RISC-V SUMMIT

NORTH AMERICA

RISC-V SUMMIT
NORTH AMERICA

Introduction to the Composable Custom Extensions Task Group (CX TG)

Darius Rad (Bluespec), Chair
Jan Gray (Individual), Vice-Chair

Oct. 21, 2024

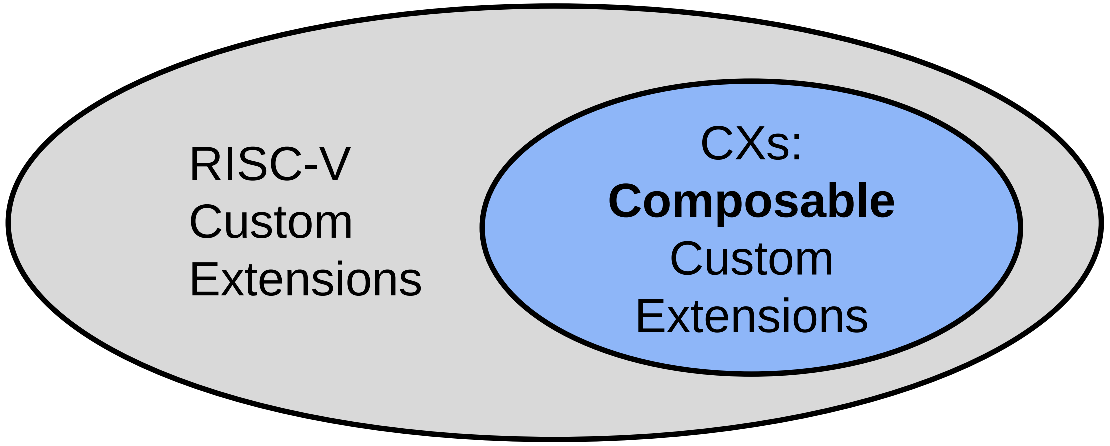
CX TG Overview

TG motivations

- Winter of Moore’s Law → proliferation of custom accelerators
- Reuse of multiple custom extensions solutions is rare
- Meta’s 2023 call for “standard specification for customization”

