OpenSoC System Architect

An Open Toolkit for Building High Performance SoC’s

David Donofrio\(^1\), Farzad Fatollahi-Fard\(^1\)
John D. Leidel\(^2,3\), Xi Wang\(^2\), Yong Chen\(^2\)

\(^1\) Computer Architecture Group, Lawrence Berkeley National Laboratory
\(^2\) Department of Computer Science, Texas Tech University
\(^3\) Tactical Computing Laboratories, LLC
Overview

• Motivation

• Design Acceleration with OpenSoC System Architect

• OpenSoC Tool Chain and Design Flow

• CoreGen Backend
Great *opportunities* exist for innovation through the end of Moore’s Law

**Post Moore Technology Curve**

**End of Moore’s Law**
End of Moore’s Law requires a different set of optimizations to continue performance scaling. Opportunities for additional specialization, reconfigurable computing, hardware / software co-design, etc.

**Now – 2025**
Moore’s Law continues through ~5 or 7nm. Beyond which diminishing returns are expected. Dark Silicon will begin to encourage specialization

**Post Moore Scaling**
New materials possibly introduced to allow continued process and performance scaling.

**Throughout...**
Continued increases in parallelism and heterogeneity will require advanced runtimes, programming environments and compiler optimizations in order to take full advantage of these new architectures.
Current architectures are wasteful

How far can we push performance scaling using specialization?

Only small fraction of chip area goes to computation

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>10.1%</td>
</tr>
<tr>
<td>Fetch</td>
<td>8.9%</td>
</tr>
<tr>
<td>Decode</td>
<td>6.0%</td>
</tr>
<tr>
<td>Reg Files</td>
<td>2.7%</td>
</tr>
<tr>
<td>Rename</td>
<td>12.1%</td>
</tr>
<tr>
<td>Mul/Div</td>
<td>4.0%</td>
</tr>
<tr>
<td>Int ALU</td>
<td>13.8%</td>
</tr>
<tr>
<td>Scheduler</td>
<td>10.8%</td>
</tr>
<tr>
<td>FPU</td>
<td>7.9%</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>23.7%</td>
</tr>
</tbody>
</table>
In all likelihood, Weddington concedes, the resulting technology "will never be as good as what is commercially available." But perhaps it could be made good enough "to bring the power and ability to design your own IC, or microprocessor, to smaller and smaller groups of people and drive down the enormous capital requirements of an entrenched, dinosaur industry."

Similarly, Michael Cooney of Network World describes the state of open-source hardware today as roughly where open-source software was during the mid-1990s – waiting for commercial suppliers to provide higher levels of support. "What made open-source software acceptable for many businesses was the arrival of support for it, such as Red Hat," he says, adding, "Something similar may take place with the hardware."

- Rapid growth in the adoption and number of open source software projects
- More than 95% of web servers run Linux variants, approximately 85% of smartphones run Android variants
- Will open source hardware ignite the semiconductor industry? Is RISC-V the hardware industry’s Linux?
Why Open Source Hardware?

• More effective vendor engagement
  • Generate *real* hardware that can be measured
  • Reduced the cost of development
  • By creating and sharing open hardware (RISC-V, OpenSoC Fabric)

• Closed source IP is a major drag on innovation in all technology spaces
  • Don’t need to be a big company to play – lower barrier of entry
  • Open nature enables customization at all levels of the design – not just at the periphery

• Lower Cost / Complexity for Development
  • Share hardware AND software stack
    • Compilers, debug, Linux ports
  • Focus NRE and license on new/innovative IP blocks
  • Stop squeezing license costs out of items that students can implement in a summer (license *hard* stuff instead)
“Those Who Don’t Remember the Past...”

• GoblinCore
  • Extensions to core RISC-V architecture for data intensive computing

• “OpenHPC”/“Project Whiskey Run”
  • Rocket-based design with multiple cores, network-on-chip, message passing, local scratchpad, and scatter/gather

• In both cases, we ended up doing a lot of the similar work to achieve only a point design
  • Why not make a generator to avoid repeating the same steps?
Design Acceleration with OpenSoC System Architect

Rapid construction of high performance, RISC-V SoC’s
Traditional Hardware Development

• Traditional hardware development is expensive!
• Why?
  • Development Time
  • Development Resources
• The entire development lifecycle requires months – years and non-trivial development teams
  • Completely ignoring the ”wet” costs such as masks, bring-up, etc
The Problem...

