Composable Extensions (CX) Task Group Proposal

Draft 2024-02-15

Jan Gray, Guy Lemieux
Soft CPU SIG

Outline

- TG ask, value proposition, support, charter
- How it might work from software
- Overview of deliverables
- Q & A

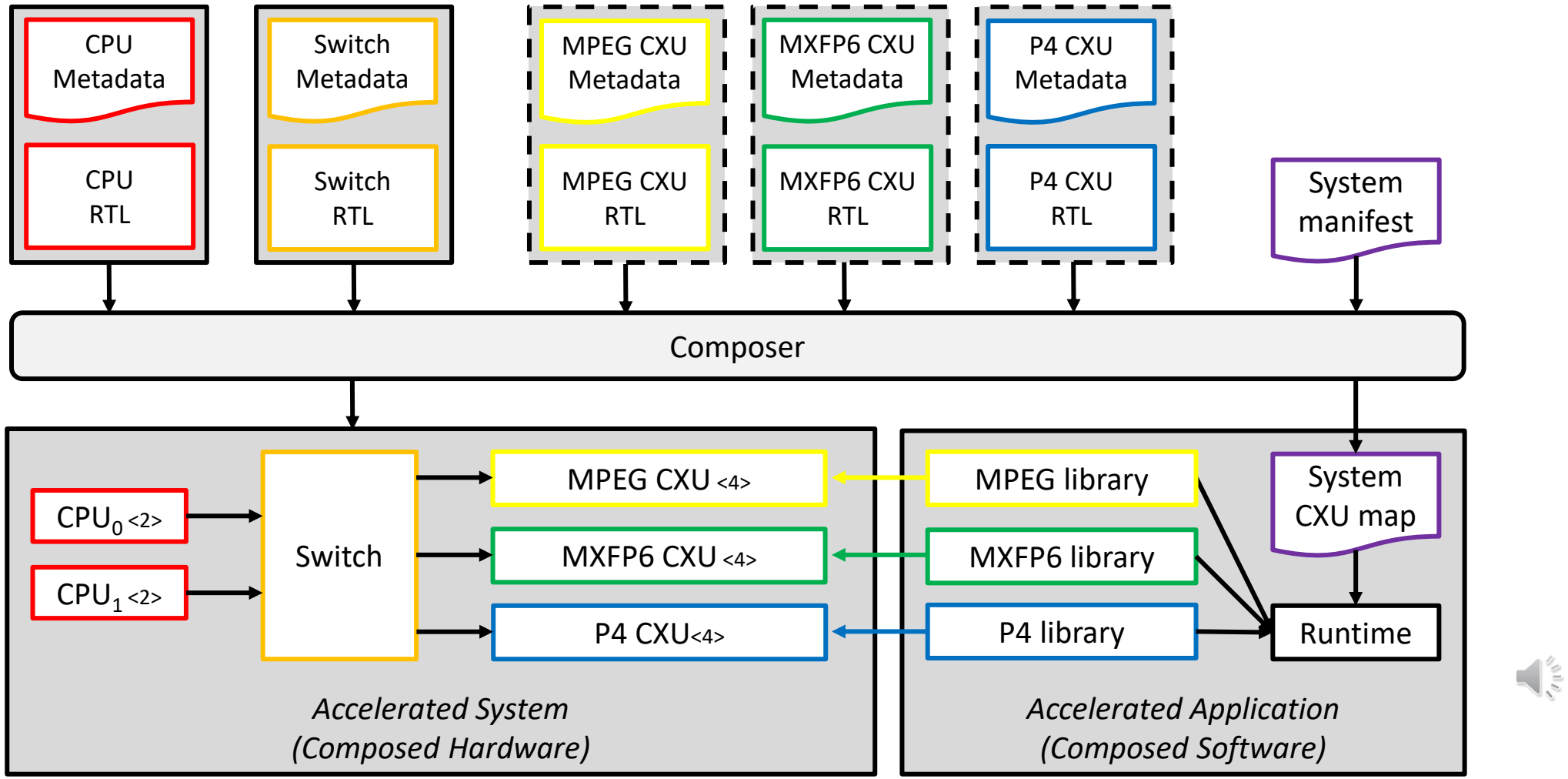
Composable Extensions TG Value Proposition

- We propose a new Task Group
- To specify ISA extensions + software and hardware interop interfaces
- To compose *my composable custom extensions* with *yours*, without prior coordination and without changing our libraries or OS

- Enables **robust reuse** of anyone's extensions and libraries
- Provides **uniform** software access to such extensions
- For a **marketplace** of extensions & libraries (especially on FPGAs)

