A New Era of Open-Source System-on-Chip Design

Christopher Batten

Computer Systems Laboratory
Electrical and Computer Engineering, Cornell University

On Sabbatical as a Visiting Scholar
Computer Laboratory, University of Cambridge
Motivating Trends in Computer Architecture

Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten
Hardware Specialization from Cloud to IoT

**Google TPU**
- Training is done using the TensorFlow C++ framework
- Training can take weeks
- Google TPU is custom chip to accelerate training and inference

**Movidius Myriad 2**
- Custom chip for ML on embedded IoT devices
- Specifically focused on vision processing
- 12 specialized vector VLIW processors
How can we accelerate innovation in accelerator-centric system-on-chip design?
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
Chip Costs Are Skyrocketing

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

Adapted from M. Taylor, “Open Source HW in 2030,” Arch 2030 Workshop @ ISCA’16; original: Gartner Group & 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.
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, C$lash</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, qrouter, 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)
```bash
% 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
A New Era of Open-Source SoC Design

The PyMTL Framework
- PyMTL Motivation
- PyMTL Version 2
- PyMTL Version 3 (Mamba)
- PyMTL & Open-Source Hardware

The Celerity SoC
- Celerity Architecture
- Celerity Case Study
- Celerity & Open-Source Hardware

A Call to Action

Derek Lockhart, Gary Zibrat, Christopher Batten
47th ACM/IEEE Int’l Symp. on Microarchitecture (MICRO)

Mamba: Closing the Performance Gap in Productive Hardware Development Frameworks

Shunning Jiang, Berkin Ilbeyi, Christopher Batten
55th ACM/IEEE Design Automation Conf. (DAC)
San Francisco, CA, June 2018
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

Applications
Algorithms
Compilers
Instruction Set Architecture
Microarchitecture
VLSI
Transistors
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
VLSI Design Methodologies

**HDL**

- **HDL (Verilog)**

**HPF**

- **Mixed (Verilog+Perl)**

**HGF**

- **Host Language (Scala)**

---

**Fast edit-sim-debug loop**

✓ Single language for structural, behavioral, + TB

✓ Difficult to create highly parameterized generators

✗ Slower edit-sim-debug loop

✗ Multiple languages create "semantic gap"

✗ Easier to create highly parameterized generators

✗ Cannot use power of host language for verification

Example: Genesis2

Example: Chisel
Productive Multi-Level Modeling and VLSI Design

Multi-Level Modeling
SystemC

HGSF
Hardware Generation and Simulation Framework

Host Language (Python)

DUT Sim

TB

cosim

HDL (Verilog)

DUT'

Sim

FPGA/ASIC

gen

synth

VLSI Design
Chisel

- Single framework for ML modeling & VLSI design
- Fast edit-sim-debug loop
- Single language for structural, behavioral, + TB
- Easy to create highly parameterized generators
- Use power of host language for verification

Cornell University
Christopher Batten
PyMTL is a Python-based hardware generation and simulation framework for SoC design which enables productive multi-level modeling and VLSI implementation.
The PyMTL Framework

**PyMTL Specifications (Python)**
- Test & Sim Harnesses
- Config
- Model

**PyMTL "Kernel" (Python)**

**PyMTL Elaboration**
- Model Instance

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

**PyMTL Passes Outputs**
- Simulatable Model
- Verilog
- Analysis Output
- New Model
PyMTL v2 Syntax and Semantics

```python
from pymtl import *

class RegIncrRTL( Model ):

    def __init__( s, dtype ):
        s.in_ = InPort( dtype )
        s.out = OutPort( dtype )
        s.tmp = Wire( dtype )

    @s.tick_rtl
    def seq_logic():
        s.tmp.next = s.in_

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

- Class `RegIncrRTL` defines a simple register incrementer.
- `__init__` initializes the inputs and outputs.
- `seq_logic` updates the temporary variable `s.tmp`.
- `comb_logic` sets the output `s.out` to be the incremented value of `s.tmp`.

