Composable Custom Extensions and Custom Function Units for RISC-V

Jan Gray (Gray Research) , Tim Vogt (Lattice Semiconductor), Tim Callahan (Google), Charles Papon (SpinalHDL),
Guy Lemieux (University of British Columbia), Maciej Kurc (Antmicro), Karol Gugala (Antmicro) ***draft #3, 2022-04-27***

RISC-V custom extensions' interop problem

- Standard extensions layer and compose. But take years to ratify
- Custom extensions allow rapid in-house accelerator & library solutions
  - Solutions may not work *together* – conflicting encodings, different means of discovery, computation, state, error handling, versioning
  - Silos caused by poor interop limits reuse, fragments the ecosystem

Let us build a mix-and-match custom extensions future

- *Agility* of custom extensions with *composability* of standard extensions
- Proposed HW-SW and HW-HW interop interfaces enable reusable accelerators that *just work* together – a *marketplace* of accelerators

Key ideas

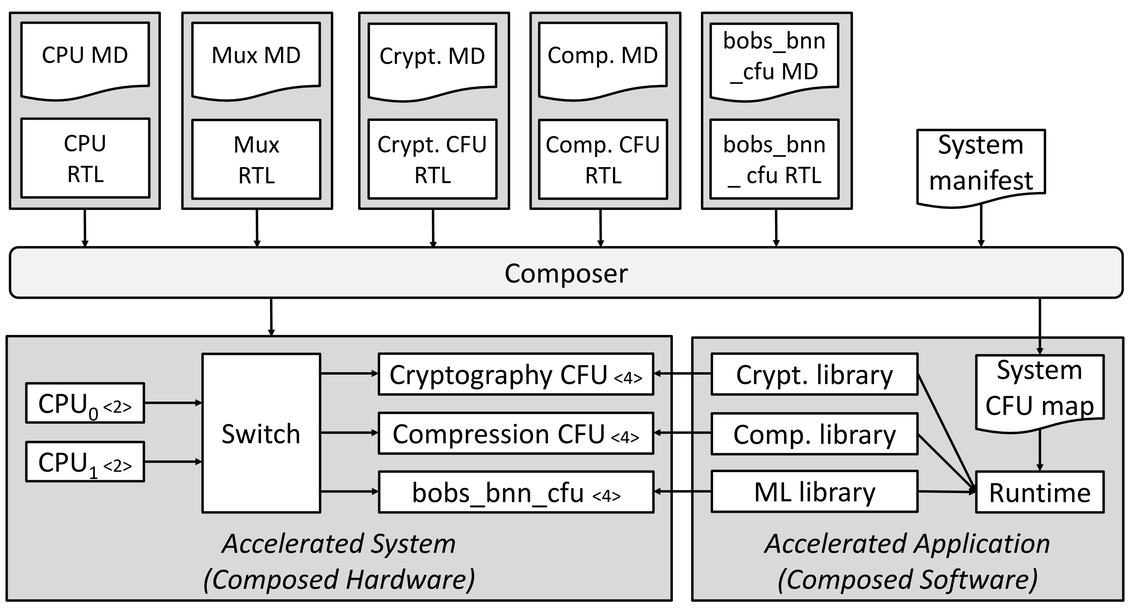
- **Custom interface (CI):** abstracts a *composable* custom extension
- **Custom function unit (CFU)**: core that implements a custom interface
- **Accelerated library***:* issues custom instructions to a custom interface

New interop interfaces

- **HW-SW: interface multiplexing**: libs select hart's *current* CI & CI-state
  → each custom interface enjoys full custom instruction encoding space
- **HW-HW: CFU Logic Interface (CFU-LI**): CFU signaling standard
  → automatic composition of CPU+CFU complexes