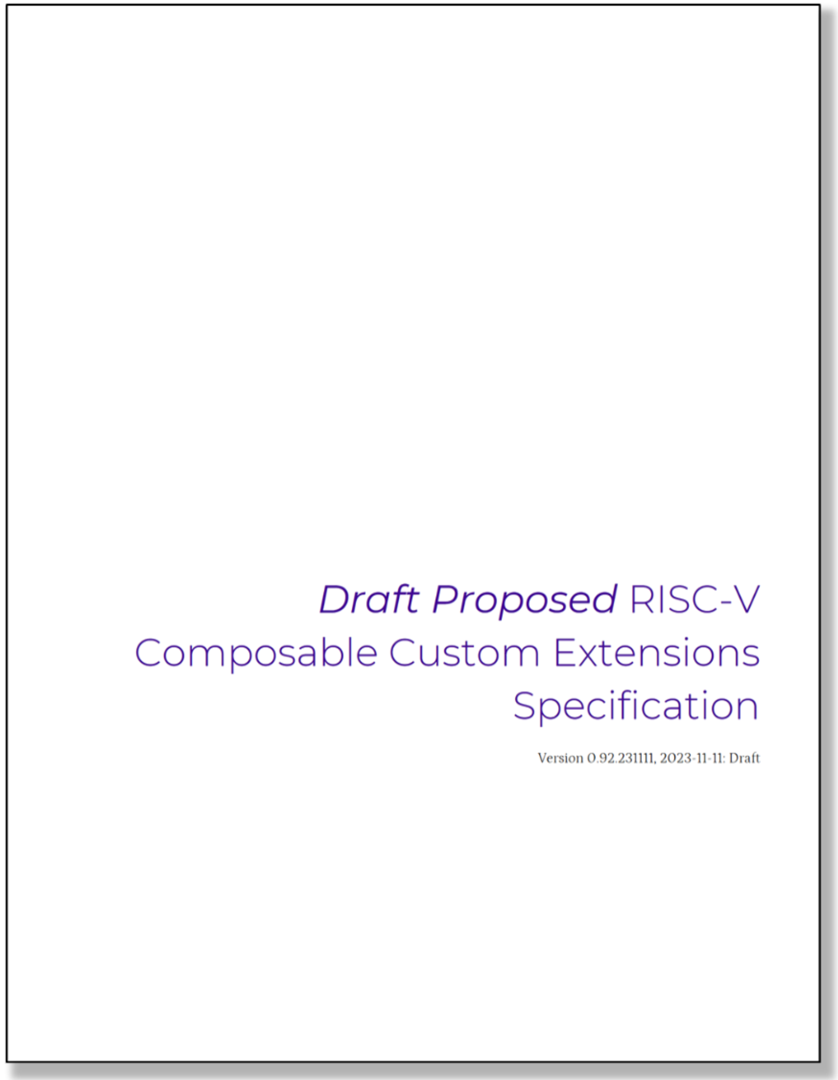
Composable Custom Extensions TG

- Develop ISA and non-ISA specifications for decentralized, cooperative reuse of the custom opcode space, compatible with ordinary custom extensions
- Enable reuse of **composable** custom extensions, libraries, and HW



TG history

- 2018: Lemieux isa-dev thread
- 2019: SoftCPU SIG's *CFU workgroup*
- 2022: Draft Proposed *RISC-V Composable Custom Extensions Specification* (~50 pp) – the TG's **basis spec**
  - VexRiscv, CFU Playground, Lattice, Efinix, AMD
- 2024: TG inception and approval
- Today: active status, plan milestone



Draft Proposed RISC-V Composable Custom Extensions Specification

Version 0.92.231111, 2023-11-11: Draft

TG key objectives

- Make it practical and routine to create and combine composable custom extensions’ software libraries and hardware modules
- Enable an ecosystem of reusable extensions IP, providing a catalog of accelerator solutions, and a market for accelerator developers

TG deliverables

A framework of ISA & non-ISA specs for a reusable extensions IP HW-SW stack:

1. Criteria for a custom extension to be deemed *composable*;
2. ISA extensions for CX multiplexing & OS CX state mgmt & access control
3. API and ABI specs for a uniform, composable CX programming model
4. Optional logic interface spec for portable, modular CX Unit hardware

Also, proof of concept SW & HW development

TG status

- Plan milestone of specification development
  - Requirements → work breakdown (schedule) → Ratification Plan presentation

- Governing Committee: Privileged Spec IC
- Consults: SoftCPU SIG, Unified Discovery TG, Platform Runtime Services TG, Toolchains & Runtimes SIG, psABI TG