Diagram:
- Input `s.in_` is connected to the input of a register.
- The register has an output `s.tmp`.
- The incremented value of `s.tmp` is connected to the output `s.out`.

```python
class GcdUnitFL(Model):
    def __init__(s):

        # Interface
        s.req = InValRdyBundle(GcdUnitReqMsg())
        s.resp = OutValRdyBundle(Bits(16))

        # Adapters (e.g., TLM Transactors)
        s.req_q = InValRdyQueueAdapter(s.req)
        s.resp_q = OutValRdyQueueAdapter(s.resp)

        # Concurrent block
        @s.tick_fl
        def block():
            req_msg = s.req_q.popleft()
            result = gcd(req_msg.a, req_msg.b)
            s.resp_q.append(result)
```

```
# Interface
s.req = InValRdyBundle(GcdUnitReqMsg())

# Adapters (e.g., TLM Transactors)

# Concurrent block
@s.tick_fl
def block():
    req_msg = s.req_q.popleft()
    result = gcd(req_msg.a, req_msg.b)
    s.resp_q.append(result)
```
Performance/Productivity Gap

Python is growing in popularity in many domains of scientific and high-performance computing. How do they close this gap?

- Python-Wrapped C/C++ Libraries
  (NumPy, CVXOPT, NLPy, pythonoCC, gem5)

- Numerical Just-In-Time Compilers
  (Numba, Parakeet)

- Just-In-Time Compiled Interpreters
  (PyPy, Pyston)

- Selective Embedded Just-In-Time Specialization
  (SEJITS)
PyMTL Hybrid Python/C++ Co-Simulation

SimJIT-RTL Tool

Verilog Source \rightarrow Verilator \rightarrow RTL C++ Source \rightarrow Translation Cache

Translation \rightarrow C Interface Source \rightarrow LLVM/GCC \rightarrow C Shared Library

PyMTL RTL Model Instance \rightarrow PyMTL CFFI Model Instance

Cornell University
Christopher Batten
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
  - HGSF: PyMTL, MyHDL, PyRTL, Migen
Productivity/Performance Gap

- Higher is better
- Log scale (gap is larger than it seems)
- Commercial Verilog simulator is 20× faster than Icarus
- Verilator requires C++ testbench, only works with synthesizable code, takes significant time to compile, but is 200× 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.
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
Mamba is $20 \times$ faster than PyMTLv2, $4.5 \times$ faster that PyMTLv2 with hybrid co-simulation, comparable to commercial simulators

Mamba uses careful co-optimization of the framework and the JIT
**PyMTL ASIC Tapeouts**

**BRGTC1 in 2016**
RISC processor, 16KB SRAM
HLS-generated accelerator
2x2mm, 1.2M-trans, IBM 130nm

**BRGTC2 in 2018**
4xRV32IMAF cores with “smart” sharing L1$/LLFU, PLL
1x1.2mm, ≈10M-trans, TSMC 28nm
PyMTL and Open-Source Hardware

State-of-the-art in open-source HDL simulators
- *Icarus Verilog*: Verilog interpreter-based simulator
- *Verilator*: Verilog AOT-compiled simulator
- *GHDL*: VHDL AOT-compiled simulator
- No open-source simulator supports modern verification environments

PyMTL as an open-source design, simulation, verification environment
- Open-source hardware developers can use Verilog RTL for design and Python, a well-known general-purpose language, for verification
- PyMTL for FL design enables creating high-level golden reference models
- PyMTL for RTL design enables creating highly parameterized hardware components which is critical for encouraging reuse in an open-source ecosystem
PyMTL and Open-Source Hardware

