A Qualitative Approach to Agile Hardware Design

Daniel William Ruelas-Petrisko

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2025

Reading Committee:

Michael Taylor, Co-Chair

Mark Oskin, Co-Chair

Zachary Tatlock

Program Authorized to Offer Degree:

Computer Science and Engineering

University of Washington

Abstract

A Qualitative Approach to Agile Hardware Design

Daniel William Ruelas-Petrisko

Co-Chairs of the Supervisory Committee:
Michael Taylor
Computer Science & Engineering

Mark Oskin
Computer Science & Engineering

The recent, dramatic rise of small hardware startups illustrates the demand for rapid, low-cost ASIC design. While researchers borrow Agile principles from software engineering, such as quick iteration, aggressive reuse, and continuous integration, their methodologies have struggled to escape small labs. Unfortunately, there remain clear challenges to broader adoption. Effective research evaluation techniques often do not scale to larger, more complex design flows. Conversely, risk aversity means that traditional hardware design methodologies can be rigid and slow to adapt. The result is that despite measurable quantitative improvements, traditional Agile design methodologies cannot be practically applied in complex SoC designs. This thesis presents a comprehensive approach to Agile Hardware Design through three tools: BSG Pearls, BlackParrot, and ZynqParrot.

All three projects are open-source, silicon-proven, and available for immediate

use under a permissive BSD-3 License. Hardware designers can leverage these efforts to make Agile Hardware Design qualitatively more feasible across a wide variety of research and commercial projects.

TABLE OF CONTENTS

Page

List of Figures iv

List of Tables vi

Glossary vii

Chapter 1: Introduction 1
1.1 Agile Hardware Design 1
1.1.1 Background 1
1.2 BSG Pearls: Effortlessly Synthesizable Building Blocks 5
1.2.1 Introduction 5
1.2.2 Collateral Reuse in Hierarchical Designs 6
1.2.3 BSG Pearls Library 8
1.2.4 Related Work 11
BaseJump STL 11
Pulp Platform 11
ChipYard 12
Hammer 12
Other Design Methodologies 12
1.3 NoC Symbiosis: Lessons Learned from Agile Tapeouts 13
1.3.1 Abstract 13
1.3.2 Introduction 14
1.3.3 Network-on-Chip Symbiosis 17
1.3.4 NoC Symbiosis in the Wild 19
BlackParrot v0 21
BlackParrot v1 22

BlackParrot v2 . . . 23
OpenPiton . . . 24
Summary . . . 24
Symbiosis Gallery . . . 24
1.3.5 NoC Symbiosis in Captivity . . . 25
Utilization vs. Network Link Width . . . 28
Utilization in DLDT . . . 28
Utilization with SLDT . . . 30
Attainable Symbiotic Area . . . 32
Networks vs Area . . . 33
1.3.6 Symbiotic Design Methodology . . . 33
1.3.7 Conclusion . . . 37

Chapter 2: BlackParrot . . . 38
2.1 The BlackParrot Core . . . 38
2.1.1 Abstract . . . 38
2.1.2 Introduction . . . 39
Success Metrics . . . 39
Design Manifesto . . . 42
System Architecture . . . 44
Race-Free Programmable Cache Coherence . . . 44
2.1.3 Heterogeneous Multicore Tiles . . . 45
BlackParrot Microarchitectural Tile Types . . . 46
Networks-on-Chip . . . 48
Decoupled Core Microarchitecture . . . 49
Efficiency Through Thin Interfaces . . . 50
Agile Development Process . . . 53
Conclusion . . . 60

Chapter 3: ZynqParrot . . . 61
3.1 ZynqParrot . . . 61
3.1.1 Abstract . . . 61
3.1.2 Introduction . . . 62

Hardware Architecture 70
Emulation Layer 72
Scale-Down Decomposition 74
PanicRoom: Portable Bare-Metal Benchmarking 75
3.1.3 Case Studies: ZynqParrot in the Wild 78
A Scale-Down Pandamonium Cluster 79
Microarchitectural Optimization: CatchUp ALU 81
ZynqParrot ASIC Bring-up Boards 85
High Fidelity Sampling with ZynqParrot 88
3.1.4 Conclusion 90
Related Work 90
Future Work 93
Final Thoughts 93

Chapter 4: Conclusion 95
4.1 Contributions 95
4.2 Future Work 97
4.3 Final Thoughts 98

Bibliography 99

Appendix A: Open-Source Link Farm 128

LIST OF FIGURES

Figure Number Page

1.1 Industrial-Academic Feedback Loop 2
1.2 Cross-Coupling in Latency-Insensitive Designs 3
1.3 Generational IP Integration Challenges 4
1.4 Tool-Agnostic BSG Pearls Flow 5
1.5 Collateral Redundancy in Design Flows 7
1.6 BSG Pearl IP Catalog 9
1.7 BSG Pearl Complexes 10
1.8 Symbiotic Taxonomy 15
1.9 Symbiotic Router Area 20
1.10 Double-Track Utilization vs Width 26
1.11 Unutilized Double-Track Area 27
1.12 Double-Track Bandwidth vs Area 29
1.13 Single-Track Utilization vs Width 30
1.14 Unutilized Single-Track Area 31
1.15 Co-located Router Area 32
1.16 Co-located Logic and SRAM Area 34
1.17 Multi-router Symbiosis 35

2.1 BlackParrot Success Metrics 40
2.2 Contemporary Core Comparison 42
2.3 Tile Taxonomy 45
2.4 Core Microarchitecture 51
2.5 BlackParrot Memory End 52
2.6 Modular Testing Framework 55
2.7 Cosimulation Flow 57
2.8 BlackParrot v0 Tapeout Diagram 59

3.1 Scale-Down Process 64
3.2 ZynqParrot Architectural Overview 69
3.3 UART Bridge 71
3.4 AXI DPI Co-Routines 72
3.5 Multi-Board Decomposition 75
3.6 PanicRoom Diagram 77
3.7 Pandamonium Cluster 80
3.8 Performance Improvement of CatchUp ALU 83
3.9 CatchUp ALU Diagram 84
3.10 ZynqParrot BringUp Board Architecture 85
3.11 Slowdown vs Sample Granularity 87
3.12 Sampling Cycle-Stack Breakdown 89

LIST OF TABLES

Table Number Page

1.1 Symbiosis Achieved by Tiled Architectures 21

3.1 FPGA Emulation Price Comparison 66

3.2 PanicRoom vs Other Proxy Solutions 76

Glossary

Accelerator Specialized hardware designed to perform specific tasks more efficiently, typically under orchestration by a general-purpose control processor.

Agile A methodology for design that emphasizes iterative development across short design cycles. Agile is often contrasted with traditional "waterfall" development methodologies with strict scheduling of deliverables.

ASIC Application-Specific Integrated Circuit. In this thesis, we consider the ASIC designs for a variety of applications.

Collateral Additional timing constraints, design constraints, test vectors, and other supporting materials needed for synthesis or verification.

Design Rule Check (DRC) A verification step that checks the physical design against a fab-provided set of rules that ensure manufacturability.

EDA Electronic Design Automation. Software tools used for designing, testing and manufacturing circuits. In this work, we mostly focus on digital design flows.

Functional Verification The process of ensuring that a design meets specified requirements and functions correctly. In this work, we focus on

dynamic techniques such as simulation and emulation to validate correctness and refer to this process as Verification.

Gate Level Simulation A type of simulation that models the behavior of a digital circuit at the gate level, typically with annotated delays based on the results of STA.

Hardening The replacement of generic design components with vendor-specific implementations. This is often done automatically by FPGA vendors and often done manually by ASIC designers to port between technology nodes.

Latency Insensitive Design A design methodology that allows low-level hardware components to communicate through ready-valid handshakes without considering their relative latencies.

Netlist A textual representation of a digital circuit that describes the standard cells and their interconnections, but typically lacks physical placement information.

Network-On-Chip (NoC) An interconnection methodology that uses packetized links to communicate between chip components, rather than a shared bus.

Place-and-Route (APR) The process of placing standard cells resulting from synthesis into physical locations and routing wires between them.

Release Candidate A version of the design that is considered a "successful"

tapeout. Typically, any additional feature that result in a breaking change will be pushed to the next Release Candidate.

Static Timing Analysis (STA) A verification step that checks the timing of a design to ensure that it meets the required performance specifications. This step is a pessimistic analysis that provides confidence in the design's timing closure, but does not consider timing exceptions or dynamic behavior.

Synthesis The process of converting a high-level (RTL) description of a design into a netlist consisting of standard cells.

Tapein Non-standard term for an internal design checkpoint that meets critical timing, verification and design rule checks but has not been sent to fabrication.

Tapeout The final step in the ASIC design process where the design files are sent to fabrication.

Technology Node A quality metric used in semiconductor manufacturing, roughly corresponding to the minimum feature size of an equivalent planar transistor.

ACKNOWLEDGMENTS

Building chips is an unfathomably complex process, requiring deep-seated, highly interconnected infrastructure across institutions, companies and individuals. After your first successful tapeout, you teem with lessons best summarized as "how not to build a chip".

This thesis is the same. It could not be done without my committee, my colleagues, my friends, my family, open-source communities and the support of countless individuals. After extensively exploring "how not to finish a PhD", I thank everyone for shaping myriad side-projects into a cohesive story.

From my previous academic life at UIUC: Professor Rakesh Kumar, Professor Puneet Gupta, (now Professor) Henry Duwe, and Saptadeep Pal. Jose Rodrigo Sanchez Vicarte for teaching me, among other things, vim shortcuts and tea preparation. Thank you to the PASSAT group for taking in a hacker with no research experience.

My committee: Professors Michael Taylor, Jonathan Balkind, Zach Tatlock, Akshay Mehra and Mark Oskin. Thank you for advice, guidance and opportunities over the years. Michael for connecting me with all-star teams to build real chips, without which I would be a much narrower researcher. Jonathan for guiding me through the tenuous and exciting world of open-source hardware research. Zach for fostering a PLSE lab culture so strong that it permeated the entire department. Akshay for asking insightful questions that pushed me

to be clearer. And Mark for interesting, crazy ideas to balance my engineering cynicism, as well as stepping up to push me over the finish line.

The BlackParrot team: Professors Michael Taylor, Mark Oskin and Ajay Joshi, as well as Mark Wyse, Farzam Gilani, Zahra Azad and many others. It was a blast ideating and building an industrial-strength open-source design.

The Bespoke Silicon Group: Scott Davidson, Paul Gao, Tommy Jung, Dustin Richmond, Anoop Mysore, Bandhav Veluri, Farzam Gilani, Yuan-Mao Cheuh, Max Ruttenberg. Whenever designing something, I find myself considering the "BSG way" to do it. I will always remember the late night hacking.

My mentees: Brian Arnold, Songchun Li, Sripathi Muralitharan, Shashank Vijaya Ranga, Yuan-Mao Chueh, Martin Schreiber, Timon Platz, Kinza Qamar Zaman, Adithya Sunil, Brandon Page, Anoop Mysore Nataraja, Colin Knizek. I hope to see many open-source contributions in the future.

The many open-source hardware communities: RISC-V Foundation, the OpenROAD team, the PULP Platform Team. Special thanks to the FOSSi Foundation Board: Andrew Back, Jonathan Balkind, Julius Baxter, Alex Bradbury, Simon Cook, Peter Gielda, Olof Kindgren, Matt Venn, Philipp Wagner, Stefan Wallentowitz. I cannot wait to CatchUp at LatchUp 2050.

Maxentric Technologies: Houman Ghajari, Paul Theilmann, Tuan Do, and the rest of the team for providing support for a real-world application of Agile Hardware Design. Without a near irrational level of trust in my research, MaXSoC (and this thesis) would not have succeeded.

#binary-blobs, #meet-and-eat, and #spicelords: Max Ruttenberg, Pratyush Patel, Gus Smith, Jacob GeffenWood, Katie Lim, Ellis Michael, Anthony

Marucci, Vinitha Ranganeni, and others for keeping me sane with dinners, boardgames, disc golf and completely unreasonable amounts of Twilight Imperium.

#clamily and other Seattle friends: Lan Le, Revanth Rameshkumar, Anne McLaughlin, Tim and Ike Lagnese, Luciana Pinheiro, and Mitchell Young. Thank you for introducing me to clam digging, crabbing, and boating adventures. Seattle is a much more interesting place with you in it.

My Midwestern friends: Tom Targosz, Yuliya Semibratova, Clayton Quasny, Matt Nilles, and Daniel Reyburn, who have always shown up for me. Thank you for countless memories.

My family: Daniel, Margaret, and Jessica Petrisko; Andrew Mason; Scrappie and Koda. I am grateful you visited me all around the country; and always were a safe place to return. Thank you for always picking up the phone.

My grandparents: Bill and Anne Petrisko. Bill taught me there was a right way to fix just about anything. They may have been prouder had I stayed at John Deere.

My newly-extended family: Sabas, Mary, Alexandra and Brianna Ruelas; Mitchell Ida; Lucas, Dante and Cheddar. I am proud to be part of the Ruelas clan and look forward to enjoying paperless Thanksgivings.

My wonderfully supportive pod-mate turned fiancé turned wife Luzdary Ruelas-Petrisko, who has semi-patiently waited years for me to finish this thesis. You have the uncanny ability to both keep me grounded and turn life into a constant adventure. You inspire me to be my best person. I cannot wait to see where we (and Schnapps) go next together.

This is how humans are: We question all our beliefs, except for the ones that we really believe in, and those we never think to question.

Orson Scott Card

You must stay drunk on writing so reality cannot destroy you.

Ray Bradbury

When things are going sweetly and peacefully, please pause a moment, and then say out loud, "If this isn't nice, what is?"

Kurt Vonnegut

Portions of this work were partially supported by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement numbers FA8650-18-2-7863 and FA8650-18-2-7856; NSF grants SaTC-1563767, SaTC-1565446. This work intersects and leverages research and infrastructure created by the members of the Bespoke Silicon Group, spanning across accelerators ([47, 88, 275, 263, 178, 175, 279, 38, 248, 202, 98, 213, 17, 203, 97, 99, 140, 247, 25, 180, 179, 181]), ML ([246]), ASIC Clouds ([239, 264, 234, 225, 141, 142, 157, 227]), open source hardware ([237, 228, 82, 199]) RISC-V ([183, 197, 196, 77, 278, 4, 245, 125, 192, 166, 70, 191, 154, 128]), Network-on-Chips ([126, 187, 281, 231, 144]), security ([55, 23, 54, 110, 111, 9]), benchmark suites ([241, 146, 27]), dark silicon ([224, 223, 224, 226, 42, 100, 97, 247]), multicore ([66, 65, 183, 105, 103, 104, 222, 231, 233, 229, 221, 144, 230, 232, 220, 249]), compiler tools ([124, 90, 122, 92, 123, 121, 91, 274, 2, 20]) and FPGAs ([280, 119, 27, 93]).

Chapter 1

INTRODUCTION

1.1 Agile Hardware Design

1.1.1 Background

Agile Hardware Design methods promise cheaper, faster ASIC development but face adoption challenges. While the quantitative benefits of reuse and rapid iteration are well-established, this thesis argues many remaining barriers are qualitative: complex hardware library integrations; discrepancies between disparate verification environments; and hierarchical leakage preventing effective reuse in ASIC backend flows. In aggregate, these barriers increase the friction of Agile Hardware methods, limiting their effectiveness in practice.

To address these challenges, this thesis presents three open-source projects: BSG Pearls, BlackParrot and ZynqParrot. BSG Pearls introduces a hierarchy-insensitive methodology of middleware component assembly that improves tool predictability of mid-sized SoC designs. BlackParrot is a modular, multicore RISC-V processor designed to simplify system integration for accelerator designs. Finally, ZynqParrot provides a unified pre-silicon and post-silicon infrastructure that enables reuse of verification efforts across the design lifecycle.

The intersection of decreasing test chip tapeout costs, and increasing costs of pre-silicon verification has led to a dramatic increase in N=1 tapeouts for small research labs and startups. Quickly, these teams realize that the

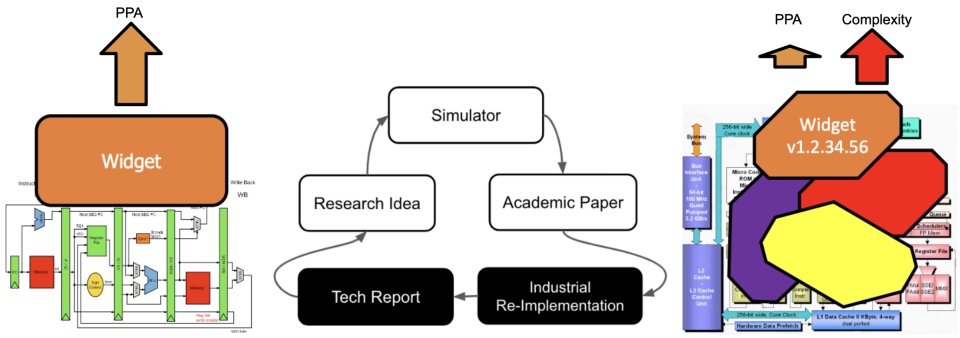


Figure 1.1: Impactful computer architecture research requires a feedback loop between academia and industry. Unfortunately, the high cost of hardware prototyping limits holistic analysis of design tradeoffs, where benefits may be reduced or eliminated by second or third order effects. By dramatically reducing the iteration cost of hardware design, researchers can build realistic prototypes more readily embraced by industry.

relatively small effort to prepare their accelerator design is dwarfed by "out-of-scope" concerns such as on-chip control orchestration and memory system design. We designed the BlackParrot [185] multicore processor to fill this gap as a "rest of the owl" control system for rapidly designing accelerator SoCs. While other open-source control cores are in use [19] [57] [273], BlackParrot is unique in its focus on modularity and ease of integration.

Verification efforts are a large and ever-growing chip design cost. Pre-silicon verification focuses on thorough IP-level code coverage through cycle-accurate but painfully slow simulations. Architects have historically bridged this gap with FPGA emulation [137, 217, 69, 67]. However, FPGA emulation frameworks struggle with both scalability and cycle-accuracy, typically requiring costly single-use infrastructure or sacrificing generality. We introduce Zynq-Parrot: a unified pre-silicon / post-silicon accelerator design infrastructure capable of verifying functionality and performance at tunable granularity.

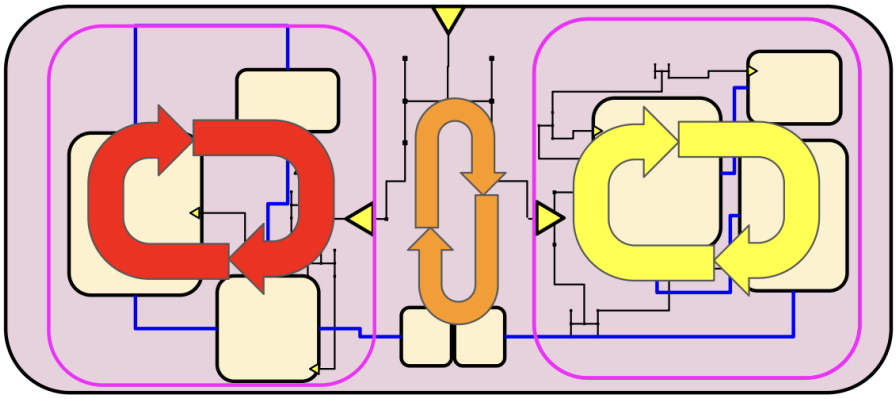
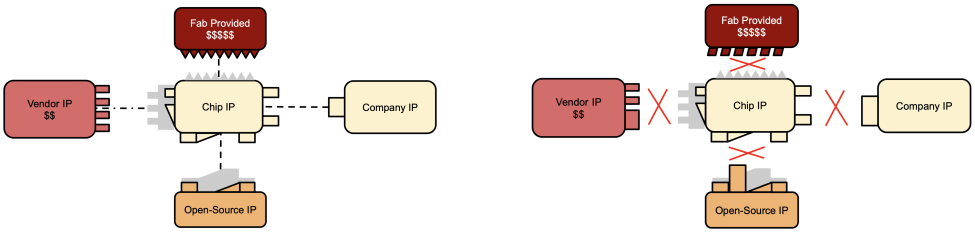


Figure 1.2: While latency-insensitive design logically partitions discrete hardware components, it does not prevent physical interaction due to shared resources such as clock and reset trees. This means that even small changes in one component can cause unpredictable and costly schedule delays during a tapeout.

Compared to burdensome prerequisites of existing FPGA emulation frameworks, ZynqParrot is designed for *Scale-Down* emulation: it can run on a single Zynq-7000 SoC, making it accessible to small teams and research labs.

When silicon teams design their second ASIC, they integrate the lessons learned from their first chip. Unfortunately, changes in a single IP can disturb power, area and timing budgets across the chip, forcibly reopening mature IP blocks and eliminating respin cost savings. The challenge with reuse at this granularity is minimizing the overhead of full isolation while maintaining predictability at every stage of the flow. Traditional IP reuse proponents focus on the verification savings at the logic level; however, even changes that are externally non-observable from a logical perspective can have significant impact on the backend flow. From the perspective of the tools, any change must be re-verified through the entire flow, from synthesis to place-and-route, to ensure that the design meets specifications. We describe



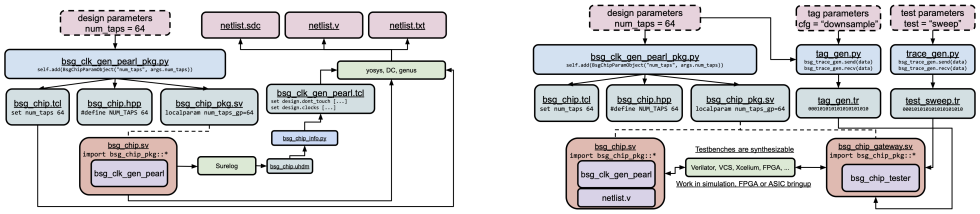
(a) When developing a new IP, designers must consider integration with a variety of internal and external stakeholders. This often results in the development of non-reusable "glue code", increasing development costs and overall design complexity.

(b) When reusing an existing IP, even minor changes in upstream dependencies can cause unforeseen incompatibilities. Re-design and re-verification efforts can lead to increased costs and delays, undermining reuse benefits.

Figure 1.3: IP integration in hardware design has a high, atomic, cost of change, leading to "glue code" proliferation and the creation of non-reusable, bespoke components. This increases the cost of each tapeout and expand delivery schedules.

a methodology to increase predictability and reuse through BSG Pearls: middleware components that enable rapid and safe iteration of mid-sized SoC designs. While non-monolithic design methodologies are not new [56, 205, 147], they have historically focused on quantitative benefits such as area and power savings rather than the predictability and ease of integration provided by BSG Pearls.

This thesis presents a comprehensive approach to Agile Hardware Design through BSG Pearls, BlackParrot, and ZynqParrot. All three projects are open-source, silicon-proven, and available for immediate use under a permissive BSD-3 License. Hardware designers can leverage these efforts to make Agile Hardware Design qualitatively more feasible across a wide variety of research and commercial projects.



(a) BSG Pearls synthesis flows identify necessary constraints for special cases such as hardened cells, synchronizers, I/O constraints and generated clocks.

(b) BSG Pearls verification flows identify necessary annotations for accurate gate-level simulation, even with circuit delays and clock-crossing synchronizers.

Figure 1.4: In traditional design flows, there is tremendous constraint redundancy: IP level, block level, top level, synthesis and STA might rewrite the same constraints each time. Similarly, each tool may have its own design exploration flow, requiring designers to manually verify that intents are consistent across tools. BSG Pearls uses Surelog to provide tool-agnostic design annotations that can be easily consumed by all tools in the flow.

1.2 BSG Pearls: Effortlessly Synthesizable Building Blocks

1.2.1 Introduction

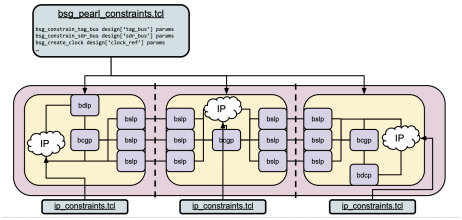
Agile Hardware Design methodologies rely on aggressive reuse of components to reduce iteration time. There is a broad understanding that more component reuse leads to less time spent on design and verification. In order to achieve practical reuse of components, designers employ latency-insensitive design principles [56]. However, latency-insensitive design principles alone do not address the challenges of hierarchical design, where components are integrated into larger systems. Although handshake-based protocols such as ready-valid provide logical guarantees of correctness at the RTL level, they have implicit dependencies on the timing and structure of the design. For instance, if timing is worsened in one latency-insensitive component, adjacent components may fail timing, or degrade in power or area.

Hierarchy-Insensitive Design is based on the insight that the design flow can

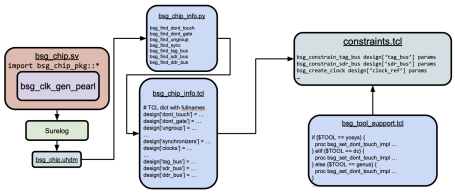
be made more predictable and manageable by making opinionated decisions about the architecture of components and their integration. This thesis presents BSG Pearls: a collection of middleware components that enable rapid and safe iteration of mid-sized SoC designs, as well as demonstrating the hierarchy-insensitive *Clamshell* design methodology for isolating immature components from the rest of the design flow. Clamshell complexes are design assemblies that are designed to surround immature components, such as accelerators, with hardened interfaces to provide predictable design closure. Changes within the *Clamshell* complex are isolated from the rest of the design, allowing for rapid iteration and experimentation without affecting other blocks. Additionally, because boundaries are consistent across levels of hierarchy, designers are able to reuse complex constraints throughout the design flow, reducing collateral management overhead and reducing the risk of unexpected design impacts. By providing reliability at a higher level of abstraction, BSG Pearls reduces the likelihood of design flow errors, allowing engineers to explore innovative solutions with higher confidence.

1.2.2 Collateral Reuse in Hierarchical Designs

RTL is only one part of the design flow, along with synthesis constraints, place-and-route guidance and verification scripts, commonly referred to as "collateral". This collateral is tightly coupled not only to the specific hierarchy of the design, but also to tools or even the state observed at an intermediate stage of the design flow. Although there is correspondence between the original design and the collateral, this fragility means that in practice new scripts are often created for each new design, each stage of the backend and even each iteration of the toolflow. Managing this collateral can be a significant burden, especially for mid-sized SoC designs where the number of components and their interactions can grow rapidly. For small teams, this



(a) Collateral generation reduces the manual IP integration and verification effort. The benefits are greatest in Clamshell architectures, where only BSG Pearls interfaces are exposed outside of the hierarchy, providing maximal design isolation.



(b) BSG Pearls generate collateral from a single source of truth that can be used in a variety of tools. By outputting parameterized artifacts, manual effort (and corresponding mistakes) are minimized.

Figure 1.5: While alt-hdls such as Chisel [29] accelerate generation of simulation and synthesis-capable Verilog, designers manually construct SDC contraints and layout files for a specific parameterization.

can lead to situations where design intent is silently lost between handoffs, fixes are dropped, and bugs go uncaught (or are introduced) late in the design flow.

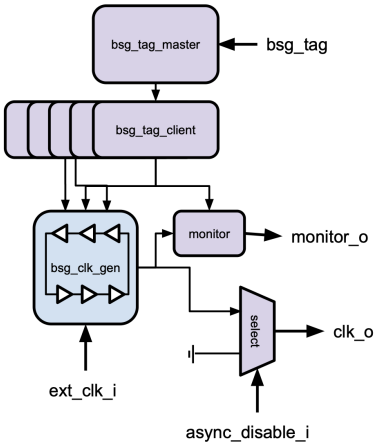
Compared to software, unit testing in hardware is more complex and laborious. One reason is that hierarchical designs necessitates testing at multiple levels of hierarchy. Unlike in software, where a single function can be tested in isolation, hardware components have complex spatial and sequential dependencies that cannot be easily mocked. Faithfully simulating an entire design can be overly resource-intensive, while modifying the environment to speed up testing can lead to false bugs or regressions. To avoid this, designs replicate the same tests at multiple levels of hierarchy. For each level, the testbench must be adapted to the specific design interface, potentially requiring new drivers, checkers, and assertions. This can lead to significant overhead in managing testbenches, as well as increased risk of bugs due to inconsistencies between hierarchical testbench environments.

Just as Clamshell assemblies provide common constraints, designs which leverage BSG Pearls on the boundary can reuse the same testbench drivers and checkers across the hierarchy. Because bsg_tag and bsg_link interfaces are source-synchronous, latency-insensitive and synthesizable, they can be used in RTL simulation, annotated gate-level simulation, and even post-silicon FPGA emulation. Testbench traces for configuration, send data and check data can be synthesized into ROMs and reused.

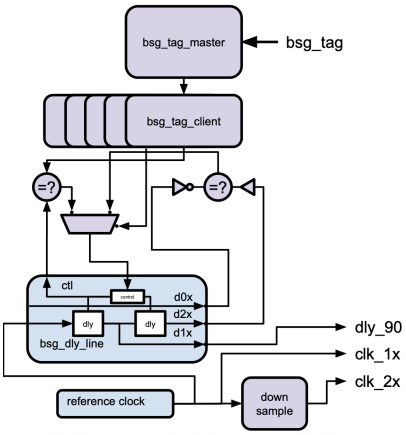
1.2.3 BSG Pearls Library

The initial release of BSG Pearls is designed to provide a minimal set of functionality sufficient for an accelerator test-chip. Every design needs a clock, reset, static configuration, and some form of communication between components. Figure 1.6 shows the set of initial pearls that are available in the BSG Pearls library. Pearls are intended to be hardened early in the design flow, so they have minimal parameters that must be set. When possible, design settings are provided by the bsg_tag configuration bus interface which allows for static reconfiguration without requiring resynthesis. Communication between pearls is provided by latency-insensitive, source-synchronous links, which allow for decoupled communication between components. Because different IPs often have different requirements for clock and reset, asynchronous FIFOs provided a reliable way to buffer mismatched consumers and producers. Pearls are designed to be hierarchy-insensitive, meaning that they can be used in any level of hierarchy without requiring changes to the design flow or collateral.