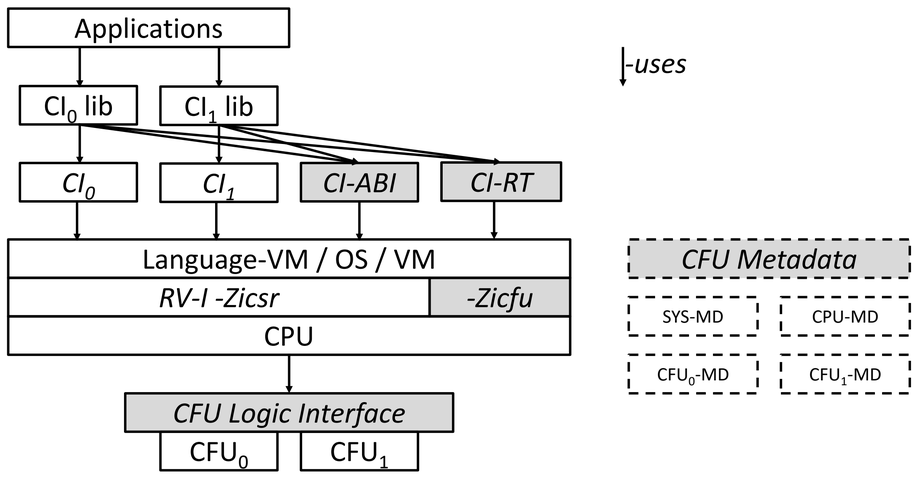
Example

Composable CPU and CFU Packages



HW-SW stack changes

- Custom Interface Runtime: accelerated library services (discovery …)
- "-Zicfu": interface multiplexing CSRs: **`mcfu_select`**, **`cfu_status`**
- CFU Logic Interface & metadata: automatic CPUs+CFUs composition



Some composition challenges we address

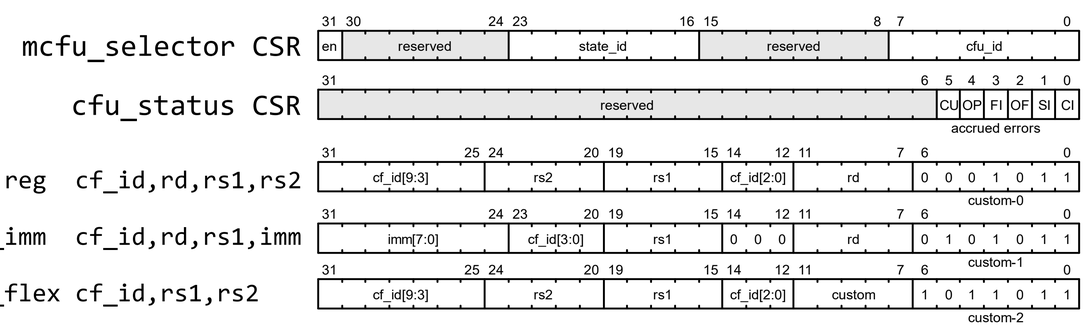
- Namespace / ID management with no central authority
- Custom instruction encoding space management
- Correct composition of stateful custom extensions
- Extension-agnostic context save/restore
- Discovery of custom extensions
- Uniform error handling
- Versioning of custom extensions over time
- Privileged systems: access control to extensions and state
- *Please see spec for all the details*

