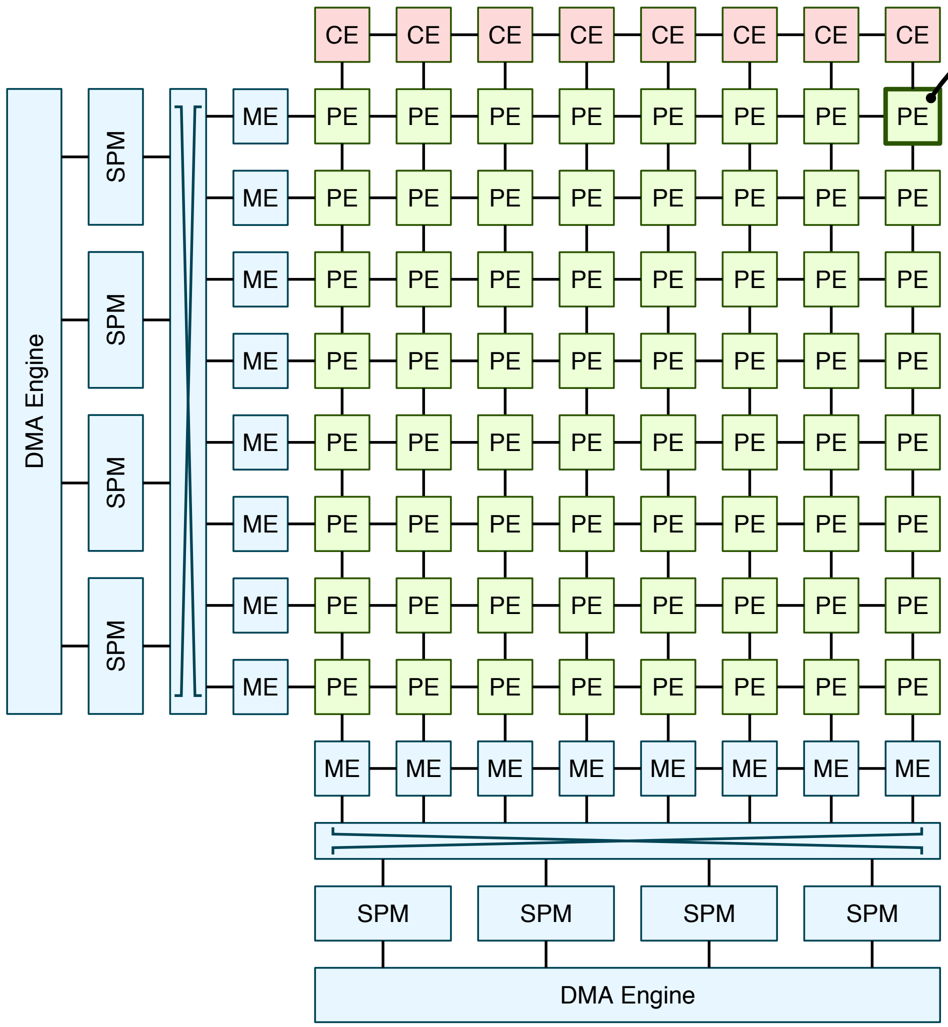
```
def bubble_sort( a ):
 n = len(a)
 for i in range(n-1):
  for j in range(0,n-i-1):
   if a[j] > a[j+1]:
    tmp    = a[j]
    a[j]   = a[j+1]
    a[j+1] = tmp
```

