Automatically retargeting hardware and compilers for RISC-V custom instructions

What is the big idea?

A new open-source HW/SW co-design toolset lets teams describe **custom instructions (CIs)** once and automatically produce both the **coprocessor hardware** and the **compiler support** for RISC-V systems. Crucially, it attaches accelerators through **standard sockets**—**Core-V eXtension Interface (CV-X-IF)** and **Rocket Custom Coprocessor (RoCC)**—so you don't modify the core RTL, easing verification and certification. In published results, this approach yields **double-digit speedups** with modest area/timing overheads. [llvm.org], [github.com], [github.com]

Why should you care?

- **Performance & efficiency**: The method reduces **execution time** by **38–40%** on CVA6 cores and **27%** on Rocket, with **7–8%** and **6%** area overhead respectively. Code streams also get denser: **static size −9%** and **instruction-fetch bits −41%**, which can lower cache pressure and system bandwidth. [llvm.org]
- **Modularity & reuse**: CV-X-IF and RoCC are widely used sockets; staying out-of-pipeline keeps the CPU largely unchanged, minimizing re-verification, which matters in safety-critical and certified environments. [github.com], [www2.eecs....rkeley.edu]
- **Developer experience**: An LLVM-based flow **dynamically loads** out-of-tree target plugins generated from the architecture description, so many patterns compile **automatically**; for tricky cases (e.g., **int4** math), auto-generated **C intrinsics** give precise control. [llvm.org], [codasip.com]

How does it compare?

- **SCAIE-V** (open-source) integrates **inside the pipeline**, supporting advanced instruction classes (control flow, memory, multi-cycle, decoupled) and often lower latency for very small CIs—but requires **core modifications** and deeper verification. [github.com]
- **Commercial ASIP suites** (Codasip Studio, Synopsys ASIP Designer) generate **entire processors** (compiler, simulator, RTL) from a single description—great for building new ISAs/VLIW/vector cores—at the cost of **licenses** and less community transparency. [github.com], [riscv.or.jp]

Bottom line: If you want **fast, low-risk** CI acceleration on established RISC-V cores, the CV-X-IF/RoCC approach is pragmatic and effective. If your workload demands **fine-grained control-flow CIs** with minimal latency, consider in-pipeline interfaces (SCAIE-V). If you're designing a **bespoke processor**, commercial ASIP tools may be a better fit. [llvm.org], [github.com], [riscv.or.jp]

Open, modular CPU ecosystems have made it far easier to tailor systems to specific workloads, but bridging the gap between **custom hardware** and **compiler support** remains

hard. The IEEE TVLSI article by Hepola et al. proposes an open HW/SW co-design toolset that **derives both coprocessor RTL and a retargetable LLVM-based compiler** from one architecture description, targeting two mainstream RISC-V accelerator sockets: **CV-X-IF** and **RoCC**. Their key claims are substantial: **average execution-time reductions** of **38–40%** (CVA6) and **27%** (Rocket) at **7–8%** and **6%** area overhead; and **dynamic code-generation support** that avoids rebuilding LLVM. [llvm.org]

A single source for hardware & compiler.

Hepola et al. capture CI semantics using **directed acyclic graphs (DAGs)** and **HDL snippets**. From that description, their toolset generates (1) coprocessors with wrappers for **CV-X-IF** and **RoCC**; and (2) an **LLVM retargeting** flow that compiles out-of-tree backend **plugins** so new instructions can be used **without** rebuilding the whole compiler. The team integrates with **CVA6** (single- and dual-issue) via CV-X-IF and with **Rocket** via RoCC, reporting **up to 88%** speedups for an **int4 GEMM** kernel and **double-digit** gains elsewhere, with modest area/timing costs. Figure call-outs make the interface structures explicit: *Fig. 7a* shows RoCC's **ready–valid** command/response; *Fig. 7b* shows CV-X-IF's **issue/register/commit/result** channels and the **instruction tracker** for out-of-order commit; *Fig. 9* summarizes benchmark gains; and **Tables V–VI** present synthesis/ density data. [llvm.org]

Interface design in context (CV-X-IF vs. RoCC).