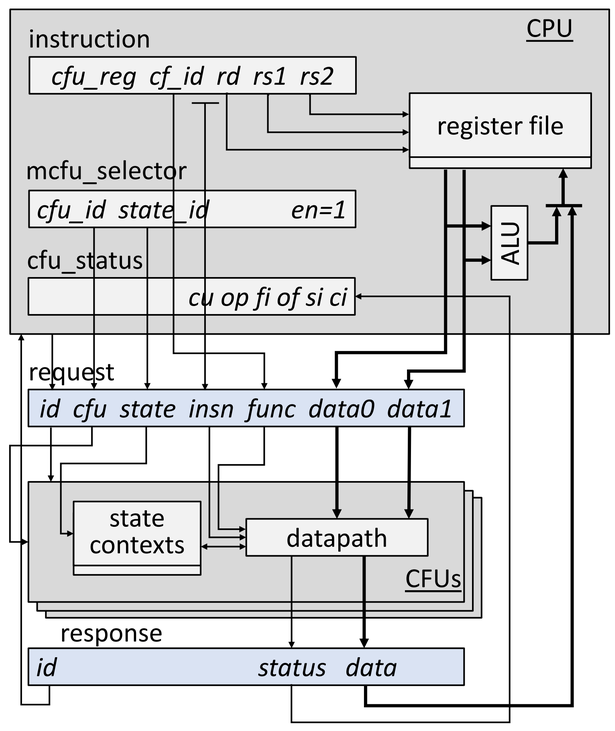
Correct composition via isolation

- Behavior of an extension **must not** change when composed with others
- V1: custom instructions only access register file & selected state context
  - Each interface/CFU may have 0, 1, *#harts*, or *n* isolated state contexts

HW-SW interface: custom interface multiplexing

- Inexhaustible, collision-free instruction encodings
- **`mcfu_selector`** CSR selects hart's *current* CFU and its state context
- **`custom-0/-1/-2`** functions routed to the selected CFU
- CFU performs function, may update its state context
- CFU response updates destination register and **`cfu_status`** CSR





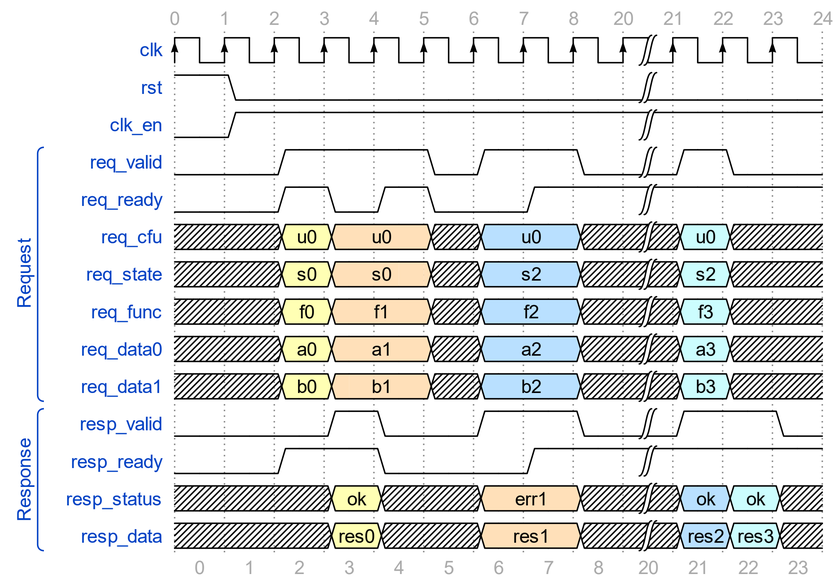
Example accelerated library programming model

- Try to select a custom interface, issue custom instructions if CFU present

```
CI bitmanip(CI_ID_IBitmanip);   // csrrw mcfu_selector …
if (bitmanip)                   // accelerator present:
  count = cf(pcnt_cf, data, 0); //   cfu_reg pcnt_cf,rd,data,x0
else                            // accelerator not present:
  count = popcount(data);       //   software version
```

HW-HW interface: CFU Logic Interface (CFU-LI)

- Flexible feature levels: combinational, fixed, variable latency, reordering
- Prebuilt switches & adapters for glueless composition
- Example: CFU-L2 variable latency signaling:



Status, more information, help us shape these interfaces

- *Draft Proposed RISC-V Composable Custom Extensions Specification,* https://github.com/grayresearch/CFU
- Status: refining spec, building consensus, writing RTL to demo mix-and-match composition across various CPUs and multiple CFUs
- Discussions and meetings: see RISC-V [sig-soft-cpu] list

Let us add just enough architecture
that our custom extensions just work together