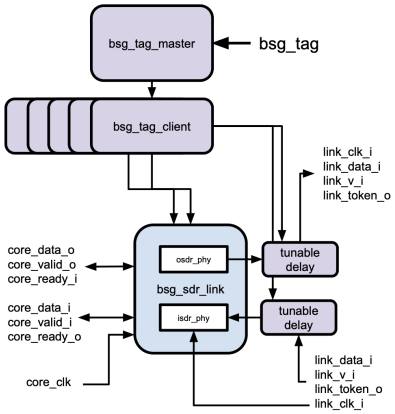
BSG Pearls also provides a sample set of Clamshell complexes that can be used as a starting point for designs. These complexes are designed to demonstrate assembly of BSG Pearls components into larger systems. Figure 1.7 shows the two initial complexes that are available in the BSG Pearls



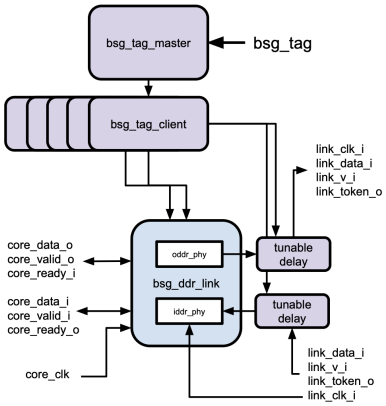
(a) bsg_clk_gen_pearl is a fully-digital, ring-oscillator-based clock generator, glitchlessly tuneable at runtime, as well as switcheable between multiple clock sources: raw oscillator, downsampled frequency, external clock, or static.



(b) bsg_clk_dly_pearl is a fully-digital delay line which can automatically produce common clock variations: 90 degree phase shift, inversion, or 2x downsample. These variations are common to I/O as well as LPDDR interfaces.

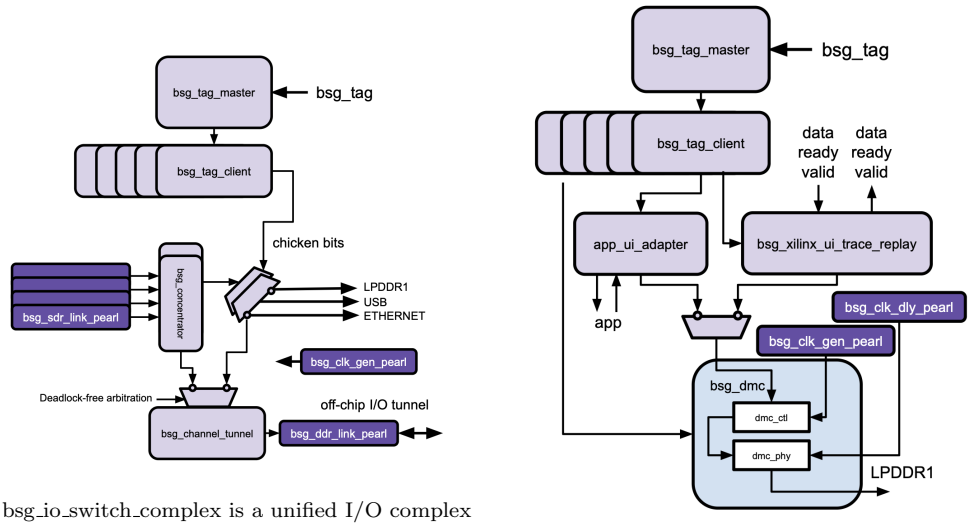


(c) bsg_sdr_link_pearl is a fully-digital source-synchronous link for single-data-rate (SDR) interfaces. It provides local ready/valid while using credit-based global flow-control to connect two IP blocks with decoupled communication.



(d) bsg_sdr_link_pearl is a fully-digital source-synchronous link for double-data-rate (DDR) interfaces. It provides local ready/valid while using credit-based global flow-control for off-chip I/O with low clock frequency.

Figure 1.6: The four canonical BSG Pearls for IP communication. Using these pearls, designers can build a wide variety of hierarchy-insensitive IP blocks. This powerful abstraction enables the rapid construction of large complex SoCs with minimal coordination between subteams.



(a) bsg_io_switch_complex is a unified I/O complex that provides a reliable tunneled interface to an off-chip bridge. It tunnels bidirectional SDR links (representing I/O and memory) over a single DDR link. Arbitration is run-time configurable, allowing independent redirection to on-board I/Os or off-chip emulation.

(b) bsg_dmc_phy_complex is an all-in-one DDR controller, PHY and tester. Built on top of bsg_clk_dly_pearl, it provides a fully digital interface to the PHY with reliable and tunable configuration. Testers can send cycle-level traces to the PHY, which can be used to verify timing and functionality.

Figure 1.7: bsg_pearl complexes provide silicon-validated examples of hierarchy-insensitive designs that leverage BSG Pearls. SoC designers can use these complexes directly or as a starting point for their interfaces.

library: the bsg_io_switch_complex and bsg_dmc_phy_complex complexes. The bsg_io_switch_complex complex provides a reliable tunneled interface to an off-chip bridge, while the bsg_dmc_phy_complex complex provides an all-in-one DDR controller, PHY and tester. Both complexes have been silicon-validated and refined based on real-world tapeouts, making them a reliable starting point.

1.2.4 Related Work

BaseJump STL

BaseJump STL [235] provides a collection of reuseable, latency-insensitive hardware components. Compared to BSG Pearls, BaseJump STL is focused on generic components such as FIFOs, arbiters, memories and other basic building blocks. BaseJump STL is focused on providing functionality that is common to all users, much like STL does in C++. While BaseJump STL provides primitives as well as portability layer to abstract technology-specific details, BSG Pearls builds on top of BaseJump STL to provide a higher level of abstraction and functionality. By making opinionated decisions about the design of components, BSG Pearls is able to provide a more predictable and consistent design flow while maintaining flexibility and efficiency.

Pulp Platform

Pulp Platform [206] Common Cells provide a collection of RISC-V cores and peripherals, including AXI interconnects, standard library cells and other common components. As BlackParrot depends on BaseJump STL, designs such as CVA6 [273] leverage the Common Cells in addition to design-specific components. However, the library does not generally provide collateral for synthesis, place-and-route, or verification. Testing is generally limited to component-level verification and cannot be reused during system integration tests. Additionally, Pulp Platform Common Cells are not designed to be hierarchy-insensitive, and thus do not provide the same level of tool predictability as BSG Pearls.

ChipYard

ChipYard [12] is an integrated SoC research platform supporting chip design, simulation and implementation. While ChipYard provides interfaces to a variety of ASIC design tools and libraries, it does not ascribe any particular design methodology. Generally, ChipYard is designed to be a more general *all-in-one* SoC design platform, while BSG Pearls is focused on validating subcomponents to be integrated into external designs. Additionally, ChipYard components tend to be more monolithic, like an entire RocketChip. Of course it is entirely possible to use BSG Pearls components within ChipYard, although the ChipYard flow is not hierarchy-insensitive by default. BSG Pearls works with standard Python, SystemVerilog, and TCL, while ChipYard is built on top of the Chisel hardware description language [28].

Hammer

Hammer [251] is an open-source ASIC physical design flow that provides a set of wrapper scripts for synthesis, place-and-route, and verification. Its modular design allows for easy integration of a variety of CAD tools and scripts, separating out RTL integration, toolflow configuration, and PDK specialization. BSG Pearls RTL, synthesis collateral and test methodologies can be easily integrated into the Hammer flow. Using the Clamshell methodology with Hammer would reduce iteration time as well as with traditional flows.

Other Design Methodologies

Globally Asynchronous, Locally Synchronous (GALS) [56] design methodologies have been proposed to address the challenges of hierarchical design. However, these methodologies typically focus on the synchronization and communication between different clock domains in order to increase perfor-

mance or lower power consumption. More radical design methodologies such as the fully asynchronous circuits used in the Illiac II [49] can achieve even greater efficiency and performance. In contrast, BSG Pearls focuses on the predictability and ease of integration of components within designs that traditionally would use only a single clock domain. This approach allows for a more straightforward design flow while managing the complexity of hierarchical designs. Prior works [156, 75, 29, 168] have proposed language extensions to support property generation as well as testbench generation. However, these approaches do not support a general methodology for hierarchy-insensitive design using standard tools.

1.3 NoC Symbiosis: Lessons Learned from Agile Tapeouts

This section has been adapted from NOCS 2020 "NoC Symbiosis" [188].

1.3.1 Abstract

Conventional wisdom states that Network-on-Chip router area grows quadratically with the channel width, and this perception has fundamentally shaped the assumptions of thousands of NoC papers that have been written to date, and many chip designs. However, this assumption is not entirely true. Simple analysis and empirical data from this paper shows that, in modern standard cell technology, a router's *standard cell logic area* actually grows only linearly; it is solely the *wire routing area* that grows quadratically.

If we think of a NoC as a standalone block as is done in standard hierarchical VLSI design, then the overall area growth is indeed quadratic. But this approach either vastly under-utilizes logic area, or, in designs that match wire and logic area, leads to small network links. At the same time, many standard non-NoC logic blocks like processors or accelerator blocks typically

use the standard cell logic area but need only a fraction of available wiring resources.

We propose an alternative approach, *NoC Symbiosis*, in which router logic and the node logic it services are jointly placed together. The router absorbs excess wiring resources from the node logic, and the node logic absorbs excess standard cell area from the router. Current-day automatic place and route (APR) tools already automatically distribute the router logic across the node logic, in order to provide enough space for the wiring resources. With this approach, future SoC's can leverage vastly larger amounts of wiring bandwidth than ever before, or alternatively, reduce the area overhead of existing routers.

We describe how we first encountered this phenomena, perform experiments to demonstrate its behavior, and provide design tips to help teams realize the potential of NoC Symbiosis.

1.3.2 Introduction

Network-on-Chip (NoC) is a common design pattern in modern SoCs, which scales to hundreds or thousands of nodes in a single chip while maintaining performance. They are a widely-used workhorse of multicore processor architectures [249, 220, 185, 34, 209, 255, 37], parallel architectures [198], and accelerators [141, 97, 226, 76, 62, 81].

Common practice envisions NoC routers as standalone blocks in SoC architectures. Network components, wiring, and other architectural logic can be designed independently of the architectural logic that they connect to and facilitate data transport to and from, then be implemented inside of independent bounding boxes, and replicated across a floorplan.

This separation of concerns in design practices is widely modeled in the many

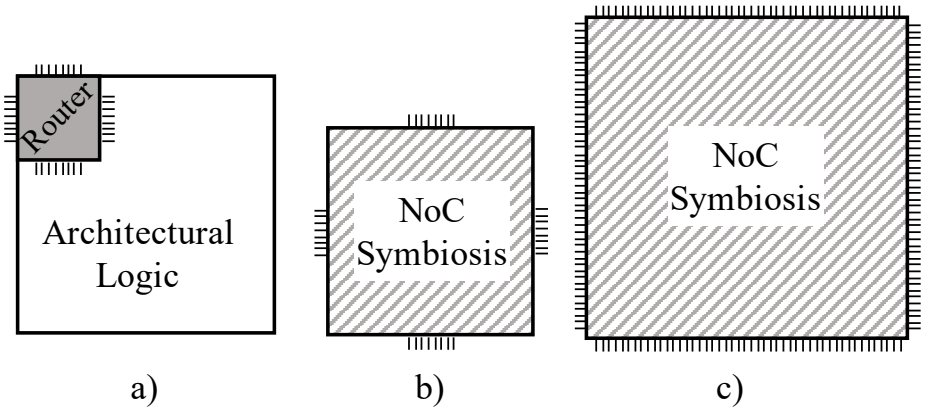


Figure 1.8: Conventional design practices suggests that NoC and architectural logic should be placed and routed in separate hierarchical boxes. We propose an alternate approach, *NoC Symbiosis*, where the components are placed and routed together to save area for the same bandwidth, or to allow greatly enhanced bandwidth and throughput with minimal area impact.

tools that estimate NoC area, power, and performance metrics [133, 132, 130, 216, 31], and many papers that analyze NoC area and power [208, 80, 204, 31].

Because of this practice, conventional wisdom states that Network-on-Chip router area grows quadratically with the channel width, and this perception has fundamentally shaped the assumptions of thousands of NoC papers that have been written to date, and many chip designs. However, this assumption is not entirely true. Simple analysis and empirical data from this paper shows that, in modern standard cell technology, a router's *standard cell logic area* actually grows only linearly; it is solely the *wire routing area* that grows quadratically.

If we think of a NoC as a standalone block as is done in standard hierarchical VLSI design, then the overall area growth is indeed quadratic. But this approach either vastly under-utilizes logic area, or, in designs that match

wire and logic area, leads to small network links. At the same time, many standard non-NoC logic blocks like processors or accelerator blocks typically use the standard cell logic area but need only a fraction of available wiring resources.

We propose an alternative approach, *NoC Symbiosis*, in which router logic and the node logic it services are jointly placed together. The router absorbs excess wiring resources from the node logic, and the node logic absorbs excess standard cell area from the router. Unlike in the past, current-day automatic place and route (APR) tools can now automatically distribute the router logic across the node logic, in order to provide enough space for the wiring resources. With this approach, future SoC's can leverage vastly larger amounts of wiring bandwidth than ever before, or alternatively, reduce the area overhead of existing routers.

These design studies estimate NoC area, power, and performance in isolation. Isolated network architectures can produce inefficient layouts that under-utilize the silicon area even when other quality metrics are good. This phenomena occurs because the required perimeter of the router bounding box scales faster than the internal cells and causes the router area to be under-utilized. Strict hierarchical design flows can prevent this lost area from being recovered by non-network architectural logic. This yields a sub-optimal performance, power, and area result when a network topology is integrated with architectural logic late in the design flow.

The concept of *NoC Symbiosis*, is illustrated in Figure 1.8. NoC Symbiosis exists when the bounding box that contains a network component, such as a router, and non-network architectural logic is smaller than the sum of the area of their independent bounding boxes. When NoC Symbiosis exists, implementing each component separately can produce sub-optimal QoR and

lengthen design iterations.

In this paper, we describe our experiences with NoC Symbiosis. We have seen NoC Symbiosis in our own work and have designed to exploit it. We also perform experiments in a 12 nm process, sweeping across a variety of network topologies, widths, and wiring constraints. These experiments explore the parameters that determine when NoC Symbiosis is possible, and expose the benefits of leveraging and the costs of ignoring symbiosis. Our experiments show that other architectures could benefit from a Symbiosis-aware design flow and we provide a methodology to recognize, leverage, and model the behavior of Symbiosis.

The remainder of this paper is organized as follows: In Section 1.3.3 we describe the concept of NoC Symbiosis. In Section 1.3.4 we relate where we discovered NoC Symbiosis. In Section 1.3.5 we use our experience to develop experiments that demonstrate when NoC Symbiosis occurs. In Section 1.3.6 we distill our findings into a set of concrete design principles and we outline how NoC Symbiosis can be realized in future work. We conclude in Section 1.3.7.

1.3.3 Network-on-Chip Symbiosis

We introduce the concept of *Network-on-Chip Symbiosis*. Symbiosis is defined as a cooperative relationship between two dissimilar organisms. NoC Symbiosis occurs when the network components of a chip are placed and routed in conjunction with dissimilar logic, producing better Quality-of-Results (QoR) than independently placing and routing each component within its own bounding box.

Figure 1.8 demonstrates two applications of NoC Symbiosis. Instead of (a) instantiating a NoC router and non-network architectural logic separately, merging the logic can unlock two opportunities. First, as shown in (b), the

total area can be potentially shrink. Second, as shown in (c), for surprisingly small incremental area, a designer can greatly increase bandwidth into and out of the tile, whether by widening existing channels or add more NoC channels. This bandwidth can be applied to decreasing congestion, reducing serialization latency, or adding new traffic classes or functionality (e.g. how about security or QoS features?) in a NoC.

Figure 1.9 illustrates why the bounding box for a 2D mesh router becomes under-utilized as the network link width increases. For a small network link width, the area of the bounding box (A_{BB}) is determined by the area of the router cells A_{SC}, which scale linearly with the network link width (N). As the network link width increases, the design reaches an inflection point where the bounding box is determined by the number of wires and the required wire pitch forces the perimeter to expand, quadratically growing the bounding box area (A_{BB}).

The relationship for router bounding box area in Figure 1.9 is summarized in Equation 1.1:

$$A_{BB} = max(\frac{A_{SC}}{U}, (D * N * P)^2) \tag{1.1}$$

Where A_{BB} is the estimated area of the bounding box, A_{SC} is the sum of the area of the synthesized standard cells, U is the target area utilization of the bounding box, D is the duplex factor, N is the number of network link width and P is the effective wire pitch. In this paper we analyze full-duplex networks, so D is 2.

NoC Symbiosis is possible when the router bounding box is larger than the area required to fit its standard cells at a given utilization. This occurs when $(D * N * P)^2 > \frac{A_{SC}}{U}$. When this holds, the bounding box area scales quadratically with the number of wires, and the area becomes underutilized.

In this Symbiotic Region, combined place and route of the network and non-network components will yield a better QoR.

1.3.4 NoC Symbiosis in the Wild

We first encountered NoC Symbiosis in BlackParrot (BP): a tiled, cache-coherent, Linux-capable RISC-V multicore introduced in [185]. BlackParrot implements the RISC-V 64-bit RV64G ISA, which includes the integer (I), multiplication and division (M), atomics (A), as well as single and double precision floating-point (F/D) instructions. It supports three privilege levels—machine, supervisor, and user-as well as SV39 virtual memory. These extensions are sufficient to run a full-featured operating system such as Linux. Development efforts have prioritized the use of intentional interfaces, modularity, silicon validation as first-order design metrics. Rather than employing a bus-based architecture, high-efficiency NoCs are used to provide scaling and flexibility so that BlackParrot can service a wide variety of design points. The three BlackParrot NoCs are the Coherence Network, I/O Network and Memory Network. For further implementation details, see [185].

We describe the evolution of BlackParrot across three versions, illustrating distinct points on the NoC Symbiosis spectrum. We then discuss implications of NoC Symbiosis on a variety of other tiled architectures which were not explicitly designed to be Symbiotic. These systems are summarized in

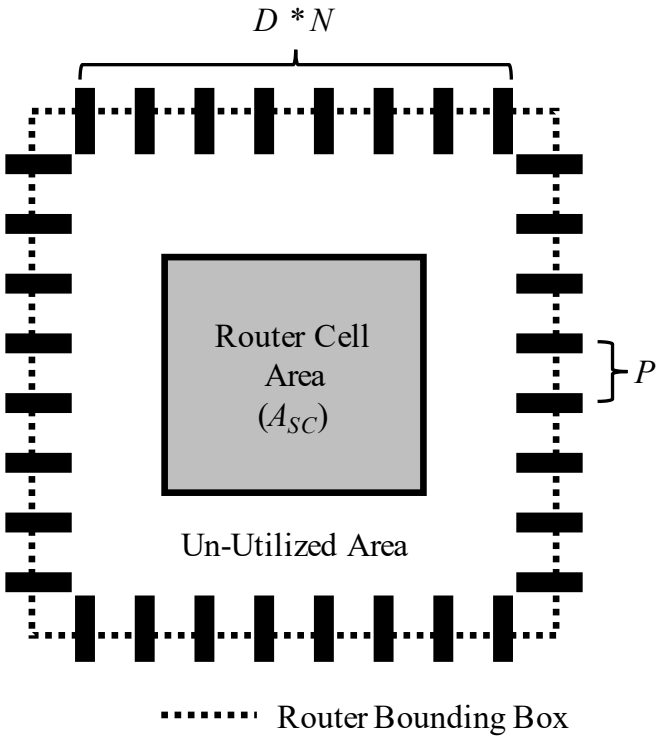


Figure 1.9: In a NoC router, conventional wisdom holds that the area of the router bounding box (A_{BB}) scales quadratically with the number of wires on an edge ($N * D$). Instead, we demonstrate that the router cell area (A_{SC}) scales linearly with the network link width, while the bounding box perimeter scales linearly with the number of wires and the wire pitch (P). As the number of network link width increases, the area of the bounding box will first grow linearly with the router cell area, until perimeter requirements force the area to grow quadratically. Beyond this point, the bounding box will become underutilized by the router cells and provide an opportunity for NoC Symbiosis.

Table 1.1.

	Tilera [255]	**Raw** [219]	**OpenPiton** [34]	**HammerBlade** [200]	**EMM** [209]	**BP v0** [39]	**BP v1** [40]	**BP v2** [185] [41]
Process Node	90nm	180nm	32nm	12nm	45nm	40nm	12nm	12nm
Tile Area (mmˆ2)	9.6	16	1.175	0.025	0.784	0.832	0.360	0.360
Est. 2X Wire Pitch (nm) [1]	540	1080	192	128	270	240	128	128
Networks	5x34b	4x34b	3x66b	1x56b 1x97b	6x66b	1x130b 2x578b 2x642b	5x66b	3x98b 2x130b (Half-Duplex)
Max Wires Per Side	340	272	396	300	792	5140	648	848
Effective Link Width	170	136	198	150	396	2570	324	424
Pin Utilization [2]	5.9%	7.3%	7.0%	24.3%	24.2%	90.2%	13.8%	18.1%

Table 1.1: Network specifications for various tiled architectures. Tilera and Raw are early examples of tiled manycores. OpenPiton and HammerBlade are more recent projects focusing on massive parallelism. The Execution Migration Machine is a tiled multicore without support for instruction-granularity thread migration. BlackParrot is a tiled, cache-coherent, application-class multicore.

[1] All 2X wire pitches are estimated as 6x node size, except for 12nm [257].

[2] All pin utilizations are with DLDT pins except for BP v0, which used 3 pin layers.

BlackParrot v0

BlackParrot v0 was designed with 5 wide network links that could shuttle entire 512-bit cache lines in a single cycle among cores and coherence directories, resulting in 4-10x greater bandwidth than contemporary NoCs in Table 1.1, as well as 6-8x shorter serialization latency compared to BlackParrot v1 or v2. In addition to reducing memory access latency, this quick inter-tile communication provides order of magnitude speedup for coherence operations, which can cripple heavily cooperative multi-threaded programs. Overall, the design priorities were functionality and performance rather than power or area.

We first attempted to tape in BlackParrot v0 as part of a VLSI course. Following conventional practice, we partitioned the physical design of the core into the Front End, the Back End, and the Memory End. Each portion was

allocated to a different student. The Front and Back Ends quickly converged. The Memory End, with NoC routers, was unable to route without DRC errors and had incredibly low area utilization. Students were extremely unhappy with their assignment. At that point, we started to realize how insanely provisioned the NoC was!

We discovered NoC Symbiosis when a postdoc on our team mentioned that he had no problems placing and routing BP. It seemed impossible. But he, taking the easy route and wanting to avoid partitioning, had placed and routed the entire tile, instead of the individual components. The tools easily routed the entire tile, with a *smaller* bounding box than the three components implemented separately. Figure 1.8 depicts this phenomenon. This discovery impacted our internal tape-in builds, and led to the subsequent development of BlackParrot v1 and the study of NoC Symbiosis.

BlackParrot v1

While BlackParrot v0 reaped latency benefits from its wide NoCs, without a prefetcher or multithreaded memory system the core could not provide enough requests to use a reasonable portion of the average bandwidth available. As a relatively small core, it simply was not powerful enough to take advantage of the Symbiotic potential of 2500+ bits per cycle of bandwidth. Additionally, several other nodes on the NoC were incapable of sinking a full cache line per cycle, requiring excessive buffering storage and serialization penalties. To solve these issues, BlackParrot v1 consolidates the on-chip networks, reducing the network link width (and bandwidth!) using wormhole routers. This reduced the aggregate network link width to 324, which easily fit the router into dense core tiles.

BlackParrot v1 is shown in Figure 1.3.4. Wormhole routing decreased the

link width significantly (at the cost of increased serialization latency and decreased bandwidth). The v0 I/O Network separated into two new networks: a 1D I/O network which passes through special purpose I/O tiles at the top of the chip, and a Memory Network, which is a half-duplex 1D network that flows from each core tile to the on-chip DRAM controller. Decomposing the 2D I/O network into 1D networks results in dramatic PPA benefits, as the router crossbars shrink from 5 ports to 3.

The changes in v1 moved the design to the edge of the Symbiotic Region described in Equation 1.1. While this resulted in PPA savings, BlackParrot v1 loses the advantages that v0 enjoyed from NoC Symbiosis. In BlackParrot v2, our current version, we leverage the implications of NoC symbiosis with a design that lies between v0 and v1.

BlackParrot v2

We developed BlackParrot v2 with additional parameterization to explore the NoC symbiosis design space. This version introduces additional RTL parameters to control the size of the core logic and router clocks, in addition to the wormhole routers introduced in v1. This allows the design to be tailored to particular application bandwidth or performance requirements. In particular, the parameterizations provided by v2 allow BlackParrot developers to compose its NoCs anywhere between area-optimized v1 and maximum-performance v0 configurations.

The Coherence Network widened so that an entire packet header fits in a single flit. This eliminates serialization latency from payload-free packets, such as loads and acknowledgements. The Memory networks expanded to a half-duplex 128b to reduce serialization latency while filling an L2 cache line.

Despite increasing the number of ports on each side of the tile by 70%, there

was negligible impact on utilization. NoC Symbiosis allowed us to adjust system level bandwidth requirements with trivial backend adjustments.

OpenPiton

Notably, BlackParrot is not the only existing architecture to leverage NoC Symbiosis. OpenPiton [35] is an open-source manycore research platform that also leverages NoC Symbiosis principles at the tile level. The OpenPiton L1.5, L2, and NoC modules are designed to be flattened into the tile. From this, we observe that NoC Symbiosis is not limited to a specific core architecture and can be applied as a general design principle.

Summary

Future iterations of BlackParrot will account for the extra flexibility that NoC Symbiosis grants, allowing for surprisingly wide topologies and router configurations. While BlackParrot has encountered NoC Symbiosis in a variety of designs, we will demonstrate in the next section that it is a general concept applicable to nearly all modern SoCs.

Symbiosis Gallery

Table 1.1 shows the Symbiotic capabilities of a variety of tiled architectures. Most designs do not take advantage of NoC Symbiosis, and are either firmly in the Standalone Router Region or near the inflection point (see Figure 1.10).

One reason designers may not have leveraged NoC Symbiosis in the past may is a lack of tooling support. For instance, Raw and Tilera were designed before timing-driven placement, so most blocks painstakingly positioned by hand. Such a flow does not allow for counter-intuitive experimentation like distributing router cells throughout the tile. This disconnect helps explain

the relatively low pin utilization in these designs.

HammerBlade is a massively parallel manycore focusing on ML and graph applications, which will happily consume any available bandwidth. The Execution Migration Machine on the other hand aims to reduce on-chip bandwidth consumption of tiled manycores, instead migrating threads to cached data. However, its novel protocols employed many separate physical networks to ensure deadlock avoidance, resulting in a moderate pin utilization.

The remaining systems are all cache-coherent multicores: systems which tend to be latency-constrained rather than bandwidth-constrained. As opposed to EMM which requires complete transfers of state before resuming execution, techniques such as critical word first cache fills can alleviate serialization latency penalties incurred by narrower network links. As an aggressively scalable manycore, OpenPiton prefers narrower network links, allowing a designer to pack more small tiles into a chip. BlackParrot targets moderate scalability, between 1-16 cores, and so provides a wider range of pin utilization configurations, optimizing by specializing networks.

BlackParrot v0 is a notable outlier. It is a coherent multicore, yet it uses over 90% of its maximum pinout. As described above, cache line wide links allow for extremely low latency coherence operations. While conventional wisdom dictates that such large routers would be infeasible, we demonstrate that NoC Symbiosis enables this design space without any significant manual effort.

1.3.5 NoC Symbiosis in Captivity

In this section we study NoC Symbiosis through a series of directed experiments. These experiments demonstrate the parameters where NoC Symbiosis exists and show how to recapture the un-utilized area that NoC Symbiosis provides.

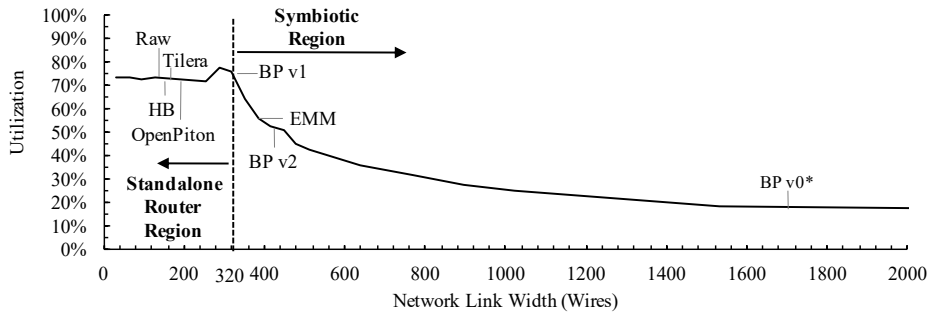


Figure 1.10: Plot of router bounding box area utilization ($\frac{A_{SC}}{A_{BB}}$) as a function of DLDT network link width (N). Design points are annotated for related work in Table 1.1. The utilization is initially constant, but when the bounding box becomes wire-limited the utilization decreases proportional to the inverse square, contrary to conventional wisdom. The annotations illustrate design points along the NoC symbiosis spectrum in Table 1.1. * BlackParrot v0 actually used way more wires but on three layers; we normalize to DLDT.

We perform three experiments: First, we measure the relationship between network width, and bounding box utilization to demonstrate verify our hypothesis in Figure 1.9. Next, we determine how much of the un-utilized area in the bounding box can be reused by dissimilar logic to verify that the un-utilized area can be recycled. Finally, we examine the interaction between the number of networks, aggregate network width, and router area utilization to demonstrate that our experiments generalize to more than one network.