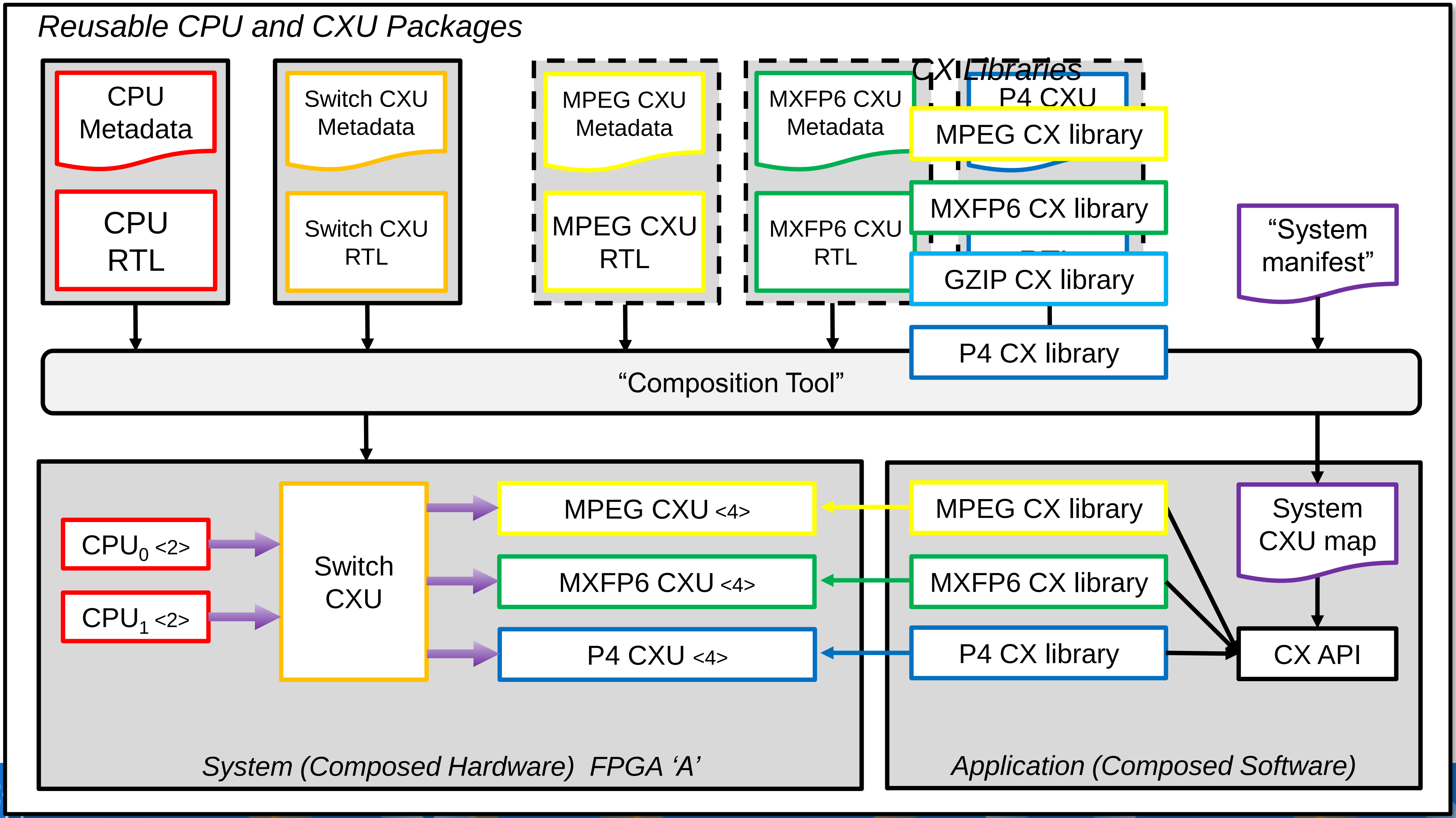
- *https://lf-riscv.atlassian.net/browse/RVG-160*

