POSE: Phase II: An Open-Source Compiler Ecosystem for Pervasive Hardware Acceleration

1 Overview

Hardware accelerators are the best tools we have to compensate for the rapidly waning advances in general-purpose computing [26]. Specialized hardware is routinely two orders of magnitude more efficient than mainstream, general-purpose hardware [25]. Even formerly "software companies" (e.g., Google [32, 47], Microsoft [17, 44], Meta [16], and Amazon [6]) all fabricate bespoke silicon optimized for their specific workloads. And via increasingly affordable field-programmable gate arrays (FPGAs), hardware accelerators are coming within reach of small companies, scientific researchers, and even hobbyists.

The accelerator-centric future of computing is already here, but it is not yet evenly distributed [19]. Even as the physical technology costs continue to fall, *designing* new accelerators remains extremely expensive. Today, hardware projects require specialized electrical engineering skills, large development teams, and costly commercial toolchains available only from a handful of large vendors. The end result is that the immense efficiency advantages of acceleration remain accessible to a handful of the world's largest technology companies—not to the long tail of smaller teams that make up the rest of the technology industry.

The key to widespread hardware specialization lies in lowering the cost of accelerator design. Open-source ecosystems have a critical role to play in supporting diverse programming languages and high-quality tools that power cheap, accessible, productive accelerator development. There is a strong analogy in software development: the software industry's productivity relies on a diverse ecosystem of interoperating languages and compilers that are predominantly based on open-source foundations [34]. Hardware development, in contrast, is stuck in the past: a small number of extremely low-level languages dominate, in part because of their ties to entrenched commercial toolchains. The next revolution in accessible accelerator design requires an open approach to proliferating novel, high-level languages.

The Calyx compiler infrastructure. Since 2019, our team has been building Calyx [1, 41], an MIT-licensed open-source compiler infrastructure for the construction of high-level *accelerator design languages* (ADLs) [48]. ADLs are distinct from hardware description languages (HDLs), the low-level digital design tools that dominate mainstream hardware design today. HDLs like Verilog, VHDL, Bluespec [42], and Chisel [9] work at the *register transfer level* (RTL): an extremely detailed abstraction that gives programmers direct control over individual clock cycles, explicit wiring topologies, and the allocation of physical resources. ADLs raise the level of abstraction, offering *algorithmic* abstractions that hide some of these concrete hardware details, and eventually compile into HDL descriptions as a target.

In the same way that high-level software languages build on top of C and assembly, ADLs offer an alternative to HDLs that trade off some control for vastly improved productivity. Conventionally, however, designing and implementing ADLs has been a Herculean task. As a result, few successful ADLs currently exist. The dominant examples today are proprietary *high-level synthesis* (HLS) compilers [7, 11, 29, 38] that attempt to translate legacy software languages—typically C or C++—into HDL designs. Despite years of commercial investment, even these tools are exceptionally buggy [27] and yield wildly unpredictable performance [40].

In our view, the monolithic commercial approach to ADLs is hopeless. The problem is that a single language or a single compiler cannot begin to cover the diversity of use cases for an entire industry of accelerator development. At the same time, the industry cannot sustain n siloed ADL compilers for n different use cases. The only practical approach is an open-source ecosystem, where many stakeholders build on and contribute to a common foundation. In the same way that

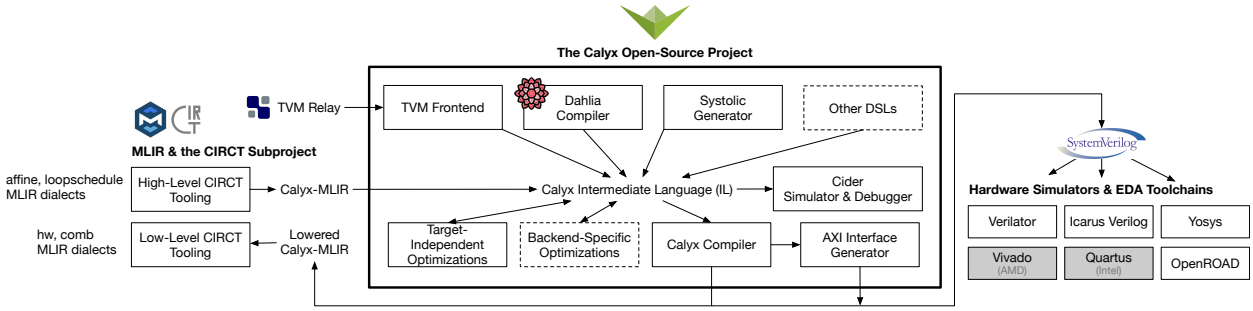

Figure 1: The Calyx open-source project and its interactions with other ecosystems. Shaded blocks denote proprietary tools. Dashed outlines mark extensions to the Calyx infrastructure proposed in this work.

the software industry has largely abandoned proprietary compilers in favor of building on GCC and LLVM [31], accelerator design needs an open-source basis for the next generation of ADLs.

Calyx aims to lower the barrier to entry for the creation of new ADLs and their compilers. Instead of focusing on a single input language, Calyx is a *compiler infrastructure* that encompasses an array of interacting components that developers can use to construct a toolchain for any ADL. There is an analogy between LLVM [34], a dominant open-source infrastructure for building software compilers, and Calyx's vision: through an open-source, language-neutral toolkit that different projects can mix and match for their own use cases, both projects aim to harness the collective effort of many different compiler projects.

Figure 1 depicts the Calyx open-source project. The centerpiece is Calyx's intermediate language (IL) [41], which represents hardware throughout the lifecycle of an accelerator design. An ensemble of tools all work with the IL: frontends for high-level languages such as the TVM machine learning compiler [12] generate the IL, a stack of optimizations transform the IL to improve its performance and efficiency, an interactive debugger executes the IL directly [10], and a compiler backend translates the IL into SystemVerilog to hand off to hardware simulators or EDA toolchains. Calyx is also a critical component of CIRCT [36], an industry-led initiative to collaborate on interoperable hardware compilers and tools.

A nascent open-source ecosystem. While other research-stage compiler infrastructures have been proposed [13, 22, 23, 49, 57], few of them have developed into full open-source projects with sustained investment. In contrast, Calyx has been continuously developed since its inception in 2019 and has supported an array of research and industrial users (see Section 2.4). As far as we are aware, Google's XLS project [20] is the only other currently supported, open-source ADL compiler toolchain. XLS has complementary goals to Calyx: its primary priority is accelerating the productivity of Google engineers on the company's silicon products. MLIR's CIRCT project [36] has some similar aims, but it relies on Calyx to generate hardware designs.

The Calyx open-source project already has 56 unique contributors, robust testing and continuous integration (CI) infrastructure, 39,000 words of technical documentation, and multiple ongoing external collaborations. The next step is for the framework to develop an open-source ecosystem that can harness the development investment of academic, industry, and government stakeholders.

Building a robust, multi-stakeholder ecosystem. Progress in the Calyx ecosystem so far has come exclusively as a *byproduct* of research efforts. Its potential is gated by the informal organization and governance that comes with an organically growing open-source project. This proposal is to establish a robust organization behind Calyx that will help it reach the momentum of a sustainable, multilateral ecosystem driven by investment from many stakeholders. Our vision is to realize the potential of a modular, open infrastructure: each user's investment in one piece of the ecosystem

feeds back into the larger community and bolsters all other use cases. Over time, Calyx will develop into a toolchain that will be unmatched by any single-vendor, proprietary approach.