For our experiments we use silicon-validated wormhole routers from BaseJump STL[235]. Wormhole routers allow us to make simplifying assumptions about the physical design space and remove confounding variables, such as header width and payload size. The router is a 2D mesh router with 5 ports (North, East, South, West, and Local). Our network is input buffered, and stores two flits per router direction. This allows us to remove the amount of FIFO

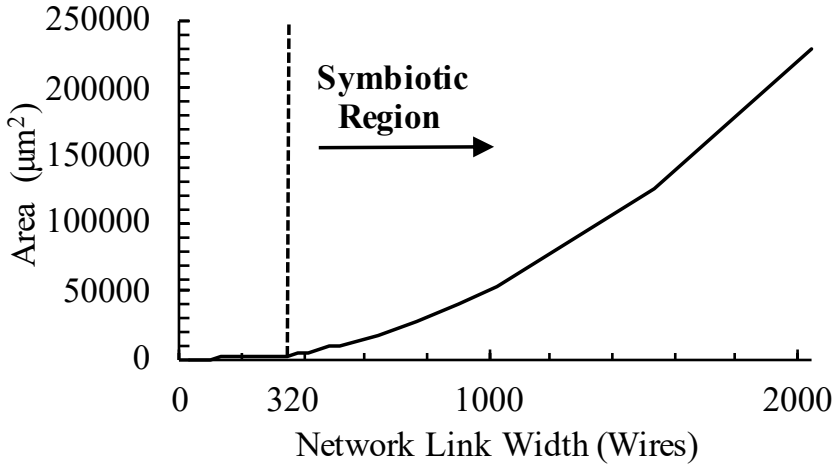


Figure 1.11: Plot of un-utilized bounding box area ($A_{BB} - A_{SC}$) as a function of DLDT network link width (N). The available area is initially approximately 0, and remains constant as N increases, but as the width increases the un-utilized area begins to grow quadratically when it reaches the Symbiotic Region.

buffering from our parameter space. The cardinal directions are routed to the edge of the bounding box.

For each experiment we place and route the wormhole router inside of a bounding box determined by Equation 1.1 and an aspect ratio of 1:1. We set the target utilization (U) to 80% and target a multi-corner 800 MHz operating frequency. To emulate the routing environment of a large SoC, we configure input and output delays for an 825 ps arrival time.

Our experiments are performed using the 12nm GlobalFoundries PDK. We use Design Compiler O-2018.06-SP4 to perform synthesis and IC Compiler II O-2018.06-SP5 to perform place and route.

Utilization vs. Network Link Width

To demonstrate why NoC Symbiosis arises, we first study how the utilization of the router bounding box scales as the network link width increases. This experiment aims to confirm our hypothesis in Figure 1.15. We place a router within a hierarchical bounding box and sweep the network link width (N) from 32 to 2048. We measure and plot the utilization of the bounding box ($\frac{A_{SC}}{A_{BB}}$), and the un-utilized area ($A_{BB} - A_{SC}$).

We examine two wire pitch configurations for network link wires to demonstrate how P from Equation 1.1 affects symbiosis. First, we space alternating metal layers two tracks apart (Double-Layer-Double-Track, or DLDT). This strategy is the default for newer process nodes where routing is plentiful, as it approximates a single track spacing while preserving signal integrity at increased density. Older nodes with fewer metal layers may have fewer options for accommodating NoC pins. For this case, we analyse layouts using a single pin layer with double track spacing (Single-Layer-Single-Track, or SLDT). Switching from DLDT to SLDT has the effect of doubling the value of P.

Utilization in DLDT

Figure 1.10 plots the utilization ($\frac{A_{SC}}{A_{BB}}$) as a function of the network link width (N) for DLDT. The data show that there is a range of network widths where the utilization decrease is linear, followed by a quadratic decrease. This inflection point is highlighted in the plot at a network link width of 320. The region to the left is called the Standalone Router Region because this is where independent place and route of the router will not affect design QoR. The region to the right is called the Symbiotic Region because this is where symbiotic place and route of NoC router and logic components will yield

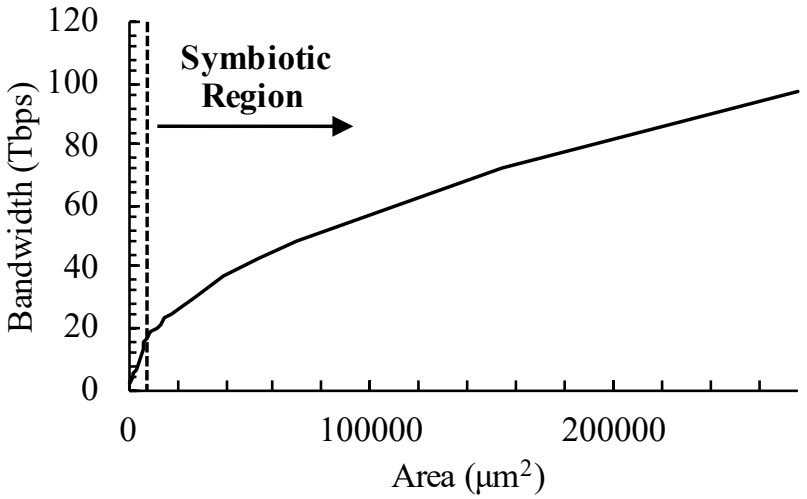


Figure 1.12: Plot of router bandwidth, as a function of bounding box area (A_{BB}), for DLDT pins. Initially, the bandwidth grows linearly with the area, but when the symbiotic region is reached, the area grows quadratically and provides diminishing bandwidth returns.

better QoR.

Figure 1.10 is annotated with network link widths that correspond to the network configurations in related work.

Figure 1.11 plots the un-utilized area ($A_{BB} - A_{SC}$) as a function of the DLDT network link width (N). In the Symbiotic Region, the un-utilized area increases quadratically as hypothesized in Figure 1.9.

Figure 1.12 shows the relationship between bounding box area (A_{BB}) and router bandwidth. The bandwidth is derived from the network link width (N) and the target frequency of 800 MHz to compute bandwidth for each area result in our experiments.

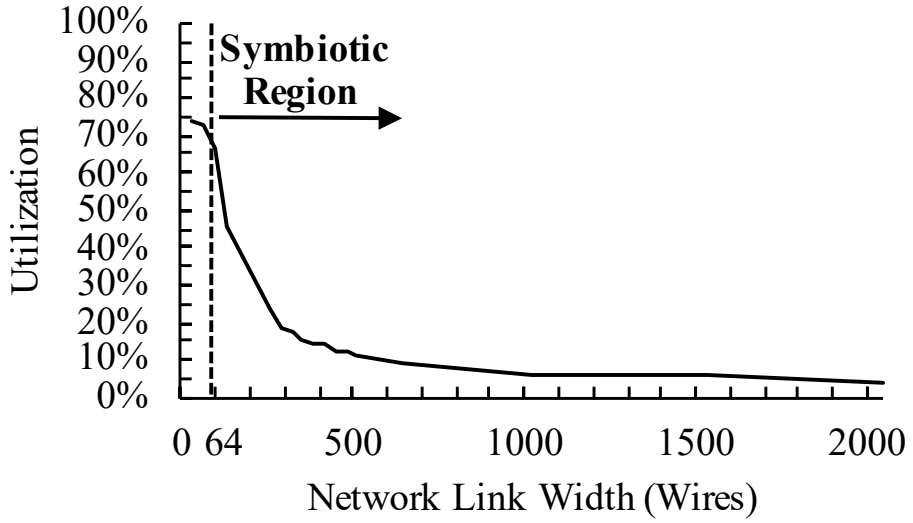


Figure 1.13: Plot of router bounding box area utilization ($\frac{A_{SC}}{A_{BB}}$) as a function of SLDT network link width (N). The utilization is initially constant but when the bounding box becomes wire-limited, the utilization decreases proportional to the inverse square and provides an opportunity for NoC Symbiosis. This transition happens 4× earlier than in Figure 1.11 (64 vs 320)

Utilization with SLDT

As described in our experimental setup, in SLDT we have only a single layer of pins, resulting in an effective pitch P that is two times that of DLDT. Since the wire pitch is larger, we expect that the Symbiotic Region will begin at $\frac{320}{4}$ network link width, according to Equation 1.1.

Figure 1.13 plots the utilization ($\frac{A_{SC}}{A_{BB}}$) as a function of the SLDT network link width (N). As in Figure 1.10, the Symbiotic Region is annotated at the place where the bounding box switches from linear to quadratic growth.

As hypothesized, this occurs with approximately $\frac{1}{4}$ compared to DLDT designs. This is particularly relevant to older designs that were implemented in previous generation design nodes with fewer metal layers.

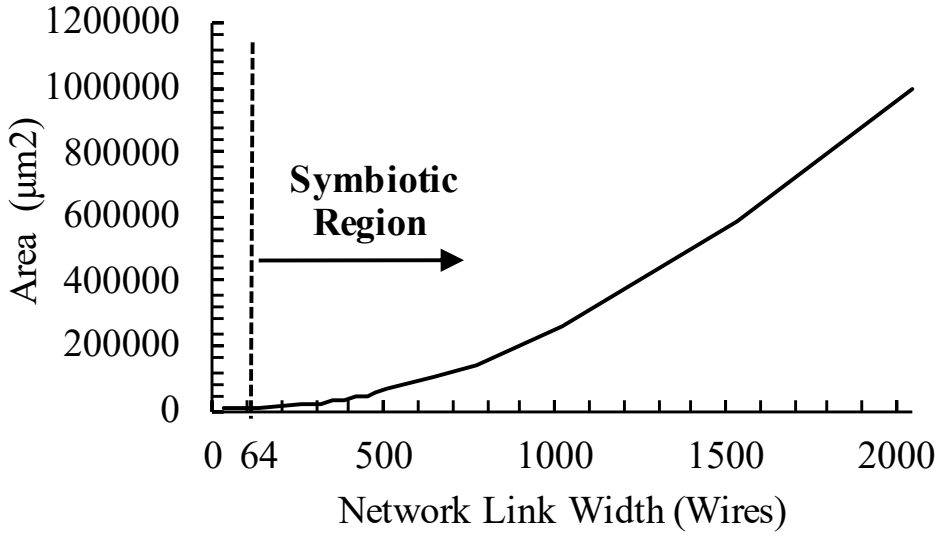


Figure 1.14: Plot of un-utilized bounding box area ($A_{BB} - A_{SC}$) as a function of SLDT network link width (N). The available area is initially low, and remains constant as N increases, but as the network link width increases the un-utilized area begins to grow quadratically and can be filled with symbiotic logic.

Figure 1.14 shows the un-utilized area ($A_{BB} - A_{SC}$) as a function of the network link width (N). It is annotated with the Symbiotic Region. As in Figure 1.13 the Symbiotic Region starts earlier. The unused area also increases more quickly, increasing to due to the increased value of P

Comparing the plots of the DLDT experiments and the SLDT experiments in the previous two sections, we can see that the wiring resources have a significant impact on NoC Symbiosis. This is important to consider when choosing a design node, metal stack, or track allocation strategy, as these decisions affect Symbiotic behavior.

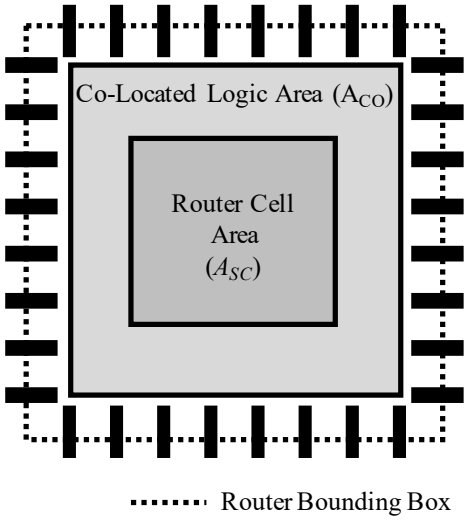


Figure 1.15: The router bounding box from Figure 1.11 with co-located logic (A_{CO}) inserted to recover un-utilized area within the bounding box.

Attainable Symbiotic Area

In this experiment we will measure the resources made available through NoC Symbiosis by attempting to recover the un-utilized area. First, we determine un-utilized area for each network link width configuration using the data from DLDT experiments in Section 1.3.5. Then, we pack the remaining cell area with co-located logic by synthesizing shift registers between the inputs and outputs of the router. We repeat the process until until the estimated utilization reaches 80%, and no DRC errors occur. We then measure and report the area of the co-located logic, A_{CO} from Figure 1.15.

Figure 1.16 records the relationship between attainable co-located logic area (A_{CO}) and network link width (N). All points outside of the symbiotic region are ignored because the router bounding box is cell limited and introducing additional logic produced DRC errors. This figure demonstrates that the Attained co-located logic area increases along with the Theoretical un-utilized

area. Thus, much of the un-utilized cell area can be recovered.

Networks vs Area

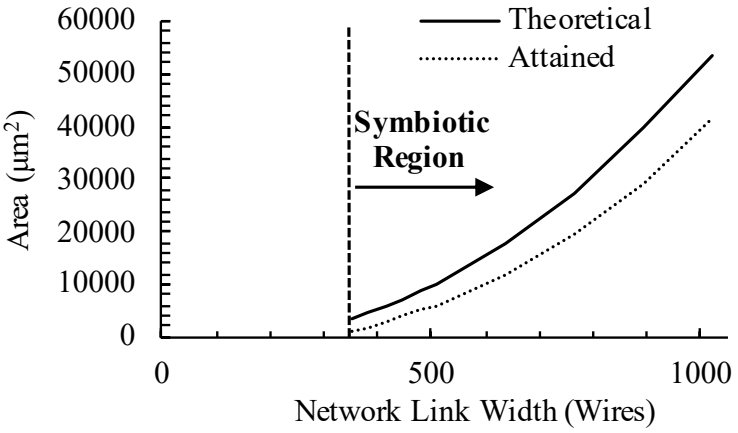
In this section we will measure how the number of networks affect the router standard cell area ($A - SC$) area as the aggregate network link width (N) increases. In the previous sections we demonstrated results for single 2D mesh router inside of a bounding box. In these experiments, we will repeat the data collection shown in the previous experiments, but we will also vary the number of routers in the bounding box.

Figure 1.17 plots the standard cell area of the routers in the bounding box (A_{SC}) as the the aggregate network link width varies from 128 to 512. The sweep is performed three times: with 1, 2, and 4 routers in the bounding box. As shown in the figure, the number of networks has no effect on A_{SC}. We conclude that *aggregate* network link width has the first order impact on physical design for wormhole routed networks.

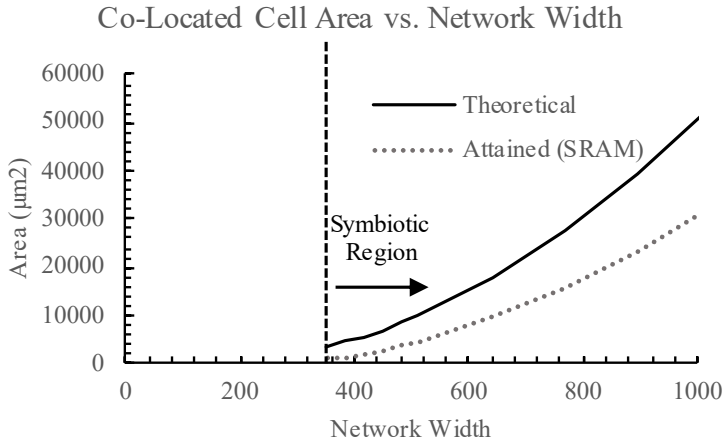
1.3.6 Symbiotic Design Methodology

In this section we summarize our experiments and experiences into a set of helpful design tips. Our goal is to help teams *Recognize Opportunities for NoC Symbiosis*; *Design for NoC Symbiosis* when it is available; and build *Models for NoC Symbiosis* that can predict symbiotic impacts early in the design cycle.

Recognize NoC Symbiosis NoC Symbiosis is possible when a network bounding block becomes underutilized. In Figure 1.10 and Figure 1.13 we demonstrated that there are many designs in, or near the Symbiotic Region. It is important to realize when a design reaches this region so that an optimal



(a) Co-located logic area vs. network link width



(b) Co-located SRAM area vs. network link width

Figure 1.16: Attained area shows the maximum co-located area that can be inserted into the un-utilized region. The theoretical limit is the un-utilized area from Figure 1.11.

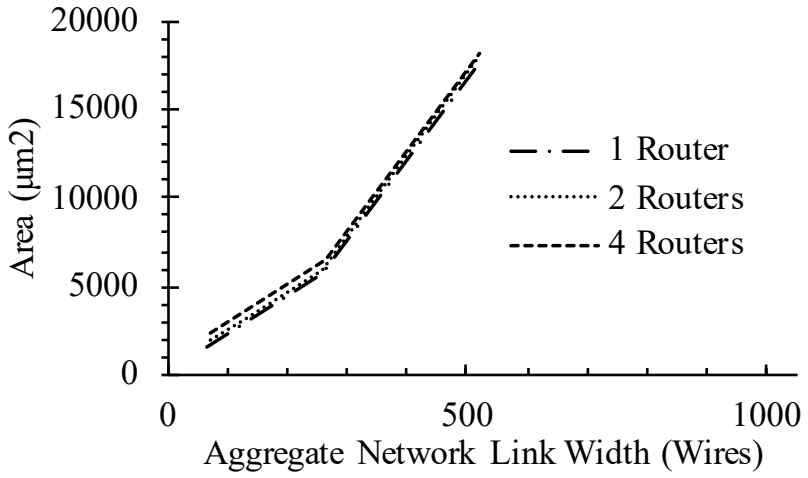


Figure 1.17: Plot of router standard cell area (A_{SC}) vs. aggregate link width for 1, 2, and 4 routers within the same bounding box. The aggregate link width is the total width of links entering the side of a bounding box, with links distributed evenly between routers, with the links distributed equally between the routers. For a given point on the X-axis, the number of links entering the box remains constant.

design is produced.

Equation 1.1, summarized below, can detect when a design enters the symbiotic region:

$$\frac{A_{SC}}{U} < (D * N * P)^2 \tag{1.2}$$

The Symbiotic Region of a specific design depends on system parameters like topology and link width as we demonstrated in our experiments. It also depends on process technology, metal stack and hierarchical strategy as we demonstrated in Section 1.3.5. Designers should be aware of these parameters and use them throughout the design process.

Design for NoC Symbiosis NoC Symbiosis can be harnessed as a beneficial design parameter to improve QoR. Our experiences in Section 1.3.4

demonstrated that symbiosis can reduce design times as well as area. As we showed in Section 1.3.5, the unused area inside of a symbiotic router can be recaptured as useful design space.

With the right techniques, designers can move a design into the Symbiotic Region and tune their design appropriately. For example, designers can employ techniques such as wormhole routing, which allows link width to be varied without disrupting the architecture.

There are several ways to move a design into the Symbiotic Region:

- Increase the router link width.
- Increase the number of networks.
- Constrain the number of routing layers.

If a design is Symbiotic but wire limited, there are many ways to tune to take advantage of the extra area including:

- Increase sizes of local memory (cache, scratchpad, etc.).
- Combine multiple tiles and attach to a single router.
- Increase the datapath width of tile components.

Some of these can be tuned without major architectural implications, or architecture-level Quality-of-Result. Designers should consider high-level architectural goals and modeling when making these decisions.

Model NoC Symbiosis Accurate estimation of chip metrics early in the design process is critical for reducing cost [129]. NoC estimation tools [133, 132, 130, 216, 31] are an essential tool for performing early estimation of chip metrics.

We believe that current generation non-parametric modeling tools [133] can be trained to estimate chip metrics in the presence of NoC Symbiosis. Previous work estimates NoC area in isolation and has not been tested with NoC Symbiosis. As we recounted in Section 1.3.5, an isolated router can produce under-utilized area. In Section 1.3.4 we recounted how it increased design time. Using our technique described in Section 1.3.5, non-parametric tools should be able to model the transition into the Symbiotic Region, and the attainable area of co-located logic.

1.3.7 Conclusion

In this paper, we introduced the concept of *Network-on-Chip Symbiosis*. NoC Symbiosis occurs when the network components of a chip are placed and routed in conjunction with dissimilar logic components to produce better Quality-of-Result (QoR) than independent place and route of each component.

We recounted our experiences designing BlackParrot [185] and how our discovery of Symbiosis affected our design methodology. We used our knowledge to build experiments that demonstrated the symbiotic design space, and how to leverage NoC Symbiosis in a design. Finally, we distilled our experiences into a set of best practices for NoC Symbiosis.

Chapter 2

BLACKPARROT

2.1 The BlackParrot Core

This section has been adapted from IEEE Micro 2020, "BlackParrot: BlackParrot: An agile open-source RISC-V multicore for accelerator SoCs".

2.1.1 Abstract

This paper introduces BlackParrot, which aims to be the default open-source, Linux-capable, cache-coherent, 64-bit RISC-V multicore used by the world. In executing this goal, our research aims to advance the world's knowledge about the "software engineering of hardware". Although originally bootstrapped by the University of Washington and Boston University via DARPA funding, BlackParrot strives to be community-driven and infrastructure agnostic; a multicore which is Pareto optimal in terms of power, performance, area and complexity. In order to ensure BlackParrot is easy to use, extend and most importantly trust, development is guided by three core principles: Be Tiny, Be Modular, and Be Friendly. Development efforts have prioritized the use of intentional interfaces and modularity and silicon validation as first order design metrics, so that users can quickly get started and trust that their design will perform as expected when deployed. BlackParrot has been validated in a GlobalFoundries 12nm FinFET tapeout. BlackParrot is ideal as a standalone Linux processor or as a malleable fabric for an agile accelerator SoC design flow.

2.1.2 Introduction

RISC-V [252] is a disruptive technology. Never before has such a large, global community worked together to put forth a complete open source instruction set, machine model, and software stack. There is a strong belief in the community that RISC-V will find their foothold as low-NRE, high performance host cores to the agile-developed specialized accelerators that are being developed to in response to the end of Dennard Scaling.

Strangely lacking, however, is a similarly globally-maintained open source *implementation* of a RISC-V SoC. BlackParrot, described in this work, is designed to fill this gap. BlackParrot is open source and available now under the BSD license. BlackParrot is written in standard SystemVerilog and hence easily integrates into existing design methodologies and is easily understood and modified by industrial designers. BlackParrot has been fabricated in the GlobalFoundries 12nm process, with several iterations of refactoring to attain high area, delay and power efficiency.

Taking lessons from software engineering and the scalability of Linux development, BlackParrot has a modular design that puts the interfaces first. We believe that this will enable scalable collaboration by allowing contributors to work independently without having to completely understand all of BlackParrot's components, or how they are evolving. We seek not only to advance the state-of-the-art in open source processor architecture, but also to develop approaches that change the way the world designs hardware.

Success Metrics

We believe adoption of BlackParrot will be driven by optimizing across four dimensions, as shown in Figure 2.1: quality, virality, functionality and efficiency.

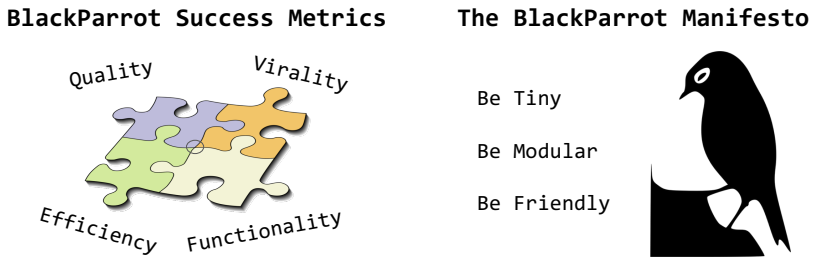


Figure 2.1: The BlackParrot Success Metrics and Manifesto. The Success Metrics strategically align the project for widespread adoption, and the BlackParrot Manifesto provides tactical guidance for technical decisions.

Quality From the beginning, the BlackParrot was architected not simply to achieve RISC-V compliance, but to produce a quality codebase that engineers could inherently *trust* as a secure and high-quality design. At the heart of this approach (which we term informally *the software engineering of hardware*) is the pervasive use of intentionally designed, narrow, modular interfaces that make the design easy to reason about without sacrificing performance, power or area. Leveraging years of processor design experience, we developed a high-level design document that partitioned the multicore into three major modules with easy-to-understand, light-weight transactional interfaces. Each module then had its own design document with specified its own internal modular interfaces, and worked through the important nuances and special cases. From there, we produced schematics and then SystemVerilog RTL which leveraged BaseJump Standard Template Library [235], an expansive set of intentionally designed interfaces for common computer architecture and hardware atoms that comes with corresponding silicon-verified SystemVerilog implementations.

The RTL then underwent extensive code review, and both unit and random testing. We are systematically measuring toggle, line, and functional coverage, and driving up the coverage of our verification methodology on a daily basis. The goal is that an experienced engineer can evaluate the documentation, code, and tests; appreciate BlackParrot's quality and use it confidently.

Virality While BlackParrot is a quality design, we realize it will ultimately be unsuccessful if is not widely adopted and if the community does not collectively take stewardship of it. For this reason, we have focused on the out-of-the-box experience, making it simple to get up and running via a github checkout, and pulling in as few external components as possible. We employ a widely known languages, SystemVerilog, instead of Chisel or BlueSpec. We have a focus on friendliness and inclusiveness in our social interactions. Too many online collaboration forums are marred by a tolerance of abusive behavior particularly by respected members of the community, as highlighted by a recent case where Linus Torvalds himself stepped away from Linux for several months to try to contemplate the toxic effects of his curmudgeonly attitude. We actively try to prevent "not invented here" syndrome from taking hold in the BlackParrot effort. Our effort welcomes your contributions and the modular nature of the design makes it easy for individual contributors to get up to speed.

Functionality BlackParrot boots Linux. It implements all of the core features of the RISC-V instruction set that are used by current software stacks. It also contains all the useful features required by modern SoCs, such as interrupt controllers, cache coherence and cache hierarchies, and easy-to-integrate accelerator interfaces.

	CoreMark/MHz (Higher better)	**Language**	**Design Schema**	**Natively Multicore**	**License**
BlackParrot	3.04	SystemVerilog	Pervasive Modular Interfaces; Standard Template Library	✓	BSD-3
Rocket	2.62	Chisel	Generator	✓	BSD-3
Ariane	2.45	SystemVerilog	Monolithic		SHL-2
Shakti	2.20	BlueSpec	Generator		BSD-3
Flute	1.00	BlueSpec	Generator		Apache-2
SiFive U54MC	3.01	Chisel	Generator	✓	Closed Source

Figure 2.2: Recent Energy/Performance optimized 64-bit RISC-V Open Source Linux-capable ASIC application class processor cores. Per-core performance is given in CoreMark / MHz as is standard in the industry. For reference we given the equivalent closed source, for-money, Linux-capable SiFive U54 core. (Note that since these open source projects are living, breathing projects, this is just a snapshot in time!)

Efficiency BlackParrot must have best-of-class PPA for its target domain; the Linux host multicore for accelerator chips. Early Coremark scores show BlackParrot achieves competitive performance with both academic and commercial cores of its class, as shown in Table 2.2. Extensive design and RTL code review are used to ensure BlackParrot contains efficient implementations of modern microarchitectural features expected of a Linux-class microprocessor. SRAM and logic structures are sized to be performance, power, and area (PPA) efficient. To validate the design we have fabricated BlackParrot in GlobalFoundries 12nm process node, and are deploying new features across frequent 12nm and 40nm tapeouts.

Design Manifesto

During the course of the project, there have been many cases where there are two reasonable technical directions to take the project. To guide our effort, we developed an informal *manifesto* to help decide these difficult chases. The manifesto has three key rules:

1. *Be tiny.* When choosing among alternatives, we choose the option that results in a smaller, more understandable code base and in less die area, simpler critical paths, and fewer bugs. The result is a code base that is as small and understandable as possible, and hardware that is PPA efficient. We take care to *not* implement esoteric and non-performance critical components in RTL, and to avoid a common problem in recent generator-based RTL methodologies: a multitude of tunable knobs in which most combinations have been untested and yield dubious PPA benefit. If a feature is required by the RISC-V spec but is not performance critical, we implement it through emulation. The code is "all the RTL you need and nothing that you don't".

2. *Be modular.* We employ clean, latency-insensitive interfaces that do not rely on knowledge of the other module's internals. This allows multiple contributors to work independently of each other, and to minimize bugs that emerge from incomplete understanding of the entire code base; and

3. *Be friendly.* We ask ourselves both in design decisions and in our presentation: does this make the project more approachable? With this we can build a large open source project culture that encourages contributions and avoids the "not invented here" syndrome.

To facilitate widespread adoption, the BlackParrot infrastructure is built with community in mind. Environment dependencies are kept to a minimum. BlackParrot is developed on a modern Linux version with easily portable Makefile-based simulation and synthesis flows. Support for open-source tools such as Verilator and GTKWave is prioritized as a first-class concern, and BlackParrot will be one of the first major test cases for the upcoming OpenROAD open-source CAD flow. All of these considerations culminate in a focus on "out-of-box experience". Prospective users are able to go from

GitHub to simulation with a handful of simple commands.

System Architecture

The BlackParrot multicore implements the RISC-V 64-bit "RV64G" architecture, which includes the base integer ISA "I", multiplication and division "M", atomics "A", single and double precision floating-point "F/D". It supports three privilege levels—machine, supervisor, and user—as well as SV39 virtual memory; these extensions are sufficient to efficiently run full-featured operating systems such as Linux.

Race-Free Programmable Cache Coherence

BlackParrot implements a distributed directory-based cache coherence protocol, which currently supports VI, MSI and MESI. The underlying implementation, *BedRock*, consists of a collection of *Local Cache Engines (LCE)*, each controlling an L1 cache, that connect over an interconnection network to the programmable *Cache Coherence Engines (CCEs)*, which collectively maintain the address-sharded directory state.

