Extending Cores

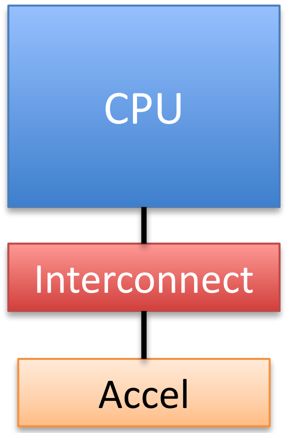
- Directly integrate into core
- Loosely-coupled accelerators
- Tightly-coupled accelerators
- Instruction-Set Extension Interfaces

Extending Processor Cores

- Pros
  - Potentially low complexity
  - Easier to achieve performance gains
- Cons
  - Very complex beyond arithmetic
  - Not transferrable between cores

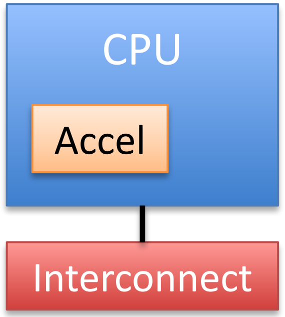
Loosely-coupled accelerators

- Pros
  - Easy to re-use, cost and time saving
  - Well-defined interface
- Cons
  - Performance impact for small data items
  - Accessed only as device, complex

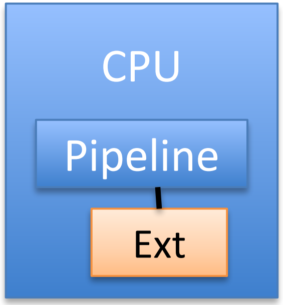


Tightly-coupled accelerators

- Pros
  - Better performance for small items
  - Easier access via CSRs
- Cons
  - May need changes to core data path
  - Re-use limited, integration effort



Instruction Set Extension Interfaces



- Pros
  - Better performance, low latency
  - Portable to other supported cores
- Cons
  - Can be complex to integrate for complex cores
  - Limited to certain problem subset

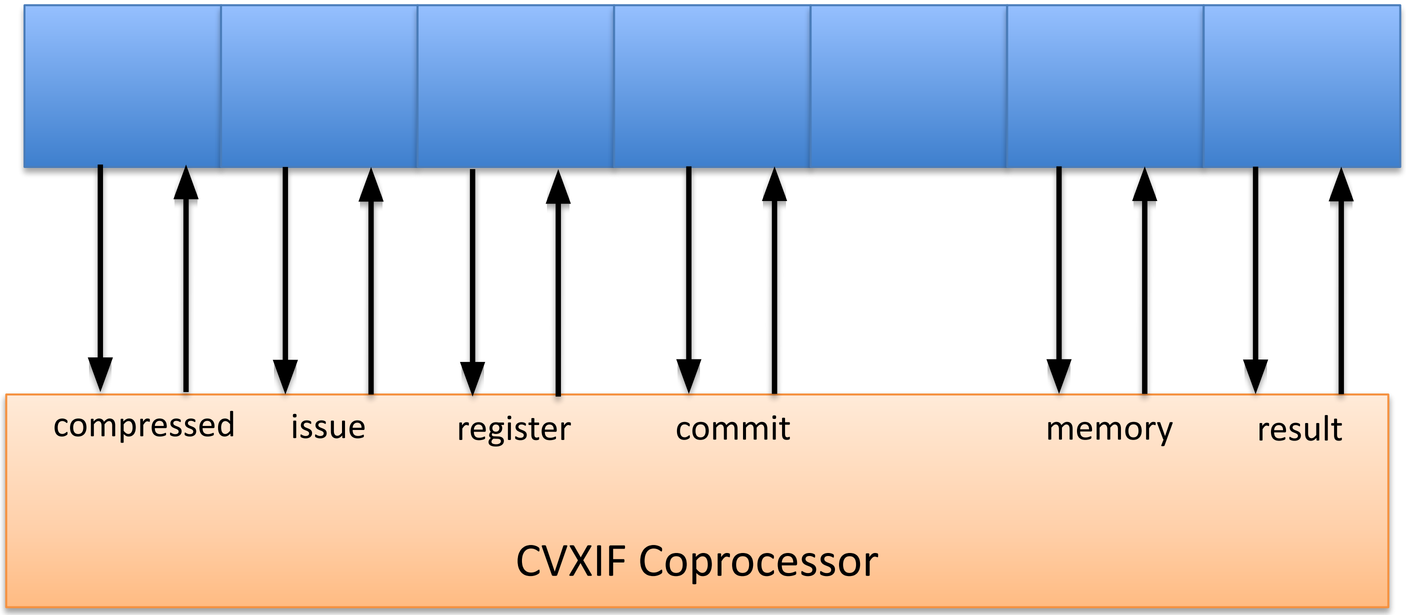
CORE-V Extension Interface

- Invented at Pulp team (FPU, then ARA)
- Now maintained by OpenHW Group
- Useful for standard and custom extensions
- Cores forward all unknown instructions
- Defines: Interfaces and handshake rules

CVXIF - Interfaces

- **compressed**: Decode compressed instruction
- **issue**: Start execution of instruction
- **register**: Operands ready
- **commit**: Speculation support
- **result**: End execution
- **memory**: Memory accesses

CVXIF – Integration



CVXIF – Common signals

- Multi-hart support: `hart_id`
- Track instructions: `id`
  - Handle multiple in parallel
  - Multi-coprocessor support
- `valid` and `ready` handshaking

CVXIF – compressed

- Core: valid and instr (16-bit instrustion)
- Coproc:
  - ready: done with decoding
  - accept: matches instruction
  - instr: decoded 32-bit instruction

CVXIF – issue

- Core: `valid` and `input` (32-bit instrustion)
- Coproc:
  - `accept`: This is a valid instruction -> start
  - `writeback`: Will write rd
  - `register_read [1:0]`: Will read `rs`
- Uses standard coding for `rs` and `rd`

CVXIF – register

- Core:
  - valid
  - rs [XLEN:0][1:0] register values
  - rs_valid [1:0] operand ready
- Coproc:
  - ready

CVXIF - commit

- Core:
  - `valid`: Commit info available
  - `commit_kill`: Speculation info

Once valid, kill signals if this was aborted

CVXIF – writeback

- Core: `ready`
- Coproc:
  - `data`: Data to be written

Practical Tutorial

- Create CVXIF coprocessor and test it
- Idea: Complex instruction support
  - Addition
  - Conjugate
- CVXIF development and cocotb tests