Appears in the Proceedings of the RISC-V Day Workshop at the 51st Int'l Symp. on Microarchitecture (MICRO-51)

A New Era of Silicon Prototyping in Computer Architecture Research

Christopher Torng, Shunning Jiang, Khalid Al-Hawaj, Ivan Bukreyev, Berkin Ilbeyi,
Tuan Ta, Lin Cheng, Julian Puscar, Ian Galton, and Christopher Batten

School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{clt67,cbatten}@cornell.edu

***Abstract*—Silicon prototyping has historically played a key role in computer architecture research by validating assumptions, enabling measurement of real system-level performance and energy efficiency, and grounding simulation-based studies. The demand for silicon prototyping has risen dramatically due to a surge in hardware innovation driven by fast-paced advances in software (e.g., machine learning applications). Silicon prototyping in this climate further provides critical answers to business concerns including estimated total cost of ownership and product/market fit. However, in conventional wisdom, building chips is a massive undertaking that is simply out of reach for most researchers. In this paper, we argue that advancements in open-source infrastructure for the RISC-V ecosystem are synergizing with continued development of productive open-source design tools to significantly reduce design challenges. At the same time, multi-project wafer services have begun supporting advanced technology nodes with very small minimum sizes for significantly reduced costs, bringing forth a new era of silicon prototyping in which anyone can build prototypes. This paper provides a case study for cost-effective and productive silicon prototyping and describes how we paired the RISC-V ecosystem with productive open-source design tools to build a small 1×1.25 mm RISC-V system in TSMC 28 nm. The prototype is written in an open-source Python-based hardware modeling language, includes a fully synthesizable PLL written in SystemVerilog, and was built with an open-source modular VLSI build system to organize the ASIC toolflow. We hope that our experience convinces architects that silicon prototyping of RISC-V systems is both feasible and attractive for supporting future research.**

We present a case study for cost-effective and productive silicon prototyping and describe our experience building BRGTC2 (i.e., Batten Research Group Test Chip 2), a small 1×1.25 mm 6.7M-transistor RISC-V system in TSMC 28 nm. BRGTC2 is designed and implemented using PyMTL, a new Python-based hardware modeling framework [5, 6]. Figure 1 shows the chip block diagram including four RISC-V RV32IMAF cores which share a 32kB instruction cache, 32kB data cache, and single-precision floating-point unit along with microarchitectural mechanisms to mitigate the performance impact of resource sharing. The chip also includes a fully synthesizable high-performance PLL written in SystemVerilog and ported from the Celerity SoC [1, 4]. In this paper, we provide an overview of various key aspects of our silicon prototyping experience, including our timeline and costs, open-source software toolchain and ISA, open-source cycle-level modeling, open-source RTL modeling, open-source modular VLSI build system, and synthesizable analog IP. The landscape of connected open-source tools for computer architecture research now extends across the computing stack, making RISC-V silicon prototypes both feasible and attractive as vehicles for future research.

Timeline and Costs – Our team of seven graduate students completed the 28 nm SoC in two months. This design period encompassed developing simple applications, porting an in-house work-stealing runtime to our RISC-V target, cycle-level design-space exploration of sharing architectures in gem5 [3], RTL development and testing of each component including SRAMs (see Figure 1), composition testing at RTL and gate level, SPICE-level modeling of the synthesizable PLL, IO floorplanning and physical design, post-place-and-route performance tuning, and final tapeout. About one person-month was required for a student with prior ASIC experience to bring up the TSMC 28 nm design flow for the first time, including the process libraries, standard cell libraries, IO cell libraries, Synopsys DC, Cadence Innovus, and Calibre signoff tools, in order to pass DRC/LVS for dummy logic surrounded by staggered IO pads and no SRAM blocks. The entire chip RTL was designed in the final one-month period by seven graduate students using PyMTL for design, test, and composition. Multi-project wafer services have recently begun to support advanced technology nodes (e.g., 28 nm) with very small minimum sizes (e.g., 1×1 mm) at very reasonable pricing (e.g., $14K). We chose the Tiny2 program with MOSIS, selecting a 1×1.25 mm die size and one hundred parts for about $18K. Other services are also available for university researchers at similar pricing (e.g., Muse Semiconductor [7]). Other costs included packaging (less than $2K for twenty parts), board costs (less than $1K for PCB and assembly), graduate student salaries, physical IP costs, and EDA tool licenses. The open-source RISC-V ecosystem helped us avoid any costs associated with the ISA and also helped avoid long communication delays with third parties, which can take months to resolve and can significantly delay a time-sensitive project. Many small benefits also made a difference (e.g., a very short but descriptive RISC-V ISA spec saving us from reading thousand-page specs, no time and effort required to bring up and modify a software toolchain, open-source VLSI implementations of previously taped out RISC-V SoCs for reference including Rocket [2] and Celerity [1, 4]).

