RISCV Verification – Opportunities and Challenges
Verification Futures Austin

Divyang Agrawal
Sr. Director

Sep 2023
Agenda

- Introduction and RISC-V Processor Family
- CPU Design and Verification
- RISCV
- OS Boot Case Study
Introduction
Tenstorrent builds computers for AI. Our mission is to address the open-source compute demands for software 2.0 through industry-leading AI/ML accelerators, high-performing RISC-V CPUs, and infinitely-configurable ML and CPU Chiplets.
Chip Roadmap

2021

ML Processor

2022

Networked ML Processor

2023

Standalone ML computer

2024

Low Power, Low Cost ML Chiplet

Highly Configurable, Highly Performant ML Chiplet Complex

Grayskull

12nm, 620mm²
315 8b TFLOPS
PCIE gen 4
8 channels LPDDR4

Wormhole

12nm, 650 mm²
350 8b TFLOPS
400GB/sec ethernet
6 channels GDDR6
16 lanes of PCIE gen 4

Black Hole

6nm, 600mm²
1000 8b TFLOPS
1200GB/sec ethernet
8 channels of GDDR6X
11TB/sec d2d interface

Quasar

4nm, 250mm²
650 8b TFLOPS
4 channels of GDDR6X
8 channels of GDDR6
32 lanes of PCIE Gen5

Grendel

3nm AI & CPU Chiplets
7nm Memory & IO Chiplets
Expandable TFLOPS
GDDR6 Channels
PCIE Lanes 128 RISC-V Cores
Non-blocking d2d interface
Problem Statement: CPU Design and Verification
Goal: RISC-V 0-o-0 Processor Family

Performance

Open & Free

4-Wide Decode
Sonic Boom with Vector

2-Wide Decode

3-Wide Decode

4-Wide Decode

6-Wide Decode

8-Wide Decode

Higher Performance

Decode Width

tenstorrent Confidential
Ascalon: High Performance 0-o-0 Superscalar Processor

RV64IACFDHMV

- Advanced branch predictions
- 8-wide decode
- 3 LD/ST with large load/store queues
- 6 ALU/2 BR
- 2 256-bit vector units
- 2 FPU units
CPU Design

• Design CPU is complex
  • High functional complexity
  • Meeting performance, power, and area goals even harder
  • Big team prone to communication errors

• Design flow innovation
  • Clean interfaces enforced by design modularity
  • Design abstraction compatibility
    • C++ == function model == RTL
  • Plug-and-play modules for co-simulations
Typical High Performance CPU Development Cycle

Identify target market
- Workload analysis
- Technology
- Engineering
- Resource planning

Perf, power, area bounding box
- Technology
- Engineering
- Resource planning

Building blocks
- Micro arch
- Design Meth
- DV Meth

Interfaces
- Micro arch
- Architectural
- SOC
- Platform

Development
- RTL coding
- Arch tools
- DV testbenches
- PD feasibility
- Platform planning

Convergence
- Rinse repeat
- Silicon planning

Silicon
- Post-si validation
- Customer sampling
- Design feedback
- Re-spin!

Product
- Customer support

36+ months

What does this mean for Verification?
Verification State Space

• Product / IP enablement (OS, application software, firmware)
• Post-silicon (performance, power, functionality)
• Design convergence
• Functional enablement
• Design bootstrap
• Specification
Lessons learned: DV starts with a Product and Silicon Mindset

- **Product / IP Enablement**
  - DV workloads
  - OS enablement
  - Parameterization
  - System level reference model

- **Post-silicon**
  - Debug visibility
  - Workaround capabilities
  - Stimulus
  - Perf and power characterization

- **Design Convergence**
  - Coverage and bug analysis
  - Diversity of stimulus
  - Formal and Emulation

- **Functional enablement**
  - Hierarchical testbenches
  - Stimulus controllability

- **Bootstrap**
  - Testbench architecture
  - Scalable HW DevOps
  - Reference model
Why RISC-V?