The BedRock protocol implementations have the unique property of being *race-free*, because they ensure that coherence state transitions occur at the CCEs and not at the local caches or LCEs, which significantly reduces protocol complexity.

BlackParrot contains many novel and useful uncore/multi-core features, including a programmable cache coherence engine and coherent accelerator interfaces. This programmable coherence engine implements a novel distributed directory-based cache coherence protocol which we call BedRock. BedRock is fast and simple enough that verification tools can prove its correctness. BedRock enables effortless composition of systems with coherent

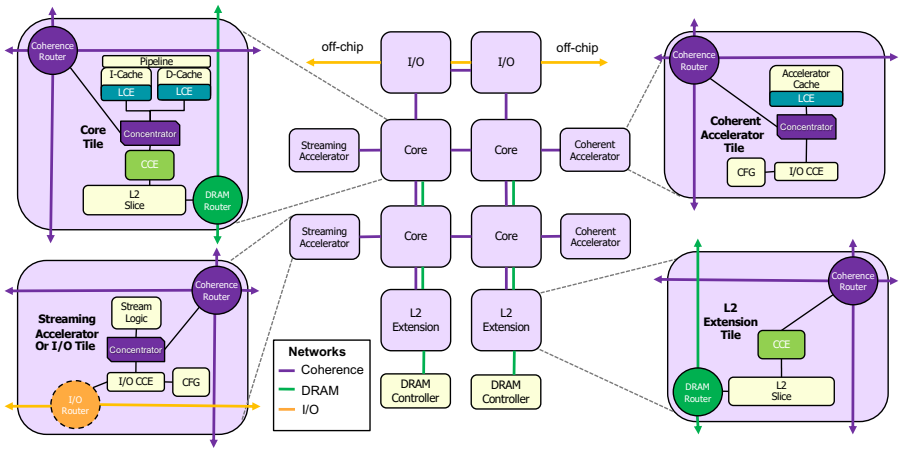


Figure 2.3: BlackParrot multicore SoCs are comprised of a mesh of heterogeneous tiles, allowing flexible composition of cores, accelerators, L2 cache slices, I/O and DRAM controllers. Four different kinds of tiles are pictured. *Core tiles* implement a processor, a directory shard and an L2 slice. *Coherent Accelerator tiles* implement an accelerator that has access to the cache coherent memory system. *L2 extension tiles* allow the amount of L2 cache to be changed. *Streaming accelerator or I/O tiles* allow flexible interfacing of a common memory system via a shared non L1-cached address space that is routed over the coherence network.

cores, accelerators and I/O devices.

2.1.3 Heterogeneous Multicore Tiles

BlackParrot is designed as a scalable, heterogeneously tiled multicore microarchitecture, as shown in Figure 3a. (We use the term multicore microarchitecture because the user or programmer is not aware of how the multicore's components are arranged, it is hidden beneath the multicore ISA layer.) Decomposing the system into replicated sub-blocks has several benefits: structures are regularized for scalability and ease of timing closure, systems can be flexibly composed into different topologies and protocol complexity can be shifted from the component level to the network level. Rather than connecting

these tiles with a shared bus, BlackParrot uses a collection of NoCs. The routers used in BlackParrot are dimension-ordered and wormhole-switched, using credit-based flow-control to limit contention.

In order to promote regularity for hierarchical ASIC flow designs, the system is designed as a 2D mesh with a single router per network contained in each tile. Some networks may have multiple endpoints within a tile – these endpoint connections are combined and connected to the router through a wormhole concentrator. While many other processors use standard bus-based interfaces such as AXI or AHB for all on-chip communication, these protocols are highly complex and require sophisticated IP blocks to achieve reasonable performance. Rather than couple its internal networks with any particular implementation of a standard bus, BlackParrot provides a set of adapters to transduce between a set of well-known protocols, such as AXI or WishBone.

An additional level of hierarchy useful for system design is grouping sets of similar tiles into complexes, which greatly simplifies network address mapping and maximally reduces the dimensionality of routers. Figure 2.3 shows the initial BlackParrot Processor Families available. The Core Complex (CC) contains BlackParrot tiles. The I/O Complex (IOC) primarily transduces between uncached BedRock messages and I/O messages which allow off-chip communication. The Streaming Accelerator and Coherent Accelerator Complexes (SAC and CAC) contain accelerators connected to the BedRock Network. Last, the Memory Complex (MC) contains addition on-chip memory, as well as a connection to off-chip memory, such as a DRAM controller.

BlackParrot Microarchitectural Tile Types

Tiles in a BlackParrot system attach to the BedRock network as both an LCE and a CCE. However, a given LCE or CCE may only support a subset of

operations provided by the interface. For instance, some LCEs may not have a coherent cache attached to them; some CCEs may not contain directory tags. Given these properties, BlackParrot microarchitectural tiles fall into one of the four categories enumerated in Figure 3a, which we detail below.

BlackParrot Core Tile A BlackParrot Core Tile contains a full BlackParrot processor or an accelerator which acts a processor, comprising one or more coherent caches as well as a directory shard and an L2 slice. A typical system has many Core Tiles.

L2 Extension Tile An L2 Extension Tile provides a simple scale-out method to increase the amount of on-chip L2 in a BlackParrot system. Each L2 extension contains a directory and a non-inclusive non-exclusive L2 slice. By distributing the L2 slices, a system designer can easily change the compute to cache ratio of a BlackParrot system without perturbing critical paths within the cores or the NoCs.

Coherent Accelerator Tile Attaching a Coherent Accelerator Tile to the BlackParrot network can be done with a few degrees of specialization. From an abstract system view, a coherent accelerator is simply an LCE with a backing coherent cache. Depending on the accelerator's needs and the project's complexity budget, users may (in increasing order of complexity):

1. Attach the accelerator directly to the provided BlackParrot data cache.
2. Reuse the provided BlackParrot LCE and provide a specialized cache.
3. Provide a specialized LCE implementation that interfaces with directly with the network.

Streaming Accelerator Tile Streaming Tiles are tiles which have no locally cached memory and do not logically control any physical memory. That is, these tiles do not contain a backing coherent cache for its LCE, nor a directory. These tiles may be used for basic I/O devices, network interface links, or even heavyweight streaming-based accelerators such as GPUs.

Other tile components As a truly tiled micro-architecture, BlackParrot distributes as many system resources as possible. Each BlackParrot core tile contains a memory-mapped configuration block, a slice of the Core Level Interrupt Controller (CLINT) and a global L2 slice. Splitting these system resources promotes regularity in the tiles, removes globally routed configuration and interrupt signals, ultimately easing physical design implementation. With simple adjustments to the network memory map, components can easily be attached to the BlackParrot system and addressed from any other tile.

Networks-on-Chip

There are 3 NoC classes in BlackParrot: Coherence (BedRock), DRAM and I/O. Although the NoCs are implemented using standard BaseJump STL modules, BlackParrot specializes each network for the protocol, including flit width, packet length and coordinate width, to optimize physical design.

BedRock Network The BedRock network is a cache-coherent fabric connecting all tiles in a BlackParrot system. Specifically, the network is the connection point for all LCEs and CCEs in the system. The BedRock protocol has 3 logical channels: request, command and response. The request channel is used by an LCE to initiate a transaction, specifying whether it is a read or write, whether it is cached or uncached, and additional metadata used for return addressing. The command channel is used by a CCE to modify

the system's LCE state. Example commands include setting tags, filling data, and completing synchronization sequences. Additionally, LCEs may be commanded to transfer cache lines among each other – these transfers travel over the command network as well. Finally, the response channel is used for coherence acknowledgements, allowing for serialization of requests.

Although the BedRock protocol does not require it, the current implementation of the BedRock network is a wormhole-routed 2D mesh, with 1 physical channel per logical channel.

DRAM Network The DRAM network connects CCEs to devices which are able to service memory requests, for example DRAM, Flash or on-chip ROMS. Since all requests are initiated by a tile and serviced by memory devices at the bottom of the chip, the memory network is a lightweight 0.5D network. The DRAM network is particularly suited to wormhole routing, as DRAM controllers tend to return least significant word first.

IO Network The I/O network exists to connect a BlackParrot processor to peripherals such as serial ports, PCIe controllers, external I/O devices and debug interfaces. Messages may be initiated on or off chip, so the I/O network is implemented as a 1D wormhole network. This network only exists in the I/O Complex; generally, it serves as a lightweight transducer and physical transport layer between BlackParrot protocols and standard protocols such as AXI, WishBone and simple bit banging.

Decoupled Core Microarchitecture

BlackParrot is designed to be modular, reaping the usual benefits of simpler verification, more agile development and easier onboarding of users and developers. Additionally, by focusing on interfaces rather than concrete

implementations, BlackParrot is provisioned to support a wide variety of possible microarchitectures. Figure 3b shows the current BlackParrot core microarchitecture.

Efficiency Through Thin Interfaces

While software interface abstractions are a useful tool, hardware interfaces have physical overheads which can cripple a design. Interfaces in BlackParrot are designed to have minimal overhead, partitioning regions which are both logically and physically separated. Each interface described here is implemented as a parameterizable SystemVerilog struct passed through a latency insensitive port, usually via a small FIFO. Decoupling the Ends ensures there are no timing paths between these logically separated components and provides confidence that implementation changes in one End will not break another.

Front End The Front End presents an in-order but potentially speculative instruction stream to the Back End. The issue queue decouples the Front End fetch from the Back End execution, allowing speculative fetching during long latency Back End operations such as services cache misses. During instruction fetch, exceptions may arise. Since exceptions in this domain are purely speculative, they are sent to the Back End to be serviced in-line with instructions. Because the RISC-V virtual memory scheme may modify architectural state during instruction fetch (setting the "Access (A) bit"), all TLB misses in the Front End are sent to the Back End to be handled inline with other exceptions. Along with the PC/instruction/exception pair, the Front End also sends metadata associated with the branch prediction that resulted in that particular PC fetch.

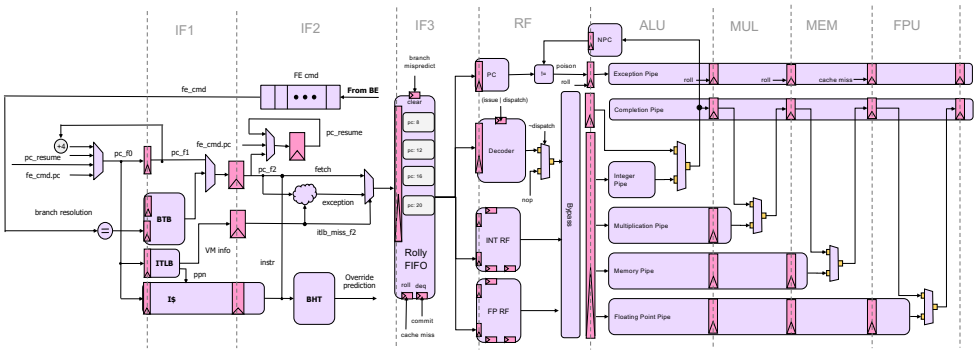


Figure 2.4: Fig 3b. **BlackParrot Core Microarchitecture.** Both the Front End (IF1 and IF2 stages in the diagram) and Back End adhere to the interface specifications, with simple and efficient pipeline implementations. Because of a modest misprediction penalty, complex branch predictors are unnecessary. In order to remove a physical design intensive global stall signal, the Back End is non-stalling after the issue stage, instead flushing the pipeline and replaying instructions upon infrequent cache misses. Cache misses are handled entirely through the BedRock coherence system.

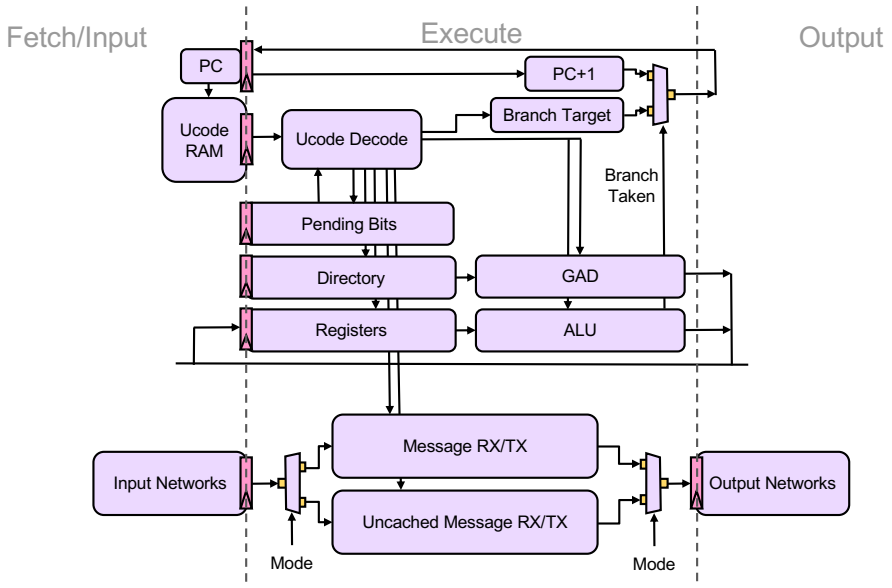


Figure 2.5: Microprogrammed Cache Coherence Engine. The microarchitecture of the CCE is fairly simple. By minimizing the coherence protocol complexity and pushing the management logic into software, the hardware is easily verifiable.

Back End The Back End executes instructions, handles exceptions, and generally maintains the architectural state of the processor. Messages from the Back End to the Front End are used to correct mispeculation and update shadow state in the Front End. Messages include branch resolution, interrupt redirection, iTLB manipulation and privilege mode changes. Upon branch resolution, the branch metadata associated with the branch is forwarded back to the Front End. This metadata is never inspected by the Back End; the particular branch prediction scheme used by the Front End is completely opaque to the Back End.

Memory End BlackParrot's memory end, *BedRock*, is a scalable, distributed, directory-based coherence scheme designed with an emphasis on simplicity and verification. Nodes in the coherence system are Local Cache Engine (LCE) or Cache Coherence Engines (CCE). CCEs are responsible for managing coherence for a slice of physical address space. LCEs are responsible for initiating and responding to coherence requests on behalf of a coherence cache. BedRock connects all components of a BlackParrot multicore, including non-coherent or I/O devices.

Three types of CCE are available in Bedrock: a novel microprogrammed variant, a traditional fixed-function management engine, and a minimal controller which implements only uncached requests and is used for I/O or simple accelerators. The microprogrammed CCE is the default for a BlackParrot core and provides substantial flexibility and adaptability when implementing variants of coherence protocols, and comes at a small area cost. Notably, since microcode can be changed out by a simple firmware update, advanced coherence experimentation and security patches can be applied even on existing silicon designs.

Some configurations of BlackParrot (e.g. single core, with software coherence, leveraging external coherence systems) do not require a full coherence system. In these cases, caches simply need a mechanism to bootstrap and service misses. By standardizing the cache miss handling interface and constraining it to be a subset of the Cache to LCE interface, different cache implementations and coherence schemes can be mixed and matched in BlackParrot.

Agile Development Process

In this section we describe the key features of the BlackParrot development effort that lead to be a design users can trust.

Microarchitectural Specification (MAS) Documents Components of BlackParrot are *designed* before they are implemented. Initially, we separated the design into three key components: the front end, the back end, and the memory system. Module interfaces for these components were developed and further sub modules designed. Each component was designed, documented and peer reviewed prior to any RTL being implemented.

Along the way new features are requested. All new features first undergo a three day waiting period – the person requesting it had better really want it. It is then closely scrutinized and alternatives considered. If the feature is deemed worth the "technical debt" of added complexity, it undergoes the same design, documentation and review process prior to implementation.

Leveraging Open-Source Libraries In order to rapidly iterate on BlackParrot, it is important to leverage established open-source codebases. By building upon battle-tested hardware libraries such as BaseJump Standard Template Library (STL) for SystemVerilog [235] and Berkeley Verilog HardFloat [109], and integrating using latency insensitive design principles, BlackParrot is able to minimize its universe of possible bugs. Development of BlackParrot is done using commercial tools, such as Synopsys VCS, but supporting at least one open-source alternative, such Verilator [210], is a first-class infrastructure concern.

Evaluating Design Complexity and Out-of-Box Experience One of the most difficult-to-evaluate components of the BlackParrot manifesto is whether we have achieved our goals of virality and complexity. To evaluate this, we assigned BlackParrot as a 3-week class project in an VLSI class attended by 35 fourth and fifth year students, and all of the students (including ones that had not taken computer architecture before!) were successful in proposing

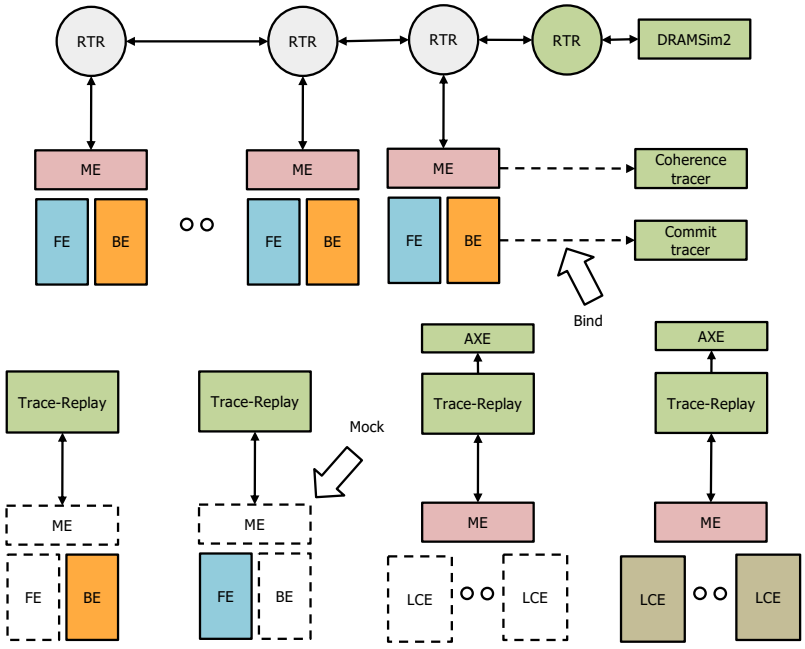


Figure 2.6: Modular Testing in BlackParrot. Testing in BlackParrot is driven by the insight that testing is invaluable, but testbenches are expensive to maintain. Testbenches are designed to verify multiple dimensions of a system and are composed of flexible, reusable components.

and implementing unique, previously unsupported features into the core, such as modifying the pipeline to change the load-use latency to 2 cycles instead of 3, adding variable fill widths to the L1 cache, enabling different bank sizes in the L1 cache, and implementing more complex branch predictors in the frontend. We intended to have yearly BlackParrot-related projects to continuously self-evaluate these aspects of the project.

Continuous Integration, Code Review Cleanly specified interfaces between the front-end, back-end and memory system components allow

for stubbed-out versions to quickly be constructed. Test harnesses at the component level were also developed.

The first instruction to flow through the BlackParrot core occurred four weeks after completion of the MAS documents and team formation. After this milestone BlackParrot remained fully functional throughout the development process. New features, such as virtual memory support, were added by forming new teams and adding them into the existing working BlackParrot core.

For some modules we implemented multiple alternatives. For example, the programmable cache coherence controller was also implemented in a fixed-logic finite state machine form. We did this to fully understand the performance/area trade-off of the design.

Along the way, refinements to the initial module partitions did occur. For example, to remove a potential security flaw in the front-end of the processor, all page table walks are handled non-speculatively in the back-end. This required refinements to the inter-module messages that occur within the processor. However, by having a clean interface between modules, no significant changes to the MAS documents and overall module structure were necessary to add this support, despite the fact that the early stages of the processor were not designed for it.

Co-simulation Testing Framework The majority of testing in BlackParrot is done through a system-level testbench driven by program level execution. In addition to a handful of directed white-box tests, BlackParrot supports the riscv-tests suite, BEEBS suite, Spec and CoreMark as a baseline functional regression. Our plan is to greatly expand this selection.

RISCV-DV [94] is a UVM-based instruction generator framework recently re-

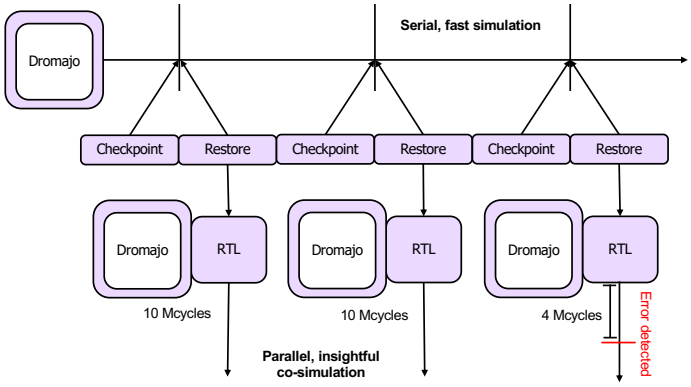


Figure 2.7: Hybrid Co-simulation Using Dromajo. Parallel co-simulation dramatically speeds up traditional RTL simulation methodologies without sacrificing flexibility.

leased with support for RV64GC, MSU privilege modes, page-table generation and trap-handling support. Due to a dearth of open-source directed privileged mode tests (i.e. other than booting Linux), constrained random testing is an effective way to expose edge cases. In a matter of days, RISCV-DV was able to expose several hardware bugs in BlackParrot which may otherwise have been found millions of cycles into a Linux boot test.

In the spirit of modularity, testing is also done at the End level (Figure 2.6). Mock versions of other components are substituted in order to isolate bugs. By maintaining a suite of End level tests, external BlackParrot developers can easily verify their changes comply with the system interfaces.

Under traditional architectural research methodologies, there exist clear trade-offs between speed and accuracy. In order to temper excessive debug cycle times, hardware designers often analogously sacrifice system visibility for speed through techniques such as FPGA emulation [136] or high level modelling of sub-components. Debugging at the RTL level is commonly done by comparing execution traces of a simulation model with a "golden

trace", typically generated by a well-trusted functional model, and examining waveforms in the case of a trace mismatch. Another approach is to implement the RTL model in an FPGA, outputting periodic status and debug information. (A third option largely out of reach to academics and hobbyists is to use sophisticated FPGA emulation frameworks such as Synopsys ZeBu.)

Instead, BlackParrot employs a hybrid approach of parallel co-simulation using an open-source RISC-V ISA simulator, Dromajo [68]. First, a long-running program is simulated using Dromajo. Every N cycles, Dromajo collects the architectural state of the system and creates both a memory dump as well as a checkpoint ROM. The checkpoint ROM comprises ordinary RISC-V instructions designed to initialize a freshly rebooted processor to a well-defined architectural state. Next, *in parallel*, the BlackParrot RTL model is restored using each checkpoint ROM and co-simulated alongside the Dromajo model. Imprecise events such as interrupts and device I/O are relayed from RTL to Dromajo to keep the two versions aligned.

ASIC Validation In addition to our 12nm BlackParrot chip, which is running in our lab, several upcoming BlackParrot tapeouts are planned both as standalone chips and as accelerator hosts. Directories containing tape-out parameterizations, constraints and ASIC infrastructure are all provided as references. Work is in progress to push BlackParrot through the UW OpenROAD Free45 Reference Flow [170] [5], so that users will simply be able to clone the repository and generate a fully placed-and-routed, representative BlackParrot. Providing this capability will enable architects to quickly validate their hardware experiments without the financial and intellectual overheads of maintaining a commercial CAD flow, or needing to sign restrictive foundry NDAs.

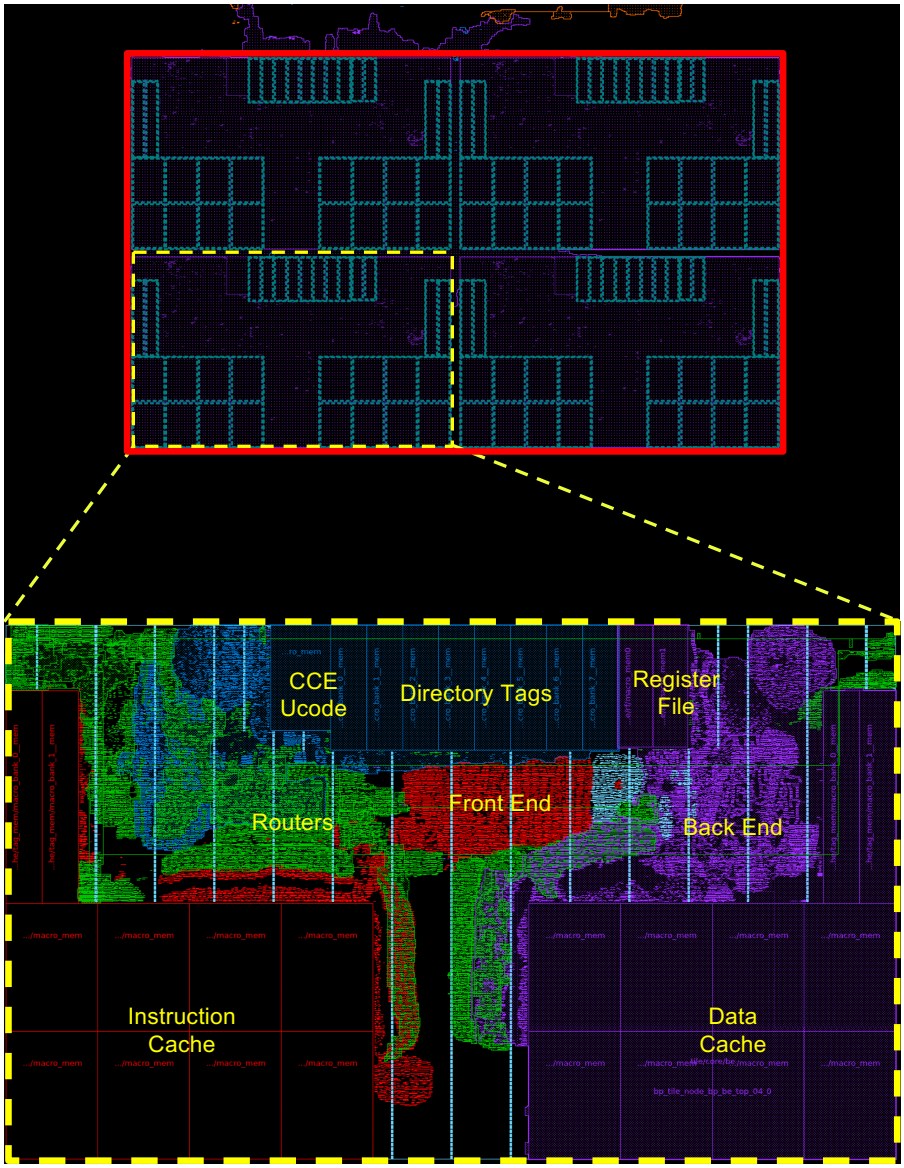


Figure 2.8: BlackParrot GF12 Placed and Routed. A quad-core BlackParrot system was taped out in July 2019 using GlobalFoundries 12nm. Lessons learned from this tapeout have driven several major changes in the BlackParrot system architecture, resulting in a 50% reduction in tile area as well as a 50% reduction in total wire length. Although RTL-level simulation is effective at analysing and comparing relative performance, an experiment without ASIC validation can mask serious physical limitations of a design.

Conclusion

RISC-V challenges the world order of x86 and ARM. UC Berkeley has bootstrapped a global stewardship to maintain the RISC-V ISA and its software base. However, yet to emerge is a similar global stewardship of a Linux capable RISC-V implementation that is documented, PPA efficient and implemented in standard SystemVerilog. BlackParrot is architected from the ground up to fill this role, in contrast to prior efforts [19][57][273] which have centralized stewardship models and evolved organically to their current state.

BlackParrot is tiny, modular and friendly. It is an ideal SoC "base class" to integrate with accelerators and build Linux-capable systems with. We welcome your enhancements! As BlackParrot becomes more widely used, community experts can contribute back to BlackParrot, ensuring that it remains representative of state of the art processor designs. BlackParrot is now available on GitHub (`https://github.com/black-parrot/`) under a BSD-3 license.

Chapter 3

ZYNQPARROT

3.1 ZynqParrot

This section has been adapted from an Arxiv 2025 submission, "ZynqParrot: A Scale-Down Approach to Cycle-Accurate, FPGA-Accelerated Co-Emulation".

3.1.1 Abstract

As processors increase in complexity, costs grow even more rapidly, both for functional verification and performance validation. Additionally, performance models become ever more sensitive to slight microarchitecture inaccuracies. Runtime measurements of key workloads are an essential part of the performance debugging process. Most often, silicon characterizations comprise simple performance counters, which are aggregated and separated to tell a story. Based on these inferences, performance engineers employ microarchitectural simulation to inspect deeply into the core. Unfortunately, dramatically longer runtimes make simulation infeasible for long workloads.

Traditionally, architects have bridged this gap by performing early prototyping on FPGA. Yet, the scale of modern designs is impractical to implement on a single emulation board. Large companies use Scale-Up solutions such as commercial emulation platforms, but these are unaffordable to academics, hobbyists and startups. Others have proposed Scale-Out solutions, leveraging cloud FPGA clusters to emulate large System-On-Chips. However, this

approach prescribes certain I/O and memory system architectures instead of the native interface timings of any given subsystem.

Instead, we propose a *Scale-Down* approach to modelling and validation. Rather than up-sizing a prototyping platform to fit large and complex system designs, we show that it can be more accurate, faster, and more economical to decompose a system into manageable sub-components that can be prototyped independently. By carefully designing the prototyping interface, it is possible to adhere to strict non-interference of the Device Under Test (DUT). This allows architects to have the best of both worlds: the speed of FPGA acceleration while eliminating the inaccuracies of Scale-Out and the inherent costs of Scale-Up.

