An Open-Source Python-Based Hardware Generation, Simulation, and Verification Framework

Shunning Jiang, Christopher Torng, Christopher Batten
Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University

1 Abstract

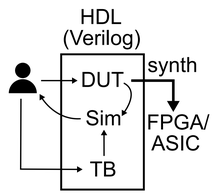
We present an overview of previously published features and work in progress for PyMTL, an open-source Python-based hardware generation, simulation, and verification framework that brings compelling productivity benefits to hardware design and verification. PyMTL provides a natural environment for multi-level modeling using method-based interfaces, features highly parametrized static elaboration and analysis/transform passes, supports fast simulation and property-based random testing in pure Python environment, and includes seamless SystemVerilog integration.

2 Hardware Development Workflows

Existing HDL, HPF, HGF, and HGSF Workflows

We categorize state-of-the-art hardware development workflows into four main groups with increasing productivity: **traditional HDLs, hardware preprocessing frameworks (HPF), hardware generation frameworks (HGF), and hardware generation and simulation frameworks (HGSF)** .

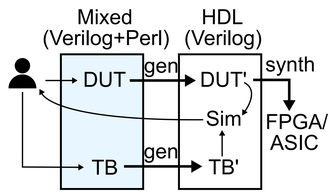
* DUT = design under test; DUT' = generated DUT; TB = test bench; TB* = TB with limited functionality; TB' = generated TB; Sim = simulation.



(a) Traditional

Other examples: VHDL

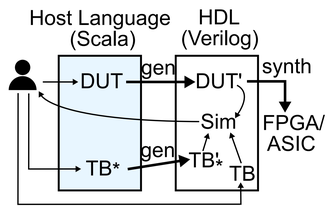
- ✓ Fast edit-debug-sim loop
- ✓ Single language for design and testbench
- ✗ Not-so-good parametrization
- ✗ Difficult to build powerful testbench



(b) HPF (e.g., Genesis2)

Other examples: Verischemelog

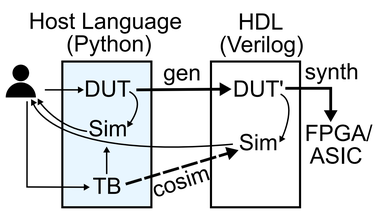
- ✓ Better parametrization with insignificant coding style change
- ✗ Multi-language semantic gap
- ✗ Still difficult to build powerful testbench



(c) HGF (e.g., Chisel)

Other examples: Lava, HML, Stratus, PHDL

- ✓ Powerful parametrization
- ✓ Single language for design
- ✗ Slower edit-debug-sim loop
- ✗ Yet still difficult to build powerful testbench (can only generate simple testbench)

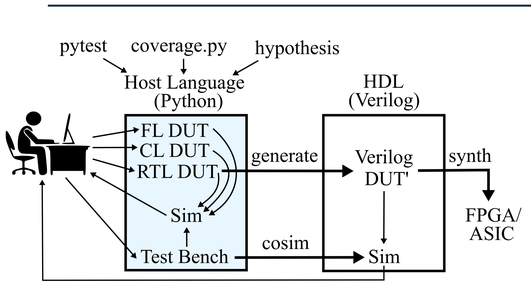


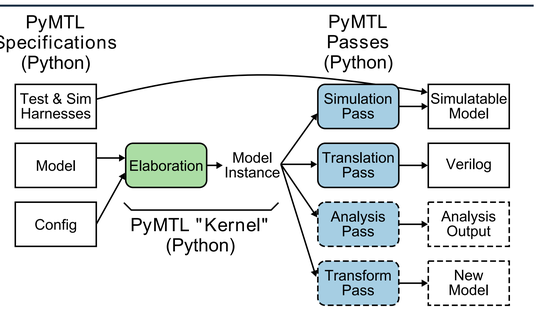
(d) HGSF (e.g., PyMTL)

- ✓ **Fast(?)** edit-debug-sim loop in host language
- ✓ Single language for design and testbench
- ✓ Powerful parametrization
- ✓ Powerful testbench (unleash Python's full power!)