Our plan for the Calyx OSE includes a formal, multi-stakeholder governance model; technical infrastructure that integrates with mainstream hardware toolchains; and community-building efforts to induct new users and contributors into the ecosystem. Together, this work will let Calyx support an era of *pervasive acceleration*, where all performance-intensive computing applications—not just the most popular and best funded—benefit from the unmatched efficiency advantages of domain-specific hardware.

2 Context of the Open-Source Ecosystem

This section summarizes Calyx's goals and its progress since its inception in 2019; Section 3 then describes our plans to build a full open-source ecosystem (OSE) around the project.

2.1 Principles, Vision, and Impact

We articulate a central vision statement for the proposed OSE around Calyx:

> *Realizing the potential of pervasive hardware acceleration requires a rapid expansion in the number of high-level programming languages that compile to hardware accelerators. Calyx aims to act as the catalyst for the proliferation of these languages.*

Our vision intentionally embraces an analogy to LLVM [34], an open-source infrastructure that has has accelerated innovation in software programming languages by lowering the cost to build new high-quality compilers. New software languages like Rust and Swift, by building on LLVM, have more quickly become serious challengers to entrenched languages like C and C++. And by remaining the "default choice" for new compiler implementation efforts, LLVM consolidates engineering effort across many different projects and avoids duplication of work. Calyx aims to bring the same pattern of rapid language innovation to accelerator design.

Programming languages, however, are not the end goal of this vision: *the ultimate goal is to bring custom hardware acceleration to a long tail of computing applications.* Today, specialized silicon is reserved for large, well-funded teams and only the highest-value applications. In a particularly stark example, industrial artificial intelligence (AI) hardware has focused on the most well-studied machine learning models, which helps further entrench their advantage over more experimental approaches to machine learning (ML) [28]. In a world with a lower barrier to entry for silicon design, smaller teams and earlier-stage ideas could compete on a more level playing field with conventional approaches.

While AI has monopolized the recent attention of custom hardware investment, many more categories of applications stand to benefit. Bespoke hardware to support climate modeling or genomic analysis, for instance, could bring qualitative advances in the capabilities of those fields *if we can empower domain experts to design it*. A conventional accelerator development model, with a strict separation between hardware engineers and application experts, is unlikely to yield pervasive, cost-effective hardware projects. We see domain-specific ADLs as the key to enabling application experts from any domain to become accelerator designers—and infrastructures like Calyx as the catalyst for those programming models.

2.2 A Comprehensive Intermediate Language for Accelerator Compilation

The Calyx project (Figure 1) encompasses a spectrum of tools that all intersect on a common intermediate language. The IL is Calyx's central differentiator from other compiler infrastructures: it provides a *single program representation* that bridges the gap between high-level semantics and

low-level control over the hardware structure. Consequently, all of Calyx's frontends, analyses, optimizations, and ancillary tools can work on the same representation with well-defined semantics: there are no "hidden" data structures that control the hardware's performance or behavior.

The Calyx IL was originally introduced in an ASPLOS 2021 paper about its design [41]. Calyx explicitly represents both the physical *structure* of the hardware and its logical *control*. The structural component resembles a traditional HDL, representing physical elements like functional units and memories. The control component resembles an imperative software language, describing how to orchestrate those structural resources over time to compute a function. Calyx's ADL frontends generate code that relies heavily on control to implement high-level semantics, such as those of the TVM deep learning compiler [12] or the Dahlia loop-based ADL [40]. A suite of lowering and optimization passes gradually converts these high-level constructs into structural elements, instantiating specialized control logic to implement them. Finally, after applying these passes, the Calyx IL reaches a form that is *fully structural* and isomorphic to a traditional HDL. The Calyx compiler backend is a simple translation to Verilog, which can then feed into a traditional closed- or open-source EDA toolchain to produce silicon or FPGA designs.

The Calyx open-source project has expanded beyond the original compiler stack from 2021. Calyx now includes Cider, an interpreter and debugger that can directly execute the Calyx IL *without first compiling it to Verilog* [10]. Cider forced the project to formalize and clarify the semantics of the IL, and it provides an indispensable debugging tool that lets programmers interactively step through accelerator executions—much like a traditional software debugger like GDB or LLDB. Calyx has also developed deep connections to MLIR's CIRCT subproject [36], a broader effort to develop new open-source hardware compilation tools. Calyx serves as an integral component of several hardware-generation flows within CIRCT by exchanging code bidirectionally with the MLIR ecosystem. Section 2.4 elaborates on the connections between Calyx and CIRCT as well as other components that have already built upon the Calyx infrastructure.

2.3 Current Open-Source Organization

The Calyx project has so far been managed informally, controlled by a small group of researchers at Cornell University who share equal rights to contribute and manage the project. Despite this informal structure, we have already set up our technical management of the project to prepare it for sustainable development. Calyx follows several best practices for open-source work:

- Calyx has used the permissive MIT license since its inception.
- Calyx is a *mono-repository* (monorepo) that includes many disparate components under a single source tree: the core compiler, the IL's standard library, documentation and tutorials, and a separate Python library for generating Calyx programs. The monorepo organization makes it easy to implement coordinated changes between these disparate components and to catch any unintended breakage through continuous testing.
- Calyx has approximately 500 tests across all the components in the monorepo. The development model relies heavily on snapshot-based regression testing, where every bug fix or feature comes with test inputs and their expected outputs.
- The entire test suite runs on every commit via continuous integration (CI). Our development model enforces an "always green on main" policy, where all new work happens in branches, and we can only merge a branch once the complete test suite passes. All contributions, even from the core team, go through GitHub pull requests to enable code review and discussion.

- Calyx uses continuous delivery (CD) for documentation and Docker container images. Every new commit automatically updates the publicly visible documentation and produces new Docker images for Intel and ARM architectures for easy setup..
- We regularly tag versions of the Calyx library for external consumption. Releases follow the semantic versioning [43] scheme to convey the severity of changes to dependent projects.
- All communication about the Calyx project happens *in the open*, via GitHub issues and discussion threads. The Cornell team is careful to keep private communication channels to a minimum so every decision is visible to the community and archived for future reference.

Calyx's infrastructure is written in 56,000 lines of Rust, 18,000 lines of Python, and several thousand lines of tests and other infrastructure. To date, it has had 56 unique contributors, 2,200 issues and pull requests with cumulatively hundreds of thousands of words of discussion. The project has 39,000 words of documentation, spanning getting-started tutorials, a detailed language reference, and users' guides for each tool in the ecosystem.

2.4 Collaboration and Adoption

Calyx has already been the focal point of several industrial and academic collaborations. Instead of an exhaustive list, this section enumerates a few examples that highlight the critical role that Calyx already plays in high-level accelerator design.

The CIRCT project. CIRCT [36] is a recent multilateral open-source effort that includes contributors from SiFive, AMD, Microsoft, and academia to build new hardware tooling on the MLIR [35] framework. CIRCT encompasses several new MLIR dialects for low-level hardware description and for high-level synthesis flows. Our team contributed an MLIR dialect that corresponds to Calyx IL and tooling to convert between the MLIR dialect and Calyx's "native" representation (see Figure 1). Since then, the MLIR project has embraced Calyx as a critical component of its HLS flows: several initiatives rely on translating other MLIR dialects into Calyx, passing this representation through the Calyx compiler, and then returning to MLIR for low-level Verilog generation. This proposal includes letters of collaboration from two experts in the CIRCT ecosystem, Stephen Neuendorffer (AMD) and Julian Opperman (Codeplay), that provide more details on this symbiotic relationship.