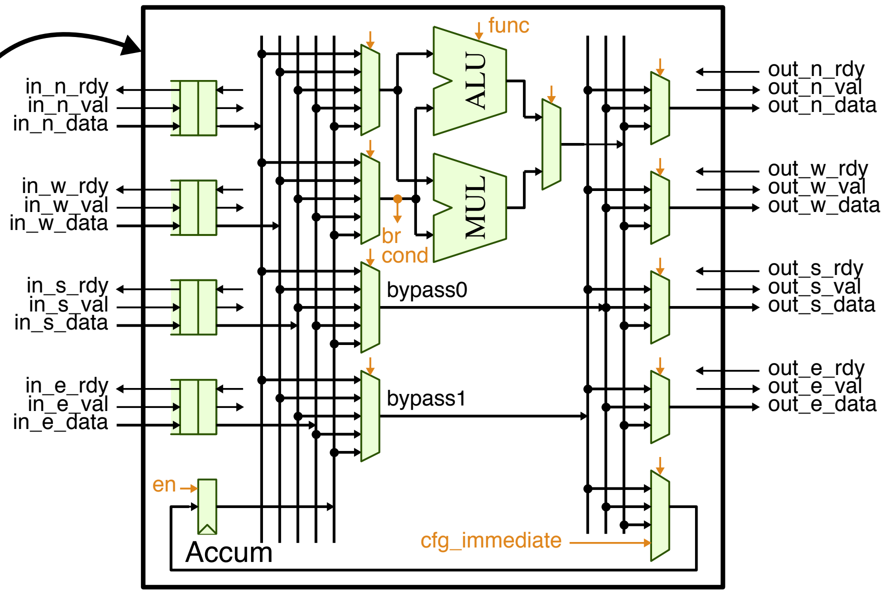
PyMTL + XILINX

- **Python** for Packaging and Using Xcels
  - `pip install` your xcel of choice
  - Leverage PYNQ to use xcel
- **Python** for Developing HLS Xcels
  - Avoid C++ for HLS, active research area
  - `pip install` an existing HLS xcel
  - Start by modifying this existing HLS xcel
  - Potentially develop HLS xcel from scratch
- **Python** for Developing RTL Xcels
  - Avoid Verilog for RTL, **use PyMTL!**
  - `pip install` an existing RTL xcel
  - Start by modifying this existing RTL xcel
  - Potentially develop RTL xcel from scratch

```
from pymtl3 import *

class Swap( Component ):
 def construct( s, B ):
  s.in0     = InPort (B)
  s.in1     = InPort (B)
  s.out_min = OutPort(B)
  s.out_max = OutPort(B)

  @update
  def block():
   if s.in0 >= s.in1:
    s.out_max @= s.in0
    s.out_min @= s.in1
   else:
    s.out_max @= s.in1
    s.out_min @= s.in0
```

PyMTL3 CGRA on Alveo Case Study



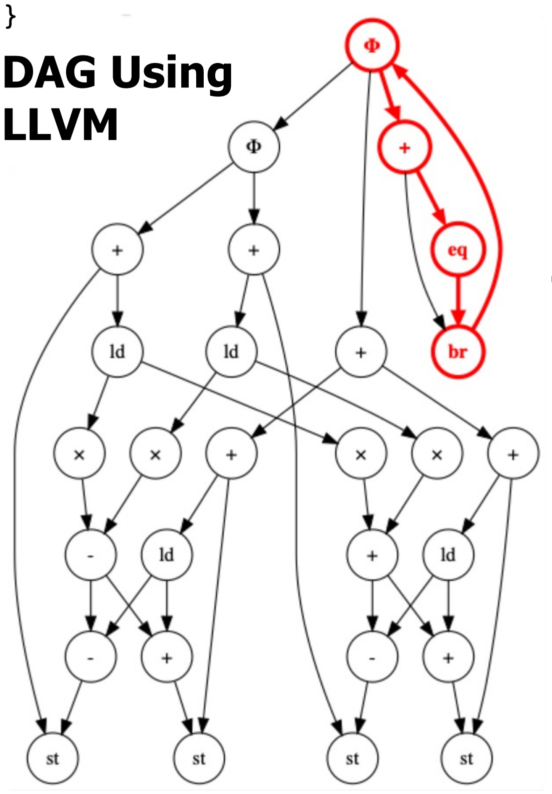


- Elastic latency-insensitive interfaces simplify compilation & MC integration
- 32-bit fxp/fp add, subtract, multiply, madd, accumulator
- copy0, copy1, sll, srl, and, or, xor, eq, ne, gt, geq, lt, leq
- phi and branch for control flow
- concurrent routing bypass paths