Other examples: JHDL, CλaSH, PyMTL, PyRTL, Migen, PyHDL

PyMTL's HGSVF Workflow and Software Architecture





3 PyMTL's Eight Features

▷ Multi-level modeling

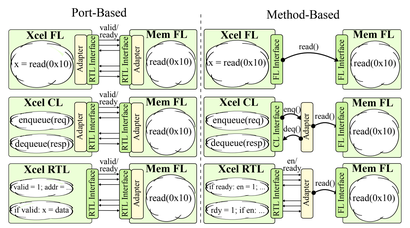
```
# FL implementation for calculating log2(N)
@s.tick_fl
def fl_algorithm():
  # put/get have blocking semantics
  s.out.put( math.log( s.in.get(), 2 ) )

# CL implementation emulates a 3-cycle pipeline
s.pipe = Pipeline( latency = 3 )
@s.tick_cl
def cl_algo_pipelined():
  if s.out_q.enq_ready():
    if s.pipe.can_pop(): s.out_q.push( s.pipe.do_pop() )
    else:                s.pipe.advance()
  if not s.in_q.deq_ready():
    s.pipe.do_push( math.log( s.in_q.deq(), 2 ) )

# Part of RTL implementation
s.N   = Reg( Bits32 )
s.res = RegEn( Bits32 )
s.connect( s.res.out, s.out.msg )
...
@s.combinational
def rtl_combN():
  s.res.in_  = s.res.out + 1
  s.N.in_    = s.N.out >> 1
  if s.N.out == 0:  s.res.en = Bits1( 0 )
  else:             s.res.en = Bits1( 1 )
```

▷ Pure-Python simulation

▷ Property-based random testing

▷ Highly Parametrized Static Elaboration

▷ Method-based interfaces



▷ Python/SystemVerilog integration

```
# By default PyMTL imports module DUT of DUT.v
# in the same folder as the python source file.
class DUT( VerilogModel ):
  def __init__( s ):
    s.in_ = InPort  ( Bits32 )
    s.out = OutPort ( Bits32 )

    # Connect top level ports of DUT
    # to corresponding PyMTL ports
    s.set_ports({
      'clk'   : s.clk,
      'reset' : s.reset,
      'in'    : s.in_,
      'out'   : s.out,
    })
```

▷ Analysis and Transform passes

```
# Analysis pass example:
# Get a list of processors with >=2 input ports
def count_pass( top ):
  ret = []
  for m in top.get_all_modules_filter(
    lambda m: len( m.get_input_ports() ) >= 2 ):
    if isinstance( m, AbstractProcessor ):
      ret.append( m )
  return m

# Transform pass example:
# Wrap every ctrl with CtrlWrapper
def debug_port_pass( top ):
  for m in top.get_all_modules():
    if m.get_full_name().startswith("ctrl"):
      p = m.get_parent()
      ctrl = p.delete_component( "ctrl" )
      w = p.add_component( "ctrl_wrap", CtrlWrapper() )
      new_ctrl = w.add_component( "ctrl", m )
      ...
      < connect ports >
      ...
```

▷ Fast Pure-Python Simulation

	PyMTL	MyHDL	PyRTL	Migen	IVerilog	CVS	Mamba
Divider	118K CPS	0.8×	2.2×	0.03×	0.6×	9.3×	20×
1-core	20K CPS	-	-	-	1×	15×	16×
32-core	360 CPS	-	-	-	1.8×	25×	12×