- **CV-X-IF** is a ratified OpenHW spec providing independent channels and commit/squash semantics for offloaded instructions, supporting multi-hart, flexible widths, and out-of-order commit (v1.0.0). This suits medium/complex offloads while keeping the CPU unchanged. [github.com], [github.com]
- **RoCC** (Rocket/Chipyard) is a compact, well-documented socket for tightly-coupled accelerators, exposing a simple **ready–valid** command/response with optional memory/PTW/FPU paths; the trade-off is offload overhead that can exceed native execution for *very small* CIs (observed in **nettle-aes**). [www2.eecs....rkeley.edu], [github.com]

Sidebar — Ready–Valid vs. Commit/Squash (how the sockets differ)

RoCC (ready–valid): The core sends a command when **valid**; the accelerator accepts when **ready**; results return similarly via response. Minimal control logic and high throughput—but the extra hop can add **latency** that hurts tiny CIs (e.g., a 3-op AES chain), as the paper shows for Rocket (*Fig. 9*).

CV-X-IF (multi-channel): The core **issues** an unknown instruction, supplies **register** operands, later **commits** or **kills** for correct out-of-order behaviour, and the accelerator **writes back** via **result**. The paper's **instruction tracker** (keyed by hart+issue ID) and **round-robin** result selection implement this (see *Fig. 7b*), aligning with the ratified spec.

Bottom line: Use **RoCC** for simple, tight coupling when you can amortize offload latency; prefer **CV-X-IF** when you need clean separation and **commit/squash** control in deeper pipelines. [github.com], [llvm.org] [github.com], [llvm.org] [github.com], [github.com]

Compiler retargeting and intrinsics: how automatic is "automatic"?

The toolset **dynamically loads** target extensions into LLVM's llc: it auto-generates **TableGen** descriptors from DAG semantics, builds a small **plugin** (~8 MB), and caches it for reuse across builds. For patterns that heuristic selectors miss—especially **loop-carried**, sub-byte operations (e.g., packed **int4**)—the tool auto-emits **C headers** with intrinsics wrapping inline assembly, a pragmatic balance between usability and portability. While upstream LLVM documents TableGen and llc behavior (and advocates in-tree targets), the **out-of-tree plugin** approach is consistent with LLVM's modularity and is practical for CI experimentation. [llvm.org], [codasip.com], [deepwiki.com], [github.com]

Benchmarks and code density

On **BEEBS** workloads (**crc**, **aha-compress**, **nettle-aes**) and a packed **int4 GEMM** kernel, the toolchain delivers the headline speedups and improves **code density**: **static size −9%** and **fetched instruction bits −41%**, amplifying system-level benefits (less I-cache bandwidth). The authors note compressed ISA interactions can limit density gains in some chains (e.g., AES), an expected effect given register allocation and compressibility. [llvm.org], [github.com]

Feature & Integration Comparison Table

Option	Scope & Focus	Instruction Types	CPU Changes Required	Compiler Support Path	Typical Latency/ Overhead	Licensing	Best Use Case
CV-X-IF (OpenHW)	Standard coprocessor socket for CORE-V cores (e.g., CVA6 family)	ALU/dataflow CIs with **issue/register/commit/result**; out-of-order commit	**None** (socket integration only)	LLVM backend **plugin** + DAG patterns; **intrinsics** for complex ops	Offload latency, flexible commit; good for medium-granularity ops	**Open-source**	Add domain-specific accelerators while preserving **core verification** [github.com], [llvm.org]
RoCC (Rocket/ Chipyard)	Tight socket in Rocket tiles; easy accelerator	Command/response with **ready–valid**; optional L1/L2/PTW/FPU	**None** (tile config + LazyRoCC)	Same plugin/intrinsics flow	Very low wrapper cost; offload latency can **hurt small CIs**	**Open-source**	Rapid integration of accelerators in **Rocket-based SoCs** [www2.eecs..