<table>
<thead>
<tr>
<th>Requirements</th>
<th>RISC-V</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-Source Architecture</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Addition of Custom Extensions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ISA extensions for high-perf</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ISA extensions for ML</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Toolchain and software ecosystem availability</td>
<td>Yes, significant work remains but Linux-like growth</td>
<td>Yes</td>
</tr>
<tr>
<td>Deployment</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Segment growth</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
RISCV Difference

• No legacy technical debt (x86: real mode, ARM: Aarch32)
• Base ISA + Extensions
  • Reference model is modular by design
  • Legacy stimulus baggage significantly reduced
  • Custom instructions, at will!
  • Custom data formats!
• How does this help?
  • Reduced cost of enduring 1st generation design decisions
  • Lends itself much more cleanly to Formal on the ISA
  • New extensions will see faster adoption
## RISCV Verification – Challenges and Opportunities

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Opportunity</th>
<th>Potential Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>General lack of silicon workloads</td>
<td>Conversion from x86 and ARM</td>
<td>Open source!</td>
</tr>
<tr>
<td>Limited open source or vendor stimulus generation tools</td>
<td>Need a portfolio of products targeting ISA, System Arch, Platform</td>
<td>Both open source and commercially available solutions</td>
</tr>
<tr>
<td>Large scale deployment of software stack</td>
<td>Trailblazers with large footprint</td>
<td>Open source!</td>
</tr>
<tr>
<td>Fractured architectural compatibility views</td>
<td>RVI + collaborative efforts</td>
<td></td>
</tr>
<tr>
<td>Application programmer’s guide</td>
<td></td>
<td>Significantly deficient options compared to x86 and ARM</td>
</tr>
<tr>
<td>Reference platforms, Devkits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OS Boot DV Case Study
Simulation OS Boot

- Problem Statement
  - Linux has ~66M instructions; will take a month to run on simulation
  - Onion peeling the issues - not feasible

- Approach
  - Pass #1: Run target program on ISS
  - Produce a snapshot of memory and all registers after “n” instructions
  - Pass #2: Execution may be resumed by playing a specific snapshot on a DUT
Tenstorrent Open Source Collateral

• Ocelot: BOOM with Vector Unit
  • https://github.com/tenstorrent/riscv-ocelot/tree/ocelot

• TT Whisper: Instruction set simulator
  • https://github.com/tenstorrent/whisper

• RISCV DV Kit
  • https://github.com/tenstorrent/rv-core-dv-kit

• RISCV Vector Tests
  • https://github.com/tenstorrent/riscv_arch_tests/tree/main/riscv_tests/rvv

• RISCV Arch Checker and Stimulus
  • https://github.com/tenstorrent/cosim-arch-checker
  • https://github.com/tenstorrent/riscv_arch_tests
Whisper: Open Source Instruction Set Simulator

• Provides infra for DV, perf modeling and debugging RISC-V software
• Supports multi-hart multi-core systems
• Supports all major RISC-V extensions
• Fast – over 150M instr/sec
• Used in DV in multiple ways
  • Conventional core-level lockstep checking
  • Block-level interface checking
  • Ex: Models micro-ops that can be used for frontend unit interface checking
  • Disassembler for RISC-V instructions
• Used in architectural modeling and exploration
  • Trace generation
  • Benchmark analysis
**Example: Legacy Architectural Painpoints, X86 and ARM**

<table>
<thead>
<tr>
<th></th>
<th>X86</th>
<th>ARM</th>
<th>RISCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Bootup Sequence (# instructions)</td>
<td>1000s</td>
<td>100s</td>
<td>&lt;5 instructions</td>
</tr>
<tr>
<td>Compatibility testing (# tests)</td>
<td>100s of thousands</td>
<td>100s of thousand</td>
<td>~1000, hierarchical</td>
</tr>
<tr>
<td>Architectural State</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Cost of adding new ISA features</td>
<td>Very high</td>
<td>Very high</td>
<td>Low-ish</td>
</tr>
</tbody>
</table>

- Architectural complexity has grown exponentially while ISA innovation has increased perhaps linearly
- Software to deal with legacy arch increasingly more complex
- Time to evaluate 30+ years of baggage on legacy architectures