Halide-to-Calyx compiler. A 2023 master's thesis from the Universitat Politècnica de Catalunya [21] built a new compiler from the Halide image-processing DSL [46] to hardware accelerators using Calyx. The thesis demonstrates how realizing high-level semantics in hardware would be prohibitively challenging without such an infrastructure and details how to use Calyx to produce accelerator implementations that are ready to use from software. See the included letter of collaboration from this thesis's author, Sergi Granell Escalfet, for more commentary on the role Calyx played in this work.

The Spade HDL. The Calyx team has an ongoing collaboration with the primary developer of the Spade HDL [50], a modern alternative to traditional hardware description languages that brings type safety and new features for composability. See the included letter from Frans Skarman, the project's lead developer, on his plans for integrating Spade and Calyx to combine the advantages of low-level and high-level hardware design.

Google XLS. The Calyx team also interacts regularly with the Google team developing XLS [20], another open-source ADL compiler that aims to accelerate Google's own chip development work. The complementary goals of Calyx and XLS mean that, while the concrete directions remain distinct, the two projects can exchange ideas and techniques that are relevant to any open-source

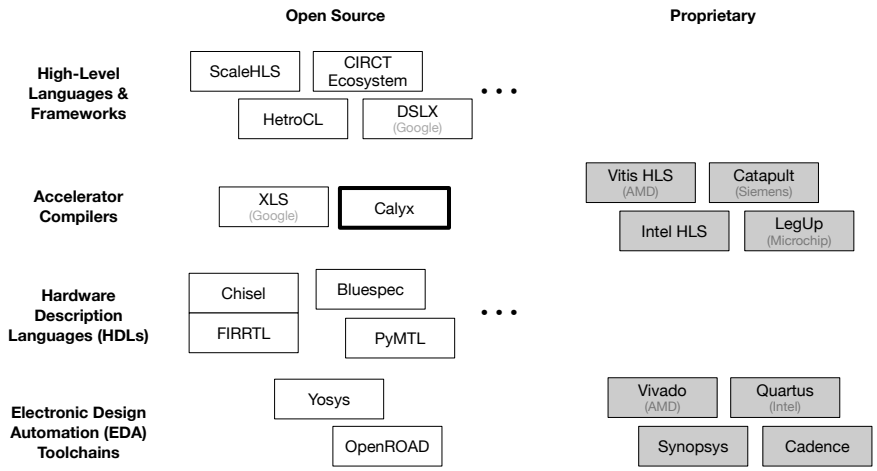

Figure 2: Open-source and proprietary technologies for hardware design. This proposal focuses on compiling high-level languages to hardware—above the level of traditional hardware description languages and electronic design automation toolchains, where robust open-source ecosystems are already established.

accelerator generation toolchain. Chris Leary (Google) has provided a letter of collaboration expanding on this symbiotic relationship.

2.5 The Open-Source Hardware Design Landscape

Via high-profile projects like RISC-V [56], Chisel [9], and OpenROAD [5], open-source hardware has gained momentum over the last decade. These open-source toolchain alternatives, however, have centered on the lower layers in the hardware design stack: hardware description languages and electronic design automation tools. While these components form the foundation for open-source hardware, they occupy a fundamentally different space from Calyx: they support low-level digital design by trained hardware engineers. This proposal focuses on an ecosystem that builds on these lower layers to make high-level hardware generation accessible to all software engineers.

Figure 2 depicts the layers in the open-source hardware design stack and their proprietary counterparts. The lower levels of the stack are populated by robust ecosystems like Chisel, its FIRRTL compiler infrastructure, and the OpenROAD toolchain. There are even several open-source efforts to design high-level languages for accelerator design [33, 36, 58]. However, *these other languages rely on proprietary HLS compilers to generate their hardware*. This reliance on specific commercial compilers limits these tools' practicality and flexibility. Calyx aims to act as a fully open-source backend for these languages, delivering the adaptability that practical solutions require.

The only other actively maintained open-source ADL toolchain we are aware of is Google's XLS [20]. XLS focuses on supporting a specific language frontend, DSLX, that targets Google's silicon workflows. Calyx aims instead to build a generic, extensible foundation for a wide range of languages, tools, and optimizations for any hardware target, including reconfigurable hardware. The two projects' focus areas complement each other and form an important avenue for future collaboration (see Section 2.4).

2.6 Team and Qualifications

PI Adrian Sampson will lead the proposed ecosystem-building work. He has led the Calyx project since its inception, advising an expanding team of PhD students and undergraduate contributors who implemented the entire system of interoperating tools. PI Sampson brings a combination of perspectives from both computer architecture and compilers; he runs a lab at Cornell that trains similarly balanced researchers who routinely publish in not only ISCA, MICRO, and ASPLOS but

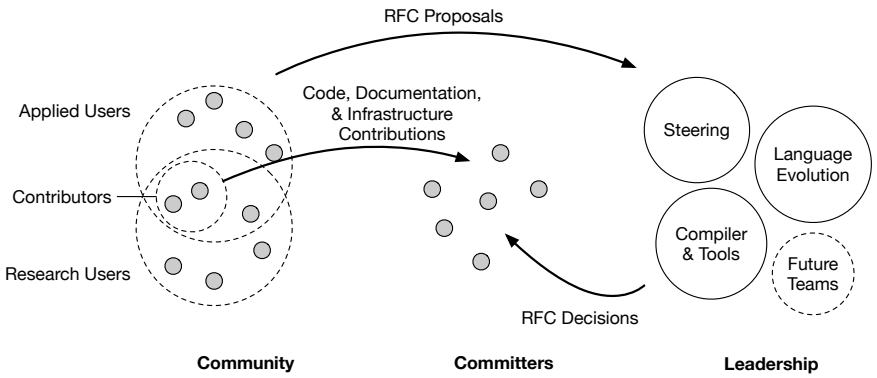

Figure 3: The proposed relationship between the Calyx project governance structure and its broader community of users. The key mechanism is a *request for comment (RFC) process*, where decisions are discussed and resolved in the open. The *leadership teams* are small groups who make final decisions on RFCs from the community. *Committers* implement RFCs by accepting contributions from the community.

also PLDI and OOPSLA. He won the 2021 IEEE TCCA Young Computer Architect Award in part for his contributions to generating hardware accelerators from high-level descriptions. The team will also include the PhD students in PI Sampson's lab who have acted as the lead developers of the Calyx ecosystem and, after spending hundreds of person-hours on maintaining and extending it, know the technical details best.

3 Proposed Open-Source Ecosystem

This section describes our plans to develop the Calyx open-source project into a robust, self-sustaining ecosystem. It covers three main thrusts: implementing a formal governance structure for the ecosystem (Section 3.1), building the auxiliary technical infrastructure to make Calyx the "default choice" for new acceleration projects (Section 3.2), and community building to expand the set of contributors to the ecosystem (Section 3.3).

3.1 Governing the Calyx Ecosystem

A main focus will be to establish a formal governance structure for Calyx. The project began as an informal *do-ocracy* [39], where the only barrier to contribution is the willingness to do the work. While it suffices for small projects, this informal model is not sustainable: it depends on individual people instead of long-lasting institutional structures that persist as contributors come and go.