Option	Scope & Focus	Instruction Types	CPU Changes Required	Compiler Support Path	Typical Latency/ Overhead	Licensing	Best Use Case
	attachment						..rkeley.edu], [github.com]
SCAIE-V	**In-pipeline** ISA extension interface across multiple cores	**Combinational, multi-cycle, memory, control-flow, decoupled** ISAX	**Yes** (core extension points modified)	External (interface tooling; compiler mapping must be arranged)	Often **lowest latency**; cost scales with ISAX features	**Open-source**	Fine-grained/ custom control & memory ops with tight coupling [github.com]
Codasip Studio	Full **ASIP** suite: compiler, simulator, RTL from **CodAL**	Any (new ISAs, custom instructions)	**New/modified processor** generated	Auto-retargeted **LLVM** backend; SDK generated	Depends on architecture explored	**Commercial**	Building a proprietary **custom processor** end-to-end [github.com]
Synopsys ASIP Designer	Full **ASIP** suite: C/C++ compiler, cycle+instruction ISS, RTL from **nML**	Any (VLIW/vector, encoded ISAs, special data types)	**New/modified processor** generated	**Compiler-in-the-loop** exploration; SDK & ISS auto-generated	Depends on architecture explored	**Commercial**	Industrial ASIP design with rich **verification & examples** [riscv.or.jp]

Note: The performance/area numbers cited are for CV-X-IF on CVA6 and RoCC on Rocket synthesized in **ASAP7** (predictive 7-nm FinFET PDK), with caches treated as closed boxes to isolate datapath effects. [llvm.org], [github.com]

Methods Appendix

Interfaces and implementation.

- **RoCC wrapper** (*see paper Fig. 7a*): Maps command operands (rs1/rs2/rd + instruction bits) through **ready–valid** queues to the function unit and returns results via **response**; integrated into **Chipyard** as a SystemVerilog black box under **LazyRoCC**,

with **OpcodeSet** routing configured at tile elaboration. This matches Chipyard's documented accelerator flow. [llvm.org], [github.com]

- **CV-X-IF wrapper** (*see paper Fig. 7b*): Implements **issue**, **register**, **commit**, **result** subchannels; adds an **instruction tracker** (hart+issue ID) and a **round-robin** selector at pipeline tail to handle multiple latencies and out-of-order results. Design behavior aligns with CV-X-IF v1.0.0 handshake semantics. [llvm.org], [github.com]

Compiler retargeting.

- **DAG-based selection**: Operation DAGs convert to **TableGen SelectionDAG** patterns; the llc driver loads a cached **plugin** keyed by a descriptor hash, or regenerates and caches one. Generated **intrinsics** wrap inline assembly for non-DAG or sub-byte patterns (e.g., int4 dot product). The approach leverages LLVM's TableGen machinery and user-facing llc behavior. [llvm.org], [codasip.com], [deepwiki.com], [github.com]

Benchmarks & synthesis corner.

- Benchmarks: **crc**, **aha-compress**, **nettle-aes** from **BEEBS** plus an **int4 128×128 GEMM**; BEEBS is widely used for embedded energy/performance studies. [llvm.org], [github.com]
- Synthesis: **Cadence Genus** on **ASAP7** for timing/area; ASAP7 is a standard academic, predictive PDK with public rule sets and cell libraries. [llvm.org], [github.com], [researchgate.net]
- Observations: CVA6 single-issue timing overhead ~**7%** (issue logic), superscalar ~**1%**; Rocket ~**2%** critical path increase; area **+7–8%** (CVA6), **+6%** (Rocket). The **GEMM int4** kernel sees **85–88%** speedups; **nettle-aes** on Rocket is slower with CIs due to **offload latency**, demonstrating socket selection trade-offs. [llvm.org]