- Our current design methods separate core verification from core development
  - There are known intermediate test bench methods, but this doesn’t address full-chip verification
- This is especially true when developing large-scale SoC’s
  - This also ignores any potential software development required to perform the verification!
  - Compilers, debuggers, system libraries, etc

- *Can we provide a design flow with associated tools to perform primary and secondary verification in high-level languages in order to dramatically reduce the time to solution?*
The Solution: *OpenSoC System Architect*

- Combination of multiple tools to form a well-defined development flow for complex, RISC-V SoC’s
- Support for standard RISC-V modules & custom extensions
- The output of the tool is:
  - Pre-verified Chisel implementation for the SoC
  - Synthesizable verilog implementation for the SoC
  - LLVM compiler implementation for the SoC
OpenSoC Tool Chain and Flow

*Designing Complex RISC-V SoC’s with Custom Design Extensions*
Foundational Tools:
Chisel - A DSL for Hardware Design
A productive, flexible language for hardware design and simulation

• Increase the productivity of hardware designers
  • Chisel raises the abstraction level of hardware design
  • Introduces OOP techniques to hardware development
  • Encourages code reuse

• Hardware Generators are a more efficient technique for generating designs
  • New design methodology
  • Reduce waste in design
  • Reduce design time
  • Reduce risk
  • Reduce cost
Foundational Tools: RISC-V Based Processors
Open source, chisel based processors based on a new ISA

- Multiple flavors
  - Out-of-order (BOOM)
  - In-Order (Rocket)
  - IoT (Z-Scale)
- Powerful features
  - 32, 64, 128-bit addressing
  - Double precision floating point
  - Vector accelerators
- Complete SW stack available
- Highly configurable
  - Only add the features you want!
OpenSoC Fabric
An Open-Source, Flexible, Parameterized, NoC Generator

• Greater number of cores per chip driving the evolution of sophisticated on-chip networks
  • Needed new tools and techniques to evaluate tradeoffs

• Chisel-based
  • Allows high level of parameterization
    • Dimensions, topology, VCs, etc. all configurable
  • Fast, functional SW model for integration with other simulators
  • Verilog model for FPGA and ASIC flows

• Multiple Network Interfaces
  • Integrate with wide variety of existing processors
    • Tensillica, RISC-V, ARM, etc.
  • Integrate with IO devices as well
Accelerating the Design Process
Creating a complete suite of tools for rapid processor and compiler generation

OpenSoC

- OpenSoC Fabric
- OpenSoC Compiler
- OpenSoC Cores
- OpenSoC System Architect
OpenSoC Tool Chain & Flow

- OpenSoC Tool Chain & Flow consists of several integral moving parts:
  - OpenSoC System Architect Frontend
  - CoreGen IR & Backend
  - OpenSoC Fabric NoC Interconnect
  - RISC-V Standard Cores (in Chisel)
  - LLVM Compiler Infrastructure

- Users input design specs and extensions via frontend
- CoreGen ensures design correctness and continuity
  - Also generates backend Chisel and LLVM implementation
- OpenSoC Fabric “glues” all intermediate modules together with scalable NoC
CoreGen Backend

Maintaining coherency in the design using traditional compiler dependence analysis
What is **CoreGen**?

- CoreGen is an integrated infrastructure that utilizes traditional compiler techniques to build and verify complex SoC designs
  - CoreGen is essentially a compiler IR for SoC design and verification
- CoreGen utilizes an intermediate representation that permits us to:
  - Build high-level optimizations and verification of SoC control and data paths
  - Build high-level descriptions of complex SoC’s
  - **Automatically** build HDL representations of the SoC
  - **Automatically** build LLVM compiler backend implementations of the SoC and any extensions
  - Permits users to quickly and easily extend the SoC using new or existing IP
- The result: significant reduction in design, implementation and verification of complex SoC’s
What CoreGen is *NOT*

- CoreGen handles the high-level verification and integration of multiple HDL and high-level language constructs for SoC designs

- It also integrates the SoC compiler backend generation

- It *DOES NOT*:
  - Handle SoC layout (this is process specific)
  - Handle MAC generation for specific external interfaces (this is process specific)
  - Handle downstream tool optimization (Verilog/VHDL)
CoreGen IR

- CoreGen IR is a semi-strict, well-formed IR that permits users & tools to construct SoC designs that can be automatically verified by the CoreGen tools.
- The IR is stored in memory as C++ objects for easy processing while in use.
- The IR is stored on disk in well-formed XML.
- XML is not ideal, but:
  - It's easily human readable.
  - Abstracts the low-level circuit details of the implementation.
  - Is easily portable between machine architectures.