In this work, we present ZynqParrot: a Scale-Down FPGA-based modelling platform, capable of executing non-interfering, cycle-accurate co-emulations of arbitrary RTL designs. ZynqParrot is capable of verifying functionality and performance with arbitrary granularity. We also provide case studies using ZynqParrot to analyse the full-stack performance of an open-source RISC-V processor.

3.1.2 Introduction

As processors increase in complexity, verification costs grow even more rapidly, both for functionality and performance. The end of Dennard Scaling [46] has led to a Cambrian explosion of domain-specific accelerators which present unique, full-stack verification challenges. Besides functional correctness, performance validation is critical to achieving worthwhile gains from accelerator integration. The higher performance the design, the more sensitive it is to subtle disturbances in the microarchitecture. When characterizing full-system performance in silicon, software engineers use simple performance counters

which must be decided upon early in the design process, before problematic subsystems have been identified. These counters are aggregated and separated to divine reasons for the performance of a target application. On the other end of the spectrum, architects can leverage the deep verification capabilities of pre-silicon waveform inspection to identify subsystem bottlenecks before tapeout, when fixes are much cheaper. However, cycle-accurate simulations are painfully slow, so architects must settle for sampling applications [107] to complete in a reasonable timescale.

Traditionally, architects have bridged this gap by performing early prototyping in FPGA. By doing so, RTL similar to tapeout designs can be emulated with cycle accuracy at 10-100x faster than simulation alone. However, modern designs are too large to economically fit on a single emulation board. Large companies can Scale-Up their prototyping systems using commercial emulation platforms [215, 53, 160], but these are unaffordable to academics, hobbyists and startups. Other academics have proposed Scale-Out solutions that leverage cloud FPGA clusters [10, 6] to emulate large System-On-Chips. However, this approach generally relies on regularity in the design, prescribes certain standardized I/O and memory system architectures and couples platforms to proprietary vendor IP which may interfere with the native interface timings of any given system and provide no insight or ability to adapt for system needs.

Inspired by biotechnological process modelling [164], we propose a *Scale-Down* approach for architectural prototyping, shown in Figure 3.1. Instead of unilaterally scaling up a chip design (process) from an FPGA prototype (lab-scale experiment) to a full tapeout (industrial manufacturing process), it is more economical to iteratively Scale-Up and Scale-Down the design to identify subsystem issues at scale and precisely debug performance in a smaller, more

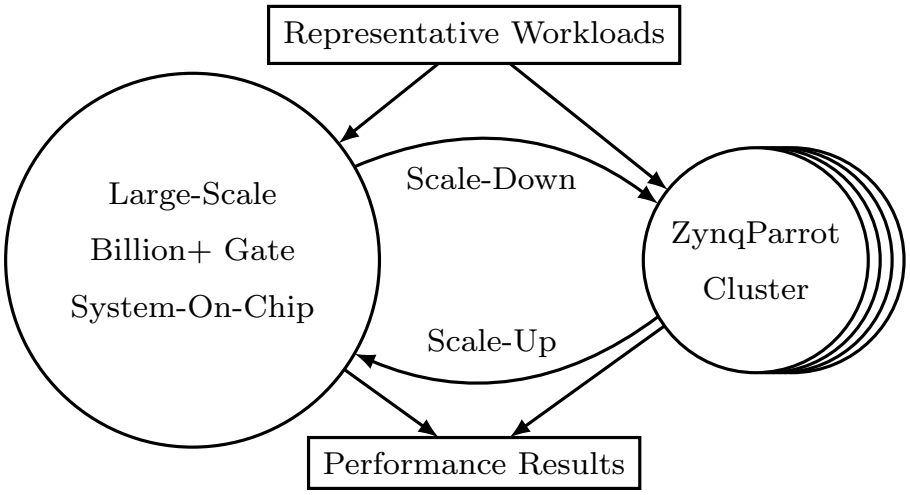


Figure 3.1: A Scale-Up/Scale-Down cycle transforms industrial manufacturing processes into laboratory experiments. Experiments accept identical inputs as the full process. Results are extrapolated to predict impacts on the modified process before committing to costly change orders. Iterating over the process lifecycle leads to continuous improvement.

tightly instrumented system. By closely correlating the two environments, Scale-Down results can give deep debugging insights into process deviations as well as accurately predict future deviations for orders of magnitude lower cost than a full production run.

In this work, we show that decomposing and recomposing the system is more accurate, faster, and more economical than state of-the-art alternatives. By carefully designing the platform interfaces, it is possible to provide flexible environments that represent the interactions of real systems while maintaining system timings at the component interfaces. This allows architects to have the best of both words: the speed of FPGA acceleration while eliminating the inaccuracies of Scale-Out and the inherent costs of Scale-Up. Because iteration time is much faster than monolithic prototypes, small design teams can quickly bootstrap new Scale-Down subsystems, run large simulations on

abstracted Scale-Up models and return to Scale-Down to rapidly debug and enhance components.

When Scaling-Down, great care must be taken to accurately transform the design and exactly mimic the environment between the full design and the subsystem. In particular, I/O interface timings must adhere to accurate timing models. Previous works [136, 69] have focused on instrumenting FPGA timing for common interfaces such as DRAM and Ethernet rather than the fully custom models required for subsystem partitioning. Others [14] have focused on providing full-system emulation for software development and too high a level of abstraction for microarchitectural debugging. Instead, this work aims to provide more precise and more flexible interface emulation to allow for finer-grained partitioning without sacrificing accuracy.

Previous FPGA emulation platforms [215] [53] [136] [69] are expensive, dependent on vendor IP, or cumbersome and prone to lock-up. In contrast, ZynqParrot builds upon the BaseJump STL [235] library to provide generic and completely open bridges to commonly available AXI and UART interfaces. When using Zynq-based FPGAs [268], the only requirement to use ZynqParrot is an SSH-capable machine running Vivado. For non-Zynq FPGAs, ZynqParrot requires a UART connection to the FPGA as well as a JTAG connection for bitstream programming, although the platform architecture easily supports plugins for additional host functionality. Contrast this to traditional solutions which require expensive PCIe-capable accelerators. These setups are hard to maintain, built on top of proprietary PCIe IP and software layers such as Xilinx XDMA [267]. Failure to interface correctly to PCIe can lockup not only the DUT but also the host server machine, requiring remote restart capabilities. A cluster of such host machines and FPGAs can easily exceed 10s of thousands of dollars and require full-time system administration (see

Table 3.1). In contrast, ZynqParrot provides verification teams the ability to begin with the minimal possible Total Cost of Ownership (TCO) and scale costs alongside the design progress.

Strategy	**$/year/FPGA[0]**	**Logic Unit**	**Required I/O**
Scale-Up[1]	∼$5000	Full Design	Native PCIe
Scale-Out[2]	∼$2000	Tile	PCIe Tunnel
Scale-Down[2]	**∼$100**	**IP Block**	**SSH/Serial**

Table 3.1: On a per-FPGA basis Scale-Down systems require a much smaller investment, allowing teams to incrementally build up their verification infrastructure. Parallelizing a benchmark suite rather than a subcomponents of a design has further economic benefits.

[0] 2000 hours is equivalent to a year of 8-hour regressions.
[1] $1.4167 per AWS f1.16xlarge hour [11].
[2] $0.6744 per AWS f1.4xlarge hour [11].
[3] $300 per Avnet Ultra96v2 board [21], with a replacement rate of once per three years. Cluster MTBF is 100+ years [266].

ZynqParrot adheres to strict non-interference of internal design timings through strategic clock-gating during unpredictable host back-pressure. This allows subsystems to execute with the illusion that they are running in situ within the full system. Each Device-Under-Test (DUT) execution cycle can be verified against a emulated model, taking into account functional correctness along with verification of internal and external performance. Additionally, ZynqParrot is able to synthesize complex performance counters without interfering with mature or frozen RTL. Such a deep dissection of the subsystem can allow an architect to design experiments to identify subsystem bottlenecks and quickly iterate without requiring microarchitectural changes.

We provide case studies of how to analyse a complex RISC-V processor, *BlackParrot*, using ZynqParrot. We identify useful software and hardware enhancements to quickly verify functionality and performance and how they easily fit into the ZynqParrot framework. These non-invasive, instrumented

measurements and host software abstraction layers were written for use in BlackParrot; however, they are generally applicable to a wide range of RISC-V projects. Additionally, we demonstrate real-world uses of ZynqParrot: first to host the term projects of an undergraduate/graduate architecture class of 20 students, second to bring up first-batch silicon during a commercial tapeout, and third to analyse a bottleneck and access the improvement of a new microarchitectural widget in BlackParrot.

Our major contributions are:

1. We present ZynqParrot[1], the first Scale-Down open-source prototyping platform for deterministic, cycle-accurate local FPGA co-emulation.
2. We provide a co-simulation capable prototyping library that eschews vendor dependencies and is cycle-identical to ZynqParrot co-emulation.
3. We explore the diversity of ZynqParrot usage via a series of case studies, from the classroom to academic research to commercial bring-up.
4. We introduce PanicRoom[2]: an ultra-portable library that leverages DRAM to run POSIX benchmarks on bare-metal systems.

Complex IP designs often require specific Vivado versions to ensure reproducible builds. However, each Vivado version only supports a small subset of operating systems, often directly conflicting with strict ASIC EDA vendor compatibility guides. Although Vivado-based build systems can be broken up into bitstream generation and programming machines, this requires supplemental system administration.

---

[1]The hardware and software code for ZynqParrot is open-source under a permissive BSD-3 License.

[2]The software code for PanicRoom is open-source and as of submission time being actively upstreamed to the Newlib project

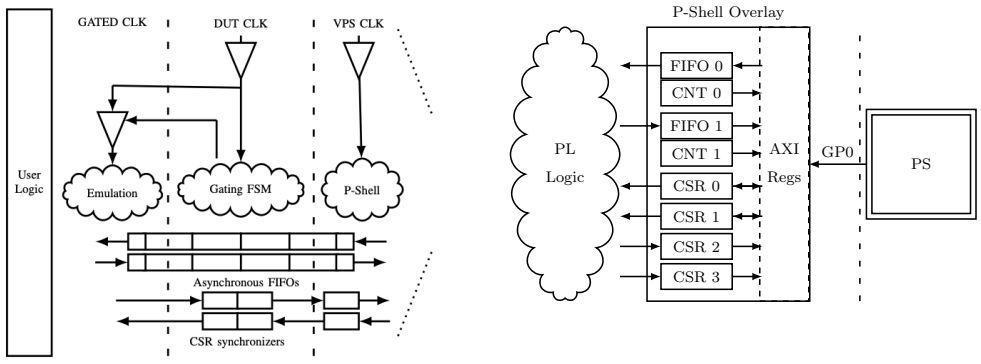
Additionally, many FPGAs are only available in PCIe-attached variants, resulting in complex and fragile bridge solutions. These connections require proprietary vendor solutions such as Xilinx XDMA [267], non-portable RTL and software layers and are prone to full-system lockup due to subtle bugs in configuration or execution. Lockup can cause long-running experiments to be lost, force costly manual debugging and complicate systems with fallback-recovery mechanisms.

Considering these limitations, emulation management software is often tailored to each specific host-FPGA pair and disparate from simulation-based testbench infrastructure, which must be maintained independently. As a result, to normalize environments across designs for powerful prototyping boards, groups must maintain costly heterogeneous server infrastructures with multiple x86 environments customized to each design.

Traditional FPGA prototyping systems require powerful discrete host servers to:

1. Compile a netlist and generate a bitstream, often requiring hours for large designs.
2. Program the bitstream, typically over a USB/JTAG connection.
3. Manage emulation execution, running software to load test vectors, monitor performance or verify results.

In this section, we describe the ZynqParrot architecture and how it addresses these challenges cost-effectively and with lower maintenance than previous solutions.



(a) ZynqParrot subcomponents interface with the P-Shell through a parameterizable set of SB-FIFOs and CSRs. The P-Shell logic is run asynchronously to the DUT, allowing for decoupled co-emulation. Clock gating logic on the VPS side ensures accurate co-emulation by maintaining internal timings of the DUT.

(b) As DUT logic may be buggy during design, it is essential to not hang GP0, which could lead to VPS lockup. In ZynqParrot, the P-Shell prevents lockup regardless of DUT state, by lifting generic DUT interfaces to a set of nonblocking FIFO and read/write CSRs.

Figure 3.2: The ZynqParrot system provides system architects with full co-emulation capabilities through a simple C++ MMIO drivers, identically accessible from simulation, co-emulation or on deployed systems. Users parameterize the P-Shell to for control or monitor execution, while the VPS runs any necessary software functional models.

Hardware Architecture

Zynq FPGA boards couple hardened ARM cores (*PS*) with a programmable fabric (*PL*). Common peripherals such as USB, Ethernet, and DRAM are connected to the PS, while the PS communicates with the PL via hardened AXI [15] interfaces. The PS master ports are called *GP* ports and cover a small address space. PS client ports (*HP*) are larger and higher performance, allowing the PL to indirectly access DRAM and peripherals.

ZynqParrot leverages the Zynq architecture to decompose prototyping systems into these orthogonal functionalities. Bitstream generation can be done on any machine with a compatible Vivado version to the particular IP. From there, users can login to the PS over a standard Ethernet or UART connection, copy over the compiled bitstream and using the Pynq API, program the overlay and DUT on the PL.

ZynqParrot provides an overlay (shown in Figure 3.2) that includes the *P-Shell*, the main interface between the host emulation and the DUT user logic. The P-Shell provides a parameterizable array of input/output Control and Status Registers *CSRs*, as well as an array of semi-blocking *SB-FIFOs*. An SB-FIFO exposes blocking ready/valid [235] interfaces to the PL side to support latency-insensitive interfaces, while the PS interacts with a non-blocking credit/valid interface to prevent system lockup. While non-blocking interfaces require multiple transactions for each read and write, they generally have little overall performance impact as the PS outpaces the PL during large system prototyping.

While ZynqParrot fits seamlessly into the Zynq PS-PL paradigm, there are many FPGA architectures which lack hardened CPU cores. For these boards, ZynqParrot provides hardware bridges (shown in Figure 3.3) which can convert

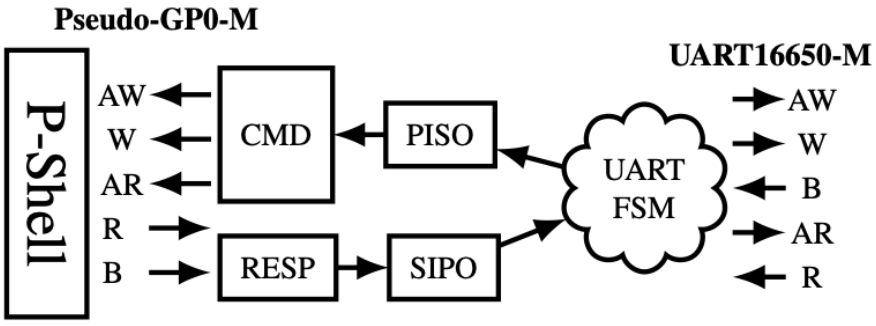


Figure 3.3: When prototyping on non-Zynq FPGAs, ZynqParrot transparently tunnels requests into a "pseudo-GP0" accessing the P-Shell. For instance, GP0 writes are deserialized from UART RX while GP0 reads are deserialized from RX and then reserialized into UART TX.

C++ P-Shell requests to a pseudo-GP master bus via a transparent software translation. We refer to this combination of C++ co-simulation code, host transport layer and GP master bus as a *PS*. The PS abstraction supports the flexibility of arbitrary MMIO interaction with the DUT while switching out transport layers optimized for the specific execution environment, all while maintaining lockup safety and reasonable performance.

When prototyping ASICs, the DUT clock is often limited by poor mapping of standard cells to FPGA primitives [259], limiting the overall emulation performance. While some efficiency may be regained by explicit manual remapping of problematic primitives (CAMs, large muxes, heavily retimed modules), this duplicates design efforts and forces dependencies between FPGA and ASIC teams. Unfortunately, even the best mapping efforts cannot solve this problem for aggressive submicron designs. To alleviate performance bottlenecks and decouple the prototype and emulation, asynchronous FIFOs and CSR synchronizers bridge the PS and DUT clock domains. This decou-

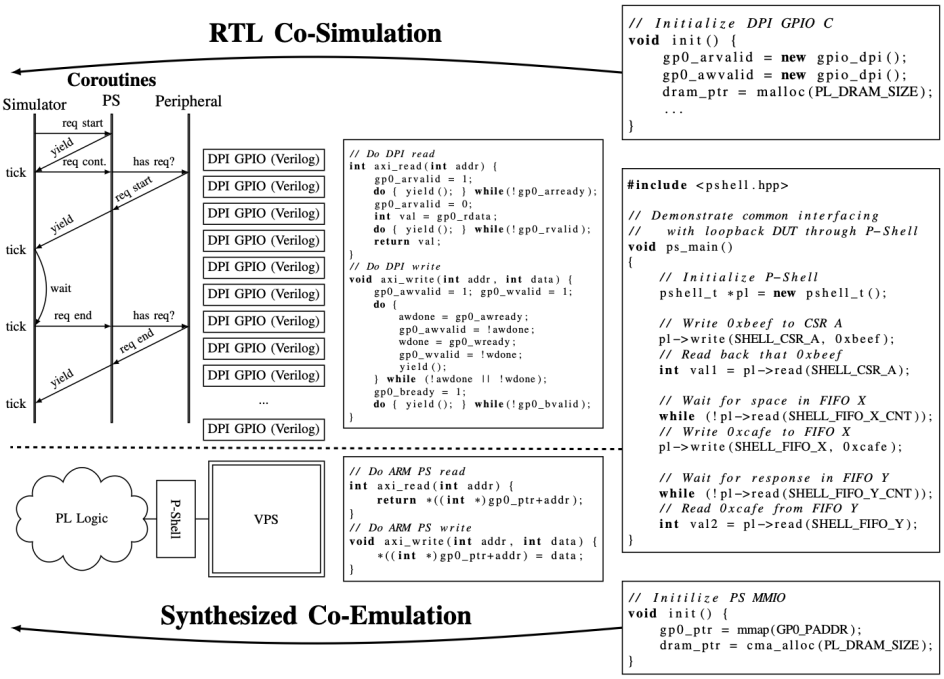


Figure 3.4: ZynqParrot enables designs to run identical C++ code on the VPS of a Zynq ARM core, over a UART bridge or in vendor-agnostic simulation. Instead of relying on Verilog tasks to interact with the DUT, ZynqParrot exposes pins on the P-Shell through a DPI-C interface. The result is fine-grained control over DUT execution, enabling software flow-control and thorough verification. As multithreading is disallowed by many commerical simulators, C++ coroutines are used to co-simulate the DUT with blocking transactions such as AXI requests, providing parallelism and deadlock avoidance.

pling allows the PS to run ahead of PL and averts complex emulation models slowing the DUT execution.

Emulation Layer

During deep performance profiling, the PS may need to process monitoring information or system-call emulation every DUT cycle while it is also handling context switching, network bridging or other asynchronous processing. If

the PS is not ready to accept a new packet and an asynchronous FIFO fills, either the FIFO must backpressure such that cycle-accuracy is lost, or the packet is dropped. Most systems using latency-insensitive I/O constructs use ready/valid handshakes to pause the DUT operation upon backpressure. However, doing so perturbs the system and eliminates cycle-accuracy, making the emulation non-reproducible. Figure 3.4 shows an abstraction of the PS responsibilities.

Another approach to avoid degradation is to run a Real-Time Operating System (RTOS) on the PS. However hard real-time guarantees are difficult to meet, restricting maximum performance; and proofs would need to be rewritten for each DUT interface, slowing iteration time. During profiling the information bandwidth needed varies dramatically based on the specific performance aspect being monitored. These guarantees will need to be retuned for each monitoring mode, as well as relaxed for running in a simulation mode that has dramatically different (wall-clock) timing characteristics. In addition, compiling arbitrary programs is much more difficult on a specialized RTOS compared to a full POSIX operation system.

ZynqParrot leverages the PS-DUT asynchrony to implement cycle-accurate emulation by gating the DUT clock upon interfering backpressure. Once gated, the asynchronous FIFOs are drained and execution can safely resume. This approach masks non-determinism in the PS, which may be running a full PetaLinux [270] operating system. Clock gating in the P-Shell means that both PS software and DUT logic can be completely unaware of the other side of the interface, operating in an ideal environment. Clearly defined boundaries between PS and DUT domains simplify necessary timing constraints during synthesis and standardized, validated asynchronous primitives shield users from the subtle gotchas of multi-clock systems.

On the other hand, modelling exact I/O timing is an essential functionality in a Scale-Down system. In ZynqParrot, hardware model timers exist in the DUT clock domain, but timing information is stored in the PS program where it is exchanged via a simple handshake. For instance to prototype a system with cutting-edge HBM DRAM, the DUT may emit a DRAM request which causes DUT execution to stop. The PS receives the request, calculates the predicted timing of the specific HBM model, and programs the expected timing through P-Shell CSRs. The DUT clock then resume, waiting for the DRAM request to return but executing any other parallel tasks. If the DRAM request returns before the hardware model timer finishes, it will be paused until the correct cycle. If the hardware model timer expires before the DRAM request returns, the DUT will return to a gated state. This event-driven co-emulation maintains cycle-accuracy while ensuring there are minimal wasted cycles.

Scale-Down Decomposition

A key element of Scale-Down is shrinking the design size while maintaining the integrity of inputs and outputs of the system. As illustrated in Figure 3.5, even for a single design hierarchy there are various decomposition strategies. The best strategy will depend on the size of the design, verification team and FPGA supply. At later stages in the verification cycle it may be advantageous to hyperfocus on smaller modules, attempting to expose subtle optimizations that may be clouded by full-system effects. Small, cost-effective FPGAs are especially advantageous at this stage. Firstly, smaller designs will benefit from shorter compilation times and faster emulation speeds, leading to quicker overall turnaround time. Secondly, when running a large number of tests (for example running a benchmark suite on a processor design), each independent run requires system resources such as a DRAM storage and bandwidth to

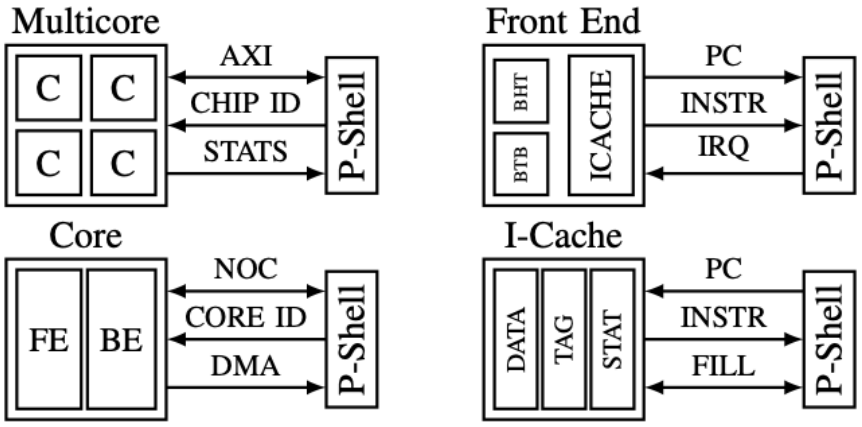


Figure 3.5: Decomposition into smaller components leverage smaller FPGAs to verify modules in parallel as well as hyperfocus on specific performance or verification targets. In the above decomposition, verification engineers can tune their analysis for either a full multicore, a single core, the front end or just the instruction cache. Because the P-Shell supports both unidirectional and latency-insensitive links, any hardware interface can be exposed. Rather than cumbersome hardened RTL models, VPS software mimics the interface timings.

support it. Allocating a cheap board for each test scales testing throughput linearly, while investing in larger FPGAs quickly becomes cost prohibitive.

PanicRoom: Portable Bare-Metal Benchmarking

Due to the complexity of benchmarking experimental processor designs, architects normalize performance across a wide range of applications such as [115, 116, 50, 44]. There are several significant challenges associated with reusing the same applications for fabricated chips as for pre-silicon designs: notably, the scale of commercial benchmarks are incompatible with the massive slowdowns of RTL simulation. Prior works have proposed reducing input set size [145, 242] and reducing instruction count through statistical

Table 3.2: PanicRoom supports host functionality more portably than comparable proxy solutions. While a full debugging solution such as ARM Semi-Hosting provides an end-to-end experience, PanicRoom is able to run identically in simulation and hardware, increasing verification correlation.

Proxy Solution	**Hardware Needed[0]**	**LoC (Userspace)[1]**	**Open-Source**
RISCV-PK	Host core	14157	✓
RAW Interface	Host core	6999	✓
ARM Semi-Hosting	Debugger	-	
PanicRoom	DRAM	**20**	✓

[0] While ZynqParrot provides a VPS host, PanicRoom can run fully untethered.

[1] We refer to non-benchmark, bare-metal code as "Userspace".

sampling [107, 254]. Other approaches is creating targeted benchmarks which are intrinsically small and portable [89, 106, 176], but these have questionable correlation to high-performance microarchitectures.

In addition to scale, commercial benchmarks also suffer from oversized scope. The most commonly evaluated suites rely on functions from the C standard library for I/O capability, filesystem operations and memory management. While it is interesting to evaluate the performance of an end-to-end system, during deep microarchitectural optimization architects often wish to observe bare-metal behavior. Yet, without operating system, it is impossible to run all but the most intentionally portable applications. Instead of glibc [86], embedded systems typically rely on smaller stdlib implementations, but these lack necessary system call compatibility.

To bridge this gap, we introduce *PanicRoom*: a minimal, mostly-platform-agnostic C standard library implementation that enables running POSIX applications on bare-metal systems. PanicRoom is built as a Board Support

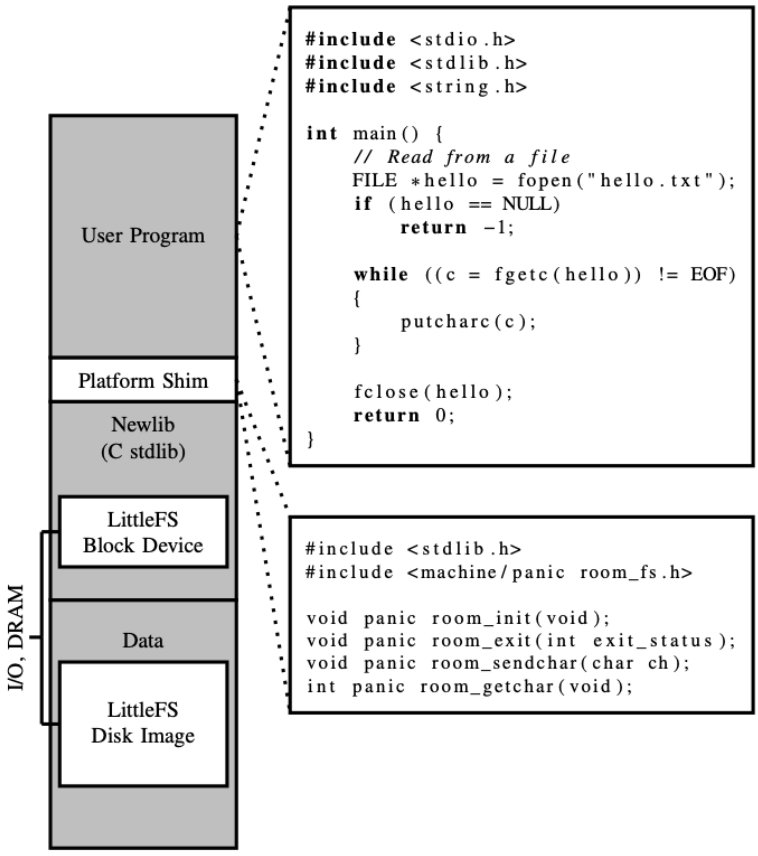


Figure 3.6: PanicRoom provides filesystem and I/O operations to POSIX applications. Platform support needs 4 non-portable syscalls: init, exit, sendchar and getchar. All other syscall functionality is platform-independently provided by the PanicRoom libgloss implementation. Programs cannot differentiate between PanicRoom or a full OS, simply running benchmarks which otherwise require esoteric environments.

Package (BSP) on top of the light-weight C standard library newlib [85]. newlib elegantly separates system-specific functionality into an easily portable portion called libgloss. PanicRoom implements the libgloss functionality using an open-source, lightweight, DRAM-based filesystem designed for embedded flash memories, ARM LittleFS [16].

As shown in Figure 3.6, PanicRoom implements file I/O system calls by translating them to LFS function calls, which in turn operate on memory. In contrast, proxy-based solutions such as RAW [250] and RISCV-PK [84] work by packaging I/O calls and tunneling to a host core, which executes the actual filesystem functionality.

PanicRoom eschews platform-dependent syscalls, whereas previous works require porting syscalls to open-source simulators, commercial simulators, FPGA emulation frameworks, ASIC test boards and PCIe hosted chips (see Table 2). A more subtle benefit is that transforming the I/O emulation from an asynchronous host interaction to a synchronous function, which makes execution deterministic and easily reproducible.

3.1.3 Case Studies: ZynqParrot in the Wild

We first developed ZynqParrot for undergraduate/graduate architecture class performing full-stack analysis and optimizations to an open-source Linux-capable RISC-V multicore. Our main goals were:

1. Provide a cheap, flexible platform for designing and analyzing microarchitectural modifications to the core.

2. Avoid supporting all laptop to FPGA mappings by standardizing the host system (the Zynq PS).

3. Synchronize co-simulation and co-emulation execution to minimize

FPGA debugging.

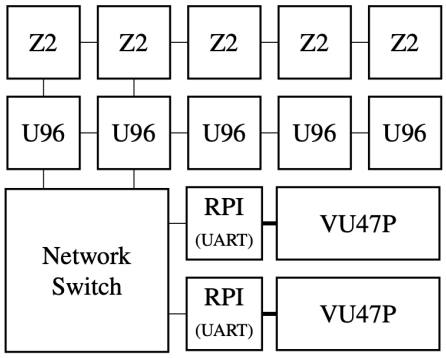
4. Make the platform robust to fatal RTL bugs by construction, impossible to hang the host system.