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
PyMTL Take-Away Points

- PyMTL is a productive Python-based hardware generation and simulation framework to enable productive multi-level modeling and VLSI design.
- PyMTL v3 helps close the HGSF performance/productivity gap.
- PyMTL is an example of emerging frameworks that can potentially transform open-source hardware design.
- Beta version of PyMTL v2 is available for researchers to experiment with: https://github.com/cornell-brg/pymtl
```python
from pymtl import *

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

        @s.tick_rtl
        def seq_logic():
            s.tmp.next = s.in_

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

A New Era of Open-Source SoC Design

- The PyMTL Framework
  - PyMTL Motivation
  - PyMTL Version 2
  - PyMTL Version 3 (Mamba)
  - PyMTL & Open-Source Hardware
- The Celerity SoC
  - Celerity Architecture
  - Celerity Case Study
  - Celerity & Open-Source Hardware
- A Call to Action
The Celerity Open-Source 511-Core RISC-V
Tiered Accelerator Fabric:
Fast Architectures & Design Methodologies for Fast Chips

Scott Davidson, Shaolin Xie, Christopher Torng, Khalid Al-Hawaj
Austin Rovinski, Tutu Ajayi, Luis Vega, Chun Zhao, Ritchie Zhao
Steve Dai, Aporva Amarnath, Bandhav Veluri, Paul Gao, Anuj Rao
Gai Liu, Rajesh K. Gupta, Zhiru Zhang, Ronald G. Dreslinski
Christopher Batten, Michael B. Taylor.

IEEE Micro, 38(2):3041, Mar/Apr. 2018
Celerity System-on-Chip Overview

Target Workload: High-Performance Embedded Computing

- 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
- 9-months from PDK access to tape-out
Tiered Accelerator Fabrics

Flexibility
- General-purpose computation
- Operating systems, I/O
- Flexible and energy-efficient
- Exploits coarse- and fine-grain parallelism
- Fixed-function
- Extremely energy efficient

Efficiency

General-Purpose Tier
Massively Parallel Tier
Specialization Tier
Celerity: General-Purpose Tier

- Role of the General-Purpose Tier
  - General-purpose computation
  - Exception handling
  - Operating system (e.g., TCP/IP)
  - Cache memory hierarchy for all tiers

- In Celerity
  - 5 Rocket cores from UC Berkeley
  - Generated from Chisel
  - RV64G ISA
  - 5-stage, in-order, scalar processor
  - Double-precision floating point
  - I-Cache: 16KB 4-way assoc.
  - D-Cache: 16KB 4-way assoc.
  - 0.97 mm² per core @ 625 MHz
Celerity: Massively Parallel Tier
Celerity: Massively Parallel Tier

- Role of the Massively Parallel Tier
  - Improve energy efficiency over general-purpose tier by exploiting massive parallelism

- 496 low-power RISC-V Vanilla-5 cores
  - RV32IM ISA
  - 5-stage, in-order, scalar cores
  - 4KB instruction memory per tile
  - 4KB data memory per tile
  - 0.024 mm² per tile @ 1.05 GHz

- 16 × 31 tiled mesh array
  - MIMD programming model
  - XY-dimension network-on-chip (NoC)
  - 32 b/cycle channels
  - Manycore I/O uses same network
Celerity: Specialization Tier

Role of Specialization Tier
Ultra-high-energy efficiency for critical applications
Case Study: Mapping Flexible Image Recognition to a Tiered Accelerator Fabric

- **Step 1:** Implement the algorithm using the general-purpose tier
- **Step 2:** Accelerate algo using either massively parallel tier OR specialization tier
- **Step 3:** Improve performance by cooperatively using both the specialization AND the massively parallel tier
Step 1: Algo to App – Binarized Neural Networks

- Training usually uses floating point, while inference usually uses lower precision weights and activations (often 8-bit or lower) to reduce implementation complexity.

- Recent work has shown single-bit precision weights and activations can achieve an accuracy of 89.8% on CIFAR-10.

- Performance target requires ultra-low latency (batch size of one) and high throughput (60 classifications/second).

---

Step 1: Algo to App – Characterizing BNN Execution

- Using just the general-purpose tier is $200 \times$ slower than performance target of 60 classifications/second
