A New Era of Open-Source Hardware

Christopher Batten

Computer Systems Laboratory
Electrical and Computer Engineering
Cornell University

On Sabbatical as a Visiting Scholar
SLICE Laboratory
University of California, Berkeley
Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

A Call to Action


Transistors (Thousands)


Single-Core

Power (W)

Frequency (MHz)

SPECint (single-core)

SPECrate (4-7 cores)

Parallelization & Specialization

Number of Cores

Number of Accelerators

Number of Accelerators

Pipelining & Caches

Superscalar Execution

Superscalar Out-of-Order Execution

Aggressive Superscalar Out-of-Order Execution


Cornell University

Christopher Batten
Top-five software companies are all building custom accelerators

- **Facebook**: MTIA for DLRM
- **Amazon**: Echo, Oculus, Inherentivia/Trainium
- **Microsoft**: In-house AI chips
- **Google**: TPU, Pixel, convergence
- **Apple**: SoCs for phones and laptops

Chip startup ecosystem for machine learning accelerators is thriving!

How can we accelerating innovation in accelerator-centric hardware design?

- Graphcore
- Nervana
- Cerebras
- Wave Computing
- Horizon Robotics
- Cambricon
- DeePhi
- Esperanto
- SambaNova
- Eyeriss
- Tenstorrent
- Mythic
- ThinkForce
- Groq
- Lightmatter
Software Innovation Today

Like climbing an iceberg – much is hidden!

Your proprietary code
- Instagram
- $500K seed with 13 people → $1B

Open-source software
- Python
- Django
- Memcached
- Postgres/SQL
- Redis
- nginx
- Apache, Gnuicorn
- Linux
- GCC

"What Powers Instagram: Hundreds of Instances, Dozens of Technologies"
https://goo.gl/76fWrM

Adapted from M. Taylor, “Open Source HW in 2030,” Arch 2030 Workshop @ ISCA’16
Hardware Innovation Today

Like climbing a mountain – nothing is hidden!

What you have to build
- New machine learning accelerator
- Other unrelated components, anything you cannot afford to buy or for which COTS IP does not do

Closed source
- ARM A57, A7, M4, M0
- ARM on-chip interconnect
- Standard cells, I/O pads, DDR Phy
- SRAM memory compilers
- VCS, Modelsim
- DC, ICC, Formality, Primetime
- Stratus, Innovus, Voltus
- Calibre DRC/RCX/LVS, SPICE

Adapted from M. Taylor, “Open Source HW in 2030,” Arch 2030 Workshop @ ISCA’16
Trends in Open-Source HW

Chip Costs Are Skyrocketing

Adapted from M. Taylor, “Open Source HW in 2030,” Arch 2030 Workshop @ ISCA’16; original: International Business Strategies & T. Austin.
Minimum Viable Product/Prototype

Adapted from M. Taylor, “Open Source HW in 2030,” Arch 2030 Workshop @ ISCA’16; original: International Business Strategies & T. Austin.

Cost to Market ($ million)

- Mask Costs
- S/W Development and Testing
- H/W Design and Verification

Silicon Technology Node

- 0.5u
- 0.35u
- 0.25u
- 0.18u
- 0.13u
- 90nm
- 65nm
- 45nm
- 28nm
- 20nm

$500K for 4x Performance Penalty (post-Dennard scaling)

$5M per mask iteration

$120M
Minimum Viable Product/Prototype

Can we use open-source software/hardware to address remaining costs?

Adapted from M. Taylor, “Open Source HW in 2030,” Arch 2030 Workshop @ ISCA’16; original: International Business Strategies & T. Austin.
How can HW design be more like SW design?