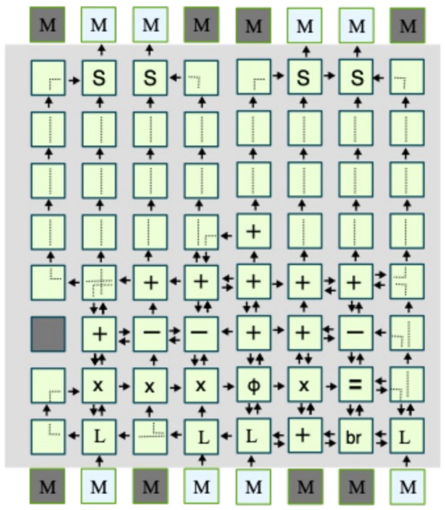
FFT Kernel

```
for ( int k = 0; k < G; ++k ) {
  t_r = Wr*r[2*j*G+G+k]
         - Wi*i[2*j*G+G+k];
  t_i = Wi*r[2*j*G+G+k]
         + Wr*i[2*j*G+G+k];
  r[2*j*G+G+k] = r[2*j*G+k]-t_r;
  r[2*j*G+k] += t_r;
  i[2*j*G+G+k] = i[2*j*G+k]-t_i;
  i[2*j*G+k] += t_i;
}
```

