DORA: Open-Source Infrastructure for Prototyping Reconfigurable Fabrics

Shwet Chitnis*, Jiayi Wang, Fergus Xu, Ayush Kulkarni,
Rampranav Navendran, Juwon Jun, Jingqun Zhang, and Ang Li
University of Washington

***Abstract*—Open-source CGRA research needs infrastructure that can vary fabric architecture, RTL, compiler collateral, and configuration flow from one specification. Existing frameworks such as OpenCGRA and CGRA-ME, and fabric studies such as HyCUBE, have made CGRAs more accessible, but often center on one fabric family or compiler contract. We present Domain-Optimized Reconfigurable Architecture (DORA), a recipe-based Python framework for prototyping CGRA-style reconfigurable fabrics. A DORA recipe composes user-provided hardware *leaves* as a single source of truth for the compiler-visible architecture, generated RTL microarchitecture, and configuration feature space. From one recipe, DORA emits SystemVerilog, MRRG and latency metadata, operation metadata, FASM feature encodings, and bitstream metadata. DORA is paired with CGRA-Solve, a mapper that consumes DORA-emitted collateral and emits FASM assignments for DORA Bitgen. This recipe-to-artifact flow keeps the RTL, mapper view, and bitstream view synchronized as the fabric evolves. DORA has been used to prototype spatial fabrics spanning multiple initiation intervals, functional-unit implementations, and on-chip network organizations. This includes Fringe, a recently published precision-aware CGRA communication fabric from FCCM 2026, for which the DORA-based flow helped compress the path from architectural idea to concrete experiments in under 20 days. These experiences suggest DORA can serve as a compact, executable interface for future agentic design-space exploration.**

I. Motivation

CGRAs occupy a useful middle point between FPGAs and ASICs (Fig. 1). FPGAs provide fine-grained programmability, but their bit-level generality can obscure architecture-level questions. ASICs expose those questions sharply, but only after committing to a fixed design point. CGRAs offer a research substrate between these extremes: word-level compute, programmable interconnect, explicit memory structure, and enough regularity to support compiler and implementation experiments.

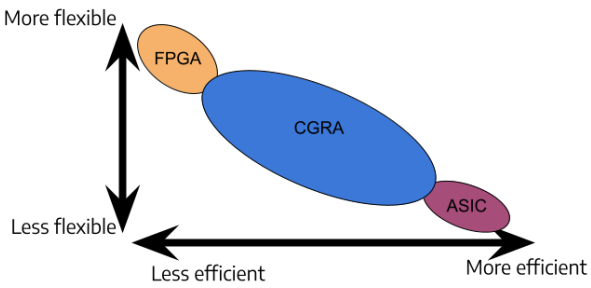

Fig. 1. Flexibility versus efficiency tradeoff.

*Contact: s2chitni@uw.edu

The difficulty is that CGRA innovations do not vary along a single dimension. Changing the execution model, routing granularity, memory interface, or configuration protocol often changes both the compiler-visible resource graph and the RTL-facing microarchitecture. This is why statically scheduled meshes such as HyCUBE [1], dataflow accelerators such as RipTide and Plasticine [2], [3], and FPGA-overlay-inspired fabrics are hard to compare using one rigid flow.

OpenCGRA [4] and CGRA-ME [5] are important steps toward open CGRA research infrastructure. PRGA [6] targets FPGA research and prototyping rather than CGRAs, but its Python-based methodology for separating architecture and microarchitecture models in hardware generators strongly influenced DORA's design. DORA applies that methodology to CGRA-style fabrics and adds an explicit control/configuration/datapath decomposition. In our experience, reusable exploration requires the framework to expose the same design choices that researchers want to perturb, rather than hiding them inside one canonical fabric generator.

II. Toolchain Overview

Figure 2 summarizes the toolchain contract. A DORA design begins as a Python recipe: an executable fabric specification that instantiates and connects DORA leaves. A leaf is a hardware object at the lowest DORA abstraction boundary. Like a macro in an EDA flow, it can contain arbitrary user-provided implementation detail, while the compiler treats it as opaque and sees only the interface and collateral that DORA exposes. Recipes let researchers bring their own leaves—functional units, registers, switches, memories, controllers, or configuration state—and compose them into rich reconfigurable dataflow fabrics.

DORA organizes this recipe along two axes. The first separates *architecture* from *microarchitecture*. The architecture view declares compiler-visible resources, operations, data types, ports, memories, routing resources, and timing constraints. The microarchitecture view commits how those resources become module instances, connections, registers, configuration storage, and generated SystemVerilog. The second axis separates the fabric into *control*, *configuration*, and *datapath* planes. Treating these planes as first-class objects lets a designer change, for example, a scan-chain topology or a multi-context schedule without rewriting the datapath generator or the compiler interface.