Technical Backgrounder

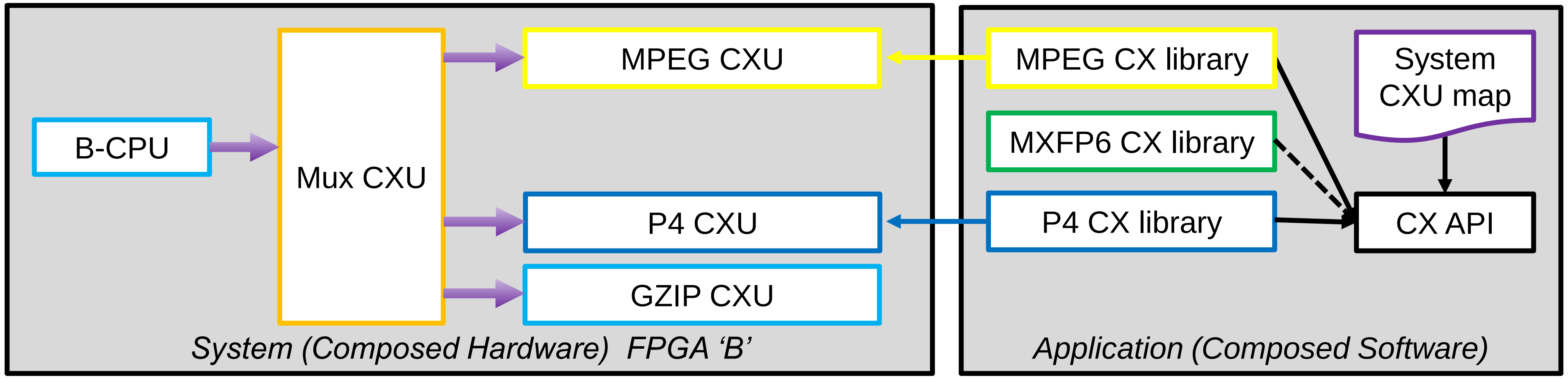
Technical backgrounder outline

- Composition examples
- Design tenets and key abstractions
- CX multiplexing
- CX access control
- CX state context management
- CX API
- CX ABI
- CXU logic interface

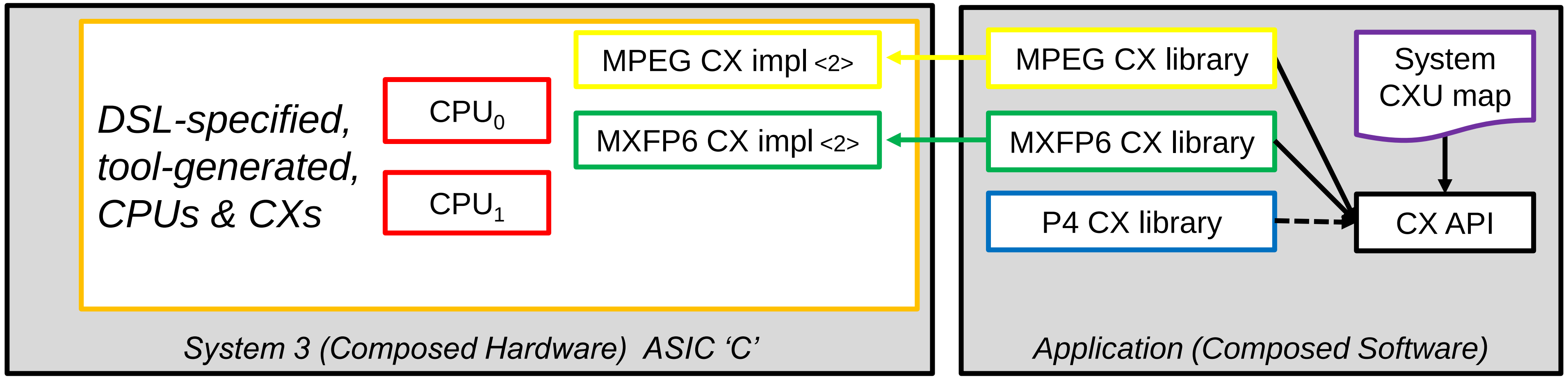
Composing composable custom extensions with CXUs