<table>
<thead>
<tr>
<th>Open-Source</th>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>high-level languages</td>
<td>Python, Ruby, R, Javascript, Julia</td>
<td>Chisel, PyMTL, PyRTL, MyHDL, JHDL, Cλash, Calyx, DFiant</td>
</tr>
<tr>
<td>libraries</td>
<td>C++ STL, Python std libs</td>
<td>BaseJump</td>
</tr>
<tr>
<td>systems</td>
<td>Linux, Apache, MySQL, memcached</td>
<td>Rocket, Pulp/Ariane, OpenPiton, Boom, FabScalar, MIAOW, Nyuzi</td>
</tr>
<tr>
<td>standards</td>
<td>POSIX</td>
<td>RISC-V ISA, RoCC, TileLink</td>
</tr>
<tr>
<td>tools</td>
<td>GCC, LLVM, CPython, MRI, PyPy, V8</td>
<td>Icarus Verilog, Verilator, qflow, Yosys, TimberWolf, qrrouter, magic, klayout, ngspice</td>
</tr>
<tr>
<td>methodologies</td>
<td>agile software design</td>
<td>agile hardware design</td>
</tr>
<tr>
<td>cloud</td>
<td>IaaS, elastic computing</td>
<td>IaaS, elastic CAD</td>
</tr>
</tbody>
</table>
# Ubuntu Server 16.04 LTS (ami-43a15f3e)
% sudo apt-get update
% sudo apt-get -y install build-essential qflow
% mkdir qflow && cd qflow
% wget http://opencircuitdesign.com/qflow/example/map9v3.v
% qflow synthesize place route map9v3 # yosys, graywolf, qrouter
% wget http://opencircuitdesign.com/qflow/example/osu035_stdcells.gds2
% magic # design def/lef -> magic format
>>> lef read /usr/share/qflow/tech/osu035/osu035_stdcells.lef
>>> def read map9v3.def
>>> writeall force map9v3
% magic # stdcell gds -> magic format
>>> gds read osu035_stdcells.gds2
>>> writeall force
% magic map9v3
>>> gds write map9v3  # design + stdcells magic format -> gds
% sudo apt-get -y install libqt4-dev-bin libqt4-dev libz-dev
% wget http://www.klayout.org/downloads/source/klayout-0.24.9.tar.gz
% tar -xzvf klayout-0.24.9.tar.gz && cd klayout-0.24.9
% ./build.sh -noruby -nopython
% wget http://www.csl.cornell.edu/~cbatten/scmos.lyp
% ./bin.linux-64-gcc-release/klayout -l scmos.lyp ../map9v3.gds
“I’m doing a (free) operating system (just a hobby, won’t be big and professional like gnu) for 386(486) AT clones.”
— Linus Torvalds, 1991
Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

A Call to Action

RISC-V International

RISC-V: The Open era of computing

RISC-V Announces First New Specifications of 2022, Adding to 16
Ratified in 2021 | RISC-V International

RISC-V Community News | Announcements, What's New

Efficient Trace, Supervisor Binary Interface,
Unified Extensible Firmware Interface, and
Zmmul Multiply-Only Extension Accelerate
Embedded- and Large-System Design. Six
Additional Specifications Already In the
Pipeline As Development Extends Into
Vertical...

Check out our local language pages!

Get in touch!

Press: press@riscv.org

Analysts: analysts@riscv.org

General: info@riscv.org
Industry Interest in RISC-V is Growing
RISC-V Hardware and Software Ecosystem

Open-source software:
- Gcc, binutils, glibc, Linux, BSD, LLVM, QEMU, FreeRTOS, ZephyrOS, LiteOS, SylixOS, ...

Commercial software:
- Lauterbach, Segger, IAR, Micrium, ExpressLogic, Ashling, AntMicro, Imperas, UltraSoC, ...

Software

RISC-V Foundation
- ISA specification
- Golden Model
- Compliance

Hardware

Open-source cores:
- Rocket, BOOM, RISCY, Ariane, PicoRV32, Piccolo, SCR1, Shakti, Swerv, Hummingbird, ...

Commercial core providers:
- Andes, Bluespec, Cloudbear, Codasip, Cortus, C-Sky, InCore, Nuclei, SiFive, Syntacore, ...