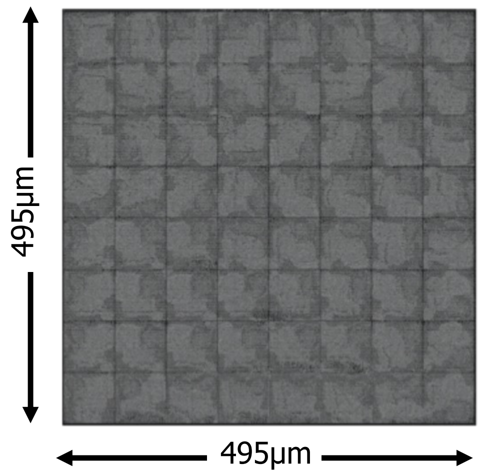
DAG Using LLVM



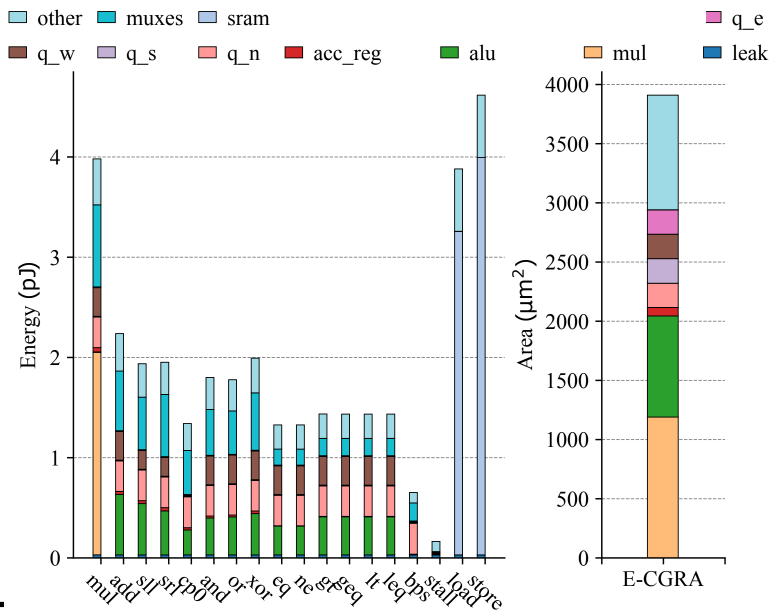
Schedule Using Custom LLVM Pass



TSMC 28nm Test Layout



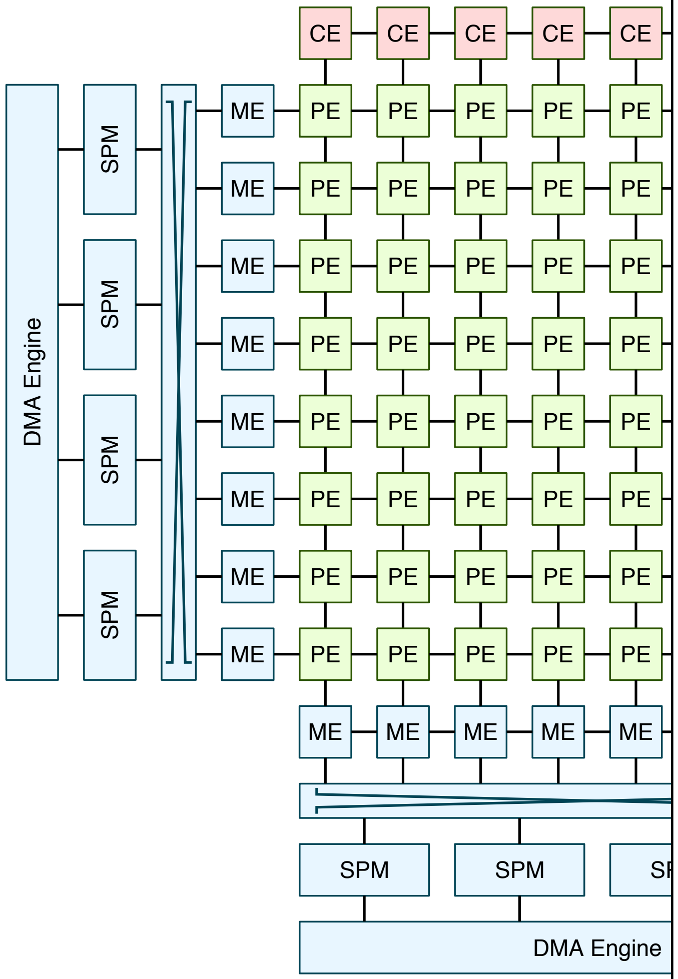
Gate-Level Energy & Area Analysis



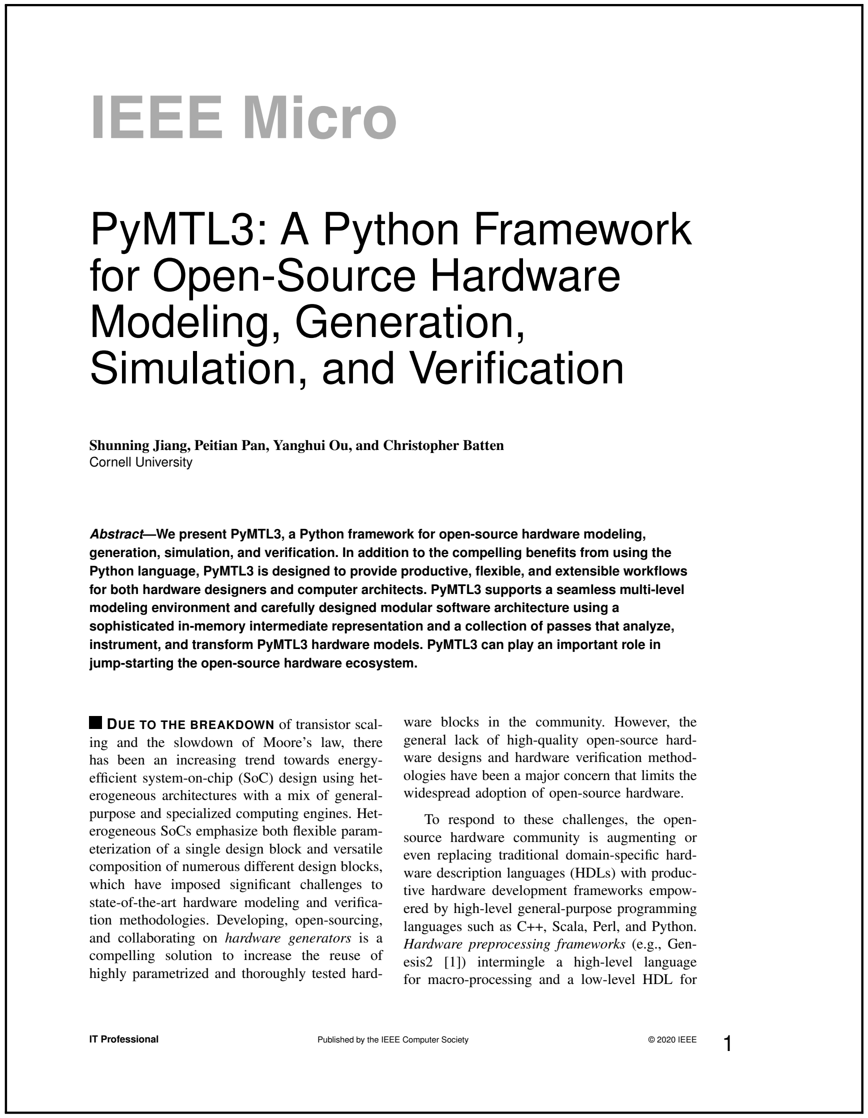
- LLVM compiler flow maps kernel to DAG and schedules on CGRA
- Energy and area evaluation using TSMC 28nm test layout
- **7.8x speedup on FFT vs. single RV32IM tile**
- **~5x energy efficiency improvement vs. single RV32IM tile**