However, we quickly realized that the abstract capability of interacting out-of-band with arbitrary devices was applicable in a large number of diverse settings. In this section we describe a few use-cases that we have found for ZynqParrot since its inception.

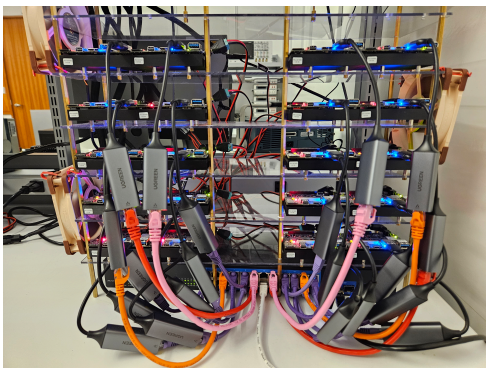
A Scale-Down Pandamonium Cluster

We built the first ZynqParrot on TUL Z2 [244], inexpensive educational boards available at an academic discount. Z2 boards are out-of-box compatible with the open-source Xilinx Pynq [269] SDK, providing a Python-based interface for bitstream programming, peripheral management and PS configuration, among many other convenience features. Because the Pynq software makes interaction with the Z2 boards so convenient, students can buy and develop on their own device. Most FPGA development boards, including the Z2, feature a watchdog timer which forces a reset upon hanging the board. In academia, this feature is invaluable to ensure that inexperienced students can always access their board.

Needing a more structured approach to coherently integrate a large number of boards, we designed Pandamonium, a scalable cluster of network-attached FPGAs running ZynqParrot. All ZynqParrot data is stored on a host system and shared with the boards through a network-based distributed file-system. For parallel development, team members log into each board to independently program and run experiments. Bulk regression can be run from standard job-scheduling software. The setup shown in Figure 3.7(b) is built with commodity components: USB-Ethernet controllers, a network switch, and hand-cut



(a) ZynqParrot clusters connect to a standard network switch to enable remote connections. While homogeneous clusters of Pynq Boards is the lowest maintenance options, some labs may be restricted to non-Zynq FPGAs and use small controllers such as Raspberry Pi to bridge to a VPS interface.



(b) A twenty-server Ultra96v2 cluster. Students can time-share boards for parallel builds and serialized, private experiments. By connecting the cluster to a network switch, students are able to work fully remotely, important during events such as the COVID-19 pandemic.

Figure 3.7: Pandamonium can be configured for a variety of Pareto frontiers along cost, capacity and design parallelism. For a design space exploration of heterogenous components, a fleet of small FPGAs may minimize build times, whereas for a suite of long-running benchmarks, medium-sized FPGAs may be able to complete overnight regressions on a full system.

plexiglass shelves. Comprising 20 Ultra96v2 boards, this Pandamonium cost around $4500 and supports multiple projects and Continuous Integration (*CI*) runners for a modestly sized research group. Based on Table 3.1, we estimate that a Pandamonium outperforms in TCO after less than a full year of usage.

Maintaining a Pandamonium is typically as simple as ensuring the central network switch is remotely accessible. If the watchdog timers are properly configured, any temporary glitch with the board has a fail-safe backup and connections can generally be restored after reboot. For further robustness, a remote network-attached reset switch (or Raspberry Pi [83]) removes the

need to physically reset the system even upon unlikely watchdog failures. A centralized job-scheduler can dynamically prevent interference between users and regression jobs.

As shown in Figure 3.7(a), ZynqParrot easily supports heterogeneous Pandamoniums as there are only two classes of network interface to maintain. Standard Zynq boards connect directly via Ethernet while non-Zynq parts tunnel through a network-attached UART-capable device such as a Raspberry Pi. In this way, clusters can simultaneously service a wide range of IP blocks that each may leverage specific board features. A diverse setup is ideal for a large continuous integration server as generic jobs can be assigned to minimally-sized boards, reducing regression time and improving energy efficiency.

To ensure the correctness of the BlackParrot execution and the observed performance data, user needs to be able to verify the correct execution of a benchmark during its runtime. For example, ZynqParrot's infrastructure can also be used to extract instruction commit information of a RISC-V core and cross-verify it with Dromajo [68] abstract software-based golden model of DUT in PS. Similarly, on each instruction commit and register write, corresponding information is written to asynchronous FIFOs that can gate the DUT clock while PS drains and verifies the data. Using this feature, ZynqParrot can be integrated into the CI as part of the chip development cycle. On a major change, FPGAs can be used to accelerate design verification using longer benchmarks that are impractical for RTL simulation.

Microarchitectural Optimization: CatchUp ALU

In addition to modelling interface timings, the P-Shell can be used for verification of BlackParrot logic by

extracting commit information and cross-verifying it with Dromajo [68], an abstract software-based golden model. Upon each commit PC, instruction metadata, and writeback information are written to asynchronous FIFOs. PS backpressure gates the DUT clock as PS drains the commits. With cycle-accurate co-emulation, ZynqParrot can be integrated into the CI as part of the chip development cycle.

Because ZynqParrot is able to maintain cycle-accuracy with arbitrary bandwidth instrumentation, it enables deep insight into subtle microarchitectural bottlenecks. Previous works have proposed sampling-based architectures that accurately detect long latency stalls such as page table walks and L1 cache misses, but cannot diagnose ultra fine-grained stall sources such as irregular dependency bubbles. Section 3.1.3 more thoroughly explores the trade-off between emulation speed and attribution accuracy, demonstrating the need for ultra fine-grained sampling to detect certain types of microarchitectural bottlenecks.

After adding synthesizable stall counters to the P-Shell, Figure 3.8 shows the cycle-stack breakdown of stalls during execution of CoreMark [89]. While CoreMark is a flawed benchmark for full-system characterization, it is widely used as proof of microarchitectural optimization. Additionally, it is an ideal demonstration of performance optimization frameworks since there is so little low-hanging fruit remaining. Because BlackParrot is an in-order pipeline with large L1 caches, load-use stalls are a primary performance bottleneck, accounting for 18% of stalls in CoreMark. Load-use stalls have two subtypes: load-arithmetic and load-control operations. For number crunching applications, load-arithmetic stalls prevent optimal operation of tight loops. For pointer chasing segments, load-control stalls add extra delays on every null check.

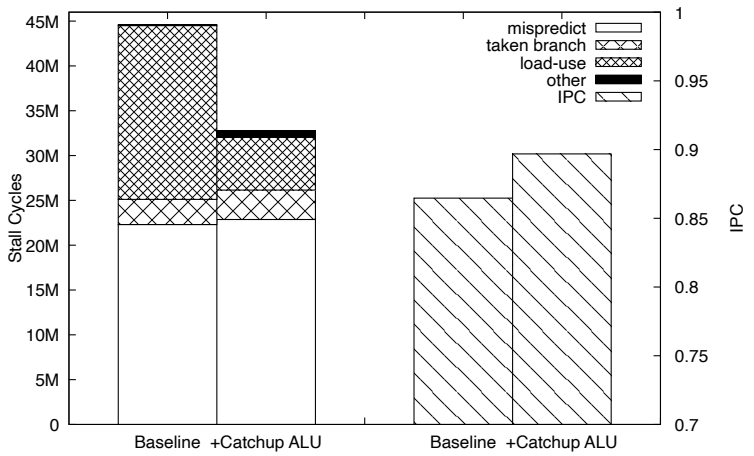


Figure 3.8: After basic core optimization, remaining low latency stalls (1-5 cycles) are difficult to detect via coarse-grained sampling. Tailored event counters can identify problematic categories, but lose PC association during aggregation. ZynqParrot allows VPS software to monitor stalls at a per-PC, per-cycle granularity.

To reduce load-use stalls, we add a *CatchUp* ALU which is a secondary ALU located serially after the first ALU. Catch-up ALUs are a common way to improve performance in in-order cores. Out-of-order execution is often able to tolerate L1 hit latencies, so extra resources are better spent on more parallel ALUs for wider issue. For in-order cores, however, single threaded performance is sensitive to head-of-line blocking and so Catch-up ALUs can provide a substantial benefit. After justifying the idea in a high-level (Scale-Up) simulation model, we implement an RTL version of the idea in ZynqParrot to evaluate marginal performance gains.

The CatchUp ALU resides in EX2, parallel with the second stage of the D$ access. When an integer or branch instruction has all dependencies met during issue, it is dispatched as normal to the Early ALU. Alternatively, when those dependencies are anticipated to be produced in EX2, the instruction is dispatched to the Catch-up ALU, which adds an additional cycle of latency,

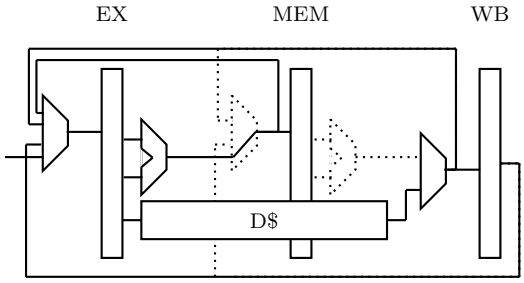


Figure 3.9: A second CatchUp ALU and set of bypass multiplexers allows the CatchUp ALU to execute pipelined instructions. However, a dependent non-integer instruction following a CatchUp operation will cause a bubble.

although fully-pipelined.

In addition to arithmetic operations, the Catch-up ALU also processes control flow instructions. Because RISC-V branch comparisons are easily transformed from existing subtraction and comparison operations, this support is cheap to add. However, this feature adds complexity to the handling of branch mispredictions. The BlackParrot pipeline resolves branches early in EX1 to reduce the misprediction penalty. In order for load-branch operations to take advantage of the CatchUp ALU, the pipeline must suppress PC mismatches in EX1. Now, when the CatchUp ALU detects a PC mismatch, the pipeline must be flushed in addition to redirecting the front-end. Therefore in BlackParrot, CatchUp ALU mispredictions are treated as synchronous exceptions, reusing their mechanism for replaying and recovering state. Figure 3.9 illustrates CatchUp ALU modifications to a sample five-stage pipeline.

As shown in Figure 3.9, the CatchUp ALU reduces load-use stalls from 43% of stalls to 18% of stalls, resulting in an overall 4% performance increase. There are additional stalls from dependencies on CatchUp ALU instructions, which now have an additional cycle of latency. However, these extra stalls

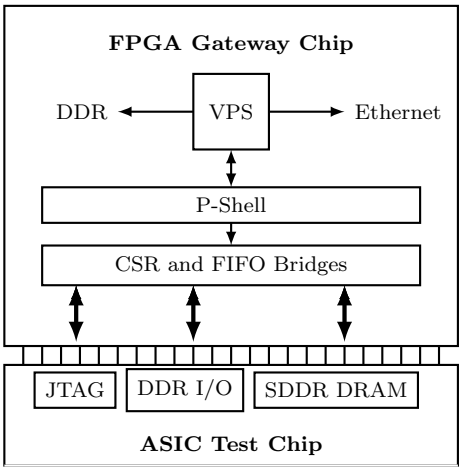


Figure 3.10: When taping out a chip verification is scoped sequentially, starting with C++ and RTL models and finishing with annotated gate-level simulations. Differences in simulation and bring-up environments prevent sharing infrastructure between pre- and post-silicon environments. ZynqParrot unifies tape-in and tape-out infrastructure, reducing maintenance times and accelerating bring-up.

do not diminish the gains from optimizing the more common load-use case. Interestingly, branch-related stalls increase by 1.04×, as deeper speculation past EX1 triggers additional mispredictions. A further optimization could restrict speculation only to branches which are predicted strongly taken which would increase load-branch stalls but should reduce CatchUp mispredictions. Leveraging cycle-accurate profiling with ZynqParrot allows architects to easily identify potential bottlenecks as well as confirm both the positive and negative effects of their proposed improvements.

ZynqParrot ASIC Bring-up Boards

During its initial evolution, ZynqParrot was also used as the design, prototype and bring-up infrastructure for a 14M gate, 28 nm ASIC developed by a boutique FPGA-based research and development firm. As a first genera-

tion test chip, operational mode fallbacks were essential. For instance, an experimental open-source LPDDR controller was taped out to accelerate applications. However, it was essential that the chip function enough to bring up all components independently. Additionally, it was not only possible but expected that during experiments on the chip to bring-up new subsystems, debug systemic issues and generate Shmoo plots [30] for operational PPA, the chip will fail to respond, generate illegal traffic or otherwise operate out of specification.

As shown in Figure 3.10, ZynqParrot leverages the P-Shell to allow bring-up software to interact with subsections of the DUT for large and slow gate-level simulations. Test chips are able to offload their memory controllers and I/O devices to the PS, which allows pre-silicon bring-up to verify fallback functionality much earlier in the life-cycle. In the other direction, the flexible P-Shell can connect to a wide variety of bitbanged configuration devices (JTAG, SPI, I2C), higher performance I/O links (CAN, UART, custom SERDES), and DRAM (LPDDR, SDDR, HBM). In addition to bringing up the actual chip with ZynqParrot, users are able to substitute out fully detailed simulated execution models for faster-simulating smoke tests.

Using ZynqParrot, the ASIC was able to pass initial smoke tests on the first day of bring-up despite power and packaging problems that prevented it from operating in normal voltage operations. By using non-blocking registers to bitbang and configure the attached test chip, the team was able to quickly iterate and explore modes without hanging the system, needing to re-flash the FPGA for different tests or debug the infrastructure itself.

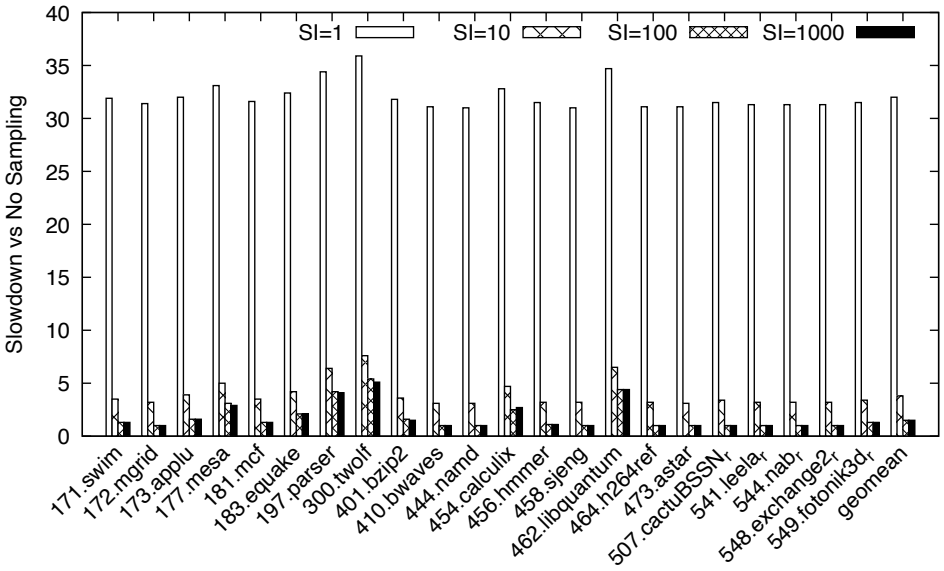


Figure 3.11: ZynqParrot is able to dynamically switch co-emulation speed for sample rate. As sampling granularity decreases down to single step, there is a 32× slowdown. Therefore, best practice is to identify regions-of-interest and change sampling frequency to match importance.

High Fidelity Sampling with ZynqParrot

While ZynqParrot accelerates design emulation, validating and optimizing performance at a Scale-Down granularity additionally requires deep introspection. Unfortunately, FPGA-enabled acceleration of designs is famously opaque and extracting microarchitecural information is unintuitive. Traditional solutions include using vendor IP such as JTAG scan-chains or Xilinx ILA [18] to extract signals. However, these solutions are slow, proprietary, have a high area overhead, and operate out-of-band, therefore lacking capability to maintain cycle-accuracy at full bandwidth. In this section, we explore how to leverage ZynqParrot's sampling infrastructure to characterize BlackParrot through time-proportional [96, 95] performance profiling.

To extract arbitrarily precise microarchitectural information from RTL, ZynqParrot leverages the same clock-gating mechanism used for I/O co-emulation. Users instantiate a parameterizable number of synthesizable performance counters in the P-Shell. These counters can be explicitly instantiated in the RTL, automatically generated by tools like FirePerf [138], or by hierarchically connecting PL counters to internal DUT signals. The latter does not require any modification to the DUT RTL and so is the simplest and least invasive solution.

When profiling BlackParrot, the profiler annotates each cycle of execution with a PC and event classification (stall type or commit), attributing at the commit stage to maintain time-proportionality. The DUT streams samples to PS across asynchronous FIFOs at a configurable sample rate, if necessary clock-gating identically to how ZynqParrot manages emulation of interface timings. Critically, due to the backpressure mechanism, tuning profiling granularity becomes a simple trade-off between slowdown and precision. TEA [96] and TIP [95] have demonstrated the benefits of Oracular stall classification, but

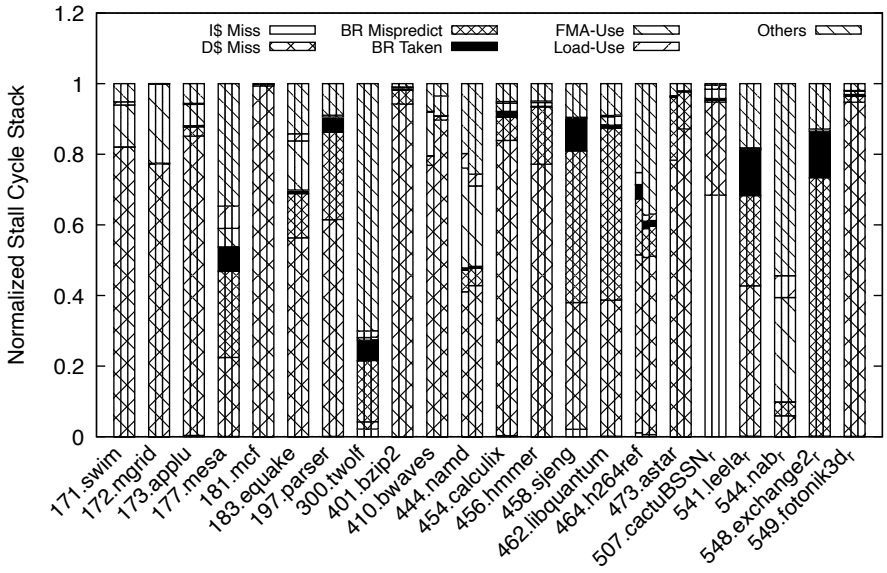


Figure 3.12: Stall stacks for sampling intervals 1, 10, and 100. Due to time-proportionality, stall stacks do not generally vary across sampling intervals. However, a few benchmarks such as 454.calculix and 464.h264ref have variances as high as 6.2%. Oracular sampling through ZynqParrot is able to accurately identify these stall sources.

concluded that the bandwidth overhead is impractical. Figure 3.11 illustrates that with ZynqParrot, an Oracle incurs only moderate overhead and enables unprecedented insight for performance debugging.

The PS post-processes the stall information asynchronous to the DUT. Based on this information, profiler runtimes may chose to manipulate the sampling rate, perturb the DUT with emulated I/O traffic or monitor the runtime execution. Figure 3.11 shows the emulation slowdown for performance sampling of the core with error-free per-cycle sampling or with different sampling frequencies that results from DUT clock gating. Note that due to other clock gating factors (such as maintaining a memory timing model), with increasing sampling interval, the slowdown curve saturates to a different value based on the running benchmark. Due to the time-proportional nature of the profiler,

stall stacks do not vary across sampling rates, as shown in Figure 3.12. There are two resulting modalities for performance profiling in ZynqParrot: running a coarse-grained regression suite to gain a sense of important stall categories, and running a fine-grained analysis to produce Oracular stall attributions to individual PCs.

3.1.4 Conclusion

Related Work

Platform	Cost Model	Design Ratio [0]	Host	Cycle-Accurate	Co-Simulation [1]	Coverage	Vendor-Agnostic	Open-Source
Commercial [2]	High-End Cluster	1:1	Proprietary	✓	✓	✓		
FireSim [33]	Cloud Rental	N:N	AWS F1	✓		✓		✓
SMAPPIC [19]	Cloud Rental	N:1	AWS F1					✓
FreezeTime [40]	High-End Board	1=1	PCIe Server	✓			✓	
Condominium	**Low-End Board/ Low-End Cluster**	**N=N**	**None**	✓	✓	✓	✓	✓

[0] The ratio of FPGAs:Designs in a single system emulation. Commercial tools map large hierarchies into large clusters. Firesim is able to emulate arbitrarily large systems using cloud auto-scaling. SMAPPIC is able to split large designs across FPGAs. FreezeTime maps a single design to a single FPGA. Finally, ZynqParrot maps a number of designs to a fixed-sized local cluster.

[1] Co-simulation refers to the ability to reproduce the cycle-exact output of a system emulation on an RTL simulator such as Verilator [211] (albeit at significant slowdown). This ability is essential in emulation-system debugging.

[2] We combine Synopsys Zebu [215], Cadence Palladium [53] and Mentor Veloce [160] with similar features and limitations.

While ZynqParrot shares similarities with many FPGA-accelerated prototyping platforms, its Scale-Down focus and aggressive portability make it uniquely cost and effort effective. In this section, we compare to existing projects which offer subsets of the features in ZynqParrot.

Gate-Level Accelerated Emulation Teams desiring a turnkey solution to RTL emulation employ commercial tools for FPGA-accelerated design modeling, such as Cadence Palladium [53], Synopsys Zebu [215] and Mentor Veloce [160]. Unfortunately this convenience is costly, with obfuscated pricing up to millions of dollars. In contrast ZynqParrot is free and open-source, with an initial required investment up to hundreds of dollars.

Emulating Large Systems with FPGAs FireSim [137], DIABLO [217] and SMAPPIC [69] focus on scaling up emulations to analyze large-scale designs such as datacenter-scale systems. They work by partitioning the system design over multiple FPGAs and using Ethernet-based token-passing systems to capture inter-node timing. Because they are based on AWS F1 [10] infrastructure, the emulation model relies on proprietary vendor libraries for the hardened AWS shell as well as PCIe DMA interfaces. As ZynqParrot focuses on single-node systems, it allows for local execution with open-source simulations, resulting in a much lower recurring cost. In contrast, a local version for a comparable F1 FPGA setup, may cost tens of thousands of dollars.

Decomposed FPGA Emulation Similar to a Scale-Down methodology, Protoflex [71] and FAST [67] accelerate performance analysis using FPGAs. However, they focus on acceleration of large, slow, cycle-accurate models, attempting to gain performance insights into systems too large to simulate in a reasonable time frame. In contrast, ZynqParrot allows for cycle-accurate emulation of arbitrary RTL so that architects can easily validate and debug performance with the deep introspection that RTL provides.

FreezeTime [165] uses time multiplexing for architectural virtualization of system components. Similar to ZynqParrot, Freezetime leverages BUFGCE FPGA primitives to stall emulated blocks while virtualized blocks process cycle-accurate timing models. However, ZynqParrot achieves greater flexibility and lower resource overheads by executing standard C++ timing models in the P-Shell rather than custom control logic per virtualized interface.

FPGA-Accelerated Performance Analysis While custom cycle-level simulators and silicon performance counters are state-of-art for commercial

performance validation, researchers have also proposed using accelerated sampling for microarchitectural debugging.

FirePerf [138] provides two categories of microarchitectural analysis: commit tracing via TraceRV and out-of-band hardware profiling via AutoCounters. ZynqParrot supports not only commit tracing and out-of-band event counters via P-Shell CSRs but also dispatch-time stall tracing, allowing for deeper debugging insights. Additionally, ZynqParrot is written in standard SystemVerilog rather than Chisel [28], making it more familiar to hardware designers.

TEA [96] and TIP [95] propose time-proportional event analysis by creating Per-Instruction Cycle Stacks (*PICS*) to unify performance profiling and performance event analysis. While TEA and TIP are able to accurately ascribe microarchitectural events on average, they rely on statistical sampling by periodically interrupting the program that disrupts non-interference. Because ZynqParrot combines commit-stage cycle attribution with cycle accurate tracing, it is able to accurately attribute stalls without any sampling error, as well as trade co-emulation speed for sampling accuracy.

FPGA-based Coverage Collection FirePerf [138] injects synthesizable coverpoints and then extracts coverage through a scan-chain. Instead, ZynqParrot automatically instruments designs and extracts stall data through the P-Shell, without dedicated scan hardware. Simulator Independent Coverage [150] introduces a flow for injecting hardware coverpoints into Chisel [28] designs through the introduction of a new *cover* keyword. In contrast, ZynqParrot coverage instrumentation uses standard SystemVerilog primitives.

Future Work

While ZynqParrot has been successfully used for classwork, academic research and industrial prototyping, in this section we consider several directions to enhance versatility and robustness:

1. While Scale-Down methodology greatly benefits prototyping iteration time, subdividing the design into co-emulatable subcomponents is generally a manual process. Automatically partitioning commercial-size architecturables could lead to faster full-design iterations, detecting bugs earlier in the design schedule.

2. Although ZynqParrot currently supports DRAM, UART and Ethernet as well as a small set of demonstration AXI-lite masters and clients, we anticipate community development of a diverse open-source library of functional and timing models for common peripherals.

3. ZynqParrot was intentionally developed to minimize dependence on proprietary vendor IP. While current implementations use Xilinx Zynq and Virtex parts, only a small subset of the infrastructure (¡ 100 lines of TCL) must be ported to support new boards. We hope in the future to support Intel and Lattice alternatives to broaden the reach of ZynqParrot to teams which are bound to these vendors.

Final Thoughts

We present ZynqParrot, a Scale-Down FPGA-based modelling platform capable of non-interfering, cycle-accurate co-emulations of arbitrary RTL designs. Vendor-agnostic and fully open-source, ZynqParrot provides architects with a accurate, convenient and low-cost infrastructure to prototype designs. ZynqParrot is a cheaper alternative to FPGA cloud infrastructures

for small-scale experiments and provides vendor-agnosticism. An iterative Scale-Down/Scale-Up methodology allows architects to focus on subtle microarchitectural optimizations and avoid re-analysis of issues that only exist at either scale.

In this work we have explored diverse use-cases for ZynqParrot distributed acceleration for a class of architecture students, performance debugging, functional verification and tapeout bringup for a complex 28 nm SoC. These studies are meant to illustrate ZynqParrot's fitness for teaching, research and industrial development. We examined instrastructure enablements: ultra fine-grained sampling and hybrid FPGA-software coverpoint strategies. We believe the ZynqParrot, the P-Shell interface and its open-source co-simulation libraries will accelerate RTL prototyping across teams and fields, helping develop better architectures faster.

Chapter 4

CONCLUSION

4.1 Contributions

The works described in this thesis have been battle-tested in multiple contexts, from academic research to commercial ASIC development. They have been presented at various conferences and workshops and have received positive feedback from the community.

- Papers published or under revision:
  - BlackParrot: An agile open-source RISC-V multicore for accelerator SoCs (IEEE Micro 2020) (1st author)
  - Noc Symbiosis (NOCS 2020) (1st author)
  - ZynqParrot: A Scale-Down Approach to Cycle-Accurate, FPGA-Accelerated Co-Emulation (Arxiv 2025) (1st author)
  - The BlackParrot BedRock Cache Coherence System (Arxiv 2025) (2nd author)
  - RACE: RISC-V SoC for en/decryption acceleration on the edge for homomorphic computation (ISLPED 2022) (3rd author)
  - RISE: RISC-V SoC for En/decryption acceleration on the edge for homomorphic encryption (IEEE VLSI 2023) (3rd author)
  - Scaling Program Synthesis Based Technology Mapping with Equal-

ity Saturation (WOSET 2024) (3rd author)

  - Scalable, Programmable and Dense: The HammerBlade Open-Source RISC-V Manycore (ISCA 2024) (Nth author)

- Chips designed using infrastructure:
  - BP0: 4-core BlackParrot (GF12 3x3mm)
  - BP1: TSMC40 4x6mm: 4 core multicore (TSMC40, tapein only)
  - BigBlade: 16 x (2 unicore + CGRA + 128 core manycore) (GF12 10x10mm)
  - MaxSDRv1: 1 big, 2 little cores, 192 core manycore, open-source DRAM control, scalable clock generator, scalable DLL (TSMC28 4x6mm)
  - TT-DLL: all-digital delay-locked-loop (Sky130 160x100um)
  - MaxSDRv2: 1 big, 2 little cores, 192 core manycore, open-source DRAM control, scalable clock generator, scalable DLL (TSMC28 4x6mm, tapein only)
  - MiniBlade: DBI testing infrastructure, scalable clock generator (GF12 1x1mm, tapein only)

- Talks given to publicize projects:
  - bsg_pearls: Effortlessly Synthesizable Building Blocks That Work Right Out of the Shell (Latchup 2025)
  - bsg_tag: A minimal open-source ASIC configuration system (Latchup 2023)
  - BlackParrot: An Agile Open Source RISC-V Multicore for Accel-

erator SoCs (FOSDEM 2020)

- BaseJump STL: A Standard Template Library for Hardware Design (Latchup 2019)
- BlackParrot: a Cache-Coherent RV64G Multicore For and By the World (RISC-V Summit WI 2020)
- BlackParrot: a Cache-Coherent RV64G Multicore For and By the World (International Conference on Supercomputing 2020)

4.2 Future Work

These projects are living research platforms. focusing on improving usability and maintainability. One of the primary goals of this thesis is to ensure these tools are usable by the new generation of architecture students without maintainer intervention. There are several potential areas for further development:

- **BlackParrot:** Publication of a canonical BlackParrot SoC design with an open-source Ethernet controller and LPDDR1 DRAM controller to serve as a reference design for future users. The IP has already been developed and tested, so the remaining work will be to provide a complete, ready-to-use system that can be easily adapted for various applications.
- **ZynqParrot:** Support for the Kria and Alvea SoC families, which will allow users to leverage the ZynqParrot infrastructure for a wider range of applications. Most of the infrastructure is board-agnostic, so this task will primarily involve the addition of board-specific drivers and peripherals.
- **BSG Pearls:** Example applications of BSG Pearls Assemblies, focus-

ing on end-to-end flows of synthesis, simulation and verification flows. Individual IP blocks have already been developed and tested, so the remaining work is primarily to package and document the library for public use.