We envision a sustainable governance system that embodies three core values:

1. *Accountability and legitimacy.* Technical decisions and the allocation of work must reflect the priorities of the Calyx community.

2. *Inclusiveness.* All identities and backgrounds are welcome in the Calyx community. We actively work to make every community member feel safe and empowered to contribute.

3. *Research/application synergy.* Calyx is both a practical tool for industrial applications *and* a platform for experimental research. The project balances stability and innovation so that these goals work together instead of conflicting.

Figure 3 illustrates our planned governance structure and how it interacts with the broader community around it. The core organizing mechanism is a *request for comment* (RFC) process, where all organizational and technical decisions are made in the open (Section 3.1.1). We will create a set of consensus-based *leadership teams* (Section 3.1.2) to manage the project's direction with the formal

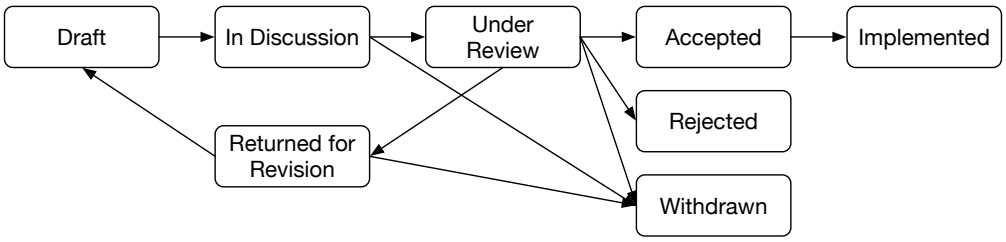

Figure 4: The process for proposing and accepting requests for comment (RFCs), the Calyx ecosystem's primary mechanism for making decisions in the open.

power to accept or reject RFC proposals. Finally, the project will designate involved contributors as *committers*, granting them the power to merge contributions as directed by the leadership teams and the RFCs they ratify. This organization is modeled after the governance structure behind several modern programming languages, such as Swift, Rust, and Python, which also emphasize participatory transparency in the language evolution process.

During the first year of this project, we will contract with an open governance consultant, Shauna Gordon-McKeon, to implement the formal bylaws that reflect this structure. Gordon-McKeon is an expert on open governance and open-source projects who will ensure that Calyx follows the best known practices for setting up sustainable ecosystems.

3.1.1 The RFC-Based Evolution Process for Open Decision-Making

The core governance mechanism in the Calyx ecosystem will be *request for comment* (RFC) documents. The RFC process ensures that all decisions are made *in the open*, to keep the process transparent and accountable to the entire community. While we rely on smaller leadership teams to make final calls (see Section 3.1.2), the full community is involved in proposing, discussing, designing, and implementing every change to Calyx.

RFCs, modeled after the Internet Engineering Task Force (IETF)'s standardization process, are documents that describe technical or organizational changes. Many open-source language projects implement a similar process, such as Python Enhancement Proposals (PEPs) [45] or Swift's evolution process [51], to codify every new unit of work in detailed, archived written justifications. Embodying the value of *accountability and legitimacy*, the RFC process ensures that every decision is made transparently, with input from the entire community. RFCs cover both technical work (e.g., new features in the Calyx language or tooling) and nontechnical, organizational decisions (e.g., policy changes or the creation of a new leadership team). While minor code changes, such as bug fixes and performance improvements, can happen without an RFC, all major changes—including any user-visible functional changes to the Calyx tools—must be authorized by an accepted RFC.

Figure 4 depicts the flowchart for proposing, discussing, accepting, and eventually implementing RFCs. Each RFC has a specific status assigned at any given time. Anyone in the community can draft a new RFC by sending a pull request to the Calyx project's RFC repository in *Draft* status. To submit an RFC for community input, the author posts a broad announcement to the project's discussion forum. While *In Discussion*, and the entire community—not just the most closely interested members—are encouraged to provide criticism and context.

When the author believes an RFC has been revised sufficiently for a final decision, they freeze the text and mark it as *Under Review* for the appropriate leadership team. If the leadership team agrees to accept a proposal, the RFC text is finalized and becomes guidance to the project's contributors and committers. All functional changes to Calyx require an accepted RFC, so *Accepted* is the state where the project formally solicits code contributions that implement the change. After a committer merges changes that implement an accepted RFC, that RFC is marked as *Implemented*

to indicate that it is now part of the Calyx project. The RFC then becomes a permanent part of the project's documentation.

3.1.2 Leadership Teams

The Calyx ecosystem will adopt a set of *leadership teams:* small, consensus-based groups whose primary responsibility is to review and decide on RFCs. Leadership teams are the key to the ecosystem's sustainability; their roles persist beyond the involvement of any individual contributor. We envision a growing set of leadership teams in the future, but the project will start with three:

- The *Steering team* provides high-level guidance to the entire project. This team handles organizational (non-technical) RFCs, including ones to create new leadership teams. This team is also responsible for enforcing the community's Code of Conduct (see Section 3.3.1).
- A *Language Evolution team* manages the specification of the Calyx IL. This team handles any RFCs that add features or break backwards-compatibility in the language.
- The *Compiler & Tools team* covers the concrete infrastructure in the Calyx project: the compiler, debugger, frontends, and so on. Initially, it will review all technical RFCs that do not affect the Calyx IL's semantics. This team is also responsible for handling security disclosures and responding to vulnerabilities.

All Calyx leadership teams use consensus to make decisions. While most communication among the teams should happen in the open, on GitHub issues and discussions, we plan to set up a private communication channel for sensitive topics (such as security reports and Code of Conduct violations). Teams will meet monthly to supplement these asynchronous communications. The membership for all three will start with the current Calyx development team; the team can agree to add new members via consensus. As the community grows, the Steering team may consider changing the membership model to use community elections instead.

We expect to add new, topic-specific leadership teams as the project scales. Via the standard RFC process, community members can create a new team for any topic that becomes a distinct area of focus—such as splitting specific components from the umbrella Compiler & Tools team.

3.2 Technical Infrastructure to Support the Ecosystem

The second major focus in this proposal is on putting into place the technical infrastructure for the ecosystem to succeed. The key insight is that, to become self-sustaining, the Calyx OSE needs to engage stakeholders across a variety of different accelerator application domains. It is therefore critical for the ecosystem to integrate with the existing, mainstream tools that accelerator designers already use. These integrations will provide entry points for Calyx users and ultimately "jump start" a virtuous cycle of adoption, contribution, and development.

Concretely, the greatest need is for deep integrations with FPGA and ASIC development workflows. We propose three main ecosystem initiatives: supporting FPGA toolchains (Section 3.2.1), integrating with open-source ASIC ecosystems (Section 3.2.2), and a new target-specific optimization framework to extract the best performance from both technology backends (Section 3.2.3).

3.2.1 Deploying on FPGAs and FPGA/CPU Hybrids

We aim to add first-class support for compiling Calyx-based accelerators to AMD and Intel FPGAs via each vendor's proprietary toolchains. We see three main technical challenges in making this kind of development workflow practical:

1. *Automating interactions with vendor toolchains.* The toolchains for the two leading FPGA vendors (AMD and Intel) typically require manual intervention to produce final FPGA

binaries. Calyx already has initial support for automatically orchestrating a specific version of AMD's tools, but this support is limited to basic, bulk-transfer accelerator interfaces. This work will build equivalent support for Intel's FPGA toolchain and add a new self-hosted continuous integration (CI) pipeline that automatically tests the end-to-end workflow on every Calyx commit. The end result will be a documented, robust mechanism to produce FPGA binaries for either platform from Calyx-based compilers with a single command.