PyMTL + XILINX®

- **Python** for Packaging and Using Xcels
  - `pip install` your xcel of choice
  - Leverage PYNQ to use xcel
- **Python** for Developing HLS Xcels
  - Avoid C++ for HLS, active research area
  - `pip install` an existing HLS xcel
  - Start by modifying this existing HLS xcel
  - Potentially develop HLS xcel from scratch
- **Python** for Developing RTL Xcels
  - Avoid Verilog for RTL, **use PyMTL!**
  - `pip install` an existing RTL xcel
  - Start by modifying this existing RTL xcel
  - Potentially develop RTL xcel from scratch



Two Upcoming PyMTL3 Publications



IEEE Micro

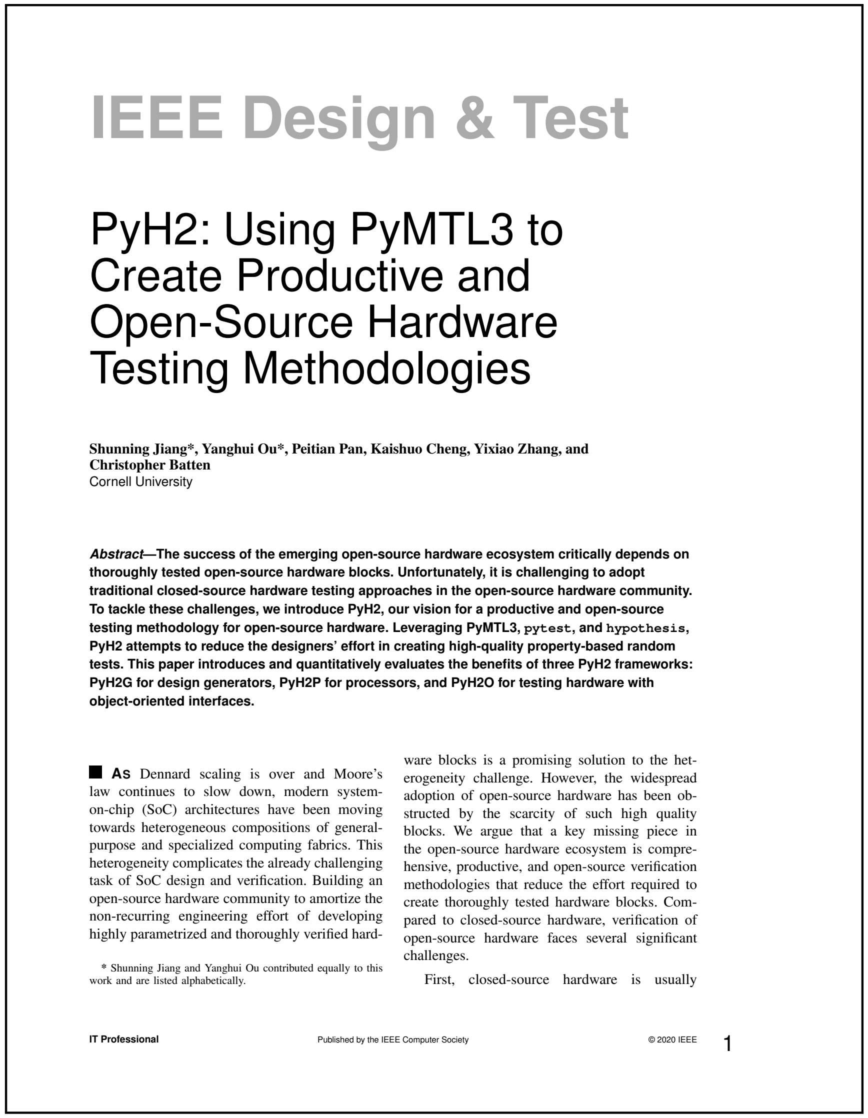
PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification

