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

- HDL (Verilog)
- **RTL**, Sim, TB
- **FPGA/ASIC**
- **synth**
- **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, Sim, TB
- **FPGA/ASIC**
- **gen**, **synth**
- **Slower edit-sim-debug loop**
- **Multiple languages create "semantic gap"**
- **Easier to create highly parameterized generators**

**HGF**
Hardware Generation Framework

- Host Language (Scala)
- RTL, Sim, TB
- **FPGA/ASIC**
- **gen**, **synth**
- **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?
PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification

PyMTL

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

Python

Functional-Level
Cycle-Level
RTL
Multi-Level
Simulation
Test Bench

SystemVerilog

RTL

synthesize

generate

cosimulate

FPGA

ASIC

prototype
bring-up

Batten Research Group
CHIPKIT Tutorial @ ISCA'20
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 DSL (Python)**

- Test & Sim Harnesses
- Model
- Config

**PyMTL3 In-Memory Intermediate Representation (Python)**

- Elaboration

**PyMTL3 Passes (Python)**

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

**Simulatable Model**

- Verilog
- Analysis Output
- New Model
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()

FL/CL components can use method-based interfaces

Structural composition via connecting methods

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() )
```

Batten Research Group
CHIPKIT Tutorial @ ISCA'20
```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
```

- 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’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
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

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
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.

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

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**

© Dave Burbank, 2019
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 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 the Defense Advanced Research Projects Agency through a DARPA POSH Award #FA8650-18-2-7852.

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 Maclver, 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 thereon. 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.