Inhouse cores:
- Nvidia, +others
OpenROAD: The Future of Open-Source EDA

**Flow Setup**
- Flow parameters
- Library, techfile preparation
- Macro wrappers
- Dont_use list

**Logic Synthesis**
- Logic optimization
- Technology mapping
- Buffering, sizing

**OpenROAD v1.0**
- IO placement
- Mixed-size + macro placement
- Tapcell insertion
- PDN generation

- Global placement
- Placement-based optimization
- Detailed placement

- Clock tree synthesis
- CTS, hold, ERC repair
- Placement legalization

- Global routing
- Antenna check + repair
- Detailed routing

- Filler cell + BEOL fill insertion
- Merge wrapped macros
- Merge GDS

```
design.v + .lib + .lef + .sdc + parameters.cfg
```

```
result.def + ppa.rpt + drc.rpt + results.gds
```
Google Partners with SkyWater and Efabless to Enable Open Source Manufacturing of Custom ASICs

11/12/2020 | Press Releases

ANNOUNCEMENT

First Open Source PDK Enables Full Manufacturing Chain for Open Hardware
chipIgnite

Rapid IC Creation

Shuttle 2110C

$9750 per project

13 of 40 project slots reserved

Tapeout: November 26, 2021 23:59 PT
Deliver: April 01, 2022

10 mm² User Design Area

WCSP Package
C2S2: Cornell Custom Silicon Systems Project Team

Three-year student-led project team focused on designing, fabricating, and testing chips in SkyWater 130nm to implement a proof-of-concept system for a campus partner

- Open-Source VexRISCV microcontroller
- Open-Source OpenROAD chip flow
- Open PDK for SkyWater 130nm
- ChipIgnite w/ efabless

100+ applications → 30 team members

- Digital Subteam (2022/2023: FFT Xcel)
- Analog Subteam (2022/2023: OpAmp)
- Software Subteam
- System Architecture Subteam
- Project Management Subteam
Join the 2nd
AI Generated Open-Source Silicon Design Challenge

Ignite your creativity and be part of our Second AI-Driven Open-Source Silicon Design Challenge

Witness firsthand how Generative AI is transforming the chip design landscape, empowering you to
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

A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

JIT-Compiled Simulation [DAC’18]

Property-Based Testing [D&T’21]

Gradual Typing [MEMOCODE’23]

Future Research

A Call to Action
A New Era of Open-Source Hardware

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

A Call to Action

```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
```

Trends in Open-Source HW
Traditional Hardware Design Methodologies

HDL
Hardware Description Language

HDL (Verilog)
RTL
Sim
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
Sim
TB

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

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

Are HGFs the best we can do in terms of a productive hardware design methodology?
PyMTL

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

Python

SystemVerilog

Functional-Level

Cycle-Level

RTL

Multi-Level Simulation

Test Bench

generate

cosimulate

synthesize

RTL

FPGA

ASIC

prototype

bring-up

Roo

Cornell University

Christopher Batten
PyMTL

- **PyMTL2**: [https://github.com/cornell-brg/pymtl](https://github.com/cornell-brg/pymtl)
  - released in 2014
  - extensive experience using framework in research & teaching

- **PyMTL3**: [https://github.com/pymtl/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 DSL (Python)**
- Test & Sim Harnesses
- Model
- Config