The new CI infrastructure will require physical equipment to run the vendor toolchains, which can require many hours per invocation, and to run experiments on real FPGAs. Because we expect this testing workload to scale with the Calyx ecosystem this proposal budgets for one multi-FPGA-equipped server per year to host the growing workload.

2. *The host–accelerator interface.* Realistic deployments of FPGA-based accelerators need robust, efficient interfaces between the CPU and the FPGA. On both Intel and AMD platforms, this interface consists of an AXI or Avalon protocol for low-level data exchange together with a driver and C software library for invoking this interface [8, 30]. These interfaces are very flexible: they support custom bus widths, streaming to overlap communication and computation, and bursting to mitigate per-transfer overhead. Implementing a high-performance interface customized for a specific application is therefore typically an involved, manual engineering project. Calyx has proof-of-concept support for generating a basic, low-performance AXI wrapper for simple synchronous accelerators that works with AMD's XRT host interface [8]. This component of the project will develop a flexible, high-performance AXI interface generator that can stream data for a wider range of accelerator data ports. The plan is to design a small interface definition language (IDL) for describing the desired interface and a compiler from this IDL into Calyx code to implement the interface for either AMD or Intel devices. We will also generate host-side wrapper code for C and Rust programs on top of the two vendors' software libraries to make it simple to use these interfaces.

3. *Incorporating off-chip memories.* While FPGAs have small, fast on-chip SRAMs, even medium-scale workloads require storing data in the FPGA's off-chip memory. Several modern FPGAs include dedicated high-bandwidth memory (HBM) stacks specifically for this purpose. To make Calyx-based FPGA acceleration practical, we will build new abstractions for using these memories from within Calyx code. The abstraction will consist of a new library interface for accessing generic, variable-latency, block-oriented data stores, together with concrete implementations of this interface that link to vendors' provided IP libraries for interacting with these memories. The end result will be a push-button mechanism for Calyx frontends to exploit bulk data storage without per-vendor engineering.

3.2.2 Integrating with Open-Source ASIC Ecosystems

This thrust of the infrastructure work will focus on integrating with the emergent open-source ecosystem for ASIC development. Specifically, we aim to build first-class support for synthesizing Calyx-based accelerators with the OpenROAD EDA toolchain [5] and open-source PDKs [2, 4]. Extending our push-button FPGA toolchain automation support from Section 3.2.1, we plan to build support for sending Calyx-generated HDL code through the OpenROAD toolchain to produce GDS designs. A critical component of this effort will be integrating with an open-source memory compiler such as DFFRAM [52] or OpenRAM [24] to include on-chip SRAMs optimized for a specific silicon process.

As a "challenge goal" to demonstrate the viability of this fully open-source accelerator toolchain, we aim to participate in a future iteration of the TinyTapeout multi-project fabrication run that uses

the open-source SKY130 PDK [4]. We will demonstrate a push-button, fully automated workflow that generates a tapeout-ready design from a high-level language that compiles to Calyx.

3.2.3 Target-Specific Optimization Framework

Calyx's current compiler passes focus on *target-independent* optimization, in the sense that they heuristically improve power, performance, and area (PPA) on average for any physical implementation. While this approach works for coarse-grained, generic transformations where there are no complex trade-offs, many realistic optimizations depend deeply on the concrete target. A given FPGA model will prefer specific bit-widths for arithmetic operations, for example, and every ASIC process and standard-cell library comes with different implications for timing closure and pipeline depth. Extracting the best performance from each hardware backend requires incorporating information about the target process into the optimization pipeline.

Rather than design individual target-specific optimizations, the proposed work will implement generic infrastructure for supporting this category of optimization. The framework will include a format for specifying a *cost model* for a specific backend workflow: relative area and delay estimates for each element in an abstract standard-cell library based on the common Liberty [3] format. The Rust compiler will expose this cost-model information via Rust interfaces that lets optimization passes estimate the area and delay impact of a specific transformation before committing to it. We plan to showcase the new framework by enhancing Calyx's existing *resource sharing* optimization to minimize a target-specific area estimate instead of a generic one.

3.3 Community Building and Engagement

While Calyx already has an established user and developer base, a core focus in the proposed work is building the infrastructure to support users and contributors as they join the ecosystem. A vibrant, sustainable ecosystem requires pluralistic participation from a large community of stakeholders. To this end, it is a critical priority to make it as easy as possible to start using Calyx and to start contributing back.

Reflecting Calyx's principle of balancing research with applications, our work on this theme focuses on both categories of contributors. Our aims in this thrust are threefold: we will build the fundamental project infrastructure for a welcoming, accessible community, including documentation and community norms (Section 3.3.1); we will conduct active outreach to research communities, including tutorials and training workshops (Section 3.3.2); and, we will establish direct collaborations with industry and government users to help adopt Calyx for specific accelerator generation projects (Section 3.3.3).

3.3.1 Project and Community Infrastructure

This thrust of the proposed work focuses on developing *community infrastructure* to foster the long-term health of the Calyx ecosystem. Our aim is to support Calyx's contributors as they learn to make technical contributions while feeling safe in the community.

Community norms. A healthy open-source ecosystem requires an inclusive, respectful, and equitable community. Calyx will adopt the Contributor Covenant [14], a widely used code of conduct for open-source projects that specifies standards around respect, inclusiveness of minorities and underrepresented groups, and keeping discussions constructive. The Contributor Covenant includes explicit mechanisms for community moderation, including an escalating scale of warnings that can eventually lead to banning people who repeatedly violate the guidelines.

Maintaining a safe and inclusive environment will require ongoing effort to enforce the Contributor Covenant. This proposal includes support for developers and administrative support staff; these staff members will have specific responsibilities to monitor communications in the

community to flag violations of the Covenant and other interpersonal problems that arise in the community. When disputes arise that cannot be handled by individual community members, the staff will involve the Steering team (Section 3.1.2) to moderate and provide final decisions.

Expanded and reorganized documentation. A main focus over the first year of this award will be on expanding and refining Calyx's technical documentation. By the typical standards of research projects, Calyx already has extensive documentation: it has "getting started" guides, a basic tutorial for the Calyx IL and its associated tooling, an elaborate hands-on tutorial that guides users through writing a DSL-to-hardware compiler using the complete framework, and a detailed language reference. However, this documentation has grown organically along with the infrastructure itself, and its level of coverage can be inconsistent.

A main priority for the development staff and research personnel during this project will be auditing the entire "surface area" of the Calyx language and tooling and adding documentation throughout. We plan to build tools that automatically measure the fraction of API features and IL constructs that have documentation so we can track our progress toward complete coverage. By the second year of the proposed project, we aim to enter a maintenance phase where we enforce complete coverage by implementing continuous integration (CI) checks that automatically reject new contributions that do not come with associated documentation for new features.

Technical contribution onramps. Expanding the Calyx community will require "onramps" that make it easy to convert passive users into active contributors. We believe that the best way to begin contributing is to *learn by doing:* to start by making a compact, low-stakes contribution that nonetheless benefits the project in a small way. This onramp process provides a critical opportunity for the current community to offer guidance through the process and technical feedback on the contribution. During the two years of this project, the technical staff will focus on building and maintaining a library of GitHub issues labeled specifically for newcomers to the project. We plan to build a section in Calyx's documentation that points to this evolving list of onramp issues and includes step-by-step guidance about how to approach one. When a new contributor starts addressing one of these issues, all members of the project will prioritize offering hands-on feedback and guidance to help induct the contributor into the community.