Practical guidance: choosing an extension path

- Need **fast wins** on **dataflow-heavy kernels** (bit manipulation chains, packed dot products, crypto)? Use **CV-X-IF/RoCC** wrappers to **avoid touching core RTL** and preserve verification/certification. [llvm.org], [github.com], [github.com]
- Rely on fine-grained **control-flow** or **very small** CIs where interface latency dominates? Evaluate **SCAIE-V** or other **in-pipeline** approaches and budget for tighter core integration and verification. [github.com]
- Want to design a **new ASIP** (custom ISA, VLIW/vector, special datatypes)? Consider **Codasip Studio** or **Synopsys ASIP Designer** for mature compilers, ISS, and verification flows. [github.com], [riscv.or.jp]

Limitations and future directions

Automatic retargeting is strongest for **DAG-expressible** patterns and scalar register-file semantics. Integrating **vector (V) extension** semantics would require vector state connections and richer encodings at the sockets; packed SIMD using the scalar RF (as demonstrated) fits today's interfaces. The authors also identify **retargetable simulation** (e.g., **QEMU**/**gem5**) as a next step, consistent with open tooling; expanding packed SIMD patterns via OpenASIP's DAG flow could further improve gains. [llvm.org]

Hepola et al. deliver an **open, pragmatic bridge** between RISC-V CI hardware and compiler support, centred on **standard coprocessor sockets** (CV-X-IF/RoCC) and **out-of-tree LLVM**

target plugins. For teams balancing **performance**, **verification reuse**, and **tooling agility**, this architecture-description-driven flow is a credible route to **double-digit speedups** with modest area/timing costs. Compared with **SCAIE-V**, it trades some instruction generality for **modularity**; compared with **Codasip** and **Synopsys**, it offers a no-license path with strong community potential. Your choice depends on **workload granularity**, **verification constraints**, and **toolchain philosophy**. [llvm.org], [github.com], [github.com], [github.com], [github.com], [riscv.or.jp]

References

- Asanović, K., et al. (2016). *The Rocket Chip Generator (UCB/EECS-2016-17).* University of California, Berkeley. [www2.eecs....rkeley.edu]
- Chipyard Team. (2025). *Adding a RoCC Accelerator — Chipyard documentation (v1.11).* UC Berkeley. [github.com]
- Hepola, K., Ranasinghe Arachchige, T., Multanen, J., & Jääskeläinen, P. (2025). *Automatically Retargeting Hardware and Code Generation for RISC-V Custom Instructions. IEEE Transactions on VLSI Systems*, 33(10). [llvm.org]
- OpenHW Group. (2024). *Core-V eXtension Interface (CV-X-IF) Specification v1.0.0.* [github.com]
- Pallister, J., Hollis, S., & Bennett, J. (2013). *BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms.* arXiv:1308.5174. [github.com]
- Clark, L. T., et al. (2016). *ASAP7: A 7-nm FinFET predictive process design kit. Microelectronics Journal*, 53, 105–115. [github.com]
- Damian, M., Oppermann, J., Spang, C., & Koch, A. (2022). *SCAIE-V: An Open-Source SCAlable Interface for ISA Extensions for RISC-V Processors.* In *DAC '22*. [github.com]
- LLVM Project. (2025). *TableGen Programmer's Reference; TableGen Overview; llc Command Guide.* [codasip.com], [deepwiki.com], [github.com]
- Codasip. (2023). *Re-targetable LLVM C/C++ compiler for RISC-V (blog).* [github.com]
- Synopsys. (2025). *ASIP Designer tools and eUpdate (April 2025).* [synopsys.com], [www2.eecs....rkeley.edu]

Source: Automatically Retargeting Hardware and Code Generation for RISC-V Custom Instructions | IEEE Journals & Magazine | IEEE Xplore