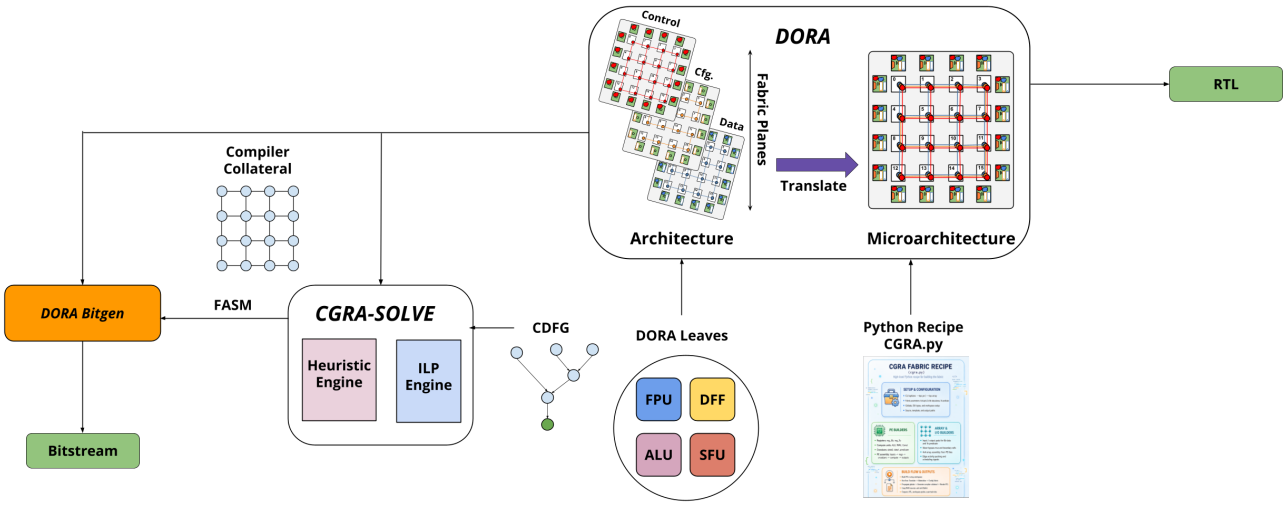

Fig. 2. A DORA Python recipe composes black-box hardware leaves into fabric planes, emits RTL and compiler collateral, and connects CGRA-Solve mappings to DORA Bitgen through FASM.

From the same recipe, DORA emits both implementation and mapping artifacts. The implementation path produces SystemVerilog RTL for simulation, synthesis, and physical-design experiments. The mapping path produces compiler collateral: an MRRG, operation and latency metadata, and feature encodings that describe which programmable choices can appear in a configuration. These artifacts are generated from the same source, so the compiler-visible resource graph, RTL structure, and bitstream features remain synchronized as the fabric evolves.

CGRA-Solve closes the loop from architecture model to programmed fabric across this mapper-agnostic boundary. Given an application CDFG, DORA-emitted compiler collateral, and latency information, it maps with either an ILP backend or a heuristic backend built from CGRA-ME-style mapping algorithms and PathFinder-style routing [5], [7]. CGRA-Solve emits FASM feature assignments, which DORA Bitgen combines with recipe-derived feature encodings to produce the final bitstream. This keeps mappers replaceable while tying bitstream generation to the same fabric model that generated the RTL.

DORA has already supported publication-grade fabric studies. Fringe, a precision-aware CGRA communication fabric published at FCCM 2026 [8], was prototyped in DORA, and CGRA-Solve was extended with precision-aware mapping; the framework helped move from an architecture idea to concrete mapping and implementation experiments in under 20 days. This experience points to a broader opportunity for agentic hardware-design workflows. Recent work on agentic AI for physical-design R&D argues that tool-integrated agents can interpret specifications, modify code, run EDA tools, analyze results, and iteratively refine design heuristics [9]. DORA extends this opportunity to reconfigurable-architecture exploration: executable recipes define a constrained loop for agents to edit fabrics, regenerate artifacts, run mapper and EDA tools, compare metrics, and refine architectures under recipe-level tests.

References

[1] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *Proc. Design Automation Conf. (DAC)*, 2017, pp. 45:1–45:6.

[2] G. Gobieski, S. Ghosh, M. Heule, T. Mowry, T. Nowatzki, N. Beckmann, and B. Lucia, "RipTide: A programmable, energy-minimal dataflow compiler and architecture," in *Proc. IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, 2022, pp. 546–564.

[3] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel patterns," in *Proc. ACM/IEEE Int'l Symp. on Computer Architecture (ISCA)*, 2017, pp. 389–402.

[4] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, "OpenCGRA: An open-source unified framework for modeling, testing, and evaluating CGRAs," in *Proc. IEEE Int'l Conf. on Computer Design (ICCD)*, 2020, pp. 381–388.

[5] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. H. Anderson, "CGRA-ME: A unified framework for CGRA modelling and exploration," in *Proc. IEEE 28th Int'l Conf. on Application-Specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 184–189.

[6] A. Li and D. Wentzlaff, "PRGA: An open-source FPGA research and prototyping framework," in *Proc. ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2021, pp. 127–137.

[7] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *Proc. ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 1995, pp. 111–117.

[8] S. Chitnis, F. Xu, A. Kulkarni, J. Wang, J. Zhang, A. Raje, and A. Li, "Precision-aware communication in CGRAs," in *Proc. IEEE Int'l Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2026.

[9] A. Ghose, A. B. Kahng, S. Kundu, and B. Pramanik, "Invited: Agentic AI for physical design R&D: Status and prospects," in *Proc. Int'l Symp. on Physical Design (ISPD)*, 2026, pp. 133–141.