- Binarized convolutional layers consume over 97% of the dynamic instruction count
- Perfect acceleration of just the binarized convolutional layers is still $5 \times$ slower than performance target
Step 2: App to Accel – BNN Specialized Accelerator

1. Accelerator is configured to process a layer through RoCC command messages.
2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers.
3. Binary convolution compute unit processes input fmaps and weights to produce output fmaps.
Step 2: App to Accel – Design Methodology

- Enabled quick implementation of an accelerator for an emerging algorithm
  - Algo to initial accelerator in weeks
  - Rapid design-space exploration

- Greatly simplified timing closure
  - Improved clock frequency by 43% in few days
  - Easily mitigated long paths at interfaces with latency insensitive design

- Tools are still evolving
  - Six weeks to debug tool bug with data-dependent access to multi-dimensional arrays
### Performance Benefits of Cooperatively Using the Specialization and Massively Parallel Tiers

<table>
<thead>
<tr>
<th></th>
<th>Step 1 GP Tier</th>
<th>Step 2 Spec Tier</th>
<th>Step 3 Spec+MP Tiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (ms)</td>
<td>4,024.0</td>
<td>20.0</td>
<td>3.2</td>
</tr>
<tr>
<td>Performance (images/sec)</td>
<td>0.3</td>
<td>50.0</td>
<td>312.5</td>
</tr>
<tr>
<td>Power (Watts)</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Efficiency (images/J)</td>
<td>2.5</td>
<td>250.0</td>
<td>625.0</td>
</tr>
<tr>
<td>Relative Efficiency</td>
<td>1×</td>
<td>100×</td>
<td>250×</td>
</tr>
</tbody>
</table>

- **GP Tier**: Software implementation assuming ideal performance estimated with an optimistic one instruction per cycle.

- **Spec/MP Tier**: Full-system post-place-and-route gate-level simulation of the spec/MP tiers running with a frequency of 625 MHz.
How were we able to build such a complex SoC?

- 5 × 5mm in TSMC 16 nm FFC
- 385 million transistors
- mixed-signal design
- front- and back-end design
- in nine months
- with 10 core graduate students
- with little tapeout experience
- across four locations

Open-source software and hardware was critical to the success of the project!
Leveraging the Open-Source RISC-V Ecosystem

- **RISC-V Software Toolchain**
  - Complete, off-the-shelf software stack for both GP and manycore

- **RISC-V Instruction Set Architecture**
  - Designed to be modular and extensible
  - Easy to connect to RoCC interface
  - Standard instruction verification suites

- **RISC-V Microarchitecture**
  - Rocket: high-performance RV64G core
  - Vanilla-5: high-efficiency RV32IM core
  - Standard on-chip network specs (NASTI)

- **RISC-V VLSI and System Design**
  - Previous spins of chips for reference
  - Turn-key FPGA gateware

Developed at UC Berkeley
http://riscv.org
Leveraging the Open-Source BaseJump Ecosystem

BaseJump STL: Standard library of hardware components

BaseJump Socket: IO Parding

BaseJump DoubleTrouble: HW Emulation Motherboard

BaseJump: Open FPGA Firmware

BaseJump RealTrouble: Bring-up Motherboard

BaseJump Socket: BGA Package

Developed at UCSD and University of Washington

http://bjump.org

Developed at UCSD and University of Washington
Contributing Back to the Open-Source Ecosystem

http://opencelerity.org

Upstream patches to gem5 for multi-core RISC-V simulation
A New Era of Open-Source SoC Design

The PyMTL Framework
- PyMTL Motivation
- PyMTL Version 2
- PyMTL Version 3 (Mamba)
- PyMTL & Open-Source Hardware

The Celerity SoC
- Celerity Architecture
- Celerity Case Study
- Celerity & Open-Source Hardware

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 electronic design automation tools and open-source hardware designs
Shreesha Srinath, Christopher Torng, Berkin Ilbeyi, Moyang Wang
Shunning Jiang, Khalid Al-Hawaj, Tuan Ta, Lin Cheng
and many M.S./B.S. students

Equipment, Tools, and IP
Intel, NVIDIA, Synopsys, Cadence, Xilinx, ARM