**PyMTL3 In-Memory Intermediate Representation (Python)**
- Elaboration
- Model Instance

**PyMTL3 Passes (Python)**
- Simulation Pass
- Translation Pass
- Analysis Pass
- Transform Pass

Simulation Pass → Simulatable Model
Translation Pass → Verilog
Analysis Pass → Analysis Output
Transform Pass → New Model
**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

```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 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
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”
A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

A Call to Action

1 from pymtl3 import *
2
3 class RegIncrRTL( Component ):
4     def construct( s, nbits ):
5         s.in_ = InPort( nbits )
6         s.out = OutPort( nbits )
7         s.tmp = Wire( nbits )
8
9     @update_ff
10     def seq_logic():
11         s.tmp <<= s.in_
12
13     @update
14     def comb_logic():
15         s.out @= s.tmp + 1
PyMTL has been used in many chip tapeouts

TSMC 180nm, 28nm, 16nm; Sky 130nm
GF 130nm, 12nm; Intel 22FFL

> RISC-V microcontrollers
> Coarse-grain reconfigurable arrays
> Clustered manycore architectures
> Mesh on-chip networks
> Crossbar interconnects
BRG Test Chip #2 (2018)

Four RISC-V RV32IMAF cores with “smart” sharing of L1$/LLFU
1x1.2mm, 6.7M-trans, TSMC 28nm
95% done using PyMTL2
BRG Test Chip #5 (2022)

- RISC-V RV32IM micro-controller
- $2 \times 2.5$mm in TSMC 180nm
- 16KB of instruction SRAM, 16KB of data SRAM
- SPI interface for config, SPI master, GP I/O
- 100% done using PyMTL3
- PyMTL3-based post-silicon test enviroment

Collaboration with UC San Diego & University of Michigan

- 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
- PyMTL2 used for testing and integration of BNN accelerator

[HOTCHIPS’17, IEEE Micro’18, SCCL’18]
CIFER System-on-Chip (2021–2023)

Collaboration with Princeton University

- 4 × 4mm in GF 12 nm FinFET
- 456 million transistors
- 22 RISC-V cores
  - 4 RV64GC Ariane cores
  - 3 Tiny Core clusters each with 6 RV32IM cores
- Embedded FPGA
- Heterogeneous cache coherent memory system
- PyMTL3 used for tiny cores and on-chip mesh network

[CICC’23, SCCL’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
A New Era of Open-Source Hardware

### Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

A Call to Action

---

**PyMTL3 in Research**

- JIT-Compiled Simulation [DAC’18]
- Property-Based Testing [D&T’21]
- Gradual Typing [MEMOCODE’23]

**Future Research**

---

**A Call to Action**

---

```python
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
```
Evaluating HDLs, HGFs, and HGSFs

- Apple-to-apple comparison of simulator performance
- 64-bit radix-four integer iterative divider
- All implementations use same control/datapath split with the same level of detail

Modeling and simulation frameworks:
- Verilog: Commercial verilog simulator, Icarus, Verilator
- HGF: Chisel
- HGSFs: PyMTL, MyHDL, PyRTL, Migen
Productivity/Performance Gap

- Higher is better
- Log scale (gap is larger than it seems)
- Commercial Verilog simulator is $20 \times$ faster than Icarus
- Verilator requires C++ testbench, only works with synthesizable code, takes significant time to compile, but is $200 \times$ faster than Icarus
Productivity/Performance Gap

- Chisel (HGF) generates Verilog and uses Verilog simulator
Using CPython interpreter, Python-based HGSFs are much slower than commercial Verilog simulators; even slower than Icarus!
Using PyPy JIT compiler, Python-based HGSFs achieve $\approx 10 \times$ speedup, but still significantly slower than commercial Verilog simulator.
Productivity/Performance Gap

- Hybrid C/C++ co-simulation improves performance but:
  - only works for a synthesizable subset
  - may require designer to simultaneously work with C/C++ and Python
PyMTL3 achieves impressive simulation performance by co-optimizing the framework and JIT
## PyMTL3 Performance

<table>
<thead>
<tr>
<th>Technique</th>
<th>Divider</th>
<th>1-Core</th>
<th>16-core</th>
<th>32-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event-Driven</td>
<td>24K CPS</td>
<td>6.6K CPS</td>
<td>155 CPS</td>
<td>66 CPS</td>
</tr>
<tr>
<td><strong>JIT-Aware HGSF</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ Static Scheduling</td>
<td>13×</td>
<td>2.6×</td>
<td>1×</td>
<td>1.1×</td>
</tr>
<tr>
<td>+ Schedule Unrolling</td>
<td>16×</td>
<td>24×</td>
<td>0.4×</td>
<td>0.2×</td>
</tr>
<tr>
<td>+ Heuristic Toposort</td>
<td>18×</td>
<td>26×</td>
<td>0.5×</td>
<td>0.3×</td>
</tr>
<tr>
<td>+ Trace Breaking</td>
<td>19×</td>
<td>34×</td>
<td>2×</td>
<td>1.5×</td>
</tr>
<tr>
<td>+ Consolidation</td>
<td>27×</td>
<td>34×</td>
<td>47×</td>
<td>42×</td>
</tr>
<tr>
<td><strong>HGSF-Aware JIT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ RPython Constructs</td>
<td>96×</td>
<td>48×</td>
<td>62×</td>
<td>61×</td>
</tr>
<tr>
<td>+ Huge Loop Support</td>
<td>96×</td>
<td>49×</td>
<td>65×</td>
<td>67×</td>
</tr>
</tbody>
</table>

- RISC-V RV32IM five-stage pipelined cores
- Only models cores, no interconnect nor caches
A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

• PyMTL3 in Research •

A Call to Action

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

Cornell University
Christopher Batten
Testing HW Design Generators 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_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

... 

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)**
  - Search strategies, auto shrinking
  - Detects error quickly
  - Produces minimal failing test case
  - Increasingly state-of-the-art in software testing