- CoreGen IR has multiple predefined modules that can represent unique portions of the architecture.
- **SoC's**
  - Analogous to a “socket” with all the contained cores, cache and modules.
  - Homogeneous or heterogeneous.
- **Core's**
  - RISC-V core with an ISA mode, addressing mode and base implementation (Rocket, Zscale, etc).
- **Extensions**
  - Register classes, registers, instructions, etc beyond the base implementation.
CoreGen IR Internal Dependence Graph
CoreGen Sample Extension IR

```
<extension>
  <core_extension type="rocc" name="SIMPLE-EXT28-EXT" num_instructions="1"
    num_register_class="1">
  <register_class name="simple-ext28-regclass" num_registers="1">
    <register name="reg0" index="0" width="64" register_attr="readwrite" simd="no" />
  </register_class>
  <instruction name="inst0" format="R" opcode="63" imm_enabled="no" funct3="0"
    funct7="0">
    <arg name="rd" type="output" register_class="GR64" register_class_type="standard" register_index_fixed="false" />
    <arg name="rs2" type="input" register_class="GR64" register_class_type="standard" register_index_fixed="false" />
    <arg name="rs1" type="input" register_class="simple-ext28-regclass" register_class_type="extended"
      register_index_fixed="false" />
  </instruction>
  </core_extension>
</extension>
```

This translates to:

```
inst0 xN, reg0, xN
```
What’s Next for System Architect?

• Better support for Chisel3!

• More integration with existing RISC-V tools/environments
  • Boom, SIMD implementations, RV128, etc

• Frontend support to import:
  • Existing Chisel modules
  • Verilog
  • SystemVerilog

• CoreGen support for standalone extension IR
  • Analogous to standalone library IR
Acknowledgements

• Department of Energy, Office of Science

• Laboratory of Physical Science
More Info

- **OpenSoC Website**: [http://www.opensoc.community](http://www.opensoc.community)
More Info

• **LBL Computer Architecture Group:** https://crd.lbl.gov/departments/computer-science/computer-architecture-group/

• **TTU DISCL:** http://discl.cs.ttu.edu

• **Tactical Computing Labs:** http://www.tactcomplabs.com

• **Source Code:**
  • **LBL-CoDEx Github:** https://github.com/LBL-CoDEx/

  • **TTU OpenHPC Gitlab:** http://discl.cs.ttu.edu/gitlab/groups/openhpc
“Open Source” *doesn’t* mean “Low Performance”

Sometimes you get more than paid for…

<table>
<thead>
<tr>
<th>Category</th>
<th>ARM Cortex A5</th>
<th>RISC-V Rocket</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>32-bit ARM v7</td>
<td>64-bit RISC-V v2</td>
</tr>
<tr>
<td>Architecture</td>
<td>Single-Issue In-Order 8-stage</td>
<td>Single-Issue In-Order 5-stage</td>
</tr>
<tr>
<td>Performance</td>
<td>1.57 DMIPS/MHz</td>
<td>1.72 DMIPS/MHz</td>
</tr>
<tr>
<td>Process</td>
<td>TSMC 40GPLUS</td>
<td>TSMC 40GPLUS</td>
</tr>
<tr>
<td>Area w/o Caches</td>
<td>0.27 mm²</td>
<td>0.14 mm²</td>
</tr>
<tr>
<td>Area with 16K Caches</td>
<td>0.53 mm²</td>
<td>0.39 mm²</td>
</tr>
<tr>
<td>Area Efficiency</td>
<td>2.96 DMIPS/MHz/mm²</td>
<td>4.41 DMIPS/MHz/mm²</td>
</tr>
<tr>
<td>Frequency</td>
<td>&gt;1GHz</td>
<td>&gt;1GHz</td>
</tr>
<tr>
<td>Dynamic Power</td>
<td>&lt;0.080 mW/MHz</td>
<td>0.034 mW/MHz</td>
</tr>
</tbody>
</table>

Y. Lee, UCB @ Hotchips 2017
“Open Source” doesn’t mean “Low Performance”

Sometimes you get more than paid for…

Y. Lee UCB @ Hotchips 2017
Frontend & CoreGen

• OpenSoC System Architect
  • Serves as the user interface to CoreGen and other tools
  • Cross-platform GUI: Linux, OSX (Windows possible)
  • Integrated build & design environment

• CoreGen
  • C++ Library that handles IR, dependence analysis for SoC & extensions
  • CoreGen generates Chisel and LLVM compiler implementation
OpenSoC Fabric Interconnect

- Handles all the “glue” logic between multiple cores and extensions
  - Caching hierarchy
  - Extensions
    - Black Box modules
    - RoCC modules
    - User-driven modules
  - On-chip fabric
  - Peripheral modules
  - Memory interfaces
- Written in Chisel to support the entire design flow
OpenHPC CoreGen Implementation

• As an excellent test, we implemented the entire OpenHPC register, instruction and dependency specification in the CoreGen tool

• The development time is reduced from days to minutes!
Contacts

• **David Donofrio**: ddonofrio@lbl.gov

• **Farzad Fatollahi-Fard**: ffard@lbl.gov

• **John Leidel**: john.leidel@ttu.edu / jleidel@tactcomplabs.com

• **Xi Wang**: xi.wang@ttu.edu

• **Yong Chen**: yong.chen@ttu.edu