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 #1, 2022-04-25***

RISC-V extensions allow diverse uses. But fragmentation!

- Standard extensions layer and compose, by design
  - But consume finite encoding space, take years to ratify
- Custom extensions enable rapid in-house accelerator+library solutions
  - FPGA RISC-V SoCs: *insight* → *accelerated SoC* in an hour
  - But custom extensions may not work *together* in a system, due to conflicting encodings, CPU pipelines, logic interfaces, models of discovery, computation, state, error handling, versioning, tooling
  - Siloed solutions hurt HW & SW reuse and fragment the ecosystem

Towards a mix-and-match custom extensions ecosystem

- *Agility* of custom extensions with *composability* of standard extensions
- Common HW-SW and HW-HW interop interfaces can enable a new ecosystem of reusable accelerators & libraries that *just work* together

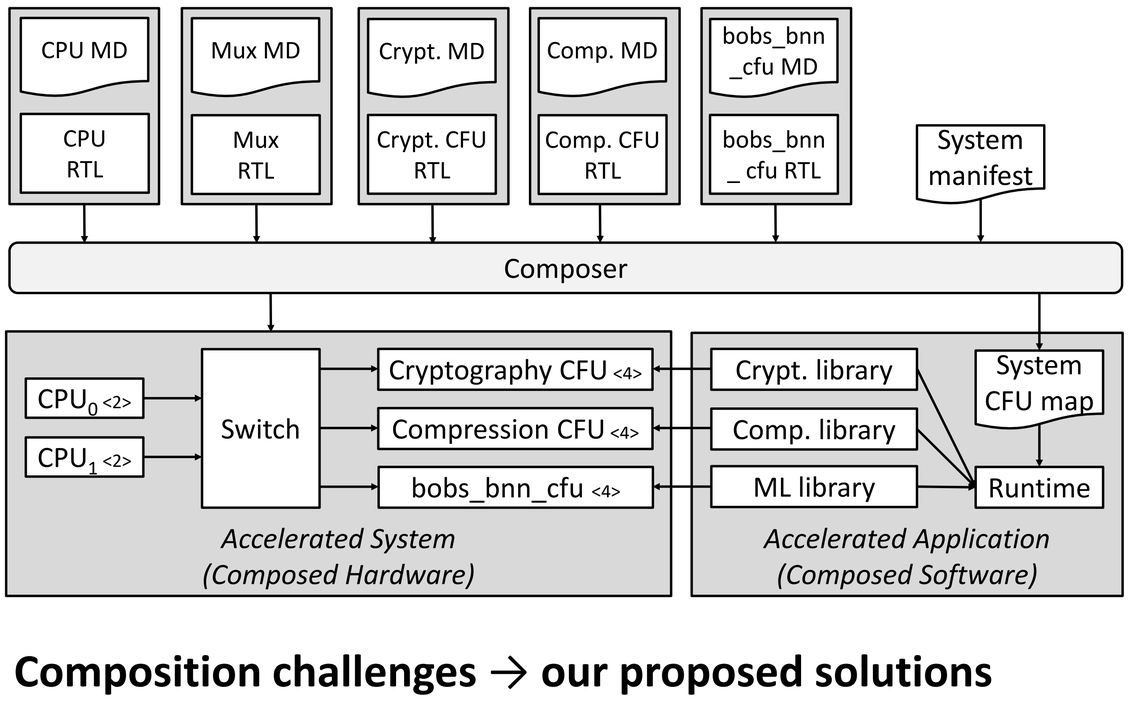
New concepts

- *Custom interface:* a composable custom extension: a named, immutable set of composition-safe *custom function instructions*
- *Custom function unit (CFU)*: a core that implements a custom interface
- *Accelerated library:* issues custom instructions to a custom interface
- Different CFUs may implement an interface; different libs may use one
- *Anybody* can create a custom interface, CFU, or library; a CPU that composes CFUs; tools for same; and which works well with all others.

Example

- Alice's MPSoC system uses a custom cryptography library and CFU
- Alice adds a compression CFU package, accelerates her compression lib
- Bob adds his new binary neural net custom interface, ML lib, and CFU
- Using Runtime, each lib *selects* its interface, then issues its instructions

Composable CPU and CFU Packages

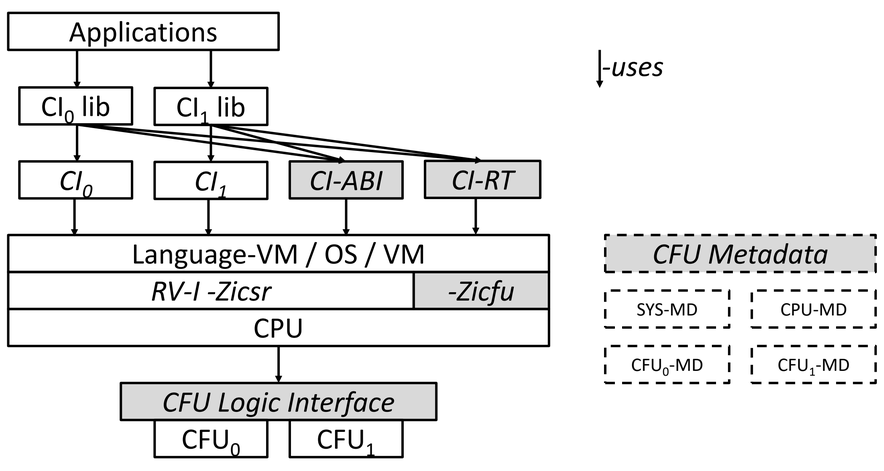


Composition challenges → our proposed solutions

- Naming? Anyone can mint a new 128b GUID custom interface ID
- Opcode space? Inexhaustible with custom interface multiplexing
- Correct composition? Strict isolation of interfaces' states
- Error handling? **`mcfu_status`** CSR accumulates CFU error responses
- Discovery? Library asks Runtime, Runtime searches System CFU Map
- Context mgmt? IStateContext's standard functions save&load contexts
- Versioning? Immutable interfaces + interface discovery + multiplexing
- Security? Optional privileged access control of CFUs & state contexts
- See spec for much more on these matters