↑ uniformity ↑ reuse ↑ agility ↓ conflicts ↓ silos ↓ fragmentation

Composing Custom Extensions



Composable Extensions TG Charter

https://github.com/riscv-admin/sig-soft-cpu/blob/main/TG/CX/CHARTER.md

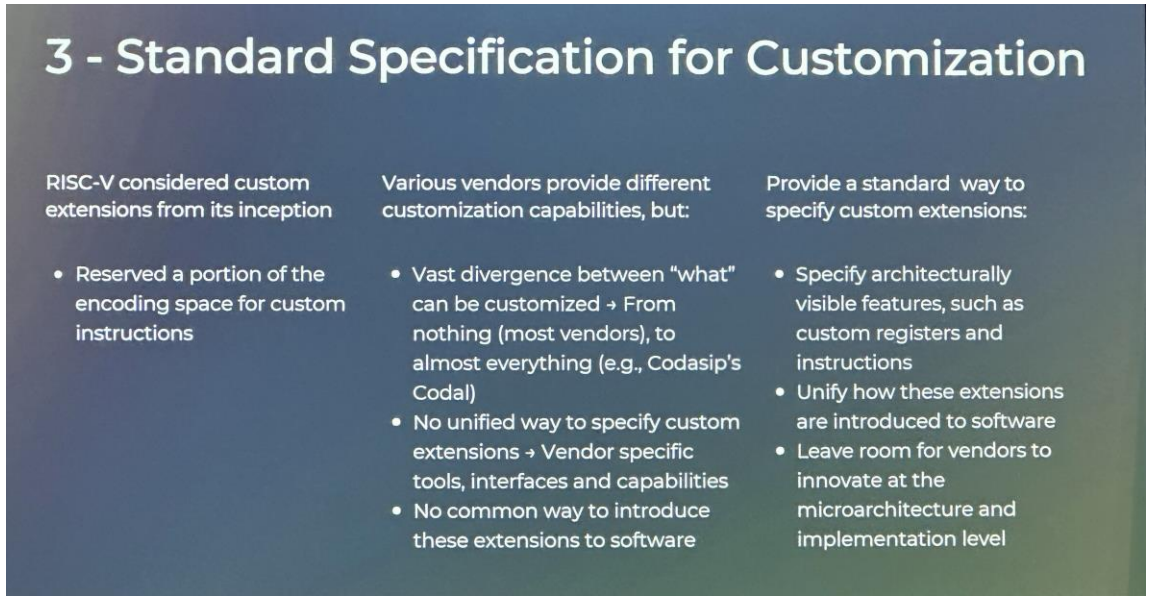
Executive Summary

This Charter governs the TG for a Composable Extensions (CX) framework enabling **robust composition** of multiple independently authored composable custom extensions, alongside legacy custom extensions, assembled conflict-free in one RISC-V system.

By multiplexing the custom opcode space, and adopting common software and hardware interop interfaces, CX enables **uniform** extension naming, discovery, and versioning, error handling, state context management, extension hardware reuse, and stable software binaries for each target system -- all without a central management authority.

Support for the Task Group

Meta (RISC-V Summit NA 2023)



CX TG supporters/participants

- AMD (Xilinx)
- Lattice
- Bluespec
- Antmicro
- Alibaba
- Individual members
- *More to come*

Composable Extension Multiplexing

For conflict-free reuse of the custom opcode space

- Multiplexing off: legacy custom instructions, unchanged
- Extension library:
  - **Discover**: is named extension available here? → CX selector *index*
  - **Select**: `csrrw prev-index,cx_index,index`
  - **Issue**: `custom-*` instructions
- CPU:
  - **Decode**: `custom-*` instruction → (function-ID, operands)
  - **Issue**: send request (state-ID, function-ID, [insn,] operands) to *selected* extension
  - **Retire**: receive response, update dest-register and `cx_status` CSR

Stateful Composable Extensions

- CX custom instructions may be stateful
  - May access accumulators, registers, register files, …
  - Some CX state accessible as *CX scoped CSRs (CX-CSRs)*
- Uniform CX state context management
  - Supports 100s of CX state contexts, per CX, per system
  - Any mapping from harts to CX state contexts, 1:1, 1:*n*, *n*:1
  - The same OS code can save/restore any state context of any CX

Example CX Programming Model, in C++

```
struct MyEncoderCX : CX {
    static cx_guid_t guid;      // unique canonical ID of MyEncoderCX contract
    // CX for stateless sum of absolute differences, 8x8b SIMD inputs, 64b output
    static uint64_t sad_cx(uint64_t a, uint64_t b) { return cxfn(1, a, b); }
    static uint64_t sad_sw(uint64_t a, uint64_t b); // software version
    ...
};

// CX library example: accelerated sum of absolute differences of two vectors
static uint64_t MyEncoderCX::sad_vecs(uint64_t as[], uint64_t bs[], size_t n) {
    CX::Scope encoder(guid);    // discover & select CX if present; writes cx_index
    uint64_t total = 0;
    for (size_t i = 0; i < n; ++i) {
        // use CX instruction if present, software otherwise;
        // here 'encoder' test is loop invariant, can be hoisted
        total += encoder ? sad_cx(as[i], bs[i]) : sad_sw(as[i], bs[i]);
    }
    return total;
    // encoder.~Scope() automaticaly restores caller's CX selection; writes cx_index
}
```