4.3 Final Thoughts

Despite worthy efforts of the research community, Agile Hardware Design methods have not yet achieved widespread adoption in the industry. This thesis argues that remaining barriers are qualitative rather than quantitative, and proposes that open-source EDA tools and hardware can help realize the benefits of Agile practices, namely: design reuse, verification effort, and backend iteration time.

While Agile Hardware Design offers the promise of faster and more cost-effective ASIC development, its practical adoption is hindered by qualitative challenges such as complex library integration, verification environment discrepancies, and backend flow unpredictability. This thesis addresses these barriers through three open-source, silicon-proven projects —– BSG Pearls, BlackParrot, and ZynqParrot—– which together provide modular system integration, unified verification infrastructure, and hierarchy-insensitive middleware assembly. By lowering integration and verification hurdles, these tools enhance the feasibility of Agile Hardware Design in a wide variety of domains.

BIBLIOGRAPHY

[1] N. Abeyratne, R. Das, Q. Li, K. Sewell, B. Giridhar, R. G. Dreslinski, D. Blaauw, and T. Mudge. Scaling towards kilo-core processors with asymmetric high-radix topologies. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 496–507, Feb 2013.

[2] A. Agarwal, S. Amarasinghe, R. Barua, M. Frank, W. Lee, V. Sarkar, D. Srikrishna, and M. Taylor. The RAW compiler project. In *Proceedings of the Second SUIF Compiler Workshop*, pages 21–23, 1997.

[3] Anant Agarwal, Ricardo Bianchini, David Chaiken, Kirk L. Johnson, David Kranz, John Kubiatowicz, Beng-Hong Lim, Kenneth Mackenzie, and Donald Yeung. The MIT alewife machine. In *Proceedings of the 22nd annual international symposium on Computer architecture - ISCA '95*. ACM Press, 1995.

[4] Tutu Ajayi, Khalid Al-Hawaj, Aporva Amarnath, Steve Dai, Scott Davidson, Paul Gao, Gai Liu, Atieh Lotfi, Julian Puscar, Anuj Rao, Austin Rovinski, Loai Salem, Ningxiao Sun, Christopher Torng, Luis Vega, Bandhav Veluri, Xiaoyang Wang, Shaolin Xie, Chun Zhao, Ritchie Zhao, Christopher Batten, Ronald G. Dreslinski, Ian Galton, Rajesh K. Gupta, Patrick P. Mercier, Mani Srivastava, Michael Bedford Taylor, and Zhiru Zhang. Celerity: An Open Source RISC-V Tiered Accelerator Fabric. In *HOTCHIPS*, Aug 2017.

[5] Ajayi et al. Toward an open-source digital flow: First learnings from the openroad project. In *DAC*, 2019.

[6] Alibaba. `https://www.alibabacloud.com/product/computing`, 2023.

[7] Chips Alliance. https://github.com/chipsalliance/surelog, 2023.

[8] Charles J Alpert and Gustavo E Tellez. The importance of routing congestion analysis. *DAC Knowledge Center Online Article*, 2010.

[9] Alric Althoff, Joseph McMahan, Luis Vega, Scott Davidson, Timothy Sherwood, Michael Taylor, and Ryan Kastner. Hiding Intermittant

Information Leakage with Architectural Support for Blinking. In *International Symposium on Computer Architecture (ISCA)*, 2018.

[10] Amazon. Amazon web services. 2022. amazon ec2 f1 instances. `https://aws.amazon.com/ec2/instance-types/f1/`, 2023.

[11] Amazon. `https://aws.amazon.com/ec2/spot/pricing/`, 2023.

[12] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *Ieee Micro*, 40(4):10–21, 2020.

[13] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4):10–21, 2020.

[14] AntMicro. https://github.com/renode/renode, 2023.

[15] ARM. `https://developer.arm.com/documentation/102202/0300/AXI-protocol-overview`, 2023.

[16] ARM. `https://github.com/littlefs-project/littlefs`, 2023.

[17] Manish Arora, Jack Sampson, Nathan Goulding-Hotta, Jonathan Babb, Ganesh Venkatesh, Michael Bedford Taylor, and Steven Swanson. Reducing the Energy Cost of Irregular Code Bases in Soft Processor Systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2011.

[18] Khalil Arshak, Essa Jafer, and Christian Ibala. Testing fpga based digital system using xilinx chipscope logic analyzer. In *2006 29th International Spring Seminar on Electronics Technology*, pages 355–360. IEEE, 2006.

[19] Asanovic et al. The rocket chip generator. *UC Berkeley EECS Tech Report UCB/EECS-2016-17*, 2016(17):1–12, 2016.

[20] Saahil Athrij. Vectorizing the Hamerblade Compiler. Master's thesis, University of Washington, 2024.

[21] AVnet. https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/ultra96-v2/, 2023.

[22] AVnet. https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/ultra96-v2/, 2023.

[23] Z. Azad, G. Yang, R. Agrawal, D. Petrisko, M. Taylor, and A. Joshi. Race: Risc-v soc for en/decryption acceleration on the edge for homomorphic computation. In *ISLPED*, 2022.

[24] Zahra Azad, Guowei Yang, Rashmi Agrawal, Daniel Petrisko, Michael Taylor, and Ajay Joshi. Race: Risc-v soc for en/decryption acceleration on the edge for homomorphic computation. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 1–6, 2022.

[25] Zahra Azad, Guowei Yang, Rashmi Agrawal, Daniel Petrisko, Michael Taylor, and Ajay Joshi. RISE: RISC-V SoC for En/Decryption Acceleration on the Edge for Homomorphic Encryption. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[26] Zahra Azad, Guowei Yang, Rashmi Agrawal, Daniel Petrisko, Michael Taylor, and Ajay Joshi. Rise: Risc-v soc for en/decryption acceleration on the edge for homomorphic encryption. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(10):1523–1536, 2023.

[27] Jonathan Babb, Matthew Frank, Victor Lee, Elliot Waingold, Rajeev Barua, Michael Taylor, Jang Kim, Srikrishna Devabhaktuni, and Anant Agarwal. The Raw Benchmark Suite: Computation Structures for General Purpose Computing. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 1997.

[28] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: Constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, page 1216–1225, New York, NY, USA, 2012. Association for Computing Machinery.

[29] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: constructing hardware in a scala embedded language. In *Proceedings of the 49th annual design automation conference*, pages 1216–1225, 2012.

[30] Keith Baker and Jos Van Beers. Shmoo plotting: The black art of ic testing. *IEEE Design & Test of Computers*, 14(3):90–97, 1997.

[31] James Balfour and William J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *ACM International Conference on Supercomputing*

25th Anniversary Volume, page 390–401, New York, NY, USA, 2006. Association for Computing Machinery.

[32] Jonathan Balkind, Katie Lim, Fei Gao, Jinzheng Tu, David Wentzlaff, Michael Schaffner, Florian Zaruba, and Luca Benini. Openpiton+ ariane: The first open-source, smp linux-booting risc-v system scaling from one to many cores. In *Workshop on Computer Architecture Research with RISC-V (CARRV)*, pages 1–6, Pheonix, AZ, USA, 2019. IEEE.

[33] Jonathan Balkind, Katie Lim, Michael Schaffner, Fei Gao, Grigory Chirkov, Ang Li, Alexey Lavrov, Tri M Nguyen, Yaosheng Fu, Florian Zaruba, et al. Byoc: a" bring your own core" framework for heterogeneous-isa research. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 699–714, Lausanne, Switzerland, 2020. ACM.

[34] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, et al. Openpiton: An open source manycore research framework. *ACM SIGPLAN Notices*, 51(4):217–232, 2016.

[35] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. Openpiton: An open source manycore research framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 217–232, New York, NY, USA, 2016. ACM.

[36] Luiz André Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. Piranha. In *Proceedings of the 27th annual international symposium on Computer architecture - ISCA '00*. ACM Press, 2000.

[37] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. Tile64 - processor: A 64-core soc with mesh interconnect. In *IEEE International Solid-State Circuits Conference*, pages 88–598, Feb 2008.

[38] B. Beresini, S. Ricketts, and M.B. Taylor. Unifying manycore and fpga

processing with the RUSH architecture. In *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, pages 22 –28, 2011.

[39] Bespoke Silicon Group. Blackparrot: A linux-capable accelerator host multicore. `https://github.com/black-parrot/black-parrot/tree/archive/pparrot_ee477_wi19`, 2020.

[40] Bespoke Silicon Group. BlackParrot: A Linux-Capable Accelerator Host Multicore. `https://github.com/black-parrot/black-parrot/tree/tapeout_bp_rc2`, 2020.

[41] Bespoke Silicon Group. BlackParrot: A Linux-Capable Accelerator Host Multicore. `https://github.com/black-parrot/black-parrot/tree/tapeout_bp_rc3`, 2020.

[42] Vikram Bhatt, Nathan Goulding-Hotta, Qiaoshi Zheng, Jack Sampson, Steve Swanson, and Michael B. Taylor. Sichrome: Mobile web browsing in Hardware to save Energy . In *Dark Silicon Workshop, ISCA*, 2012.

[43] Christian Bienia, Sanjeev Kumar, and Kai Li. PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In David Christie, Alan Lee, Onur Mutlu, and Benjamin G. Zorn, editors, *4th International Symposium on Workload Characterization (IISWC 2008), Seattle, Washington, USA, September 14-16, 2008*, pages 47–56. IEEE Computer Society, 2008.

[44] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81, 2008.

[45] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[46] Mark Bohr. A 30 year retrospective on dennard's mosfet scaling paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, 2007.

[47] Ajay Brahmakshatriya, Emily Furst, Victor Ying, Claire Hsu, Max Ruttenberg, Yunming Zhang, Tommy Jung, Dustin Richmond, Michael Taylor, Julian Shun, Mark Oskin, Daniel Sanchez, and Saman Amarasinghe. Taming the zoo: A unified graph compiler framework for novel architectures. In *ISCA*, 2021.

[48] Brahme and Abraham. Functional testing of microprocessors. *IEEE transactions on Computers*, 100(6):475–485, 1984.

[49] HC Brearley. Illiac ii-a short description and annotated bibliography. *IEEE Transactions on Electronic Computers*, pages 399–403, 2006.

[50] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 41–42, 2018.

[51] Buildroot.

[52] Busybox.

[53] Cadence. `https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html`, 2023.

[54] Sadullah Canakci, Leila Delshadtehrani, Furkan Eris, Michael Bedford Taylor, Manuel Egele, and Ajay Joshi. Directfuzz: Automated test generation for rtl designs using directed graybox fuzzing. In *DAC*, 2021.

[55] Sadullah Canakci, Chathura Rajapaksha, Leila Delshadtehrani, Anoop Nataraja, Michael Bedford Taylor, Manuel Egele, and Ajay Joshi. Processorfuzz: Processor fuzzing with control and status registers guidance. In *HOST*, 2023.

[56] Luca P Carloni, Kenneth L McMillan, and Alberto L Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 20(9):1059–1076, 2002.

[57] Celio et al. The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor. *UC Berkeley EECS Tech. Rep. UCB/EECS-2015-167*, 2015(167):1–12, 2015.

[58] Lucien M. Censier and Paul Feautrier. A new solution to coherence problems in multicache systems. *IEEE Trans. Computers*, 27(12):1112–1118, 1978.

[59] D. Chaiken and A. Agarwal. Software-extended coherent shared memory: performance and cost. *ACM SIGARCH Computer Architecture News*, 22(2):314–324, April 1994.

[60] Aliaksei Chapyzhenka. Wavedrom.

[61] Chen Chen, Xiaoyan Xiang, Chang Liu, Yunhai Shang, Ren Guo, Dongqi Liu, Yimin Lu, Ziyi Hao, Jiahui Luo, Zhijian Chen, et al. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance risc-v processor with vector extension: Industrial product. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 52–64. IEEE, 2020.

[62] Y. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016.

[63] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[64] Y. Chen, T. Yang, J. Emer, and V. Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[65] Lin Cheng, Peitian Pan, Zhongyuan Zhao, Krithik Ranjan, Jack Weber, Bandhav Veluri, Seyed Borna Ehsani, Max Ruttenberg, Dai Cheol Jung, Preslav Ivanov, Dustin Richmond, Michael B. Taylor, Zhiru Zhang, and Christopher Batten. A tensor processing framework for cpu-manycore heterogeneous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1620–1635, 2022.

[66] Lin Cheng, Max Ruttenberg, Dai Cheol Jung, Dustin Richmond, Michael Taylor, Mark Oskin, and Christopher Batten. Beyond Static Parallel Loops: Supporting Dynamic Task Parallelism on Manycore Architectures with Software-Managed Scratchpad Memories. In *ASPLOS*, 2023.

[67] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil A Patil, William Reinhart, Darrel Eric Johnson, Jebediah Keefe, and Hari Angepat. Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 249–261, Washington, DC, USA, 2007. IEEE, IEEE.

[68] ChipsAlliance. `https://github.com/chipsalliance/dromajo`, 2023.

[69] Grigory Chirkov and David Wentzlaff. Smappic: Scalable multi-fpga architecture prototype platform in the cloud. In *Proceedings of the 28th ACM International Conference on Architectural Support for Program-*

ming Languages and Operating Systems, Volume 2, pages 733–746, New York, NY, USA, 2023. ACM.

[70] Yuan-Mao Chueh. A Complete Open Source Network Stack For BlackParrot. Master's thesis, University of Washington, 2022.

[71] Eric S Chung, Michael K Papamichael, Eriko Nurvitadhi, James C Hoe, Ken Mai, and Babak Falsafi. Protoflex: Towards scalable, full-system multiprocessor simulations using fpgas. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2(2):1–32, 2009.

[72] M Clarke, D Hammerschlag, M Rardon, and A Sood. Eliminating routing congestion issues with logic synthesis. cadence white paper. *Cadence White Paper*, 2011.

[73] Henry Cook, Wesley Terpstra, and Yunsup Lee. Diplomatic design patterns: A tilelink case study. In *1st Workshop on Computer Architecture Research with RISC-V*, volume 23, pages 1–6, Boston, MA, USA, 2017. IEEE.

[74] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 684–689, June 2001.

[75] Deeksha Dangwal, Georgios Tzimpragos, and Timothy Sherwood. Agile hardware development and instrumentation with pyrtl. *IEEE Micro*, 40(4):76–84, 2020.

[76] S. Davidson, S. Xie, C. Torng, K. Al-Hawai, A. Rovinski, T. Ajayi, L. Vega, C. Zhao, R. Zhao, S. Dai, A. Amarnath, B. Veluri, P. Gao, A. Rao, G. Liu, R. K. Gupta, Z. Zhang, R. Dreslinski, C. Batten, and M. B. Taylor. The celerity open-source 511-core risc-v tiered accelerator fabric: Fast architectures and design methodologies for fast chips. *IEEE Micro*, 38(2):30–41, Mar 2018.

[77] Scott Davidson, Shaolin Xie, Chris 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 Dreslinski, Christopher Batten, and Michael Bedford Taylor. The Celerity Open-Source 511-core RISC-V Tiered Accelerator Fabric. *Micro, IEEE*, Mar/Apr. 2018.

[78] Enjoy Digital. https://github.com/enjoy-digital/litex?tab=readme-ov-file, 2025.

[79] D.L. Dill, A.J. Drexler, A.J. Hu, and C.H. Yang. Protocol verification

as a hardware design aid. In *Proceedings 1992 IEEE International Conference on Computer Design: VLSI in Computers & Processors*. IEEE Comput. Soc. Press, 2004.

[80] R. Dreslinski, K. Sewell, T. Manville, S. Satpathy, N. Pinckney, G. Blake, M. Cieslak, R. Das, T. Wenisch, D. Sylvester, D. Blaauw, and T. Mudge. Swizzle switch: A self-arbitrating high-radix crossbar for noc systems. In *2012 IEEE Hot Chips 24 Symposium (HCS)*, pages 1–44, Aug 2012.

[81] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. Shidiannao: Shifting vision processing closer to the sensor. In *International Symposium on Computer Architecture (ISCA)*, pages 92–104, June 2015.

[82] Hadi Esmaeilzadeh and Michael Bedford Taylor. Open Source Hardware: Stone Soups and Not Stone Satues, Please. In *SIGARCH Computer Architecture Today*, Dec 2017.

[83] Raspberry Pi Foundation. https://www.raspberrypi.org, 2023.

[84] RISC-V Foundation. https://github.com/riscv-software-src/riscv-pk, 2023.

[85] FSF. `https://sourceware.org/newlib/`, 2023.

[86] FSF. `https://www.gnu.org/software/libc/`, 2023.

[87] César Fuguet. Hpdcache: Open-source high-performance l1 data cache for risc-v cores. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, pages 377–378, Bologna, Italy, 2023. ACM.

[88] Emily Furst. *Code Generation and Optimization of Graph Programs on a Manycore Architecture*. PhD thesis, University of Washington, 2021.

[89] Shay Gal-On and Markus Levy. Exploring coremark a benchmark maximizing simplicity and efficacy. *The Embedded Microprocessor Benchmark Consortium*, 2012.

[90] S. Garcia, Donghwan Jeon, C. Louie, and M.B. Taylor. The Kremlin Oracle for Sequential Code Parallelization. *Micro, IEEE*, 32(4):42–53, July-Aug. 2012.

[91] Saturnino Garcia, Donghwan Jeon, Chris Louie, Sravanthi Kota Venkata, and Michael Bedford Taylor. Bridging the Parallelization Gap: Automating Parallelism Discovery and Planning. In *USENIX Workshop on Hot Topics in Parallelism (HOTPAR)*, 2010.

[92] Saturnino Garcia, Donghwan Jeon, Chris Louie, and Michael Bedford Taylor. Kremlin: Rethinking and Rebooting gprof for the Multicore Age. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*, 2011.

[93] Farzam Gilani. *Methodologies for Accelerated Open-Source Hardware Verification and Optimization*. PhD thesis, University of Washington, 2025.

[94] Google. Riscv-dv. `https://github.com/google/riscv-dv`.

[95] Bjorn Gottschall, Lieven Eeckhout, and Magnus Jahre. Tip: Time-proportional instruction profiling. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 15–27, Piscataway, NJ, USA, 2021. IEEE.

[96] Bjorn Gottschall, Lieven Eeckhout, and Magnus Jahre. Tea: Time-proportional event analysis. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–13, Piscataway, NJ, USA, 2023. IEEE.

[97] Nathan Goulding, Jack Sampson, Ganesh Venkatesh, Saturnino Garcia, Joe Auricchio, Jonathan Babb, Michael B Taylor, and Steven Swanson. Greendroid: A mobile application processor for a future of dark silicon. In *2010 IEEE Hot Chips 22 Symposium (HCS)*, pages 1–39. IEEE, 2010.

[98] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. Taylor. The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future. *Micro, IEEE*, pages 86–95, March 2011.

[99] Nathan Goulding-Hotta. *Specialization as a Candle in the Dark Silicon Regime*. PhD thesis, University of California, San Diego, 2020.

[100] Nathan Goulding-Hotta, Jack Sampson, Qiaoshi Zheng, Vikram Bhatt, Steven Swanson, and Michael Taylor. GreenDroid: An Architecture for the Dark Silicon Age. In *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2012.

[101] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger. Implementation and evaluation of on-chip network architectures. In *2006 International Conference on Computer Design*, pages 477–484, Oct 2006.

[102] Matthias Gries, Ulrich Hoffmann, Michael Konow, and Michael Riepen.

SCC: A flexible architecture for many-core platform research. *Comput. Sci. Eng.*, 13(6):79–83, 2011.

[103] Anshuman Gupta, Jack Sampson, and Michael Bedford Taylor. DR-SNUCA: An energy-scalable dynamically partitioned cache. In *International Conference on Computer Design (ICCD)*, 2013.

[104] Anshuman Gupta, Jack Sampson, and Michael Bedford Taylor. Time Cube: A Manycore Embedded Processor with Interference-Agnostic Progress Tracking. In *International Conference On Embedded Computer Systems: Architectures, Modeling And Simulation (SAMOS)*, 2013.

[105] Anshuman Gupta, Jack Sampson, and Michael Bedford Taylor. Qualitytime: A simple online technique for quantifying multicore execution efficiency. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014.

[106] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*, pages 3–14. IEEE, 2001.

[107] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism*, 7(4):1–28, 2005.

[108] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002. (MICRO-35). Proceedings.*, pages 294–305, Nov 2002.

[109] John Hauser. *Berkeley HardFloat.* `http://www.jhauser.us/arithmetic/HardFloat.html`.

[110] Byron Hawkins, Brian Demsky, and Michael Bedford Taylor. BlackBox: Lightweight Security Monitoring for COTS Binaries. In *Code Generation and Optimization*, 2016.

[111] Byron Hawkins, Brian Demsky, and Michael Bedford Taylor. A Runtime Approach to Security and Privacy. In *European Security and Privacy*, 2016.

[112] John Heinlein. *Optimized Multiprocessor Communication and Synchronization Using a Programmable Protocol Engine.* PhD thesis, Stanford University, Stanford, CA, USA, 1998.

[113] Mark A. Heinrich, Jeffrey Kuskin, David Ofelt, John Heinlein, Joel Baxter, Jaswinder Pal Singh, Richard Simoni, Kourosh Gharachorloo, David Nakahira, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, and John L. Hennessy. The performance impact of flexibility in the stanford FLASH multiprocessor. In Forest Baskett and Douglas W. Clark, editors, *ASPLOS-VI Proceedings - Sixth International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, California, USA, October 4-7, 1994*, pages 274–285. ACM Press, 1994.

[114] Mark Andrew Heinrich. *The Performance and Scalability of Distributed Shared-Memory Cache Coherence Protocols*. PhD thesis, Stanford University, Stanford, CA, USA, 1998.

[115] John L Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33(7):28–35, 2000.

[116] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[117] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61, Sep. 2007.

[118] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. V. D. Wijngaart, and T. Mattson. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 108–109, 2010.

[119] Hu, Zhu, Taylor, and Cheng. FPGA Global Routing Architecture Optimization Using a Multicommodity Flow Approach . In *ICCD*, 2007.

[120] IntelSCC.

[121] Donghwan Jeon, Saturnino Garcia, Chris Louie, Sravanthi Kota Venkata, and Michael Bedford Taylor. Kremlin: Like gprof, but for Parallelization. In *Principles and Practice of Parallel Programming (PPoPP)*, 2011.

[122] Donghwan Jeon, Saturnino Garcia, Chris Louie, and Michael Bedford Taylor. Kismet: Parallel Speedup Estimates for Serial Programs. In

Conference on Object-Oriented Programming, Systems, Language and Applications (OOPSLA), 2011.

[123] Donghwan Jeon, Saturnino Garcia, Chris Louie, and Michael Bedford Taylor. Parkour: Parallel Speedup Estimates from Serial Code. In *USENIX Workshop on Hot Topics in Parallelism (HOTPAR)*, 2011.

[124] Donghwan Jeon, Saturnino Garcia, and Michael Bedford Taylor. Skadu: Efficient Vector Shadow Memories for Poly-scopic Program Analysis. In *Conference on Code Generation and Optimization (CGO)*, 2013.

[125] Dai Cheol Jung. Caches for Complex Open Source System-on-Chip Designs. Master's thesis, University of Washington, 2019.

[126] Dai Cheol Jung, Scott Davidson, Chun Zhao, Dustin Richmond, and Michael Bedford Taylor. Ruche Networks: Wire-Maximal, No-Fuss NoCs. In *NOCS*, 2020.

[127] Dai Cheol Jung, Max Ruttenberg, Paul Gao, Scott Davidson, Daniel Petrisko, Kangli Li, Aditya K Kamath, Lin Cheng, Shaolin Xie, Peitian Pan, et al. Scalable, programmable and dense: The hammerblade open-source risc-v manycore. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 770–784. IEEE, 2024.

[128] Dai Cheol Jung, Max Ruttenberg, Paul Gao, Scott Davidson, Daniel Petrisko, Kangli Li, Aditya K Kamath, Lin Cheng, Shaolin Xie, Peitian Pan, Zhongyuan Zhao, Zichao Yue, Bandhav Veluri, Sripathi Muralitharan, Adrian Sampson, Andrew Lumsdaine, Zhiru Zhang, Christopher Batten, Mark Oskin, Dustin Richmond, and Michael Bedford Taylor. Scalable, Programmable and Dense: The HammerBlade Open-Source RISC-V Manycore. In *ISCA*, 2024.

[129] A. B. Kahng. New directions for learning-based ic design tools and methodologies. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 405–410, 2018.

[130] A. B. Kahng, Bin Li, L. Peh, and K. Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 423–428, April 2009.

[131] A. B. Kahng, B. Li, L. Peh, and K. Samadi. Orion 2.0: A power-area simulator for interconnection networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(1):191–196, Jan 2012.

[132] A. B. Kahng, B. Lin, and S. Nath. Explicit modeling of control and data for improved noc router estimation. In *DAC Design Automation Conference 2012*, pages 392–397, June 2012.

[133] A. B. Kahng, B. Lin, and S. Nath. Orion3.0: A comprehensive noc router estimation tool. *IEEE Embedded Systems Letters*, 7(2):41–45, June 2015.

[134] A. B. Kahng, B. Lin, and K. Samadi. Improved on-chip router analytical power and area modeling. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 241–246, Jan 2010.

[135] A. B. Kahng, M. Luo, and S. Nath. Si for free: machine learning of interconnect coupling delay and transition effects. In *2015 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, pages 1–8, 2015.

[136] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, et al. Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 29–42. IEEE, 2018.

[137] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, pages 29–42, Piscataway, NJ, USA, 2018. IEEE Press.

[138] Sagar Karandikar, Albert Ou, Alon Amid, Howard Mao, Randy Katz, Borivoje Nikolić, and Krste Asanović. Fireperf: Fpga-accelerated full-system hardware/software performance profiling and co-design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 715–731, New York, NY, USA, 2020. Association for Computing Machinery.

[139] Sagar Karandikar, Albert Ou, Alon Amid, Howard Mao, Randy Katz, Borivoje Nikolić, and Krste Asanović. Fireperf: Fpga-accelerated full-system hardware/software performance profiling and co-design. In *Proceedings of the Twenty-Fifth International Conference on Architec-*

tural Support for Programming Languages and Operating Systems, pages 715–731, 2020.

[140] Moein Khazraee. *Reducing the development cost of customized cloud infrastructure*. PhD thesis, University of California, San Diego, 2020.

[141] Moein Khazraee, Luis Vega Gutierrez, Ikuo Magaki, and Michael Bedford Taylor. Specializing a planet's computation: Asic clouds. *IEEE Micro*, 37(3):62–69, 2017.

[142] Moein Khazraee, Lu Zhang, Luis Vega, and Michael Taylor. Moonwalk: NRE Optimization in ASIC Clouds or, accelerators will use old silicon. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[143] Donggyu Kim, Christopher Celio, Sagar Karandikar, David Biancolin, Jonathan Bachrach, and Krste Asanović. Dessert: Debugging rtl effectively with state snapshotting for error replays across trillions of cycles. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 76–764. IEEE, 2018.

[144] Jason Kim, Michael B. Taylor, Jason Miller, and David Wentzlaff. Energy Characterization of a Tiled Architecture Processor with On-Chip Networks. In *International Symposium on Low Power Electronics and Design (ISLPED)*, August 2003.

[145] AJ KleinOsowski, John Flynn, Nancy Meares, and David J Lilja. Adapting the spec 2000 benchmark suite for simulation-based computer architecture research. *Workload characterization of emerging computer applications*, pages 83–100, 2001.

[146] Sravanthi Kota Venkata, IkkJin Ahn, Donghwan Jeon, Anshuman Gupta, and Michael Taylor. SD-VBS: The San Diego Vision Benchmark Suite. In *IEEE International Symposium on Workload Characterization (IISWC)*, 2009.

[147] Milos Krstic, Eckhard Grass, Frank K Gürkaynak, and Pascal Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design & Test of computers*, 24(5):430–441, 2007.

[148] Jeffrey Kuskin, David Ofelt, Mark A. Heinrich, John Heinlein, Richard Simoni, Kourosh Gharachorloo, John Chapin, David Nakahira, Joel Baxter, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, and John L. Hennessy. The stanford FLASH multiprocessor. In David A. Patterson, editor, *Proceedings of the 21st Annual International Symposium on*

Computer Architecture. Chicago, IL, USA, April 1994, pages 302–313. IEEE Computer Society, 1994.

[149] Jeffrey S. Kuskin. *The FLASH Multiprocessor: Designing a Flexible and Scalable System*. PhD thesis, Stanford University, Stanford, CA, USA, 1997.

[150] Kevin Laeufer, Vighnesh Iyer, David Biancolin, Jonathan Bachrach, Borivoje Nikolić, and Koushik Sen. Simulator independent coverage for rtl hardware languages. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 606–615, New York, NY, USA, 2023. Association for Computing Machinery.

[151] Kevin Laeufer, Jack Koenig, Donggyu Kim, Jonathan Bachrach, and Koushik Sen. Rfuzz: Coverage-directed fuzz testing of rtl on fpgas. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2018.

[152] J. Lee, J. Kim, J. Kim, S. Seo, and J. Lee. An opencl framework for homogeneous manycores with no hardware cache coherence. In *2011 International Conference on Parallel Architectures and Compilation Techniques*, pages 56–67, 10 2011.