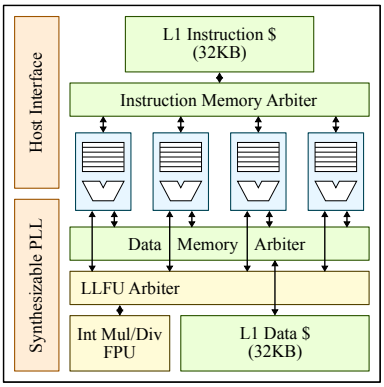

Figure 1. Block Diagram of BRGTC2 – We paired the RISC-V ecosystem with productive open-source design tools to build a 1×1.25 mm RISC-V system in TSMC 28 nm with seven graduate students in two months.

Taped out: May 2018
Silicon: Fall 2018

Open-Source Software and ISA – The RISC-V software toolchain was tremendously useful as an out-of-the-box and standard solution for compiling applications for our system. In particular, we leveraged recent GCC support with options targeting RV32IMAF, and we were also able to use inline assembly in our in-house work-stealing runtime library to implement hints and to track stats. The RISC-V ISA itself was also a tremendous success. Because the ISA is designed as a small base set of instructions with modular extensions, we were able to apply an incremental design approach by writing RTL to first support the base set (i.e., RV32I), then add multiply/divide support (i.e., RV32IM), then add atomic support (i.e., RV32IMA), and finally add floating-point support (i.e., RV32IMAF). We also leveraged the control and status registers for many custom purposes including tracking stats.

Open-Source Cycle-Level Modeling – The gem5 simulator system [3] is a popular platform for simulator-based cycle-level modeling in the computer architecture research community. Multicore support has recently been added for RISC-V [9], providing computer architects a critical tool for cycle-level design-space exploration of complex RISC-V systems. We leveraged RISC-V support on gem5 to explore our sharing architecture shown in Figure 1, which shares caching resources and long-latency functional units. We swept important parameters including the latency to shared resources, the number of each resource to share, the impact of memory coalescing techniques, the size and capacity of caches and buffers, and the impact of various arbitration schemes.

Open-Source RTL Modeling – We paired the RISC-V ecosystem with a new Python-based hardware modeling framework, PyMTL [5, 6], to build our RISC-V system. PyMTL leverages the Python programming language to create a highly productive and flexible environment for test, design, and composition in BRGTC2. *Testing* in PyMTL enables access to full-featured software testing frameworks built for Python (e.g., pytest [8]), providing useful features including automatic test discovery, modular fixtures, and rich customizable plugins. We leveraged PyMTL support for two-state simulation and state initialization to one, zero, and random values. *Designing* in PyMTL was more accessible than in Verilog for students new to RTL design, avoiding many well-known quirks of the older language while also enabling a familiar style of debugging in Python. *Composition* in PyMTL enabled powerful multi-level co-simulation of functional-level, cycle-level, and RTL models. For example, to debug an issue with atomics, we swapped in a functional model of the cache to narrow the bug location down to other components. The PyMTL framework generates Verilog for our standard ASIC toolflow. Overall, we found the PyMTL framework to be a tremendous success for designing a RISC-V system from scratch with rigorous testing support.

Open-Source ASIC Flow Organization – The availability of high-quality, community-developed reference ASIC flows is a tremendously useful resource for both new and experienced chip designers. We designed our RISC-V silicon prototype (see Figure 2) using a modular VLSI build system[1], which we have open-sourced as a reference organization of the ASIC toolflow for architecture and VLSI researchers interested in silicon prototyping. One of the most challenging aspects of working with ASIC flows is managing the many moving pieces (e.g., PDK, physical IP libraries, ASIC-specific tools), which come from many different vendors and yet must still be made to work together coherently. Despite the great effort required to successfully assemble a working ASIC flow, teams typically end up with little reuse across projects as designers frequently tweak steps, target different technology nodes, or even use different vendors for physical IP. Furthermore, while architectural design-space exploration tends to require just a few stages (e.g., synthesis, simple floorplanning, no IO cells, need not be DRC or LVS clean), a full tapeout requires many more stages to guarantee manufacturability. The key idea behind a modular VLSI build system is to avoid rigidly structured ASIC flows that cannot be repurposed, and to instead break the ASIC flow into modular steps that can be re-assembled into different flows. With a modular build system, architecture projects can omit detailed steps from a chip flow (e.g., use default floorplan, skip IO pads, skip DRC/LVS), while a VLSI project can include key steps for VLSI research (e.g., custom floorplanning for synchronizer research), while reusing the common steps in between. Our approach also includes the idea of an ASIC design kit, which is the specific set of physical backend files required to successfully build chips, as well as a unified and standard interface to those files. A well-defined interface enables swapping process and IP libraries without modification to the scripts that use them. Finally, this approach embraces plugins that hook into steps across the entire ASIC flow for design-specific customization. We have found that a modular ASIC flow enables productive reuse of steps across projects, and we have open-sourced our flow for reference.