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

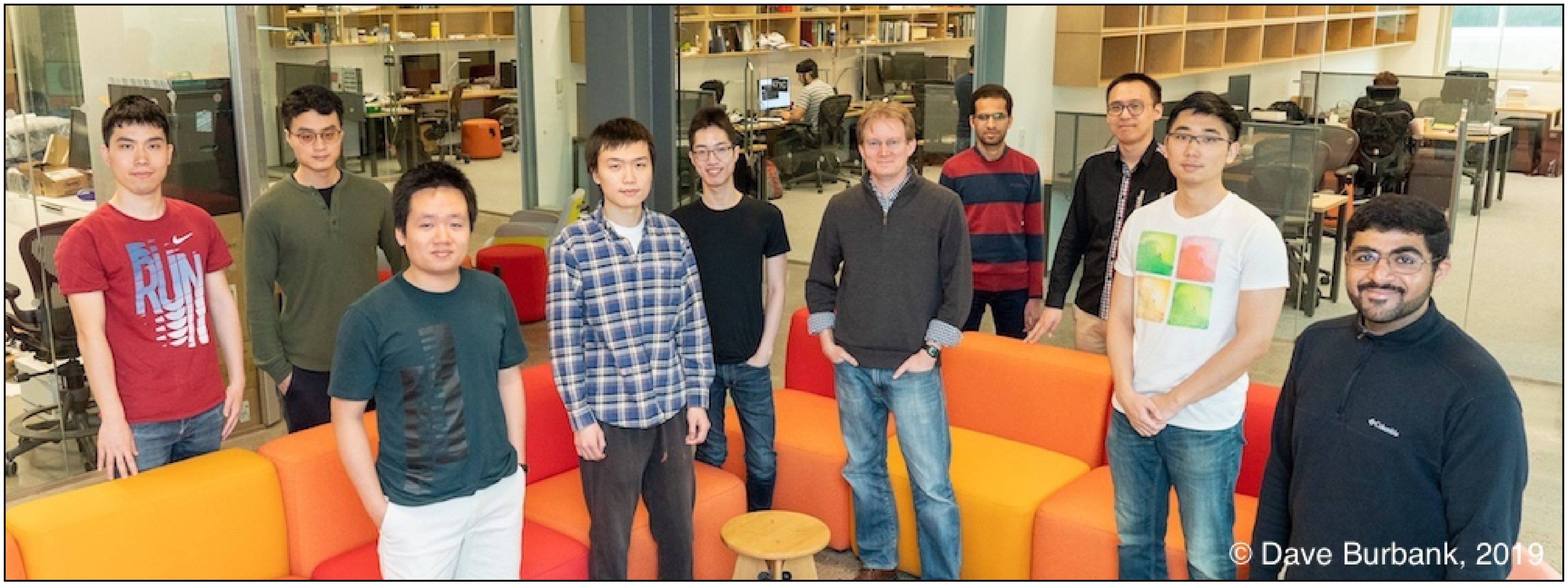
***Abstract*—The success of the emerging open-source hardware ecosystem critically depends on thoroughly tested open-source hardware blocks. Unfortunately, it is challenging to adopt traditional closed-source hardware testing approaches in the open-source hardware community. To tackle these challenges, 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

