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

Applications

Algorithms

Compilers

Instruction Set Architecture

Microarchitecture

VLSI

Transistors

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

HDL (Verilog)
RTL
Sim
FPGA/ASIC
TB

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

HPF
Hardware Preprocessing Framework

Mixed (Verilog+Perl)
RTL
gen
Sim
FPGA/ASIC
TB

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

Example: Genesis2

HGF
Hardware Generation Framework

Host Language (Scala)
RTL
TB
gen
Sim
FPGA/ASIC

Slower edit-sim-debug loop
Cannot use power of host language for verification
Easier to create highly parameterized generators

Example: Chisel

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](https://github.com/cornell-brg/pymtl)
  - released in 2014
  - extensive experience using framework in research & teaching

- PyMTLv3: [https://github.com/cornell-brg/pymtl3](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

PyMTL Specifications (Python)

- Test & Sim Harnesses
- Model
- Config

PyMTL "Kernel" (Python)

- Elaboration

PyMTL Passes (Python)

- Simulation Pass
- Translation Pass
- Analysis Pass
- Transform Pass

- Simulatable Model
- Verilog
- Analysis Output
- New Model
PyMTLv3 High-Level Modeling

1 class QueueFL( Component ):
2     def construct( s, maxsize ):
3         s.q = deque( maxlen=maxsize )
4     @non_blocking(
5         lambda s: len(s.q) < s.q.maxlen )
6     def enq( s, value ):
7         s.q.appendleft( value )
8     @non-blocking(
9         lambda s: len(s.q) > 0 )
10     def deq( s ):
11         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         s.q1 = QueueFL(2)
19         s.q2 = QueueFL(2)
20         s.connect( s.enq, s.q1.enq )
21         s.connect( s.q2.deq, s.deq )
22     @s.update
23     def upA():
24         if s.q1.deq.rdy() and s.q2.enq.rdy():
25             s.q2.enq( s.q1.deq() )
from pymtl3 import *

class RegIncrRTL( Component ):
    def construct( s, dtype ):
        s.in_ = InPort( dtype )
        s.out = OutPort( dtype )
        s.tmp = Wire( dtype )

        @s.update_on_edge
        def seq_logic():
            s.tmp = s.in_

        @s.update
        def comb_logic():
            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’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
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

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

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

Batten Research Group

PyMTL Tutorial @ ISCA’19

16 / 115
## PyMTL Tutorial Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:45 – 2:00pm</td>
<td>Virtual Machine Installation and Setup</td>
</tr>
<tr>
<td>2:00 – 2:15pm</td>
<td><em>Presentation</em>: PyMTL Overview</td>
</tr>
<tr>
<td>2:15 – 2:55pm</td>
<td><em>Part 1</em>: PyMTL Basics</td>
</tr>
<tr>
<td>2:55 – 3:30pm</td>
<td><em>Part 2</em>: Multi-Level Modeling with PyMTL</td>
</tr>
<tr>
<td>3:30 – 4:00pm</td>
<td>Coffee Break</td>
</tr>
<tr>
<td>4:00 – 4:45pm</td>
<td><em>Part 3</em>: Processor Modeling with PyMTL</td>
</tr>
<tr>
<td>4:45 – 5:30pm</td>
<td><em>Part 4</em>: Multi-Level Composition in PyMTL</td>
</tr>
</tbody>
</table>