3.3.2 Training for a Research Audience

Researchers, especially in academia, are a core audience for the Calyx ecosystem. We want to include both *systems* researchers, who are interested in optimization passes and new language constructs, and *applications* researchers, who simply want to generate efficient hardware accelerators for their workloads.

Our main plan for engaging the research audience is through hands-on tutorials, which we will run both in person at conferences and virtually. The Cornell team already ran a successful Calyx tutorial at FCRC 2023, where a dozen in-person attendees and several virtual attendees wrote their first Calyx IL code and built proof-of-concept compilers for a high-level DSL. As part of this tutorial, the team also produced comprehensive asynchronous materials for the same "lesson plan" that are now part of the public Calyx documentation.

The proposed work will include two synchronous tutorials per year: one co-located with a systems-focused venue (e.g., PLDI, ISCA, or ASPLOS) and one at an application venue (e.g., NeurIPS, SIGMOD, or Supercomputing). As part of preparing each tutorial, the team will iterate on the associated open-source materials each time so the same approach remains available asynchronously to interested users. A major focus in the curricular development will be on designing novel examples that are relevant to each application domain. Each tutorial offering will be "hybrid" to allow worldwide participation via videoconferencing.

3.3.3 Non-Academic Collaborations

Industry, government, and non-profit collaborators will be an important component of the Calyx community's growth. In the second year of the proposed work, our focus will shift to working directly with external teams to help them start using Calyx for their specific applications. While the long-term goal is to support these teams through a self-sustaining community, we aim to "bootstrap" this community in part through direct assistance to a small number of potential new users. Our technical team will directly contact teams in industry and at national labs who have ongoing hardware acceleration efforts, such as at Sandia National Labs [15], Google, and OctoML, the stewards of the TVM open-source machine learning compiler [12]. Our aim is to find teams who are interested in compiling a new or existing high-level language to hardware and provide direct technical guidance with the ultimate goal of building the community around the project. Through these case-study applications, we aim to identify technical and documentation gaps in the Calyx ecosystem and guide future development to support similar applications.

As part of the Calyx OSE's long-term sustainability goals, the project will implement a formal membership structure for private companies that want to participate in the Calyx ecosystem. Organizations will join by paying yearly dues in the form of unrestricted gifts, which will subsidize infrastructure development work and research. Member companies will receive formal acknowledgment on the Calyx project website and opportunities to advertise open positions to Calyx contributors. Critically, membership entails no special access to the Calyx ecosystem itself—all code remains equally available to everyone, and RFC-based decision-making remains open to the full community. The primary benefit of membership is helping sustain an ecosystem that provides value to the companies that use it.

4 Broader Impacts

A flourishing open-source ecosystem around Calyx will lower the barrier to entry for hardware acceleration. The broader impacts on society are potentially profound: any application of computing and any small technology enterprise has the potential to gain access to the hundred-fold efficiency gains that accelerators can offer over CPUs [25]. This magnitude of efficiency gain can amount to the difference between practicality and impracticality. Improved approaches to widespread problems like machine learning, physical simulation, or genomic assembly may become competitive with entrenched methods where, on CPUs, they would be too costly to be practical. Efficient accelerators will also mitigate the growing energy costs and emissions from the technology sector, which has already surpassed 1% of worldwide electricity use [37].

A key feature of the proposed Calyx OSE is in its planned contributions to diversity, equity and inclusion (DEI) and educational activities. We plan to partner with multiple on-campus groups at Cornell, such as Women in Computing at Cornell (WICC) and the Underrepresented Minorities in Computing Club (URMC), to expose a broader and more diverse set of computing students to open-source development. In collaboration with these clubs, the Calyx technical staff will create a new yearly workshop where WICC and URMC members learn the basics of open-source contributions: learning about projects on forges like GitHub, finding places to contribute, and following the project's guidance about how to help. These skills are part of the "hidden curriculum" in an undergraduate CS degree: while they are never taught in any standard course, they are an integral part of many computing careers. A history of open-source contributions can even help enhance applications to software engineering roles. This new workshop will use a hands-on style to guide students through their first open-source contribution. After the two-year period of this proposed project, we plan to scale up this effort by disseminating the materials to other universities as a "recipe" to run similar workshops.

PI Sampson also co-founded Cornell's Bowers CIS Undergraduate Experience (BURE) program, which involves undergraduates in paid research projects over the summer and helps prepare them for PhD applications. He will create a new lecture about open-source research software as part of BURE's Research Skills Seminar Series. The new content will expose these new researchers to open-source forms of impact and emphasize that—beyond just publishing papers—computer scientists have a responsibility to disseminate the software systems behind their work.

Finally, the Cornell team will launch a new educational initiative to disseminate the fundamental concepts and skills behind accelerator generation. Hardware design and compiler implementation are both notoriously difficult topics to learn from scratch, and both are rapidly becoming more important parts of the technology landscape. This component of the proposed work will take the form of a series of blog posts, written by the entire team at Cornell (the PhD student, the postdoctoral researcher, the staff programmer, and the PI), that teach the basics of hardware acceleration, synthesis toolchains, and ADL design. The main goal of these blog posts is to help people build the skills and background necessary to become Calyx contributors—but they will also serve as more general educational vehicles on the topic of accelerator generation. This educational blog will become a part of the official Calyx website, and it will persist as an ongoing project throughout the proposed term of work and beyond.

5 Execution Plan

This two-year project will fund three interlocking streams of work comprising *governance*, *technical*, and *collaboration* activities. The common goal is to bootstrap a virtuous cycle that sustains the Calyx OSE in the long term. We will establish a formalized development process, which supports technical development, which in turn drives adoption, which leads to industrial funding and engineering contributions, which powers the ecosystem as a whole. Two years of effort in each of these categories will build the momentum that Calyx needs to become self-sustaining. We will quantify progress in each of these categories to evaluate the health of the Calyx OSE.

Establishing governance. We plan to work with an external consultant on open-source governance to implement our proposed organization for the Calyx OSE (see Section 3.1). Shauna Gordon-McKeon (see letter included in supplementary materials) is an experienced advisor to open-source projects and runs the *Governing Open* collection of best practices for running self-sustaining open-source projects. Gordon-McKeon will be instrumental in helping launch a Calyx OSE that productively combines efforts from industry and academia.

This proposal will also support a new, half-time administrative coordinator who will manage the Calyx OSE. The administrator will be responsible for coordinating and running bi-weekly meetings for all Calyx staff, organizing workshops and tutorials with our community partners, liaising with non-academic collaborators, and helping moderate the technical communication channels for the open-source project.

Technical integrations. This project will construct the technical foundations necessary for widespread Calyx adoption. Our goals do *not* include general development of the Calyx infrastructure; instead, we are narrowly focused on implementing integrations that connect Calyx to the broader landscape of hardware development tooling (see Section 3.2 for details). These externally-facing integrations will make it straightforward for industrial and academic teams to start using Calyx for accelerator development, which is a critical ingredient in establishing a self-sustaining ecosystem.