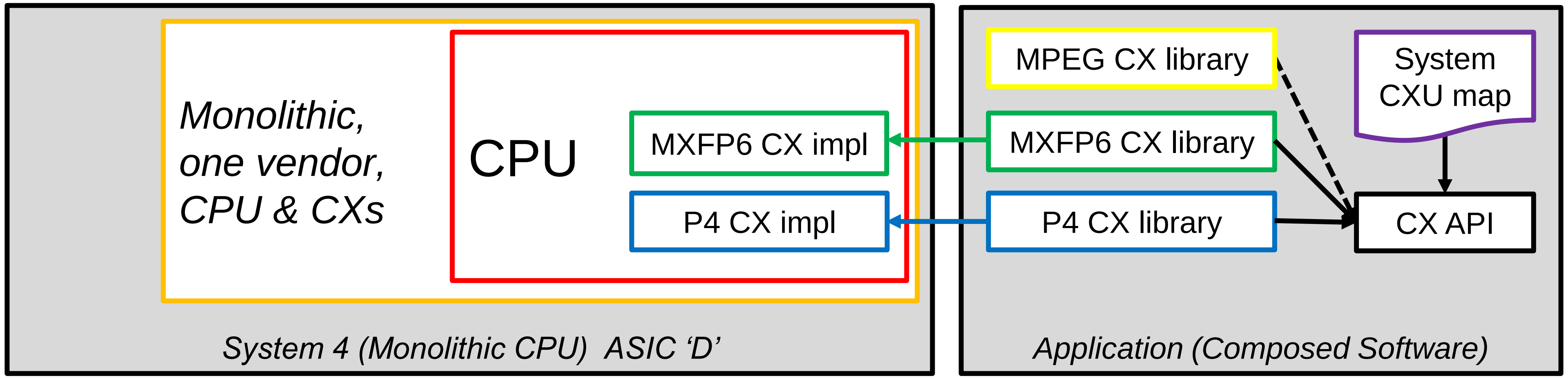
Composing composable custom extensions with CXUs (2)



Composing CXs with an ‘Extensions DSL’



Composing CXs within a monolithic CPU


Monolithic, one vendor, CPU & CXs
CPU
MXFP6 CX impl
P4 CX impl
System 4 (Monolithic CPU) ASIC 'D'
MPEG CX library
MXFP6 CX library
P4 CX library
System CXU map
CX API
Application (Composed Software)

Some problems to address

1. Custom opcode & CSR conflicts
2. Ensuring dependable composition
3. Per extension programming models
4. Per extension state models and OS support
5. Security concerns
6. Reimplementing the same custom extension HW in many CPU cores

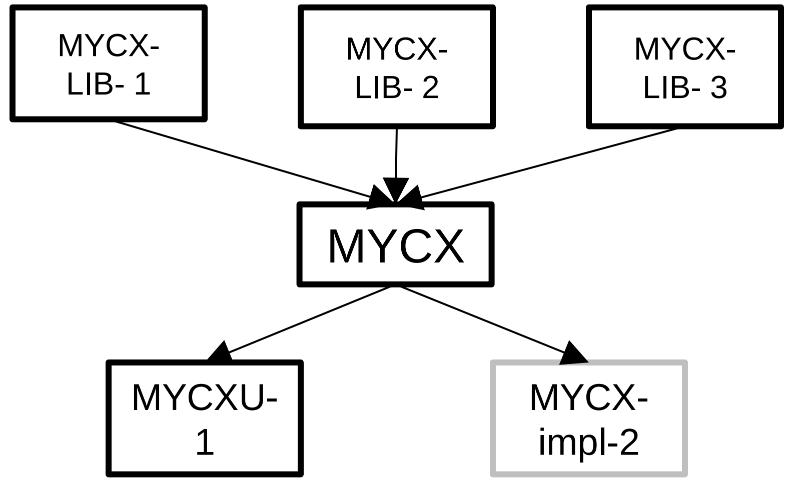
https://lists.riscv.org/g/tech-composable-custom-extensions/message/18

TG design tenets

- Composition
- Decentralized framework
- Diverse systems
- Stable software binaries
- Stable modular hardware
- Uniform software: naming, discovery, versioning, state, errors, …
- Simple, frugal, fast
- Secure
- Forward compatibility