[153] Jaejin Lee, Sangmin Seo, Chihun Kim, Junghyun Kim, Posung Chun, Zehra Sura, Jungwon Kim, and Sangyong Han. COMIC: a coherent shared memory interface for cell be. In Andreas Moshovos, David Tarditi, and Kunle Olukotun, editors, *17th International Conference on Parallel Architectures and Compilation Techniques, PACT 2008, Toronto, Ontario, Canada, October 25-29, 2008*, pages 303–314. ACM, 2008.

[154] Kangli Li. An Open Source Non-Blocking Manycore L2 Cache. Master's thesis, University of Washington, 2024.

[155] M. Lis, Keun Sup Shim, B. Cho, I. Lebedev, and S. Devadas. Hardware-level thread migration in a 110-core shared-memory multiprocessor. In *2013 IEEE Hot Chips 25 Symposium (HCS)*, pages 1–27, Aug 2013.

[156] Derek Lockhart, Gary Zibrat, and Christopher Batten. Pymtl: A unified framework for vertically integrated computer architecture research. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 280–292. IEEE, 2014.

[157] Ikuo Magaki, Moein Khazraee, Luis Vega, and Michael Taylor. ASIC

Clouds: Specializing the Datacenter. In *International Symposium on Computer Architecture (ISCA)*, 2016.

[158] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G Cota, Michele Petracca, Christian Pilato, and Luca P Carloni. Agile soc development with open esp. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[159] M. McKeown, A. Lavrov, M. Shahrad, P. J. Jackson, Y. Fu, J. Balkind, T. M. Nguyen, K. Lim, Y. Zhou, and D. Wentzlaff. Power and energy characterization of an open source 25-core manycore processor. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 762–775, Feb 2018.

[160] Mentor. https://eda.sw.siemens.com/en-us/ic/veloce/, 2023.

[161] Maged M. Michael, Ashwini K. Nanda, and Beng-Hong Lim. Coherence controller architectures for scalable shared-memory multiprocessors. *IEEE Trans. Computers*, 48(2):245–255, 1999.

[162] Maged M. Michael, Ashwini K. Nanda, Beng-Hong Lim, and Michael L. Scott. Coherence controller architectures for smp-based CC-NUMA multiprocessors. In Andrew R. Pleszkun and Trevor N. Mudge, editors, *Proceedings of the 24th International Symposium on Computer Architecture, Denver, Colorado, USA, June 2-4, 1997*, pages 219–228. ACM, 1997.

[163] Microsoft. `https://docs.microsoft.com/en-us/azure/virtual-machines/np-series`, 2023.

[164] F. Mirasol. The principle of scaling down. *BioPharm International 32*, 2019.

[165] Sergiu Mosanu, Joshua Fixelle, Mohammad Nazmus Sakib, Kevin Skadron, and Mircea Stan. Freezetime: Towards system emulation through architectural virtualization. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 129–136, Piscataway, NJ, USA, 2023. IEEE, IEEE.

[166] Sripathi Muralitharan. TinyParrot: An Integration-Optimized Linux-Capable Host Multicore. Master's thesis, University of Washington, 2021.

[167] Vijay Nagarajan, Daniel J. Sorin, Mark D. Hill, and David A. Wood.

A primer on memory consistency and cache coherence, second edition. *Synthesis Lectures on Computer Architecture*, 15(1):1–294, 2020.

[168] Rishiyur Nikhil. Bluespec system verilog: efficient, correct rtl from high level specifications. In *Proceedings. Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE'04.*, pages 69–70. IEEE, 2004.

[169] A. Nowatzyk, G. Aybay, M. Browne, E. Kelly, D. Lee, and M. Parkin. The s3.mp scalable shared memory multiprocessor. In *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences HICSS-94*. IEEE Comput. Soc. Press, 1994.

[170] University of Washington. UW openroad free45 pdk reference flow. `https://github.com/bsg-idea/uw_openroad_free45/`.

[171] Opensbi.

[172] Oracle. OpenSPARC T1.

[173] Oracle. OpenSPARC T1 microarchitectural specification, April 2008.

[174] Sreepathi Pai, R. Govindarajan, and Matthew J. Thazhuthaveetil. Fast and efficient automatic memory management for GPUs using compiler-assisted runtime coherence scheme. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques - PACT '12*, page 33, Minneapolis, Minnesota, USA, 2012. ACM Press.

[175] S. Pal, D. Park, S. Feng, P. Gao, J. Tan, A. Rovinski, S. Xie, C. Zhao, A. Amarnath, T. Wesley, J. Beaumont, K. Chen, C. Chakrabarti, M. Taylor, T. Mudge, D. Blaauw, H. Kim, and R. Dreslinski. A 7.3 M Output Non-Zeros/J Sparse Matrix-Matrix Multiplication Accelerator using Memory Reconfiguration in 40 nm. In *Symposium on VLSI Circuits*, pages C150–C151, 2019.

[176] James Pallister, Simon Hollis, and Jeremy Bennett. Beebs: Open benchmarks for energy measurements on embedded platforms. *arXiv preprint arXiv:1308.5174*, 2013.

[177] Mark S. Papamarcos and Janak H. Patel. A low-overhead coherence solution for multiprocessors with private cache memories. *ACM SIGARCH Computer Architecture News*, 12(3):348–354, 6 1984.

[178] D. Park, S. Pal, S. Feng, P. Gao, J. Tan, A. Rovinski, S. Xie, C. Zhao, A. Amarnath, T. Wesley, J. Beaumont, K. Chen, C. Chakrabarti, M. B. Taylor, T. Mudge, D. Blaauw, H. Kim, and R. G. Dreslinski. A 7.3 M

Output Non-Zeros/J, 11.7 M Output Non-Zeros/GB Reconfigurable Sparse Matrix-Matrix Multiplication Accelerator. *IEEE Journal of Solid-State Circuits*, pages 933–944, April 2020.

[179] Huwan Peng. *Methodologies and Architectures for AI Inference Hardware: From Foundational Networks to Large Language Models*. PhD thesis, University of Washington, 2025.

[180] Huwan Peng, Scott Davidson, Richard Shi, Shuaiwen Leon Song, and Michael Taylor. Chiplet Cloud: Building AI Supercomputers for Serving Large Generative Language Models. *arXiv:2307.02666 [cs]*, 2024.

[181] Huwan Peng, Scott Davidson, Richard Shi, and Michael Taylor. ReaLLM: A Trace-Driven Framework for Rapid Simulation of Large-Scale LLM Inference. In *ASAP*, 2025.

[182] Giuseppe Della Penna, Benedetto Intrigila, Igor Melatti, Enrico Tronci, and Marisa Venturini Zilli. Exploiting transition locality in automatic verification of finite-state concurrent systems. *International Journal on Software Tools for Technology Transfer*, 6(4):320–341, July 2004.

[183] D. Petrisko, F. Gilani, M. Wyse, D. C. Jung, S. Davidson, P. Gao, C. Zhao, Z. Azad, S. Canakci, B. Veluri, T. Guarino, A. Joshi, M. Oskin, and M. B. Taylor. BlackParrot: An Agile Open-Source RISC-V Multicore for Accelerator SoCs. *IEEE Micro*, pages 93–102, Jul/Aug. 2020.

[184] Daniel Petrisko. `https://www.youtube.com/watch?v=r62g-wwBLkI&list=PLUg3wIOWD8ypF1GQqFl5oT1UEDBJS9LxR&index=38&t=83s`, 2023.

[185] Daniel Petrisko, Farzam Gilani, Mark Wyse, Dai Cheol Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, Bandhav Veluri, et al. Blackparrot: An agile open-source risc-v multicore for accelerator socs. *IEEE Micro*, 40(4):93–102, 2020.

[186] Daniel Petrisko, Farzam Gilani, Mark Wyse, Dai Cheol Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, Bandhav Veluri, Tavio Guarino, Ajay Joshi, Mark Oskin, and Michael Bedford Taylor. Blackparrot: An agile open-source RISC-V multicore for accelerator socs. *IEEE Micro*, 40(4):93–102, 2020.

[187] Daniel Petrisko, Chun Zhao, Scott Davidson, Paul Gao, Dustin Richmond, and Michael Bedford Taylor. NoC Symbiosis. In *NOCS*, 2020.

[188] Daniel Petrisko, Chun Zhao, Scott Davidson, Paul Gao, Dustin Rich-

mond, and Michael Bedford Taylor. Noc symbiosis (special session paper). In *2020 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8. IEEE, 2020.

[189] A. Pinto, L. P. Carloni, and A. Sangiovanni-Vincentelli. Cosi: A framework for the design of interconnection networks. *IEEE Design Test of Computers*, 25(5):402–415, Sep. 2008.

[190] John Probell. Routing congestion: The growing cost of wires in systems-on-chip. Technical report, Technical report, Arteris, 2011.

[191] Robert "Max" Ramstad. Enabling Vector Load and Store instructions on HammerBlade Architecture. Master's thesis, University of Washington, 2024.

[192] Shashank Vijaya Ranga. ParrotPiton and ZynqParrot: FPGA Enablements for the BlackParrot RISC-V Processor. Master's thesis, University of Washington, 2021.

[193] Antti Rautakoura and Timo Hämäläinen. Does soc hardware development become agile by saying so: A literature review and mapping study. *ACM transactions on embedded computing systems*, 22(3):1–27, 2023.

[194] Steven K. Reinhardt, James R. Larus, and David A. Wood. Tempest and typhoon: User-level shared memory. In David A. Patterson, editor, *Proceedings of the 21st Annual International Symposium on Computer Architecture. Chicago, IL, USA, April 1994*, pages 325–336. IEEE Computer Society, 1994.

[195] The RISC-V Foundation. *The RISC-V Instruction Set Manual Volume 1: Unprivileged ISA*, 20190608-base-ratified edition, 6 2019.

[196] A. Rovinski, C. Zhao, K. Al-Hawaj, P. Gao, S. Xie, C. Torng, S. Davidson, A. Amarnath, L. Vega, B. Veluri, A. Rao, T. Ajayi, J. Puscar, S. Dai, R. Zhao, D. Richmond, Z. Zhang, I. Galton, C. Batten, M. B. Taylor, and R. G. Dreslinski. A 1.4 GHz 695 Giga Risc-V Inst/s 496-Core Manycore Processor With Mesh On-Chip Network and an All-Digital Synthesized PLL in 16nm CMOS. In *2019 Symposium on VLSI Circuits*, pages C30–C31, 2019.

[197] A. Rovinski, C. Zhao, K. Al-Hawaj, P. Gao, S. Xie, C. Torng, S. Davidson, A. Amarnath, L. Vega, B. Veluri, A. Rao, T. Ajayi, J. Puscar, S. Dai, R. Zhao, D. Richmond, Z. Zhang, I. Galton, C. Batten, M. B. Taylor, and R. G. Dreslinski. Evaluating Celerity: A 16-nm 695 Giga-RISC-V Instructions/s Manycore Processor With Synthesizable PLL. *IEEE Solid-State Circuits Letters*, 2(12):289–292, 2019.

[198] A. Rovinski, C. Zhao, K. Al-Hawaj, P. Gao, S. Xie, C. Torng, S. Davidson, A. Amarnath, L. Vega, B. Veluri, A. Rao, T. Ajayi, J. Puscar, S. Dai, R. Zhao, D. Richmond, Z. Zhang, I. Galton, C. Batten, M. B. Taylor, and R. G. Dreslinski. Evaluating celerity: A 16-nm 695 giga-risc-v instructions/s manycore processor with synthesizable pll. *IEEE Solid-State Circuits Letters*, 2(12):289–292, 2019.

[199] Daniel Ruelas-Petrisko. *A Qualitative Approach to Agile Hardware Design*. PhD thesis, University of Washington, 2025.

[200] M. Ruttenberg and M. Taylor. The hammerblade risc-v manycore, 2020.

[201] Christos Sakalis, Carl Leonardsson, Stefanos Kaxiras, and Alberto Ros. Splash-3: A properly synchronized benchmark suite for contemporary research. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2016, Uppsala, Sweden, April 17-19, 2016*, pages 101–111. IEEE Computer Society, 2016.

[202] Jack Sampson, Manish Arora, Nathan Goulding-Hotta, Ganesh Venkatesh, Jonathan Babb, Vikram Bhatt, Michael Bedford Taylor, and Steven Swanson. An Evaluation of Selective Depipelining for FPGA-based Energy-Reducing Irregular Code Coprocessors. In *Conference on Field Programmable Logic and Applications (FPL)*, 2011.

[203] Jack Sampson, Ganesh Venkatesh, Nathan Goulding-Hotta, Saturnino Garcia, Steven Swanson, and Michael Bedford Taylor. Efficient Complex Operators for Irregular Codes. In *High Performance Computing Architecture (HPCA)*, 2011.

[204] S. Satpathy, Z. Foo, B. Giridhar, R. Dreslinski, D. Sylvester, T. Mudge, and D. Blaauw. A 1.07 tbit/s 128 $\times$ 128 swizzle network for simd processors. In *Symposium on VLSI Circuits*, pages 81–82, June 2010.

[205] John Y Sayah, Rajesh Gupta, Deepak D Sherlekar, Philip S Honsinger, Jitendra M Apte, S Wayne Bollinger, Hai Hsia Chen, Sumit DasGupta, Edward P Hsieh, Andrew D Huber, et al. Design planning for high-performance asics. *IBM Journal of Research and Development*, 40(4):431–452, 1996.

[206] Pasquale Davide Schiavone, Davide Rossi, Antonio Pullini, Alfio Di Mauro, Francesco Conti, and Luca Benini. Quentin: an ultra-low-power pulpissimo soc in 22nm fdx. In *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pages 1–3, Burlingame, CA, USA, 2018. IEEE.

[207] Ioannis Schoinas, Babak Falsafi, Alvin R. Lebeck, Steven K. Reinhardt,

James R. Larus, and David A. Wood. Fine-grain access control for distributed shared memory. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VI, pages 297–306, New York, NY, USA, 1994. ACM.

[208] K. Sewell, R. G. Dreslinski, T. Manville, S. Satpathy, N. Pinckney, G. Blake, M. Cieslak, R. Das, T. F. Wenisch, D. Sylvester, D. Blaauw, and T. Mudge. Swizzle-switch networks for many-core systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(2):278–294, June 2012.

[209] K. S. Shim, M. Lis, M. H. Cho, I. Lebedev, and S. Devadas. Design tradeoffs for simplicity and efficient verification in the execution migration machine. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 145–153, 2013.

[210] Wilson Snyder. Verilator: Fast, free, but for me? *DVClub Presentation*, page 11, 2010.

[211] Wilson Snyder. https://github.com/verilator/verilator, 2024.

[212] Daniel J. Sorin, Manoj Plakal, Anne Condon, Mark D. Hill, Milo M. K. Martin, and David A. Wood. Specifying and verifying a broadcast and a multicast snooping cache coherence protocol. *IEEE Trans. Parallel Distributed Syst.*, 13(6):556–578, 2002.

[213] Steven Swanson and Michael Taylor. GreenDroid: Exploring the next evolution for smartphone application processors. In *IEEE Communications Magazine*, March 2011.

[214] Paul Sweazey and Alan Jay Smith. A class of compatible cache consistency protocols and their support by the IEEE futurebus. *ACM SIGARCH Computer Architecture News*, 14(2):414–423, 6 1986.

[215] Synopsys. https://www.synopsys.com/verification/emulation/zebu-server.html, 2023.

[216] C. Tan, Y. Ou, S. Jiang, P. Pan, C. Torng, S. Agwa, and C. Batten. Pyocn: A unified framework for modeling, testing, and evaluating on-chip networks. In *International Conference on Computer Design (ICCD)*, pages 437–445, 2019.

[217] Zhangxi Tan, Zhenghao Qian, Xi Chen, Krste Asanovic, and David Patterson. Diablo: A warehouse-scale computer network simulator using fpgas. *ACM SIGPLAN Notices*, 50(4):207–221, 2015.

[218] Calvin K. Tang. Cache system design in the tightly coupled multiprocessor system. In *American Federation of Information Processing Societies: 1976 National Computer Conference, 7-10 June 1976, New York, NY, USA*, volume 45 of *AFIPS Conference Proceedings*, pages 749–753. AFIPS Press, 1976.

[219] M. B. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, and J. Kim. Evaluation of the raw microprocessor: an exposed-wire-delay architecture for ilp and streams. In *International Symposium on Computer Architecture, 2004.*, pages 2–13, 2004.

[220] M Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar operand networks: On-chip interconnect for ilp in partitioned architectures. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, pages 341–353. IEEE, 2003.

[221] MB Taylor, J Kim, J Miller, F Ghodrat, B Greenwald, P Johnson, W Lee, A Ma, N Shnidman, V Strumpen, et al. The raw processor-a scalable 32-bit fabric for embedded and general purpose computing. In *Proceedings of Hot Chips XIII*, 2001.

[222] Michael Taylor. *Tiled Microprocessors*. PhD thesis, Massachusetts Institute of Technology, 2007.

[223] Michael Taylor. A Landscape of the New Dark Silicon Design Regime. *Micro, IEEE*, Sept-Oct. 2013.

[224] Michael Taylor. A Landscape of the New Dark Silicon Design Regime. In *Design Automation and Test in Europe*, April 2014.

[225] Michael Taylor. The Evolution of Bitcoin Hardware. *Computer, IEEE*, Sept-Oct. 2017.

[226] Michael B Taylor. Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the 49th annual design automation conference*, pages 1131–1136, 2012.

[227] Michael B. Taylor. Bitcoin and the Age of Bespoke Silicon. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, 2013.

[228] Michael B. Taylor. BaseJump STL: SystemVerilog needs a Standard

Template Library for Hardware Design. In *Design Automation Conference*, June 2018.

[229] Michael B. Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. In *IEEE Micro*, March 2002.

[230] Michael B. Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffmann, Paul Johnson, Walter Lee, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Saman Amarasinghe, and Anant Agarwal. A 16-issue Multiple-Program-Counter Microprocessor with Point-to-Point Scalar Operand Network. In *IEEE International Solid-State Circuits Conference (ISSCC)*, February 2003.

[231] Michael B. Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar Operand Networks. In *IEEE Transactions on Parallel and Distributed Systems*, February 2005.

[232] Michael B. Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar Operand Networks. In *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, February 2005.

[233] Michael B. Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *International Symposium on Computer Architecture (ISCA)*, June 2004.

[234] Michael Bedford Taylor. Geocomputers and the Commercial Borg. In *SIGARCH Computer Architecture Today*, Dec 2017.

[235] Michael Bedford Taylor. Basejump stl: Systemverilog needs a standard template library for hardware design. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, New York, NY, USA, 2018. ACM.

[236] Michael Bedford Taylor. Basejump STL: systemverilog needs a standard template library for hardware design. In *Proceedings of the 55th Annual*

Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018, pages 73:1–73:6. ACM, 2018.

[237] Michael Bedford Taylor. Your agile open source HW stinks (because it is not a system). In *ICCAD*, 2020.

[238] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, et al. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE micro*, 22(2):25–35, 2002.

[239] Michael Bedford Taylor, Luis Vega, Moein Khazraee, Ikuo Magaki, Scott Davidson, and Dustin Richmond. ASIC clouds: Specializing the datacenter for planet-scale applications. *CACM*, pages 103–109, 2020.

[240] Riccardo Tedeschi, Gianmarco Ottavi, Côme Allart, Nils Wistoff, Zexin Fu, Filippo Grillotti, Fabio De Ambroggi, Elio Guidetti, Jean-Baptiste Rigaud, Olivier Potin, et al. Cva6s+: A superscalar risc-v core with high-throughput memory architecture. *arXiv preprint arXiv:2505.03762*, 1(1):1–3, 2025.

[241] Shelby Thomas, Chetan Gohkale, Enrico Tanuwidjaja, Tony Chong, David Lau, Saturnino Garcia, and Michael Bedford Taylor. CortexSuite: A Synthetic Brain Benchmark Suite. In *International Symposium on Workload Characterization (IISWC)*, Oct. 2014.

[242] Rajat Todi. Speclite: using representative samples to reduce spec cpu2000 workload. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538)*, pages 15–23. IEEE, 2001.

[243] Linus Torvalds. Linux.

[244] TUL. https://www.tulembedded.com/fpga/productspynq-z2.html, 2023.

[245] Luis Vega and Michael Bedford Taylor. RV-IOV: Tethering RISC-V Processors via Scalable I/O Virtualization . In *CARRV*, 2017.

[246] Bandhav Veluri, Collin Pernu, Ali Saffari, Joshua Smith, Michael Taylor, and Shyam Gollakota. Neuricam: Low-power video acquisition using dual-mode iot cameras. In *MobiCom*, 2023.

[247] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and

Michael Bedford Taylor. Conservation cores: reducing the energy of mature computations. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.

[248] Ganesh Venkatesh, John Sampson, Nathan Goulding, Sravanthi Kota Venkata, Michael Bedford Taylor, and Steven Swanson. QsCores: Configurable Co-processors to Trade Dark Silicon for Energy Efficiency in a Scalable Manner. In *International Symposium on Microarchitecture (MICRO)*, 2011.

[249] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, et al. Baring it all to software: Raw machines. *Computer*, 30(9):86–93, 1997.

[250] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, et al. Baring it all to software: Raw machines. *Computer*, 30(9):86–93, 1997.

[251] Edward Wang. Hammer: A platform for agile physical design. *Master's thesis, EECS Dept, UC Berkeley*, 2018.

[252] Waterman et al. The risc-v instruction set manual. volume 1: User-level isa, version 2.0. Technical report, University of California, Berkeley, 2014.

[253] Wolf-Dietrich Weber. *Scalable directories for cache-coherent shared-memory multiprocessors*. PhD thesis, Stanford University, 1993.

[254] Thomas F Wenisch, Roland E Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C Hoe. Simflex: statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, 2006.

[255] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, Sep. 2007.

[256] Joonho Whangbo, Edwin Lim, Chengyi Lux Zhang, Kevin Anderson, Abraham Gonzalez, Raghav Gupta, Nivedha Krishnakumar, Sagar Karandikar, Borivoje Nikolić, Yakun Sophia Shao, et al. Fireaxe: Partitioned fpga-accelerated simulation of large-scale rtl designs. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 501–515, Buenos Aires, Argentina, 2024. IEEE, IEEEE.

[257] Wikichip. 14 nm lithography process. https://en.wikichip.org/wiki/14_nm_lithography_process.

[258] Claire Wolf. https://github.com/yosyshq/picorv32, 2025.

[259] Henry Ting-Hei Wong. *A superscalar out-of-order x86 soft processor for fpga*. University of Toronto (Canada), 2017.

[260] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In David A. Patterson, editor, *Proceedings of the 22nd Annual International Symposium on Computer Architecture, ISCA '95, Santa Margherita Ligure, Italy, June 22-24, 1995*, pages 24–36. ACM, 1995.

[261] Mark Wyse. *The BedRock Cache Coherence Protocol and System v1.1*, 2022.

[262] Mark Wyse, Daniel Petrisko, Farzam Gilani, Yuan-Mao Chueh, Paul Gao, Dai Cheol Jung, Sripathi Muralitharan, Shashank Vijaya Ranga, Mark Oskin, and Michael Taylor. The blackparrot bedrock cache coherence system. *arXiv preprint arXiv:2211.06390*, 1:1–20, 2022.

[263] Chenhao Xie, Xie Li, Yang Hu, Huwan Peng, Michael Taylor, and Shuaiwen Leon Song. Q-VR: System-level design for future mobile collaborative virtual reality. In *ASPLOS*, 2021.

[264] Shaolin Xie, Scott Davidson, Ikuo Magaki, Moein Khazraee, Luis Vega, Lu Zhang, and Michael B. Taylor. Extreme datacenter specialization for planet-scale computing: Asic clouds. In *ACM Sigops Operating System Review*, 2018.

[265] Xilinx. *DS890 UltraScale Architecture and Product Data Sheet: Overview, v4.1.1*, 2022.

[266] Xilinx. Device reliability report (ug116). Technical report, Xilinx, 2023.

[267] Xilinx. https://docs.xilinx.com/r/en-us/pg195-pcie-dma, 2023.

[268] Xilinx. https://www.xilinx.com/products/boards-and-kits/device-family/nav-zynq-7000.html, 2023.

[269] Xilinx. http://www.pynq.io/board.html, 2023.

[270] Xilinx. https://docs.xilinx.com/r/en-US/ug1144-petalinux-tools-reference-guide/Introduction, 2023.

[271] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer, 2003.

[272] F. Zaruba and L. Benini. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11):2629–2640, Nov 2019.

[273] Florian Zaruba and Luca Benini. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11):2629–2640, 2019.

[274] Karen Zee, Viktor Kuncak, Michael Taylor, and Martin C. Rinard. Runtime checking for program verification. In *RV*, 2007.

[275] Xingyao Zhang, Haojun Xia, Donglin Zhuang, Hao Sun, Xin Fu, Michael Taylor, and Shuaiwen Leon Song. η-LSTM: Co-designing highly-efficient large lstm training via exploiting memory-saving and architectural design opportunities. In *ISCA*, 2021.

[276] Jerry Zhao, Animesh Agrawal, Borivoje Nikolic, and Krste Asanović. Constellation: an open-source soc-capable noc generator. In *2022 15th IEEE/ACM International Workshop on Network on Chip Architectures (NoCArc)*, pages 1–7, Chicago, IL, USA, 2022. IEEE, IEEE.

[277] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. Sonicboom: The 3rd generation berkeley out-of-order machine. In *Fourth Workshop on Computer Architecture Research with RISC-V*, volume 5, pages 1–7. International Symposium on Computer Architecture Valencia, Spain, 2020.

[278] Ritchie Zhao, Chun Zhao, Shaolin Xie, Bandhav Veluri, Luis Vega, Christopher Torng, Ningxiao Sun, Austin Rovinski, Anuj Rao, Gai Liu, Paul Gao, Scott Davidson, Steve Dai, Aporva Amarnath, KhalidAl-Hawaj, Tutu Ajayi Christopher Batten, Ronald G. Dreslinski, Rajesh K.Gupta, Michael B.Taylor, and Zhiru Zhang. Celerity: An Open Source RISC-V Tiered Accelerator Fabric. In *7th RISC-V Workshop*, 2017.

[279] Qiaoshi Zheng, Nathan Goulding-Hotta, Scott Ricketts, Steven Swanson, Michael Bedford Taylor, and Jack Sampson. Exploring energy scalability in coprocessor-dominated architectures for dark silicon. *Transactions on Embedded Computing Systems (TECS)*, Mar 2014.

[280] Yi Zhu, Yuanfang Hu, Michael Taylor, and Chung-Kuan Cheng. Energy and switch area optimizations for FPGA global routing architectures. In *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, January 2009.

[281] Yi Zhu, Michael Taylor, Scott B. Baden, and Chung-Kuan Cheng. Advancing supercomputer performance through interconnection topology synthesis. In *International Conference on Computer-Aided Design (ICCAD)*, pages 555–558, 2008.

Appendix A

OPEN-SOURCE LINK FARM

Below is a list of open-source projects referenced in this thesis, organized by category.

BlackParrot

- BlackParrot:

  `https://github.com/black-parrot/black-parrot`

- BlackParrot Tools:

  `https://github.com/black-parrot/black-parrot-tools`

- BlackParrot SDK:

  `https://github.com/black-parrot-sdk/black-parrot-sdk`

- BlackParrot Simulation Environment:

  `https://github.com/black-parrot/black-parrot-sim`

- BlackParrot Subsystems:

  `https://github.com/black-parrot-hdk/black-parrot-subsystems`

- ZynqParrot:

  `https://github.com/black-parrot-hdk/zynq-parrot`

- libperch:

  `https://github.com/black-parrot-sdk/libperch`

- BP Tests:

  `https://github.com/black-parrot-sdk/bp-tests`

- BP Demos:

  `https://github.com/black-parrot-sdk/bp-demos`

- For Reference Only

  - BlackParrot v0 tapeout:

    `https://github.com/black-parrot-examples/bsg_tapeout_v0_gf12`

  - BlackParrot v1 tapeout:

    `https://github.com/black-parrot-examples/bsg_tapeout_v1_tsmc40`

  - BlackParrot Tapeins:

    `https://github.com/black-parrot-examples/bsg_scratch_designs`

  - BlackParrot FPGA bringup:

    `https://github.com/black-parrot-examples/bsg_fpga`

Bespoke Silicon Group

- BSG Pearls:

  `https://github.com/bespoke-silicon-group/bsg_pearls`

- BSG Replicant:

  `https://github.com/bespoke-silicon-group/bsg_replicant`

- BSG Manycore:

  `https://github.com/bespoke-silicon-group/bsg_manycore`

- BaseJump STL:

  `https://github.com/bespoke-silicon-group/basejump_stl`

- BSG BladeRunner:

`https://github.com/bespoke-silicon-group/bsg_bladerunner`

- libgloss-dramfs (PanicRoom):

  `https://github.com/black-parrot-sdk/libgloss-dramfs`

- HardFloat (fork of Berkeley Verilog HardFloat):

  `https://github.com/bsg-external/HardFloat`

Miscellaneous Collaborations

- OpenPiton:

  `https://github.com/PrincetonUniversity/openpiton`

- LiteX:

  `https://github.com/enjoy-digital/litex`

- CVA6 (Ariane):

  `https://github.com/openhwgroup/cva6`

- Rocket:

  `https://github.com/chipsalliance/rocket-chip`

- Dromajo:

  `https://github.com/ChipsAlliance/dromajo`

- Surelog:

  `https://github.com/ChipsAlliance/Surelog`

- UHDM:

  `https://github.com/ChipsAlliance/UHDM`

Excellent Tools Used By the Author

- ZachJS SV2V:

  `https://github.com/zachjs/sv2v`

- BSG SV2V:

  `https://github.com/bespoke-silicon-group/bsg_sv2v`

- BSG FakeRam:

  `https://github.com/bespoke-silicon-group/bsg_fakeram`

- Verilator:

  `https://github.com/verilator/verilator`

- Surfer:

  `https://gitlab.com/surfer-project/surfer`

- PULP AXI:

  `https://github.com/pulp-platform/axi`