A small software engineering team, consisting of one full-time and one half-time engineer, will focus on implementing these system integrations. The full-time staff member will focus on deep integration with open-source ASIC toolchains (Section 3.2.2) and accompanying optimizations that will make Calyx a competitive option for ASIC development (Section 3.2.3). The second,

part-time engineer will focus on the critical near-term task of integrating with FPGA toolchains (Section 3.2.1), which can make Calyx useful even for acceleration projects with a limited budget that precludes ASIC deployment. This team will also be responsible for overhauling Calyx's approach continuous integration and performance measurement, writing new documentation, and responding to requests from external collaborators.

Collaborations with academia and industry. This proposal will support the creation of a suite of collaborations between the core Calyx team and external users in both industry and academia. These collaborations are the key to expanding the audience of users who are invested in the Calyx OSE, which in turn is required for a self-sustaining project after the two-year period of this award. Our strategy is to view these open-source collaborations as tightly integrated with research efforts: we will seek collaborations that yield *both* open-source contributions *and* publishable scientific results. Our lab has an established track record of pursuing research with this dual form of impact, including multiple collaborations with industry teams oriented around open-source artifacts [54, 55]. We see open-source impact as a coequal form of research dissemination to peer-reviewed papers and aim to pursue both together.

To lead this effort, this proposal includes *half* support for one postdoctoral research and *half* support for each of two PhD students. All three of these personnel will also be supported by other awards with a pure research focus. They will dedicate half their time to building open-source collaborations with research teams either at other universities or in industry or government labs (see Section 3.3.3). These collaborations will also entail traditional research projects that occupy the other half of their time, not supported by this award. They will also necessarily involve coordination with the Calyx software engineers to build new capabilities to drive adoption of the ecosystem.

Evaluation. To evaluate the success of the Calyx OSE, we will track a collection of technical and adoption metrics on a quarterly basis. The quantitative metrics include activity statistics from Calyx's monorepo: the number of unique contributors, commit counts, lines of code added or deleted, and the word count among new discussions and issue comments. We also plan to manually track the number of external projects we are aware of that build on the Calyx ecosystem, categorizing each into "exploring" or "implementing" stages. Finally, we will track the number of participants in the tutorials we run to train new researchers on the Calyx ecosystem (Section 3.3.2). In general, we aim to increase these various activity metrics by roughly 2× by the end of the first year and 5× by the end of the second (measured relative to the start date of the award). Depending on quarterly progress reports on these metrics, we plan to adapt our plan and address problems that are limiting the ecosystem's growth.

6 Results from Prior NSF Support

PI Sampson is PI on NSF Award "CAREER: Type-Driven Heterogeneous Programming" (#1845952; $549,997; March 2019–June 2024). **Intellectual Merit.** This project focused on designing programming languages for heterogeneous systems: GPUs, FPGAs, and DSPs [18, 40, 53]. This award supported the creation of the original Calyx IL design and its compiler [41]. This POSE proposal is to build on that fundamental research contribution to deliver a robust, reusable ecosystem that serves both as a platform for future research and as a tool for industrial applications. **Broader Impacts.** PI Sampson designed and disseminated a self-guided online course on compilers that has been viewed over 6,000 times. He initiated a new set of programs in Cornell's Bowers CIS College to broaden access to research for undergraduates, including a funded summer mentorship program and a new academic-year matching mechanism.

References Cited

[1] Calyx: A compiler infrastructure for accelerator generators. GitHub Repository. https://github.com/cucapra/futil.

[2] GF180MCU open-source process design kit. https://gf180mcu-pdk.readthedocs.io/.

[3] Liberty user guides and reference manual suite version 2013.03. URL https://people.eecs.berkeley.edu/~alanmi/publications/other/liberty13_03.pdf.

[4] SKY130 open-source process design kit. https://skywater-pdk.readthedocs.io/.

[5] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, Geraldo Pradipta, Sherief Reda, Mehdi Saligane, Sachin S. Sapatnekar, Carl Sechen, Mohamed Shalan, William Swartz, Lutong Wang, Zhehong Wang, Mingyu Woo, and Bangqi Xu. Toward an open-source digital flow: First learnings from the OpenROAD project. In *Design Automation Conference (DAC)*, 2019.

[6] Amazon. How silicon innovation became the 'secret sauce' behind AWS's success. *Amazon Science*, July 2022. URL https://www.amazon.science/how-silicon-innovation-became-the-secret-sauce-behind-awss-success.

[7] AMD. Vitis HLS, . URL https://www.xilinx.com/products/design-tools/vitis/vitis-hls.html.

[8] AMD. Xilinx runtime library (XRT), . URL https://www.xilinx.com/products/design-tools/vitis/xrt.html.

[9] 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 *Design Automation Conference (DAC)*, 2012.

[10] Griffin Berlstein, Rachit Nigam, Chris Gyurgyik, and Adrian Sampson. Stepwise debugging for hardware accelerators. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[11] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H Anderson, Stephen Brown, and Tomasz Czajkowski. LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2011.

[12] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[13] Nikil D Dutt, Tedd Hadley, and Daniel D Gajski. An intermediate representation for behavioral synthesis. In *Design Automation Conference (DAC)*, 1991.

[14] Coraline Ada Ehmke. Contributor covenant code of conduct, v2.1. URL https://www.contributor-covenant.org/version/2/1/code_of_conduct/.

[15] Noah Evans. Sandia's formal hardware design and verification, present and future. In *Programming Languages for Architecture (PLARCH)*, 2023.

[16] Amin Firoozshahian, Joel Coburn, Roman Levenstein, Rakesh Nattoji, Ashwin Kamath, Olivia Wu, Gurdeepak Grewal, Harish Aepala, Bhasker Jakka, Bob Dreyer, Adam Hutchin, Utku Diril, Krishnakumar Nair, Ehsan K. Aredestani, Martin Schatz, Yuchen Hao, Rakesh Komuravelli, Kunming Ho, Sameer Abu Asal, Joe Shajrawi, Kevin Quinn, Nagesh Sreedhara, Pankaj Kansal, Willie Wei, Dheepak Jayaraman, Linda Cheng, Pritam Chopda, Eric Wang, Ajay Bikumandla, Arun Karthik Sengottuvel, Krishna Thottempudi, Ashwin Narasimha, Brian Dodds, Cao Gao, Jiyuan Zhang, Mohammed Al-Sanabani, Ana Zehtabioskuie, Jordan Fix, Hangchen Yu, Richard Li, Kaustubh Gondkar, Jack Montgomery, Mike Tsai, Saritha Dwarakapuram, Sanjay Desai, Nili Avidan, Poorvaja Ramani, Karthik Narayanan, Ajit Mathews, Sethu Gopal, Maxim Naumov, Vijay Rao, Krishna Noru, Harikrishna Reddy, Prahlad Venkatapuram, and Alexis Bjorlin. MTIA: First generation silicon targeting Meta's recommendation systems. In *International Symposium on Computer Architecture (ISCA)*, 2023.

[17] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. A configurable cloud-scale DNN processor for real-time AI. In *International Symposium on Computer Architecture (ISCA)*, 2018.

[18] Dietrich Geisler, Irene Yoon, Aditi Kabra, Horace He, Yinnon Sanders, and Adrian Sampson. Geometry types for graphics programming. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, November 2020.

[19] William Gibson. "The future is already here—it's just not evenly distributed." ca. 1990, paraphrased. cf `https://quoteinvestigator.com/2012/01/24/future-has-arrived/`.