Key terms / abstractions

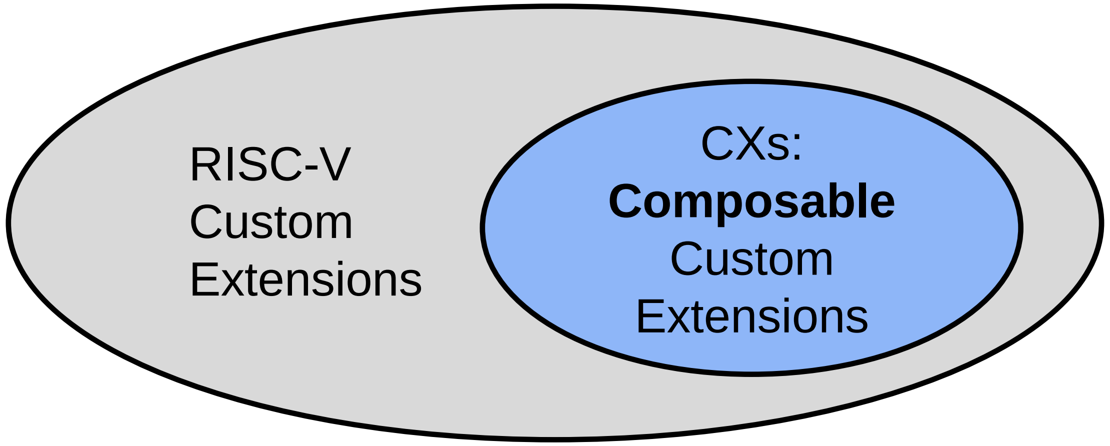
- Composable custom extension (CX) – an *abstract* ISA contract
- CX library – software that first *discovers* and *selects* a CX prior to *issuing* its custom instructions
- CX unit (CXU) – a *concrete,* reusable HW implementation of a CX



Composability criteria

- What custom extensions are deemed **composable**
- "What works separately, works together"
  - Composition invariance (state isolation), portability
  - Decide what hart state CX instructions may access

https://lists.riscv.org/g/tech-composable-custom-extensions/message/143



CX multiplexing *[unpriv ISA]*

For compatible & conflict-free reuse of the custom opcode & CSR space

- A new CSR indicating hart's current custom extension
- Multiplexing off: built-in custom instructions, unchanged
- CX library:
  - **Discover**: is my CX available? → *`selector`*
  - **Select**: **`cxselect`** *`prev,selector`* *TBD*
  - **Issue**: **`custom-*`** …
    `csrrw` …,*`cxcsr`*,…
  - **Errors**: *exception and/or a CX status CSR* *TBD*

Stateful composable extensions

- CX custom instructions may be stateful
  - May access new accumulators, registers, register files, …
- Conflict-free CX custom CSRs
- Uniform CX state context management
  - Mapping from harts to CX state contexts is *1:1*, or *m:n?* TBD
  - Potentially dozens of CX state contexts, per CX, per system
  - Uniform OS access control, virtualization, context switching of CX state

An exemplary CX programming model (sketch only)

- Once on library init

```
sel = cx_open(MYCX_ID, …);           // discover if MYCX is available
```

- Per library API call

```
if (sel < 0) return sw(…);           // pure software if MYCX absent
cx_select(sel);                      // cxselect sel  // select MYCX
for (…)
 c[i] = cx_reg(FN,a[i],b[i]);        // custom-0 rd,FN,rs1,rs2
cx_select(0);                        // cxselect zero  // select built-in custom ext.
```

CX access control *[priv ISA]*

- Enable more-priv software to grant/deny access to CXs by less-priv
  - While supporting fast unpriv CX multiplexing without a kernel detour
- Enable trap-emulation of absent CXs
- Perhaps controlled via CSR(s) or protected memory tables, TBD

Uniform CX state context management *[priv ISA]*

- Enable OS to virtualize CXs → to save/restore *any* CX state context
- Implemented by every composable custom extension instance
- New instructions and/or CSRs to uniformly manage the hart’s currently selected CX state context and save/restore its CX state context blob