HW-SW stack with proposed new interop interfaces

- Custom Interface ABI, Custom Interface Runtime
- "-Zicfu": CSRs: **`mcfu_select`**, **`cfu_status`**; instructions: **`custom-0/1/2`**
- CFU Logic Interface for easy, glueless CPUs + CFUs composition

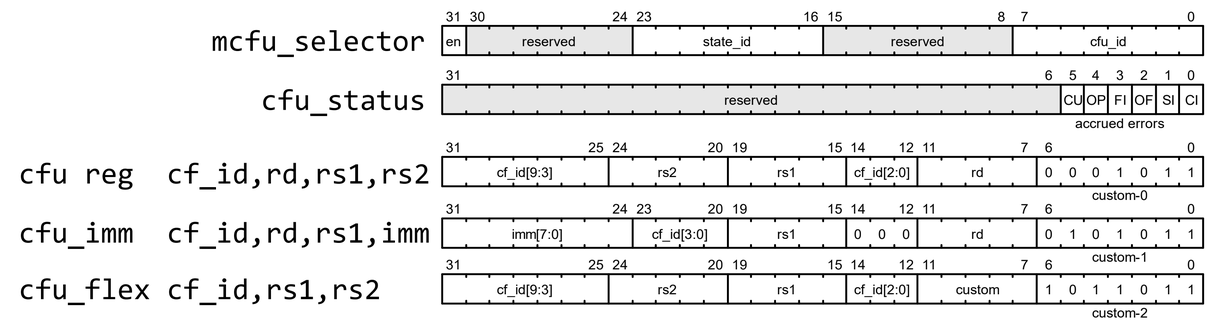


Reliable composition via strict isolation of interface state

- Behavior of interfaces/CFUs **must not** change if composed with others
- Therefore, each *custom function instruction* only accesses integer registers and an isolated state context of the selected CFU
- A CFU may be stateless, have one state context per hart, or otherwise
- Presently no access to shared resources (memory, CSRs, PC, exceptions)

Custom interface multiplexing

- Inexhaustible, collision-free opcode space / no central opcode authority
- **`mcfu_selector`** CSR selects hart's *current* CFU and state context
- **`custom-0/-1/-2`** instructions send a CFU request to hart's *current* CFU
- CFU executes request, may read & write a private state context
- CFU response updates destination register and **`cfu_status`** CSR



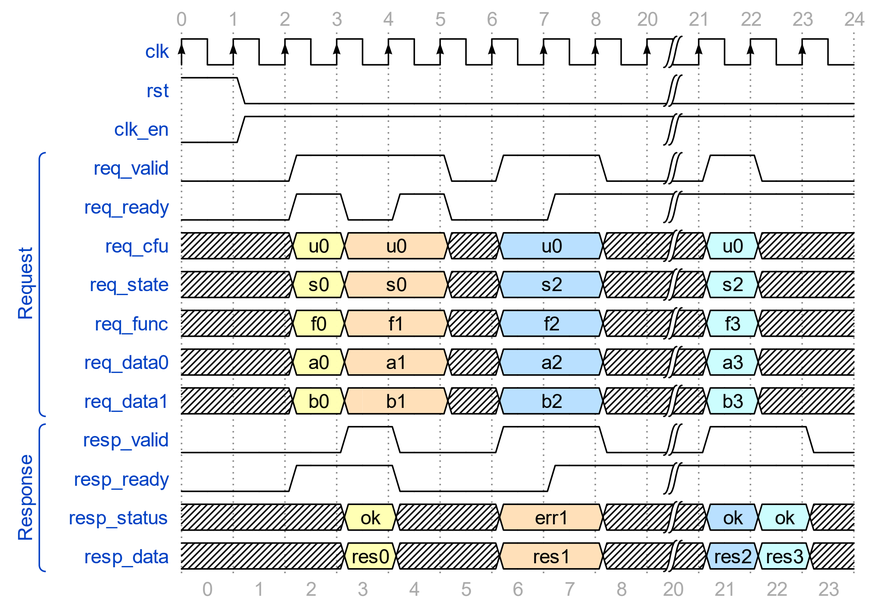
Possible accelerated library programming model

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

```
CI ci(CI_ID_IBitmanip); // Runtime: … csrrw mcfu_selector,… …
if (ci)
  count = cf(pcnt_cf, data, 0); // cfu_reg pcnt_cf,rd,data,x0
else
  count = popcount(data); // pure software, unaccelerated
```

CFU Logic Interface (CFU-LI)

- Standard, interoperable CPUs-to-CFUs signaling and metadata
- Flexible feature levels: combinational, fixed, variable latency, reordering
- Standard switches and adapters enable glueless automatic composition



Status, more information, help shape these interfaces

- *Draft Proposed RISC-V Composable Custom Extensions Specification,* https://github.com/grayresearch/CFU
- Status: reviewing draft spec, writing RTL to demonstrate plug-and-play composition across different CPUs and multiple CFUs
- Future discussions and meetings: please see RISC-V [sig-soft-cpu] list
- See also: CFU Playground