4 PyMTL Use Cases

Course Lab Assignments

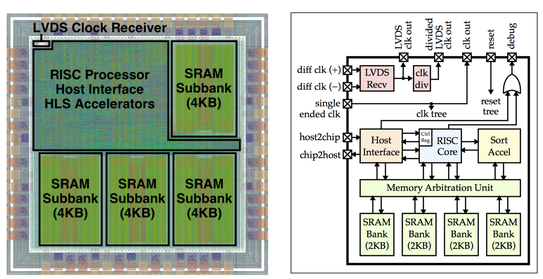
PyMTL has been used by over 400 students across two universities, including in a senior-level undergraduate computer architecture course at Cornell University(ECE 4750), in a similar course at Boston University (EC 513), and in a graduate-level ASIC design course at Cornell University (ECE 5745). The computer architecture courses involved multiple design labs (integer multiplier, simple RISC-V processor, set-associative blocking cache, and bus/ring network), culminating in a final lab composing all previous components to build a multi-core system. Students chose whether to design in PyMTL, in SystemVerilog, or with a mix, but they were required to test their designs using PyMTL.

Computer Architecture Research

Cosimulate PyMTL cycle-level accelerator model with gem5 CPU/memory:
Shreesha Srinath, Berkin Ilbeyi, Mingxing Tan, Gai Liu, Zhiru Zhang, and Christopher Batten. **"Architectural Specialization for Inter-Iteration Loop Dependence Patterns."** 47th ACM/IEEE Int'l Symp. on Microarchitecture (MICRO-47), Dec. 2014
Ji Kim, Shunning Jiang, Christopher Torng, Moyang Wang, Shreesha Srinath, Berkin Ilbeyi, Khalid Al-Hawaj, and Christopher Batten. **"Using Intra-Core Loop-Task Accelerators to Improve the Productivity and Performance of Task-Based Parallel Programs."** 50th ACM/IEEE Int'l Symp. on Microarchitecture (MICRO-50), Oct. 2017.

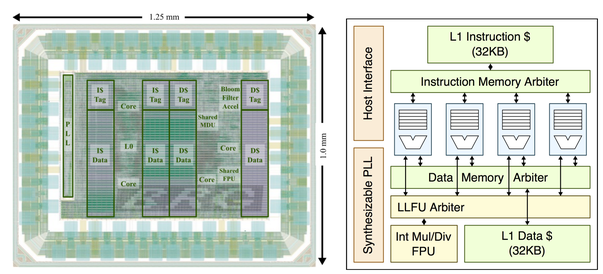
Create architecture templates of tuned accelerator with PyMTL RTL modeling:
Tao Chen, Shreesha Srinath, Christopher Batten, and Edward Suh. **"An Architectural Framework for Accelerating Dynamic Parallel Algorithms on Reconfigurable Hardware."** 51st ACM/IEEE Int'l Symp. on Microarchitecture (MICRO-51), Oct. 2018.
Tao Chen and Edward Suh. **"Efficient Data Supply for Hardware Accelerators with Prefetching and Access/Execute Decoupling."** 49th ACM/IEEE Int'l Symp. on Microarchitecture (MICRO-49), Oct. 2016.

Batten Research Group Test Chip 1 (2016)



▷ Fabricated in IBM 130nm; 2 x 2mm die; 1.2M transistor; RISC processor; LVDS clock receiver.

Batten Research Group Test Chip 2 (2018)



▷ Fabricated in TSMC 28nm; 1mm x 1.25mm die; 6.7M transistor; quad-core in-order RV32IMAF; smart sharing techniques for LLFUs and caches; synthesizable PLL.

This work was supported in part by NSF CRI Award #1512937, NSF SHF Award #1527065, DARPA POSH Award #FA8650-18-2-7852, and a donation from Intel. The authors acknowledge and thank Derek Lockhart for his valuable feedback and his work on the original PyMTL framework. The authors would like to thank Ajay Joshi for using PyMTL for the computer architecture course at Boston University. The author also thank all the students who have provided feedback to PyMTL. 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.

Publication: Appears in The First Workshop on Open-Source EDA Technology (WOSET 2018), Nov. 2018. **URL:** http://github.com/cornell-brg **Contact Author:** Shunning Jiang, 471 Rhodes Hall, Ithaca, NY 14853, sj634@cornell.edu