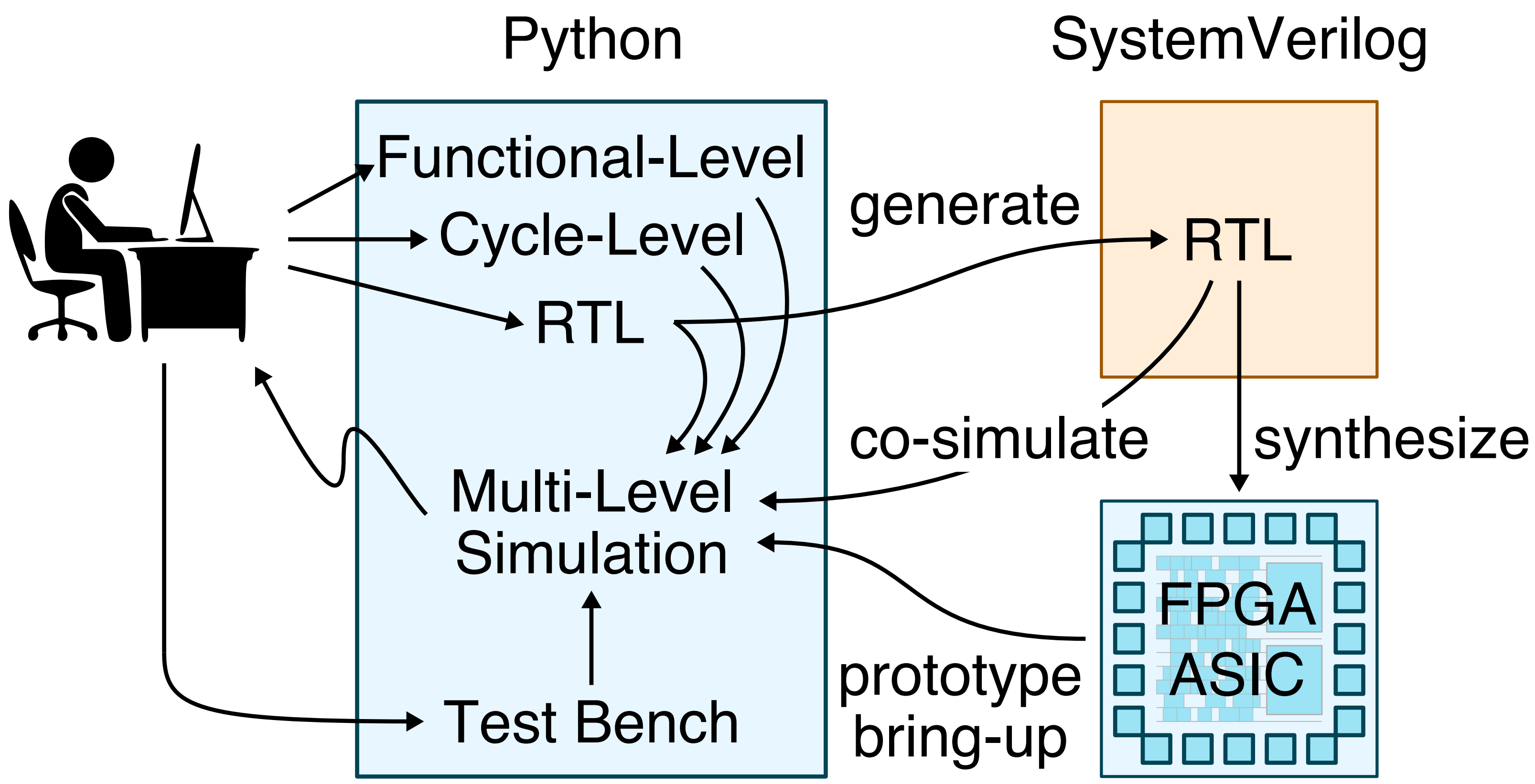


- **Shunning Jiang** : Lead researcher and developer for PyMTL3
- **Peitian Pan** : Leading work on translation & gradually-typed HDL
- **Yanghui Ou** : Leading work on property-based random testing

- Tuan Ta, Moyang Wang, Khalid Al-Hawaj, Shady Agwal, Lin Cheng

PyMTL

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



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