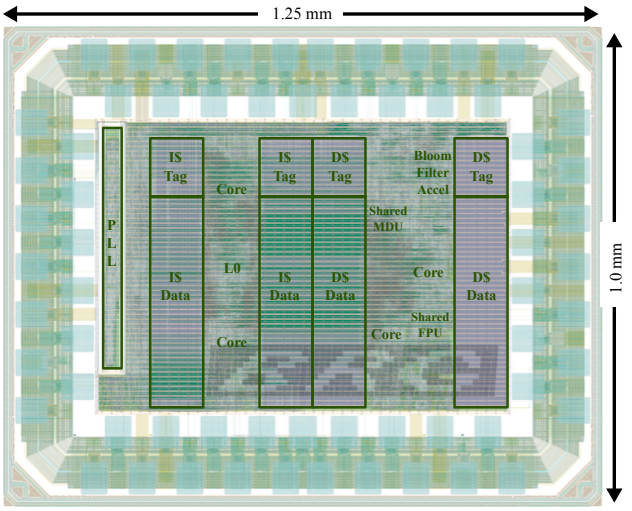

Figure 2. Chip Plot of BRGTC2 – The research goal of the chip was to provide detailed performance, area, and energy numbers for various projects in our research group, while also evaluating the usability of synthesizable analog IP and experimenting with PyMTL support for ASICs.

Synthesizable Analog IP – Our RISC-V system is clocked by a synthesizable PLL that was first designed for use in the Celerity SoC [1, 4], but has been adapted for use in a

[1]Modular VLSI Build System: https://github.com/cornell-brg/alloy-asic

TSMC 28 nm process. Management of analog IP is traditionally a significant challenge in the design of a complex SoC. Mixed-signal crossings between analog and digital domains are well-known sources of costly design mistakes, and the communication between analog and digital design teams adds project management overhead that is nevertheless crucial to the overall success of the chip. Synthesizable analog IP is an approach that migrates blocks that are traditionally full-custom into the digital domain to mitigate these challenges. At a high level, the PLL relies on an array of internal ring oscillators with varying numbers of stages and configurable load capacitances (i.e., configurable NAND2 loads) controlled by a digital feedback loop. Our design experience using this PLL was a success, as we would not have been able to quickly design a PLL to generate programmable clocks from scratch with confidence. To test our design, we ported our PLL as a GDS from Cadence Innovus to Virtuoso, where we ran SPICE-level extracted simulations. The synthesizable PLL is planned to be open-sourced, which will provide architects an additional useful tool for silicon prototyping.

Final Thoughts – Silicon prototyping is a key aspect of computer architecture research with rising demand from researchers in both industry and academia. We argue that advancements in open-source infrastructure for the RISC-V software/hardware ecosystem, the flexibility of new productive open-source design tools (e.g., PyMTL, open reference ASIC flows, synthesizable analog IP), and the availability of multi-project wafer services in advanced technology nodes with small minimum sizes have significantly reduced both design costs and challenges, bringing forth a new era of silicon prototyping in which anyone can build prototypes. We hope that our overview of BRGTC2 helps convince architects that the broader ecosystem of connected open-source tools for computer architecture research now extends across the computing stack, making RISC-V silicon prototypes both feasible and attractive as vehicles for future research.

Acknowledgements – This work was supported in part by NSF CRI Award #1512937, NSF SHF Award #1527065, DARPA POSH Award #FA8650-18-2-7852, and equipment, tool, and/or physical IP donations from Intel, Xilinx, Synopsys, Cadence, and ARM. We thank U.C. Berkeley, the RISC-V Foundation, and other contributors to the RISC-V software and hardware ecosystem. We thank Shreesha Srinath for his support and work on sharing architectures which motivated our design. 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.

REFERENCES

[1] T. Ajayi et al. Celerity: An Open Source RISC-V Tiered Accelerator Fabric. *Symp. on High Performance Chips (Hot Chips)*, Aug 2017.

[2] K. Asanović et al. The Rocket Chip Generator. Technical report, EECS Department, University of California, Berkeley, Apr 2016.

[3] N. Binkert et al. The gem5 Simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, Aug 2011.

[4] S. Davidson et al. The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips. *IEEE Micro*, Mar 2018.

[5] S. Jiang, B. Ilbeyi, and C. Batten. Mamba: closing the performance gap in productive hardware development frameworks. *Design Automation Conf.*, Jun 2018.

[6] D. Lockhart, G. Zibrat, and C. Batten. PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research. *Int'l Symp. on Microarchitecture*, Dec 2014.

[7] Muse Semiconductor. Online Webpage, 2018 (accessed August 20, 2018). `https://www.musesemi.com`.

[8] pytest: helps you write better programs. Online Webpage, 2018 (accessed August 20, 2018). `https://pytest.org`.

[9] T. Ta, L. Cheng, and C. Batten. Simulating Multi-Core RISC-V Systems in gem5. *Workshop on Computer Architecture Research with RISC-V*, Jun 2018.