CX API / CX programming model *[non-ISA]*

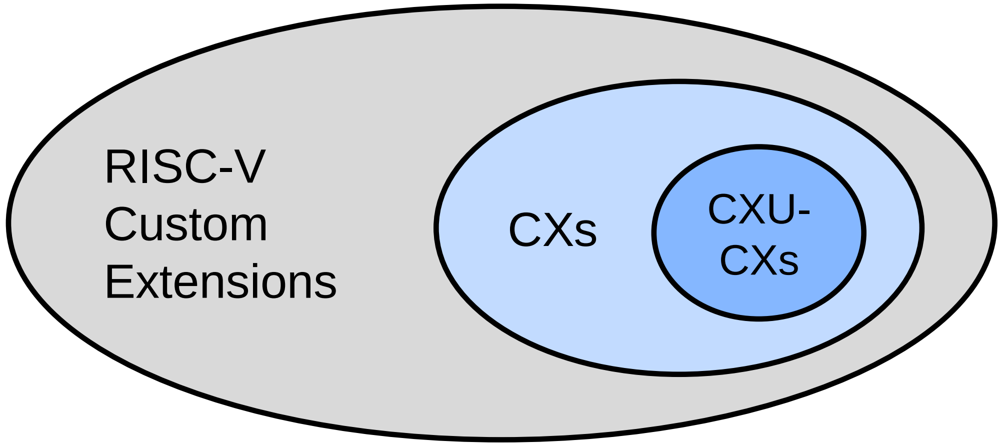
- Specify a uniform CX library software programming model: uniform CX naming, discovery, versioning, state model(s), error handling
- Support composition over independently authored and versioned CX software library binaries
    - Including applications with multiple libraries that use a given extension

CX ABI *[non-ISA]*

- Specify a calling convention for the hart’s CX selection mechanism for correct CX library composition
  - Support nested CX libraries
  - Support mixes of built-in ‘legacy’ custom extensions code alongside new CXs software

CXU Logic Interface (CXU-LI) *[non-ISA]*

- Specify an *optional* logic interface for reusable CX implementations
  - Modular, composable, decentralized, stable CPU and CXU RTL, for diverse CPU & CXU pipelines, simple, frugal, high performance
  - Optional, a peer of RoCC, CV-X-IF, CFU-LI, others
- Stable CPU RTL → fixed CX instruction schema *for CXUs?*



CX TG Status and Current Ongoing Work

Draft Charter

“The TG will define a framework of ISA and non-ISA specifications that together facilitate the decentralized, cooperative reuse of the custom instruction and custom CSR space, enabling practical reuse, within a system, of multiple, independently authored composable custom extensions (CXs), CX libraries, and CX unit (CXU) logic modules, while also remaining backwards compatible with legacy custom extensions.”

https://github.com/riscv-admin/composable-custom-extensions/blob/main/charter.adoc

Current TG work

- Plan milestone of specification development
    - Requirements → work breakdown (schedule) → Ratification Plan
- Some current requirements scoping discussions
    - Extension state: one or many per hart? Hart state or orthogonal to it?
    - Portable shared memory model for CXs' custom instructions
    - Opcode allocation: use all custom opcodes, or only a subset? Switch them collectively, or in some other way?
    - Support hotpluggable composable custom extensions?
    - Capabilities of the optional logic interface

https://lists.riscv.org/g/tech-composable-custom-extensions/message/251

CX TG Call to Action!

- Agility of custom extensions + composability of standard extensions
- For a marketplace of reusable extensions IP
- (And extend 32b opcode space)

- A sweet HW-SW codesign problem
  - ISA, hardware, software, verification, apps / market segments
- Decisions being made, work being planned – perfect time to engage
- Join us!

References and Resources

- CX TG list: https://lists.riscv.org/g/tech-composable-custom-extensions
- Home: https://github.com/riscv-admin/composable-custom-extensions/
- Charter: https://github.com/riscv-admin/composable-custom-extensions/blob/main/charter.adoc
- JIRA: https://lf-riscv.atlassian.net/browse/RVG-160

- *Basis spec: https://raw.githubusercontent.com/grayresearch/CX/main/spec/spec.pdf*
- *Basis spec rationale: https://www.youtube.com/watch?v=7daY_E2itpo*

Q & A