```python
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)
```
PyH2 Creatively Adopts PBT for SW to Test HW

- PyH2 combines PyMTL3, a unified hardware modeling framework, with Hypothesis, a PBT framework for Python software and creates a property-based testing framework for hardware.

- PyH2 leverages PBT to explore not just the input values for an HW design but to also explore the parameter values used to configure an HW design generator.

<table>
<thead>
<tr>
<th></th>
<th>Complete Random Testing</th>
<th>Iterative Deepened Testing</th>
<th>PyH2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small number of test cases to find bug</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Small number transactions in bug trace</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simple transactions in bug trace</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simple design instance for bug trace</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
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

A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

JIT-Compiled Simulation [DAC’18]

Property-Based Testing [D&T’21]

Gradual Typing [MEMOCODE’23]

Future Research

A Call to Action
### Statically vs. Dynamically Typed HDLs

<table>
<thead>
<tr>
<th></th>
<th>Design Productivity</th>
<th>Testing Productivity</th>
<th>Simulation Performance</th>
<th>Static Correctness Guarantees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verilog/SystemVerilog</td>
<td>Low  🌒</td>
<td>Low  🌒</td>
<td>High  🌒</td>
<td>Low  🌒</td>
</tr>
<tr>
<td>Bluespec</td>
<td>Medium 🌒</td>
<td>Low  🌒</td>
<td>Medium 🌒</td>
<td>High 🌒</td>
</tr>
<tr>
<td>Clash/Chisel/SpinalHDL</td>
<td>Medium 🌒</td>
<td>Low  🌒</td>
<td>Medium 🌒</td>
<td>Medium 🌒</td>
</tr>
<tr>
<td>PyRTL/MyHDL/Migen/PyMTL</td>
<td>High ⚫</td>
<td>High ⚫</td>
<td>Low ⚫</td>
<td>None ⚫</td>
</tr>
<tr>
<td>PyMTL3</td>
<td>High ⚫</td>
<td>High ⚫</td>
<td>Medium 🌒</td>
<td>Low  🌒</td>
</tr>
</tbody>
</table>

Can we achieve the best of both statically and dynamically typed HDLs in a single unified framework?
Gradually Typed HDLs

Code in Reticulated Python, a Gradually Typed Dialect of Python

```
1 class Foo:
2    bar = 42
3 def g(x):
4    x.bar = 'hello world'
5 def f(x:Object({bar:Int}))\rightarrow Int:
6    g(x)
7    return x.bar
8 f(Foo())
```

Dynamic

```
1 T_W = TypeVar( "T_W", bound=Bits )
2
3 class RegIncrRTL( Component, Generic[T_W] ):