[20] Google. XLS: Accelerated HW synthesis. `https://google.github.io/xls/`.

[21] Sergi Granell Escalfet. Accelerating Halide on an FPGA. Master's thesis, Universitat Politècnica de Catalunya, May 2023.

[22] Zhi Guo, Betul Buyukkurt, John Cortes, Abhishek Mitra, and Walild Najjar. A compiler intermediate representation for reconfigurable fabrics. *International Journal of Parallel Programming*, 2008.

[23] S Gupta, Renu Gupta, Nikil Dutt, and Alex Nicolau. *SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits*. January 2004.

[24] Matthew R. Guthaus, James E. Stine, Samira Ataei, Brian Chen, Bin Wu, and Mehedi Sarwar. OpenRAM: An open-source memory compiler. In *IEEE International Conference on Computer-Aided Design (ICCAD)*, 2016.

[25] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *International Symposium on Computer Architecture (ISCA)*, 2010.

[26] John L. Hennessy and David A. Patterson. A new golden age for computer architecture. *Communications of the ACM*, 62(2):48–60, January 2019.

[27] Yann Herklotz, Zewei Du, Nadesh Ramanathan, and John Wickerson. An empirical study of the reliability of high-level synthesis tools. In *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021.

[28] Sara Hooker. The hardware lottery, 2020. URL `https://arxiv.org/abs/2009.06489`. arXiv preprint, cs.CY 2009.06489.

[29] Intel. Intel HLS Compiler, . URL `https://www.altera.com/products/design-software/high-level-design/intel-hls-compiler/overview.html`.

[30] Intel. Open programmable acceleration engine (OPAE), . URL `https://opae.github.io`.

[31] Intel. Intel C/C++ compilers complete adoption of LLVM, August 2021. URL `https://www.intel.com/content/www/us/en/developer/articles/technical/adoption-of-llvm-complete-icx.html`.

[32] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a Tensor Processing Unit. In *International Symposium on Computer Architecture (ISCA)*, 2017.

[33] Yi-Hsiang Lai, Yuze Chi, Yuwei Hu, Jie Wang, Cody Hao Yu, Yuan Zhou, Jason Cong, and Zhiru Zhang. HeteroCL: A multi-paradigm programming infrastructure for software-defined reconfigurable computing. In *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

[34] Chris Lattner and Vikram Adve. LLVM: a compilation framework for lifelong program analysis and transformation. In *ACM/IEEE International Symposium on Code Generation (CGO)*, 2004.

[35] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: Scaling compiler infrastructure for domain specific computation. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021.

[36] LLVM Project. CIRCT: Circuit IR compilers and tools. URL `https://circt.llvm.org`.

[37] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[38] Mentor Graphics. Catapult high-level synthesis. https://www.mentor.com/hls-lp/catapult-high-level-synthesis/.

[39] Dave Neary, Josh Berkus, Katrina Novakovic, and Bryan Behrenshausen. Understanding open source governance models, July 2020. URL https://www.redhat.com/en/blog/understanding-open-source-governance-models.

[40] Rachit Nigam, Sachille Atapattu, Samuel Thomas, Zhijing Li, Theodore Bauer, Yuwei Ye, Apurva Koti, Adrian Sampson, and Zhiru Zhang. Predictable accelerator design with time-sensitive affine types. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2020.

[41] Rachit Nigam, Samuel Thomas, Zhijing Li, and Adrian Sampson. A compiler infrastructure for accelerator generators. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[42] Rishiyur S. Nikhil and Arvind. What is Bluespec? *ACM SIGDA Newsletter*, 39(1), January 2009.

[43] Tom Preston-Werner. Semantic versioning 2.0.0, June 2013. URL https://semver.org.

[44] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth, Gopal Jan, Gray Michael, Haselman Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Y. Xiao, and Doug Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *International Symposium on Computer Architecture (ISCA)*, 2014.

[45] Python project. Python enhancement proposals (PEPs), July 2000. URL https://peps.python.org/pep-0000/.

[46] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman P. Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, 2013.

[47] Parthasarathy Ranganathan, Daniel Stodolsky, Jeff Calow, Jeremy Dorfman, Marisabel Guevara, Clinton Wills Smullen IV, Aki Kuusela, Raghu Balasubramanian, Sandeep Bhatia, Prakash Chauhan, Anna Cheung, In Suk Chong, Niranjani Dasharathi, Jia Feng, Brian Fosco, Samuel Foss, Ben Gelb, Sara J. Gwin, Yoshiaki Hase, Da-ke He, C. Richard Ho, Roy W. Huffman Jr., Elisha Indupalli, Indira Jayaram, Poonacha Kongetira, Cho Mon Kyaw, Aaron Laursen, Yuan Li, Fong Lou, Kyle A. Lucke, JP Maaninen, Ramon Macias, Maire Mahony, David Alexander Munday, Srikanth Muroor, Narayana Penukonda, Eric Perkins-Argueta, Devin Persaud, Alex Ramirez, Ville-Mikko Rautio, Yolanda Ripley, Amir Salek, Sathish Sekar, Sergey N. Sokolov, Rob Springer, Don Stark, Mercedes Tan, Mark S. Wachsler, Andrew C. Walton, David A. Wickeraad, Alvin Wijaya, and Hon Kwan Wu. Warehouse-scale video acceleration: Co-design and deployment in the wild. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[48] Adrian Sampson. From hardware description languages to accelerator design languages. *Computer Architecture Today*, June 2021. URL https://www.sigarch.org/hdl-to-adl/.

[49] Amirali Sharifian, Reza Hojabr, Navid Rahimi, Sihao Liu, Apala Guha, Tony Nowatzki, and Arrvindh Shriraman. μir: An intermediate representation for transforming and optimizing the microarchitecture of application accelerators. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[50] Frans Skarman and Oscar Gustafsson. Spade: An HDL inspired by modern software languages. In *International Conference on Field-Programmable Logic and Applications (FPL)*, 2022.

[51] Swift project. Swift evolution process. URL `https://github.com/apple/swift-evolution/blob/main/process.md`.

[52] The American University in Cairo. DFFRAM Compiler. URL `https://github.com/AUCOHL/DFFRAM`.

[53] Alexa VanHattum, Rachit Nigam, Vincent T. Lee, James Bornholt, and Adrian Sampson. Vectorization for digital signal processors via equality saturation. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[54] Alexa Vanhattum, Daniel Schwartz-Narbonne, Nathan Chong, and Adrian Sampson. Verifying dynamic trait objects in rust. In *ICSE SEIP*, 2022.

[55] Alexa VanHattum, Monica Pardeshi, Chris Fallin, Adrian Sampson, and Fraser Brown. Lightweight, modular verification for WebAssembly-to-native instruction selection. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[56] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. The RISC-V instruction set manual, volume I: Base user-level ISA. Technical Report UCB/EECS-2011-62, EECS Department, University of California, Berkeley, May 2011. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.html`.

[57] Qiang Wu, Yunfeng Wang, Jinian Bian, Weimin Wu, and Hongxi Xue. A hierarchical CDFG as intermediate representation for hardware/software codesign. In *International Conference on Communications, Circuits and Systems (ICCCAS)*, 2002.

[58] Hanchen Ye, Cong Hao, Jianyi Cheng, Hyunmin Jeong, Jack Huang, Stephen Neuendorffer, and Deming Chen. ScaleHLS: A new scalable high-level synthesis framework on multi-level intermediate representation. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.