CORES TG – *Relocating PULP encoding and changing HWLoop ISA*

Davide Schiavone

John Martin

OPENHW GROUP™
PROVEN PROCESSOR IP

Overview

PULP instructions can be split into the following groups:

- Hardware loop
- Multiply accumulate
- Post-increment and register-indexed load/store
- Direct branches
- General ALU operations
- Bit manipulation
- SIMD

Bit manipulation and SIMD will not be included in the toolchain

Will still exist in the hardware and still need a new encoding

Overview

Current encoding has instructions in the following blocks:

- custom-0
- custom-1
- custom-2
- custom-3
- load
- store
- op
- branch
- reserved-a

To be RISC-V compliant the instructions in the red blocks must be moved into the custom encoding space

New Encoding Scheme

custom-0 and custom-1 includes instructions from:

- Post-increment and register-indexed load/store
- Direct branches

custom-1 includes two sub-encodings called “Plane A” and “Plane B”

custom-2 includes some instructions from:

- Bit manipulation
- General ALU operations
- Multiply accumulate

custom-3 includes all the SIMD instructions

New Encoding Scheme

Plane A includes:

- Post-increment and register-indexed load/store
- Bit manipulation
- General ALU operations
- Multiply accumulate

Plane B includes all the hardware loop instructions

Notes

New encoding has not been implemented in RTL yet

- It may make sense to re-arrange a few encodings to optimize the HW
- Procedure for this?

Toolchain work is already proceeding with proposed encodings

HWLoop ISA Changes

- CORE TG proposes to change the specs to address problems summarized here:
    - *https://github.com/openhwgroup/cv32e40p/issues/584*

HWLoop ISA Changes (1)

- CORE TG proposes:
  - **Add Illegal conditions for instructions within a HWLoop**
    - https://github.com/openhwgroup/core-v-docs/issues/265
    - HWLoop cannot contain RVC but also (new in doc):
      - branches/jumps, fence*, xret instructions, ecall, wfi (otherwise illegal)
    - RTL already compliant with this
      - except for to check the minimum distance between two nested loops, for min instructions size, and misaligned check for start and end.
    - ***If SW team accepts this, we will change doc and RTL***

HWLoop ISA Changes (2)

- CORE TG proposes:

  - **Redundancy and not equivalence of HWLoop CSRs**
    - https://github.com/openhwgroup/cv32e40p/issues/189
    - Currently one can write to CSRs either with ***start/end/count*** instructions or with CSRW operations. They both write in the CSRs, but they are not equivalent from an RTL point of view (risky). In addition, they are almost redundant.
    - ***start/end*** operations are PC relative defined. CSRW are not.
    - ***start/end*** operations with register source are missing. We propose to add them to the ISA to be used also in context switch and ISRs.

    - ***If SW team accepts this, we will change doc and RTL***

HWLoop ISA Changes (3)

- CORE TG proposes:
  - **Extend maximum PC-relative loop size**
    - https://github.com/openhwgroup/cv32e40p/issues/583
    - Currently the HWLoop uses the uimmL[11:0] field (bit 31:20) to calculate offset from PC in ***setup***, ***start*** and ***end*** instructions as:
      - Target = PC + uimmL[11:0] << 1
    - This is because instructions are half-word aligned (so LSB always 0, <<1)
      As the HWLoop is never misaligned, nor contains RVC instructions, we propose to define the target as:
      - Target = PC + uimmL[11:0] **<< *2***
    - This allows for bigger loops, and for not wasting 1bit of encoding which defore must be 0 (*thus a new condition of illegal to add*)
  - ***If SW team accepts this, we will change doc and RTL***

HWLoop ISA Changes (4)

- SW TG proposes:
  - **Change definition of HWLoop end-target**
    - No open issue yet
    - Currently the HWLoop end target is indeed the final instruction
      - Two challenges:
      - 1. To compilers, we have to introduce unnecessary "control flow" into the instruction stream.
      - 2. To end users it introduces the possibility of either misreading how long the loop is, or mis-labelling the loop endpoint
    - So RTL jumps if PC+4 == END → now if PC+4 == END-4 → PC+8==END
    - ***If SW and CORE team accepts this, we will change doc and RTL***