4 def construct( s, W: Type[T_W] ):
5    s.in_ = InPort( W )
6    s.out = OutPort( W )
7    s.tmp = Wire( W )
8
9 @update_ff
10 def seq_logic():
11    s.tmp <<= s.in_
12
13 @update
14 def comb_logic():
15    s.out @= s.tmp + 1
```

Static

Component Hierarchy in GT-HDL
Symbolic Elaboration

How can we prove matching bitwidths for the assignment in upblock in all possible instances?

Instead of using concrete bitwidths at elaboration time, we symbolically determine the bitwidths ahead of time.

We can translate generator properties into integer constraints suitable for use with an SMT solver.
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

A New Era of Open-Source Hardware

Trends in Open-Source HW
PyMTL3 Framework
PyMTL3 in Practice
PyMTL3 in Research

JIT-Compiled Simulation [DAC’18]
Property-Based Testing [D&T’21]
Gradual Typing [MEMOCODE’23]

Future Research

A Call to Action
PyMTL3 Design and Verification using LLMs

Python

Functional-Level

Cycle-Level

RTL

Multi-Level Simulation

Test Bench

SystemVerilog

generate

cosimulate

RTL

synth

FPGA

ASIC

prototype

bring-up

LLM
README.md

to take a more manual approach and prompt ChatGPT more precisely. For things like register files, it was easier because ChatGPT may have seen similar source code in its training.

Use a different method for testing your Verilog than iverilog. We decided to use PyMTL3 to test our design and it was very fruitful. Because ChatGPT is better at generating Python than Verilog, it became trivially easy to use GPT4 to generate test cases for our design. This allowed us to have a feedback loop of generating test cases alongside hardware to verify its functionality. This enabled us to generate a larger project in a shorter amount of time.

The management SoC does not support floating point instructions. We thought there was an FPU implemented on the Management Core, but it turns out that was not synthesized. If you want to use the floating-point portions of this project, you will need to add an additional C library to compile floating-point instructions in software.
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 (e.g., design in a productive language, modular, and extensible design), PyMTL3 offers a seamless multilevel modeling environment and a 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 heterogeneous computing. In recent years, many new-computing paradigms have been proposed to tackle this challenge. Domain-specific hardware accelerators (e.g., GPUs, FPGAs) are key to low-energy and high-performance computing. These accelerators are benefitting from a tight design of a single-stage back-end compiler, in contrast to conventional general-purpose microprocessors that make use of many-level front-end optimization stages to manufacture back-end metal and horizontal modeling and optimization.

PyMTL3 is a Python framework designed to provide flexibility, modularity, and extensibility to both hardware designers and computer architects. PyMTL3 supports both heterogeneous SoC designs, where various computing engines are tightly integrated and heterogeneous SoC designs, where various computing engines are flexibly parameterized at runtime.


PyMTL3 Publications


```python
from pymtl3 import *

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

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

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

A New Era of Open-Source Hardware

Trends in Open-Source HW

PyMTL3 Framework

PyMTL3 in Practice

PyMTL3 in Research

JIT-Compiled Simulation [DAC’18]

Property-Based Testing [D&T’21]

Gradual Typing [MEMOCODE’23]

Future Research

A Call to Action
A Call to Action

- Open-source hardware needs developers who
  - ... are idealistic
  - ... have lots of free time
  - ... will work for free

- Who might that be?
  **Students!**

- Academics have a practical and ethical motivation for using, developing, and promoting open-source EDA tools and open-source hardware designs

- Two key trends make this a particularly exciting time to contribute
  - Open-source verification methodologies and tools are a key challenge
  - Large-language models offer a new opportunity to address this challenge
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, DARPA SDH Award #FA8650-18-2-7863, a Linux University Program industry gift, and 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 the core PyMTL developers, Derek Lockhart, Shunning Jiang, Peitian Pan, Yanghui Ou, along with Khalid Al-Hawaj, Moyang Wang, Tuan Ta, Ji Kim, Shreesha Srinath, Berkin Ilbeyi, Dilan Lakhani, Jack Brzozowski, Kyle Infantino, Yixiao Zhang, Jacob Glueck, Aaron Wisner, Gary Zibrat, Christopher Torng, Cheng Tan, Raymond Yang, Kaishuo Cheng, Carl Friedrich Bolz, David MacIver, and Zac Hatfield-Dodds for their help designing, developing, testing, and using PyMTL.

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.