Design Tenets

- **Routine robust reuse of composable extensions & libraries**
- No prior coordination or central authority
- Stable software binaries
- Stable modular hardware
- Uniform SW: naming, discovery, versioning, state, errors, …
- Simple, frugal, fast
- Secure
- Interop across the decades

CX TG Extensions and Interface Spec Deliverables

- New ISA extension(s): **new custom CSRs & multiplexed custom instructions** only

	Unpriv and Priv ISA
CX Multiplexing	`mcx_selector cx_status` **(basic muxing) +** `mcx_table cx_index` **(optional privileged access control)**
CX State Management	***CX scoped* custom CSRs +** **+ 3 CX CSRs for OS to read/write any CX state context**

- New interfaces
  - Software: CX-API: extension services: uniform discovery, versioning, state mgmt
  - Software: CX-ABI: software rules for CX-Mux CSRs
  - Hardware: CXU-LI (logic interface): auto-composition of CPUs + CX Units

Composable Extensions Collateral

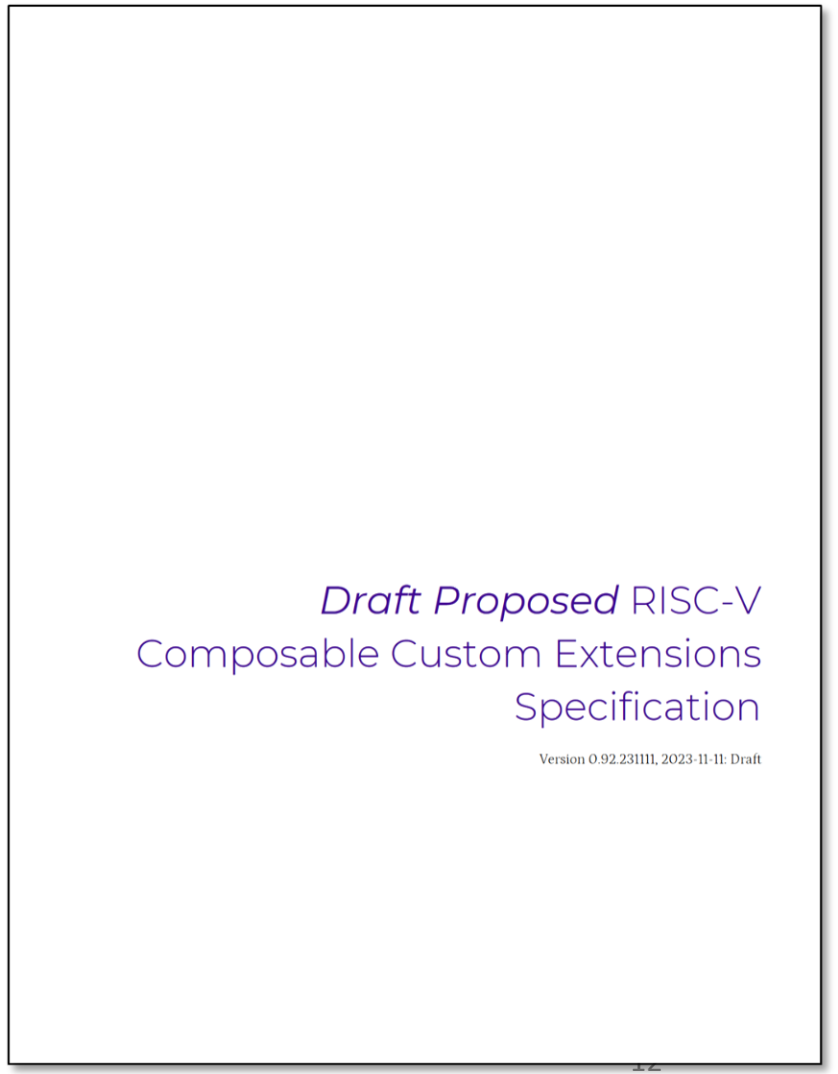
TG Charter:
https://github.com/riscv-admin/sig-soft-cpu/blob/main/TG/CX/CHARTER.md

Draft Spec:
https://raw.githubusercontent.com/grayresearch/CX/main/spec/spec.pdf

Design and Rationale Talk:
https://www.youtube.com/watch?v=7daY_E2itpo

Talk slides:
https://raw.githubusercontent.com/grayresearch/CFU/main/collateral/design-rationale-CX-CXU-spec.pdf

Repo: https://github.com/grayresearch/CX



Task Group Personnel

- Past contributors
  - Tim Ansell, Tim Callahan, Jan Gray, Karol Gugala, Olof Kingdren, Maciej Kurc, Guy Lemieux, Charles Papon, Tim Vogt
    (Antmicro, Lattice, Google, Gray Research, QAMCOM, SpinalHDL/VexRiscv, UBC)
- TG interest
  - AMD, Lattice, Bluespec, AntMicro, Alibaba, individuals
- TG Chair/Vice-Chair Interest
  - Jan Gray, Guy Lemieux, TBD / open invite

Conclusion / Q & A